Unit 1 - Primitive Types and Variables

- **Compiler** - Software that translates the Java source code (ends in .java) into the Java class file (ends in .class).
- **Compile time error** - An error that is found during the compilation. These are also called syntax errors.
- **Main Method** - Where execution starts in a Java program.
- **Boolean** - An expression that is either `true` or `false`.
- **Camel Case** - One way to create a variable name by appending several words together and uppercasing the first letter of each word after the first word (`myScore`).
- **Casting a Variable** - Changing the type of a variable using *(type) name*.
- **Double** - A type in Java that is used to represent decimal values like -2.5 and 323.203.
- **Declare a Variable** - Specifying the type and name for a variable. This sets aside memory for a variable of that type and associates the name with that memory location.
- **Initializing a Variable** - The first time you set the value of a variable.
- **Integer** - A whole number like -32 or 6323.
- **modulo** - The `%` operator which returns the remainder from one number divide by another.
- **Operator** - Common mathematical symbols such as `+` for addition and `*` for multiplication.
- **Shortcut Operators** - Operators like `x++` which means `x = x + 1` or `x *=y` which means `x = x * y`.
- **Variable** - A name associated with a memory location in the computer.

Keywords

- **boolean** - used to declare a variable that can only have the value `true` or `false`.
- **double** - used to declare a variable of type double (a decimal number like 3.25).
- **false** - one possible value for a boolean variable.
- **int** - used to declare a variable of type integer (a whole number like -3 or 235).
- **static** - means that the field or method exists in the object that defines the class.
- **true** - one possible value for a boolean variable.

Common Mistakes

- forgetting that Java is case sensitive - `myScore` is not the same as `myscore`.
- forgetting to specify the type when declaring a variable (using `name = value;` instead of `type name = value;`)
- using a variable name, but never declaring the variable.
- using the wrong name for the variable. For example calling it `studentTotal` when you declare it, but later calling it `total`.

- using the wrong type for a variable. Don't forget that using integer types in calculations will give an integer result. So either cast one integer value to double or use a double variable if you want the fractional part (the part after the decimal point).
- using `==` to compare double values. Remember that double values are often an approximation. You might want to test if the absolute value of the difference between the two values is less than some amount instead.
- assuming that some value like 0 will be smaller than other `int` values. Remember that `int` values can be negative as well. If you want to set a value to the smallest possible `int` values use `Integer.MIN_VALUE`.

Unit 2 - Using Objects

- **class** - defines a new data type. It is the formal implementation, or blueprint, of the *attributes* and *behaviors* of the objects of that class.
- **object** - a specific instance of a class with defined attributes. Objects are declared as variables of a class type.
- **constructors** - code that is used to create new objects and initialize the object's attributes.
- **new** - keyword used to create objects with a call to one of the class's constructors.
- **instance variables** - define the attributes for objects.
- **methods** - define the behaviors or functions for objects.
- **dot (.) operator** - used to access an object's methods.
- **parameters (arguments)** - the values or data passed to an object's method inside the parentheses in the method call to help the method do its job.
- **return values** - values returned by methods to the calling method.
- **immutable** - String methods do not change the String object. Any method that seems to change a string actually creates a new string.
- **wrapper classes** - classes that create objects from primitive types, for example the Integer class and Double class.

Keywords

- **new** is used to create a new object.
- **null** is used to indicate that an object reference doesn't refer to any object yet.
- The following String methods and constructors, including what they do and when they are used, are part of the Java Quick Reference in the AP exam:
    - **String(String str)** : Constructs a new String object that represents the same sequence of characters as str.
    - **int length()** : returns the number of characters in a String object.
    - **String substring(int from, int to)** : returns the substring beginning at index from and ending at index (to – 1).
    - **String substring(int from)** : returns substring(from, length()). A string identical to the single element substring at position index can be created by calling substring(index, index + 1

    - **int indexOf(String str)** : returns the index of the first occurrence of str; returns -1 if not found.
    - **boolean equals(String other)** : returns true if this (the calling object) is equal to other; returns false otherwise.
    - **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.
- The following Integer methods and constructors, including what they do and when they are used, are part of the Java Quick Reference.
    - Integer(value): Constructs a new Integer object that represents the specified int value.
    - Integer.MIN_VALUE : The minimum value represented by an int or Integer.
    - Integer.MAX_VALUE : The maximum value represented by an int or Integer.
    - int intValue() : Returns the value of this Integer as an int.
- The following Double methods and constructors, including what they do and when they are used, are part of the Java Quick Reference Guide given during the exam:
    - Double(double value) : Constructs a new Double object that represents the specified double value.
    - double doubleValue() : Returns the value of this Double as a double.
- The following static Math methods are part of the Java Quick Reference:
    - **int abs(int)** : Returns the absolute value of an int value (which means no negatives).
    - **double abs(double)** : Returns the absolute value of a double value.
    - **double pow(double, double)** : Returns the value of the first parameter raised to the power of the second parameter.
    - **double sqrt(double)** : Returns the positive square root of a double value.
    - **double random()** : Returns a double value greater than or equal to 0.0 and less than 1.0 (not including 1.0)!
- **(int)(Math.random()*range) + min** moves the random number into a range starting from a minimum number. The range is the **(max number - min number + 1)**. For example, to get a number in the range of 5 to 10, use the range 10-5+1 = 6 and the min number 5: (int)(Math.random()*6) + 5).

Common Mistakes

- Forgetting to declare an object to call a method from main or from outside of the class, for example object.method();
- Forgetting () after method names when calling methods, for example object.method();
- Forgetting to give the right parameters in the right order to a method that requires them.
- Forgetting to save, print, or use the return value from a method that returns a value, for example int result = Math.pow(2,3);

- Using `==` to test if two strings or objects are equal. This is actually a test to see if they refer to the same object. Usually you only want to know if they have the same characters in the same order. In that case you should use `equals` or `compareTo` instead.
- Treating upper and lower case characters the same in Java. If `s1 = "Hi"` and `s2 = "hi"` then `s1.equals(s2)` is false.
- Thinking that substrings include the character at the last index when they don't.
- Thinking that strings can change when they can't. They are immutable.
- Trying to invoke a method like `indexOf` on a string reference that is null. You will get a null pointer exception.

Unit 3 - Boolean Expressions and If Statements

- **Block of statements** - One or more statements enclosed in an open curly brace '{' and a closing curly brace '}'.
- **Boolean expression** - A mathematical or logical expression that is either true or false.
- **complex conditional** - A Boolean expression with two or more conditions joined by a logical and '&&' or a logical or '||'.
- **conditional** - Used to execute code only if a Boolean expression is true.
- **DeMorgan's Laws** - Rules about how to distribute a negation on a complex conditional.
- **logical and** - Used to only execute the following statement or block of statements if both conditions are true
- **logical or** - Used to execute the following statement or block of statements if one of the conditions are true
- **negation** - turns a true statement false and a false statement true
- **short circuit evaluation** - The type of evaluation used for logical and '&&' and logical or '||' expressions. If the first condition is false in a complex conditional with a logical and the second condition won't be evaluated. If the first condition is true is a complex conditional with a logical or the second condition won't be evaluated.

Keywords

- **if (Boolean expression)** - used to start a conditional statement. This is followed by a statement or a block of statements that will be executed if the Boolean expression is true.
- **else** - used to execute a statement or block of statements if the Boolean expression on the if part was false.
- **else if (Boolean expression)** - used to have 3 or more possible outcomes such as if x is equal, x is greater than, or x is less than some value. It will only execute if the condition in the 'if' was false and the condition in the else if is true.

Common Mistakes

- Using = instead of == in `if`'s. Remember that = is used to assign values and == is used to test. Ifs always use ==.
- Putting a ; at the end of `if (test);`. Remember that the if statement ends after `if (test) statement;` or use curly brackets `if (test) { statements; }`.
- Using two `if`'s one after the other instead of an `if` and `else`.
- Trouble with complex conditionals which are two or more Boolean expressions joined by `&&` or `||`.
- Not understanding that || is an inclusive-or where one or *both* conditions must be true.
- Trouble with understanding or applying negation (`!`). See the section on DeMorgan's Laws.
- Not understanding short circuit evaluation which is that if evaluation of the first Boolean expression is enough to determine the truth of a complex conditional the second expression will not be evaluated.

Unit 4 - Iteration and Loops

- **Body of a Loop** - The single statement or a block of statements that *can* be repeated (a loop may not execute at all if the condition is false to start with). In Java the body of the loop is either the first statement following a `while` or `for` loop is the body of the loop or a block of statements enclosed in `{` and `}`.
- **For Loop** - A loop that has a header with 3 optional parts: initialization, condition, and change. It does the initialization one time before the body of the loop executes, executes the body of the loop if the condition is true, and executes the change after the body of the loop executes before checking the condition again.
- **Infinite Loop** - A loop that never ends.
- **Loop** - A way to repeat one or more statements in a program.
- **Nested Loop** - One loop inside of another.
- **Out of Bounds error** - A run-time error that occurs when you try to access past the end of a string or list in a loop.
- **Trace Code** - Writing down the values of the variables and how they change each time the body of the loop executes.
- **While Loop** - A loop that repeats while a Boolean expression is true.

Keywords

- **while** - used to start a while loop
- **for** - used to start a for loop or a for each loop
- **System.out.println(variable)** - used to print the value of the variable. This is useful in tracing the execution of code and when debugging.

Common Mistakes

- Forgetting to change the thing you are testing in a `while` loop and ending up with an infinite loop.
- Getting the start and end conditions wrong on the `for` loop. This will often result in you getting an **out of bounds error**. An **out of bounds** error occurs when you try to access past the end of a string.
- Jumping out of a loop too early by using one or more return statements inside of the loop.

Unit 5 - Writing Classes

- **Class** - A class defines a type and is used to define what all objects of that class know and can do.
- **Compiler** - Software that translates the Java source code (ends in .java) into the Java class file (ends in .class).
- **Compile time error** - An error that is found during the compilation. These are also called syntax errors.
- **Constructor** - Used to initialize fields in a newly created object.
- **Field** - A field holds data or a property - what an object knows or keeps track of.
- **Java** - A programming language that you can use to tell a computer what to do.
- **Main Method** - Where execution starts in a Java program.
- **Method** - Defines behavior - what an object can do.
- **Object** - Objects do the actual work in an object-oriented program.
- **Syntax Error** - A syntax error is an error in the specification of the program.

Keywords

- **class** - used to define a new class
- **public** - a visibility keyword which is used to control the classes that have access. The keyword public means the code in any class has direct access.
- **private** - a visibility keyword which is used to control the classes that have access. The keyword private means that only the code in the current class has direct access.

Common Mistakes

- Forgetting to declare an object to call its methods.
- Forgetting to write get/set methods for private instance variables.
- Forgetting to write a constructor.
- Mismatch in name, number, type, order of arguments and return type between the method definition and the method call.
- Forgetting data types for every argument in the method definition.
- Forgetting to use what the method returns.

Unit 6 - Arrays

- **Array** - An array can hold many items (elements) of the same type. You can access an item (element) at an index and set an item (element) at an index.
- **Array Declaration** - To declare an array specify the type of elements that will be stored in the array, then (`[]`) to show that it is an array of that type, then at least one space, and then a name for the array. Examples: `int[] highScores;` `String[] names;`
- **Array Creation** - To create an array type the name and an equals sign then use the *new* keyword, followed by a space, then the type, and then in square brackets the size of the array (the number of elements it can hold). Example: `names = new String[5];`
- **Array Index** - You can access and set values in an array using an index. The first element in an array called `arr` is at index 0 `arr[0]`. The last element in an array is at the length minus one - `arr[arr.length - 1]`.
- **Array Initialization** - You can also initialize (set) the values in the array when you create it. In this case you don't need to specify the size of the array, it will be determined from the number of values that you specify. Example: `int[] highScores = {99,98,98,88,68};`
- **Array Length** - The length of an array is the number of elements it can hold. Use the public `length` field to get the length of the array. Example: given `int[] scores = {1,2,2,1,3,1};`, `scores.length` equals 6.
- **Element Reference** - A specific element can be referenced by using the name of the array and the element's index in square brackets. Example: `scores[3]` will return the 4th element (since index starts at 0, not 1). To reference the last element in an array, use `array[array.length - 1]`
- **For-each Loop** - Used to loop through all elements of an array. Each time through the loop the loop variable will be the next element in the array starting with the element at index 0, then index 1, then index 2, etc.
- **Out of Bounds Exception** - An error that means that you tried to access an element of the array that doesn't exist maybe by doing `arr[arr.length]`. The first valid indices is 0 and the last is the length minus one.

Keywords

- **for** - starts both a general for loop and a for-each loop. The syntax for a for each loop is `for (type variable : array)`. Each time through the loop the variable will take on the next value in the array. The first time through the loop it will hold the value at index 0, then the value at index 1, then the value at index 2, etc.
- **static** - used to create a class method, which is a method that can be called using the class name like `Math.abs(-3)`.

Common Mistakes

- forgetting to create the array - only declaring it (`int[ ] nums;`)
- using 1 as the first index not 0
- using `array.length` as the last valid index in an array, not `array.length - 1`.
- using `array.length()` instead of `array.length` (not penalized on the free response)
- using `array.get(0)` instead of `array[0]` (not penalized on the free response)
- going out of bounds when looping through an array (using `index <= array.length`). You will get an `ArrayIndexOutOfBoundsException`.
- jumping out an loop too early by using one or more return statements before every value has been processed.

Unit 7 - ArrayLists

- **Autoboxing** - Automatically wrapping a primitive type in a wrapper class object. For instance if you try to add an `int` value to a list, it will automatically be converted to an `Integer` object.
- **Abstract Method** - A method that only has a declaration and no method body (no code inside the method).
- **ArrayList** - An ArrayList can hold many objects of the same type. It can grow or shrink as needed. You can add and remove items at any index.
- **Add** - You can add an object to the end of a list using `listName.add(obj)`. You can add an object at an index of a list using `add(index,obj)`. This will first move any objects at that index or higher to the right one position to make room for the new object.
- **Declaration** - To declare an ArrayList use `ArrayList<Type> name`, where `Type` is the class name for the type of objects in the list. If you leave off the `<Type>` it will default to `Object`.
- **Creation** - To create an ArrayList use `new ArrayList<Type>`, where `Type` is the class name for the type of objects you want to store in the list. There are other classes that implement the `List` interface, but you only need to know the `ArrayList` class for the exam.
- **Get** - To get an object at an index from a list use `listName.get(index)`.
- **Index** - You can access and set values in a list using an index. The first element in a list called `list1` is at index 0 `list1.get(0)`. The last element in a list is at the length minus one - `list1[list1.size() - 1]`.
- **Remove** - To remove the object at an index use `ListName.remove(index)`. This will move all object past that index to the left one index.
- **Set** - To set the value at an index in a list use `listName.set(index,obj)`.
- **Size** - Use `listName.size()` to get the number of objects in the list.
- **Wrapper Class** - Classes used to create objects that hold primitive type values like `Integer` for `int`, `Double` for `double` and `Boolean` for `boolean`.
- **Unboxing** - Automatically converting a wrapper object like an `Integer` into a primitive type such as an `int`.

Common Mistakes

- forgetting that `set` replaces the item at the index
- forgetting that `remove` at an index moves all items that were to the right of that index left one index
- forgetting that `add` at an index moves everything that was at the index and greater to the right one index
- incrementing an index when looping through a list even though you removed an item from the list
- using `nameList[0]` instead of `nameList.get(0)`.
- using `nameList.length` instead of `nameList.size()` to get the number of elements in a list

---

THIS INFO BELOW WILL NOT BE COVERED IN THE 2020 APCS EXAM

Unit 8 - 2D Arrays

- **2d Array** - An array that holds items in a two dimensional grid. You can think of it as storing items in rows and columns (like a bingo card or battleship game). You can access an item (element) at a given row and column index.
- **2d Array Declaration** - To declare an array, specify the type of elements that will be stored in the array, then (`[][]`) to show that it is a 2d array of that type, then at least one space, and then a name for the array. Examples: `int[][] seats;` `String[][] seatingChart;`
- **2d Array Creation** - To create a 2d array, type the name and an equals sign then use the *new* keyword, followed by a space, then the type, and then [numRows][numCols]. Example: `seatingChart = new String[5][4];`. This will have 5 rows and 4 columns.
- **2d Array Index** - You can access and set values in a 2d array using the row and column index. The first element in an array called `arr` is at row 0 and column 0 `arr[0][0]`.
- **2d Array Initialization** - You can also initialize (set) the values in the array when you first create it. In this case you don't need to specify the size of the array, it will be determined from the number of values that you specify. Example: `String[][] seatingInfo = { {"Jamal", "Maria"}, {"Jake", "Suzy"}, {"Emma", "Luke"}};` This will create a 2d array with 3 rows and 2 columns.
- **2d Array Number of Rows** - The number of rows (or height) is the length of the outer array. For an array `arr` use `arr.length` to get the number of rows in the array.
- **2d Array Number of Columns** - The number of columns (or width) is the length of the inner array. For an array `arr` use `arr[0].length` to get the number of columns.

- **nested for loop** - A for loop inside of another for loop. These are used to loop through all the elements in a 2d array. One loop can work through the rows and the other the columns.
- **out of bounds error** - This happens when a loop goes beyond the last valid index in an array. Remember that the last valid row index is `arr.length - 1`. The last valid column index is `arr[0].length - 1`.

Common Mistakes

- forgetting to create the array - only declaring it (`int[][] nums;`).
- using 1 as the first index not 0 for rows and/or columns.
- using `array.length` as the last valid row index, not `array.length - 1`.
- using `array[0].length` as the last valid column index, not `array[0].length - 1`.
- using `array.length()` instead of `array.length` (not penalized on the free response)
- going out of bounds when looping through an array (using `index <= array.length`). You will get an `ArrayIndexOutOfBoundsException`.
- jumping out an loop by using one or more return statements before every value has been processed.
- using the wrong starting and ending indicies on loops.
- using `array.length` for both the number of rows and columns. Use `array[0].length` for the number of columns.

Unit 9 - Inheritance

- **object** - Objects do the action in an object-oriented program. An object can have things it knows (attributes) and things it can do (methods). An object is created by a class and keeps a reference to the class that created it.
- **class** - A class defines what all objects of that class know (attributes) and can do (methods). You can also have data and behavior in the object that represents the class (class instance variables and methods). All objects of a class have access to class instance variables and class methods, but these can also be accessed using `className.variable` or `className.method()`.
- **inheritance** - One class can inherit object instance variables and methods from another. This makes it easy to reuse another class by extending it (inheriting from it). This is called specialization. You can also pull out common instance variables and/or methods from several related classes and put those in a common parent class. This is called generalization.
- **polymorphism** - The runtime type of an object can be that type or any subclass of the declared type. All method calls are resolved starting with the class that created the object. If the method isn't found in the class that created the object, then it will look in the parent class and keep looking up the inheritance tree until it finds the method. The method must exist, or the code would not have complied.

- **parent class** - One class can inherit from another and the class that it is inheriting from is called the parent class. The parent class is specified in the class declaration using the `extends` keyword followed by the parent class name.
- **child class** - The class that is doing the inheriting is called the child class. It inherits access to the object instance variables and methods in the parent class.
- **subclass** - A child class is also called a subclass.
- **superclass** - A parent class is also called a superclass.
- **declared type** - The type that was used in the declaration. `List aList = new ArrayList()` has a declared type of `List`. This is used at compile time to check that the object has the methods that are being used in the code.
- **run-time type** - The type of the class that created the object. `List aList = new ArrayList()` has a run-time type of `ArrayList`. This is used at run-time to find the method to execute.
- **overrides** - A child class can have the same method signature (method name and parameter list) as a parent class. Since methods are resolved starting with the class that created the object, that method will be called instead of the inherited parent method, so the child method overrides the parent method.
- **overload** - At least two methods with the same name but different parameter lists. The parameter lists can differ by the number of parameters and/or the types.
- **getter** - A method that returns the value of an instance variable in an object.
- **setter** - A method that sets the value of am instance variable in an object.
- **accessor** - Another name for a getter method - one that returns the value of a instance variable.
- **mutator** - Another name for a setter method - one that changes the value of a instance variable.

Keywords

- **extends** - Used to specify the parent class to inherit from. It is followed by the name of the parent class, like this: `public class ChildName extends ParentName`. If no `extends` keyword is used in the class declaration, then the class will automatically inherit from the `Object` class.
- **static** - Keyword used to indicate that a instance variable or method is part of the class and not part of each object created by the class.
- **super** - Keyword used to call a method in a parent class. This is useful if a child class overrides an inherited method, but still wants to call it.

Common Mistakes

- Using inheritance (is a kind of) when you should use association (has a). A school has classes, it is not a type of class. A high school is a kind of school.

- Using an instance variable for a type of class instead of subclasses. If you ever find yourself creating conditionals based on the type of object use subclasses instead.
- Copying code instead of creating a subclass or pulling out a common superclass. If you ever find yourself copying object attributes or methods try creating a subclass instead or pull out a common superclass.

Unit 10 - Recursion

- **base case** - A way to stop the recursive calls. This is a return without a recursive call.
- **call stack** - A class defines what all objects of that class know (fields) and can do (methods). You can also have data and behavior in the object that represents the class (class fields and methods). All objects of a class have access to class fields and class methods, but these can also be accessed using `className.field` or `className.method()`.
- **recursive method** - A method that contains at least one call to itself inside the method.

Common Mistakes

- Missing the recursive call. Be sure to look for a call to the same method.
- Getting confused about when a recursive method returns and what it returns.
- Assuming you understand what the recursion is doing without tracing all of it.