Understanding Static Fields, Methods and Classes

Dan Geabunea
SENIOR SOFTWARE DEVELOPER

@romaniancoder

Overview

Static modifier

Understand static fields and methods

Explore static initializer blocks

Demonstrate static nested classes

Demo: Demonstrating static fields and methods

A “static” modifier means that something is directly related to the class itself, not to an instance of the class

The “static” Modifier

Static fields / Class variables

Static methods / Class methods

Static initialization blocks

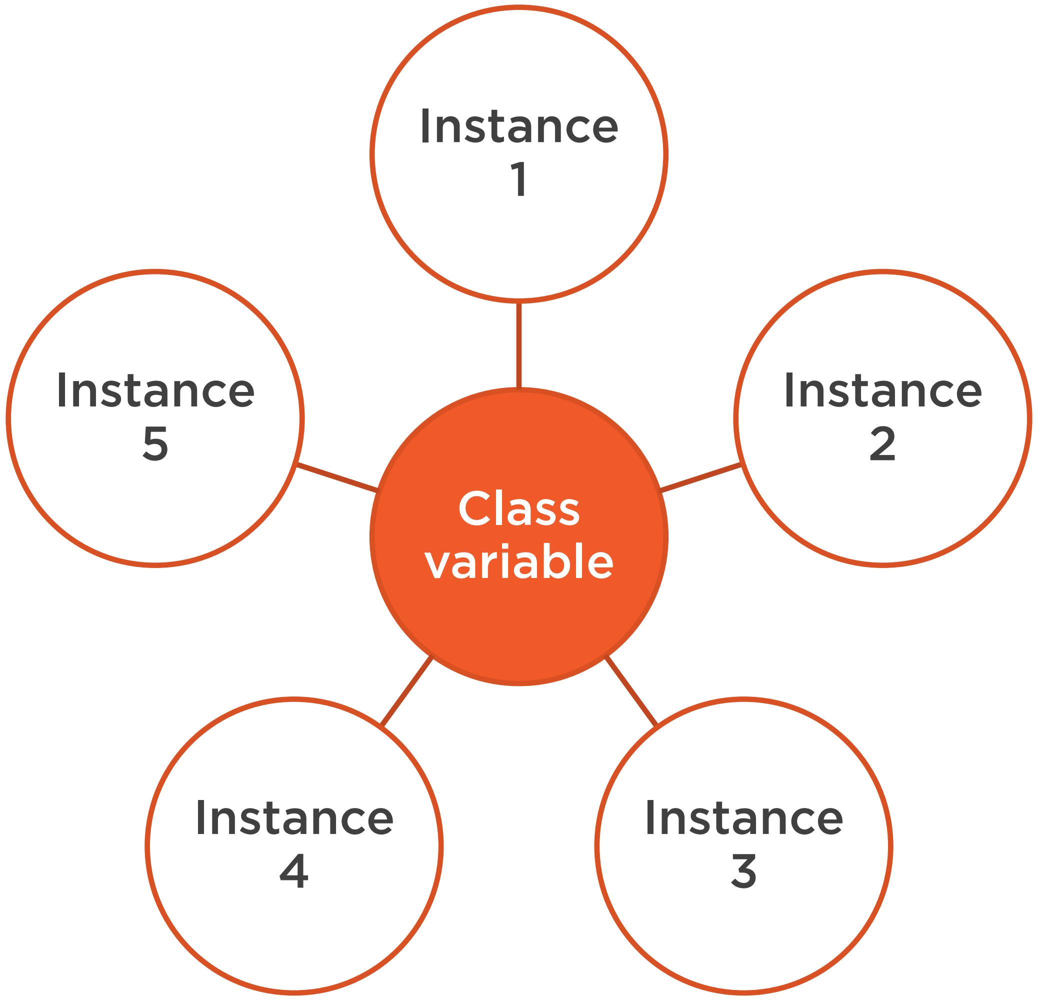
Static (nested) classes

Static Fields

Static Fields / Class Variables

Fields that are associated with the class. Every instance of a class shares a class variable

Class Variable



Static Fields

```java
class JetEngine {
    String serialNumber;
    public static int currentModelNb;


    JetEngine() {
        currentModelNb++;
        this.serialNumber =
            String.format("%08d", currentModelNb);
    }
}
```

Use the “static” modifier

They are shared by all instances, so you can not access them using “this”

Static fields can use access modifiers to limit access to their value

Static Fields

```
JetEngine je1 = new JetEngine();

JetEngine je2 = new JetEngine();

System.out.println(je1.serialNumber);   // Instance field

System.out.println(je2.serialNumber);   // Instance field

// Output: 00000001 00000002



System.out.println(JetEngine.currentModelNb);
```

Each constructor call increases the *currentModelNb* class variable

From outside the class, you can access the static field using the class name, followed by the class variable name

Accessing Static Fields

Use Class Name Followed by Class Variable Name

JetEngine.class

```
package pluralsight.oop;

public class JetEngine {

   public String serialNumber;

   public static int currentModelNb;


   public JetEngine() {

      currentModelNb++;

      this.serialNumber = String.format("%08d",     currentModelNb);

   }

}
```

Main.java

```
import pluralsight.oop.JetEngine;


public class Main {

   public static void main(String[] args) {

      System.out.println(JetEngine.currentModelNb);

   }

}
```

Accessing Static Fields

Use Static Import

JetEngine.class

```
package pluralsight.oop;

public class JetEngine {

    public String serialNumber;

    public static int currentModelNb;


    public JetEngine() {

        currentModelNb++;

        this.serialNumber = String.format("%08d",       currentModelNb);

    }

}
```

Main.java

```
import static pluralsight.oop.JetEngine.currentModelNb;


public class Main {

    public static void main(String[] args) {

        System.out.println(currentModelNb);

    }

}
```

```
public class Constants {

    public static final int SPEED_OF_SOUND = 343;   // meters per sec

}


int halfSpeedOfSound = Constants.SPEED_OF_SOUND / 2;
```

Constants

You can add the “final” modifier to a static field and effectively make it constant

Once a value is assigned, the field can not be assigned another value

Constants are named using upper case and words are separated by _

Static Methods

Static Methods / Class Methods

Methods that can be invoked on the class, without the need to create instances of the class

Static Method Example

```
package pluralsight.oop;

public class FuelCalculator {

    static final int AVG_LITERS_PER_SEC = 4;


    public static int calculateNeed(int durationMin){

        int durationInSec = durationMin * 60;

        return AVG_LITERS_PER_SEC * durationInSec;

    }

}
```

Use the “static” modifier to create a static method

Static methods can use static fields

Static methods can have access modifiers

Usually used for utility methods

Calling Static Methods

Use Class Name Followed by Method Name

```java
package pluralsight.oop;

public class FuelCalculator {

    static final int AVG_LITERS_PER_SEC = 4;


    public static int calculateNeed(int durationMin){

        int durationInSec = durationMin * 60;

        return AVG_LITERS_PER_SEC * durationInSec;

    }

}
```

```java
import pluralsight.oop.FuelCalculator;


public class Main {

    public static void main(String[] args) {

        int twoHourFlight = 120;

        int quantity =  FuelCalculator.calculateNeed(twoHourFlight);

        System.out.println(quantity);

    }

}
```

Calling Static Methods

Use Static Import

```
package pluralsight.oop;

public class FuelCalculator {

    static final int AVG_LITERS_PER_SEC = 4;


    public static int calculateNeed(int durationMin){

        int durationInSec = durationMin * 60;

        return AVG_LITERS_PER_SEC * durationInSec;

    }

}
```

```
import static pluralsight.oop.FuelCalculator.calculateNeed;


public class Main {

    public static void main(String[] args) {

        int twoHourFlight = 120;

        int quantity =    calculateNeed(twoHourFlight);

        System.out.println(quantity);

    }

}
```

Static Methods Used as Utility Methods

```
Math.max(10,20);

Math.sqrt(16.0);

Character.toUpperCase(‘a’);

String.format("%08d", 23);

System.exit(0);
```

```
public class Utils {

    public static int sum(int a, int b){...}

    public static void log(String msg){}

    public static int calculateFuel(){}

}
```

Don’t Create God Utility Classes

Each utility class should have a very specific purpose

The Singe Responsibility Principles applies to these classes as well

```
public class JetEngine {

    public String serialNumber;

    public static int currentModelNb;



    static {

        currentModelNb = 10;

    }

}
```

Static Initialization Block

Code block enclosed in braces and preceded by the “static” keyword

Used to initialize class variables

They can only access class variables and methods directly

A class can have multiple static blocks; they will be executed in order

Instance Vs. Class

Instance methods

Can access instance variables and other instance methods

Can access class variables and class methods

Class methods

Can access class variables and other class methods

Cannot access instance variables or methods directly, they must use an object reference to do this

Can not use the “this” keyword

Static Nested Classes

```java
public class FlightDataRecorder {

    private List<Entry> entries = new ArrayList<>();

    public void log(String msg) {

        entries.add(new Entry(LocalDateTime.now(), msg));

    }


    private static class Entry {          // Full fledged class, but we do not want to expose it

        private LocalDateTime dateTime;

        private String log;

        private Entry(LocalDateTime dateTime, String log) {

            this.dateTime = dateTime;

            this.log = log;

        }

    }

}
```

Static Nested Class

A top-level class that has been nested in another top-level class for convenience

A static nested class interacts with the instance members of its outer class just like any other class

```java
public class FlightDataRecorder {

    public static class Entry {

        private LocalDateTime dateTime;

        private String log;

        public Entry(LocalDateTime dateTime, String log) {

            this.dateTime = dateTime;

            this.log = log;

        }

    }

}

// Can also be exposed at package or public level; To instantiate it, you need to use the outer class name

FlightDataRecorder.Entry e = new FlightDataRecorder.Entry(LocalDateTime.now(), "msg");
```

Serialization Example

JSON

```
{
"sac": "045",
"sic": "032",
"position": {
    "lat": "23.5",
    "lon": "34.7“
},
"altitude": {
    "barometric": "34000",
    "measured": "335" }
}
```

JAVA

```
public class RadarJson {
    public String sac, sic;


    public static class Position{
        public double lat, lon;
    }


    public static class Altitude{
        public int barometric, measured;
    }
}
```

Keep together things that change together

Demo

Demo: Demonstrating Static Fields and Methods

- Creating helper methods

Summary

You can use “static” to create class variables and methods

Static members belong to the class, not to individual instances

Static methods can not access instance members directly, only through an object reference

A static nested class is a normal class that has been declared inside a class for convenience

Up Next:
Using Encapsulation and Inheritance