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

II. Program Implementation

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

There are topics not included in the course outline that will be part of any introductory course. For example, input and output must be part of a course on computer programming. However, in a modern object-oriented approach to programming, there are many ways to handle input and output, including console-based character I/O, graphical user interfaces, and applets. Consequently, the AP CS 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.