howest
hogeschool

Object Oriented Architectures and Secure Development

01-01 Recap Object Orientation

Arne Debou
Mattias De Wael
Frédéric Vlummens

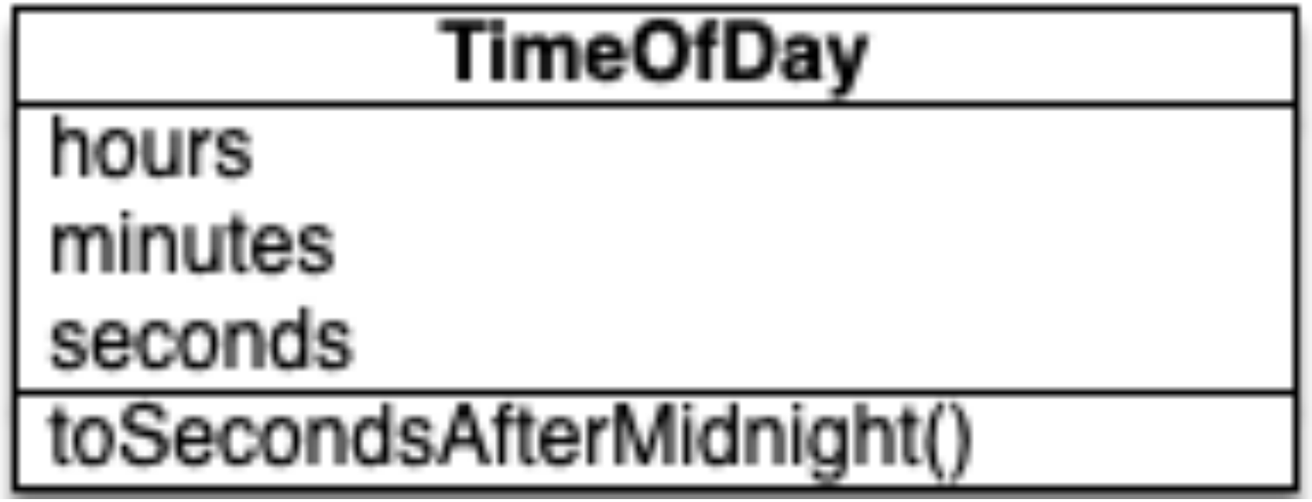
Recap Object Orientation

- Classes
- Objects
- Properties/getters and setters
- Constructors
- Interfaces
- Equality
- Collections
- Exceptions
- Enums
- Static
- Final
- Packages
- Inheritance
- Abstract
- Inheritance vs Composition

Classes

- Fields (state)
- Methods (behaviour)
- Classes start with a capital letter

- UML:



Objects

- Objects are instances of classes

```
Cat c = new Cat("Garfield");

System.out.println(c.getName());

c.eat();
c.sleep();
```

Fields and properties: getters and setters (slide © Mr. De Wael)

```
class TimeOfDay {

  private int hours;

  public int getHours() { return hours; }

  public void setHours(int newHours) {
    if (newHours >= 0 && newHours <= 24) { // protect local state from invalid data.
      hours = newHours;
    }
  }
}
```

a ***getter*** is a method that returns the value of a field.

a ***setter*** is a method that updates the value of a field.

Do NOT write getters and/or setters for fields you do not want other users to access!

howest hogeschool

Constructors

- Special methods
- Called when creating (constructing) an object
- Don’t forget about:
  - Constructor chaining
  - Copy constructors

Interfaces

- Different use cases:
  - For example, a printer driver for macOS, Windows and Linux
  - Should support same behaviour, but implementation differs

  or

  - Tag classes as able to do a specific task
  - Comparable/Comparator
  - …

Interfaces

```
public interface Printable {
    void print(Printer prn);
}

public class Page implements Printable {
    @Overrides
    public void print(Printer prn) {
        prn.enqueue(getContents());
    }
}
```

Equality and comparison

- Don’t forget:
  - Java Strings: compare using **.equals** method
  - Do not use ==

- Implementing **equals** and **hashCode** methods

- Comparing objects: make a class Foo Implement Comparable<Foo>
  → compareTo method

Collections

Don’t forget about:

- List
- Map
- Set

their properties and their implementations

Exceptions (and using them in testing)

- When things could go wrong
- Catch the exception that gets thrown

```
try {
        // some code
} catch (ExceptionType1 | ExceptionType 2 ex) {
        // handle the exception
}
```

- Throwing IllegalArgumentException of IllegalStateException
- Later: our own exceptions

Enum

```
public Enum Weekday {
        MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY;
}
```

You can compare enum instances using ==

Using switch-case is also allowed

Static

- Static fields
  - When the value is independent of objects
  - When the value is supposed to be shared across all objects

- Static methods
  - To access/manipulate static variables and other static methods that don't depend upon objects
  - *static* methods are widely used in utility and helper classes
  - To create (a collection) of instances of the class: e.g. createDeck in the class Card. (factory method)
  - psvm (public static void main)

Final

- Use the **final** keyword to make variables (params, local vars, fields) final
- A final variable: cannot be reassigned
- Must have an initial value:
  - Parameters: when method is called
  - Local variables: when declared
  - Fields:
    - Either in declaration
    - Or in constructor
- Rule of thumb: make your fields final where possible
- Attention: final collections → cannot be reassigned, but you can modify the contents (elements inside the collection). See later to prevent this.

Packages

- Convention: reverse domain names
- Example: **be.howest.ti.ooasd.helloworld**
- Don’t forget about packages and visibility

Inheritance

- A Dog is an Animal → inheritance: Dog extends Animal
- A Page can be Printed → interface: Page implements Printable

Abstract

- When it doesn’t make sense to create objects of a class

```
public abstract class Shape {
        public abstract double calculateSurface(); // We don’t know how to calculate a shape’s surface
}

public class Rectangle extends Shape {
        @Override
        public double calculateSurface() {
                return getWidth() * getHeight();
        }
}
```

Inheritance vs composition

- You can only inherit from one super class
- You can compose multiple instances of objects
- Delegate

```
public class ShapeWithBorder extends Shape {

    private Border border;
    private Shape base;

    public ShapeWithBorder(Shape base, Color borderColor) {
        super(base.getX(), base.getY(), base.getColor());
        this.base = base;
        this.border = new Border(base, borderColor);
    }

    @Override protected int getMaxHeight() { return base.getMaxHeight(); }

    @Override protected int getMaxWidth() { return base.getMaxWidth(); }

    @Override protected boolean contains(int i, int j) { return base.contains(i, j); }

    @Override
    public void drawOn(DrawBoard drawBoard) {
        base.drawOn(drawBoard);
        border.drawOn(drawBoard);
    }

}
```

composition

delegation

howest hogeschool