COURSEWARE

Static

Contents

Overview

In Java, The `static` keyword indicates that the member (variable, method, class, etc.) belongs to the type itself, rather than to an instance of that type.

This means that only one instance of the `static` member is created, and is then shared between each instance of the class.

Tutorial

Static Class Variables

Static class variables are perhaps the most common use of the `static` keyword. If a class variable is declared `static`, a single copy of that variable is created that every instance of the class will access.

```
public class Person {
    private String name;
    private String eyeColour;

    public static int numberOfPeople;

    public Person(String name, String colour) {
        this.name = name;
        this.eyeColour = colour;
        numberOfPeople++;
    }

    public getName(){
        return this.name;
    }
    public getEyeColour(){
        return this.eyeColour;
    }
}
```

In the above example, for each instance of `Person` the `static` variable `numberOfPeople` will increment by 1.

We can access this variable by directly calling the class:

```
public static void main(String[] args) {
    Person chris = new Person("Chris","Blue");

    System.out.println(Person.numberOfPeople);
}

output: 1
```

Static variables can be accessed regardless of whether an instance of the class has been created:

```
public static void main(String[] args) {
    System.out.println(Person.numberOfPeople);
}

output: 0
```

Static Methods

Similarly to `static` fields, `static` methods also belong to a class instead of the object, and so they can be called without creating the object of the class in which they reside.

Static methods are generally used to perform an operation that is not dependent upon instance creation. We can add a `static` method to our above example to improve functionality:

```
public class Person {
    private String name;
    private String eyeColour;

    public static int numberOfPeople;

    public Person(String name, String colour) {
        this.name = name;
        this.eyeColour = colour;
        numberOfPeople++;
    }
    // Getters and Setters

    public static void setNumOfPeople(int numPeople){
        Person.numberOfPeople = numPeople;
    }
}
```

Now we can pass in a new value of our choosing. For instance, to reset the count.

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

    Person chris = new Person("Chris","Blue");
    Person tom = new Person("Tom","Brown");

    Person.setNumOfPeople(0);

    Person stephan = new Person("Stephan","Greenish-Brownish-Blueish");

    System.out.println(Person.numberOfPeople);
}

output: 1
```

Static Block

A `static` block is used to initialise `static` variables that require multi-line logic.

Of course, `static` variables can be initialised during declaration:

```
public static int num = 24;
```

In some cases, you will need to use more than one line to initialise the variable.

```
public class StaticBlockDemo {
    public static List<String> languages = new LinkedList<>();

    static {
        languages.add("Java");
        languages.add("C++");
        languages.add("Python");
    }

    static {
        languages.add("HTML");
        languages.add("Groovy");
    }
}
```

As you can see, it is possible to have multiple `static` blocks that will all be executed at runtime to populate the list with our required values.

Static Class

Java allows us to create a class within a class. It provides a way of grouping elements that are only going to be used in one place, which can help to keep code more organised and readable.

- nested classes that are declared `static` are called *static nested classes*.
- nested classes that are non-static are called *inner classes*.

The main difference between these two is that while inner classes have access to all members of the enclosing class (including the `private` ones!), `static` nested classes only have access to `static` members of the outer class:

```
public class Singleton  {
    private Singleton() {}

    private static class SingletonHolder {
        public static final Singleton INSTANCE = new Singleton();
    }

    public static Singleton getInstance() {
        return SingletonHolder.INSTANCE;
    }
}
```

Static nested classes do not have access to any instance members of the enclosing outer class; it can only access them through an object's reference.

Static nested classes can access all `static` members of the enclosing class, including `private` ones.

Java doesn't allow us to declare the top-level class as `static`; only classes within the classes (nested classes) can be made as `static`.

Exercises

There are no exercises for this module.