Java basics

Tamás Ambrus

Eötvös Loránd University
ambrus.thomas@gmail.com

Retrospective

Final fields

Marking a class field final means:

- it will have a value at the end of the construction (by decl. line or contructor)
- **the reference won't change**, although the state of the referred object still can change
- *immutable objects have only final fields (multithreaded environment)*

When a programmer looks at a type and sees final fields, calms down. Those fields won't make any magic trick (e.g. change when you would not expect that).

Constructors, getters, setters

Good to know:

- you don't necessarily need getters and setters for all of your fields
- if you can choose from parameterized constructor or a no-argument constructor with setters, choose the former one
    - if you have both in your type, that makes the programmer confused
    - when it is possible, avoid setters and use fully parameterized constructors, as a setter can be called many times on an object

Best practice: consider what you really need in your type and implement only them (with final fields)

Java basics for the further practices

Static methods

We have learned about regular methods:

- that are related to data types (encapsulation)
- can be called on the objects of the type

There are also static methods (or static type fields) that belong to the type itself, not the objects of it.

```
public static void shuffle(List<?> list) { ... }
```

Static methods

Regular method:

```
public class IntList {
    // ...
    public void shuffle() { ... }
}
// call: new IntList(3, 5, 6, 8, 8, 91).shuffle();

------
```

Static method:

```
List<Integer> intList = Arrays.asList(3, 5, 6, 8, 8, 91);
Collections.shuffle(intList);
// shuffle is static method in "Collections" type
// docs.oracle.com/javase/7/docs/api/java/util/Collections.html
```

Usages of static

Static is good for a couple of things:

- a static class field belongs to the type so is stored only once, not per objects
- a static method is a utility method, "helps objects of the type"
- there are also **utility class**es that **have only static methods** designed to help other types' objects
- static methods can play the role of named constructors

Named constructors

```java
public class Rectangle {

    private final double a;
    private final double b;

    private Rectangle(double a, double b) {
        this.a = a;
        this.b = b;
    }

    public static Rectangle createSquare(double a) {
        return new Rectangle(a, a);
    }

    public static Rectangle create(double a, double b) {
        return new Rectangle(a, b);
    }
}

// Rectangle square = Rectangle.createSquare(6.0);
```

Enumerations

An enum type is a special data type. *It is basically a class*, that can work only with predefined constants.

```
public enum Direction {
    NORTH, SOUTH, EAST, WEST
}
```

Programmers use enums when they want to work with a fixed set of constants, e.g. days, months, planets.

Enumerations

```
public enum Day {
    MONDAY(":'("),
    TUESDAY(":("),
    WEDNESDAY(":("),
    THURSDAY(":/"),
    FRIDAY(":)"),
    SATURDAY(":D"),
    SUNDAY("^_^");

    private final String smiley;

    private Day(String smiley) {
        this.smiley = smiley;
    }

    public void expressFeeling() {
        System.out.println("I feel about " +
          name().toLowerCase() + "s like " + smiley);
    }
} // Day.MONDAY.expressFeeling();
// or Day d = Day.MONDAY; d.expressFeeling();
```

Best practices

Eclipse shortcuts

Most of these shortcuts makes your developing process easier and faster:
`http://www.vogella.com/tutorials/EclipseShortcuts/article.html`

For daily use:

- search for Java types - either yours or built-in ones
- content assist / code completion
- show all methods of the current type
- format source code
- organize the imports

Java conventions

Every programming language has its own conventions:

- how type names should be written
- where to put the brackets - new line or not
- how many spaces should be used as indentations
- etc.

Learn them, look at the examples of the Java books, read the community.

Refactoring

Programmers often solve their task the easiest way they can:

- use wrong or short names for variables
- put the code into 1-2 methods resulting a huge mess

Let's apply refactoring minutes in our development process:

- cut the functionalities into well-named and short methods using well-named variables
- simplify your methods and algorithms
- delete empty lines, format the source code
- remove the unused imports, etc.

You will be grateful when you look back months later.

"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live."

References

- https://docs.oracle.com/javase/tutorial/java/javaOO/classvars.html
- https://docs.oracle.com/javase/tutorial/java/javaOO/enum.html
- http://www.oracle.com/technetwork/java/codeconventions-135099.html