Static Members and Enums

Java Developer

StayAhead Training; May, 2022

Static Members

Static Members and Enums

- Fields and methods come in two varieties - instance and static
- An instance field/method is one that "belongs" to the instances/objects
- Imagine that the field/method is copied into the instance/object upon creation
- A static field/method is one that "belongs" to the class
- Static fields/methods are not copied into the instance/object
- NB: technically each instance/object has its own set of instance fields but not instance methods - this would involve unnecessary duplication of code

Static Members

Static Members and Enums

- The static keyword marks a field/method as being static
- Each of a class's static members (fields and methods) is accessed via the dot notation (Class.member), e.g.

```
MyClass.staticField;
MyClass.staticMethod();
```

- Unfortunately static members may be accessed via the object too (object.member) but doing so is bad practice - it gives the impression that the member belongs to the instance/object when it does not

Static Fields

Static Members and Enums

- A field whose value is shared by all instances/objects should be static, e.g.

```
static int numInstances;
```

- The number of instances/objects is a classic use case and such a field would most likely be incremented inside the constructor
- Consider making a field static when the value for said field will be the same for every instance/object

Static Fields

Static Members and Enums

- Static fields are often constants, e.g.

```
static final int MAX_CAPACITY = 10;
```

- Recall that a final variable is one that cannot be reassigned
- Conventions dictate that such field names should be SCREAMING_SNAKE_CASE

Static Methods

Static Members and Enums

- A method that does not ref. any of the instance fields should be static, e.g.

```
static double circumference(int radius) {
  return 2 * radius * Math.PI;
}
```

- Think carefully before making a method static
- If the method does not ref. any of the instance fields does it belong?
- Static methods are often used to code in a procedural way/to avoid having to devise an object-oriented solution

The Static Context

Static Members and Enums

- A class must be loaded into memory before it can be instantiated
- Static members, therefore, are accessible before the class has been instantiated/objects have been created
- Effectively, the static context is the set of fields and methods that can be accessed inside a static method
- As the static method may be called before any instances/objects exist then no instance field/method can be accessed inside the static method

The Static Context

Static Members and Enums

```
class Thing {

  String name;

  static void writeNameToStdout() {
    // compilation error
    // non-static field ref'd in a static context
    System.out.println(name);
  }

}
```

The main Method*

Static Members and Enums

- The main method is static because the JVM attempts to call it without first having instantiated the main class

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

- Assuming a class named App the JVM effectively calls `App.main(<command_line_args>);`

Static Imports

Static Members and Enums

- The repeated prefixing of static members with the class name is tedious, e.g.

```
var randomNumber = (int) Math.floor(Math.random() * 10);
```

- Static imports enable the use of static members without the class name, e.g.

```
import static java.math.Math.*;

var randomNumber = (int) floor(random() * 10);
```

- Where a regular import references a class name/package of classes, a static import references a static member/class of static members

Enums

Static Members and Enums

- An enum is a class that contains static constant fields only
- An enum is a good choice for fields that have a small, fixed set of valid values

```
class Movie {
  String title;
  Genre genre;
}

enum Genre {
  ACTION, COMEDY, DRAMA;
}
```

Enums

Static Members and Enums

- This…

```
enum Genre {
  ACTION, COMEDY, DRAMA;
}
```

- Compiles into this…

```
class Genre {
  public static final Genre ACTION = new Genre();
  public static final Genre COMEDY = new Genre();
  public static final Genre DRAMA = new Genre();
}
```

Enums

Static Members and Enums

- When you write an enum field to, say, stdout you get a String representation of the object - this leads devs to assume that enum fields are Strings

```
System.out.println(Genre.ACTION); // ACTION
```

- Enum fields are not of type String; the type is the enum type, e.g. Genre
- It is possible to override the default behaviour so that writing the enum field yields something other than its name

Enums

Static Members and Enums

- Each enum has a method name valueOf that accepts a String and returns an instance of the enum, e.g.

```
var genre = Genre.valueOf("ACTION");
```

- The relational operator equals (==) can be used with enums, e.g.

```
var genre = Genre.valueOf(scanner.nextLine());

if (genre == Genre.COMEDY) { ... }
```