CS1421: Object Oriented Programming Lab

LAB # 05

Instructors
Safiyah Batool
Samia Bashir

Lab 05

Static variables & Static methods

Objective

After completing this lab, the students should be able to

- Understand the initializing and usage of static variables
- Understand static methods

Static Members:

It is possible to create a field or method that does not belong to any instance of a class. Such members are known as static fields and static methods. When a value is stored in a static field, it is not stored in an instance of the class. In fact, an instance of the class doesn't even have to exist in order for values to be stored in the class's static fields. Likewise, static methods do not operate on the fields that belong to any instance of the class. Instead, they can operate only on static fields. You can think of static fields and static methods as belonging to the class instead of an instance of the class.

Static Fields:

When a field is declared with the key word static, there will be only one copy of the field in memory, regardless of the number of instances of the class that might exist. A single copy of a class's static field is shared by all instances of the class. For example: it can be used to keep count of the number of instances of the class that are created..

- The objective of using static field is efficient memory management
- It is a variable which **belongs to the class** and **not** to **object**(instance)
- Static variables are **initialized only once**, at the start of the execution. These variables will be initialized first, before the initialization of any instance variables
- A **single copy** to be shared by all instances of the class
- A static variable can be **accessed directly** by the **class name** and doesn't need any object
- Syntax: <***class-name*>.<*variable-name*>**

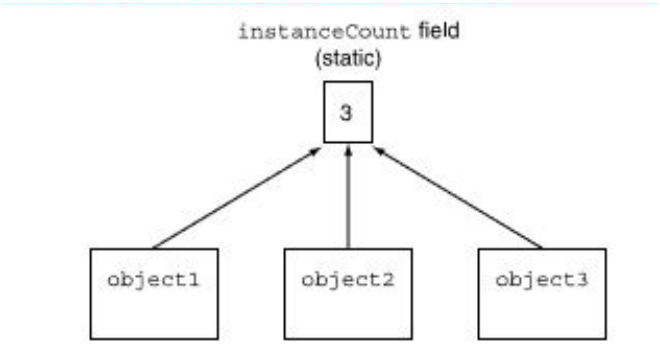
Example: The Countable class shown uses a static field to keep count of the number of instances of the class that are created.

```
// This class demonstrates a static field.
public class Countable   {
```

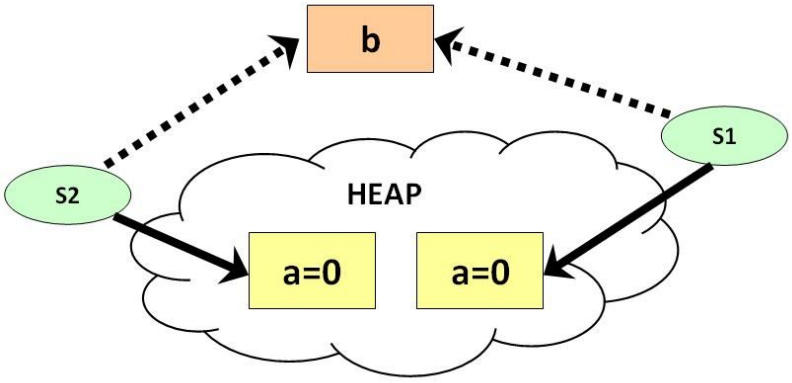
```
    private static int instanceCount = 0;

    /**The constructor increments the static field instanceCount. This keeps track
of the number of instances of this  class that are created.     */

 public Countable()     {
 instanceCount++;
    }

 /** The getInstanceCount method returns the number of instances of this class
that have been created. @return The value in the instanceCount field.  */
   public int getInstanceCount()    {
       return instanceCount;
}
}
```

A static field is created by placing the key word static after the access specifier and before the field's data type. Notice that we have explicitly initialized the instanceCount field with the value 0. This initialization takes place only once, regardless of the number of instances of the class that are created.

Java automatically stores 0 in all uninitialized static member variables. The instanceCount field in this class is explicitly initialized so it is clear to anyone reading the code that the field starts with the value 0.



Instances of the `Countable` class



Static Methods:

When a class contains a static method, it isn't necessary for an instance of the class to be created in order to execute the method.

```
1  /**
2     This class demonstrates static methods.
3  */
4
5  public class Metric
6  {
7     /**
8        The milesToKilometers method converts a
9        distance in miles to kilometers.
10       @param m The distance in miles.
11       @return The distance in kilometers.
12    */
13
14    public static double milesToKilometers(double m)
15    {
16       return m * 1.609;
17    }
18
19    /**
20       The kilometersToMiles method converts
21       a distance in kilometers to miles.
22       @param k The distance in kilometers.
23       @return The distance in miles.
24    */
25
26    public static double kilometersToMiles(double k)
27    {
28       return k / 1.609;
29    }
30 }
```

A static method is created by placing the key word static after the access specifier in the method header. The Metric class has two static methods: milesToKilometers and kilometersToMiles. Because they are declared as static, they belong to the class and may be called without any instances of the class being in existence. You simply write the name of the class before the dot operator in the method call.

- The objective of static method is convenience
- It is a method which **belongs to the class** and **not** to the **object**(instance)
- A static method **can access only static data**. It cannot access non-static data (instance variables)
- A static method **can call only** other **static methods** and cannot call a non-static method from it.
- A static method can be **accessed directly** by the **class name** and doesn't need any object
- Syntax : <***class-name>.<method-name>***

Example:

```
public class Cars{
   private static int noOfCars;
   private String color;
   private String model;

   StaticVariables(String color,String model){
       this.color=color;
       this.model=model;
       newCar();
   }
   public static int getNoOfCars(){
       return noOfCars;
   }
   public static void newCar(){
       noOfCars++;
   }
   public String getColor(){
       return color;
   }
   public String getModel(){
```

```
        return model;
    }

}

//TestClass
public class Test {
   public static void main(String[] args) {

        Cars car= new Cars("Green","AJ43");
        Cars car2= new Cars("Red","43os");

       System.out.println("Car 1 Color: "+car.getColor());
       System.out.println("Car 1 Model: "+car.getModel());
       System.out.println("No. of Cars: "+Cars.getNoOfCars());

      System.out.println("Car 2 Color: "+car2.getColor());
       System.out.println("Car 2 Model: "+car2.getModel());
       System.out.println("No. of Cars: "+Cars.getNoOfCars());

    }
}
```

Output:

Car 1 Color: Green
Car 1 Model: AJ43
No. of Cars: 2

Car 2 Color: Red
Car 2 Model: 43os
No. of Cars: 2

Static methods are convenient for many tasks because they can be called directly from the class, as needed. They are most often used to create utility classes that perform

operations on data, but have no need to collect and store data. The Metric class is a good example. It is used as a container to hold methods that convert miles to kilometers and vice versa, but is not intended to store any data. The only limitation that static methods have is that they cannot refer to non-static members of the class. This means that any method called from a static method must also be static. It also means that if the method uses any of the class's fields, they must be static as well.

Practice Tasks

Task 1:

Write a class **AreaClass** that has three overloaded static methods for calculating the areas of the following geometric shapes:

- circles
- rectangles
- Cylinders

Here are the formulas for calculating the area of the shapes.

Area of a circle: **Area= πr^2**

where π is Math.PI and r is the circle's radius

Area of a rectangle: **Area = Width x Length**

Area of a cylinder: **Area = $\pi r^2 h$**

where π is Math.PI, r is the radius of the cylinder's base, and h is the cylinder's height

Because the three methods are to be overloaded, they should each have the same name, but different parameter lists. Demonstrate the class in a complete program.

Task 2:

Create a **SavingAccount** class

- Use a static data member **annualInterestRate** of type **double** to store interest rate.
- Use a **double** data member **savingBalance.**
- Provide a method **calculateMonthlyInterest** that calculates interest by:

  *savingBalance = balance+ (balance*annualInterestRate)/12*
- Provide a method **printBalance** that will display the saving balance.
- Provide a static method **modifyInterestRate** that sets the static **annualInterestRate** to a new value.

Write a driver/client program **TestSavingAccount** to test class **SavingAccount**

- Create two objects **saver1** and **saver2** with the balance of 2000 and 3000 respectively.
- Set the **annualInterestRate** to **3 percent** i.e. 3/100 = 0.03 and calculate **monthlyInterest** and display the balance of both savers.

- Now, set the **annualInterestRate** to **4 percent** i.e. 4/100 = 0.04 and calculate **monthlyInterest** and display the balance of both savers.

Task 3:

Create a class that will bundle together several static methods for tax computations. This class should not have a constructor. Its attributes are

basicRate—the basic tax rate as a static double variable that starts at 4 percent
luxuryRate—the luxury tax rate as a static double variable that starts at 10 percent

Its methods are

computeCostBasic(price)—a static method that returns the given price plus the basic tax, rounded to the nearest value.
computeCostLuxury(price)—a static method that returns the given price plus the luxury tax, rounded to the nearest value.
changeBasicRateTo(newRate)—a static method that changes the basic tax rate.
changeLuxuryRateTo(newRate)—a static method that changes the luxury tax rate.
roundToNearestvalue(price)—a private static method that returns the given price rounded to the nearest Rupee value.

Task 4:

Create a class **Android** whose objects have unique data. The class has the following attributes:
tag—a static integer that begins at 1 and changes each time an instance is created
name—a string that is unique for each instance of this class

Android has the following methods:
Android—a default constructor that sets the name to "Bob" concatenated with the value of tag. After setting the name, this constructor changes the value of tag by calling the private method changeTag.
getName—returns the name portion of the invoking object.

isPrime(n)—a private static method that returns true if n is prime—that is, if it is not divisible by any number from 2 to n − 1.
changeTag—a private static method that replaces tag with the next prime number larger than the current value of tag.

Task 5:

In a ABC Bank software, generation of Account number is manual. You have been assigned a task to create a module that automatically generate an account number with respect to previous generated account number, whenever a new account is opened/created.
For that purpose, create a class **Account** with instance variables accountTitle and AccountNumber. Create a class variable Previous Account Number that holds the status of last generated Account number.

Task 6:
Write static methods

```
public static double sphereVolume(double r)
public static double sphereSurface(double r)
public static double cylinderVolume(double r,double h)
public static double cylinderSurface(double r,double h)
Public static double coneVolume(double r,double h)
public static double coneSurface(double r,double h)
```

that compute the volume and surface area of a sphere with a radius r, a cylinder with a circular base with radius r and height h, and a cone with a circular base with radius r and height h. Place them into a class Geometry. Then write a program that prompts the user for the values of r and h, calls the six methods, and prints the results.