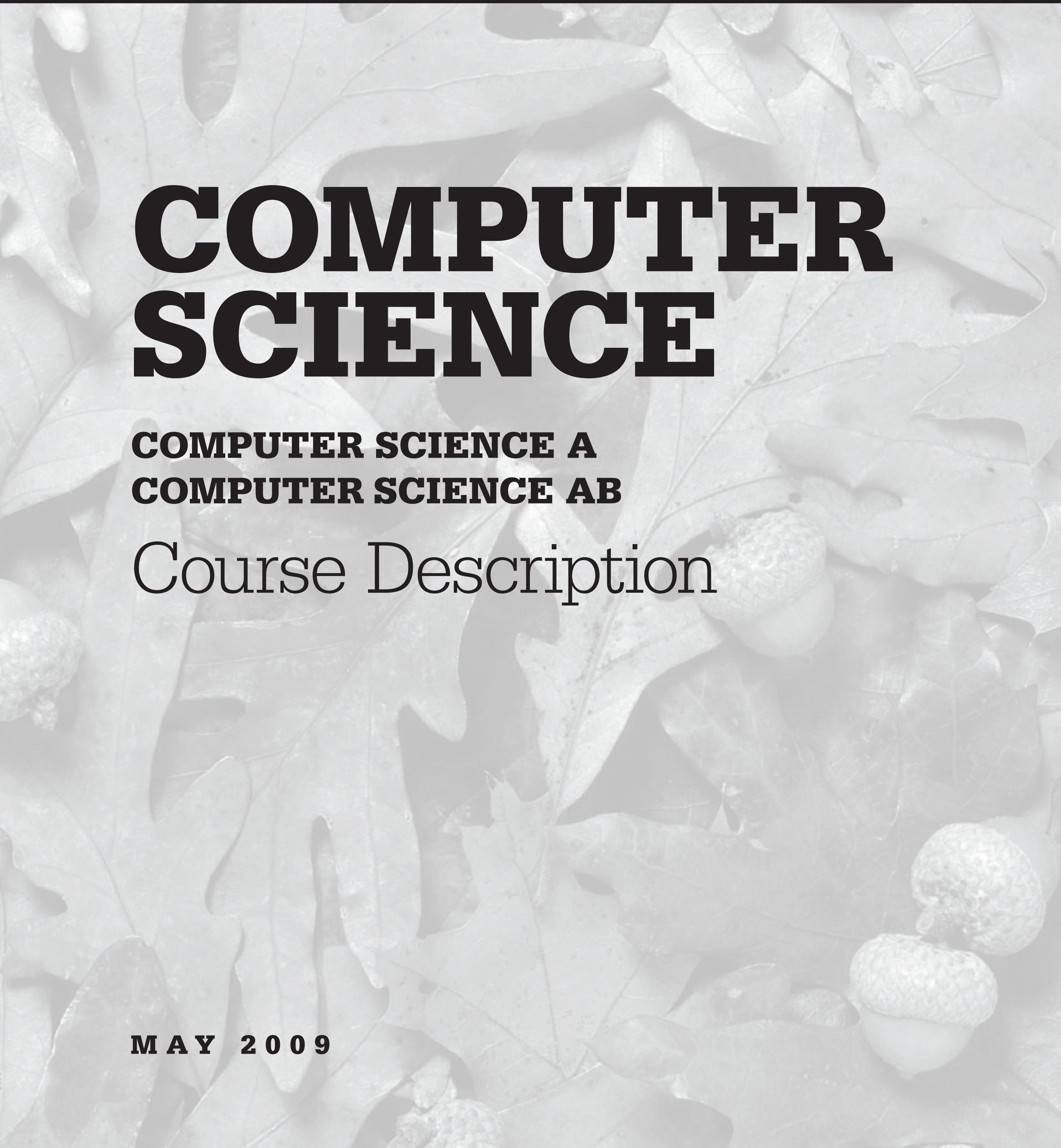
CollegeBoard AP



COMPUTER SCIENCE

COMPUTER SCIENCE A
COMPUTER SCIENCE AB

Course Description

MAY 2009

The College Board: Connecting Students to College Success

The College Board is a not-for-profit membership association whose mission is to connect students to college success and opportunity. Founded in 1900, the association is composed of more than 5,400 schools, colleges, universities, and other educational organizations. Each year, the College Board serves seven million students and their parents, 23,000 high schools, and 3,500 colleges through major programs and services in college admissions, guidance, assessment, financial aid, enrollment, and teaching and learning. Among its best-known programs are the SAT®, the PSAT/NMSQT®, and the Advanced Placement Program® (AP®). The College Board is committed to the principles of excellence and equity, and that commitment is embodied in all of its programs, services, activities, and concerns.

For further information visit www.collegeboard.com.

The College Board and the Advanced Placement Program encourage teachers, AP Coordinators, and school administrators to make equitable access a guiding principle for their AP programs. The College Board is committed to the principle that all students deserve an opportunity to participate in rigorous and academically challenging courses and programs. All students who are willing to accept the challenge of a rigorous academic curriculum should be considered for admission to AP courses. The Board encourages the elimination of barriers that restrict access to AP courses for students from ethnic, racial, and socioeconomic groups that have been traditionally underrepresented in the AP Program. Schools should make every effort to ensure that their AP classes reflect the diversity of their student population.

Dear Colleague:

We know that AP® is a unique collaboration among motivated students, dedicated teachers, and committed high schools, colleges, and universities. Without your contributions, the rigorous instruction that takes place in classrooms around the world would not be possible.

In 2007, approximately 1.4 million students took more than 2.5 million AP Exams. Guiding these students were talented, hardworking teachers, who are the heart and soul of the AP Program. The College Board is grateful for the dedication of AP teachers and the administrators who support them.

One example of the collaboration that makes AP possible is the AP Course Audit, the process through which college faculty review AP teachers' syllabi to ensure that both teachers and administrators are aware of the expectations colleges and universities have for AP courses. This yearlong intensive assessment involved the review and analysis of more than 134,000 syllabi to determine which courses fulfill or exceed standards for college-level curricula. In total, 14,383 secondary schools worldwide succeeded in developing one or more courses that have received authorization from the College Board.

Through the AP Audit, teachers received a number of benefits. For example, you or your colleagues told us that the AP Audit helped you to obtain more current college textbooks for your students. A significant number of teachers said they were able to prevent the reduction of lab or instructional time that was scheduled to affect their courses. Because of the audit, 22,000 teachers said they were able to incorporate advances in their discipline that had not yet been added to their curricula. The searchable AP Course Ledger is online at collegeboard.com.

The College Board remains committed to supporting the work of AP teachers. AP workshops and Summer Institutes held around the world provide stimulating professional development for more than 60,000 teachers each year. Workshops provide teachers not only with valuable course-specific information but the opportunity to interact and network with their colleagues in the AP community.

This community is extended online at AP Central® where teachers can access a wide range of resources, information, and tools to support their work in the AP classroom. In response to requests from educators to make our Web site easier to use, the College Board implemented extensive improvements to collegeboard.com. A new "K–12 Teacher" homepage makes it easier to find an array of content and services. AP Central serves as an integral part of this enhanced collegeboard.com Web site.

We appreciate all of your efforts in the AP classroom and in the courses that prepare students for the rigor and challenge of AP. It is through the dedication and hard work of educators like you that a wider range of students than ever before is being given the opportunity to succeed in AP.

Sincerely,

[illegible]

Gaston Caperton
President
The College Board

Contents

Welcome to the AP® Program

The Advanced Placement Program® (AP) is a collaborative effort among motivated students; dedicated teachers; and committed high schools, colleges, and universities. Since its inception in 1955, the Program has enabled millions of students to take college-level courses and exams, and to earn college credit or placement, while still in high school.

Most colleges and universities in the United States, as well as colleges and universities in more than 40 other countries, have an AP policy granting incoming students credit, placement, or both on the basis of their AP Exam grades. Many of these institutions grant up to a full year of college credit (sophomore standing) to students who earn a sufficient number of qualifying AP grades.

Each year, an increasing number of parents, students, teachers, high schools, and colleges and universities turn to the AP Program as a model of educational excellence.

More information about the AP Program is available at the back of this Course Description and at AP Central, the College Board's online home for AP professionals (apcentral.collegeboard.com). Students can find more information at the AP student site (www.collegeboard.com/apstudents).

AP Courses

Thirty-seven AP courses in a wide variety of subject areas are available now. A committee of college faculty and master AP teachers designs each AP course to cover the information, skills, and assignments found in the corresponding college course. See page 2 for a complete list of AP courses and exams.

AP Exams

Each AP course has a corresponding exam that participating schools worldwide administer in May (except for AP Studio Art, which is a portfolio assessment). AP Exams contain multiple-choice questions and a free-response section (essay, problem solving, or oral response).

AP Exams are a culminating assessment in all AP courses and are thus an integral part of the Program. As a result, many schools foster the expectation that students who enroll in an AP course will take the corresponding AP Exam. Because the College Board is committed to providing access to AP Exams for homeschooled students and students whose schools do not offer AP courses, it does not require students to take an AP course prior to taking an AP Exam.

AP Course Audit

The AP Course Audit was created at the request of secondary school and college and university members of the College Board who sought a means to provide teachers and administrators with clear guidelines on the curricular and resource requirements that must be in place for AP courses. The AP Course Audit also helps colleges and universities better interpret secondary school courses marked "AP" on students' transcripts. To receive authorization from the College Board to label a course "AP," schools must demonstrate how their courses meet or exceed these requirements, which colleges and universities expect to see within a college-level curriculum.

The AP Program unequivocally supports the principle that each individual school must develop its own curriculum for courses labeled “AP.” Rather than mandating any one curriculum for AP courses, the AP Course Audit instead provides each AP teacher with a set of expectations that college and secondary school faculty nationwide have established for college-level courses. AP teachers are encouraged to develop or maintain their own curriculum that either includes or exceeds each of these expectations; such courses will be authorized to use the “AP” designation. Credit for the success of AP courses belongs to the individual schools and teachers that create powerful, locally designed AP curricula.

Complete information about the AP Course Audit is available at AP Central.

AP Courses and Exams

Art
Art History
Studio Art: 2-D Design
Studio Art: 3-D Design
Studio Art: Drawing

Biology

Calculus
Calculus AB
Calculus BC

Chemistry

Chinese Language and Culture

Computer Science
Computer Science A
Computer Science AB*

Economics
Macroeconomics
Microeconomics

English
English Language and Composition
English Literature and Composition

Environmental Science

French
French Language
French Literature*

German Language

Government and Politics
Comparative Government and Politics
United States Government and Politics

History
European History
United States History
World History

Human Geography

Italian Language and Culture*

Japanese Language and Culture

Latin
Latin Literature*
Latin: Vergil

Music Theory

Physics
Physics B
Physics C: Electricity and Magnetism
Physics C: Mechanics

Psychology

Spanish
Spanish Language
Spanish Literature

Statistics

*AP Computer Science AB, AP French Literature, and AP Latin Literature will be discontinued after the May 2009 exam administration. AP Italian may also be discontinued if external funding is not secured by May 2009. Visit AP Central for details.

AP Reading

AP Exams—with the exception of AP Studio Art, which is a portfolio assessment—consist of dozens of multiple-choice questions scored by machine, and free-response questions scored at the annual AP Reading by thousands of college faculty and expert AP teachers. AP Readers use scoring standards developed by college and university faculty who teach the corresponding college course. The AP Reading offers educators both significant professional development and the opportunity to network with colleagues. For more information about the AP Reading, or to apply to serve as a Reader, visit apcentral.collegeboard.com/readers.

AP Exam Grades

The Readers' scores on the free-response questions are combined with the results of the computer-scored multiple-choice questions; the weighted raw scores are summed to give a composite score. The composite score is then converted to a grade on AP's 5-point scale:

AP GRADE	QUALIFICATION
5	Extremely well qualified
4	Well qualified
3	Qualified
2	Possibly qualified
1	No recommendation

AP Exam grades of 5 are equivalent to A grades in the corresponding college course. AP Exam grades of 4 are equivalent to grades of A−, B+, and B in college. AP Exam grades of 3 are equivalent to grades of B−, C+, and C in college.

Credit and Placement for AP Grades

Thousands of four-year colleges grant credit, placement, or both for qualifying AP Exam grades, because these grades represent a level of achievement equivalent to that of students who take the corresponding college course. That college-level equivalency is ensured through several AP Program processes:

1. The involvement of college faculty in course and exam development and other AP activities. Currently, college faculty:
   - Serve as chairs and members of the committees that develop the Course Descriptions and exams in each AP course.
   - Are responsible for standard setting and are involved in the evaluation of student responses at the AP Reading. The Chief Reader for each AP subject is a college faculty member.
   - Teach professional development institutes for experienced and new AP teachers.
   - Serve as the senior reviewers in the annual AP Course Audit, ensuring AP teachers' syllabi meet the curriculum guidelines of college-level courses.

2. AP courses and exams are reviewed and updated regularly based on the results of curriculum surveys at up to 200 colleges and universities, collaborations among the College Board and key educational and disciplinary organizations, and the interactions of committee members with professional organizations in their discipline.
3. Periodic college comparability studies are undertaken in which the performance of college students on AP Exams is compared with that of AP students to confirm that the AP grade scale of 1 to 5 is properly aligned with current college standards.

For more information about the role of colleges and universities in the AP Program, visit the Higher Ed Services section of collegeboard.com at professionals .collegeboard.com/higher-ed.

Setting Credit and Placement Policies for AP Grades

The College Board Web site for education professionals has a section geared toward colleges and universities that provides guidance in setting AP credit and placement policies and additional resources, including links to AP research studies, released exam questions, and sample student responses at varying levels of achievement for each AP Exam. Visit professionals.collegeboard.com/higher-ed/placement/ap.

The AP Credit Policy Info online search tool provides links to credit and placement policies at more than 1,000 colleges and universities. The tool helps students find the credit hours and advanced placement they can receive for qualifying exam scores within each AP subject. AP Credit Policy Info is available at www.collegeboard.com/ap/creditpolicy.

AP Computer Science

Important Revisions to This Course Description

- Clarification on required skill levels for inheritance in AP CS A and AB courses (see Topic Outline, page 9; Commentary on the Topic Outline, page 13; Java Subset in Appendix A, page 92)
- Clarification on required skill levels for recursion in AP CS A and AB courses (see Topic Outline, page 9)
- Movement of advanced algorithms from the AP CS A course to the AP CS AB course (see Topic Outline, page 9; Commentary on the Topic Outline, page 13)
- Removal of the course goal for identifying hardware and software components (see Topic Outline, page 9)

INTRODUCTION

The Advanced Placement Program offers two computer science courses: Computer Science A and Computer Science AB.* The content of Computer Science A is a subset of the content of Computer Science AB. Computer Science A emphasizes object-oriented programming methodology with a concentration on problem solving and algorithm development and is meant to be the equivalent of a first-semester college-level course in Computer Science. It also includes the study of data structures, design, and abstraction, but these topics are not covered to the extent that they are in Computer Science AB. Computer Science AB includes all the topics of Computer Science A, as well as a more formal and in-depth study of algorithms, data structures, design, and abstraction. For example, binary trees are studied in Computer Science AB but not in Computer Science A. For a listing of the topics covered, see the AP Computer Science topic outline on pages 9–13.

Computer Science A may be appropriate for schools offering an AP Computer Science course for the first time, for schools whose faculty members have not yet developed sufficient expertise to cover the material in Computer Science AB, or for schools wishing to offer a choice of courses.

The nature of both AP courses is suggested by the words "computer science" in their titles. Their presence indicates a disciplined approach to a more broadly conceived subject than would a descriptor such as "computer programming." There are no computing prerequisites for either AP course. Each is designed to serve as a first course in computer science for students with no prior computing experience.

Because of the diversity of introductory computer science courses currently offered by colleges and universities, the outline of topics described here may not match any sequence of courses exactly. The Association for Computing Machinery (ACM) and the Institute of Electrical and Electronic Engineers (IEEE) Computer Society have published standards for the content of a college-level program in computer science that include recommendations for topics to be covered in the first two years of college.

*The AP Computer Science AB course and exam will be discontinued after the 2008-09 academic year. The final AP Computer Science AB exam will be offered in May 2009.

The AP Computer Science A course is compatible with those topics that are covered in a typical CS1 course as described in the example curricula in the ACM/IEEE guidelines. The additional topics in the AP Computer Science AB course are consistent with a CS2 course in those sample curricula. Some colleges and universities may organize their curricula in alternative ways so that the topics of the AP Computer Science A and AB courses are spread over the first three or four college courses, with other topics from computer science interspersed.

Either AP Computer Science course can be offered by any secondary school that has faculty who possess the necessary expertise and have access to appropriate computing facilities. These courses represent college-level achievement for which most colleges and universities can be expected to grant advanced placement and credit. Placement and credit are granted by institutions in accordance with their own policies, not by the College Board or the AP Program.

THE COURSES

The AP Computer Science courses are introductory courses in computer science. Because the development of computer programs to solve problems is a skill fundamental to the study of computer science, a large part of the course is built around the development of computer programs or parts of programs that correctly solve a given problem. The course also emphasizes the design issues that make programs understandable, adaptable, and, when appropriate, reusable. At the same time, the development of useful computer programs and classes is used as a context for introducing other important concepts in computer science, including the development and analysis of algorithms, the development and use of fundamental data structures, and the study of standard algorithms and typical applications. In addition, an understanding of the basic hardware and software components of computer systems and the responsible use of these systems are integral parts of the course. The topic outline on pages 9–13 summarizes the content typically covered in the two AP Computer Science courses.

Goals

The goals of an AP course in computer science are comparable to those in the introductory sequence of courses for computer science majors offered in college and university computer science departments. It is not expected, however, that all students in an AP Computer Science course will major in computer science at the university level. An AP Computer Science course is intended to serve both as an introductory course for computer science majors and as a course for people who will major in other disciplines that require significant involvement with technology. It is not a substitute for the usual college-preparatory mathematics courses.

The following goals apply to both of the AP Computer Science courses when interpreted within the context of the specific course. Students should be able to:

- design and implement solutions to problems by writing, running, and debugging computer programs
- use and implement commonly-used algorithms and data structures

- develop and select appropriate algorithms and data structures to solve problems
- code fluently in an object-oriented paradigm using the programming language Java. Students are expected to be familiar with and be able to use standard Java library classes from the AP Java subset
- read and understand a large program consisting of several classes and interacting objects. Students should be able to read and understand a description of the design and development process leading to such a program. (An example of such a program is the *AP Computer Science Case Study.*)
- recognize the ethical and social implications of computer use

Computer Language

The content of the college-level introductory programming course has evolved significantly over the years. Starting as a treatment merely of language features, it eventually incorporated first the notions of procedures and procedural abstraction, then the use of modules and data abstraction. At most institutions, the current introductory programming course takes an object-oriented approach to programming that is based on encapsulating procedures and data and creating programs with interacting objects. The AP Computer Science courses have evolved to incorporate this approach.

Current offerings of the AP Computer Science Exam require the use of Java. Those sections of the exam that require the reading or writing of actual programs will use Java. The exam will not cover all the features of Java; it will be consistent with the AP Java subset. (See Appendix A.) The AP Java subset can be found in the Computer Science section of AP Central (apcentral.collegeboard.com). **Students who study a language other than Java during an AP Computer Science course will need to be prepared to use standard Java, as specified in the AP Java subset, on the AP Computer Science Exams.**

Equipment

Students should have access to a computer system that represents relatively recent technology. The system must be able to compile in seconds programs comparable in size to the current *AP Computer Science Case Study,* and response time should be reasonably rapid. This will require large hard disk drives either on individual machines or shared via a network.

Each student in the course should have a minimum of three hours per week alone on a computer throughout the academic year; additional time is desirable. This access can be made available at any time during the school day or after school and need not be made available to all students in the AP course simultaneously. It should be stressed that (1) this requirement represents a bare minimum of access; and (2) this time is not instructional time at a computer with the teacher or a tutor but is time that the student spends alone at a computer in addition to the instructional time. Schools that do not allow their facilities to be used after school hours may wish to reevaluate such a policy in light of the needs of their students who take an AP Computer Science course.

Schools offering AP Computer Science will need to have Java software and enough memory in their lab machines so that students will be able to compile and run Java programs efficiently. Both free and commercial Java systems are available from a variety of sources. At a minimum, the hardware configuration will need large hard drives and sufficient memory to support current operating systems and compilers.

Prerequisites

The necessary prerequisites for entering either of the AP Computer Science courses include knowledge of basic algebra and experience in problem solving. A student in either AP Computer Science course should be comfortable with functions and the concepts found in the uses of functional notation, such as `f(x) = x + 2` and `f(x) = g(h(x))`. It is important that students and their advisers understand that any significant computer science course builds upon a foundation of mathematical reasoning that should be acquired before attempting such a course.

Schools that offer Computer Science AB may have students who do well on the topics covering programming methodology but do not deal as well with the more advanced material on data structures and analysis of algorithms. Such students might be advised to concentrate instead on the topics listed for Computer Science A and to take the Computer Science A Exam.

Some schools may offer only Computer Science A and encourage students who move at a faster pace to study the topics covered by the Computer Science AB outline. Then, they can take the AP Exam in Computer Science AB rather than the AP Exam in Computer Science A. Other schools may offer both courses and restrict enrollment in Computer Science AB to students who have some prior programming experience. Some schools may offer each course for a full year, while others cover each in one semester. This will vary according to the background of the students and the teacher.

One prerequisite for an AP Computer Science course, competence in written communication, deserves special attention. Documentation plays a central role in the programming methodology that forms the heart of an AP Computer Science course. Students should have already acquired facility in written communication before entering such a course.

Teaching the Courses

The teacher should be prepared to present a college-level first course in computer science. Each AP Computer Science course is more than a course on programming. The emphasis in these courses is on procedural and data abstraction, object-oriented programming and design methodology, algorithms, and data structures.

Because of the dynamic nature of the computer science field, AP Computer Science teachers will continually need to update their skills. Some resources that may assist teachers in professional development are AP Computer Science workshops and summer institutes, and Web sites such as AP Central. For information on workshops, teachers should contact their College Board regional office or go to AP Central.

One particular area of change is the evolution of programming languages and programming paradigms. Teachers should endeavor to keep current in this area by investigating different programming languages.

TOPIC OUTLINE

Following is an outline of the major topics covered by the AP Computer Science Exams. This outline is intended to define the scope of the course but not necessarily the sequence. The topics in the right-hand column will not be tested on the Computer Science A Exam.

I. Object-Oriented Program Design

The overall goal for designing a piece of software (a computer program) is to correctly solve the given problem. At the same time, this goal should encompass specifying and designing a program that is understandable, can be adapted to changing circumstances, and has the potential to be reused in whole or in part. The design process needs to be based on a thorough understanding of the problem to be solved.

Computer Science A and AB	***Computer Science AB only***
A. Program design	
1. Read and understand a problem description, purpose, and goals.	1. Specify the purpose and goals for a problem.
2. Apply data abstraction and encapsulation.	
3. Read and understand class specifications and relationships among the classes ("is-a," "has-a" relationships).	3. Decompose a problem into classes; define relationships and responsibilities of those classes.
4. Understand and implement a given class hierarchy.	
5. Identify reusable components from existing code using classes and class libraries.	
B. Class design	
1. Design and implement a class.	1. Design and implement a set of interacting classes.
	2. Design an interface.
3. Choose appropriate data representation and algorithms.	3. Choose appropriate advanced data structures and algorithms.
4. Apply functional decomposition.	
5. Extend a given class using inheritance.	5. Design and implement inheritance hierarchies.

II. Program Implementation

The overall goals of program implementation parallel those of program design. Classes that fill common needs should be built so that they can be reused easily in other programs. Object-oriented design is an important part of program implementation.

Computer Science A and AB	***Computer Science AB only***
A. Implementation techniques	
1. Methodology	
a. Object-oriented development	
b. Top-down development	
c. Encapsulation and information hiding	
d. Procedural abstraction	
B. Programming constructs	
1. Primitive types vs. objects	
2. Declaration	
a. Constant declarations	
b. Variable declarations	
c. Class declarations	
d. Interface declarations	
e. Method declarations	
f. Parameter declarations	
3. Console output (System.out.print/println)	
4. Control	
a. Methods	
b. Sequential	
c. Conditional	
d. Iteration	
e. Understand and evaluate recursive methods	e. Design and implement recursive solutions
C. Java library classes (included in the A-level AP Java Subset)	C. Java library classes (included in the AB-level AP Java Subset)

III. Program Analysis

The analysis of programs includes examining and testing programs to determine whether they correctly meet their specifications. It also includes the analysis of programs or algorithms in order to understand their time and space requirements when applied to different data sets.

Computer Science A and AB	***Computer Science AB only***
A. Testing	
1. Test classes and libraries in isolation.	
2. Identify boundary cases and generate appropriate test data.	
3. Perform integration testing.	
B. Debugging	
1. Categorize errors: compile-time, run-time, logic.	
2. Identify and correct errors.	
3. Employ techniques such as using a debugger, adding extra output statements, or hand-tracing code.	
C. Understand and modify existing code	
D. Extend existing code using inheritance	
E. Understand error handling	
1. Understand runtime exceptions.	
	2. Throw runtime exceptions
F. Reason about programs	
1. Pre- and post-conditions	
2. Assertions	
G. Analysis of algorithms	
1. Informal comparisons of running times	
2. Exact calculation of statement execution counts	
	3. Big-Oh notation
	4. Worst-case and average-case time and space analysis
H. Numerical representations and limits	
1. Representations of numbers in different bases	
2. Limitations of finite representations (e.g., integer bounds, imprecision of floating-point representations, and round-off error)	

IV. Standard Data Structures

Data structures are used to represent information within a program. Abstraction is an important theme in the development and application of data structures.

Computer Science A and AB	***Computer Science AB only***
A. Simple data types (int, boolean, double)	
B. Classes	
C. One-dimensional arrays	
	D. Two-dimensional arrays
	E. Linked lists (singly, doubly, circular)
	F. Stacks
	G. Queues
	H. Trees
	I. Heaps
	J. Priority queues
	K. Sets
	L. Maps

V. Standard Algorithms

Standard algorithms serve as examples of good solutions to standard problems. Many are intertwined with standard data structures. These algorithms provide examples for analysis of program efficiency.

Computer Science A and AB	***Computer Science AB only***
A. Operations on A-level data structures previously listed	A. Operations on AB-level data structures previously listed
1. Traversals	1. Traversals
2. Insertions	2. Insertions
3. Deletions	3. Deletions
	4. Iterators
B. Searching	
1. Sequential	
2. Binary	
	3. Hashing
C. Sorting	
1. Selection	
2. Insertion	
3. Mergesort	4. Quicksort
	5. Heapsort

VI. Computing in Context

An awareness of the ethical and social implications of computing systems is necessary for the study of computer science. These topics need not be covered in detail but should be considered throughout the course.

Computer Science A and AB	***Computer Science AB only***
A. System reliability B. Privacy C. Legal issues and intellectual property D. Social and ethical ramifications of computer use	

COMMENTARY ON THE TOPIC OUTLINE

The AP Computer Science (AP CS) course is an introductory course in computer science. Because the design and implementation of computer programs to solve problems involve skills that are fundamental to the study of computer science, a large part of the AP CS course is built around the development of computer programs that correctly solve a given problem. These programs should be understandable, adaptable, and, when appropriate, reusable. At the same time, the design and implementation of computer programs is used as a context for introducing other important aspects of computer science, including the development and analysis of algorithms, the development and use of fundamental data structures, the study of standard algorithms and typical applications, and the use of logic and formal methods. In addition, the responsible use of these systems is an integral part of the course. The topic outline summarizes the content of the AP CS curriculum. In this section, we provide more details about the topics in the outline.

I. Object-Oriented Program Design

Computer science involves the study of complex systems. Computer software is a part of a complex system. To understand the development of computer software, we need tools that can make sense of that complexity. Object-oriented design and programming form an approach that enables us to do that, based on the idea that a piece of software, just like a computer itself, is composed of many interacting parts.

The novice will start not by designing a whole program but rather by studying programs already developed, then writing or modifying parts of a program to add to or change its functionality. Only later in the first course will a student get to the point of working from a specification to develop a design for a program or part of a program.

In an object-oriented approach, the fundamental part of a program is an object, an entity that has state (stores some data) and operations that access or change its state and that may interact with other objects. Objects are defined by classes; a class specifies the components and operations of an object, and each object is an instance of a class.

A. Program Design

A student in a first computer science course (AP CS A) would learn to work with the design of a program but would not be expected to develop a full program design. However, this student should be able to work from a given design to develop the parts of the program. This would include an understanding of how to apply the data abstractions covered in the first course (classes and one-dimensional arrays).

Students in the first course should be able to understand the inheritance and composition relationships among the different classes that comprise a program. They should also be able to implement a class inheritance hierarchy when given the specifications for the classes involved—which classes are subclasses of other classes.

Students in the second course (AP CS AB) should learn to work from a given problem statement to define the purpose and goals of a program intended to solve that problem. They then should be able to decompose the problem into interacting objects and specify the classes needed to define those objects, as well as the relationships among those classes.

B. Class Design

A fundamental part of the development of an object-oriented program is the design of a class. Students in the first course should be able to design a class—write the class declaration including the instance variables and the method signatures (the method bodies would comprise the implementation of this design)—when they are given a description of the type of entity the class represents. Such a description would include the data that must be represented by the class and the operations that can be applied to that data. These operations range from simple access to the data or information that can be derived from the data, to operations that change the data (which stores the state) of an instance of the class. The design of a class includes decisions on appropriate data structures for storing data and algorithms for operations on that data. The decomposition of operations into subsidiary operations—functional decomposition—is part of the design process. An example of the process of designing a class is given in the sample free-response question, which documents the logical considerations for designing a savings account class.

A student in the second course (AP CS AB) should be able to develop a design for a set of interacting classes, given the specifications for those classes and their relationships. This student will also have a much broader set of data structures and algorithms to use in design decisions.

Given a design for a class, either their own or one provided, students in the first course should then be able to implement the class. They should also be able to extend a given class using inheritance, thereby creating a subclass with modified or additional functionality. While we expect most AP CS A students to have some exposure to the more complex and technical aspects of inheritance (such as writing abstract classes), these aspects are not tested on the AP CS A exam.

An *interface* is a specification for a set of operations that a class must implement. In Java, there is a specific construct, the `interface`, that can be specified for this purpose, so that another class can be specified to *implement* that interface. Students in the second course (AP CS AB) should be able to design an interface by declaring all its methods, given a specification of the operations that these methods represent.

C. Java Library Classes

An important aspect of modern programming is the existence of extensive libraries that supply many common classes and methods. One part of learning the skill of programming is to learn about available libraries and their appropriate use. The AP CS curriculum specifies the classes from the Java libraries with which students should be familiar, and students should be able to recognize the appropriate use of these classes.

In addition, students should recognize the possibilities of reuse of components of their own code or other examples of code, such as the *AP Computer Science Case Study*, in different programs.

D. Design as an Exam Topic

As noted in the topic outline, the A Exam may include questions that ask about the design as well as the implementation of classes or a simple hierarchy of classes. The AB Exam may include questions that ask about the design of multiple classes that specify interacting objects, as well as the implementation of such classes.

A design question would provide students with a description of the type of information and operations on that information that an object should encapsulate. Students would then be required to provide part or all of an interface or class declaration to define such objects. An example of this type of question appears as one of the sample free-response questions for Computer Science A (see page 49).

A design question may require a student to develop a solution that includes the following:

- appropriate use of inheritance from another class using the keyword `extends`
- appropriate implementation of an interface using the keyword `implements`
- declaration of constructors and methods with
  - meaningful names
  - appropriate parameters
  - appropriate return types
- appropriate data representation
- appropriate designation of methods as `public` or `private`
- all data declared `private`
- all client accessible operations specified as `public` methods

A design question might only require that a student specify the appropriate constructor and method signatures (access specifier, return type, method identifier, parameter list) and not require that the body of the constructors or methods be implemented. A question focusing on a simple class hierarchy might also require implementation of the body of some or all methods for some of the classes.

II. Program Implementation

To implement a program, one must understand the fundamental programming constructs of the language, as well as the design of the program. The fundamental principles of encapsulation and information hiding should be applied when implementing classes and data structures. A good program will often have components that can be used in other programs.

There are topics not included in the course outline that will be part of any introductory course. For example, input and output must be part of a course on computer programming. However, in a modern object-oriented approach to programming, there are many ways to handle input and output, including console-based character I/O, graphical user interfaces, and applets. Consequently, the AP CS course does not prescribe any particular approach and will not test the details of input and output (except for the basic console output, `System.out.print/ln` in Java), so that teachers may use an approach that fits their own style and whatever textbook and other materials they use.

In the A exam, students are expected to demonstrate an understanding of the concept of recursion and to trace recursive method calls. In the AB exam, students are expected to be able to design and implement recursive solutions to programming problems.

III. Program Analysis

Some of the techniques for finding and correcting errors, for "debugging" a program or segment of a program, include hand-tracing code, adding extra output statements to trace the execution of a program, or using a debugger to provide information about the program as it runs and when it crashes. Students should be encouraged to experiment with available debugging facilities. However, these will not be tested since they vary from system to system.

Students should be able to read and modify code for a program. They should also be able to extend existing code by taking a given class declaration and declaring a new class using inheritance to add or change the given class' functionality. The *AP Computer Science Case Study* contains examples of using inheritance to create new classes.

Students in the AP CS A course should understand runtime exceptions. Students in the AP CS AB course should also be able to write code to throw runtime exceptions under appropriate circumstances, such as an `IllegalArgumentException`.

Students need to be familiar with the concepts of preconditions, postconditions, and assertions and correctly interpret them when presented as pseudocode. The `assert` keyword of the Java language is not tested.

In the AP CS A course, students should be able to make informal comparisons of running times of different pieces of code: for example, by counting the number of loop iterations needed for a computation. In the AP CS AB course, students learn about asymptotic analysis of algorithms: how the algorithms behave as the data sets get larger and larger. Asymptotic analysis uses the "Big-Oh" notation to derive a bound for an algorithm's running time in terms of standard functions such as n, n^2, $log(n)$, etc. Students in the AP CS AB course should understand asymptotic analysis of running times for the worst case, average case (when it can be reasonably defined), and best case for standard searching and sorting algorithms. Students should also be able to analyze a given algorithm of moderate complexity. In addition, these students should be able to make a similar analysis of the space (memory) needed to carry out a given algorithm.

Many programs involve numerical computations and therefore are limited by the finite representations of numbers in a computer. Students should understand the representation of positive integers in different bases, particularly decimal, binary, hexadecimal, and octal. They should also understand the consequences of the finite representations of integer and real numbers, including the limits on the magnitude of numbers represented, the imprecision of floating point computation, and round-off error.

IV. Standard Data Structures

There are a number of standard data structures used in programming. Students should understand these data structures and their appropriate use. For the AP Computer Science A and AB courses, students need to be able to use the standard representations of integers, real numbers, and Boolean (logical) variables. The other primitive types in Java, `char` and `float`, are not part of the AP Java subset but may be useful in an AP CS course.

Students are responsible for understanding the Java `String` class and the methods of the `String` class that are listed in the AP Java subset (see Appendixes).

Students in the AP CS A course should be comfortable working with one-dimensional arrays. Students in both the AP CS A and AB courses should be familiar with both Java arrays and the structure `ArrayList`. They should be able to use either in a program and should be able to select the most appropriate one for a given application. The methods for `ArrayList` for which students are responsible are specified in the AP Java subset (see Appendixes).

The AP CS AB course has a major focus on abstract data types and data structures. Consequently, there are several additional data structures that these students must understand. The following abstract data types should be covered in the AP CS AB course:

- lists
- stacks
- queues
- priority queues
- sets
- maps

There are a number of data structures that can be used to implement these abstractions. They include, of course, the one-dimensional arrays that are part of the AP CS A course. Other fundamental data structures are:

- two-dimensional arrays
- linked lists, including singly, doubly, and circular
- trees, including binary trees
- heaps (and their standard array implementation)
- hash tables

Students should be able to implement the abstract data types with appropriate data structures.

In addition, many implementations of abstract data types are given as part of the Java libraries. Those that students are required to understand are specified in the AP Java subsets for the A and AB courses. These include `ArrayList` for the A course and the following for the AB course:

- `List` interface
- `LinkedList`
- `Set` interface
- `HashSet`
- `TreeSet`
- `Map` interface
- `HashMap`
- `TreeMap`
- `Iterator` interface
- `ListIterator` interface
- `Queue` interface
- `Stack`
- `PriorityQueue`

These are all found in the `java.util` package.

The Java library structures `ArrayList`, `LinkedList`, `HashSet`, `TreeSet`, `HashMap`, and `TreeMap`, which implement the data types `List`, `Set`, and `Map`, all have specifications about their runtime in the standard documentation. For example, the documentation specifies that the `get` and `set` operations for an `ArrayList` can be done in constant time. On the other hand, the documentation indicates that the operations for `LinkedList` take time for a doubly linked list implementation, so a `set` operation would take `O(n)` for a list of `n` items in the worst case. Students should understand these time estimates so that they can estimate the asymptotic time for operations involving these data structures.

The following is a list of some useful information that can be found in the Java documentation:

1. Adding one item to the front of an `ArrayList` of size `n` is `O(n)`. Adding an item to the end of an `ArrayList` takes `O(1)` time. (Technically the `O(1)` bound for adding an item to the end of an `ArrayList` is an amortized bound. This means that adding `n` items to the end of an empty `ArrayList` takes `O(n)` time, averaging `O(1)` time per operation, even though some operations can take longer. This technicality will not be evaluated on the AP Exams.)
2. `LinkedList` is implemented as a doubly linked list with head and tail links. An attempt to access the kth item in the list will start at the end of the list that is closer.
3. `TreeSets` and `TreeMaps` are implemented as balanced binary trees, so `add`, `contains`, and `remove` for `TreeSet` and `put`, `get`, and `containsKey` for `TreeMap` are all `O(log n)` worst case.
4. Operations on `HashSet` and `HashMap` are `O(1)` expected time, but could be `O(n)` worst-case time.
5. `Iterator` and `ListIterator` for `List` return the elements in the order they appear in the list.
6. Items inserted in `TreeSet` and keys inserted in `TreeMap` must be `Comparable` (in the AP Java subset).
7. `Iterator` for `TreeSet` returns the elements in the order specified by `compareTo`.
8. `Iterator` for `HashSet` returns elements in an arbitrary order.
9. An `Iterator` or `ListIterator` can iterate through all elements in a collection in `O(n)` time. (Note: This is limited to the collections listed in the AP Java subset.) For a `HashSet`, `n` is the maximum size the `HashSet` has ever been; for other `Collection` classes, `n` is the current size. For a `TreeSet` iterator, the first call to `next()` takes `O(log n)` time.

See the AP Java subset for details, including the methods that students are required to know. AP CS AB students should know that some classes implement some interfaces (e.g., `LinkedList implements List`).

V. Standard Algorithms

Both the AP CS A and AB courses cover several standard algorithms. These serve as good solutions to standard problems. These algorithms, many of which are intertwined with data structures, provide excellent examples for the analysis of program efficiency. Programs implementing standard algorithms also serve as good models for program design.

The AP CS A course includes standard algorithms for accessing arrays, including traversing an array and inserting into and deleting from an array. Students should also know the two standard searches, sequential search and binary search, and the relative efficiency of each. Finally, there are three standard sorts that are required for the AP

CS A course: the two most common quadratic sorts—Selection sort and Insertion sort—and the more efficient Mergesort. Of course, the latter implies that students know the merge algorithm for sorted lists.

Students in the AP CS A course are not required to know the asymptotic (Big-Oh) analysis of these algorithms, but they should understand that Mergesort is advantageous for large data sets and be familiar with the differences between Selection and Insertion sort.

In addition to the searching and sorting algorithms listed for the AP CS A course, students in the AP CS AB course should also understand Quicksort and Heapsort. For all these standard algorithms, students in the AP CS AB course should understand the asymptotic complexity.

An important part of the AP CS AB course is the understanding and analysis of algorithms associated with the standard data structures. These include traversing the structures so as to access all elements and adding and removing elements from the structures. Iterators are an abstraction of the process of traversing a structure. Java has both `Iterator` and `ListIterator` interfaces that implement this abstraction, and students should be familiar with their use.

A hash table provides a structure for which each insertion and search operation can be carried out in constant time. Students in the AP CS AB course should understand hash tables and be able to use the Java library implementations `HashSet` and `HashMap`.

VI. Computing in Context

Given the tremendous impact computers and computing have on almost every aspect of society, it is important that intelligent and responsible attitudes about the use of computers be developed as early as possible. The applications of computing that are studied in an AP CS course provide opportunities to discuss the impact of computing. Typical issues include the:

- impact of applications using databases, particularly over the Internet, on an individual's right to privacy;
- economic and legal impact of viruses and other malicious attacks on computer systems;
- need for fault-tolerant and highly reliable systems for life-critical applications and the resulting need for software engineering standards; and
- intellectual property rights of writers, musicians, and computer programmers and fair use of intellectual property.

Attitudes are acquired, not taught. Hence, references to responsible use of computer systems should be integrated into an AP CS course wherever appropriate, rather than taught as a separate unit. Participation in an AP CS course provides an opportunity to discuss such issues as the responsible use of a system and respect for the rights and property of others. Students should learn to take responsibility for the programs they write and for the consequences of the use of their programs.

CASE STUDIES

Case studies provide a vehicle for presenting many of the topics of the AP Computer Science courses. They provide examples of good style, programming language constructs, fundamental data structures, algorithms, and applications. Large programs give the student practice in the management of complexity and motivate the use of certain programming practices (including decomposition into classes, use of inheritance and interfaces, message passing between interacting objects, and selection of data structures tailored to the needs of the classes) in a much more complete way than do small programs.

Case studies also allow the teacher to show concretely the design and implementation decisions leading to the solution of a problem and thus to focus more effectively on those aspects of the programming process. This approach gives the student a model of the programming process as well as a model program. The use of case studies also gives the student a context for seeing the importance of good design when a program is to be modified.

The AP Computer Science Exams will include questions based on the case study described in the document *AP Computer Science Case Study*. These questions may explore design choices, alternative choices of data structures, extending a class via inheritance, etc., in the context of a large program without requiring large amounts of reading during the exam. Both the AP CS A and AB Exams will contain several multiple-choice questions and one free-response question covering material from the case study. Printed excerpts from the case study programs will accompany the exams.

Questions will deal with activities such as the following:

a. modifying the procedural and data organization of the case study program to correspond to changes in the program specification;

b. extending the case study program by writing new code (including new methods for existing classes, new subclasses extending existing classes, and new classes);

c. evaluating alternatives in the representation and design of objects and classes;

d. evaluating alternative incremental development strategies; and

e. understanding how the objects/classes of the program interact.

Sample questions for the *AP Computer Science Case Study* appear on AP Central. The text and code for the *AP Computer Science Case Study* are available for downloading from AP Central.

THE EXAMS

The AP Exams for Computer Science A and Computer Science AB are each three hours long and seek to determine how well students have mastered the concepts and techniques contained in the respective course outlines. Before the exam date, students must decide which of the two exams they will take. In most cases, students will prepare during the year for one exam or the other. Some students enrolled in the AB course, however, may not feel comfortable with some of its more advanced topics. Such students might prefer to take the Computer Science A Exam.

Each exam consists of two sections: a multiple-choice section (40 questions in 1 hour and 15 minutes), which tests proficiency in a wide variety of topics, and a free-response section (4 questions in 1 hour and 45 minutes), which requires the student to demonstrate the ability to solve problems involving more extended reasoning.

The multiple-choice and the free-response sections of both AP Computer Science Exams require students to demonstrate their ability to design, write, analyze, and document programs and subprograms.

Minor points of syntax are not tested on the exams. All code given is consistent with the AP Java subset. All student responses involving code must be written in Java. Students are expected to be familiar with and able to use the standard Java classes listed in the AP Java subset. For both the multiple-choice and the free-response sections of the exams, an appendix containing a quick reference to both the case study and the classes in the AP Java subset will be provided.

In the determination of the grade for each exam, the multiple-choice section and the free-response section are given equal weight. Because each exam is designed for full coverage of the subject matter, it is not expected that many students will be able to correctly answer all the questions in either the multiple-choice section or the free-response section.

The Appendix mentioned in the **Notes** in test directions on pages 23, 41, 59, and 75 refers to material students receive on exam day, not an Appendix in this Course Description.

Computer Science A: Sample Multiple-Choice Questions

Following is a representative set of questions. Questions marked with an asterisk are also representative of AB Exam questions. The answer key for the Computer Science A multiple-choice questions is on page 40. In this section of the exam, as a correction for haphazard guessing, one-fourth of the number of questions answered incorrectly will be subtracted from the number of questions answered correctly. The AP Computer Science A Exam will include several multiple-choice questions based on the *AP Computer Science Case Study.* (See AP Central for examples.)

Directions: Determine the answer to each of the following questions or incomplete statements, using the available space for any necessary scratchwork. Then decide which is the best of the choices given and fill in the corresponding oval on the answer sheet. No credit will be given for anything written in the examination booklet. Do not spend too much time on any one problem.

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Assume that declarations of variables and methods appear within the context of an enclosing class.
- Assume that method calls that are not prefixed with an object or class name and are not shown within a complete class definition appear within the context of an enclosing class.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null`.

1. Consider the following code segment.

```
for (int k = 0; k < 20; k = k + 2)
{
  if (k % 3 == 1)
    System.out.print(k + " ");
}
```

What is printed as a result of executing the code segment?

(A) `4  16`

(B) `4  10  16`

(C) `0  6  12  18`

(D) `1  4  7  10  13  16  19`

(E) `0  2  4  6  8  10  12  14  16  18`

2. Consider the following code segment.

```
ArrayList<String> list = new ArrayList<String>();

list.add("P");
list.add("Q");
list.add("R");
list.set(2, "s");
list.add(2, "T");
list.add("u");
System.out.println(list);
```

What is printed as a result of executing the code segment?

(A) `[P, Q, R, s, T]`

(B) `[P, Q, s, T, u]`

(C) `[P, Q, T, s, u]`

(D) `[P, T, Q, s, u]`

(E) `[P, T, s, R, u]`

*3. Consider the following instance variable and method.

```
private ArrayList<Integer> nums;

/** Precondition: nums.size > 0
 */
public void numQuest()
{
  int k = 0;
  Integer zero = new Integer(0);

  while (k < nums.size())
  {
    if (nums.get(k).equals(zero))
      nums.remove(k);

    k++;
  }
}
```

Assume that `ArrayList nums` initially contains the following `Integer` values.

```
[0, 0, 4, 2, 5, 0, 3, 0]
```

What will `ArrayList nums` contain as a result of executing `numQuest` ?

(A) `[0, 0, 4, 2, 5, 0, 3, 0]`

(B) `[4, 2, 5, 3]`

(C) `[0, 0, 0, 0, 4, 2, 5, 3]`

(D) `[3, 5, 2, 4, 0, 0, 0, 0]`

(E) `[0, 4, 2, 5, 3]`

4. At a certain high school students receive letter grades based on the following scale.

Numeric Score	Letter Grade
93 or above	A
From 84 to 92 inclusive	B
From 75 to 83 inclusive	C
Below 75	F

Which of the following code segments will assign the correct string to `grade` for a given integer score ?

I.
```
if (score >= 93)
  grade = "A";
if (score >= 84 && score <= 92)
  grade = "B";
if (score >= 75 && score <= 83)
  grade = "C";
if (score < 75)
  grade = "F";
```

II.
```
if (score >= 93)
  grade = "A";
if (84 <= score <= 92)
  grade = "B";
if (75 <= score <= 83)
  grade = "C";
if (score < 75)
  grade = "F";
```

III.
```
if (score >= 93)
  grade = "A";
else if (score >= 84)
  grade = "B";
else if (score >= 75)
  grade = "C";
else
  grade = "F";
```

(A) II only

(B) III only

(C) I and II only

(D) I and III only

(E) I, II, and III

5. Consider the following output.

```
1  1  1  1  1
2  2  2  2
3  3  3
4  4
5
```

Which of the following code segments will produce this output?

(A)
```
for (int j = 1; j <= 5; j++)
{
  for (int k = 1; k <= 5; k++)
  {
    System.out.print(j + " ");
  }
  System.out.println();
}
```

(B)
```
for (int j = 1; j <= 5; j++)
{
  for (int k = 1; k <= j; k++)
  {
    System.out.print(j + " ");
  }
  System.out.println();
}
```

(C)
```
for (int j = 1; j <= 5; j++)
{
  for (int k = 5; k >= 1; k--)
  {
     System.out.print(j + " ");
  }
  System.out.println();
}
```

(D)
```
for (int j = 1; j <= 5; j++)
{
  for (int k = 5; k >= j; k--)
  {
     System.out.print(j + " ");
   }
   System.out.println();
}
```

(E)
```
for (int j = 1; j <= 5; j++)
{
  for (int k = j; k <= 5; k++)
  {
    System.out.print(k + " ");
  }
  System.out.println();
}
```

6. A car dealership needs a program to store information about the cars for sale. For each car, they want to keep track of the following information: number of doors (2 or 4), whether the car has air conditioning, and its average number of miles per gallon. Which of the following is the best design?

   (A) Use one class, `Car`, which has three data fields: `int numDoors`, `boolean hasAir`, and `double milesPerGallon`.

   (B) Use four unrelated classes: `Car`, `Doors`, `AirConditioning`, and `MilesPerGallon`.

   (C) Use a class `Car` which has three subclasses: `Doors`, `AirConditioning`, and `MilesPerGallon`.

   (D) Use a class `Car`, which has a subclass `Doors`, with a subclass `AirConditioning`, with a subclass `MilesPerGallon`.

   (E) Use three classes: `Doors`, `AirConditioning`, and `MilesPerGallon`, each with a subclass `Car`.

7. Consider the following declarations.

```
public interface Comparable

{

  int compareTo(Object other);

}

public class SomeClass implements Comparable
{
  // ... other methods not shown
}
```

Which of the following method signatures of `compareTo` will satisfy the `Comparable` interface requirement?

```
I.   public int compareTo(Object other)
II.  public int compareTo(SomeClass other)
III. public boolean compareTo(Object other)
```

(A) I only

(B) II only

(C) III only

(D) I and II only

(E) I, II, and III

Questions 8–9 refer to the following incomplete class declaration.

```
public class TimeRecord
{
  private int hours;
  private int minutes;  // 0<=minutes<60

  public TimeRecord(int h, int m)
  {
    hours = h;
    minutes = m;
  }

  /** @return the number of hours
   */
  public int getHours()
  { /* implementation not shown */ }

  /** @return the number of minutes
   *          Postcondition: 0 ≤ minutes < 60
   */
  public int getMinutes()
  { /* implementation not shown */ }


  /** Adds h hours and m minutes to this TimeRecord.
   *  @param h the number of hours
   *          Precondition: h ≥ 0
   *  @param m the number of minutes
   *          Precondition: m ≥ 0
   */
  public void advance(int h, int m)
  {
    hours = hours + h;
    minutes = minutes + m;

    /* missing code */
  }

  // ... other methods not shown

}
```

8. Which of the following can be used to replace `/* missing code */` so that `advance` will correctly update the time?

(A) `minutes = minutes % 60;`

(B) `minutes = minutes + hours % 60;`

(C)
```
hours = hours + minutes / 60;
minutes = minutes % 60;
```

(D)
```
hours = hours + minutes % 60;
minutes = minutes / 60;
```

(E) `hours = hours + minutes / 60;`

9. Consider the following declaration that appears in a client program.

```
TimeRecord[] timeCards = new TimeRecord[100];
```

Assume that `timeCards` has been initialized with `TimeRecord` objects. Consider the following code segment that is intended to compute the total of all the times stored in `timeCards`.

```
TimeRecord total = new TimeRecord(0,0);

for (int k = 0; k < timeCards.length; k++)
{
  /* missing expression */ ;
}
```

Which of the following can be used to replace `/* missing expression */` so that the code segment will work as intended?

(A) `timeCards[k].advance()`

(B) `total += timeCards[k].advance()`

(C)
```
total.advance(timeCards[k].hours,
              timeCards[k].minutes)
```

(D)
```
total.advance(timeCards[k].getHours(),
              timeCards[k].getMinutes())
```

(E)
```
timeCards[k].advance(timeCards[k].getHours(),
                     timeCards[k].getMinutes())
```

*10. Consider the following instance variable and method.

```
private int[] arr;

/** Precondition:  arr contains no duplicates;
 *                 the elements in arr are in sorted order.
 *  @param low 0 ≤ low ≤ arr.length
 *  @param high low - 1 ≤ high < arr.length
 *  @param num
 */
public int mystery(int low, int high, int num)
{
  int mid = (low + high) / 2;

  if (low > high)
  {
    return low;
  }
  else if (arr[mid] < num)
  {
    return mystery(mid + 1, high, num);
  }
  else if (arr[mid] > num)
  {
    return mystery(low, mid - 1, num);
  }
  else            // arr[mid] == num
  {
    return mid;
  }
}
```

What is returned by the call
`mystery(0, arr.length - 1, num)` ?

(A) The number of elements in `arr` that are less than `num`

(B) The number of elements in `arr` that are less than or equal to `num`

(C) The number of elements in `arr` that are equal to `num`

(D) The number of elements in `arr` that are greater than `num`

(E) The index of the middle element in `arr`

Questions 11–12 refer to the following information.

Consider the following instance variable and method `findLongest` with line numbers added for reference. Method `findLongest` is intended to find the longest consecutive block of the value `target` occurring in the array `nums`; however, `findLongest` does not work as intended.

For example, if the array `nums` contains the values
`[7, 10, 10, 15, 15, 15, 15, 10, 10, 10, 15, 10, 10]`,
the call `findLongest(10)` should return 3, the length of the longest consecutive block of 10's.

```
          private int[] nums;

          public int findLongest(int target)
          {
            int lenCount = 0;
            int maxLen = 0;

Line 1:     for (val : nums)
Line 2:     {
Line 3:       if (val == target)
Line 4:       {
Line 5:         lenCount++;
Line 6:       }
Line 7:       else
Line 8:       {
Line 9:         if (lenCount > maxLen)
Line 10:        {
Line 11:          maxLen = lenCount;
Line 12:        }
Line 13:      }
Line 14:    }
Line 15:    if (lenCount > maxLen)
Line 16:    {
Line 17:      maxLen = lenCount;
Line 18:    }
Line 19:    return maxLen;
          }
```

*11. The method `findLongest` does not work as intended. Which of the following best describes the value returned by a call to `findLongest` ?

(A) It is the length of the shortest consecutive block of the value `target` in `nums`.

(B) It is the length of the array `nums`.

(C) It is the number of occurrences of the value `target` in `nums`.

(D) It is the length of the first consecutive block of the value `target` in `nums`.

(E) It is the length of the last consecutive block of the value `target` in `nums`.

*12. Which of the following changes should be made so that method `findLongest` will work as intended?

(A) Insert the statement `lenCount = 0;` between lines 2 and 3.

(B) Insert the statement `lenCount = 0;` between lines 8 and 9.

(C) Insert the statement `lenCount = 0;` between lines 10 and 11.

(D) Insert the statement `lenCount = 0;` between lines 11 and 12.

(E) Insert the statement `lenCount = 0;` between lines 12 and 13.

*13. Consider the following instance variable and method.

```
private int[] myStuff;

/** Precondition: myStuff contains int values in no particular order.
 */
public int mystery(int num)
{
  for (int k = myStuff.length - 1; k >= 0; k--)
  {
    if (myStuff[k] < num)
    {
      return k;
    }
  }

  return -1;
}
```

Which of the following best describes the contents of `myStuff` after the following statement has been executed?

```
    int m = mystery(n);
```

(A) All values in positions `0` through `m` are less than `n`.

(B) All values in positions `m+1` through `myStuff.length-1` are less than `n`.

(C) All values in positions `m+1` through `myStuff.length-1` are greater than or equal to `n`.

(D) The smallest value is at position `m`.

(E) The largest value that is smaller than `n` is at position `m`.

14. Consider the following method.

```
/** @param x x ≥ 0
 */
public void mystery(int x)
{
  System.out.print(x % 10);

  if ((x / 10) != 0)
  {
    mystery(x / 10);
  }

  System.out.print(x % 10);
}
```

Which of the following is printed as a result of the call `mystery(1234)` ?

(A) `1441`

(B) `3443`

(C) `12344321`

(D) `43211234`

(E) Many digits are printed due to infinite recursion.

15. Consider the following two classes.

```
public class Dog
{
  public void act()
  {
    System.out.print("run");
    eat();
  }

  public void eat()
  {
    System.out.print("eat");
  }
}

public class UnderDog extends Dog
{
  public void act()
  {
    super.act();
    System.out.print("sleep");
  }

  public void eat()
  {
    super.eat();
    System.out.print("bark");
  }
}
```

Assume that the following declaration appears in a client program.

```
Dog fido = new UnderDog();
```

What is printed as a result of the call `fido.act()` ?

(A) `run eat`

(B) `run eat sleep`

(C) `run eat sleep bark`

(D) `run eat bark sleep`

(E) Nothing is printed due to infinite recursion.

*16. Consider the following recursive method.

```
public static int mystery(int n)
{
   if (n == 0)
      return 1;
   else
      return 3 * mystery(n - 1);
}
```

What value is returned as a result of the call `mystery(5)` ?

(A) 0

(B) 3

(C) 81

(D) 243

(E) 6561

*17. Consider the following instance variable and method.

```
private int[] arr;

/** Precondition: arr.length > 0
 */
public int checkArray()
{
  int loc = arr.length / 2;

  for (int k = 0; k < arr.length; k++)
  {
    if (arr[k] > arr[loc])
      loc = k;
  }

  return loc;
}
```

Which of the following is the best postcondition for `checkArray` ?

(A) Returns the index of the first element in array `arr` whose value is greater than `arr[loc]`

(B) Returns the index of the last element in array `arr` whose value is greater than `arr[loc]`

(C) Returns the largest value in array `arr`

(D) Returns the index of the largest value in array `arr`

(E) Returns the index of the largest value in the second half of array `arr`

18. Assume the following declarations have been made.

```
private String s;
private int n;

public void changer(String x, int y)
{
  x = x + "peace";
  y = y * 2;
}
```

Assume `s` has the value `"world"` and `n` is `6`. What are the values of `s` and `n` after the call `changer(s, n)`?

	s	n
(A)	world	6
(B)	worldpeace	6
(C)	world	12
(D)	worldpeace	12
(E)	peace	12

Answers to Computer Science A Multiple-Choice Questions

1 – B	4 – D	7 – A	10 – A	13 – C	16 – D
2 – C	5 – D	8 – C	11 – C	14 – D	17 – D
3 – E	6 – A	9 – D	12 – E	15 – D	18 – A

Sample Free-Response Questions

Following is a representative set of questions. Questions marked with an asterisk are also representative of AB exam questions. The AP Computer Science A Exam will include one free-response question based on the *AP Computer Science Case Study*. (See AP Central for examples.)

Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Unless otherwise noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.

1. In an instant runoff election there are two or more candidates and there are many voters. Each voter votes by submitting a ballot that is an ordered list of all the candidates, where the first name listed is the voter's first choice, the second name is the voter's second choice, and so on. There are no ties allowed on a voter's ballot.

   The election is decided by the following process.

- Initially, all candidates are placed on the current candidate list.
- As long as there are two or more candidates on the current candidate list, the following steps are repeated.
    1. Each ballot is examined for candidates on the current candidate list and a vote is counted for the current candidate that appears earliest in the list of names on the ballot. (On the first pass, this will be the first name on the ballot. In subsequent passes, it might not be the first name on the ballot. See the illustrations below.)
    2. The candidate(s) with the fewest votes is (are) eliminated from the current candidate list.
- The last remaining candidate is the winner. If there is none, the election is not decisive.

For example, suppose there are four candidates in the election: Chris, Jamie, Pat, and Sandy. Each ballot has these four names listed in order of the voter's preference, with the first choice appearing first in the list. Assume that seven ballots were submitted as shown in the following table.

Current Candidate List: Chris, Jamie, Pat, Sandy

Voter	Ballot	First Choice from Current Candidate List
0	Chris, Jamie, Pat, Sandy	Chris
1	Chris, Pat, Sandy, Jamie	Chris
2	Chris, Sandy, Pat, Jamie	Chris
3	Pat, Jamie, Sandy, Chris	Pat
4	Pat, Sandy, Chris, Jamie	Pat
5	Sandy, Pat, Jamie, Chris	Sandy
6	Jamie, Sandy, Pat, Chris	Jamie

In the first pass, Chris has 3 votes, Pat has 2 votes, Sandy has 1 vote, and Jamie has 1 vote. Jamie and Sandy are tied for the fewest votes; so both are eliminated, leaving Chris and Pat on the current candidate list. Voter preferences for these two candidates are shown in the following table.

Current Candidate List: Chris, Pat

Voter	Ballot	First Choice from Current Candidate List
0	Chris, Jamie, Pat, Sandy	Chris
1	Chris, Pat, Sandy, Jamie	Chris
2	Chris, Sandy, Pat, Jamie	Chris
3	Pat, Jamie, Sandy, Chris	Pat
4	Pat, Sandy, Chris, Jamie	Pat
5	Sandy, Pat, Jamie, Chris	Pat
6	Jamie, Sandy, Pat, Chris	Pat

In the second pass, Chris has 3 votes and Pat has 4 votes. Chris has fewest votes and is eliminated. Pat is the only remaining candidate and is therefore the winner of the election.

A ballot is modeled with the following partial class declaration.

```
public class Ballot
{
  /** @param candidateList  a list of candidate names
   *  @return  the name of the first choice candidate for this Ballot
   *           from those in candidateList
   */
  public String firstChoiceFrom(ArrayList<String> candidateList)
  { /* implementation not shown */  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

The Ballot method firstChoiceFrom returns the name of the candidate from candidateList that appears first on this ballot.

The set of ballots for all voters in an election is modeled with the following partial class declaration.

```
public class VoterBallots
{
  private ArrayList<Ballot> ballotList;
  // each entry represents one voter's ballot

  /** @param candidate  the name of a candidate
   *  @param candidateList  a list of candidate names
   *          Precondition: candidate appears in candidateList
   *  @return  the number of times that candidate is first among
   *           those in candidateList for elements of ballotList
   */
  private int numFirstVotes(String candidate,
                            ArrayList<String> candidateList)
  { /* to be implemented in part (a) */ }

  /** @param candidateList  a list of candidate names
   *          Precondition: each String in candidateList appears exactly
   *                        once in each Ballot in ballotList
   *  @return  a list of those candidates tied with the fewest first choice votes
   */
  public ArrayList<String> candidatesWithFewest(
                           ArrayList<String> candidateList)
  { /* to be implemented in part (b) */ }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

An instant runoff election is represented by the class `InstantRunoff` that encapsulates the process of selecting a winner by repeatedly applying the `VoterBallots` method `candidatesWithFewest` to a list of candidates that is reduced until only the winner remains. This class is not shown here.

(a) Write the `VoterBallots` method `numFirstVotes.` Method `numFirstVotes` should return the number of times `candidate` appears first, among those elements that are on `candidateList,` in elements of `ballotList.`

Complete method `numFirstVotes` below.

```
/** @param candidate the name of a candidate
 *  @param candidateList a list of candidate names
 *            Precondition: candidate appears in candidateList
 *  @return the number of times that candidate is first among
 *             those in candidateList for elements of ballotList
 */
private int numFirstVotes(String candidate,
                          ArrayList<String> candidateList)
```

(b) Write the `VoterBallots` method `candidatesWithFewest.` Method `candidatesWithFewest` should count the number of times each `String` in the list `candidateList` appears first in an element of `ballotList,` and return an `ArrayList` of all those `Strings` that are tied for the smallest count.

In writing method `candidatesWithFewest` you may use the `private` helper method `numFirstVotes` specified in part (a). Assume that `numFirstVotes` works as specified, regardless of what you wrote in part (a).

Complete method `candidatesWithFewest` below.

```
/** @param candidateList a list of candidate names
 *          Precondition: each String in candidateList appears exactly
 *                        once in each Ballot in ballotList
 *  @return a list of those candidates tied with the fewest first choice votes
 */
public ArrayList<String> candidatesWithFewest(
                         ArrayList<String> candidateList)
```

2. Consider the following incomplete declaration of a `LineEditor` class that allows insertions and deletions in a line of text. The line of text is stored internally as a `String`. The `insert` operation takes a `String` and inserts it into the line of text at the given index. The `delete` operation takes a `String` parameter and removes the first occurrence (if any) of that string from the line of text. The `deleteAll` operation removes all occurrences (if any) of a given `String` from the line of text, including any that are formed as a result of the deletion process.

```
public class LineEditor
{
  private String myLine;

  /** Inserts str into myLine at position index;
   *  no characters from myLine are overwritten.
   *  @param str the string to be inserted
   *  @param index the position at which to insert str
   *           Precondition: 0 ≤ index ≤ myLine.length()
   */
  public void insert(String str, int index)
  { /* to be implemented in part (a) */ }

  /** If str is found in myLine the first occurrence of str has been
   *  removed from myLine; otherwise myLine is left unchanged.
   *  @param str the string to be removed
   */
  public void delete(String str)
  { /* to be implemented in part (b) */ }

  /** Removes all occurrences of str from myLine;
   *  myLine is otherwise unchanged.
   *  @param str the string to be removed
   */
  public void deleteAll(String str)
  { /* to be implemented in part (c) */ }

  // There may be instance variables, constructors, and methods that are not
  // shown.
}
```

(a) Write the `LineEditor` method `insert` as described at the beginning of the question. The following tables show the result of several different calls to `insert`.

Method call: `insert("A.P.", 0)`

`myLine` before the call	`myLine` after the call
`"Computer Science"`	`"A.P.Computer Science"`

Method call: `insert(" is best", 16)`

`myLine` before the call	`myLine` after the call
`"Computer Science"`	`"Computer Science is best"`

Method call: `insert("Java", 4)`

`myLine` before the call	`myLine` after the call
`"Computer Science"`	`"CompJavauter Science"`

Complete method `insert` below. Assume that the other class methods work as specified.

```
/** Inserts str into myLine at position index;
 *  no characters from myLine are overwritten.
 *  @param str the string to be inserted
 *  @param index the position at which to insert str
 *          Precondition: 0 ≤ index ≤ myLine.length()
 */
public void insert(String str, int index)
```

(b) Write the `LineEditor` method `delete` as described at the beginning of the question. The following table shows the result of several different calls to `delete`.

Method call: `delete("Com")`

`myLine` before the call	`myLine` after the call
`"Computer Science"`	`"puter Science"`

Method call: `delete("ter Sc")`

`myLine` before the call	`myLine` after the call
`"Computer Science"`	`"Compuience"`

Method call: `delete("c")`

`myLine` before the call	`myLine` after the call
`"Computer Science"`	`"Computer Sience"`

Method call: `delete("Java")`

`myLine` before the call	`myLine` after the call
`"Computer Science"`	`"Computer Science"`

Complete method `delete` below.

```
/** If str is found in myLine the first occurrence of str has been
 *  removed from myLine; otherwise myLine is left unchanged.
 *  @param str the string to be removed
 */
public void delete(String str)
```

(c) Write the `LineEditor` method `deleteAll` as described at the beginning of the question. The following table shows the result of several different calls to `deleteAll`.

Method call: `deleteAll("ing")`

`myLine` before the call	`myLine` after the call
`"string programming"`	`"str programm"`

Method call: `deleteAll("r")`

`myLine` before the call	`myLine` after the call
`"string programming"`	`"sting pogamming"`

Method call: `deleteAll("aba")`

`myLine` before the call	`myLine` after the call
`"abababa"`	`"b"`

Method call: `deleteAll("oh-la")`

`myLine` before the call	`myLine` after the call
`"ooh-lah-lah"`	`"h"`

Method call: `deleteAll("zap")`

`myLine` before the call	`myLine` after the call
`"pizza pie"`	`"pizza pie"`

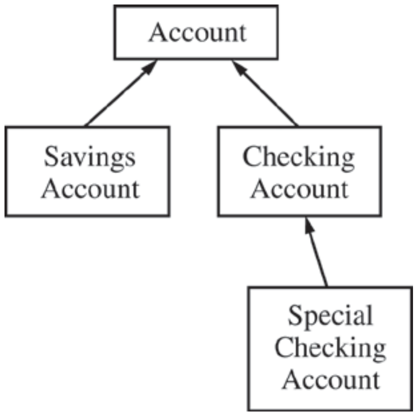
In writing `deleteAll,` you may call any of the methods in the `LineEditor` class, including `insert` and `delete` from parts (a) and (b). Assume that these methods work as specified, regardless of what you wrote in parts (a) and (b).

Complete method `deleteAll` below.

```
/** Removes all occurrences of str from myLine;
 *  myLine is otherwise unchanged.
 *  @param str the string to be removed
 */
public void deleteAll(String str)
```

Note: The following question is somewhat longer than what may appear on the AP Computer Science Exams. In particular, a question of this type appearing on the AP Computer Science A Exam might be limited to two parts.

*3. Consider the problem of modeling bank accounts. A diagram of the class hierarchy used to represent bank accounts is shown below.



The class `Account` models a bank account with the following data and operations.

Data

- the identity number for the account (The identity number is never changed once the account has been constructed.)
- the balance in the account (The balance can change as a result of some operations.)

Operations

- create an account with a given identity number and initial balance
- return the identity number
- return the current balance
- deposit some positive amount into the account, increasing the balance
- decrease the balance by a specified positive amount; if the amount is greater than the balance, throw an `IllegalArgumentException`
- return the monthly interest due

An implementation for this class is shown below.

```
public class Account
{
  private int idNum;        // identity number for this account
  private double balance;  // current balance for this account

  /** Creates an Account with identity number idNumber
   *  and current balance startBal.
   *  @param idNumber the identity number for the account
   *  @param startBal the starting balance for the account
   *            Precondition: startBal ≥ 0.0
   */
  public Account(int idNumber, double startBal)
  { /* implementation not shown */ }

  /** @return the identity number for this account.
   */
  public int idNumber()
  { /* implementation not shown */ }

  /** @return the current balance for this account.
   */
  public double currentBalance()
  { /* implementation not shown */ }

  /** Increases the current balance of this account by amount.
   *  @param amount the amount to be deposited into the account
   *            Precondition: amount ≥ 0.0
   */
  public void deposit (double amount)
  { /* implementation not shown */ }

  /** Decreases the current balance of this account by amount.
   *  @param amount the amount to be removed from the account
   *            Precondition: 0 ≤ amount ≤ currentBalance()
   */
  public void decreaseBalance (double amount)
  { /* implementation not shown */ }

  /** @return the monthly interest due for this account.
   */
  public double monthlyInterest()
  {  return 0.0;  }
}
```

(a) A savings account at a bank “is-a” bank account and is modeled by the class `SavingsAccount`. A savings account has all the characteristics of a bank account. In addition, a savings account has an interest rate, and the interest due each month is calculated from that interest rate. The operations for a savings account that differ from those specified in the class `Account` are the following.

- create a new savings account with a given annual interest rate, as well as the parameters required for all accounts
- withdraw a positive amount that does not exceed the current balance, decreasing the balance by the amount withdrawn
- calculate the monthly interest by multiplying the current balance by the annual interest rate divided by twelve

Write the complete definition of the class `SavingsAccount`, including the implementation of methods.

(b) A checking account at a bank "is-a" bank account and is modeled by the class `CheckingAccount`. A checking account has all the characteristics of a bank account. In addition, a checking account can have checks written on it. Each check written decreases the account by the amount of the check plus a per-check charge. The operations for a checking account that differ from those specified in the class `Account` are the following.

- create a new checking account with a given per-check charge, as well as the parameters required for all accounts
- clear a check for a given amount by decreasing the balance by the amount of the check plus the per-check charge
- compute and return the monthly interest

A declaration of the class `CheckingAccount` is shown below.

```
public class CheckingAccount extends Account
{
  private double checkCharge;

  public CheckingAccount(int idNumber, double startBal,
                         double chkCharge)
  {
    super(idNumber, startBal);
    checkCharge = chkCharge;
  }

  public void clearCheck(double amount)
  {
    decreaseBalance(amount + checkCharge);
  }

  public double monthlyInterest()
  { /* implementation not shown */ }
}
```

A special checking account “is-a” checking account and is modeled by the class `SpecialCheckingAccount`. A special checking account has all the characteristics of a checking account. In addition, a special checking account has a minimum balance and an annual interest rate. When the balance is above the minimum balance, the per-check charge is not deducted from the balance when a check is cleared. Otherwise, a check is cleared just as it is for a checking account. In addition, when the balance is above the minimum balance when interest is calculated, interest due is calculated on the current balance. Otherwise, the interest due is the same as for a checking account. The operations for a special checking account that differ from those specified in the class `CheckingAccount` are the following.

- create a new special checking account with a given minimum balance and interest rate, as well as the parameters required for a checking account
- clear a check for a given amount according to the rules above
- calculate the monthly interest by multiplying current balance by the annual interest rate divided by twelve if the current balance is above the minimum; otherwise, calculate the interest as it is done for a checking account

Write the complete definition of the class `SpecialCheckingAccount`, including the implementation of its methods.

(c) Consider the class `Bank` partially specified below.

```
public class Bank
{
  private ArrayList<Account> accounts;
    // all accounts in this bank
    // accounts has no null entries

  /** For each account in this bank, deposits the monthly interest due into that
   *  account.
   */
  public void postMonthlyInterest()
  {
    // to be implemented in this part
  }

  // There may be instance variables, constructors, and methods that are not
  // shown.
}
```

Write the `Bank` method `postMonthlyInterest`, which is described as follows. For each account in this bank, `postMonthlyInterest` should calculate the monthly interest and deposit that amount into the account.

In writing `postMonthlyInterest`, you may use any of the public methods of class `Account` or its subclasses. Assume these methods work as specified. Solutions that reimplement functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method `postMonthlyInterest` below.

```
/**  For each account in this bank, deposits the monthly interest due into that
 *   account.
 */
public void postMonthlyInterest()
```

Suggested Solutions to Free-Response Questions

Note: There are many correct variations of these solutions.

Question 1

(a)

```
private int numFirstVotes(String candidate,
                      ArrayList<String> candidateList)
{
    int numVotes = 0;

    for (Ballot voterBallot : ballotList)
    {
      String first = voterBallot.firstChoiceFrom(candidateList);

      if (candidate.equals(first))
       numVotes++;
    }
    return numVotes;
}
```

(b)

```
public ArrayList<String> candidatesWithFewest(
                         ArrayList<String> candidateList)
{
    int[] votes = new int[candidateList.size()];
    int minVotes = ballotList.size();

    for (int c = 0; c < candidateList.size(); c++)
    {
      String candidate = candidateList.get(c);
      votes[c] = numFirstVotes(candidate, candidateList);

      if (votes[c] < minVotes)
        minVotes = votes[c];
    }

    ArrayList<String> result = new ArrayList<String>();
    for (int c = 0; c < candidateList.size(); c++)
    {
      if (votes[c] == minVotes)
        result.add(candidateList.get(c));
    }

    return result;
}
```

(b) Alternate solution

```
public ArrayList<String> candidatesWithFewest(
                  ArrayList<String> candidateList)
{
    ArrayList<String> result = new ArrayList<String>();
    int minVotes = ballotList.size() + 1;

    for (int c = 0; c < candidateList.size();c++)
    {
      String candidate = candidateList.get(c);
      int thisVotes = numFirstVotes(candidate, candidateList);

      if (thisVotes < minVotes)
      {
        minVotes = thisVotes;
        result = new ArrayList<String>();
      }
      if (thisVotes == minVotes)
        result.add(candidateList.get(c));
    }

    return result;
}
```

Question 2

(a)

```
public void insert(String str, int index)
{
    myLine = myLine.substring(0, index) + str
               + myLine.substring(index);
}
```

(b)

```
public void delete(String str)
{
    int index = myLine.indexOf(str);

    if (index != -1)
    {
      myLine = myLine.substring(0, index)
                + myLine.substring(index + str.length());
    }
}
```

(c)

```
public void deleteAll(String str)
{
    while (myLine.indexOf(str) != -1)
    {
      delete(str);
    }
}
```

Question 3

(a)

```
public class SavingsAccount extends Account
{
    private double intRate; // annual interest rate for
                            // this account

    public SavingsAccount(int idNumber, double balance,
                          double rate)
    {
      super(idNumber, balance);
      intRate = rate;
    }

    public double monthlyInterest()
    {
      return (currentBalance() * (intRate / 12.0));
    }

    public void withdraw(double amount)
    {
      decreaseBalance(amount);
    }
}
```

(b)

```
public class SpecialCheckingAccount extends CheckingAccount
{
    private double minBalance;
    private double intRate;

    public SpecialCheckingAccount(int idNumber,
                                  double startBal,
                                  double chkCharge,
                                  double minBal, double rate)
    {
      super(idNumber, startBal, chkCharge);
      minBalance = minBal;
      intRate = rate;
    }

    public void clearCheck(double amount)
    {
      if (currentBalance() > = minBalance)
        decreaseBalance(amount);
      else
        super.clearCheck(amount);
    }

    public double monthlyInterest()
    {
      if (currentBalance() > = minBalance)
        return  (currentBalance() * (intRate / 12.0));
      else
        return super.monthlyInterest();
    }
}
```

(c)

```
public void postMonthlyInterest()
{
    double interest;

    for (Account acct : accounts)
    {
      interest = acct.monthlyInterest();
      acct.deposit(interest);
    }
}
```

Computer Science AB: Sample Multiple-Choice Questions

Following is a representative set of questions. Questions marked with an asterisk in the Computer Science A questions are also representative of Computer Science AB questions. The answer key for Computer Science AB multiple-choice questions is on page 74. In this section of the exam, as a correction for haphazard guessing, one-fourth of the number of questions answered incorrectly will be subtracted from the number of questions answered correctly. The AP Computer Science AB Exam will include several multiple-choice questions based on the *AP Computer Science Case Study.* (See AP Central for examples.)

Directions: Determine the answer to each of the following questions or incomplete statements, using the available space for any necessary scratchwork. Then decide which is the best of the choices given and fill in the corresponding oval on the answer sheet. No credit will be given for anything written in the examination booklet. Do not spend too much time on any one problem.

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- Assume that the implementation classes `ListNode` and `TreeNode` (page A4 in the Appendix) are used for any questions referring to linked lists or trees, unless otherwise specified.
- `ListNode` and `TreeNode` parameters may be `null`. Otherwise, unless noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- Assume that declarations of variables and methods appear within the context of an enclosing class.
- Assume that method calls that are not prefixed with an object or class name and are not shown within a complete class definition appear within the context of an enclosing class.

1. Consider the following code segment.

```
int [][] mat = new int [3][4];

for (int row = 0; row < mat.length; row++)
{
  for (int col = 0; col < mat[0].length; col++)
  {
    if (row < col)
      mat[row][col] = 1;
    else if (row == col)
      mat[row][col] = 2;
    else
      mat[row][col] = 3;
  }
}
```

What are the contents of `mat` after the code segment has been executed?

(A)
```
{{2  1  1}
 {3  2  1}
 {3  3  2}
 {3  3  3}}
```

(B)
```
{{2  3  3}
 {1  2  3}
 {1  1  2}
 {1  1  1}}
```

(C)
```
{{2  3  3  3}
 {1  2  3  3}
 {1  1  2  3}}
```

(D)
```
{{2  1  1  1}
 {3  2  1  1}
 {3  3  2  1}}
```

(E)
```
{{1  1  1  1}
 {2  2  2  2}
 {3  3  3  3}}
```

2. Which of the following best describes the data structure represented by `java.util.LinkedList` ?

   (A) A singly linked list with a reference to the first node only

   (B) A singly linked list with references to the first and last nodes

   (C) A doubly linked list with a reference to the first node only

   (D) A doubly linked list with references to the first and last nodes only

   (E) A doubly linked list with reference to the first, middle, and last nodes

3. Consider the following code segment.

```
Queue<String> que = new LinkedList<String>();
  // LinkedList implements Queue

Object obj;

que.add("a");
que.add("b");
que.add("c");
obj = que.peek();
que.add(obj + "d");
que.add(obj + "x" + obj);

while (! que.isEmpty())
{
  System.out.print(que.remove() + " ");
}
```

   What is printed as a result of executing this code segment?

   (A) `a b c ad`

   (B) `b c ad axa`

   (C) `axa ad c b a`

   (D) `ad axa b c a`

   (E) `a b c ad axa`

4. Consider the following declarations.

```
Stack<String> s = new Stack<String>();

Queue<String> q = new LinkedList<String>();
                     // LinkedList implements Queue
```

Assume that `s` is initially empty and that `q` initially contains the following strings.

```
   W   X   Y   Z
   ↑           ↑
 front       back
```

Consider the following code segment.

```
while (!q.isEmpty())
  s.push(q.remove());

while (!s.isEmpty())
  q.add(s.pop());
```

Which of the following best describes stack `s` and queue `q` after the code segment has been executed?

(A) Stack `s` is empty and queue `q` contains `W`, `X`, `Y`, `Z`, in that order, with `W` at the front of the queue.

(B) Stack `s` is empty and queue `q` contains `Z`, `Y`, `X`, `W`, in that order, with `Z` at the front of the queue.

(C) Stack `s` contains `Z`, `Y`, `X`, `W`, in that order, with `Z` at the top of the stack, and queue `q` is empty.

(D) Stack `s` contains `Z`, `Y`, `X`, `W`, in that order, with `Z` at the top of the stack; and queue `q` contains `W`, `X`, `Y`, `Z`, in that order, with `W` at the front of the queue.

(E) Stack `s` contains `Z`, `Y`, `X`, `W`, in that order, with `Z` at the top of the stack; and queue `q` contains `Z`, `Y`, `X`, `W`, in that order, with `Z` at the front of the queue.

5. Consider the following code segment.

```
for (int j = 1; j <= n; j++)
{
  for (int k = 1; k <= n; k = k * 2)
  {
    System.out.println(j + " " + k);
  }
}
```

Of the following, which best characterizes the running time of the code segment?

(A) O (log n)

(B) O (n)

(C) O (n log n)

(D) O (n^2)

(E) O (n!)

6. The following integers are inserted into an empty binary search tree in the following order.

```
26  20  37  31  22  18  25  29  19
```

Which traversal of the tree would produce the following output?

```
26  20  37  18  22  31  19  25  29
```

(A) Preorder

(B) Inorder

(C) Postorder

(D) Reverse postorder

(E) Level-by-level

7. Consider the following methods.

```
public List<Integer> process1(int n)
{
  ArrayList<Integer> someList = new ArrayList<Integer>();

  for (int k = 0; k < n; k++)
    someList.add(new Integer(k));

  return someList;
}

public List<Integer> process2(int n)
{
  ArrayList<Integer> someList = new ArrayList<Integer>();

  for (int k = 0; k < n; k++)
    someList.add(k, new Integer(k));

  return someList;
}
```

Which of the following best describes the behavior of `process1` and `process2` ?

(A) Both methods produce the same result and take the same amount of time.

(B) Both methods produce the same result, and `process1` is faster than `process2`.

(C) The two methods produce different results and take the same amount of time.

(D) The two methods produce different results, and `process1` is faster than `process2`.

(E) The two methods produce different results, and `process2` is faster than `process1`.

8. Consider the following instance variable and incomplete method. Method `hasItem` should return `true` if `item` is found in `myList;` otherwise, it should return false.

```
private List <Object> myList;

public boolean hasItem(Object item)
{
  Iterator <Object> itr = myList.iterator();

  while ( /* condition */ )
  {
    if (itr.next().equals(item))
        return true;
  }

  return false;
}
```

Which of the following expressions can be used to replace `/* condition */` so that `hasItem` will work as intended?

(A) `itr.hasNext()`

(B) `myList.hasNext()`

(C) `! itr.hasNext()`

(D) `! myList.hasNext()`

(E) `itr != null`

9. Consider the following method.

```
public static void mystery(ListNode p)
{
  if (p != null)
  {
    mystery(p.getNext().getNext());
    p.setNext(p.getNext().getNext());
  }
}
```

What changes does mystery make to the list whose first node is `p` ?

(A) It makes no changes to the list.

(B) It removes the first, third, and all odd nodes from the list.

(C) It removes the second, fourth, and all even nodes from the list.

(D) It removes all nodes except the first node of the list.

(E) If the number of nodes in the list is odd, it will cause a `NullPointerException;` otherwise, it removes half of the nodes from the list.

10. Consider the following partial class declaration.

```
public class LList
{
  private ListNode front;

  public LList()
  {
    front = null;
  }

  public void addToLList(Comparable obj)
  {
     front = addHelper(front, obj);
  }

  private ListNode addHelper(
          ListNode list, Comparable obj)
  {
     if (list == null ||
         obj.compareTo(list.getValue()) < 0)
     {
       list = new ListNode(obj, list);
       return list;
     }
     else
     {
       list.setNext(addHelper(list.getNext(), obj));
       return list;
     }
  }

  // There may be instance variables, constructors, and methods that are
  // not shown.
}
```

Consider the following code segment that appears in a client program.

```
LList list = new LList();

list.addToLList("manager");
list.addToLList("boy");
list.addToLList("girl");
list.addToLList("anyone");
list.addToLList("place");
list.addToLList("vector");
```

What values are in `list` after the code segment has been executed?

(A) `[anyone, boy, girl, manager, place, vector]`

(B) `[manager, boy, girl, anyone, place, vector]`

(C) `[vector, place, anyone, girl, boy, manager]`

(D) `[vector, place, manager, girl, boy, anyone]`

(E) Nothing is in `list` because a `NullPointerException` was thrown during the execution.

11. Consider the following instance variable and incomplete method, `partialSum`, which is intended to return an integer array `sum` such that for all `i`, `sum[i]` is equal to `arr[0] + arr[1] + ... + arr[i]`. For instance, if `arr` contains the values `{ 1, 4, 1, 3 }`, the array `sum` will contain the values `{ 1, 5, 6, 9 }`.

```
private int[] arr;

public int[] partialSum()
{
  int[] sum = new int[arr.length];

  for (int j = 0; j < sum.length; j++)
    sum[j] = 0;

  /* missing code */

  return sum;
}
```

The following two implementations of `/* missing code */` are proposed so that `partialSum` will work as intended.

<u>Implementation 1</u>

```
for (int j = 0; j < arr.length; j++)
  sum[j] = sum[j - 1] + arr[j];
```

<u>Implementation 2</u>

```
for (int j = 0; j < arr.length; j++)
  for (int k = 0; k <= j; k++)
      sum[j] = sum[j] + arr[k];
```

Which of the following statements is true?

(A) Both implementations work as intended, but implementation 1 is faster than implementation 2.

(B) Both implementations work as intended, but implementation 2 is faster than implementation 1.

(C) Both implementations work as intended and are equally fast.

(D) Implementation 1 does not work as intended, because it will cause an `ArrayIndexOutOfBoundsException.`

(E) Implementation 2 does not work as intended, because it will cause an `ArrayIndexOutOfBoundsException.`

Questions 12–13 refer to the following method.

```
private static void sort(List<Comparable> theList)
{
  Iterator<Comparable> itr = theList.iterator();

  PriorityQueue<Comparable> pq = new PriorityQueue<Comparable>();

  while (itr.hasNext())
  {
    pq.add(itr.next())
    itr.remove(); // removes last item just seen by itr
  }

  while ( !pq.isEmpty())
    theList.add(pq.remove());
}
```

12. If the parameter is a `LinkedList` in <u>sorted</u> order, which data structure operation dominates the running time of the sort?

    (A) `pq.add`

    (B) `itr.remove`

    (C) `pq.isEmpty`

    (D) `pq.remove`

    (E) `theList.add`

13. If the parameter is an `ArrayList`, which data structure operation dominates the running time of the sort?

    (A) `pq.add`

    (B) `itr.remove`

    (C) `pq.isEmpty`

    (D) `pq.remove`

    (E) `theList.add`

14. An electronic mail application includes a class for handling mail aliases. A mail alias is a single name that represents a collection of mail addresses. For example, someone might set up the following mail aliases.

Alias	**Collection of mail addresses**
`family`	`myMom@business.com, myDad@isp.net, mySis@school.edu`
`buddies`	`pat@school1.edu, chris@school2.edu, taylor@school3.edu`
`teammates`	`jamie@myschool.edu, alex@myschool.edu`

Which of the following describes the best choice of data structures for representing an individual's mail aliases in the electronic mail application?

(A) Represent the mail aliases as a two-dimensional array of strings with the alias name in one column and the collection of mail addresses for that alias as a single string in a second column.

(B) Represent the mail aliases as an array of linked lists of strings, where the first element in each linked list is the alias name and the remaining elements in the linked list are the mail addresses for that alias.

(C) Represent the mail aliases as an `ArrayList<Map<String, Set<String>>>`, where each `Map` has one key, the alias name, and one value, a set of strings representing the mail addresses for that alias.

(D) Represent the mail aliases as a `Map<String, Set<String>>`, using the alias names as the keys and sets of mail addresses as the values.

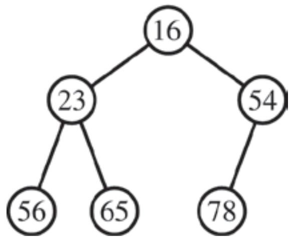
(E) Represent the mail aliases as a stack of linked lists, where the first element in each linked list is the alias and the remaining elements in the linked list are the mail addresses for that alias.

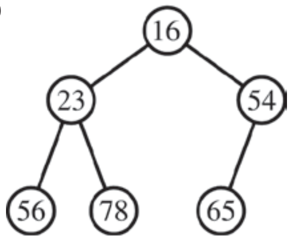
15. A min-heap with no duplicate values is a binary tree in which the value in each node is smaller than the values in its children's nodes.

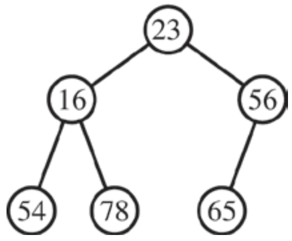
    Suppose the following values are inserted sequentially into an empty heap in the following order.

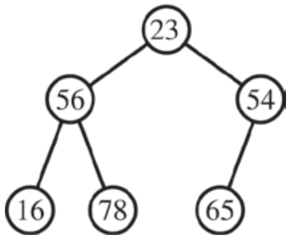
    ```
    23, 56, 54, 16, 78, 65
    ```

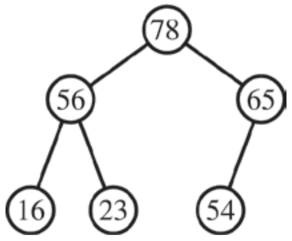
    What will the contents of the min-heap be after the values are inserted?

(A)


(B)


(C)


(D)


(E)


16. Consider the following declaration for a class that will be used to represent points in the *xy*-coordinate plane.

```
public class Point
{
  private int myX;      // coordinates
  private int myY;

  public Point()
  {
    myX = 0;
    myY = 0;
  }

  public Point(int a, int b)
  {
    myX = a;
    myY = b;
  }

  // ... other methods not shown
}
```

The following incomplete class declaration is intended to extend the above class so that points can be named.

```
public class NamedPoint extends Point
{
  private String myName;

  // constructors go here

  // ... other methods not shown
}
```

Consider the following proposed constructors for this class.

I.
```
public NamedPoint()
{
  myName = "";
}
```

II.
```
public NamedPoint(int d1, int d2, String name)
{
  myX = d1;
  myY = d2;
  myName = name;
}
```

III.
```
public NamedPoint(int d1, int d2, String name)
{
  super(d1, d2);
  myName = name;
}
```

Which of these constructors would be legal for the `NamedPoint` class?

(A) I only

(B) II only

(C) III only

(D) I and III

(E) II and III

Answers to Computer Science AB Multiple-Choice Questions

1 – D	4 – B	7 – A	10 – A	13 – B	16 – D
2 – D	5 – C	8 – A	11 – D	14 – D	
3 – E	6 – E	9 – E	12 – D	15 – B	

Sample Free-Response Questions

Following is a representative set of questions. The AP Computer Science AB Exam will include one free-response question based on the *AP Computer Science Case Study.* (See AP Central for examples.)

Directions: SHOW ALL YOUR WORK. REMEMBER THAT PROGRAM SEGMENTS ARE TO BE WRITTEN IN JAVA.

Notes:

- Assume that the classes listed in the Quick Reference found in the Appendix have been imported where appropriate.
- The `java.util.Stack` and `java.util.PriorityQueue` classes and the `java.util.Queue` interface (see page A2 in the Appendix) each inherit methods that access elements in a way that violates their abstract data structure definitions. Solutions that use objects of types `Stack`, `Queue`, and `PriorityQueue` should use only the methods listed in the Appendix for accessing and modifying those objects. The use of other methods may not receive full credit.
- Assume that the implementation classes `ListNode` and `TreeNode` (see page A4 in the Appendix) are used for any questions referring to linked lists or trees, unless otherwise specified.
- `ListNode` and `TreeNode` parameters may be `null`. Otherwise, unless noted in the question, assume that parameters in method calls are not `null` and that methods are called only when their preconditions are satisfied.
- In writing solutions for each question, you may use any of the accessible methods that are listed in classes defined in that question. Writing significant amounts of code that can be replaced by a call to one of these methods may not receive full credit.
- When Big-Oh running time is required for a response, you must use the most restrictive Big-Oh expression. For example, if the running time is $O(n)$, a response of $O(n^2)$ will not be given credit.

1. Consider using a *radix sort* to order a list of nonnegative integers. A radix sort makes as many passes through the list as there are digits in the largest number to be sorted. For example, if the largest integer in the list was 492, then the algorithm would make three passes through the list to sort it.

   Assume that the list of numbers to be sorted is in an integer array called `nums`. In each pass through the list, the radix sort algorithm sorts the numbers based on a different digit, working from the least to the most significant digit. To do this, it uses an intermediate data structure, `queues`, an `ArrayList` of ten queues. Each number is placed into the queue corresponding to the value of the digit being examined. For example, in the first pass the digit in the ones place is considered, so the number 345 would be enqueued into `queues.get(5)`. The number 260 would be enqueued into `queues.get(0)`. In each pass, the algorithm moves the numbers to be sorted from `nums` to the array of queues and then back to `nums`, as described below. After the last pass, the integers in `nums` are in order from smallest to largest.

   <u>Radix Sort Algorithm</u>:

   In each pass through the list, do the following two steps.

   <u>Step 1</u>

   Taking each integer from `nums` in order, insert the integer into the queue corresponding to the value of the digit currently being examined. If the integer being examined does not have a digit at a given place value, 0 is assumed for that place value. For example, 95 has no digit in the hundreds place, so, when examining the hundreds digit, the algorithm would assume the value in the hundreds place is zero and enqueue 95 into `queues.get(0)`.

   <u>Step 2</u>

   After all integers have been inserted into the appropriate queues, the array `nums` is filled from beginning to end by emptying the queues into `nums`, starting with the integers in `queues.get(0)` and proceeding sequentially through `queues.get(9)`.

   For example, assume that `nums` contains the integers 380, 95, 345, 382, 260, 100, and 492. The sort will take three passes, because the largest integer in `nums` has 3 digits. The following diagram shows the sorting process. (For passes II and III, only the nonempty queues are shown in order to save space.)

```
   nums      queues                                nums
  Before     After     Front          Rear         After
   Pass      Step 1      ↑              ↑          Step 2
                                          Pass I

[0]    380    [0]       380    260    100   [0]    380
[1]     95    [1]                           [1]    260
[2]    345    [2]       382    492          [2]    100
[3]    382    [3]                           [3]    382
[4]    260    [4]                           [4]    492
[5]    100    [5]        95    345          [5]     95
[6]    492    [6]                           [6]    345
              [7]
              [8]
              [9]
                                        Pass II

[0]    380    [0]       100                [0]    100
[1]    260    [4]       345                [1]    345
[2]    100    [6]       260                [2]    260
[3]    382    [8]       380    382         [3]    380
[4]    492    [9]       492     95         [4]    382
[5]     95                                 [5]    492
[6]    345                                 [6]     95
                                        Pass III

[0]    100    [0]        95                [0]     95
[1]    345    [1]       100                [1]    100
[2]    260    [2]       260                [2]    260
[3]    380    [3]       345    380    382  [3]    345
[4]    382    [4]       492                [4]    380
[5]    492                                 [5]    382
[6]     95                                 [6]    492
```

The radix sort is implemented using the following class declaration.

```
public class RadixSort
{
  /** @return the k-th digit of number, where k = 0 is the least significant digit.
   *  @param number an integer
   *         Precondition: number ≥ 0
   *  @param k the position in the number
   *         Precondition: k ≥ 0
   */
  private static int getDigit(int number, int k)
  { /* implementation not shown */ }

  /** @return an ArrayList of 10 queues as described in the example.
   *  Postcondition: the total number of values in the ArrayList
   *                 of queues is equal to nums.length
   *  @param nums an array of nonnegative integers
   *         Precondition: nums.length > 0
   *  @param k the position of the digit used to determine queue placement
   *         Precondition: k ≥ 0
   */
  private static ArrayList<Queue<Integer>> itemsToQueues(int[]
                                           nums, int k)
  { /* to be implemented in part (a) */ }

  /** @return an array that contains the integers from queues.get(0) through
   *  queues.get(9) in the order in which they were stored in the queues.
   *  Postcondition: each queue in queues is empty
   *  @param queues an ArrayList of 10 queues
   *         Precondition: queues.size() is 10
   *  @param numVals the total number of values in all 10 queues
   */
  private static int[] queuesToArray(
                          ArrayList<Queue<Integer>> queues,
                          int numVals)
  { /* to be implemented in part (b) */ }

  /** @return an array of all the values found in nums, sorted in nondecreasing order.
   *  @param nums an array of nonnegative integers
   *         Precondition: nums.length > 0
   *  @param numDigits the number of digits in the largest value of nums
   *         Precondition: the largest value in nums has numDigits digits
   */
  public static int[] sort(int[] nums, int numDigits)
  { /* to be implemented in part (c) */ }
}
```

Recall that the `LinkedList<Integer>` class implements the `Queue<Integer>` interface.

(a) Write the `RadixSort` method `itemsToQueues`, which is described as follows. Method `itemsToQueues` corresponds to step 1 of each pass of the radix sort algorithm, creating the intermediate `ArrayList` of ten queues. Each integer in `nums` is inserted into the queue corresponding to the value of the digit currently being examined. If an integer does not have a digit at the given place value, 0 is assumed for that place value. The digit being examined is found in the `k`th position of the number. Integers are processed in the order in which they occur in `nums`.

In writing `itemsToQueues`, you may call the `RadixSort` method `getDigit`, which returns the `k`th digit of its parameter, `number`. The least significant digit is indicated by a value of 0 for `k`. If `k` is greater than the number of digits in `number`, then `getDigit` returns 0.

The following table illustrates the results of several calls to `getDigit`.

`number`	`k`	`getDigit(number, k)`
95	0	5
95	1	9
95	2	0

You do not need to implement `getDigit`.

Complete method `itemsToQueues` below.

```
/** @return an ArrayList of 10 queues as described in the example.
 *   Postcondition: the total number of values in the ArrayList
 *                  of queues is equal to nums.length
 *   @param nums an array of nonnegative integers
 *               Precondition: nums.length > 0
 *   @param k the position of the digit used to determine queue placement
 *               Precondition: k ≥ 0
 */
private static ArrayList<Queue<Integer>> itemsToQueues
                                          (int[] nums, int k)
```

(b) Write the `RadixSort` method `queuesToArray`, which is described as follows. Method `queuesToArray` corresponds to step 2 of each pass of the radix sort algorithm, creating a new list from the values in the `ArrayList` of queues.

Complete method `queuesToArray` below.

```
/** @return an array that contains the integers from queues.get(0) through
 *   queues.get(9) in the order in which they were stored in the queues.
 *   Postcondition: each queue in queues is empty
 *   @param queues an ArrayList of 10 queues
 *            Precondition: queues.size() is 10
 *   @param numVals the total number of values in all 10 queues
 */
private static int[] queuesToArray(
                     ArrayList<Queue<Integer>> queues,
                     int numVals)
```

(c) Write the `RadixSort` method `sort,` as started below. In writing `sort,` you may call methods `getDigit,` `itemsToQueues,` and `queuesToArray.` Assume that `itemsToQueues` and `queuesToArray` work as specified, regardless of what you wrote in parts (a) and (b)

Complete method `sort` below.

```
/** @return an array of all the values found in nums, sorted in nondecreasing
 *   order.
 *   @param nums an array of nonnegative integers
 *        Precondition: nums.length > 0
 *   @param numDigits the number of digits in the largest value of nums
 *        Precondition: the largest value in nums has numDigits digits
 */
public static int[] sort(int[] nums, int numDigits)
```

2. Consider the following interface `CityInfo` that will be used to represent cities in the United States. Each city is represented by its name and the name of the state in which it is located.

```
public interface CityInfo
{
  String city();
  String state();
}
```

The following class, `States,` will be used to store states and their respective cities. The collection of `CityInfo` objects will be stored in this class as a `TreeMap.` In the `TreeMap,` the keys are the state names, and for each key the corresponding value is a `Set` of the cities in that state.

```
public class States
{
   private Map<String, Set<String>> theMap;

   public States()
   { theMap = new TreeMap<String, Set<String>>(); }

   /** Adds theCity to theMap.
    *  @param theCity contains information about the city to be added
    */
   public void addCityToMap(CityInfo theCity)
   { /* to be implemented in part (a) */ }

   public void printOneState(String theState)
   { /* to be implemented in part (b) */ }

   public void printAllStates()
   { /* to be implemented in part (c) */ }

   // There may be instance variables, constructors, and methods that are
   // not shown.
}
```

For example, assume that a `States` object, `stateMap`, has been initialized with the following `CityInfo` objects.

```
[Albany,NY] [Miami,FL] [Hamilton,NY] [Jacksonville,FL] [Dallas,TX]
```

The following table represents the entries in stateMap.

Key	**Value**
FL	[Miami, Jacksonville]
NY	[Albany, Hamilton]
TX	[Dallas]

(a) Write the States method `addCityToMap`, which is described as follows. Method `addCityToMap` takes one parameter: a new `CityInfo` object, and updates `theMap` to include the `CityInfo` object. Method `addCityToMap` should run in $O(\log n)$ expected time where n is the number of states in `theMap`.

The following tables show the result of two sequential calls to `addCityToMap`, when applied to the object `stateMap` shown at the beginning of the question. Assume that `city1` has been defined as the `CityInfo` object `[Albany,GA]` and `city2` has been defined as the `CityInfo` object `[Houston,TX]`.

<u>Result of the call</u>
`stateMap.addCityToMap(city1);`

Key	**Value**
FL	[Miami, Jacksonville]
GA	[Albany]
NY	[Albany, Hamilton]
TX	[Dallas]

<u>Result of the call</u>
`stateMap.addCityToMap(city2);`

Key	**Value**
FL	[Miami, Jacksonville]
GA	[Albany]
NY	[Albany, Hamilton]
TX	[Dallas, Houston]

Complete method `addCityToMap` below.

```
/** Adds theCity to theMap.
 *  @param theCity contains information about the city to be added
 */
public void addCityToMap(CityInfo theCity)
```

(b) Write method `printOneState,` which is described as follows. Method `printOneState` takes a `String` representing a state that is in `theMap`. It prints the name of the state and a list of cities in the state. The output should not include `[]`, and the cities should each be separated by a blank space. Method `printOneState` should run in $O(c + \log n)$ time, where n is the number of states in `theMap` and c is the number of cities associated with the given state.

For example, if `stateMap` contains the entries shown at the beginning of the question, the call `stateMap.printOneState("FL");` will result in the following output.

```
FL Miami Jacksonville
```

Complete method `printOneState` below. A solution that creates an unnecessary instance of any Collection class will not receive full credit.

```
public void printOneState(String theState)
```

(c) Write method `printAllStates,` which is described as follows. Method `printAllStates` outputs the cities in each state in the format shown in part (b). The states should be listed in alphabetical order. Method `printAllStates` should run in $O(C + n \log n)$ time, where n is the number of states in `theMap` and C is the total number of cities stored in `theMap.`

For example, if the `States` object `stateMap` has the following entries,

Key	**Value**
FL	[Miami, Jacksonville]
GA	[Albany]
NY	[Albany, Hamilton]
TX	[Dallas, Houston]

then the call `stateMap.printAllStates()` will produce the following output.

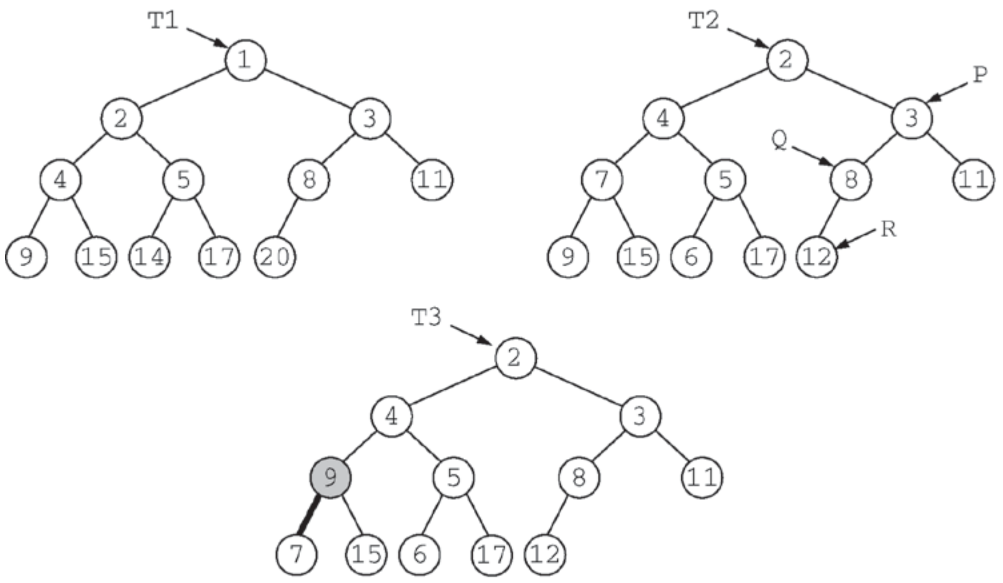
```
FL Miami Jacksonville
GA Albany
NY Albany Hamilton
TX Dallas Houston
```

In writing `printAllStates,` you may call `printOneState` as specified in part (b). Assume that `printOneState` works as specified, regardless of what you wrote in part (b).

Complete method `printAllStates` below. A solution that creates an unnecessary instance of any Collection class will not receive full credit.

```
public void printAllStates()
```

3. A min-heap with no duplicate values is a binary tree in which the value in each node is smaller than the values in its children's nodes. A tree consisting of zero or one node is by default a min-heap. For example, trees `T1` and `T2` are min-heaps, but tree `T3` is not, because the shaded node has a larger value than one of its children (as shown by the thick line).



Consider the `MinHeap` class as shown below. The nodes of the min-heap will be represented by the `TreeNode` implementation class as shown in the Quick Reference. You may assume that the values stored in the `TreeNode` objects are `Comparable` objects.

```
public class MinHeap
{
  private TreeNode root;

  private TreeNode smallerChild(TreeNode t)
  { /* to be implemented in part (a) */ }

  private boolean isHeapOrdered(TreeNode t)
  { /* to be implemented in part (b) */ }

  /** @param t  a reference to a TreeNode
   *          Precondition: t is not null
   */
  private TreeNode removeMinHelper(TreeNode t)
  { /* to be implemented in part (c) */ }

  public Object removeMin()
  {
    if (root == null)
      return null;
    else
    {
      Object result = root.getValue();
      root = removeMinHelper(root);
      return result;
    }
  }

  // There may be instance variables, constructors, and methods that are not shown.
}
```

(a) Write method `smallerChild,` which is described as follows. `smallerChild` returns a reference to the child node of `t` that contains the smaller value (or `null` if `t` is a leaf node). If `t` has only one child, `smallerChild` should return a reference to that child. If `t` is `null,` `smallerChild` should return `null.`

The following table shows the results of several calls to `smallerChild.`

Method call	**Return value**
`smallerChild(T2)`	`P`
`smallerChild(P)`	`Q`
`smallerChild(Q)`	`R`
`smallerChild(R)`	`null`
`smallerChild(null)`	`null`

Complete method `smallerChild` below.

```
private TreeNode smallerChild(TreeNode t)
```

(b) Write method `isHeapOrdered,` which is described as follows. `isHeapOrdered` returns true if `t` is a min-heap and false otherwise.

In writing `isHeapOrdered,` you may call method `smallerChild` specified in part (a). Assume that `smallerChild` works as specified, regardless of what you wrote in part (a).

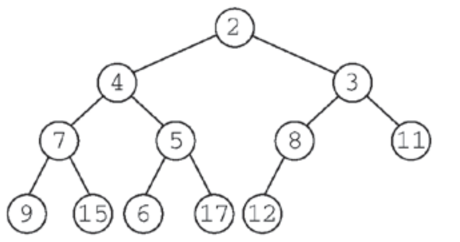
Complete method `isHeapOrdered` below.

```
private boolean isHeapOrdered(TreeNode t)
```

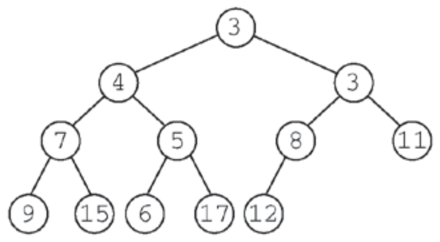
(c) The `removeMinHelper` method removes the minimum value from a min-heap and returns the modified min-heap. To do so, replace the value in the root node with the smaller of its children's values and then recursively call `removeMinHelper` on the subtree rooted at the smaller child. If the min-heap consists of a single node, that node is removed and `null` is returned.

For example, the following diagram illustrates the steps to restore the heap after removing 2 from the root node.

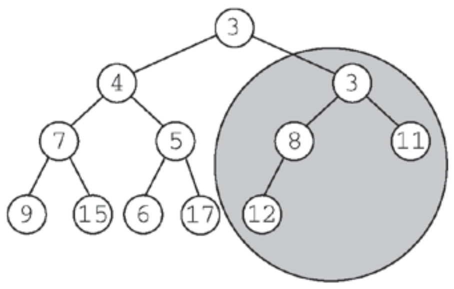
1. Initial



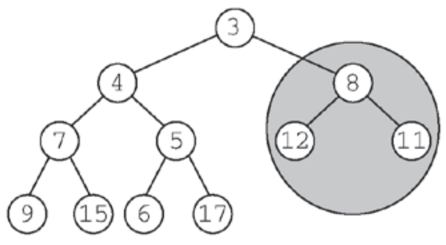
2. After replacing root node with smaller child's value



3. Subtree to be used for recursive call



4. After recursive call



Write method `removeMinHelper,` which is described as follows. `removeMinHelper` should remove the minimum item from the heap and return the resulting heap.

In writing `removeMinHelper,` you may call method `smallerChild` specified in part (a). Assume that `smallerChild` works as specified, regardless of what you wrote in part (a).

Complete method `removeMinHelper` below.

```
/** @param t  a reference to a TreeNode
 *          Precondition: t is not null
 */
private TreeNode removeMinHelper(TreeNode t)
```

Suggested Solutions to Free-Response Questions

Note: There are many correct variations of these solutions.

Question 1

(a)

```
private static ArrayList<Queue<Integer>> itemsToQueues
                                          (int[] nums, int k)
{
    ArrayList<Queue<Integer>> queues = new
    ArrayList<Queue<Integer>>(10);

    for (int j = 0; j < queues.length(); j++)
      queues.add(new LinkedList<Integer>());

    for (int j = 0; j < nums.length; j++)
    {
      int index = getDigit(nums[j], k);
      queues.get(index).add(new Integer(nums[j]));
    }

    return queues;
}
```

(b)

```
private static int[] queuesToArray(
                        ArrayList<Queue<Integer>> queues,
                        int numVals)
{
    int index = 0;
    int[] nums = new int[numVals];

    for (Queue<Integer> que : queues)
    {
      while (!que.isEmpty())
      {
        Integer val = que.remove();
        nums[index] = val.intValue();
        index++;
      }
    }

    return nums;
}
```

(c)

```
public static int[] sort(int[] nums, int numDigits)
{
  ArrayList<Queue<Integer>> qlist;

  for (int j = 0; j < numDigits; j++)
  {
    qlist = itemsToQueues(nums, j);
    nums = queuesToArray(qlist, nums.length);
  }

  return nums;
}
```

Question 2

(a)

```
public void addCityToMap(CityInfo theCity)
{
  Set<String> cities = theMap.get(theCity.state());

  if (cities == null)
  {
    cities = new HashSet<String>();
    theMap.put(theCity.state(), cities);
  }

  cities.add(theCity.city());
}
```

Commentary: Note that a `TreeSet` is not acceptable in place of a `HashSet` because it does not meet the requirements for expected running time as stated in the problem.

(b)

```
public void printOneState(String theState)
{
  System.out.print(theState);

  Set<String> cities = theMap.get(theState);

  for (String city : cities)
    System.out.print(" " + city);

  System.out.println();
}
```

Commentary: The call to `theMap.get` takes $O(\log n)$ time. Iterating through a `HashSet` takes $O(c)$ time. See the commentary on the topic outline for more information.

(c)

```
public void printAllStates()
{
  for (String state : theMap.keySet())
    printOneState(state);
}
```

Question 3

(a)

```
private TreeNode smallerChild(TreeNode t)
{
  if (t == null)
    return null;
  else if (t.getLeft() == null)
    return t.getRight();
  else if (t.getRight() == null)
    return t.getLeft();
  else
  {
    Comparable left = (Comparable) (t.getLeft().getValue());
    Comparable right = (Comparable) (t.getRight().getValue());
    if (left.compareTo(right)< 0)
      return t.getLeft();
    else
      return t.getRight();
  }
}
```

(b)

```
private boolean isHeapOrdered(TreeNode t)
{
  TreeNode smaller = smallerChild(t);
  if (smaller == null)
    return true;
  else
  {
    Comparable temp = (Comparable) (t.getValue());
    return (temp.compareTo(smaller.getValue()) < 0) &&
           isHeapOrdered(t.getLeft()) &&
           isHeapOrdered(t.getRight());
  }
}
```

(c)

```
/** @param t  a reference to a TreeNode
 *         Precondition:  t  is not null
 */
private TreeNode removeMinHelper(TreeNode t)
{
  TreeNode smaller = smallerChild(t);
  if (smaller == null)
    return null;

  t.setValue(smaller.getValue());
  if (smaller == t.getLeft())
    t.setLeft(removeMinHelper(t.getLeft()));
  else
    t.setRight(removeMinHelper(t.getRight()));

  return t;
}
```

APPENDIX A

AP Computer Science Java Subset

The AP Java subset is intended to outline the features of Java that may appear on AP Computer Science Exams. The AP Java subset is NOT intended as an overall prescription for computer science courses—the subset itself will need to be supplemented in order to cover a typical introductory curriculum. For example, I/O is essential to programming and can be done in many different ways. Because of this, specific I/O features are not tested on the AP Computer Science Exams.

This appendix describes the Java subset that students will be expected to understand when they take an AP Computer Science Exam. A number of features are also mentioned that are potentially relevant in a CS1/2 course but are not specifically tested on these exams.

The three principles that guided the formulation of the subset were as follows:

1. Enable the test designers to formulate meaningful questions
2. Help students with test preparation
3. Enable instructors to follow a variety of approaches in their courses

To help students with test preparation, the AP Java subset was intentionally kept small. Language constructs and library features were omitted that did not add significant functionality and that can, for the formulation of exam questions, be expressed by other mechanisms in the subset. For example, inner classes or the `StringBuffer` class are not essential for the formulation of exam questions—the exams use alternatives that can be easily understood by students. Of course, these constructs add significant value for programming. Omission of a feature from the AP Java subset does not imply any judgment that the feature is inferior or not worthwhile.

The AP Java subset gives instructors flexibility in how they use Java in their courses. For example, some courses teach how to perform input/output using streams or readers/writers, others teach graphical user interface construction, and yet others rely on a tool or library that handles input/output. For the purpose of the AP Computer Science Exam, these choices are incidental and are not central for the mastery of computer science concepts. The AP Java subset does not address handling of user input at all. That means that the subset is not complete. To create actual programs, instructors need to present additional mechanisms in their classes.

The following section contains the language features that may be tested on the AP Computer Science Exams. A summary table is provided that outlines the features that are tested on the A and AB Exams, the AB Exam only, and those features that are useful but are not tested on either exam. A list specifying which Standard Java classes and methods will be used on the exam is available at AP Central. There will be no extra AP classes provided as part of the subset.

Language Features

1. The primitive types `int`, `double`, and `boolean` are part of the AP Java subset. The other primitive types `short`, `long`, `byte`, `char`, and `float` are not in the subset. In particular, students need not be aware that strings are composed of `char` values. Introducing `char` does not increase the expressiveness of the subset. Students already need to understand string concatenation, `String.substring`, and `String.equals`. Not introducing `char` avoids complexities with the char/int conversions and confusion between `"x"` and `'x'`.
2. Arithmetic operators: `+`, `−`, `*`, `/`, `%` are part of the AP Java subset.
3. The increment/decrement operators `++` and `−−` are part of the AP Java subset. These operators are used only for their side effect, not for their value. That is, the postfix form (for example, `x++`) is always used, and the operators are not used inside other expressions. For example, `a[x++]` is not used.
4. The assignment operator = is part of the AP Java subset. The combined arithmetic/assignment operators `+=`, `−=`, `*=`, `/=`, `%=` are part of the AP Java subset, although they are used simply as a shorthand and will not be used in the adjustment part of a `for` loop.
5. Relational operators `==`, `!=`, `<`, `<=`, `>`, `>=` are part of the AP Java subset.
6. Logical operations `&&`, `||`, `!` are part of the AP Java subset. Students need to understand the "short circuit" evaluation of the `&&` and `||` operators. The logical `&`, `|` and `^` and the bit operators `<<`, `>>`, `>>>`, `&`, `~`, `|`, `^` are not in the subset.
7. The ternary `?:` operator is not in the subset.
8. The numeric casts `(int)` and `(double)` are part of the AP Java subset. Since the only primitive types in the subset are `int`, `double`, and `boolean`, the only required numeric casts are the cast `(int)` and the cast `(double)`. Students are expected to understand "truncation towards 0" behavior as well as the fact that positive floating-point numbers can be rounded to the nearest integer as `(int)(x + 0.5)`, negative numbers as `(int)(x − 0.5)`. Autoboxing, that is, the automatic conversion between primitive types `(int, double)` and the corresponding wrapper classes `(Integer, Double)` is not in the subset.
9. String concatenation + is part of the AP Java subset. Students are expected to know that concatenation converts numbers to strings and invokes `toString` on objects. String concatenation can be less efficient than using the `StringBuffer` class. However, for greater simplicity and conceptual clarity, the `StringBuffer` class is not in the subset.
10. The escape sequences inside strings `\\`, `\"`, `\n` are part of the AP Java subset. The `\t` escape and Unicode `\uxxxx` escapes are not in the subset. The `\'` escape is only necessary inside character literals and is not in the subset.

11. User input is not part of the AP Java subset. There are many possible ways for supplying user input: e.g., by reading from a `Scanner,` reading from a stream (such as a file or a URL) or from a dialog box. There are advantages and disadvantages to the various approaches. The exam does not prescribe any one approach. Instead, if reading input is necessary, it will be indicated in a way similar to the following:

    ```
    double x = /* call to a method that
          reads a floating-point number */;
    ```

    or

    ```
    double x = . . .;
          // read user input
    ```

    Converting strings to numeric values (e.g., with `Integer.parseInt`) is not in the subset.

12. Testing of output is restricted to `System.out.print` and `System.out.println.` As with user input, there are many possible ways for directing the output of a program: e.g., to `System.out,` to a file, or to a text area in a graphical user interface. The AP Java subset includes the ability to print output to `System.out,` because it makes it easy to formulate questions. Since most graphical environments allow printing of debug messages to `System.out` (with output being collected in a special window, e.g., the "Java console" in a browser), students are usually familiar with this method of producing output. Formatted output (e.g., with `NumberFormat` or `System.out.printf`) is not in the subset.

13. The `main` method and command-line arguments are not in the subset. The AP Computer Science Development Committee does not prescribe any particular approach for program invocation. In free-response questions, students are not expected to invoke programs. In case studies, program invocation with `main` may occur, but the `main` method will be kept very simple.

14. Arrays: One-dimensional arrays and two-dimensional rectangular arrays are part of the AP Java subset. Both arrays of primitive types (e.g., `int[]`) and arrays of objects (e.g., `Student[]`) are in the subset. Initialization of named arrays (`int[] arr = { 1, 2, 3 };`) is part of the AP Java subset. Two-dimensional arrays will only be tested on the AB Exam. Arrays with more than two dimensions (e.g., `rubik = new Color[3][3][3]`) are not in the subset. "Ragged" arrays (e.g., `new int[3][]`) are not in the subset. In particular, students do not need to know that an `int[3][3]` really is an "array of arrays" whose rows can be replaced with other `int[]` arrays. However, students are expected to know that `arr[0].length` is the number of columns in a rectangular two-dimensional array `arr.` Anonymous arrays (e.g., `new int[] { 1, 2, 3 }`) are not in the subset.

15. The control structures `if, if/else, while, for,` (including the enhanced `for` loop), `return` are part of the AP Java subset. The `do/while, switch,` plain and labeled `break` and `continue` statements are not in the subset.

16. Methods: Method overloading (e.g., `MyClass.someMethod(String str)` and `MyClass.someMethod(int num))` is part of the AP Java subset. Students should understand that the signature of a method depends on the number, types, and order of its parameters but does not include the return type of the method.

    Methods with a variable number of parameters are not in the subset.

17. Classes: Students are expected to construct objects with the `new` operator, to supply construction parameters, and to invoke accessor and modifier methods. Students are expected to modify existing classes (by adding or modifying methods and instance variables). Students are expected to design their own classes.

18. Visibility: In the AP Java subset, all classes are `public.` All instance variables are `private.` Methods, constructors, and constants (`static final` variables) are either `public` or `private.` The AP Java subset does not use `protected` and package (default) visibility.

19. The AP Java subset uses `/* */, //,` and `/**  */` comments. Javadoc comments `@param` and `@return` are part of the subset.

20. The `final` keyword is only used for `final` block scope constants and `static final` class scope constants. `final` parameters or instance variables, `final` methods, and `final` classes are not in the subset.

21. The concept of `static` methods is a part of the subset. Students are required to understand when the use of `static` methods is appropriate. In the exam, `static` methods are always invoked through a class, never an object (i.e., `ClassName.staticMethod(),` not `obj.staticMethod()).`

22. `static final` variables are part of the subset; other `static` variables are not.

23. The `null` reference is part of the AP Java subset.

24. The use of `this` is restricted to passing the implicit parameter in its entirety to another method (e.g., `obj.method(this)`) and to descriptions such as "the implicit parameter `this`". Using `this.var` or `this.method(args)` is not in the subset. In particular, students are not required to know the idiom `"this.var = var",` where `var` is both the name of an instance variable and a parameter variable. Calling other constructors from a constructor with the `this(args)` notation is not in the subset.

25. The use of **`super`** to invoke a superclass constructor **`(super(args))`** or to invoke a superclass method (i.e., **`super.method(args)`**) is part of the AP Java subset.

26. Students are expected to implement constructors that initialize all instance variables. Class constants are initialized with an initializer:

    ```
    public static final MAX_SCORE = 5;
    ```

    The rules for default initialization (with 0, `false` or `null`) are not in the subset. Initializing instance variables with an initializer is not in the subset. Initialization blocks are not in the subset.

27. Students are expected to extend classes and implement interfaces. Students are also expected to have a knowledge of inheritance that includes understanding the concepts of method overriding and polymorphism. Students are expected to implement their own subclasses.

28. Students are expected to read the definitions of interfaces and abstract classes and understand that the abstract methods need to be redefined in nonabstract classes. Students are expected to write interfaces or class declarations when given a general description of the interface or class. On the AB Exam, students are expected to define their own interfaces and abstract classes.

29. Students are expected to understand the difference between object equality (`equals`) and identity (==). The implementation of `equals` and `hashCode` methods are not in the subset.

30. Cloning is not in the subset, because of the complexities of implementing the `clone` method correctly and the fact that `clone` is rarely required in Java programs.

31. The `finalize` method is not in the subset.

32. Students are expected to understand that conversion from a subclass reference to a superclass reference is legal and does not require a cast. Class casts (generally from `Object` to another class) are part of the AP Java subset, to enable the use of generic collections; for example:

    ```
    Person p = (Person) listNode.getValue()
    ```

    The `instanceof` operator is not in the subset. Array type compatibility and casts between array types are not in the subset.

33. Students are expected to have a basic understanding of packages and a reading knowledge of `import` statements of the form

    ```
    import packageName.subpackageName.ClassName;
    ```

    `import` statements with a trailing `*`, static imports, packages and methods for locating class files (e.g., through a class path) are not in the subset.

34. Nested and inner classes are not in the subset.

35. The use of generic collection classes and interfaces are in the subset, but students need not implement generic classes or methods.

36. Enumerations, annotations, and threads are not in the subset.

37. Students are expected to understand the exceptions that occur when their programs contain errors (in particular, `NullPointerException`, `ArrayIndexOutOfBoundsException`, `ArithmeticException`, `ClassCastException`, `IllegalArgumentException`). On the AB Exam, students are expected to be able to throw the unchecked `IllegalStateException` and `NoSuchElementException` in their own methods (principally when implementing collection ADTs). Checked exceptions are not in the subset. In particular, the `try/catch/finally` statements and the `throws` modifier are not in the subset.

Summary Table

Tested in A, AB Exam	Tested in AB Exam only	Potentially relevant to CS1/CS2 course but not tested
`int, double, boolean`		`short, long, byte, char, float`
`+ , −, *, /, %, ++, −−, =, +=, -=, *=, /=, %=, ==, !=, <, <=, >, >=,`		Using the values of `++, −−` expressions in other expressions
`&&, ||, !,` and short-circuit evaluation		`<<, >>, >>>, &, ~, |, ^, ?:`
`(int), (double)`		Other numeric casts such as `(char)` or `(float)`
String concatenation		`StringBuffer`
Escape sequences `\"`, `\\`, `\n` inside strings		Other escape sequences (`\'`, `\t`, `\unnnn`)
`System.out.print`, `System.out.println`		`Scanner, System.in`, Stream input/output, GUI input/output, parsing input, formatted output

Tested in A, AB Exam	Tested in AB Exam only	Potentially relevant to CS1/CS2 course but not tested
		`public static void main(String[] args)`
1-dimensional arrays	2-dimensional rectangular arrays	Arrays with 3 or more dimensions, ragged arrays
`if`, `if/else`, `while`, `for`, enhanced `for`, `return`		`do/while`, `switch`, `break`, `continue`
Modify existing classes, design classes		
`public` classes, `private` instance variables, `public` or `private` methods or constants		`protected` or package visibility
`@param`, `@return`		`javadoc`
`static final` class variables		`final` local variables, `final` parameter variables, instance variables, methods or classes
`static` methods		`static` non-`final` variables
`null`, `this`, `super()`, `super(args)`, `super.method(args)`		`this.var`, `this.method(args)`, `this(args)`

Tested in A, AB Exam	Tested in AB Exam only	Potentially relevant to CS1/CS2 course but not tested
Constructors and initialization of `static` variables		Default initialization of instance variables, initialization blocks
Understand inheritance hierarchies. Design and implement subclasses. Modify subclass implementations and implementations of interfaces.		
Understand the concepts of abstract classes and interfaces.	Design and implement abstract classes and interfaces.	
Understand `equals`, `==`, and `!=` comparison of objects, `String.compareTo` Conversion to supertypes and `(Subtype)` casts	`Comparable.compareTo`	`clone`, implementation of `equals`, generic `Comparable<T>` `instanceof`
		Nested classes, inner classes enumerations
Package concept, `import packageName.className;`		`import packageName.*` `static import` defining packages, class path

Tested in A, AB Exam	Tested in AB Exam only	Potentially relevant to CS1/CS2 course but not tested
Exception concept, common exceptions	Throwing standard unchecked exceptions	Checked exceptions `try/catch/finally`, `throws`
`String`, `Math`, `Object`, `ArrayList`	`Comparable`, `List`, `Set`, `Map`, `Iterator`, `ListIterator`, `LinkedList`, `HashSet`, `TreeSet`, `HashMap`, `TreeMap`, `Stack`, `Queue`, `PriorityQueue`	Sorting methods in `Arrays` and `Collections`
Wrapper classes (`Integer`, `Double`)		autoboxing

A P P E N D I X B

Standard Java Library Methods Required for AP CS A

Accessible Methods from the Java Library That May Be Included on the Exam

```
class java.lang.Object
•   boolean equals(Object other)
•   String toString()

class java.lang.Integer
•   Integer(int value)
•   int intValue()

class java.lang.Double
•   Double(double value)
•   double doubleValue()

class java.lang.String
•   int length()
•   String substring(int from, int to) // returns the substring beginning at from
                                       // and ending at to-1
•   String substring(int from)         // returns substring(from, length())
•   int indexOf(String str)            // returns the index of the first occurrence of str;
                                       // returns -1 if not found
•   int compareTo(String other)        // returns a value < 0 if this is less than other
                                       // returns a value = 0 if this is equal to other
                                       // returns a value > 0 if this is greater than other
class java.lang.Math
•   static int abs(int x)
•   static double abs(double x)
•   static double pow(double base, double exponent)
•   static double sqrt(double x)
•   static double random()             // returns a double in the range [0.0, 1.0)

class java.util.ArrayList<E>
•   int size()
•   boolean add(E obj)                 // appends obj to end of list; returns true
•   void add(int index, E obj)         // inserts obj at position index (0 ≤ index ≤ size),
                                       // moving elements at position index and higher
                                       // to the right (adds 1 to their indices) and adjusts size
•   E get(int index)
•   E set(int index, E obj)            // replaces the element at position index with obj
                                       // returns the element formerly at the specified position
•   E remove(int index)                // removes element from position index, moving elements
                                       // at position index + 1 and higher to the left
                                       // (subtracts 1 from their indices) and adjusts size
                                       // returns the element formerly at the specified position
```

A P P E N D I X   C

Standard Java Library Methods Required for AP CS AB

Note: The `java.util.Stack` and `java.util.PriorityQueue` classes and the `java.util.Queue interface` (page 103 in this Appendix) each inherit methods that access elements in a way that violates their abstract data structure definitions. Solutions that use objects of types `Stack`, `Queue`, and `PriorityQueue` should use only the methods listed in the Appendix for accessing and modifying those objects. The use of other methods in free-response solutions may not receive full credit.

Accessible methods from the Java library that may be included on the exam

```
class java.lang.Object
•   boolean equals(Object other)
•   String toString()
•   int hashCode()

interface java.lang.Comparable*
•   int compareTo(Object other)  // returns a value < 0 if this is less than other
                                 // returns a value = 0 if this is equal to other
                                 // returns a value > 0 if this is greater than other

class java.lang.Integer implements java.lang.Comparable*
•   Integer(int value)
•   int intValue()

class java.lang.Double implements java.lang.Comparable*
•   Double(double value)
•   double doubleValue()

class java.lang.String implements java.lang.Comparable*
•   int length()
•   String substring(int from, int to) // returns the substring beginning at from
                                       // and ending at to-1
•   String substring(int from)         // returns substring(from, length())
•   int indexOf(String str)            // returns the index of the first occurrence of str;
                                       // returns -1 if not found

class java.lang.Math
•   static int abs(int x)
•   static double abs(double x)
•   static double pow(double base, double exponent)
•   static double sqrt(double x)
•   static double random()             // returns a double in the range [0.0, 1.0)
```

* The AP Java subset uses the "raw" `Comparable` interface, not the generic `Comparable<T>` interface.

`interface java.util.List<E>`

- `int size()`
- `boolean add(E obj)` // appends `obj` to end of list; returns `true`
- `void add(int index, E obj)` // inserts `obj` at position `index` $(0 \leq \text{index} \leq \text{size})$, // moving elements at position `index` and higher // to the right (adds 1 to their indices) and adjusts size
- `E get(int index)`
- `E set(int index, E obj)` // replaces the element at position `index` with `obj` // returns the element formerly at the specified position
- `E remove(int index)` // removes element from position `index`, moving elements // at position `index + 1` and higher to the left // (subtracts 1 from their indices) and adjusts size // returns the element formerly at the specified position
- `Iterator<E> iterator()`
- `ListIterator<E> listIterator()`

`class java.util.ArrayList<E> implements java.util.List<E>`

`class java.util.LinkedList<E> implements java.util.List<E>, java.util.Queue<E>`

- `void addFirst(E obj)`
- `void addLast(E obj)`
- `E getFirst()`
- `E getLast()`
- `E removeFirst()`
- `E removeLast()`

`interface java.util.Queue<E>` // implemented by `LinkedList<E>`

- `boolean add(E obj)` // enqueues `obj` at the end of the queue; returns `true`
- `E remove()` // dequeues and returns the element at the front of the queue
- `E peek()` // returns the element at the front of the queue; // `null` if the queue is empty
- `boolean isEmpty()`

`class java.util.PriorityQueue<E>` // `E` should implement `Comparable`*

- `boolean add(E obj)` // adds `obj` to the priority queue; returns `true`
- `E remove()` // removes and returns the minimal element from the priority queue
- `E peek()` // returns the minimal element from the priority queue; // `null` if the priority queue is empty
- `boolean isEmpty()`

`class java.util.Stack<E>`

- `E push(E item)` // pushes `item` onto the top of the stack; returns `item`
- `E pop()` // removes and returns the element at the top of the stack
- `E peek()` // returns the element at the top of the stack; // throws an exception if the stack is empty
- `boolean isEmpty()`

* The AP Java subset uses the "raw" `Comparable` interface, not the generic `Comparable<T>` interface.

```
interface java.util.Iterator<E>
•   boolean hasNext()
•   E next()
•   void remove()                    // removes the last element that was returned by next

interface java.util.ListIterator<E> extends java.util.Iterator<E>
•   void add(E obj)                  // adds obj before the element that will be returned by next
•   void set(E obj)                  // replaces the last element returned by next with obj

interface java.util.Set<E>
•   int size()
•   boolean contains(Object obj)
•   boolean add(E obj)               // if obj is not present in this set, adds obj and returns true;
                                     // otherwise, returns false
•   boolean remove(Object obj)       // if obj is present in this set, removes obj and returns true;
                                     // otherwise, returns false
•   Iterator<E> iterator()

class java.util.HashSet<E> implements java.util.Set<E>
class java.util.TreeSet<E> implements java.util.Set<E>

interface java.util.Map<K,V>
•   int size()
•   boolean containsKey(Object key)
•   V put(K key, V value)            // associates key with value
                                     // returns the value formerly associated with key
                                     // or null if key was not present
•   V get(Object key)                // returns the value associated with key
                                     // or null if there is no associated value
•   V remove(Object key)             // removes and returns the value associated with key;
                                     // returns null if there is no associated value
•   Set<K> keySet()

class java.util.HashMap<K,V> implements java.util.Map<K,V>
class java.util.TreeMap<K,V> implements java.util.Map<K,V>
```

APPENDIX D

Implementation Classes for Linked List and Tree Nodes (AP CS AB)

Implementation classes for linked list and tree nodes

Unless otherwise noted, assume that a linked list implemented from the `ListNode` class does not have a dummy header node.

```
public class ListNode
{
  private Object value;
  private ListNode next;

  public ListNode(Object initValue, ListNode initNext)
    { value = initValue; next = initNext; }

  public Object getValue() { return value; }
  public ListNode getNext() { return next; }

  public void setValue(Object theNewValue) { value = theNewValue; }
  public void setNext(ListNode theNewNext) { next = theNewNext; }
}
```

Unless otherwise noted, assume that a tree implemented from the `TreeNode` class does not have a dummy root node.

```
public class TreeNode
{
  private Object value;
  private TreeNode left;
  private TreeNode right;

  public TreeNode(Object initValue)
    { value = initValue; left = null; right = null; }

  public TreeNode(Object initValue, TreeNode initLeft, TreeNode initRight)
    { value = initValue; left = initLeft; right = initRight; }

  public Object getValue() { return value; }
  public TreeNode getLeft() { return left; }
  public TreeNode getRight() { return right; }

  public void setValue(Object theNewValue) { value = theNewValue; }
  public void setLeft(TreeNode theNewLeft) { left = theNewLeft; }
  public void setRight(TreeNode theNewRight) { right = theNewRight; }
}
```

Teacher Support

AP Central® (apcentral.collegeboard.com)

You can find the following Web resources at AP Central (free registration required):

- AP Course Descriptions, AP Exam questions and scoring guidelines, sample syllabi, and feature articles.
- A searchable Institutes and Workshops database, providing information about professional development events.
- The Course Home Pages (apcentral.collegeboard.com/coursehomepages), which contain insightful articles, teaching tips, activities, lab ideas, and other course-specific content contributed by colleagues in the AP community.
- Moderated electronic discussion groups (EDGs) for each AP course, provided to facilitate the exchange of ideas and practices.

AP Publications and Other Resources

Free AP resources are available to help students, parents, AP Coordinators, and high school and college faculty learn more about the AP Program and its courses and exams. Visit apcentral.collegeboard.com/freepubs.

Teacher's Guides and Course Descriptions may be downloaded free of charge from AP Central; printed copies may be purchased through the College Board Store (store.collegeboard.com). Released Exams and other priced AP resources are available at the College Board Store.

Teacher's Guides

For those about to teach an AP course for the first time, or for experienced AP teachers who would like to get some fresh ideas for the classroom, the *Teacher's Guide* is an excellent resource. Each *Teacher's Guide* contains syllabi developed by high school teachers currently teaching the AP course and college faculty who teach the equivalent course at colleges and universities. Along with detailed course outlines and innovative teaching tips, you'll also find extensive lists of suggested teaching resources.

Course Descriptions

Course Descriptions are available for each AP subject. They provide an outline of each AP course's content, explain the kinds of skills students are expected to demonstrate in the corresponding introductory college-level course, and describe the AP Exam. Sample multiple-choice questions with an answer key and sample free-response questions are included. (The Course Description for AP Computer Science is available in PDF format only.)

Released Exams

Periodically the AP Program releases a complete copy of each exam. In addition to providing the multiple-choice questions and answers, the publication describes the process of scoring the free-response questions and includes examples of students' actual responses, the scoring standards, and commentary that explains why the responses received the scores they did.

Contact Us

National Office
Advanced Placement Program
45 Columbus Avenue
New York, NY 10023-6992
212 713-8066
E-mail: ap@collegeboard.org

AP Services
P.O. Box 6671
Princeton, NJ 08541-6671
609 771-7300
877 274-6474 (toll free in the U.S. and Canada)
E-mail: apexams@info.collegeboard.org

AP Canada Office
2950 Douglas Street, Suite 550
Victoria, BC, Canada V8T 4N4
250 472-8561
800 667-4548 (toll free in Canada only)
E-mail: gewonus@ap.ca

AP International Office
Serving all countries outside the U.S. and Canada
45 Columbus Avenue
New York, NY 10023-6992
212 373-8738
E-mail: apintl@collegeboard.org

Middle States Regional Office
Serving Delaware, District of Columbia, Maryland, New Jersey, New York, Pennsylvania, Puerto Rico, and the U.S. Virgin Islands
Two Bala Plaza, Suite 900
Bala Cynwyd, PA 19004-1501
866 392-3019
E-mail: msro@collegeboard.org

Midwestern Regional Office
Serving Illinois, Indiana, Iowa, Kansas, Michigan, Minnesota, Missouri, Nebraska, North Dakota, Ohio, South Dakota, West Virginia, and Wisconsin
6111 N. River Road, Suite 550
Rosemont, IL 60018-5158
866 392-4086
E-mail: mro@collegeboard.org

New England Regional Office
Serving Connecticut, Maine, Massachusetts, New Hampshire, Rhode Island, and Vermont
470 Totten Pond Road
Waltham, MA 02451-1982
866 392-4089
E-mail: nero@collegeboard.org

Southern Regional Office
Serving Alabama, Florida, Georgia, Kentucky, Louisiana, Mississippi, North Carolina, South Carolina, Tennessee, and Virginia
3700 Crestwood Parkway NW, Suite 700
Duluth, GA 30096-7155
866 392-4088
E-mail: sro@collegeboard.org

Southwestern Regional Office
Serving Arkansas, New Mexico, Oklahoma, and Texas
4330 South MoPac Expressway, Suite 200
Austin, TX 78735-6735
866 392-3017
E-mail: swro@collegeboard.org

Western Regional Office
Serving Alaska, Arizona, California, Colorado, Hawaii, Idaho, Montana, Nevada, Oregon, Utah, Washington, and Wyoming
2099 Gateway Place, Suite 550
San Jose, CA 95110-1051
866 392-4078
E-mail: wro@collegeboard.org

apcentral.collegeboard.com

I.N. 070082352