Set 21: Static Methods and State Variables

Skill 21.01: Explain the static designation as it applies to static methods and state variables
Skill 21.02: Explain two uses for static
Skill 21.03: Access methods and data members from within a static method
Skill 21.04: Declare a static constant variable

Skill 21.01: Explain the static designation as it applies to static methods and state variables

Skill 21.01 Concepts

Static methods are sometimes called *class methods*. Similarly, static instance fields (static state variables) are called *class variables*. The reason for the class designation is that when we access either static methods or variables, we are accessing them at the class level rather than at the object level. (In this course, we will primarily use the word static rather than class as the designation of such methods and variables).

A static method is one type of method which doesn't need any object to be initialized for it to be called. Have you noticed *static* is used in the main function in Java? Program execution begins from there without an object being created.

```java
public class Main {
    public static void main(String[] args) {

          System.out.println("Hello World!");
    }
}
```

Skill 21.02: Explain two uses for static

Skill 21.02 Concepts

<u>The first reason for using static</u>

We are accustomed to calling a method or accessing a data member (state variable) by first creating an object and then using that object to reach the method or variable. In the following example, if we want to call a method or access abc from outside the Nerd class, here is how we have had to do it in the past,

Nerd

```java
public class Nerd {

    public String name;
    public int age;

    public Nerd() {
    }// constructor

    public double methodA(double x) {
        return x;
    }
}
```

NerdMaker

```java
public class NerdMaker{

    public static void main(String args[]){

    Nerd nerd1 = new Nerd();
    nerd1.name = "Uddi";
    nerd1.age = 16;

    Nerd nerd2 = new Nerd();
    nerd2.name = "Grace";
    nerd2.age = 17;

    System.out.println(nerd1.name);
    System.out.println(nerd2.name);
    System.out.println(nerd1.age);
    System.out.println(nerd2.age);
    }
}
```

Output

```
Uddi
Grace
16
17
```

Now, we are going to show how to do this without having to create an object. This will require a slight rewrite of the Nerd class.

Nerd

```java
public class Nerd {

  public static String name;
  public static int age;

  public Nerd() {
  }// constructor

  public static double methodA(double x)
  {
      return x;
  }
}
```

NerdMaker

```java
public class NerdMaker{

    public static void main(String args[]){

    Nerd.name = "Uddi";
    Nerd.age = 16;
    System.out.println(Nerd.methodA(3.14));
    }
}
```

Output

```
3.14
```

Notice that we did not need to create an object this time. Rather we used the name of the class. (That's why they're sometimes called class variables and methods).

While this may seem weird, we have done this before! Remember our usage of Math.PI? Math is a class within Java and PI is a static data member there. Because it is static we can access it without first creating an object.

So, there you have it, the first reason for having static variables and methods,

> *The static designation allows you the ability to access variables and methods without having to create an object*

Finally, while we are on this topic, we are now able to see why static is present in the familiar, public static void main(String args[]) signature. It is because we are accessing the main method from the "outside world" without creating an object.

The second reason for using static

We will now examine a class with static state variables and see what happens when we create various instances of this class. (Notice that is the same as saying we create various objects from the class.)

Nerd

```java
public class Nerd {

    public static String name;
    public static int nerdCount = 0;
    public static int age;

    public Nerd() {

        nerdCount++;
    }

    public static int getNerdCount() {

        return nerdCount;
    }
}
```

NerdMaker

```java
public class NerdMaker{

    public static void main(String args[]){

    Nerd nerd1 = new Nerd();
    nerd1.name = "Uddi";
    nerd1.age = 16;

    Nerd nerd2 = new Nerd();
    nerd1.name = "Grace";
    nerd1.age = 17;

    System.out.println(nerd1.name);
    System.out.println(nerd2.name);
    System.out.println(nerd1.age);
    System.out.println(nerd2.age);
    System.out.println(Nerd.nerdCount);
    }
}
```

Output

```
Grace
Grace
17
17
2
```

The above illustrates another principle of static data members. That is, they are shared by all instances (all objects) of that class. In fact, the static variables are still present and available even if no objects are ever instantiated.

Skill 21.02: Exercises 1 & 2

Skill 21.03: Access methods and data members from within a static method

Skill 21.03 Concepts

If from within a static method we try to access another method and/or data member of the same class, then that other method and/or state variable must also be static. This is illustrated by the following code,

Accessing a methods and data members from a static context

```
public class StaticContextExample {

    public static String name;
    public static int age;

    public static void main(String[] args) {

        name = "Uddi";
        age = 17;
        System.out.println(methodA(3.14));
        // setName("Kellen"); //Illegal
    }

    public static double methodA(double x) {

        return x;
    }

    public void setName(String n) {

        name = n;// legal
    }
}
```

From the above example, we can draw the following conclusions,

- Static methods can reference only static variables and never the “regular”, non-static instance variables.
- Non-static methods can reference either.

Skill 21.03: Exercises 1

Skill 21.04: Declare a static constant

Skill 21.04 Concepts

Recall constants are variables that have a constant (unchanging) value. Constants can also be static as demonstrated in the following example,

Static constant

```
public static final double PI = 3.14159;
```