Big Ideas =

Modularity: Modularity requires students to simplify concepts and processes by looking at the big picture rather than the details and developing abstractions. Whether this is in the representation of objects or concepts, in the use of preexisting processes, or in the creation and organization of code into different methods or classes. There are several instructional strategies that can help students make these connections, including using manipulatives and diagramming. large amounts of data or complex relationships. In addition, when attempting to define or interpret processes and the role that variables play in those processes, strategies of kinesthetic learning and look for a pattern are helpful.

Variables: Utilizing variables to help simplify concepts and processes is an important component of this big idea and involves the creation of data abstractions. In order to help students see the role that variables play in generalizable solutions, try the instructional strategies of creating a plan and identifying a subtask. These strategies can also be used to show how variables manage

Control: Being able to write and evaluate mathematical expressions is a necessary component in determining the computed result of an expression. Code tracing, error analysis, and simplify the problem are strategies that can help students understand the relationship between variables and quantities and provide them opportunities to practice writing and evaluating mathematical expressions. Other strategies like pair programming and predict and compare provide students with practice developing algorithmic thinking, such as defining and interpreting processes that are used in a program, like selection and iteration.

Impact of Computing: Strategies such as discussion group, jigsaw, and think aloud provide students opportunities to explain the cause and effect of a program. These opportunities can help distinguish a program's intended

Lesson Plan Objectives =

Enduring understandings are the long-term takeaways related to the big ideas that leave a lasting impression on students. Students build and earn these understandings over time by exploring and applying course content throughout the year.

Learning objectives define what a student should be able to do with content knowledge to progress toward the enduring understandings.

Essential knowledge statements describe the knowledge required to perform the learning objective.

CON-1 — The way variables and operators are sequenced and combined in an expression determines the computed result.

CON-1.A — Evaluate arithmetic expressions in a program code.

CON-1.A.1 — A literal is the source code representation of a fixed value.

CON-1.A.2 — Arithmetic expressions include expressions of type int and double.

CON-1.A.3 — The arithmetic operators consist of +, -, *, /, and %.

CON-1.A.4 — An arithmetic operation that uses two int values will evaluate to an int value.

CON-1.A.5 — An arithmetic operation that uses a double value will evaluate to a double value.

CON-1.A.6 — Operators can be used to construct compound expressions.

CON-1.A.7 — During evaluation, operands are associated with operators according to operator precedence to determine how they are grouped.

CON-1.A.8 — An attempt to divide an integer by zero will result in an ArithmeticException to occur.

CON-1.B — Evaluate what is stored in a variable as a result of an expression with an assignment statement.

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.

CON-1.B.2 — During execution, expressions are evaluated to produce a single value.

CON-1.B.3 — The value of an expression has a type based on the evaluation of the expression.

CON-1.B.4 — Compound assignments operators (+=, -=, *=, /=, %=) can be used in place of the assignment operator

CON-1.B.5 — The increment operator (++) and decrement operator (—) are used to add 1 and subtract 1 from the stored value of a variable or an array element. The new value is assigned to the variable or array element.

CON-1.C — Evaluate arithmetic expressions that use casting.

CON-1.C.1 — The casting operators (int) and (double) can be used to create a temporary value converted to a different data type.

CON-1.C.2 — Casting a double value to an int causes the digits to the right of the decimal point to be truncated.

CON-1.C.3 — Some programming code causes int values to be automatically cast (widened) to double values.

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.

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 int the range from Integer.MIN_VALUE to Integer.MAX_VALUE inclusive.

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.

CON-1.D — Evaluate expressions that use the Math class methods.

CON-1.D.1 — The Math class is part of the java.lang package.

CON-1.D.2 — The Math class contains only static methods.

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: int abs(int x) - Returns the absolute value of an int value, double abs(double x) - Returns the absolute value of a double value double, pow(double base, double exponent) - Returns the value of the first parameter raised to the power of the second parameter double, sqrt(double x) - Returns the positive square root of a double value doubl,e random() - Returns a double value greater than or equal to 0.0 and less than 1.0

CON-1.D.4 — The values returned from Math.random can be manipulated to produce a random int or double in a defined range.

CON-1.E — Evaluate Boolean expressions that use relational operators in program code.

CON-1.E.1 — Primitive values and reference values can be compared using relational operators (i.e., == and !=).

CON-1.E.2 — Arithmetic expression values can be compared using relational operators (i.e., <, >, <=, >=).

CON-1.E.3 — An expression involving relational operators evaluates to a Boolean value.

CON-1.F — Evaluate compound Boolean expressions in program code.

CON-1.F.1 — Logical operators !(not), &&(and), and ||(or) are used with Boolean values. This represents the order these operators will be evaluated.

CON-1.F.2 — An expression involving logical operators evaluates to a Boolean value.

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.

CON-1.G — Compare and contrast equivalent Boolean expressions.

CON-1.G.1 — DeMorgan*ƒ*'s Laws can be applied to Boolean expressions.

CON-1.G.2 — Truth tables can be used to prove Boolean identities.

CON-1.G.3 — Equivalent Boolean expressions will evaluate to the same value in all cases.

CON-1.H — Compare object references using Boolean expressions in program code

CON-1.H.1 — Two object references are considered aliases when they both reference the same object.

CON-1.H.2 — Object reference values can be compared, using == and !=, to identify aliases.

CON-1.H.3 — A reference value can be compared with null, using == or !=, to determine if the reference actually references an object.

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

CON-2 — Programmers incorporate iteration and selection into code as a way of providing instructions for the computer to process each of the many possible input values.

CON-2.A — Represent branching logical processes by using conditional statements.

CON-2.A.1 — Conditional statements interrupt the sequential execution of statements.

CON-2.A.2 — If-statements affect the control of flow by executing different statements based on the value of a Boolean expression

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.

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.

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.

CON-2.B — Represent branching logical processes by using nested conditional statements.

CON-2.C — Represent iterative processes using a while loop.

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.

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.

CON-2.C.3 — A loop is an infinite loop when the Boolean expression always evaluates to true.

CON-2.C.4 — If the Boolean expression evaluates to false initially, the loop body is not executed at all.

CON-2.C.5 — Executing a return statement inside an iteration statement will halt the loop and exit the method or constructor.

CON-2.D — For algorithms in the context of a particular specification that does not require the use of traversals

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.

CON-2.D.2 — There are standard algorithms to determine a minimum or maximum value, Compute a sum, average or mode.

CON-2.E — Represent iterative processes using a for loop

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.

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.

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.

CON-2.E.4 — A for loop can be rewritten into an equivalent while loop and vice versa.

CON-2.E.5 — Off by one errors occur when the iteration statement loops one time too many or one time too few.

CON-2.F — For algorithms in the context of a particular specification that involves String objects: identify standard algorithms, modify standard algorithms, develop an algorithm.

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

CON-2.G — Represent nested iterative processes.

CON-2.G.1 — Nested iteration statements are iteration statements that appear in the body of another iteration statement.

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.

CON-2.H — Compute statement execution counts and informal run-time comparison of iterative statements.

CON-2.H.1 — A statement execution count indicates the number of times a statement is executed by the program.

CON-2.I — For algorithms in the context of a particular specification that requires the use of Array traversals: (a). Identify standard algorithms (b). Modify standard algorithms (c). Develop an algorithm.

CON-2.I.1 — There are standard algorithms that utilize array traversals to:

CON-2.I.1.i — Determine a minimum or maximum value.

CON-2.I.1.ii — Compute a sum, average, or mode.

CON-2.I.1.iii — Determine if at least one element has a particular property.

CON-2.I.1.iv — Determine if all elements have a particular property.

CON-2.I.1.v — Access all consecutive pairs of elements.

CON-2.I.1.vi — Determine the presence or absence of duplicate elements.

CON-2.I.1.vii — Determine the number of elements meeting specific criteria.

CON-2.I.2 — There are standard array algorithms that utilize traversals to:

CON-2.I.2.i — Shift or rotate elements left or right.

CON-2.I.2.ii — Reverse the order of the elements.

CON-2.J — For algorithms in the context of a particular specification that does require the use of ArrayList traversals: identify standard algorithms, modify standard algorithms, develop an algorithm.

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.

CON-2.J.2 — Some algorithms require multiple String, array, or ArrayList objects to be traversed simultaneously.

CON-2.K — Apply sequential/linear search algorithms to search for specific information in array or ArrayList objects.

CON-2.K.1 — There are standard algorithms for searching.

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.

CON-2.L — Apply selection sort and insertion sort algorithms to sort the elements of array or ArrayList objects.

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.

CON-2.M — Compute statement execution counts and informal runtime comparison of sorting algorithms.

CON-2.M.1 — Informal run-time comparisons of program code segments can be made using statement execution counts.

CON-2.N — For algorithms in the context of a particular specification that requires the use of 2D array traversals: Identify standard algorithms. Modify standard algorithms. Develop an algorithm

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.

CON-2.N.2 — All standard 1D array algorithms can be applied to 2D array objects

CON-2.O — Determine the result of executing recursive methods.

CON-2.O.1 — A recursive method is a method that calls itself.

CON-2.O.2 — Recursive methods contain at least one base case, which halts the recursion, and at least one recursive call.

CON-2.O.3 — Each recursive call has its own set of local variables, including the formal parameters.

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.

CON-2.O.5 — Any recursive solution can be replicated through the use of an iterative approach. EXCLUSION STATEMENT(EK CON-2.O.5): Writing recursive program code is outside the scope of the course and AP Exam.

CON-2.P — Apply recursive search algorithms to information in String, 1D array, or ArrayList objects.

CON-2.P.1 — Data must be in sorted order to use the binary search algorithm.

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.

CON-2.P.3 — Binary search is more efficient than sequential/linear search.

CON-2.P.4 — The binary search algorithm can be written recursively.

CON-2.Q — Apply recursive algorithms to sort elements of array or ArrayList objects.

CON-2.Q.1 — Merge sort is a recursive sorting algorithm that can be used to sort elements in an array or ArrayList.

IOC-1 — While programs are typically designed to achieve a specific purpose, they may have unintended consequences.

IOC-1.A — Explain the ethical and social implications of computing systems.

IOC-1.A.1 — System reliability is limited. Programmers should make an effort to maximize system reliability.

IOC-1.A.2 — Legal issues and intellectual property concerns arise when creating programs.

IOC-1.A.3 — The creation of programs has impacts on society, economies, and culture. These impacts can be beneficial and/or harmful.

IOC-1.B — Explain the risks to privacy from collecting and storing personal data on computer systems.

IOC-1.B.1 — When using the computer, personal privacy is at risk. Programmers should attempt to safeguard personal privacy.

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.

MOD-1 — Some objects or concepts are so frequently represented that programmers can draw upon existing code that has already been tested, enabling them to write solutions more quickly and with a greater degree of confidence.

MOD-1.A — Call System class methods to generate output to the console.

MOD-1.A.1 — System.out.print and System.out.println display information on the computer monitor.

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.

MOD-1.B — Explain the relationship between a class and an object.

MOD-1.B.1 — An object is a specific instance of a class with defined attributes.

MOD-1.B.2 — A class is the formal implementation, or blueprint, of the attributes and behaviors of an object.

MOD-1.C — Identify, using its signature, the correct constructor being called

MOD-1.C.1 — A signature consists of the constructor name and the parameter list.

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.

MOD-1.C.3 — A parameter is a value that is passed into a constructor. These are often referred to as actual parameters.

MOD-1.C.4 — Constructors are said to be overloaded when there are multiple

MOD-1.C.5 — The actual parameters passed to a constructor must be compatible with the types identified in the formal parameter list.

MOD-1.C.5 — Parameters are passed using call by value. Call by value initializes the formal parameters with copies of the actual parameters.

MOD-1.D — For creating objects: a. Create objects by calling constructors without parameters. b. Create objects by calling constructors with parameters.

MOD-1.D.1 — Every object is created using the keyword new followed by a call to one of the class's constructors.

MOD-1.D.2 — A class contains constructors that are invoked to create objects. They have the same name as the class.

MOD-1.D.3 — Existing classes and class libraries can be utilized as appropriate to create objects.

MOD-1.D.4 — Parameters allow values to be passed to the constructor to establish the initial state of the object.

MOD-1.E — Call non-static void methods without parameters.

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.

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.

MOD-1.E.3 — A method signature for a method without parameters consists of the method name and an empty parameter list.

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.

MOD-1.E.5 — Non-static methods are called through objects of the class.

MOD-1.E.6 — The dot operator is used along with the object name to call non-static methods.

MOD-1.E.7 — Void methods do not have return values and are therefore not called as part of an expression.

MOD-1.E.8 — Using a null reference to call a method or access an instance variable causes a NullPointerException to be thrown.

MOD-1.F — Call non-static void methods 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.

MOD-1.F.2 — Values provided in the parameter list need to correspond to the order and type in the method signature.

MOD-1.F.3 — Methods are said to be overloaded when there are multiple methods with the same name but a different signature.

MOD-1.G — Call non-static non-void methods with or without parameters.

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.

MOD-1.H — Call static methods

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.

MOD-2 — Programmers use code to represent a physical object or nonphysical concept, real or imagined, by defining a class based on the attributes and/or behaviors of the object or concept.

MOD-2.A — Designate access and visibility constraints to classes, data, constructors, and methods.

MOD-2.A.1 — The keywords public and private affect the access of classes, data, constructors, and methods.

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.

MOD-2.A.3 — Classes are designated public.

MOD-2.A.4 — Access to attributes should be kept internal to the class. Therefore, instance variables are designated as private.

MOD-2.A.5 — Constructors are designated public.

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.

MOD-2.B — Define instance variables for the attributes to be initialized through the constructors of a class.

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.

MOD-2.B.2 — Constructors are used to set the initial state of an object, which should include initial values for all instance variables.

MOD-2.B.3 — Constructor parameters are local variables to the constructor and provide data to initialize instance variables.

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.

MOD-2.B.5 — When no constructor is written, Java provides a no-argument constructor, and the instance variables are set to default values.

MOD-2.C — Describe the functionality and use of program code through comments.

MOD-2.C.1 — Comments are ignored by the compiler and are not executed when the program is run.

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.

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.

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.

MOD-2.C.5 — Programmers write method code to satisfy the postconditions when preconditions are met.

MOD-2.D — Define behaviors of an object through non-void methods without parameters written in a class.

MOD-2.D.1 — An accessor method allows other objects to obtain the value of instance variables or static variables.

MOD-2.D.2 — A non-void method returns a single value. Its header includes the return type n place of the keyword void.

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.

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.

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.

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.

MOD-2.D.6 — The toString method is an overridden method that is included in classes to provide a description of specific object. It generally includes what values are stored in the instance data of the object.

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.

MOD-2.E — Define behaviors of an object through void methods with parameters written in a class.

MOD-2.E.1 — A void method does not return a value. Its header contains the keyword void before the method name.

MOD-2.E.2 — A mutator (modifier) method is often a void method that changes the values of instance variables or static variables.

MOD-2.F — Define behaviors of an object through non-void methods with parameters written in a class.

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.

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.

MOD-2.F.3 — It is good programming practice to not modify mutable objects that are passed as parameters unless required in the specification.

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.

MOD-2.G — Define behaviors of a class through static methods.

MOD-2.G.1 — Static methods are associated with the class, not objects of the class.

MOD-2.G.2 — Static methods include the keyword static in the header before the method name.

MOD-2.G.3 — Static methods cannot access or change the values of instance variables.

MOD-2.G.4 — Static methods can access or change the values of static variables.

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.

MOD-2.H — Define the static variables that belong to the class.

MOD-2.H.1 — Static variables belong to the class, with all objects of a class sharing a single static variable.

MOD-3 — When multiple classes contain common attributes and behaviors, programmers create a new class containing the shared attributes and behaviors forming a hierarchy. Modifications made at the highest level of the hierarchy apply to the subclasses

MOD-3.A — Designate private visibility of instance variables to encapsulate the attributes of an object.

MOD-3.A.1 — Data encapsulation is a technique in which the implementation details of a class are kept hidden from the user.

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.

MOD-3.A.3 — Instance variables are encapsulated by using the private access modifier.

MOD-3.A.4 — The provided accessor and mutator methods in a class allow client code to use and modify data.

MOD-3.B — Create an inheritance relationship from a subclass to the superclass.

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.

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.

MOD-3.B.14 — The keyword super can be used to call a superclass's constructors and methods.

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.

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.

MOD-3.B.3 — Extending a subclass from a superclass creates an is-a relationship from the subclass to the superclass.

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.

MOD-3.B.5 — Constructors are not inherited.

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.

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.

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.

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.

MOD-3.C — Define reference variables of a superclass to be assigned to an object of a subclass in the same hierarchy.

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.

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.

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 T or S.

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[ ] var ArrayList var

MOD-3.D — Call methods in an inheritance relationship.

MOD-3.D.1 — Utilize the Object class through inheritance.

MOD-3.D.2 — At compile time, methods in or inherited by the declared type determine the correctness of a non-static method call.

MOD-3.D.3 — At run-time, the method in the actual object type is executed for a non-static method call.

MOD-3.E — Call Object class methods through inheritance.

MOD-3.E.1 — The Object class is the superclass of all other classes in Java.

MOD-3.E.2 — The Object class is part of the java.lang Package

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()

MOD-3.E.4 — Subclasses of Object often override the equals and toString methods with class specific implementations.

VAR-1 — To find specific solutions to generalizable problems, programmers include variables in their code so that the same algorithm runs using different input values.

VAR-1.A — Create string literals.

VAR-1.A.1 — A string literal is enclosed in double quotes.

VAR-1.B — Identify the most appropriate data type category for a particular specification.

VAR-1.B.1 — A type is a set of values (a domain) and a set of operations on them

VAR-1.B.2 — Data types can be categorized as either primitive or reference.

VAR-1.B.3 — The primitive data types used in this course define the set of operations entify the most appropriate data type category for a particular specification.

VAR-1.C — Declare variables of the correct types to represent primitive data.

VAR-1.C.1 — The three primitive data types used in this course are int, double, and boolean.

VAR-1.C.2 — Each variable has associated memory that is used to hold its value.

VAR-1.C.3 — The memory associated with a variable of a primitive type holds an actual primitive value.

VAR-1.C.4 — When a variable is declared final, its value cannot be changed once it is initialized.

VAR-1.D — Define variables of the correct types to represent reference data.

VAR-1.D.1 — The keyword null is a special value used to indicate that a reference is not associated with any object.

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.

VAR-1.E — For String class: a. Create String objects. b. Call String methods.

VAR-1.E.1 — String objects can be created using string literals or by calling the String class constructor.

VAR-1.E.10 — A String object has index values from 0 to length. Attempting to access indices outside this range will result in an IndexOutOfBoundsException.

VAR-1.E.11 — A String object can be concatenated with an object reference, which implicitly calls the referenced object's toString method.

VAR-1.E.12 — The following String methods and constructors (including what they do and when they are used) are part of the AP Java Subset:

VAR-1.E.12.i — String(String str) Constructs a new String object that represents the same sequence of characters as str.

VAR-1.E.12.ii — int length() returns the number of characters in a String object.

VAR-1.E.12.iii — String

VAR-1.E.13 — A string identical to the single element substring at position index can be created by calling substring(index, index + 1).

VAR-1.E.2 — String objects are immutable, meaning that String methods do not change the String object.

VAR-1.E.3 — String objects can be concatenated using the + or += operator, resulting in a new String object.

VAR-1.E.4 — Primitive values can be concatenated with a String object. This causes implicit conversion of the values to String objects.

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.

VAR-1.F — For wrapper classes: a. Create Integer objects. b. Call Integer methods. c. Create Double objects. d. Call Double methods.

VAR-1.F.1 — The Integer class and Double class are part of the java.lang package.

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: Integer(int value) Constructs a new Integer object that represents the specified int value Integer.MIN_VALUE: The minimum value represented by an int or Integer

VAR-1.G — Explain where variables can be used in the program code.

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.

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.

VAR-1.G.3 — Formal parameters and variables declared in a method or constructor can only be used within that method or constructor.

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.

VAR-1.H — Evaluate object reference expressions that use the keyword this.

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.

VAR-1.H.2 — The keyword this can be used to pass the current object as an actual parameter in a method call.

VAR-2 — To manage large amounts of data or complex relationships in data, programmers write code that groups the data together into a single data structure without creating individual variables for each value.

VAR-2.A — Represent collections of related primitive or object reference data using one dimensional (1D) array objects.

VAR-2.A — Represent collections of related primitive or object reference data using two-dimensional (2D) array objects.

VAR-2.A.1 — The use of array objects allows multiple related items to be represented using a single variable.

VAR-2.A.2 — The size of an array is established at the time of creation and cannot be changed.

VAR-2.A.3 — Arrays can store either primitive data or object reference data.

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

VAR-2.A.5 — Initializer lists can be used to create and initialize arrays.

VAR-2.A.6 — Square brackets ([ ]) are used to access and modify an element in a 1D array using an index.

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.

VAR-2.B — Traverse the elements in a 1D array.

VAR-2.B.1 — Iteration statements can be used to access all the elements in an array. This is called traversing the array.

VAR-2.B.2 — Traversing an array with an indexed for loop or while loop requires elements to be accessed using their indices.

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.

VAR-2.C — Traverse the elements in a 1D array object using an enhanced for loop.

VAR-2.C.1 — An enhanced for loop header includes a variable, referred to as the enhanced for loop variable.

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.

VAR-2.C.3 — Assigning a new value to the enhanced for loop variable does not change the value stored in the array.

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.

VAR-2.D — Represent collections of related object reference data using ArrayList objects.

VAR-2.D.1 — An ArrayList object is mutable and contains object references.

VAR-2.D.2 — The ArrayList constructor ArrayList() constructs an empty list.

VAR-2.D.3 — Java allows the generic type ArrayList, where the generic type E specifies the type of the elements.

VAR-2.D.4 — When ArrayList is specified, the types of the reference parameters and return type when using the methods are type E.

VAR-2.D.5 — ArrayList is preferred over ArrayList because it allows the compiler to find errors that would otherwise be found at run-time.

VAR-2.E — Traverse the elements in an ArrayList object.

VAR-2.E.1 — Iteration statements can be used to access all the elements in an ArrayList. This is called traversing the ArrayList.

VAR-2.E.2 — Deleting elements during a traversal of an ArrayList requires using special techniques to avoid skipping elements.

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.

VAR-2.F — Traverse the elements in an ArrayList object using an enhanced for loop.

VAR-2.F.1 — When using an enhanced for loop you cannot add or remove elements in an ArrayList. Changing the size of an ArrayList during an enhanced for loop results in a ConcurrentModificationException being thrown.

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. EXCLUSION STATEMENT(EK VAR-2.F.1): 2D array objects that are not rectangular are outside the scope of the course and AP Exam.

VAR-2.F.2 — For the purposes of the exam, when accessing the element at ar?, the first index is used for rows, the second index is used for columns.

VAR-2.F.3 — The initializer list used to create and initialize a 2D array consists of initializer lists that represent 1D arrays.

VAR-2.F.4 — The square brackets? are used to access and modify an element in a 2D array.

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.

VAR-2.G — For 2D array objects: Traverse using nested for loops. Traverse using nested enhanced for loops.

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.

VAR-2.G.2 — Nested iteration statements can be written to traverse the 2D array in row-major order or column-major order.

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.