Part 5

Removing repetitive code (overloading methods and constructors)

Learning Objectives

- Becoming familiar with the term overloading
- Creating multiple constructors for a class.
- Creating multiple methods with the same name in a class.

Let's once more return to our Person class. It currently looks like this:

```
public class Person {

    private String name;
    private int age;
    private int height;
    private int weight;

    public Person(String name) {
        this.name = name;
        this.age = 0;
        this.weight = 0;
        this.height = 0;
    }

    public void printPerson() {
        System.out.println(this.name + " is " + this.age + " years old");
    }

    public void growOlder() {
        this.age++;
    }
```

```
    public boolean isAdult() {
        if (this.age < 18) {
            return false;
        }

        return true;
    }

    public double bodyMassIndex() {
        double heightInMeters = this.height / 100.0;

        return this.weight / (heightInMeters * heightInMeters);
    }

    public String toString() {
        return this.name + " is " + this.age + " years old, their BMI is " + th
    }

    public void setHeight(int height) {
        this.height = height;
    }

    public int getHeight() {
        return this.height;
    }

    public int getWeight() {
        return this.weight;
    }

    public void setWeight(int weight) {
        this.weight = weight;
    }

    public String getName() {
        return this.name;
    }
```

All person objects are 0 years old when created. This is because the constructor sets the value of the instance variable `age` to 0:

```
public Person(String name) {
        this.name = name;
        this.age = 0;
        this.weight = 0;
```

```
        this.height = 0;
    }
```

Constructor Overloading

We would also like to be able to create persons so that the constructor is provided both the age as well as the name as parameters. This is possible since a class may have multiple constructors.

Let's make an alternative constructor. The old constructor can remain in place.

```
public Person(String name) {
        this.name = name;
        this.age = 0;
        this.weight = 0;
        this.height = 0;
    }

public Person(String name, int age) {
    this.name = name;
    this.age = age;
    this.weight = 0;
    this.height = 0;
}
```

We now have two alternative ways to create objects:

```
public static void main(String[] args) {
    Person paul = new Person("Paul", 24);
    Person ada = new Person("Ada");

    System.out.println(paul);
    System.out.println(ada);
}
```

Sample output

```
Paul is 24 years old.
Ada is 0 years old.
```

The technique of having two (or more) constructors in a class is known as *constructor overloading*. A class can have multiple constructors that differ in the number and/or type of their parameters. It's not, however, possible to have two constructors with the exact same parameters.

We cannot, for example, add a `public Person(String name, int weight)` constructor since it would be impossible for Java to differentiate between this and the one that has two parameters where int parameter is used for age.

Calling Your Constructor

Hold on a moment. We'd previously concluded that "copy-paste" code is not a good idea. When you look at the overloaded constructors above, however, they have a lot in common. We're not happy with this.

The first constructor - the one that receives a name as a parameter - is in fact a special case of the second constructor - the one that's given both name and age. What if the first constructor could call the second constructor?

This is possible. A constructor can be called from another constructor using the `this` keyword, which refers to this object in question!

Let's modify the first constructor so that it does not do anything by itself, but instead calls the second constructor and asks it to set the age to 0.

```
public Person(String name) {
    this(name, 0);
    //here the code of the second constructor is run, and the age is set to 0
}

public Person(String name, int age) {
    this.name = name;
    this.age = age;
    this.weight = 0;
    this.height = 0;
}
```

The constructor call `this(name, 0);` might seem a bit weird. A way to think about it is to imagine that the call is automatically replaced with "copy-paste" of the second constructor in such a way that the age

parameter is set to 0. NB! If a constructor calls another constructor, the constructor call must be the first command in the constructor.

New objects can be created just as before:

```
public static void main(String[] args) {
    Person paul = new Person("Paul", 24);
    Person eve = new Person("Eve");

    System.out.println(paul);
    System.out.println(eve);
}
```

Sample output

```
Paul is 24 years old.
Eve is 0 years old.
```

Quiz:

Overloading a constructor

Points: 1/1

Below is a Java class that represents a pet.

```
public class Pet {
    private String name;
    private int age;

    public Pet(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public Pet(String name) {
        this(name, 0);
    }

    public Pet(int age) {
        this("Bella", age);
    }

    @Override
    public String toString() {
```

```
        return name + " (" + age + " years)";
    }
}
```

What will the program below print?

```
ArrayList<Pet> pets = new ArrayList<>();

pets.add(new Pet("Nagini", 7));
pets.add(new Pet("Crookshanks"));
pets.add(new Pet(6));

for (Pet pet: pets) {
    System.out.print(pet + ", ");
}
```

Select the correct answer

- Nagini (7 years), Crookshanks (7 years), Crookshanks (6 years),
- ✓ Nagini (7 years), Crookshanks (0 years), Bella (6 years),
- Nagini (7 years), Crookshanks (years), (6 years),
- Nagini (7 vuotta), Crookshanks (7 years), Bella (6 years),
- Nagini (7 years), Crookshanks (0 years), Nagini (6 years),
- Nagini (7 years), Crookshanks, (6 years),

The answer is correct

Answered The number of tries is not limited

Programming exercise:

Constructor Overload

Points 1/1

The exercise template has a class `Product`, which represents a product in a shop. Every product has a name, location and weight.

Add the following three constructors to the `Product` class:

- `public Product(String name)` creates a product with the given name. Its location is set to "shelf" and its weight is set to 1.
- `public Product(String name, String location)` creates a product with the given name and the given location. Its weight is set to 1.
- `public Product(String name, int weight)` creates a product with the given name and the given weight. Its location is set to "shelf".

You can test your program with the following code:

```
Product tapeMeasure = new Product("Tape measure");
Product plaster = new Product("Plaster", "home improvement section");
Product tyre = new Product("Tyre", 5);

System.out.println(tapeMeasure);
System.out.println(plaster);
System.out.println(tyre);
```

Sample output

```
Tape measure (1 kg) can be found from the shelf
Plaster (1 kg) can be found from the home improvement section
Tyre (5 kg) can be found from the shelf
```

Exercise submission instructions

How to see the solution

Method Overloading

Methods can be overloaded in the same way as constructors, i.e., multiple versions of a given method can be created. Once again, the parameters of the different versions must be different. Let's make another version of the `growOlder` method that ages the person by the amount of years given to it as a parameter.

```
public void growOlder() {
    this.age = this.age + 1;
}

public void growOlder(int years) {
    this.age = this.age + years;
}
```

In the example below, "Paul" is born 24 years old, ages by a year and then by 10 years:

```
public static void main(String[] args) {
    Person paul = new Person("Paul", 24);
    System.out.println(paul);

    paul.growOlder();
    System.out.println(paul);

    paul.growOlder(10);
    System.out.println(paul);
}
```

Prints:

Sample output

```
Paul is 24 years old.
Paul is 25 years old.
Paul is 35 years old.
```

A Person now has two methods, both called `growOlder`. The one that gets executed depends on the number of parameters provided.

We may also modify the program so that the parameterless method is implemented using the method `growOlder(int years)`:

```
public void growOlder() {
    this.growOlder(1);
}

public void growOlder(int years) {
    this.age = this.age + years;
}
```

Quiz:
Overloading a method

Points:
1/1

Below is a Java class that represents a counter.

```
public class Counter {
    private int value;

    public Counter(int value) {
        this.value = value;
    }

    public void increaseValue() {
        this.value = this.value + 1;
    }

    public void increaseValue(int number) {
        this.value = this.value + number;
    }

    public int getValue() {
        return value;
```

```
    }
}
```

What will the following program print?

```
Counter counter = new Counter(1);
counter.increaseValue(2);
counter.increaseValue();
counter.increaseValue();
System.out.println(counter.getValue());
```

Select the correct answer

- 3
- 2
- 1
- 7
- ✓ 5
- 4

The answer is correct

Answered The number of tries is not limited

konstruktorin ja metodien kuormittaminen

Programming exercise:

Overloaded Counter (2 parts)

Points 2/2

NB! By submitting a solution to a part of an exercise which has multiple parts, you can get part of the exercise points. You can submit a part by using the 'submit' button on NetBeans. More on the programming exercise submission instructions: Exercise submission instructions.

Part 1: Multiple constructors

Implement a class called `Counter`. The class contains a number whose value can be incremented and decremented. The class must have the following constructors:

- `public Counter(int startValue)` sets the start value of the counter to startValue.
- `public Counter()` sets the start value of the counter to 0.

And the following methods:

- `public int value()` returns the current value of the counter
- `public void increase()` increases the value by 1

- `public void decrease()` decreases the value by 1

Part 2: Alternative methods

Implement versions which are given one parameter of the methods `increase` and `decrease`.

- `public void increase(int increaseBy)` increases the value of the counter by the value of increaseBy. If the value of increaseBy is negative, the value of the counter does not change.
- `public void decrease(int decreaseBy)` decreases the value of the counter by the value of decreaseBy. If the value of decreaseBy is negative, the value of the counter does not change.

Exercise submission instructions

How to see the solution

You have reached the end of this section! Continue to the next section:

→ 3. Primitive and reference variables

Remember to check your points from the ball on the bottom-right corner of the material!

In this part:

1. Learning object-oriented programming
2. Removing repetitive code (overloading methods and constructors)
3. Primitive and reference variables
4. Objects and references
5. Conclusion

 Source code of the material

This course is created by the Agile Education Research -research group of the University of Helsinki.

Credits and about the material.