Static Variables and Methods

Now that we've talked a bit about instance methods, we can get back to the static ones. So far, we've explored the `Math` class and its many static methods like `Math.random()`. In doing so, we've always used a `main` method which is static. In this lesson, you will learn to write your own static variables and methods.

- **Static** variables and methods belong to a **class** and are called with the Class's name rather than using object variables, as in `ClassName.methodName();`
- If you are calling a static method from within the same class, you can leave off the ClassName and dot `.` notation and just call the `methodName()` followed by parentheses.
- There is only one copy of a static variable or method for the *whole* class. For example, the `main` method is static because there should **only be one** `main` method ever in a given package/program.
- Static methods can be `public` or `private`.
- The `static` keyword is placed right after the `public`/`private` access modifier and right before the types of variables and methods in their declarations.
- A static method does not have a `this` variable, since it is **not** called with an object. Essentially, there is no "this" to refer to.

```
class ClassName {
// static variable
public static type variableName;

// static method
public static returnType methodName(parameters) {
// implement the logic for your method here
}
}
// To call a static method or variable, use the Class Name
System.out.println(ClassName.staticVariable);
ClassName.staticMethod();
```

`static` methods only have access to other `static` variables and `static` methods (unless the static method *creates an object* through which to access instance variables and methods). **Static methods cannot access or change the values of instance variables or use the** `this` **reference (since there is no calling object for them)**, **and static methods cannot call non-static methods**. **However, non-static**

methods have access to all variables (instance or static) and methods (static or non-static) in the class.

Since there is only one copy of a static variable or method, static variables are often used to count how many objects are generated. In the following `Person` class, there is a static variable called `personCounter` that is incremented each time the `Person` constructor is called to initialize a new `Person` object. The static method `printPersonCounter()` prints out its value. You can also watch how it works in the Java visualizer by clicking the link below the IDE.

What will the following code print out? Try adding another `Person` object and see what happens.

Run

Person.java

```
public class Person {
   // instance variables
   private String name;
   private String email;
   private String phoneNumber;

   // Static counter variable
   public static int personCounter = 0;

   // static method to print out counter
   public static void printPersonCounter() {
        System.out.println("Person counter: " + personCounter);
   }

   // constructor: construct a Person copying in the data into the instance va
riables
   public Person(String initName, String initEmail, String initPhone) {
      name = initName;
      email = initEmail;
      phoneNumber = initPhone;
      personCounter++;
   }

   // toString() method
   public String toString() {
     return  name + ": " + email + " " + phoneNumber;
   }
```

```
    // main method for testing
    public static void main(String[] args) {
        // call the constructor to create a new person
        Person p1 = new Person("Sana", "sana@gmail.com", "123-456-7890");

        // add another person here
        Person p2 = new Person("Lily", "lily@gmail.com", "678694546464");
        Person.printPersonCounter();
    }
}
```

You can see this code in action in the Java visualizer.

Another common use for static variables is to keep track of a minimum or maximum value or an average of the values in a collection of objects.

#Check Your Understanding

Consider the class Temperature below which has a static variable. What is the output of the main method below?

```
public class Temperature {
private double temperature;
public static double maxTemp = 0;

public Temperature(double t) {
temperature = t;
if (t > maxTemp)
maxTemp = t;
}

public static void main(String[] args) {
Temperature t1 = new Temperature(75);
Temperature t2 = new Temperature(100);
Temperature t3 = new Temperature(65);
System.out.println("Max Temp: " + Temperature.maxTemp);
}
}
```

Max Temp: 0

There is a compiler error because the static variable maxTemp cannot be used inside a non-static constructor.

Max Temp: 100

Max Temp: 75

Max Temp: 65

#Coding Exercise

Fix the bugs in the following code.

Run

Temperature.java

```
public class Temperature {
    private double temperature;
    public static double maxTemp = Double.MIN_VALUE; // Initialize maxTemp wit
h a small value

    public Temperature(double t) {
        temperature = t;
        if (t > maxTemp)
            maxTemp = t;
    }

    public static void printMax() {
        System.out.println(maxTemp); // Print the maxTemp instead of temperatu
re
    }

    public static void main(String[] args) {
        Temperature t1 = new Temperature(75);
        Temperature t2 = new Temperature(100);
        Temperature.printMax();
    }
}
```

#Programming Challenge: Static Song and Counter

Notice that this is a class where there are no instance variables and we don't really need to generate multiple objects. Think back to previous examples we've dealt with: when working with students or pets, it made sense to have multiple objects. With the Song we saw before, we were able to make the chorus method static and have just one copy.

Add a static variable to the `Song` class below that keeps track of the number of verses. Increment this variable each time the method to print a `verse` is called and print it out. Update the main method to add a few more verses (pig says oink, chicken says cluck) and rerun the program.

Run

Song.java

```
public class Song {

  //add a static variable to count how many times the verse method is called


  //update the method to increment the counter
  public static void verse(String animal, String noise) {
    System.out.println( "Old MacDonald had a farm" );
    System.out.println( "E-I-E-I-O" );
    System.out.println( "And on that farm he had a " + animal );
    System.out.println( "E-I-E-I-O" );
    System.out.println( "With a " + noise + "-" + noise + " here") ;
    System.out.println( "And a " + noise + "-" + noise + " there" );
    System.out.println( "Here a " + noise + ", there a " + noise );
    System.out.println( "Everywhere a " + noise + "-" + noise );
    System.out.println( "Old MacDonald had a farm" );
    System.out.println( "E-I-E-I-O" );

    //increment the counter value
  }

  public static void main(String[] args) {
    verse( "cow" , "moo" );
    verse( "duck" , "quack" );
    //add a few more verses


    //print the counter value


  }
}
```

#Summary

- Static methods and variables include the keyword `static` before their name in the header or declaration. They can be public or private.
- Static variables belong to the class, with all objects of a class sharing that single static variable.
- Static methods are associated with the class, not objects of the class.
- Static variables are used with the class name and the dot `.` operator, since they are associated with a class, not objects of a class.
- Static methods **cannot** access or change the values of **instance** variables, but they can access or change the values of static variables contained within their class.
- Static methods cannot call non-static methods.

#Write a Program using static variables and methods

Create a Student class with the following:

1. Class Variables

- name (String)
- grade (int)
- principalName (String, static, default value: "Ms. McKoy")
- studentID (String)
- nextID (int, static, default value: 100)

InfoWarningTip

Note: `studentID` will be composed of the student's first initial in uppercase followed by the `nextID`

2. Constructors

- The `main` constructor should take in the student's name and grade only.
- When the `main` constructor is called, it generates a student's ID by combining the student's first initial in uppercase combined with the `nextID`. The `nextID` should then be increased by 1.

InfoWarningTip

For example: if the name is "Alfonso Lewis", and the static variable `nextID` is 105, the new `studentID` would be A105

3. Class Methods

- Write a static method called `newPrincipal` that returns nothing and takes in one String parameter that represents the new principal's name. Set the static variable `principalName` to the new principal's name.
- Write a static method called `resetID` that takes no arguments and returns nothing. This method resets the `nextID` to 100.
- Write a method called `toString` that has no parameters and returns a String. This method should return the student's name and the student's ID, separated by a space.
- The `main` method creates three instances of the `Student` class, prints out information about the students, updates the principal's name using the `newPrincipal` method, resets the student ID using the `resetID` method, and creates another student instance. The output of the program should be displayed in the console.

InfoWarningTip

Expected output:
Principal Name for student 1: Ms. McKoy

Student ID for student 1: M100

Student ID for student 2: A101

Student ID for student 3: S102

New principal name: Mr. McKoy

Student ID for student 4: K100

String representation of student 4: kevin K100

```
public class Student {
    // Declare the variables (instance and static)
    private String name;
    private int grade;
    private static String principalName = "Ms. McKoy";
    private String studentID;
    private static int nextID = 100;

    // Create Constructor
    public Student(String name, int grade) {
        this.name = name;
        this.grade = grade;
        this.studentID = Character.toUpperCase(name.charAt(0)) + String.valueO
f(nextID);
        nextID++;
```

```
    }

    // Create static method newPrincipal
    public static void newPrincipal(String newPrincipalName) {
        principalName = newPrincipalName;
    }

    // Create static method resetID
    public static void resetID() {
        nextID = 100;
    }

    // Create toString() method
    public String toString() {
        return name + " " + studentID;
    }

    // Do not modify the code below:
    public static void main(String[] args) {
        Student s1 = new Student("Muhammed", 11);
        Student s2 = new Student("Alan", 11);
        Student s3 = new Student("Sophie", 11);

        System.out.println("Principal Name for student 1: " + Student.principa
lName);
        System.out.println("Student ID for student 1: " + s1.studentID);
        System.out.println("Student ID for student 2: " + s2.studentID);
        System.out.println("Student ID for student 3: " + s3.studentID);

        Student.newPrincipal("Mr. McKoy");
        System.out.println("New principal name: " + Student.principalName);

        Student.resetID();
        Student s4 = new Student("Kevin", 11);
        System.out.println("Student ID for student 4: " + s4.studentID);
        System.out.println("String representation of student 4: " + s4.toStrin
g());
    }
}
```