Constructors

Constructors

When we create a new **Employee** object using:

```
Employee emp = new Employee();
```

We notice that all the fields are empty inside the object.

```
public class Employee {
    private String firstName;
    private String lastName;
    private String title;

    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }
    public void setTitle(String title) {
        this.title = title;
    }
}
```

Empty by default

Constructors

To set them we need to call each individual **set** method on the object.

```
Employee emp = new Employee();

emp.setTitle("Mr");
emp.setFirstName("Tore");
emp.setLastName("Nestenius");
```

What can go wrong when we initialize an object using this approach?

Constructors

What can go wrong?

```
Employee emp = new Employee();

emp.setTitle("Mr");
emp.setFirstName("Tore");
emp.setLastName("Nestenius");
```

- What happens if we **forget** to call one of them?
- What happens if we later add a **new field**?
- The user of this object must understand what methods to call, to properly initiate this object.

Constructors

We want to somehow **initialize** the data each time we create a new object.

```
Employee emp = new Employee();

emp.setTitle("Mr");
emp.setFirstName("Tore");
emp.setLastName("Nestenius");
```

We want to **force** the user to always set them.

A special kind of method called a **constructor** can help us with this!

Constructors - Instantiating an object

Remember that we always have to include parentheses at the end when we Instantiating an object.

```
Employee emp = new Employee();
```

This looks suspiciously close to a method call.

Constructors

The **constructor** is a special **method** which is always automatically called at instantiation.

The method has a slightly different syntax than other methods:

- No return type
- Has to have the **exact** same name as the class.

```
public class Employee {

    public Employee() {
    }
}
```

Anatomy of a constructor

Writing a **constructor** for the **Employee** class.

```
public class Employee {
    private String firstName;
    private String lastName;
    private String title;
```

Must be public in order to be instantiable outside of this class

Arguments to the constructor

```
    public Employee(String firstName, String lastName, String title) {
```

Must have the same name as the class

```
        this.firstName = firstName;
        this.lastName = lastName;
        this.title = title;
    }
}
```

The "**this**" keyword points to this specific instance.

Multiple constructors

We can have several different constructors

```
public class Employee {
    private String firstName;
    private String lastName;
    private String title;

    // Getters and setters omitted

    public Employee() {
        this.firstName = "";
        this.lastName = "";
        this.title = "";
    }

    public Employee(String firstName, String lastName, String title) {
        this.firstName = firstName;
        this.lastName = lastName;
        this.title = title;
    }
}


Employee emp1 = new Employee();
Employee emp2 = new Employee("Tore", "Nestenius", "Mr");
```

Default constructor

Private constructor

A private constructor prevents the class from being instantiated.

```
private Employee() {

}
```

Trying to create an instance will fail.

```
Employee emp = new Employee();
```

```
'Employee()' has private access in 'com.foo.Employee'
```

This can be useful in some cases.

Default Constructors

If you don’t provide your own constructor, the class is given an empty public constructor by the compiler.

A constructor such as this is called a **default constructor**, taking no arguments at all.

You can implement your own default constructor when needed:

```java
public Employee() {

}
```

Using the constructor

The **constructor** gets called when the object is **created**.

```java
public static void main(String[] args) {
    //NOTE: This will not compile anymore (private)
    Employee employee = new Employee();

    //Employee instance
    Employee firstEmployee = new Employee("Tore", "Nestenius", "Teacher");

    //Another instance, another data
    Employee secondEmployee = new Employee("John", "Doe", "Dr");

    // This will print out "Name: Teacher, Tore Nestenius“
    System.out.println("Name : " + firstEmployee.getEntireNameAndTitle());

    // This will print out "Name: Dr John Doe“
    System.out.println("Name : " + secondEmployee.getEntireNameAndTitle());
}
```

Values to initialize the instance with

Values to initialize another instance with

Exercise 13

Let's do exercise 13