Teaching Tomorrow's Technology.
CompuScholar

Java Programming for AP Computer Science A

Course Syllabus and Planner

Course Overview

This AP Computer Science A class uses the **CompuScholar Java Programming [1]** curriculum as the primary resource. It is taught as a one-year (two-semester) sequence and covers all required topics in the "Computer Science A" Course Description published by the College Board.

Students need to have typical computer usage skills before starting this course; other introductory programming courses are not required. All required concepts are taught from the ground up in a fun, step-by-step manner. The course uses a variety of multimedia content such as full-color, interactive text, narrated instructional videos, and guided classroom discussions. Strong emphasis is placed on hands-on programming labs to demonstrate mastery of lesson concepts.

The **CompuScholar Java Programming** curriculum is **fully aligned with the AP CS A recommended Unit Sequence**. This allows teachers to easily leverage the additional material and practice questions in the AP Classroom.

Textbook Resource

This course uses the following instructional resource:

[1] **Java Programming** online text, by CompuScholar, Inc. 2023, ISBN 978-1-946113-99-3

https://www.compuscholar.com/schools/courses/java/

Course Material

The course material is designed to appeal to a variety of students, from traditional learners who thrive on written text to audio-visual students who enjoy a multi-media format. All content is delivered through an online system that allows students to work seamlessly both in the classroom and at home.

The course consists of the following student-facing elements:

- **Instructional Videos** – optional (not required) but enjoyed by many students as an audio-visual introduction and reinforcement of the lesson topics.
- **Lesson Text** – required reading, contains full topic details and live coding exercises
- **Quizzes and Exams** – multiple-choice and automatically graded by our system
- **Chapter Activities** – hands-on projects, submitted for a grade

Teachers additionally have access to:

- **Teacher's Guides** – for each lesson, with suggested classroom discussion questions
- **Quiz and Exam Answer Keys** – PDFs for quick reference
- **Activity Solution Guides** – fully coded activity solutions for each chapter activity

Programming Environment and Device Requirements

CompuScholar provides an in-browser Java coding environment. This online feature may be used by students to complete all exercises and activities in all required AP chapters. When using the online coding environment:

- **No local software installation is needed to prepare for the AP exam.**
- **The AP material can be completed from any web browser on any device (including Chromebooks and tablets).**

Later, optional chapters contain a mixture of activities. AP teachers may select any of these topics for students after the AP exam. Some optional activities can be done in CompuScholar's online environment, while others are completed using an external IDE.

When needed, CompuScholar recommends a locally installed JDK and Eclipse platform for an external IDE (see Chapters 28 and 29 for instructions). Teachers may also select any other locally installed or online IDE. **Device requirements for an optional, external IDE depend on the IDE selected.**

Project Grading

Each chapter normally contains one or more hands-on, graded activities. The activities in **all required AP chapters are fully auto graded by the CompuScholar system**. Teachers have complete control over the auto-graded results.

Some activities in later, optional chapters are free-form (creative) or completed in an external IDE. The teacher is responsible for grading those creative or external projects.

Course Outline

The course includes the required content organized into the following units based on the AP Course and Exam Description (CED):

- **CED Unit 1: Primitive Types** – Course Chapters 2, 3, 4
- **CED Unit 2: Using Objects** - Course Chapters 5, 6, 7
- **CED Unit 3: Boolean Expressions and if Statements** - Course Chapters 8, 9, 10
- **CED Unit 4: Iteration** – Course Chapters 11, 12, 13
- **CED Unit 5: Writing Classes** – Course Chapters 14, 15, 16
- **CED Unit 6: Array** – Course Chapter 18
- **CED Unit 7: ArrayList** – Course Chapters 19, 20
- **CED Unit 8: 2D Array** – Course Chapter 21
- **CED Unit 9: Inheritance** – Course Chapters 22, 23
- **CED Unit 10: Recursion** – Course Chapter 24

Course Navigation

Chapter 1 contains computing, ethics, and security topics recommended (but not tested) by AP CS A and required by many state and national computer science curriculum standards. AP teachers may opt to complete Chapter 1 in sequence or return to the topics after the AP CS A exam.

Chapters 2 – 24 should be completed in sequence and cover all required topics on the AP CS A exam, plus certain other highly recommended software skills. These chapters include substantial, hands-on lab work over the 20-hour minimum AP requirement. The mid-term project in Chapter 17 may be omitted or postponed if desired (no new skills).

Typical classes will finish all required AP content before the exam administration in May. We recommend using the remaining time before the exam to review the College Board's published practice exams and any other external source of practice FRQ and multiple-choice problems.

Chapters 25 – 33 contain optional topics that are not required for AP CS A. Teachers may review and select any of these optional topics for students as time permits after the AP CS A exam. Some optional chapters require the use of an external IDE and/or will be graded by the teacher.

Supplemental Chapters 1 – 4 contain a variety of enrichment topics that may be required by individual states to satisfy requirements for other coding or digital literacy courses. AP teachers may select any of these topics (especially Supplemental Chapter 3) to complete "Big Ideas" like "Impact of Computing" that are not tested on the exam.

Course Planner

The following pages contain an outline of course content by AP CS A unit title and matching textbook chapter. Correlations to the AP CS A **recommended Unit Sequence** are highlighted. Additional, detailed mappings to AP CS A Learning Objectives and Essential Knowledge (LOEKs) are appended at the end of this document.

A typical school year consists of 36 calendar weeks or 180 days of school. After completing the first 24 chapters, most classes will have several weeks left for AP exam prep, make-up work, and optional topics. Teachers can select from optional topics before and after the exam, as time permits.

Each "day" listed below represents one typical day or class period of 45 – 60 minutes. In most cases, students will complete one lesson per day (including the quiz), 1 day per lab, and 1 day per chapter test. Some classes may move faster or slower than the suggested pace.

Semester 1 Timeline

Days	CompuScholar Chapter and Lab	AP CS A Unit Sequence
6	**Chapter 1: Computing Concepts** * Evolution of Computers * Computer Hardware * Computer Software * Computer Ethics * Computer Security	**General curricular requirements (N/A on AP Exam)** Schedule as time permits

Days	CompuScholar Chapter and Lab	AP CS A UNIT 1: Primitive Types
6	**Chapter 2: Getting Started with Java** * Common Programming Languages * The Java Platform * Writing Your First Program * Help and Reference Documentation **LAB: Shopping List**	1.1 Why Programming? Why Java?
5	**Chapter 3: Data Types and Variables** * Primitive Data Types * Variables * Printing Data **LAB: Treasure Map (Big Idea: VAR)**	1.2 Variables and Data Types 1.3 Expressions and Assignment Statements

Days	CompuScholar Chapter and Lab	AP CS A UNIT 1: Primitive Types
5	**Chapter 4: Working with Numbers** * Simple Math Operations * Compound Assignments and Shortcuts * Type Casting and Truncation **LAB: Magic Math (Big Idea: VAR)**	1.4 Compound Assignment Operators (shortcuts) 1.5 Casting and Ranges of Variables

Days	CompuScholar Chapter and Lab	AP CS A: UNIT 2: Using Objects
8	**Chapter 5: Introducing Objects** * Java Classes * Reference Variables and Strings * Calling Methods * Properties and Constructors * Static and Overloaded Methods * User Input with Scanner **LAB: Sketch Robot (Big Idea: VAR, MOD)**	2.1 Objects: Instances of Classes 2.2 Creating and Storing Objects (Instantiation) 2.3 Calling a Void Method 2.4 Calling a Void Method with Parameters 2.5 Calling a Non-void Method
5	**Chapter 6: Working with Strings** * Comparing Strings * Common String Operations * Formatting and Building Strings **LAB: String Theory (Big Idea: VAR, MOD)**	2.6 String Objects: Concatenation, Literals, and More 2.7 String Methods
6	**Chapter 7: Numbering Systems and Java Math** * Java Wrapper Classes & Numeric Conversion * Numbers in Binary, Octal and Hex * Java Math Class * Numeric Limitations **LAB: Math Factory (Big Idea: VAR, MOD)**	2.8 Wrapper Classes: Integer and Double 2.9 Using the Math Class

Days	CompuScholar Chapter and Lab	AP CS A: UNIT 3: Boolean Expressions and if Statements
6	**Chapter 8: Logic and Decision-Making** * Logical Expressions and Relational Operators * Making Decisions with if() * Using "else-if" and "else" * The "switch" Statement **LAB: Banking System (Big Idea: CON)**	3.1 Boolean Expressions 3.2 if Statements and Control Flow 3.3 if-else Statements 3.4 else if Statements
5	**Chapter 9: More Complex Logic** * Comparing Objects and References * Compound Expressions * Boolean Algebra and Truth Tables **LAB: Wild Card (Big Idea: CON)**	3.5 Compound Boolean Expressions 3.6 Equivalent Boolean Expressions 3.7 Comparing Objects
5	**Chapter 10: Handling Exceptions** * Understanding Exceptions * Catching Exceptions * Validating User Input **LAB: Calculator Madness (Big Idea: CON)**	**Highly recommended skills as students begin to produce more complex code.**
4	**Chapter 11: Debugging** * Finding Runtime Errors * Debugger Concepts **LAB: Bug Hunt (Big Idea: CON)**	**Highly recommended skills as students begin to produce more complex code.**

Days	CompuScholar Chapter and Lab	AP CS A: UNIT 4: Iteration
6	**Chapter 12: Iteration** * For Loops * While Loops * Continue, Break and Return * Nested Loops **LAB: Fun Factorials (Big Idea: CON)**	4.1 while Loops 4.2 for Loops
6	**Chapter 13: Algorithms** * Designing with Flowcharts * Writing Pseudocode * Common Mathematical Algorithms * Common String Algorithms **LAB: Meal Time (Big Idea: CON)**	4.3 Developing Algorithms Using Strings 4.4 Nested Iteration 4.5 Informal Code Analysis

Days	CompuScholar Chapter and Lab	AP CS A: UNIT 5: Writing Classes
7	**Chapter 14: Creating Java Classes** * Object-Oriented Concepts * Defining Classes and Packages * Class Properties * Constructors * Class Methods **LAB: Dog House (Big Idea: MOD)**	5.1 Anatomy of a Class 5.2 Constructors 5.3 Documentation with Comments 5.4 Accessor Methods 5.5 Mutator Methods
7	**Chapter 15: Working with Methods** * Documentation and Design * Variable Scope and Access * Data Encapsulation * Method Overloading * Object Interfaces **LAB: Let's Go Racing! (Big Idea: MOD)**	5.6 Writing Methods 5.8 Scope and Access
5	**Chapter 16: Static Concepts** * Static Properties * Static Methods * Static, Object, and "this" References **LAB: Art School (Big Idea: MOD)**	5.7 Static Variables and Methods 5.9 this Keyword 5.10 Ethical and Social Implications of Computing Systems (Not on AP Exam) - See Chapter 1

Days	CompuScholar Chapter and Lab	Mid-Term Project
4	**Chapter 17: Mid-Term Project** * Introducing the "Remote Control" Project **LAB: Creating the Schedule** **LAB: Building a Television** **LAB: Defining the Remote** **(Big Ideas: VAR, MOD, CON)**	Schedule as time permits (no new skills)
85 - 95	**Total Days (depending on the scheduling of Chapters 1 and 17)**	

Semester 2 Timeline

Days	CompuScholar Chapter and Lab	AP CS A UNIT 6: Array
7	**Chapter 18: 1D Arrays** * Array Concepts * Array Traversal * Iterators and Enhanced for() loops * Array Algorithms * More Array Algorithms **LAB: Whack-A-Mole (Big Ideas: VAR, CON)**	6.1 Array Creation and Access 6.2 Traversing Arrays 6.3 Enhanced for Loop for Arrays 6.4 Developing Algorithms Using Arrays

Days	CompuScholar Chapter and Lab	AP CS A UNIT 7: ArrayList
6	**Chapter 19: Lists and ArrayLists** * Java Lists * ArrayLists * Iterators and Enhanced for() Loops * Algorithms with ArrayLists **LAB: Train Yard Jumble (Big Ideas: VAR, CON)**	7.1 Introduction to ArrayList 7.2 ArrayList Methods 7.3 Traversing ArrayLists
7	**Chapter 20: Searching and Sorting** * Bubble Sort * Selection Sort * Insertion Sort * Sequential and Binary Searches **LAB: Ducks in a Row (Big Ideas: VAR, CON)**	7.4 Developing Algorithms Using ArrayLists 7.5 Searching 7.6 Sorting 7.7 Ethical Issues Around Data Collection (see Chapter 1)

Days	CompuScholar Chapter and Lab	AP CS A UNIT 8: 2D Array
6	**Chapter 21: 2D Arrays** * 2D Arrays * Traversal and Ordering * Array of Arrays * 2D Array Algorithms **LAB: Gold Rush (Big Ideas: VAR, CON, MOD)**	8.1 2D Arrays 8.2 Traversing 2D Arrays

Days	CompuScholar Chapter and Lab	AP CS A UNIT 9: Inheritance
5	**Chapter 22: Inheritance** * Superclass and Subclass Concepts * Subclass Constructors * Using Superclass and Subclass References **LAB: Lab Rats (Big Idea: MOD)**	9.1 Creating Superclasses and Subclasses 9.2 Writing Constructors for Subclasses
6	**Chapter 23: Polymorphism** * Overriding Superclass Methods * Abstract Classes and Methods * Using Superclass Features with "super" * The "Object" Superclass **LAB: Social Ladder (Big Idea: MOD)**	9.3 Overriding Methods 9.4 super Keyword 9.5 Creating References Using Inheritance Hierarchies 9.6 Polymorphism 9.7 Object Superclass

Days	CompuScholar Chapter and Lab	AP CS A UNIT 10: Recursion
5	**Chapter 24: Recursion** * Recursion * Recursive Binary Search * Merge Sort **LAB: File Explorer (Big Idea: CON)**	10.1 Recursion 10.2 Recursive Searching and Sorting
42	**Total Days in Semester 2 (all required AP CS A topics complete at this point)**	

Classes who complete the first 24 chapters at this point have spent approximately 134 days and covered all tested AP CS A topics. **The remaining class time should be spent in preparation for the AP exam and covering other required topics and big ideas (digital citizenship, legal and ethical impacts of computing, etc.) found in Chapters 1, 27, Supplemental Chapter 3, and any other teacher-selected chapters and lessons.**

Please see below for information on the **additional chapters and Supplemental topics**.

The following table suggests the timeline needed for each **additional or supplemental chapter**, along with notes as to the programming environment and grading approach. There are more chapters available than students can complete in a single year, so teachers can pick topics as time permits!

Days	CompuScholar Chapter and Lab	Notes
5	**Chapter 25: File Access** * Data Streams * Reading and Writing Text Data * Reading and Writing Binary Data **LAB: Address CSV**	CompuScholar online environment, project auto-graded by our system
5	**Chapter 26: Object Composition and Copying** * Functional Decomposition * Composite Classes * Copying Objects **LAB: Designing a Composite Class**	Teacher-graded project
10-15	**Chapter 27: Team Project** * Design Processes and Teamwork * Requirements and Design Documents **LAB: Team Project Requirements** **LAB: Project Design** **LAB: Team Project Implementation** *** Testing Your Code** **LAB: Team Project Testing** **(Big Idea: IOC)**	CompuScholar online environment or external IDE, teacher-graded project
3	**Chapter 28: Running Java Locally** * Installing the JDK * Local Source Code * Building and Running from the Command Line	"How-to" chapter to create a local development environment
4	**Chapter 29: The Eclipse IDE** * Introducing Eclipse * Eclipse Java IDE Walk-Through * Creating an Eclipse Project * The Eclipse Debugger	"How-to" chapter to install and use a local IDE

Days	CompuScholar Chapter and Lab	Notes
6	**Chapter 30: Graphical Java Programs** * Java Swing * Creating a Simple Window * Event-Driven Programming * Layout Managers **LAB: Phone Dialer**	Requires external IDE (e.g., Eclipse) with Java Swing support. Teacher-graded projects.
5	**Chapter 31: Swing Input Controls** * Text and Numeric Input * List Input * Option Input **LAB: Pizza Place**	Requires external IDE (e.g., Eclipse) with Java Swing support. Teacher-graded projects.
5	**Chapter 32: Vector and Bitmap Images** * Screen Coordinates * Drawing Shapes * Drawing Images **LAB: Sky Art**	Requires external IDE (e.g., Eclipse) with Java Swing support. Teacher-graded projects.
4	**Chapter 33: Program Efficiency** * Algorithm Performance (Big-O) * Measuring Sorting Efficiency **LAB: Comparison of Sorting Algorithms**	External IDE, teacher-graded project
12	**Supplemental Chapter 1: Enrichment Topics** * Encoding Data **Lab: Secret Message** * JavaDoc **Lab: Creating HTML** * Advanced Algorithms * Stock Market Simulation **Lab: Stock Trading (Big Idea: IOC)** * Stacks and Queues **Lab: Candy Factory (Big Idea: VAR)** * Exploring UML **Lab: UML Design (Big Idea: MOD)**	See individual lessons and activities for the programming environment and grading approach.

Days	CompuScholar Chapter and Lab	Notes
8	**Supplemental Chapter 2: Software and Industry** * Software Development Process **Lab: Your SDLC Docs** * Software Development Careers **Lab: Career Exploration** * Student Organizations **Lab: CTSO Exploration** * Technical Writing **Lab: Technical Writing**	Offline work, teacher-graded projects
4	**Supplemental Chapter 3: Computers and Modern Society** * Managing your Digital Identity **Lab: Privacy Check-up (Big Idea: IOC)** * The Impact of Computing **Lab: Impact Analysis (Big Idea: IOC)** * Artificial Intelligence **Lab: Exploring Self-Driving Cars (Big Idea: IOC)** * Productivity Tools **Lab: Productivity Report**	Offline work, teacher-graded projects
6	**Supplemental Chapter 4: Computer Networking** * Basic Networking * Network Topology * Network Addressing * Network Design * Network Protocols **Lab: Animal Palace**	Offline work, teacher-graded project

"Big Ideas" and Student Activities

The hands-on labs and student activities found within each chapter reinforce one or more "Big Ideas" (**MOD**, **VAR**, **CON**, **IOC**) from the AP Course and Exam Description (CED). The outline above lists the corresponding Big Idea(s) relevant to each lab. Those labs are categorized below by Big Idea, for convenience.

Big Idea 1: MODULARITY (MOD)	
Chapter 5 LAB: Sketch Robot	Chapter 17 LAB: Remote Control
Chapter 6 LAB: String Theory	Chapter 21 LAB: Gold Rush
Chapter 7 LAB: Math Factory	Chapter 22 LAB: Lab Rats
Chapter 14 LAB: Dog House	Chapter 23 LAB: Social Ladder
Chapter 15 LAB: Let's Go Racing	Supplemental Chapter 1 LAB: UML Design
Chapter 16 LAB: Art School	

Big Idea 2: VARIABLES (VAR)	
Chapter 5 LAB: Sketch Robot	Chapter 17 LAB: Remote Control
Chapter 6 LAB: String Theory	Chapter 21 LAB: Gold Rush
Chapter 7 LAB: Math Factory	Chapter 22 LAB: Lab Rats
Chapter 14 LAB: Dog House	Chapter 23 LAB: Social Ladder
Chapter 15 LAB: Let's Go Racing	Supplemental Chapter 1 LAB: UML Design
Chapter 16 LAB: Art School	

Big Idea 3: CONTROL (CON)	
Chapter 8 LAB: Banking System	Chapter 17 LAB: Remote Control
Chapter 9 LAB: Wild Card	Chapter 18 LAB: Whack-A-Mole
Chapter 10 LAB: Calculator Madness	Chapter 19 LAB: Train Yard Jumble
Chapter 11 LAB: Bug Hunt	Chapter 20 LAB: Ducks in a Row
Chapter 12 LAB: Fun Factorials	Chapter 21 LAB: Gold Rush
Chapter 13 LAB: Meal Time	Chapter 24 LAB: File Explorer

Big Idea 4: IMPACT OF COMPUTING (IOC)
Chapter 27 LAB: Team Project
Supplemental Chapter 1 LAB: Stock Market Trading
Supplemental Chapter 3 LAB: Privacy Check-up
Supplemental Chapter 3 LAB: Impact Analysis
Supplemental Chapter 3 LAB: Exploring Self-Driving Cars

Computational Thinking Practices

The AP Course and Exam Description (CED) lists five "Computational Thinking" practices with itemized skills under each practice. The **Java Programming[1]** course provides hands-on opportunities for students to practice these skills in the form of **in-lesson exercises** and/or **chapter labs**.

All Computational Thinking skills are reinforced multiple times within the course. The table below describes one instance of an in-lesson exercise or chapter lab that supports each skill.

Practice 1: Program Design and Algorithm Development	**Chapter and Lab**
1.A Determine an appropriate program design to solve a problem or accomplish a task (not assessed).	**Chapter 13 LAB: Meal Time** – Students will design and implement a program to simulate a specific algorithm.
1.B Determine code that would be used to complete code segments.	**Chapter 6 LAB: String Theory** – Students will complete the provided starting code to manipulate String data in specific ways.
1.C Determine code that would be used to interact with completed program code.	**Chapter 5 LAB: Sketch Robot** – Students will write the main program logic to interact with a provided SketchRobot class.
Practice 2: Code Logic	**Chapter and Lab**
2.A Apply the meaning of specific operators.	**Chapter 9 LAB: Wild Card** – Students will apply multiple relational and conditional operators to implement the rules for determining the wild cards in a game.
2.B Determine the result or output based on statement execution order in a code segment without method calls (other than output).	**Chapter 7 LAB: Math Factory** – Students are giving starting code and expected output and must complete the sequential logic to match the output.
2.C Determine the result or output based on the statement execution order in a code segment containing method calls.	**Chapter 14, Lesson 5 Exercises** - Students are challenged to predict the output from given code segments containing multiple method calls.
2.D Determine the number of times a code segment will execute.	**Chapter 12 LAB: Fun Factorials** – Students will apply loops to produce factorial calculations, taking care to iterate the correct number of times to produce the desired result.

Practice 3: Code Implementation	Chapter and Lab
3.A Write program code to create objects of a class and call methods.	**Chapter 15 LAB: Let's Go Racing** – Students will create Dragster, Cheetah, and UFO classes, each with properties and methods to participate in a race.
3.B Write program code to define a new type by creating a class.	**Chapter 14 LAB: Dog House** – Students will create a new Dog class from scratch to work with an existing House class.
3.C Write program code to satisfy method specifications using expressions, conditional statements, and iterative statements.	**Chapter 20 LAB: Ducks in a Row** – Students will write methods that include expressions, conditionals, and iterative loops to complete a sorting and searching program.
3.D Write program code to create, traverse, and manipulate elements in 1D array or ArrayList objects.	**Chapter 18 LAB: Whack-A-Mole** – Students create, traverse, and manipulate "Mole" objects in a 1D array as part of a game.
3.E Write program code to create, traverse, and manipulate elements in 2D array objects.	**Chapter 20 LAB: Gold Rush** – Students will simulate the survey of a land grid for amounts of gold by creating, traversing, and manipulating 2D arrays.
Practice 4: Code Testing	**Chapter and Lab**
4.A Use test-cases to find errors or validate results.	Every lab in Chapters 2 through 24 includes an itemized list of test cases that students will use to validate their code against expected results. Additionally, **Chapter 27 LAB 3: Project Testing**, asks students to create a custom test plan document and iteratively run test cases until all requirements are met.
4.B Identify errors in program code.	**Chapter 11 LAB: Bug Hunt** – Students will identify and resolve multiple errors in a starting program to convert temperatures between Fahrenheit and Celsius.
4.C Determine if two or more code segments yield equivalent results.	**Chapter 9, Lesson 3 Exercises** - Students will use truth tables to compare code segments and logical expressions for equivalency.

Practice 5: Documentation	Chapter and Lab
5.A Describe the behavior of a given segment of program code.	**Chapter 8, Lesson 2 Exercises** – Students are given starting code that includes if() conditions and asked to describe the expected behavior and output.
5.B Explain why a code segment will not compile or work as intended.	**Chapter 4, Lesson 3 Exercises** – Students examine type casting and truncation in the context of mathematical expressions with mixed types that produce unexpected results.
5.C Explain how the result of program code changes, given a change to the initial code.	**Chapter 12, Lesson 1 Exercises** – Students examine example for() loops and predict how output will differ when changes are made to loop expressions or body statements.
5.D Describe the initial conditions that must be met for a program segment to work as intended or described.	**Chapter 10, Lesson 3 Exercises** – Students examine input-handling logic and identify the correct type of input that must be received for that logic to work as expected.

The following pages contain detailed cross-reference tables that map every AP Computer Science A topic and essential knowledge to specific course chapters and lessons. For convenience, these cross-references are also available as a separate document at the following online location:

https://www.compuscholar.com/docs/java/AP_Exam_Cross_Reference.pdf

CompuScholar, Inc.

Alignment to the College Board AP **Computer Science A**

Learning Objectives and Essential Knowledge (LOEK)

AP Course Details:

Course Title:	AP Computer Science A
Grade Level:	9th - 12th grades
Standards Version:	Fall 2020
Standards Link:	ap-computer-science-a-course-and-exam-description.pdf

CompuScholar Course Details:

Course Title:	Java Programming
Course ISBN:	978-1-946113-99-3
Course Year:	2023

Note 1: Citation(s) listed may represent a subset of the instances where objectives are met throughout the course.

Note 2: Citation(s) for a "Lesson" refer to the "Lesson Text" elements and associated "Activities" within the course, unless otherwise noted. The "Instructional Video" components are supplements designed to introduce or reinforce the main lesson concepts, and the Lesson Text contains full details.

AP Course Description

This course teaches students the fundamentals of the Java programming language and covers all required topics defined by the College Board's AP Computer Science A course description.

AP Lab Requirements

The AP Computer Science A course must include a minimum of 20 hours of hands-on structured-lab experiences to engage students in individual or group problem solving.	**This course easily exceeds the 20-hour minimum lab requirement with hands-on lesson exercises and labs in every chapter.**

AP Topics and Essential Knowledge

UNIT 1: Primitive Types	**CITATION(S)**
Topic 1.1: Why Programming? Why Java?	
MOD-1.A.1 - System.out.print and System.out.println display information on the computer monitor.	Chapter 3, Lesson 3
MOD-1.A.2 - System.out.println moves the cursor to a new line after the information has been displayed, while System.out.print does not.	Chapter 3, Lesson 3

VAR-1.A.1 - A string literal is enclosed in double quotes.	Chapter 2, Lesson 3
TOPIC 1.2: Variables and Data Types	
VAR-1.B.1 - A type is a set of values (a domain) and a set of operations on them.	Chapter 3, Lesson 1
VAR-1.B.2 - Data types can be categorized as either primitive or reference.	Chapter 3, Lesson 1
VAR-1.B.3 - The primitive data types used in this course define the set of operations for numbers and Boolean values.	Chapter 3, Lesson 1
VAR-1.C.1 - The three primitive data types used in this course are int, double, and boolean.	Chapter 3, Lesson 1
VAR-1.C.2 - Each variable has associated memory that is used to hold its value.	Chapter 3, Lessons 1, 2
VAR-1.C.3 - The memory associated with a variable of a primitive type holds an actual primitive value.	Chapter 3, Lessons 1, 2
VAR-1.C.4 - When a variable is declared final, its value cannot be changed once it is initialized.	Chapter 3, Lesson 2
TOPIC 1.3: Expressions and Assignment Statements	
CON-1.A.1 - A literal is the source code representation of a fixed value.	Chapter 4, Lesson 1
CON-1.A.2 - Arithmetic expressions include expressions of type int and double.	Chapter 4, Lesson 1
CON-1.A.3 - The arithmetic operators consist of +, −, *, /, and %	Chapter 4, Lesson 1
CON-1.A.4 - An arithmetic operation that uses two int values will evaluate to an int value.	Chapter 4, Lesson 1
CON-1.A.5 - An arithmetic operation that uses a double value will evaluate to a double value.	Chapter 4, Lesson 1
CON-1.A.6 - Operators can be used to construct compound expressions.	Chapter 4, Lesson 2
CON-1.A.7 - During evaluation, operands are associated with operators according to operator precedence to determine how they are grouped.	Chapter 4, Lesson 1
CON-1.A.8 - An attempt to divide an integer by zero will result in an ArithmeticException to occur.	Chapter 4, Lesson 1
CON-1.B.1 - The assignment operator (=) allows a program to initialize or change the value stored in a variable. The value of the expression on the right is stored in the variable on the left.	Chapter 4, Lesson 1
CON-1.B.2 - During execution, expressions are evaluated to produce a single value.	Chapter 4, Lesson 1
CON-1.B.3 - The value of an expression has a type based on the evaluation of the expression.	Chapter 4, Lesson 1
TOPIC 1.4: Compound Assignment Operators	
CON-1.B.4 - Compound assignment operators (+=, −=, *=, /=, %=) can be used in place of the assignment operator.	Chapter 4, Lesson 2

CON-1.B.5 - The increment operator (++) and decrement operator (−−) are used to add 1 or subtract 1 from the stored value of a variable or an array element. The new value is assigned to the variable or array element.	Chapter 4, Lesson 2
TOPIC 1.5: Casting and Ranges of Variables	
CON-1.C.1 - The casting operators (int) and (double) can be used to create a temporary value converted to a different data type.	Chapter 4, Lesson 3
CON-1.C.2 - Casting a double value to an int causes the digits to the right of the decimal point to be truncated.	Chapter 4, Lesson 3
CON-1.C.3 - Some programming code causes int values to be automatically cast (widened) to double values.	Chapter 4, Lesson 3
CON-1.C.4 - Values of type double can be rounded to the nearest integer by (int)(x + 0.5) or (int)(x – 0.5) for negative numbers.	Chapter 4, Lesson 3
CON-1.C.5 - Integer values in Java are represented by values of type int, which are stored using a finite amount (4 bytes) of memory. Therefore, an int value must be in the range from Integer.MIN_VALUE to Integer.MAX_VALUE inclusive.	Chapter 3, Lesson 1 Chapter 7, Lesson 4
CON-1.C.6 - If an expression would evaluate to an int value outside of the allowed range, an integer overflow occurs. This could result in an incorrect value within the allowed range.	Chapter 3, Lesson 1 Chapter 7, Lesson 4

UNIT 2: Using Objects	**CITATION(S)**
TOPIC 2.1: Objects: Instances of Classes	
MOD-1.B.1 - An object is a specific instance of a class with defined attributes.	Chapter 5, Lesson 1
MOD-1.B.2 A class is the formal implementation, or blueprint, of the attributes and behaviors of an object.	Chapter 5, Lesson 1
TOPIC 2.2: Creating and Storing Objects (Instantiation)	
MOD-1.C.1 - A signature consists of the constructor name and the parameter list.	Chapter 5, Lesson 3
MOD-1.C.2 - The parameter list, in the header of a constructor, lists the types of the values that are passed and their variable names. These are often referred to as formal parameters.	Chapter 5, Lesson 3
MOD-1.C.3 - A parameter is a value that is passed into a constructor. These are often referred to as actual parameters.	Chapter 5, Lesson 3
MOD-1.C.4 - Constructors are said to be overloaded when there are multiple constructors with the same name but a different signature.	Chapter 5, Lesson 3
MOD-1.C.5 - The actual parameters passed to a constructor must be compatible with the types identified in the formal parameter list.	Chapter 5, Lesson 3
MOD-1.C.6 - Parameters are passed using call by value. Call by value initializes the formal parameters with copies of the actual parameters.	Chapter 5, Lesson 3
MOD-1.D.1 - Every object is created using the keyword new followed by a call to one of the class's constructors.	Chapter 5, Lesson 3
MOD-1.D.2 - A class contains constructors that are invoked to create objects. They have the same name as the class.	Chapter 5, Lesson 3

MOD-1.D.3 - Existing classes and class libraries can be utilized as appropriate to create objects.	Chapter 5, Lessons 1,2
MOD-1.D.4 - Parameters allow values to be passed to the constructor to establish the initial state of the object.	Chapter 5, Lesson 3
VAR-1.D.1 - The keyword null is a special value used to indicate that a reference is not associated with any object.	Chapter 5, Lesson 2
VAR-1.D.2 - The memory associated with a variable of a reference type holds an object reference value or, if there is no object, null. This value is the memory address of the referenced object.	Chapter 5, Lesson 2
TOPIC 2.3: Calling a Void Method	
MOD-1.E.1 - An object's behavior refers to what the object can do (or what can be done to it) and is defined by methods.	Chapter 5, Lesson 4
MOD-1.E.2 - Procedural abstraction allows a programmer to use a method by knowing what the method does even if they do not know how the method was written.	Chapter 5, Lesson 4
MOD-1.E.3 - A method signature for a method without parameters consists of the method name and an empty parameter list.	Chapter 5, Lesson 4
MOD-1.E.4 - A method or constructor call interrupts the sequential execution of statements, causing the program to first execute the statements in the method or constructor before continuing. Once the last statement in the method or constructor has executed or a return statement is executed, flow of control is returned to the point immediately following where the method or constructor was called.	Chapter 5, Lesson 4
MOD-1.E.5 - Non-static methods are called through objects of the class.	Chapter 5, Lesson 4
MOD-1.E.6 - The dot operator is used along with the object name to call non-static methods.	Chapter 5, Lesson 4
MOD-1.E.7 - Void methods do not have return values and are therefore not called as part of an expression.	Chapter 5, Lesson 4
MOD-1.E.8 - Using a null reference to call a method or access an instance variable causes a NullPointerException to be thrown.	Chapter 5, Lesson 4
TOPIC 2.4: Calling a Void Method with Parameters	
MOD-1.F.1 - A method signature for a method with parameters consists of the method name and the ordered list of parameter types.	Chapter 5, Lesson 4
MOD-1.F.2 - Values provided in the parameter list need to correspond to the order and type in the method signature.	Chapter 5, Lesson 4
MOD-1.F.3 - Methods are said to be overloaded when there are multiple methods with the same name but a different signature.	Chapter 5, Lessons 3, 4
TOPIC 2.5: Calling a Non-void Method	
MOD-1.G.1 - Non-void methods return a value that is the same type as the return type in the signature. To use the return value when calling a non-void method, it must be stored in a variable or used as part of an expression.	Chapter 5, Lesson 4
TOPIC 2.6: String Objects: Concatenation, Literals, and More	
VAR-1.E.1 - String objects can be created by using string literals or by calling the String class constructor.	Chapter 6, Lesson 1

VAR-1.E.2 - String objects are immutable, meaning that String methods do not change the String object.	Chapter 6, Lesson 2
VAR-1.E.3 - String objects can be concatenated using the + or += operator, resulting in a new String object.	Chapter 5, Lesson 2 Chapter 6, Lesson 3
VAR-1.E.4 - Primitive values can be concatenated with a String object. This causes implicit conversion of the values to String objects.	Chapter 5, Lesson 2 Chapter 6, Lesson 3
VAR-1.E.5 - Escape sequences start with a \ and have a special meaning in Java. Escape sequences used in this course include \", \\, and \n.	Chapter 3, Lesson 3 Chapter 6, Lesson 3
TOPIC 2.7: String Methods	
VAR-1.E.6 - Application program interfaces (APIs) and libraries simplify complex programming tasks.	Chapter 5, Lessons 1, 2 Chapter 6, Lesson 2
VAR-1.E.7 - Documentation for APIs and libraries are essential to understanding the attributes and behaviors of an object of a class.	Chapter 5, Lesson 1 Chapter 6, Lesson 2
VAR-1.E.8 - Classes in the APIs and libraries are grouped into packages.	Chapter 5, Lessons 1, 5
VAR-1.E.9 - The String class is part of the java.lang package. Classes in the java.lang package are available by default.	Chapter 5, Lessons 1, 2
VAR-1.E.10 - A String object has index values from 0 to length– 1. Attempting to access indices outside this range will result in an IndexOutOfBoundsException.	Chapter 6, Lesson 2
VAR-1.E.11 - A String object can be concatenated with an object reference, which implicitly calls the referenced object's toString method.	Chapter 7, Lesson 1
VAR-1.E.12 - The following String methods and constructors—including what they do and when they are used—are part of the Java Quick Reference:	See Below
String(String str) — Constructs a new String object that represents the same sequence of characters as str	Chapter 5, Lesson 3
int length() — Returns the number of characters in a String object	Chapter 6, Lessons 1, 2
String substring(int from, int to) — Returns the substring beginning at index from and ending at index to - 1	Chapter 6, Lesson 2
String substring(int from)— Returns substring(from, length())	Chapter 6, Lesson 2
int indexOf(String str) — Returns the index of the first occurrence of str; returns -1 if not found	Chapter 6, Lesson 2
boolean equals(String other)— Returns true if this is equal to other; returns false otherwise	Chapter 6, Lesson 2
int compareTo(String other)— Returns a value < 0 if this is less than other; returns zero if this is equal to other; returns a value > 0 if this is greater than other	Chapter 6, Lesson 2
VAR-1.E.13 - A string identical to the single element substring at position index can be created by calling substring(index, index + 1).	Chapter 6, Lesson 2
TOPIC 2.8: Wrapper Classes: Integer and Double	
VAR-1.F.1 - The Integer class and Double class are part of the java.lang package.	Chapter 7, Lesson 1

VAR-1.F.2 - The following Integer methods and constructors — including what they do and when they are used—are part of the Java Quick Reference:	See below
Integer(int value) — Constructs a new Integer object that represents the specified int value	Chapter 7, Lesson 1
Integer.MIN_VALUE — The minimum value represented by an int or Integer	Chapter 7, Lesson 1
Integer.MAX_VALUE — The maximum value represented by an int or Integer	Chapter 7, Lesson 1
int intValue() — Returns the value of this Integer as an int	Chapter 7, Lesson 1
VAR-1.F.3 - The following Double methods and constructors — including what they do and when they are used—are part of the Java Quick Reference:	See below
Double(double value) —Constructs a new Double object that represents the specified double value	Chapter 7, Lesson 1
double doubleValue() — Returns the value of this Double as a double	Chapter 7, Lesson 1
VAR-1.F.4 - Autoboxing is the automatic conversion that the Java compiler makes between primitive types and their corresponding object wrapper classes. This includes converting an int to an Integer and a double to a Double.	Chapter 7, Lesson 1
VAR-1.F.5 - The Java compiler applies autoboxing when a primitive value is: * Passed as a parameter to a method that expects an object of the corresponding wrapper class. * Assigned to a variable of the corresponding wrapper class.	Chapter 7, Lesson 1
VAR-1.F.6 - Unboxing is the automatic conversion that the Java compiler makes from the wrapper class to the primitive type. This includes converting an Integer to an int and a Double to a double.	Chapter 7, Lesson 1
VAR-1.F.7 - The Java compiler applies unboxing when a wrapper class object is: * Passed as a parameter to a method that expects a value of the corresponding primitive type. * Assigned to a variable of the corresponding primitive type.	Chapter 7, Lesson 1
TOPIC 2.9: Using the Math Class	
MOD-1.H.1 - Static methods are called using the dot operator along with the class name unless they are defined in the enclosing class.	Chapter 7, Lesson 3
CON-1.D.1 - The Math class is part of the java.lang package.	Chapter 7, Lesson 3
CON-1.D.2 - The Math class contains only static methods.	Chapter 7, Lesson 3
CON-1.D.3 - The following static Math methods—including what they do and when they are used—are part of the Java Quick Reference:	See below
int abs(int x) — Returns the absolute value of an int value	Chapter 7, Lesson 3
double abs(double x) — Returns the absolute value of a double value	Chapter 7, Lesson 3

double pow(double base, double exponent) — Returns the value of the first parameter raised to the power of the second parameter	Chapter 7, Lesson 3
double sqrt(double x) — Returns the positive square root of a double value	Chapter 7, Lesson 3
double random() — Returns a double value greater than or equal to 0.0 and less than 1.0	Chapter 7, Lesson 3
CON-1.D.4 - The values returned from Math.random can be manipulated to produce a random int or double in a defined range.	Chapter 7, Lesson 3

UNIT 3: Boolean Expressions and if Statements	**CITATION(S)**
TOPIC 3.1: Boolean Expressions	
CON-1.E.1 - Primitive values and reference values can be compared using relational operators (i.e., == and !=).	Chapter 8, Lesson 1 Chapter 9, Lesson 1
CON-1.E.2 - Arithmetic expression values can be compared using relational operators (i.e., <, >, <=, >=).	Chapter 8, Lesson 1
CON-1.E.3 - An expression involving relational operators evaluates to a Boolean value.	Chapter 8, Lesson 1
TOPIC 3.2: if Statements and Control Flow	
CON-2.A.1 - Conditional statements interrupt the sequential execution of statements.	Chapter 8, Lesson 2
CON-2.A.2 - if statements affect the flow of control by executing different statements based on the value of a Boolean expression.	Chapter 8, Lesson 2
CON-2.A.3 - A one-way selection (if statement) is written when there is a set of statements to execute under a certain condition. In this case, the body is executed only when the Boolean condition is true.	Chapter 8, Lesson 2
TOPIC 3.3: if-else Statements	
CON-2.A.4 - A two-way selection is written when there are two sets of statements— one to be executed when the Boolean condition is true, and another set for when the Boolean condition is false. In this case, the body of the “if” is executed when the Boolean condition is true, and the body of the “else” is executed when the Boolean condition is false.	Chapter 8, Lesson 3
TOPIC 3.4: elseif Statements	
CON-2.A.5 - A multi-way selection is written when there are a series of conditions with different statements for each condition. Multi-way selection is performed using if-else-if statements such that exactly one section of code is executed based on the first condition that evaluates to true.	Chapter 8, Lesson 3
TOPIC 3.5: Compound Boolean Expressions	
CON-2.B.1 - Nested if statements consist of if statements within if statements.	Chapter 8, Lesson 3
CON-1.F.1 - Logical operators !(not), &&(and), and ||(or) are used with Boolean values. This represents the order these operators will be evaluated.	Chapter 9, Lesson 2
CON-1.F.2 - An expression involving logical operators evaluates to a Boolean value.	Chapter 9, Lesson 2

CON-1.F.3 - When the result of a logical expression using && or || can be determined by evaluating only the first Boolean operand, the second is not evaluated. This is known as short-circuited evaluation.	Chapter 9, Lesson 2
TOPIC 3.6: Equivalent Boolean Expressions	
CON-1.G.1 - De Morgan's Laws can be applied to Boolean expressions.	Chapter 9, Lesson 3
CON-1.G.2 - Truth tables can be used to prove Boolean identities.	Chapter 9, Lesson 3
CON-1.G.3 - Equivalent Boolean expressions will evaluate to the same value in all cases.	Chapter 9, Lesson 3
TOPIC 3.7: Comparing Objects	
CON-1.H.1 - Two object references are considered aliases when they both reference the same object.	Chapter 9, Lesson 1
CON-1.H.2 - Object reference values can be compared, using == and !=, to identify aliases.	Chapter 9, Lesson 1
CON-1.H.3 - A reference value can be compared with null, using == or !=, to determine if the reference actually references an object.	Chapter 9, Lesson 1
CON-1.H.4 - Often classes have their own equals method, which can be used to determine whether two objects of the class are equivalent.	Chapter 9, Lesson 1

UNIT 4: Iteration	**CITATION(S)**
TOPIC 4.1: while Loops	
CON-2.C.1 - Iteration statements change the flow of control by repeating a set of statements zero or more times until a condition is met.	Chapter 12, Lesson 2
CON-2.C.2 - In loops, the Boolean expression is evaluated before each iteration of the loop body, including the first. When the expression evaluates to true, the loop body is executed. This continues until the expression evaluates to false, whereupon the iteration ceases.	Chapter 12, Lesson 2
CON-2.C.3 - A loop is an infinite loop when the Boolean expression always evaluates to true.	Chapter 12, Lesson 2
CON-2.C.4 - If the Boolean expression evaluates to false initially, the loop body is not executed at all.	Chapter 12, Lesson 2
CON-2.C.5 - Executing a return statement inside an iteration statement will halt the loop and exit the method or constructor.	Chapter 12, Lesson 3
CON-2.D.1- There are standard algorithms to: * Identify if an integer is or is not evenly divisible by another integer * Identify the individual digits in an integer * Determine the frequency with which a specific criterion is met	Chapter 13, Lessons 3, 4
CON-2.D.2 - There are standard algorithms to: * Determine a minimum or maximum value * Compute a sum, average, or mode	Chapter 12, Lesson 1 Chapter 13, Lesson 3 Chapter 18, Lesson 4

TOPIC 4.2: for Loops	
CON-2.E.1 - There are three parts in a for loop header: the initialization, the Boolean expression, and the increment. The increment statement can also be a decrement statement.	Chapter 12, Lesson 1
CON-2.E.2 - In a for loop, the initialization statement is only executed once before the first Boolean expression evaluation. The variable being initialized is referred to as a loop control variable.	Chapter 12, Lesson 1
CON-2.E.3 - In each iteration of a for loop, the increment statement is executed after the entire loop body is executed and before the Boolean expression is evaluated again.	Chapter 12, Lesson 1
CON-2.E.4 - A for loop can be rewritten into an equivalent while loop and vice versa.	Chapter 12, Lesson 2
CON-2.E.5 - “Off by one” errors occur when the iteration statement loops one time too many or one time too few.	Chapter 12, Lesson 2
TOPIC 4.3: Developing Algorithms Using Strings	
CON-2.F.1 - There are standard algorithms that utilize String traversals to: * Find if one or more substrings has a particular property * Determine the number of substrings that meet specific criteria * Create a new string with the characters reversed	Chapter 13, Lesson 4
TOPIC 4.4: Nested Iteration	
CON-2.G.1 - Nested iteration statements are iteration statements that appear in the body of another iteration statement.	Chapter 12, Lesson 4
CON-2.G.2 - When a loop is nested inside another loop, the inner loop must complete all its iterations before the outer loop can continue.	Chapter 12, Lesson 4
TOPIC 4.5: Informal Code Analysis	
CON-2.H.1 - A statement execution count indicates the number of times a statement is executed by the program.	Chapter 12, Lesson 1

UNIT 5: Writing Classes	**CITATION(S)**
TOPIC 5.1: Anatomy of a Class	
MOD-2.A.1 - The keywords public and private affect the access of classes, data, constructors, and methods.	Chapter 14, Lesson 1
MOD-2.A.2 - The keyword private restricts access to the declaring class, while the keyword public allows access from classes outside the declaring class.	Chapter 14, Lesson 1
MOD-2.A.3 - Classes are designated public.	Chapter 14, Lesson 2
MOD-2.A.4 - Access to attributes should be kept internal to the class. Therefore, instance variables are designated as private.	Chapter 14, Lesson 3
MOD-2.A.5 - Constructors are designated public.	Chapter 14, Lesson 4
MOD-2.A.6 - Access to behaviors can be internal or external to the class. Therefore, methods can be designated as either public or private.	Chapter 14, Lesson 5

MOD-3.A.1 - Data encapsulation is a technique in which the implementation details of a class are kept hidden from the user.	Chapter 14, Lessons 1, 3
MOD-3.A.2 - When designing a class, programmers make decisions about what data to make accessible and modifiable from an external class. Data can be either accessible or modifiable, or it can be both or neither.	Chapter 14, Lesson 3
MOD-3.A.3 - Instance variables are encapsulated by using the private access modifier.	Chapter 14, Lesson 3
MOD-3.A.4 - The provided accessor and mutator methods in a class allow client code to use and modify data.	Chapter 14, Lessons 3, 5
TOPIC 5.2: Constructors	
MOD-2.B.1 - An object's state refers to its attributes and their values at a given time and is defined by instance variables belonging to the object. This creates a "has-a" relationship between the object and its instance variables.	Chapter 14, Lesson 3
MOD-2.B.2 - Constructors are used to set the initial state of an object, which should include initial values for all instance variables.	Chapter 14, Lesson 4
MOD-2.B.3 - Constructor parameters are local variables to the constructor and provide data to initialize instance variables.	Chapter 14, Lesson 4
MOD-2.B.4 - When a mutable object is a constructor parameter, the instance variable should be initialized with a copy of the referenced object. In this way, the instance variable is not an alias of the original object, and methods are prevented from modifying the state of the original object.	Chapter 14, Lesson 4
MOD-2.B.5 - When no constructor is written, Java provides a no-argument constructor, and the instance variables are set to default values.	Chapter 14, Lesson 4
TOPIC 5.3: Documentation with Comments	
MOD-2.C.1- Comments are ignored by the compiler and are not executed when the program is run.	Chapter 15, Lesson 1
MOD-2.C.2 - Three types of comments in Java include /* */, which generates a block of comments, //, which generates a comment on one line, and /** */, which are Javadoc comments and are used to create API documentation.	Chapter 15, Lesson 1
MOD-2.C.3 - A precondition is a condition that must be true just prior to the execution of a section of program code in order for the method to behave as expected. There is no expectation that the method will check to ensure preconditions are satisfied.	Chapter 15, Lesson 1
MOD-2.C.4 - A postcondition is a condition that must always be true after the execution of a section of program code. Postconditions describe the outcome of the execution in terms of what is being returned or the state of an object.	Chapter 15, Lesson 1
MOD-2.C.5 - Programmers write method code to satisfy the postconditions when preconditions are met.	Chapter 15, Lesson 1
TOPIC 5.4: Accessor Methods	
MOD-2.D.1 - An accessor method allows other objects to obtain the value of instance variables or static variables.	Chapter 14, Lesson 5 Chapter 15, Lesson 3
MOD-2.D.2 - A non-void method returns a single value. Its header includes the return type in place of the keyword void.	Chapter 14, Lesson 5 Chapter 15, Lesson 3

MOD-2.D.3 - In non-void methods, a return expression compatible with the return type is evaluated, and a copy of that value is returned. This is referred to as “return by value.”	Chapter 14, Lesson 5Chapter 15, Lesson 3
MOD-2.D.4 - When the return expression is a reference to an object, a copy of that reference is returned, not a copy of the object.	Chapter 15, Lesson 3
MOD-2.D.5 - The return keyword is used to return the flow of control to the point immediately following where the method or constructor was called.	Chapter 15, Lesson 3
MOD-2.D.6 - The toString method is an overridden method that is included in classes to provide a description of a specific object. It generally includes what values are stored in the instance data of the object.	Chapter 15, Lesson 1
MOD-2.D.7 - If System.out.print or System.out.println is passed an object, that object’s toString method is called, and the returned string is printed.	Chapter 15, Lesson 1
TOPIC 5.5: Mutator Methods	
MOD-2.E.1 - A void method does not return a value. Its header contains the keyword void before the method name.	Chapter 14, Lesson 5 Chapter 15, Lesson 3
MOD-2.E.2 - A mutator (modifier) method is often a void method that changes the values of instance variables or static variables.	Chapter 14, Lesson 5 Chapter 15, Lesson 3
TOPIC 5.6: Writing Methods	
MOD-2.F.1 - Methods can only access the private data and methods of a parameter that is a reference to an object when the parameter is the same type as the method’s enclosing class.	Chapter 15, Lesson 3
MOD-2.F.2 - Non-void methods with parameters receive values through parameters, use those values, and return a computed value of the specified type.	Chapter 14, Lesson 5
MOD-2.F.3 - It is good programming practice to not modify mutable objects that are passed as parameters unless required in the specification.	Chapter 14, Lesson 5
MOD-2.F.4- When an actual parameter is a primitive value, the formal parameter is initialized with a copy of that value. Changes to the formal parameter have no effect on the corresponding actual parameter.	Chapter 14, Lesson 5
MOD-2.F.5 - When an actual parameter is a reference to an object, the formal parameter is initialized with a copy of that reference, not a copy of the object. If the reference is to a mutable object, the method or constructor can use this reference to alter the state of the object.	Chapter 14, Lesson 5
MOD-2.F.6 - Passing a reference parameter results in the formal parameter and the actual parameter being aliases. They both refer to the same object.	Chapter 14, Lesson 5
TOPIC 5.7: Static Variables and Methods	
MOD-2.G.1 - Static methods are associated with the class, not objects of the class.	Chapter 16, Lesson 2
MOD-2.G.2 - Static methods include the keyword static in the header before the method name.	Chapter 16, Lesson 2
MOD-2.G.3 - Static methods cannot access or change the values of instance variables.	Chapter 16, Lesson 2
MOD-2.G.4 - Static methods can access or change the values of static variables.	Chapter 16, Lesson 2

MOD-2.G.5 - Static methods do not have a this reference and are unable to use the class's instance variables or call non-static methods.	Chapter 16, Lessons 2, 3
MOD-2.H.1 - Static variables belong to the class, with all objects of a class sharing a single static variable.	Chapter 16, Lesson 1
MOD-2.H.2 - Static variables can be designated as either public or private and are designated with the static keyword before the variable type.	Chapter 16, Lesson 1
MOD-2.H.3 - Static variables are used with the class name and the dot operator, since they are associated with a class, not objects of a class.	Chapter 16, Lesson 1
TOPIC 5.8: Scope and Access	
VAR-1.G.1 - Local variables can be declared in the body of constructors and methods. These variables may only be used within the constructor or method and cannot be declared to be public or private.	Chapter 15, Lesson 2
VAR-1.G.2 - When there is a local variable with the same name as an instance variable, the variable name will refer to the local variable instead of the instance variable.	Chapter 15, Lesson 2
VAR-1.G.3 - Formal parameters and variables declared in a method or constructor can only be used within that method or constructor.	Chapter 15, Lesson 2
VAR-1.G.4 - Through method decomposition, a programmer breaks down a large problem into smaller subproblems by creating methods to solve each individual subproblem.	Chapter 15, Lesson 1
TOPIC 5.9: this Keyword	
VAR-1.H.1 - Within a non-static method or a constructor, the keyword this is a reference to the current object—the object whose method or constructor is being called.	Chapter 16, Lesson 3
VAR-1.H.2 - The keyword this can be used to pass the current object as an actual parameter in a method call.	Chapter 16, Lesson 3
TOPIC 5.10: Ethical and Social Implications of Computing Systems	
IOC-1.A.1 - System reliability is limited. Programmers should make an effort to maximize system reliability.	Chapter 1, Lesson 4 Chapter 10
IOC-1.A.2 - Legal issues and intellectual property concerns arise when creating programs.	Chapter 1, Lesson 4-5
IOC-1.A.3 - The creation of programs has impacts on society, economies, and culture. These impacts can be beneficial and/or harmful.	Chapter 1, Lesson 4-5

UNIT 6: Array	**CITATION(S)**
TOPIC 6.1: Array Creation and Access	
VAR-2.A.1 - The use of array objects allows multiple related items to be represented using a single variable.	Chapter 18, Lesson 1
VAR-2.A.2 - The size of an array is established at the time of creation and cannot be changed.	Chapter 18, Lesson 1
VAR-2.A.3 - Arrays can store either primitive data or object reference data.	Chapter 18, Lesson 1

VAR-2.A.4 - When an array is created using the keyword new, all of its elements are initialized with a specific value based on the type of elements: * Elements of type int are initialized to 0 * Elements of type double are initialized to 0.0 * Elements of type boolean are initialized to false * Elements of a reference type are initialized to the reference value null. No objects are automatically created	Chapter 18, Lesson 1
VAR-2.A.5 - Initializer lists can be used to create and initialize arrays.	Chapter 18, Lesson 1
VAR-2.A.6 - Square brackets ([]) are used to access and modify an element in a 1D array using an index.	Chapter 18, Lesson 1
VAR-2.A.7 - The valid index values for an array are 0 through one less than the number of elements in the array, inclusive. Using an index value outside of this range will result in an ArrayIndexOutOfBoundsException being thrown.	Chapter 18, Lesson 1
TOPIC 6.2: Traversing Arrays	
VAR-2.B.1 - Iteration statements can be used to access all the elements in an array. This is called traversing the array.	Chapter 18, Lesson 2
VAR-2.B.2 - Traversing an array with an indexed for loop or while loop requires elements to be accessed using their indices.	Chapter 18, Lesson 2
VAR-2.B.3 - Since the indices for an array start at 0 and end at the number of elements − 1, “off by one” errors are easy to make when traversing an array, resulting in an ArrayIndexOutOfBoundsException being thrown.	Chapter 18, Lesson 1
TOPIC 6.3: Enhanced forLoop for Arrays	
VAR-2.C.1 - An enhanced for loop header includes a variable, referred to as the enhanced for loop variable.	Chapter 18, Lesson 3
VAR-2.C.2 - For each iteration of the enhanced for loop, the enhanced for loop variable is assigned a copy of an element without using its index.	Chapter 18, Lesson 3
VAR-2.C.3 - Assigning a new value to the enhanced for loop variable does not change the value stored in the array.	Chapter 18, Lesson 3
VAR-2.C.4 - Program code written using an enhanced for loop to traverse and access elements in an array can be rewritten using an indexed for loop or a while loop.	Chapter 18, Lesson 3
TOPIC 6.4: Developing Algorithms Using Arrays	
CON-2.I.1 - There are standard algorithms that utilize array traversals to: * Determine a minimum or maximum value * Compute a sum, average, or mode * Determine if at least one element has a particular property * Determine if all elements have a particular property * Access all consecutive pairs of elements * Determine the presence or absence of duplicate elements * Determine the number of elements meeting specific criteria	Chapter 18, Lessons 4-5
CON-2.I.2 - There are standard array algorithms that utilize traversals to: * Shift or rotate elements left or right * Reverse the order of the elements	Chapter 18, Lessons 4-5

UNIT 7: ArrayList	CITATION(S)
TOPIC 7.1: Introduction to ArrayList	
VAR-2.D.1 - An ArrayList object is mutable and contains object references.	Chapter 19, Lesson 2
VAR-2.D.2 - The ArrayList constructor ArrayList() constructs an empty list.	Chapter 19, Lesson 2
VAR-2.D.3 - Java allows the generic type ArrayList<E>, where the generic type E specifies the type of the elements.	Chapter 19, Lesson 2
VAR-2.D.4 - When ArrayList<E> is specified, the types of the reference parameters and return type when using the methods are type E.	Chapter 19, Lesson 2
VAR-2.D.5 - ArrayList<E> is preferred over ArrayList because it allows the compiler to find errors that would otherwise be found at run-time.	Chapter 19, Lesson 2
TOPIC 7.2: ArrayList Methods	
VAR-2.D.6 - The ArrayList class is part of the java.util package. An import statement can be used to make this class available for use in the program.	Chapter 19, Lesson 2
VAR-2.D.7 - The following ArrayList methods—including what they do and when they are used—are part of the Java Quick Reference:	See below
int size() - Returns the number of elements in the list	Chapter 19, Lesson 2
boolean add(E obj) - Appends obj to end of list; returns true	Chapter 19, Lesson 2
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 adds 1 to size	Chapter 19, Lesson 2
E get(int index) - Returns the element at position index in the list	Chapter 19, Lesson 2
E set(int index, E obj) — Replaces the element at position index with obj; returns the element formerly at position index	Chapter 19, Lesson 2
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 subtracts 1 from size; returns the element formerly at position index	Chapter 19, Lesson 2
TOPIC 7.3: Traversing ArrayLists	
VAR-2.E.1 - Iteration statements can be used to access all the elements in an ArrayList. This is called traversing the ArrayList.	Chapter 19, Lesson 2
VAR-2.E.2 - Deleting elements during a traversal of an ArrayList requires using special techniques to avoid skipping elements.	Chapter 19, Lesson 2
VAR-2.E.3 - Since the indices for an ArrayList start at 0 and end at the number of elements − 1, accessing an index value outside of this range will result in an ArrayIndexOutOfBoundsException being thrown.	Chapter 19, Lesson 2
VAR-2.E.4 - Changing the size of an ArrayList while traversing it using an enhanced for loop can result in a ConcurrentModificationException being thrown. Therefore, when using an enhanced for loop to traverse an ArrayList, you should not add or remove elements.	Chapter 19, Lesson 3

TOPIC 7.4: Developing Algorithms Using ArrayLists	
CON-2.J.1 - There are standard ArrayList algorithms that utilize traversals to: * Insert elements * Delete elements * Apply the same standard algorithms that are used with 1D arrays	Chapter 19, Lessons 3-4
CON-2.J.2 - Some algorithms require multiple String, array, or ArrayList objects to be traversed simultaneously.	Chapter 19, Lesson 4
TOPIC 7.5: Searching	
CON-2.K.1 - There are standard algorithms for searching.	Chapter 20, Lesson 4
CON-2.K.2 - Sequential/linear search algorithms check each element in order until the desired value is found or all elements in the array or ArrayList have been checked.	Chapter 20, Lesson 4
TOPIC 7.6: Sorting	
CON-2.L.1 - Selection sort and insertion sort are iterative sorting algorithms that can be used to sort elements in an array or ArrayList.	Chapter 20, Lessons 2-3
CON-2.M.1 - Informal run-time comparisons of program code segments can be made using statement execution counts.	Chapter 20, Lessons 1-3
TOPIC 7.7: Ethical Issues Around Data Collection	
IOC-1.B.1 - When using the computer, personal privacy is at risk. Programmers should attempt to safeguard personal privacy.	Chapter 1, Lesson 4 Suppl. Chapter 3, Lesson 1
IOC-1.B.2 - Computer use and the creation of programs have an impact on personal security. These impacts can be beneficial and/or harmful.	Chapter 1, Lesson 4 Suppl. Chapter 3, Lesson 1

UNIT 8: 2D Array	**CITATION(S)**
TOPIC 8.1: 2D Arrays	
VAR-2.F.1 - 2D arrays are stored as arrays of arrays. Therefore, the way 2D arrays are created and indexed is similar to 1D array objects.	Chapter 21, Lesson 3
VAR-2.F.2 - For the purposes of the exam, when accessing the element at arr[first][second], the first index is used for rows, the second index is used for columns.	Chapter 21, Lesson 1
VAR-2.F.3 - The initializer list used to create and initialize a 2D array consists of initializer lists that represent 1D arrays.	Chapter 21, Lesson 3
VAR-2.F.4 - The square brackets [row][col] are used to access and modify an element in a 2D array.	Chapter 21, Lesson 1
VAR-2.F.5 - “Row-major order” refers to an ordering of 2D array elements where traversal occurs across each row, while “column-major order” traversal occurs down each column.	Chapter 21, Lesson 2

TOPIC 8.2: Traversing 2D Arrays	
VAR-2.G.1 - Nested iteration statements are used to traverse and access all elements in a 2D array. Since 2D arrays are stored as arrays of arrays, the way 2D arrays are traversed using for loops and enhanced for loops is similar to 1D array objects.	Chapter 21, Lesson 2
VAR-2.G.2 - Nested iteration statements can be written to traverse the 2D array in “row-major order” or “column-major order.”	Chapter 21, Lesson 2
VAR-2.G.3 - The outer loop of a nested enhanced for loop used to traverse a 2D array traverses the rows. Therefore, the enhanced for loop variable must be the type of each row, which is a 1D array. The inner loop traverses a single row. Therefore, the inner enhanced for loop variable must be the same type as the elements stored in the 1D array.	Chapter 21, Lesson 2
CON-2.N.1 - When applying sequential/linear search algorithms to 2D arrays, each row must be accessed then sequential/linear search applied to each row of a 2D array.	Chapter 21, Lesson 4
CON-2.N.2 - All standard 1D array algorithms can be applied to 2D array objects.	Chapter 21, Lesson 4

UNIT 9: Inheritance	**CITATION(S)**
TOPIC 9.1: Creating Superclasses and Subclasses	
MOD-3.B.1 - A class hierarchy can be developed by putting common attributes and behaviors of related classes into a single class called a superclass.	Chapter 22, Lesson 1
MOD-3.B.2 - Classes that extend a superclass, called subclasses, can draw upon the existing attributes and behaviors of the superclass without repeating these in the code.	Chapter 22, Lesson 1
MOD-3.B.3 - Extending a subclass from a superclass creates an “is-a” relationship from the subclass to the superclass.	Chapter 22, Lesson 1
MOD-3.B.4 - The keyword extends is used to establish an inheritance relationship between a subclass and a superclass. A class can extend only one superclass.	Chapter 22, Lesson 1
TOPIC 9.2: Writing Constructors for Subclasses	
MOD-3.B.5 - Constructors are not inherited.	Chapter 22, Lesson 2
MOD-3.B.6 - The superclass constructor can be called from the first line of a subclass constructor by using the keyword super and passing appropriate parameters.	Chapter 22, Lesson 2
MOD-3.B.7 - The actual parameters passed in the call to the superclass constructor provide values that the constructor can use to initialize the object’s instance variables.	Chapter 22, Lesson 2
MOD-3.B.8 - When a subclass’s constructor does not explicitly call a superclass’s constructor using super, Java inserts a call to the superclass’s no-argument constructor.	Chapter 22, Lesson 2

MOD-3.B.9 - Regardless of whether the superclass constructor is called implicitly or explicitly, the process of calling superclass constructors continues until the Object constructor is called. At this point, all of the constructors within the hierarchy execute beginning with the Object constructor.	Chapter 22, Lesson 2
TOPIC 9.3: Overriding Methods	
MOD-3.B.10 - Method overriding occurs when a public method in a subclass has the same method signature as a public method in the superclass.	Chapter 23, Lesson 1
MOD-3.B.11 - Any method that is called must be defined within its own class or its superclass.	Chapter 23, Lesson 1
MOD-3.B.12 - A subclass is usually designed to have modified (overridden) or additional methods or instance variables.	Chapter 22, Lesson 1 Chapter 23, Lesson 1
MOD-3.B.13 - A subclass will inherit all public methods from the superclass; these methods remain public in the subclass.	Chapter 23, Lesson 1
TOPIC 9.4: super Keyword	
MOD-3.B.14 - The keyword super can be used to call a superclass's constructors and methods.	Chapter 22, Lesson 2 Chapter 23, Lesson 3
MOD-3.B.15 - The superclass method can be called in a subclass by using the keyword super with the method name and passing appropriate parameters.	Chapter 23, Lesson 3
TOPIC 9.5: Creating References Using Inheritance Hierarchies	
MOD-3.C.1 - When a class S "is-a" class T, T is referred to as a superclass, and S is referred to as a subclass.	Chapter 23, Lesson 3
MOD-3.C.2 - If S is a subclass of T, then assigning an object of type S to a reference of type T facilitates polymorphism.	Chapter 23, Lesson 3
MOD-3.C.3 - If S is a subclass of T, then a reference of type T can be used to refer to an object of type Tor S.	Chapter 23, Lesson 3
MOD-3.C.4 - Declaring references of type T, when S is a subclass of T, is useful in the following declarations: * Formal method parameters * arrays — T[]varArrayList<T>var	Chapter 23, Lesson 3
TOPIC 9.6: Polymorphism	
MOD-3.D.1 - Utilize the Object class through inheritance.	Chapter 23, Lesson 4
MOD-3.D.2 - At compile time, methods in or inherited by the declared type determine the correctness of a non-static method call.	Chapter 23, Lesson 1
MOD-3.D.3 - At run-time, the method in the actual object type is executed for a non-static method call.	Chapter 23, Lesson 1
TOPIC 9.7: Object Superclass	
MOD-3.E.1 - The Object class is the superclass of all other classes in Java.	Chapter 23, Lesson 4
MOD-3.E.2 - The Object class is part of the java.lang package.	Chapter 23, Lesson 4

MOD-3.E.3 - The following Object class methods and constructors—including what they do and when they are used—are part of the Java Quick Reference: * boolean equals(Object other) * String toString()	Chapter 23, Lesson 4
MOD-3.E.4 - Subclasses of Object often override the equals and toString methods with class-specific implementations.	Chapter 23, Lesson 4

UNIT 10: Recursion	**CITATION(S)**
TOPIC 10.1: Recursion	
CON-2.O.1 - A recursive method is a method that calls itself.	Chapter 24, Lesson 1
CON-2.O.2 - Recursive methods contain at least one base case, which halts the recursion, and at least one recursive call.	Chapter 24, Lesson 1
CON-2.O.3 - Each recursive call has its own set of local variables, including the formal parameters.	Chapter 24, Lesson 1
CON-2.O.4 - Parameter values capture the progress of a recursive process, much like loop control variable values capture the progress of a loop.	Chapter 24, Lesson 1
CON-2.O.5 - Any recursive solution can be replicated through the use of an iterative approach.	Chapter 24, Lesson 1
CON-2.O.6 - Recursion can be used to traverse String, array, and ArrayList objects.	Chapter 24, Lesson 1
TOPIC 10.2: Recursive Searching and Sorting	
CON-2.P.1 - Data must be in sorted order to use the binary search algorithm.	Chapter 24, Lesson 2
CON-2.P.2 - The binary search algorithm starts at the middle of a sorted array or ArrayList and eliminates half of the array or ArrayList in each iteration until the desired value is found or all elements have been eliminated.	Chapter 24, Lesson 2
CON-2.P.3 - Binary search can be more efficient than sequential/linear search.	Chapter 24, Lesson 2
CON-2.P.4 - The binary search algorithm can be written either iteratively or recursively.	Chapter 24, Lesson 2
CON-2.Q.1 - Merge sort is a recursive sorting algorithm that can be used to sort elements in an array or ArrayList.	Chapter 24, Lesson 3