CollegeBoard AP

COMPUTER SCIENCE A

Course Description

MAY 2010, MAY 2011

The College Board

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 5,600 schools, colleges, universities, and other educational organizations. Each year, the College Board serves seven million students and their parents, 23,000 high schools, and 3,800 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.

Contents

Welcome to the AP® Program

For over 50 years, the College Board's Advanced Placement Program (AP) has partnered with colleges, universities, and high schools to provide students with the opportunity to take college-level course work and exams while still in high school. Offering more than 30 different subjects, each culminating in a rigorous exam, AP provides motivated and academically prepared students with the opportunity to earn college credit or placement and helps them stand out in the college admissions process. Taught by dedicated, passionate AP teachers who bring cutting-edge content knowledge and expert teaching skills to the classroom, AP courses help students develop the study skills, habits of mind, and critical thinking skills that they will need in college.

AP is accepted by more than 3,600 colleges and universities worldwide for college credit, advanced placement, or both on the basis of successful AP Exam grades. This includes over 90 percent of four-year institutions in the United States.

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 teachers (apcentral.collegeboard.com). Students can find more information at the AP student site (www.collegeboard.com/apstudents).

AP Courses

More than 30 AP courses in a wide variety of subject areas are now available. 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.

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, each AP Exam contains a free-response section (essays, problem solving, oral responses, etc.) as well as multiple-choice questions.

Written by a committee of college and university faculty and experienced AP teachers, the AP Exam is the culmination of the AP course and provides students with the opportunity to earn credit and/or placement in college. Exams are scored by college professors and experienced AP teachers using scoring standards developed by the committee.

AP Course Audit

The intent of the AP Course Audit is to provide secondary and higher education constituents with the assurance that an "AP" designation on a student's transcript is credible, meaning the AP Program has authorized a course that has met or exceeded the curricular requirements and classroom resources that demonstrate the academic rigor of a comparable college course. To receive authorization from the College Board to label a course "AP," teachers must participate in the AP Course Audit. Courses authorized to use the "AP" designation are listed in the AP Course Ledger made available to colleges and universities each fall. It is the school's responsibility to ensure that its AP Course Ledger entry accurately reflects the AP courses offered within each academic year.

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 www.collegeboard.com/apcourseaudit.

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 have taken the corresponding college course. This college-level equivalency is ensured through several AP Program processes:

- College faculty are involved 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.
  - Lead professional development seminars for new and experienced 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.
- 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.
- 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 the College Board Web site 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 specifically for colleges and universities that provides guidance in setting AP credit and placement policies. Additional resources, including links to AP research studies, released exam questions, and sample student responses at varying levels of achievement for each AP Exam are also available. 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. This tool helps students find the credit hours and/or advanced placement they may receive for qualifying exam grades within each AP subject at a specified institution. AP Credit Policy Info is available at www.collegeboard.com/ap/creditpolicy.

AP Computer Science A

Important Revisions to This Course Description

- Addition of two-dimensional arrays to list of standard data structures (see Topic Outline, page 8)
- Addition of java.util.List interface to Java A subset (see Java subset, Appendix A)
- Addition of Java constants Integer.MIN_VALUE and Integer.MAX_VALUE (see Java subset, Appendix A)
- Addition of static variables (also know as class variables) (see Java subset, Appendix A)

INTRODUCTION

The Advanced Placement Program offers a course and exam in introductory computer science. The course 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. For a listing of the topics addressed, see the AP Computer Science A topic outline on pages 8–10.

The nature of the AP course is suggested by the words "computer science" in the title. 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 the AP course. It 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 course exactly. The Association for Computing Machinery (ACM) and the Institute of Electrical and Electronics 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 taught in the first two years of college. The AP Computer Science A course is compatible with those topics that are taught in a typical CS1 course as described in the example curricula in the ACM/IEEE guidelines. Some colleges and universities may organize their curricula in alternative ways so that the topics of the AP Computer Science A course are spread over the first one or two college courses, with other topics from computer science interspersed.

The AP Computer Science A course can be offered by any secondary school that has faculty who possess the necessary expertise and have access to appropriate computing facilities. The course represents 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 COURSE

The AP Computer Science A 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 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 on pages 8–10 summarizes the content typically taught in the AP Computer Science A course.

Goals

The goals of the AP Computer Science A course are comparable to those in the introductory course for computer science majors offered in college and university computer science departments. It is not expected, however, that all students in the AP Computer Science A course will major in computer science at the university level. The AP Computer Science A 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 the AP Computer Science A course when interpreted within the context of the 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 A course has evolved to incorporate this approach.

Current offerings of the AP Computer Science A 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 include 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 A course will need to be prepared to use standard Java, as specified in the AP Java subset, on the AP Computer Science A Exam.**

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 the AP Computer Science A course.

Schools offering AP Computer Science A 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 the AP Computer Science A course include knowledge of basic algebra and experience in problem solving. A student in the AP Computer Science A 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.

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

Teaching the Course

The teacher should be prepared to present a college-level first course in computer science. The AP Computer Science A course is more than a course on programming. The emphasis in the course 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 considered for the AP Computer Science A Exam. This outline is intended to define the scope of the course but not necessarily the sequence.

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.

A. Program design
   1. Read and understand a problem description, purpose, and goals.
   2. Apply data abstraction and encapsulation.
   3. Read and understand class specifications and relationships among the classes ("is-a," "has-a" relationships).
   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.
   2. Choose appropriate data representation and algorithms.
   3. Apply functional decomposition.
   4. Extend a given class using inheritance.

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.

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

C. Java library classes (included in the 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.

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.

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

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.

A. Simple data types (int, boolean, double)
B. Classes
C. Lists
D. Arrays

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.

A. Operations on data structures previously listed
   1. Traversals
   2. Insertions
   3. Deletions
B. Searching
   1. Sequential
   2. Binary
C. Sorting
   1. Selection
   2. Insertion
   3. Mergesort

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 addressed in detail but should be considered throughout the course.

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 topic outline below summarizes the content of the AP Computer Science A 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 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

Students should be able to develop the parts of a program when given its design. This would include an understanding of how to apply the data abstractions included in the course (classes and arrays). Students are not expected to develop a full program design.

Students 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.

B. Class Design

A fundamental part of the development of an object-oriented program is the design of a class. Students 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.

Given a design for a class, either their own or one provided, students 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.

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 should be able to write a class that implements an interface.

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 A 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 reusing 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 AP CS A Exam may include questions that ask about the design as well as the implementation of classes or a simple hierarchy of 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 (see page 38).

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 <u>signatures</u> (access specifier, return type, method identifier, parameter list) and <u>not</u> 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 A 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.

Students are expected to demonstrate an understanding of the concept of recursion and to trace recursive method calls.

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; they also 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.

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.

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. 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 the AP CS A 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 should be comfortable working with one-dimensional and two-dimensional lists of data and should be familiar with using Java arrays and the `ArrayList` class to implement such lists. 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 the `List` interface (and its implementation by the `ArrayList` class) for which students are responsible are specified in the AP Java subset (see Appendixes).

V. Standard Algorithms

The AP CS A course indicates 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 Merge sort. 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.

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 the AP CS A 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 the AP CS A course wherever appropriate, rather than taught as a separate unit. Participation in the AP CS A course provides an opportunity to discuss issues such 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 A course. 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 A Exam 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. The AP Computer Science A Exam will contain several multiple-choice questions and one free-response question targeting material from the case study. Printed excerpts from the case study programs will accompany the exam.

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 EXAM

The AP Computer Science A Exam is 3 hours long and seeks to determine how well students have mastered the concepts and techniques contained in the course outline.

The 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 the AP Computer Science A Exam require students to demonstrate their ability to design, write, analyze, and document programs and subprograms.

Minor points of syntax are not tested on the exam. 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 exam, 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 the exam, the multiple-choice section and the free-response section are given equal weight. Because the 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 17 and 38 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. The answer key for the Computer Science A multiple-choice questions is on page 37. 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.

```
List<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 List<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 `List nums` initially contains the following `Integer` values.

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

What will `List 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.

```
/** Precondition: 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

19. 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}}
```

20. Consider the following methods.

```
public List<Integer> process1(int n)
{
  List<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)
{
  List<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`.

21. 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`.

22. 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 A Multiple-Choice Questions

1 – B	5 – D	9 – D	13 – C	17 – D	21 – D
2 – C	6 – A	10 – A	14 – D	18 – A	22 – D
3 – E	7 – A	11 – C	15 – D	19 – D	
4 – D	8 – C	12 – E	16 – D	20 – A	

Sample Free-Response Questions

Following is a representative set of 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 <u>ordered list of all the candidates</u>, 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(List<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 List<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,
                            List<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 List<String> candidatesWithFewest(
                    List<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,
                          List<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 List<String> candidatesWithFewest(
                    List<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"`

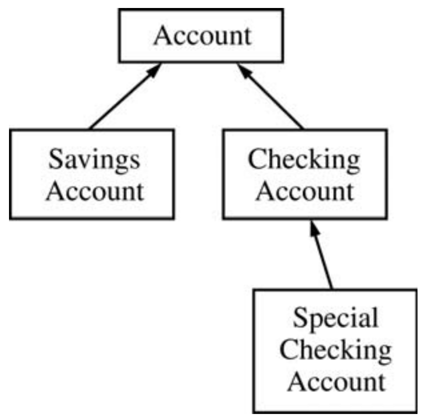
In writing `deleteAll,` you may call any of the methods in the `LineEditor` class, including `insert` and `delete` from parts (a) and (b). Assume that these methods work as specified, regardless of what you wrote in parts (a) and (b).

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 A Exam. 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 abstract 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 abstract 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 abstract double monthlyInterest();
}
```

(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,
                          List<String> candidateList)
{
    int numVotes = 0;

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

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

(b)

```
public List<String> candidatesWithFewest(
                              List<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];
    }

    List<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 List<String> candidatesWithFewest(
                 List<String> candidateList)
{
    List<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);
    }
}
```

APPENDIX A

AP Computer Science Java Subset

The AP Java subset is intended to outline the features of Java that may appear on the AP Computer Science A Exam. 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 address all topics in 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 A Exam.

This appendix describes the Java subset that students will be expected to understand when they take the AP Computer Science A Exam. A number of features are also mentioned that are potentially relevant in an introductory computer science course but are not specifically tested on the exam.

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 course. 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 A 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 A Exam. A summary table is provided that outlines the features that are tested on the exam and those features that are useful but are not tested. 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. 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, also called the for-each 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` variables are part of the subset.
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 implemented in a subclass. Students are expected to write interfaces or class declarations when given a general description of the interface or class.

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.

The `instanceof` operator is not included in the subset, although it may be used in the context of the case study. 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 AP Java 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`). 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

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

Tested in A Exam	Potentially relevant to CS1 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` (for-each), `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` 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 Exam	Potentially relevant to CS1 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 concept of abstract classes and interfaces.	
Understand `equals`, `==`, and `!=` comparison of objects, `String.compareTo`	`clone`, implementation of `equals`, generic Comparable<T>
Conversion to supertypes and `(Subtype)` casts	`instanceof`
	Nested classes, inner classes enumerations
Package concept, `import packageName.className;`	`import packageName.*` `static import` defining packages, class path

Tested in A Exam	Potentially relevant to CS1 course but not tested
Exception concept, common exceptions	Checked exceptions `try/catch/finally`, `throws`
`String`, `Math`, `Object`, `List`, `ArrayList`	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()`
- `Integer.MIN_VALUE` // minimum value represented by an `int`
- `Integer.MAX_VALUE` // maximum value represented by an `int`

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)

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 ≤ `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

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

Teacher Support

AP Central® (apcentral.collegeboard.com)

You can find the following Web resources at AP Central:

- 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

International Services
Serving all countries outside the U.S. and Canada

45 Columbus Avenue
New York, NY 10023-6992
212 373-8738
E-mail: international@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 North 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 Gaines Ranch Loop, 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. 080082747