# 4. Objects, JavaDocs, and Equality

## Objects and JavaDocs

Primitive Datatypes 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.

Here is our `Student` Class declaration again:

```
public class Student {
    String fullName = "";
    int[] grades;

    public Student(String fName,int[] grades) {
        this.fullName = fName;
        this.grades = grades;
    }

    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();
    }
}
```

In this previous example, we also use a Scanner, which is also an Object.  Notice the syntax for Scanner declaration below. And here is how we created an Object of type Student by instantiating a Student:

```
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");
        System.out.println("Name of Course:");
        String courseName = sc.nextLine();
        System.out.println("How many students are in this class:");
        int numOfStudents = sc.nextInt();
        Student[] programmingStudents = new Student[numOfStudents];
        System.out.println("How many grades do you have per student?");
        int numOfGrades = sc.nextInt();
        sc.nextLine();

        for (int i = 0; i < programmingStudents.length; i++) {
            int[] grades = new int[numOfGrades];
            System.out.print("Enter Student's Full Name: ");
            fullName = sc.nextLine();
            for (int j = 0; j < grades.length; j++) {
                System.out.print("Enter a grade: ");
                grades[j] = sc.nextInt();
            }
            programmingStudents[i] = new Student(fullName, grades);
            sc.nextLine();
        }

        System.out.println("\n\n"+lineVariable+"\n "
               + courseName + " Grade Book \n"+lineVariable);
        for (Student student : programmingStudents) {
            student.describe();
        }
        System.out.println(lineVariable+"\n");
        sc.close();
    }
}
```

## Equality

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`

```
int age1 = 21;
int age2 = 21;
System.out.println("age1 == age2: " + (age1 == age2));
```

## 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:

```java
int[] grades = { 100, 100, 100 };
Student student1 = new Student("Molly Mack", grades);
Student student2 = new Student("Molly Mack", grades);
```

Notice that the data is exactly the same, but we are creating two distinct **Objects** by using the `new` keyword. So, if we run the following comparisons, the results will be reflect that the student1 Object is not the same as the student2 Object, because their locations in memory are distinct. These are **Objects**, not **Primitive Datatypes**. The **only one** of the following examples that will be `true` is when we compare the `student1.fullName` to "Molly Mack" with the `.equals()` method.

```java
System.out.println("student1 vs. student2 Equality Example:");
System.out.println("----------------------------------------");
System.out.println("Is student1 == student2? "
                   + (student1 == student2));
System.out.println("Is student1.equals(student2)? "
                   + (student1.equals(student2)));
System.out.println("Is student1.equals(\"Molly Mack\")? "
                   + (student1.equals("Molly Mack")));
System.out.println("Is student1.fullName.equals(\"Molly Mack\")? "
                   + (student1.fullName.equals("Molly Mack")));
```

**Object Equality** is fun to explore, and when you assign two variables to the same **Object**, it's fun to see what the result is. Let's declare a new variable `student3` and initialize it with the `student2` **Object** as follows:

```java
Student student3 = student2;
```

The results of the following comparisons will be quite different than the `student1` vs. `student2` comparisons above. In fact, the results will be `true` for each comparison:

```java
System.out.println("student2 vs. student3 Equality Example:");
System.out.println("----------------------------------------");
System.out.println("Is student2 == student3? "
                   + (student2 == student3));
System.out.println("Is student2.equals(student3)? "
                   + (student2.equals(student3)));
```

**Put your knowledge to the test**: Why is the result different? Spend some time doing research into Object Equality in Java!