AP Computer Science A Java Syllabus

Course Information

Text: Coyner, Scott. *Computer Science Java*. © 2019 CPM Educational Program

Course Summary

This computer science course teaches students to code fluently in an object oriented paradigm using the programming language Java. The course explicitly covers AP Computer Science A topics, encourages collaboration, and focuses on moving students of all programming abilities to a higher level. The instantiation, reference and use of objects along with a focus on the flow of information is strongly emphasized from beginning to end. Object oriented thinking, information hiding and/or implementation abstraction is part of every project. Additionally, reading class and method documentation is presented as a helpful part of assignment completion. **[CR4]**

The students will have daily access to a classroom set of computers with Internet access and equipped with the BlueJ IDE. Nearly as important to this curriculum is that the classroom space provides areas for groups of students to collaborate without computers in the way such as tables or desks that can be moved into teams of four. Students will work together in pairs or teams to work through programming problems. By working through complex problems together, students can support each other understand complex ideas and check each other's work, with the teacher serving as a support and guide as necessary. Communication and cooperation are vital for the success of all students in the classroom!

Assignments and Projects

Of the 58 lessons in the curriculum, 48 include one or more programming projects requiring at least one hour of computer laboratory time. **[CR6]** In fact, most of the course concepts are explored or demonstrated through java code, program comments and documentation contained within `*.jar` project files. Project file names included with each lesson include an underscore "_" followed by a description. Examples are shown below.

Folder/File Suffix Example	Contains ...
`PracticalUse_Example`	Source code and comments for student learning.
`RunThis_Demo`	A working program. However at least some of the `*.java` source code is not provided but the program runs using compiled `*.class` files.
`LessonTopic_Assignment`	The starting point for student work. Students add class files or methods to these projects. Some of the `*.java` source code may not be provided but might run using compiled `*.class` files.
`LessonTopic_Solution`	The source code for the solution to a programming assignment. *Not distributed to the students*.

Problem Generator

The problem generator is the most important supplemental resource for this curriculum. It is an excellent source of multiple choice and free response questions for assessment, team problems and review. These questions can assist students in learning computer science vocabulary and common java implementations. They also provide a valuable source of help while building student confidence in answering multiple choice exam questions.

Code tracing assignments aligned with chapter content are a significant part of the free-response problems that can be created easily with the problem generator. Code traces are a great opportunity to have students work together in teams and on paper. Code tracing enforces teamwork and mixed space practice of important concepts.

- Commonly used but counter intuitive operations such a modulus and integer division.
- Methods execute only when called so the order they appear is only aesthetic.
- The order of code statements within a method is integral to program logic as statements are executed sequentially subject to rules of iteration and if/else logic.
- Method execution is always terminated when a return statement is reached and control returns to where the method was called.
- The distinction between reserved words, comments and identifiers.

Course Timeline

Chapters 1 through 5 should be completed in the first semester and Chapters 6 through 10 in the second (before the AP exam). If time remains in the year after the AP exam, additional projects or units will be explored. Lessons require approximately 1 or 2 standard length class days. After Chapter 1, every week should include a paper and pencil, individual assessment or team activity not to exceed one regular class period. Team activities may include: code traces, team quizzes, AP Exam Free Response questions, or role plays.

Chapter 1: Object Anatomy

In this chapter students learn the basic Java vocabulary and primary structures within Java objects, such as instance variables and methods. **[CR4]**

Week	Lesson	Lesson Summary
1	1.0	**What Will I Learn?** – Students start working together in their study teams and are introduced to some basic computer science vocabulary, and consider what they can expect to learn from the course. **[CR7]**
	1.1	**Using BlueJ and Submitting Programs** – Students download, modify and resubmit a project using the IDE BlueJ.
	1.2	**Objects, Comments and Identifiers** – Students instantiate an object via the IDE and a `_Demo` project, then use its methods. Students conjecture about the object's attributes and operations, then verify their conjectures by examining the `_Example` project. In the `_Assignment.jar`, students will be identifying comments, reserved words and identifying different types of identifiers.
	1.3	**Identifiers and Reserved Words** – Students continue to work with objects and are introduced to different data types, reserved words and methods with parameters. **[CR2b]**
2	1.4	**Identifiers and More Data Types** – Students will gain familiarity with explicit data typing and learn more guidelines for creating identifiers through reading and `_Example` code, then answer multiple choice questions reviewing Chapter 1 concepts thus far. **[CR2b]**
	1.5	**Writing Methods** – The students will identify object attributes (instance variables) and operations (methods) from a `_Demo` project and then provide the implementation for three of the methods.
	1.6	**The Constructor** – By instantiating a `_Example` object which requires parameters, students will introduced to the constructor method and explore its use to set the initial state of an object, then students will provide the a constructor and the implementation for 3 methods in a `XCrunner_Assignment class.`
	1.7	**Java Mathematics** – Students will explore and see the results of integer division, an example of changing a variables type by casting and be introduced the modulus, then evaluate a variety of java math expressions.
3	1.8	**Four 4s** – Students will gain experience making their own unique methods with very specific constraints. Each method must use exactly four 4s and return a specified integer value.
		Writing Classes (2 lessons) – Students will gain confidence with all Chapter 1 concepts by exploring the functionality of a working `_Demo` project and providing the complete implementation for a `_Assignment` project that duplicates the demo.
	1.9.1	**Time Conversions** – Students design, implement and test an entire `class` that simulates an analog clock.
4	1.9.2	**DollarsNcents** – Students design, implement and test an entire `class` that simulates a retail coin change dispensing machine.

Chapter 2: Using Objects

Students begin to design and implement computer based solution to problems, using objects instantiated within the program and providing a "users prospective" through the use of a terminal based UI. **[CR1]**

Week	Lesson	Lesson Summary
4		**Instantiating and Using Objects** (2 lessons) – Students experience a "has-a" implementation where an object instantiates and has reference to other objects.
	2.1.1	**Instantiating Objects** – Students will modify a driver class that instantiates `Die` objects by adding a method that uses the `Die` objects' methods based on the `Die` objects' class documentation.
5	2.1.2	**Four 4s V2** – Students will gain confidence instantiating objects and using methods with no source code provided, only documentation.
	2.2	**`System.out`** – Students implement and run `System.out.print()` and `System.out.println()`.
	2.3	**Error Types** – Students will learn how to identify and debug Compiler, run-time and logic errors by debugging an example program that purposefully includes all of these error types, then completing a matching exercise.
6		**User Interface** (3 lessons) – Students will learn about import statements to `import` library classes and how to ask the program user for, and then read, information from the terminal. **[CR5]**
	2.4.1	**`Scanner`** – After reviewing some example code the students will design implement and test an entirely new project with two methods that read user input from the terminal, perform a calculation display the results on the terminal window.
	2.4.2	**Box Object** – Students will complete a new driver `class` file in its entirety to function with a given "has-a" `Box class` and its objects. Summarizing all concepts learned thus far.
7	2.4.3	**Converter** – Students will design implement and test an entire "has-a" `class` file referenced and used by a given driver `class`. Summarizing all concepts learned thus far.
	2.5	**Car Dealership**– Students will design implement and test a simple "has-a" `Car class` referenced and used an object of type `UsedCarSalesman`.

Chapter 3: Classes from Libraries

Students use the import statement and elements of the standard Java library to expand their programming capabilities. **[CR5]**

Week	Lesson	Lesson Summary
8		**Strings as Objects** (2 lessons) – The students will gain experience using Strings as objects by using a variety of `String` methods.
	3.1.1	**String Methods** – The students `String` methods that do not require knowledge of indexing including `compareTo` and `length()` while making a driver `class` to work with a pre-compiled word generator called `RandomWords`.
	3.1.2	**String Indexes** – The students will discover indexing of Strings by using indexed `substring` methods while making a driver class to work with a pre-compiled word generator called `RandomWords2`.
	3.2	**Rounding Numbers** – Students will explore two different techniques for rounding numbers contained in an `_Example` project and then make two methods: `calcSphere()` and `calcDistance()` that require rounding. Students also gain experience with some `static` methods of the `Math class`.
9	3.3	**Random Numbers** – Students will examine a `_Example` project using the `Math class` and `java.util.Random` class to make pseudo random integers then create a password generation `class`.
	3.4	**Aliases and References** – Students explore `_Example` projects that demonstrate objects passing by reference and primitives passing by value. The concept of object aliases is demonstrated for mutable and then immutable `String` objects. The students then design implement and test a `class` whose object accepts another object by reference and modifies its contents. **[CR1]**
10	3.5	**Binary, Hexadecimal Conversions** – Students will convert whole number quantities between base 10, binary and hexadecimal number systems. **[CR2b]**

Chapter 4: Iteration and Decisions

Students are introduced to some of the most commonly used programming algorithms for program decision making and iteration. **[CR2a]**

10		**`if else` and `or`** (3 lessons) – Students experience logical operators and Boolean expresions
	4.1.1	**Cascading `if else`**– Students explore cascading `if else` logic, testing of boundary cases, and debug common errors found in a `_Example` project then they create an `if else` implementation to analyze factors and multiples of integers
11	4.1.2	**Multiple `&& ||`** – Students investigate multiple `&& ||` logic in a `_Example` project then they create a "has-a" `class` with multiple `&& ||` implementations referenced by a provided UI `class` and analyze the properties of triangles.
	4.1.3	**Truth Tables**– Students explore the truth tables by example, completion and creation on paper. Review of evaluating expressions using multiple `&& ||` logic is included.
12		**The `while` Loop** (2 lessons) **–** Students will gain experience with `while` loop implementations and iteration.
	4.2.1	**`while` Loop Math –** Students see detailed implementations of while loops in a `_Example` projects and explain their functionality in plain language. Next students complete a `_Assignment` project that finds all of the positive factors of a given integer and shows the square root of the integer in exact simplified form.
	4.2.2	**`while` Loop Strings –** Students investigate the implementation of while loop that reverses Strings in a `_Example` project, practice tracing code implementing while loops and Strings and complete an `_Assignment` project that translates any word into a mythical language by inserting a given 'syllable' every specified number of characters.
13		**The `for` Loop** (2 lessons) **–** Students will gain experience with `for` loop implementations and iteration.
	4.3.1	**Word Analysis –** Students see a detailed comparison of `for` and `while` loop implementations in a `_Example` project then perform code traces using for loops. Students then complete a `_Assignment` project by implementing a `class` that works with a UI to count specified characters with given Strings.
	4.3.2	**Sentence Analysis** – Students will continue to gain experience with `for` loop implementations and iteration and will determine program requirements and use a top-down design approach to break tasks into various methods for a complex `_Assignment` project.
14	4.4	**Nested Loops** – Students will observe and utilize the multiplicative nature of nested loops through an `Example` project, code tracing problems and a `_Assignment` project where they produce ASCII graphics on the terminal window.
	4.5	**Working with GUIs** – Students will see the use of the special `main` method and an example of a Graphical User Interface (GUI) `class` then generate and execute a plan to implement a multi step solution to determine if a `String` represents a palindrome. **[CR4]**

Chapter 5: Arrays

Students are introduced to java arrays and use them to simulate situations requiring a fixed length, linear array data structure.

Week	Lesson	Lesson Summary
15	5.1	**Arrays of Primitives** – Students examine the implementation of arrays in a `_Example` project and use the information to answer five multiple-choice questions about array implementation. In the `_Assignment` project the students design, implement and test a `class` that uses an array of primitives to make statistical calculations. **[CR2b]**
	5.2.1	**Arrays of Objects** (2 lessons) – Students explore and create implementation of java arrays holding object references. **[CR2b]**
		Library of Books – Students investigate a `_Example` project that demonstrates some of the many types of homogenous data that can be stored using arrays including objects, use the information to answer some short answer questions and then implement 3 methods that manipulate an instance array of `Book` objects as part of a `_Assignment` project. **[CR2b]**
16	5.2.2	**Deck of Cards** – Students use the information about arrays from the previous assignments to answer some short answer questions and write a method on paper. Next students implement 3 methods that manipulate an instance array of `PlayingCard` objects as part of a `_Assignment` project. **[CR2b]**
	5.3	**StuffMart Parking Lot**– Students will design implement and test a `class` referenced within a `_Assignment` project simulating vehicles searching for spaces in a virtual parking lot. **[CR4]**

Chapter 6: Two-Dimensional Arrays

Students are introduced to two-dimensional java arrays and use them to simulate situations requiring a fixed size, two-dimensional array data structure.

Week	Lesson	Lesson Summary
1		**Two-Dimensional Arrays of Primitives** (2 lessons) – Students explore examples of two-dimensional array instantiation and initialization, then design and test selection and transversal operations on two-dimensional arrays of integers. **[CR2a] [CR2b]**
	6.1.1	**Introduction to Two-Dimensional Arrays** Students examine examples of two-dimensional array instantiation, then design, implement and test two methods that perform statistical calculations on two-dimensional arrays of integers.
	6.1.2	**Matrix Objects** Students explore a `_Demo` program that represents matrix objects using two-dimensional arrays of integers as instance information, then design implement and test methods within a `class` that creates square matrix objects and performs matrix calculations.
2		**Two-Dimensional Arrays of Strings** (2 lessons) – Students investigate examples of two-dimensional arrays of Strings, then design and test selection and transversal operations on two-dimensional arrays of Strings. **[CR2a] [CR2b]**
	6.2.1	**Seating Chart** – Students design implement and test a `class` that simulates a classroom seating chart of student names with a two-dimensional array of Strings. Students are again challenged to tackle a complex programming task by breaking it into multiple methods allowing early compilation and testing.
3	6.2.2	**Flags R Fun** – In the `_Assignment` project students complete a `class` containing a two-dimensional instance array of Strings by providing the implementation for five `private` methods that make and store specified flag patterns and creating an `equals` method for object content comparison.

Chapter 7: The `ArrayList` and Sorting

Students are introduced to java ArrayLists, compare them directly to familiar arrays, and use them to simulate situations requiring a dynamically sized, linear array data structure. `ArrayList` methods included in the AP subset are examined and used. The insertion and selection sort algorithms are explore and utilized on both arrays and ArrayLists.

Week	Lesson	Lesson Summary
4	7.1	**ArrayLists of Objects**– Students explore a `_Example` project that demonstrates explicit comparisons of arrays to ArrayLists. In the `_Assignment` project students will write methods receiving ArrayLists of objects as parameters and use traversals, insertions and deletions to manipulate ArrayLists contents simulating a database of `Mathlete` objects. **[CR3]**
	7.2	**ArrayLists of Wrapped Primitives** – Students will write methods receiving ArrayLists of wrapper objects as parameters and traverse the ArrayLists with enhanced `for` loops so as to obtain certain sample statistics. **[CR2b]**
5	7.3	**Box of Chocolates** – The `_Example` project contains examples of pre and post-conditions for method documentation and the use of assertions for de-bugging. Students complete four methods in a `class` that simulates filling a box of `Chocolate` objects subject to specific constraints. **[CR4]**
6	7.4	**Sorting Activity** – Students will create their own sorting algorithms in pseudo-code and then sort physical objects using pseudo-code selection and insertion sort implementations. **[CR3]**
	7.5	**Sorting ArrayLists**– Students will construct implementations of selection and insertion sorts for an instance `ArrayList` of `Coin` objects within a `PiggyBank` class. **[CR2a]**
7	7.6	**Sorting Arrays** – Students will design, implement and test a `class` referenced by a provided GUI within a `_Assignment` project. The `class` requires implementations of selection and insertion sorts for a java array of randomly generated integers. **[CR2a]**

Chapter 8: Inheritance and Polymorphism

Students are introduced to code re-use concepts via inheritance through detailed examples of "is-a" class code with comments. Students then complete programming projects based on simulations requiring "is-a" inheritance.

Week	Lesson	Lesson Summary
7	8.1	**"is-a" Relationships** – Students examine a detailed example of an "is-a" relationship in a `_Example` project then implement a personally assigned "is-a" relationship with a superclass and subclass including a required number of unique and overridden methods. **[CR2a]**
8	8.2	**Implementing Inheritance** – By investigating two `_Example` projects students experience the `Object class`, instance variable encapsulation between sub/super classes and the effect of calling overridden methods when a sub-class is referenced as a super-class. Students then design, implement and test a sub-class within an inheritance chain as part of a `_Assignment` project. **[CR1]**
	8.3	**Polymorphism** – Students formalize the concept of polymorphism then design, implement and test an "is-a" class within a `_Assignment` project simulation of a delivery company. **[CR1]**
9	8.4	**Abstract Classes** – Students revisit their Lesson 8.1 `Is_A_Assignment` and make their `super class abstract` by adding a prescribed number of `abstract` methods. Then they must create two more additional subclasses with a required number of unique and overridden methods. **[CR1]**
	8.5	**Interfaces** – Students explore a polymorphic example of an interface within a `_Example` project with a `class` that stores and utilizes a data structure holding objects from different inheritance chains. Students then revisit the delivery company simulation and create a `class` that also utilizes a data structure holding objects from different inheritance chains. **[CR1]**

Chapter 9: Recursion

Students compare recursive implementation with iteration by `for` and `while` loops while exploring classic algorithms such as factorials and the Fibonacci sequence. Students then implement recursive designs for binary search and mergesort `_Assignment` projects.

Week	Lesson	Lesson Summary
10	9.1	**Recursive Methods** – Students are introduced to recursive implementations with a few classic examples shown in the `Factorial_Fibonacci_Examples.jar` project. Students then create a `class` with a `_Assignment` project that works with a provided GUI to reverse the order of characters within a user supplied `String`. **[CR3] [CR2a]**
	9.2	**Stack Overflow** – Students encounter and are required to fix a stack overflow error in the `_Example` project. Students then create a `class` with a `_Assignment` project that works with a provided GUI that finds the number of unique permutations, or arrangements, that can be made from (k) unique objects taken from a set of (n) unique objects. **[CR2a]**
11		**Recursive Applications** (2 lessons) – Students explore two commonly used recursive algorithms. **[CR3]**
	9.3.1	**Mergesort** – Students explore the recursive mergesort algorithm with an array of integers in a `_Example` project and an animated app, then identify the use of the merge sort among several other algorithms. Finally students design and implement a mergesort method(s) of an `ArrayList` in a `_Assignment` project.
	9.3.2	**Binary Search** – Students discover the recursive binary search algorithm in a `_Example` project then experience performance differences between a linear search and a binary search for a large `ArrayList` of objects of type `Student` in a `_Demo` project. Students then implement a recursive binary search method in a `_Assignment` project.

Chapter 10: Additional Projects and Review

This section is intended to serve as review material before the AP Computer Science A exam. Time permitting, AP Computer Science A labs can be added along with practice free-response exam questions.

Week	Lesson	Lesson Summary
12	10.1	**Craps** – Students design, implement and test a `class` in a `_Assignment` project that works with a given GUI and `Die` classes to simulate a game of craps. **[CR3][CR4] [CR7]**
	10.2	**StuffMart Parking Lot V2** – Students revisit the StuffMart parking lot simulation from Lesson 5.3 and add a variety of summary statistics and a histogram display to an enhanced simulation of the parking lot project. **[CR3][[CR4]**
13	10.3	**Tic Tac Toe** – Students design implement and test a `class` referenced within the `_Assignment` project that uses two two-dimensional arrays and a GUI to run a simulation of the game TicTacToe. **[CR1]**
	10.4	**Recursive Rectangles** – Students begin the lesson with a variety of recursive multiple choice questions then design, implement and test a `class` that works with a terminal based UI to create recursive `ASCII String` representations of rectangles. **[CR1]**
14 +	Review	Released APCS Exam Questions Multiple Choice, Free Response APCS A Labs Elevens Lab, Magpie Lab, Picture Lab Problem Generator Review Assignment Code Traces, Multiple Choice

AP Computer Science A Java Curricular Requirements [CR_]

Curricular Requirements	
CR1	The course teaches students to design and implement computer based solutions to problems.
CR2a	The course teaches students to use and implement commonly used algorithms.
CR2b	The course teaches students to use commonly used data structures.
CR3	The course teaches students to select appropriate algorithms and data structures to solve problems.
CR4	The course teaches students to code fluently in an object-oriented paradigm using the programming language Java.
CR5	The course teaches students to use elements of the standard Java library from the AP Java subset in Appendix A of the AP Computer Science A Course Description.
CR6	The course includes a structured lab component comprised of a minimum of 20 hours of hands-on lab experiences.
CR7	The course teaches students to recognize the ethical and social implications of computer use.