Week 3 – Topics

- Arrays
- Methods
- Objects
- JavaDocs
- Equality in Primitives vs Objects
- Menu App Example

Arrays

Arrays

An array in Java is a container which holds a fixed number of values of a single type. The length of an array is set at the time that it is declared or instantiated, and remains fixed for the life of an array. Additionally the type contained in an array can be a primitive datatype, or an Object type.

Some useful array information and vocabulary is contained here:

- Square Brackets or [] tell Java that we are declaring an array
- element: each item stored in an array is called an element
- elements within the same Java array must all be the same data type.
- index: each element in an array is found or accessed by its numerical index.
- arrays are zero-based, so the index values start at 0.
- the first element in any array is located at index: 0
- the last element of any array is located at index: arrayName.length – 1

Just like any variable in Java, an array must be declared. We declare the array to give it a name, and to allocate space for the size of the array that we declare. Note, that we also must determine the data type of the elements that will be stored in this array, so that Java knows how to store and manage the elements in the array.

Content Link in LMS

https://github.com/ckiefriter1/Java-Code-Examples/tree/main/src/arrays

https://github.com/ckiefriter1/Java-Code-Examples/blob/main/src/arrays/ArrayDemo.java

https://github.com/ckiefriter1/Java-Code-Examples/blob/main/src/arrays/MultiDimArrayDemo.java

More Arrays

- Creating and using Arrays of Objects
- Defining Classes and instantiating Objects from Classes

```java
package arrays;

public class Student {

    String fullName = "";
    int[] grades;

    /*
     * Constructor for Student class.
     */
    public Student(String fName,int[] grades) {
        this.fullName = fName;
        this.grades = grades;
    }

    /*
     * publicly accessible method named describe, void means this method does not return and data.
     */
    public void describe() {

        System.out.println("Student: " + this.fullName);
        System.out.println("Grades:");

        for (int grade : this.grades) {
            System.out.println("\t" + grade + " ");
        }

        System.out.println();
    }
}
```

```java
package arrays;

import java.util.Scanner;

public class GradeBook {

    public static void main(String[] args) {
        String fullName = "";
        String lineVariable = "----------------------------------------";
        Scanner sc = new Scanner(System.in);

        System.out.println("** Grade Book Example**");

        /*
         * User input the name of the course.
         */
        System.out.println("Name of Course:");
        String courseName = sc.nextLine();

        /*
         * User input how many students are in the class.
         */
        System.out.println("How many students are in this class:");
        int numOfStudents = sc.nextInt();

        /*
         * Declare and instantiate (create) an array of student objects.
         *
         * -> Notice you didn't have to import the Student class into this code becaus
         */
        Student[] studentList = new Student[numOfStudents];

        /*
         * User input the number of grades for each student.
         */
```

https://github.com/ckiefriter1/Java-Code-Examples/blob/main/src/arrays/Student.java

Content Link in LMS

https://github.com/ckiefriter1/Java-Code-Examples/blob/main/src/arrays/GradeBook.java

Methods

Methods

Java uses methods to accomplish predefined tasks. A method is code that runs when it is called. When you write a program and realize that you are writing the same code over and over, it's time to put that code into a method. Another definition of a Java Method is that it is a collection of statements that are written together and executed together to perform a task.

What is a method declaration?

```
modifier returnDatatype methodName (datatype1 par1, datatype2 par2) {
    BodyOfMethod
}
```

Here, these parts of the method declaration are explained below:

- *modifier* − modifier -- *public, protected, default* and *private*
- *returnDatatype* − the return datatype (Can be a Java primitive datatype, an Object, or a Collection of a datatype or an Object, or can be void).
- *methodName* − name of the method
- *datatype1, datatype2* - datatypes of the parameters
- *par1, par2* − formal parameter names
- *datatype1 par1, datatype2 par2* − list of parameters (Can be as many as you need, separated by commas)
- *BodyofMethod* —> this is where you put your code... anything that you want to accomplish in that particular method. All Java code, and will include variable declarations, method calls, etc.

https://docs.oracle.com/javase/tutorial/java/javaOO/methods.html

https://github.com/ckiefriter1/Java-Code-Examples/tree/main/src/methods

Objects

Primitives vs Objects

A primitive datatype is just a piece of data and nothing more.

```
int age = 19;
System.out.println("Age is: " + age);
```

An Object (e.g. String, Array, etc..) has a value, but there is much more. An Object has properties and methods which are defined on that object, and are accessed via dot-notation. For example, we can declare a String name, which is an object, and use the method name.length() to print out the length of the String stored in the variable name

```
String name = "Chip Brown";
System.out.println("The length of name is: " + name.length());
```

The main difference between a primitive datatype and an Object is that a primitive datatype does not have properties and methods defined on it.

When creating your own Objects, you can define them as you need. For example in our GradeBook example from the Array Section, we needed a Student Object, so we declared a Class named Student. Student has two fields: fullName and grades, a Constructor named Student(), and a method named describe().

Additionally, you will notice that Student is a Class. In Java, a Class is the template from which an Object can be created, and an Object is an instance of that class. We use the word "instantiate" when we describe the creation of an Object from a Class by use of the Contructor of that Class.

JavaDocs

JavaDocs

Documenting your code is crucial to help others understand it, and even to remind yourself how your own older programs work. Unfortunately, it is easy for most external documentation to become out of date as a program changes.

For this reason, it is useful to write documentation as comments in the code itself, where they can be easily updated with other changes. Javadoc is a documentation tool which defines a standard format for such comments, and which can generate HTML files to view the documentation from a web broswer. (As an example, see Oracle's Javadoc documentation for the Java libraries

http://download.oracle.com/javase/6/docs/api/

/** Indicates JavaDoc comment */

Use tags in comments to indicate JavaDoc parts

@param – defines parameters for input
@return – indicates what value and type are return from method call
@author (classes and interfaces only, required)
@version (classes and interfaces only, required. See footnote 1)
@param (methods and constructors only)
@return (methods only)
@exception (@throws is a synonym added in Javadoc 1.2)
@see
@since
@serial (or @serialField or @serialData)
@deprecated (see How and When To Deprecate APIs)

https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

https://docs.oracle.com/javase/6/docs/api/java/util/Scanner.html

Equality in Primitives vs Objects

Equality in Primitives vs Objects

- Another difference between Primitive Datatypes and Objects has to do with Equality. The Equality Operator in Java == checks to see if two references are the same. In other words when comparing Primitive Datatypes, == checks to see if two values are the same. With Primitive Datatype Equality Comparison, == checks the in-memory value of the Primitive Datatype against the in-memory value of a different Primitive Datatype.

- Primitive Datatype Equality

- The following code declares two int variables, and compares the value. Since these two variables are pointing to the same value, the System.out.println() will print the following result: age1 == age2: true

- Object Equality

- Remember that when an Object is instantiated, the programmer has access to all properties and methods that are defined within that Object, through dot-notation.

- With the declaration of a new Object, Java creates that object as its own instance in memory. Even if two Objects are created with the same exact content, the Objects themselves will be created as two separate instances in memory. The point here is that each instance has its own location in memory, and the location is not the same, even if the values within the fields are the same.

- Let's look at our Grade Book Example above. Imagine that we instantiate two students using the Student() Constructor as follows:

Menu-Driven Applications

Menu-Driven Applications

Menu-driven Applications are very useful in the coding world. They allow a program to receive data directly from a user. The user is provided a menu, and then is instructed to select an option from that menu. The chosen option will be used by the program to execute a code path (or branch) specific to that option.
In the Back End, the menus we create are text based. Adding a Client, or Front End program, which reads in user data, and then communicates to a Back End Server could provide a graphical user interface for the Menu.
Menu-driven applications are used in a variety of industries, including but not limited to computing, application development, banking (ATMs), websites, tablets, self-guided machines, word-processors, gaming, and more

Menu-Driven Application:
Advantages:
•User-friendly
- Provide guidance to the user
- No need for a user to remember commands
•Allow a user to control how and in what order a program executes

Disadvantages:
•Difficulty finding content, especially with nested sub-menus

https://github.com/ckiefriter1/Java-Code-Examples/tree/main/src/menus

Appendix