Part 6

Objects on a list and a list as part of an object

Learning Objectives

- You review the use of lists.
- You know how to use a list as an instance variable.

Next, let's have a look at objects that contain a list. Examples of objects like these include objects that describe sets, for example playlists.

In the following example, we have made a class for the concept of a playlist. The playlist contains songs: songs can be added, songs can be removed, and the songs that the playlist contains can be printed.

```
// imports

public class Playlist {
    private ArrayList<String> songs;

    public Playlist() {
        this.songs = new ArrayList<>();
    }

    public void addSong(String song) {
        this.songs.add(song);
    }

    public void removeSong(String song) {
        this.songs.remove(song);
    }

    public void printSongs() {
        for (String song: this.songs) {
```

```
            System.out.println(song);
        }
    }
}
```

Creating playlists is easy with the help of the class above.

```
Playlist list = new Playlist();
list.addSong("Sorateiden kuningas");
list.addSong("Teuvo, maanteiden kuningas");
list.printSongs();
```

Sample output

```
Sorateiden kuningas
Teuvo, maanteiden kuningas
```

Programming exercise:

Menu (3 parts)

Points 3/3

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.

The gourmet restaurant 'Unicafe' on the Kumpula campus of the University of Helsinki needs a new menu. The chef knows about programming and wants a computer system to manage the menu. In this assignment, we'll implement the heart of the system, the Menu class.

The exercise template comes with a `Main` class that you can use to test the menu. For the implementation of the menu, you'll have the following boilerplate code:

```
import java.util.ArrayList;

public class Menu {
```

```
    private ArrayList<String> meals;

    public Menu() {
        this.meals = new ArrayList<>();
    }

    // implement the required methods here
}
```

The menu object has an ArrayList as an instance variable to store the names of the dishes on the menu. The menu should provide the following methods:

- `public void addMeal(String meal)` adds a meal to the menu. If the meal is already on the list, it should not be added again.
- `public void printMeals()` prints the meals.
- `public void clearMenu()` clears the menu.

Once the menu is done, you can use it as follows.

```
Menu menu = new Menu();
menu.addMeal("Tofu ratatouille");
menu.addMeal("Chilli coconut chicken");
menu.addMeal("Chilli coconut chicken");
menu.addMeal("Meatballs with mustard sauce");

menu.printMeals();
menu.clearMenu();

System.out.println();
menu.addMeal("Tomato and mozzarella salad");
menu.printMeals();
```

Sample output

```
Tofu ratatouille
Chilli coconut chicken
Meatballs with mustard sauce

Tomato and mozzarella salad
```

Part 1: Adding a Meal

Implement the `public void addMeal(String meal)` method, which adds a new meal to the `meals` list. If the meal you want to add is already on the list, you shouldn't add it again. The list method `contains` is handy for checking an items existence on it.

Part 2: Printing the Meals

Implement the `public void printMeals()` method, which prints the meals. You can test out the program using the following example code.

```
Menu menu = new Menu();
menu.addMeal("Tofu ratatouille");
menu.addMeal("Chilli Coconut Chicken");
menu.addMeal("Chilli Coconut Chicken");
menu.addMeal("Meatballs with mustard sauce");

menu.printMeals();
```

Sample output

```
Tofu ratatouille
Chilli Coconut Chicken
Meatballs with mustard sauce
```

Part 3: Clearing the Food List

Implement the `public void clearMenu()` method, which clears the menu. The `ArrayList` class has a method which is useful here. NetBeans can hint at the available methods when you type the object name an a dot. Try to write `meals.` inside the method frame and see what happens.

Once the menu is ready, try it with the following example code.

```
Menu menu = new Menu();
menu.addMeal("Tofu ratatouille");
menu.addMeal("Chilli Coconut Chicken");
menu.addMeal("Chilli Coconut Chicken");
```

```
menu.addMeal("Meatballs with mustard sauce");

menu.printMeals();
menu.clearMenu();

System.out.println();
menu.addMeal("Tomato and mozzarella salad");
menu.printMeals();
```

Sample output

```
Tofu ratatouille
Chilli Coconut Chicken
Meatballs with mustard sauce

Tomato and mozzarella salad
```

Exercise submission instructions

How to see the solution

Programming exercise:

Stack (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.

A stack is a data structure that you can add to and take from. Always to the top of it or from the top.

Part 1: Adding to the Stack and Checking Emptiness

Create a `Stack` class that has a list of strings as an instance variable. Add the following methods to the class:

- `public boolean isEmpty()` - returns a `boolean`-type value (true or false) that tells whether or not the stack is empty.
- `public void add(String value)` - Adds the value given as a parameter to the top of the stack.
- `public ArrayList<String> values()` - returns the values contained in the stack as a list.

You can test your class with the following code:

```
Stack s = new Stack();
System.out.println(s.isEmpty());
System.out.println(s.values());
s.add("Value");
System.out.println(s.isEmpty());
System.out.println(s.values());
```

Sample output

```
true
[]
false
[Value]
```

Part 2: Taking from the Stack

Add to the `Stack` class a `public String take()` method, which returns the topmost value (i.e., the last value added to the deque) and removes it from the stack.

```
Stack s = new Stack();
System.out.println(s.isEmpty());
System.out.println(s.values());
s.add("Value");
System.out.println(s.isEmpty());
System.out.println(s.values());
String taken = s.take();
System.out.println(s.isEmpty());
```

```
System.out.println(s.values());
System.out.println(taken);
```

Sample output

```
true
[]
false
[Value]
true
[]
Value
```

```
Stack s = new Stack();
s.add("1");
s.add("2");
s.add("3");
s.add("4");
s.add("5");

while (!s.isEmpty()) {
    System.out.println(s.take());
}
```

Sample output

```
5
4
3
2
1
```

Tip! When a value is added to an ArrayList, it goes to the end of the list. As such, the most recently added value is in the last index of the list - the `size()` method provided by the list is useful for finding the last index. You can remove an element from a particular index using the `remove` method provided by the list.

Exercise submission instructions

How to see the solution

Objects in an Instance Variable List

A list that is an object's instance variable can contain objects other than strings as long as the type of objects in the list is specified when defining the list.

In the previous section, we created a class called `AmusementParkRide`, which was used to check whether or not a person was eligible to get on a particular ride. The `Amusement park` class looks like the following.

```
public class AmusementParkRide {
    private String name;
    private int minimumHeight;
    private int visitors;

    public AmusementParkRide(String name, int minimumHeight) {
        this.name = name;
        this.minimumHeight = minimumHeight;
        this.visitors = 0;
    }

    public boolean isAllowedOn(Person person) {
        if (person.getHeight() < this.minimumHeight) {
            return false;
        }

        this.visitors++;
        return true;
    }

    public String toString() {
        return this.name + ", minimum height requirement: " + this.minimumHeigh
            ", visitors: " + this.visitors;
    }
}
```

We'll extend the class so that the amusement park keeps track of the people on the ride. In this version, the ride has, as an instance variable, a list of the people who have been allowed on the ride. The list is created in the constructor.

```
public class AmusementParkRide {
    private String name;
    private int minimumHeigth;
    private int visitors;
    private ArrayList<Person> riding;

    public AmusementParkRide(String name, int minimumHeigth) {
        this.name = name;
        this.minimumHeigth = minimumHeigth;
        this.visitors = 0;
        this.riding = new ArrayList<>();
    }

    // ..
}
```

Let's change the method `isAllowedOn`. The method adds to the list all the persons who meet the height requirements.

```
public class AmusementParkRide {
    private String name;
    private int minimumHeight;
    private int visitors;
    private ArrayList<Person> riding;

    public AmusementParkRide(String name, int minimumHeight) {
        this.name = name;
        this.minimumHeight = minimumHeight;
        this.visitors = 0;
        this.riding = new ArrayList<>();
    }

    public boolean isAllowedOn(Person person) {
        if (person.getHeight() < this.minimumHeight) {
            return false;
        }

        this.visitors++;
        this.riding.add(person);
        return true;
    }
```

```
public String toString() {
    return this.name + ", minimum height requirement: " + this.minimumHeigh
        ", visitors: " + this.visitors;
}
```

Programming exercise:

MessagingService

Points 1/1

The exercise template comes with a pre-defined `Message` class that can be used to create objects representing messages. Each message has a sender and some content.

Implement the `MessagingService` class. The class must have a parameterless constructor and contain a list of `Message` objects. After that, add the following two methods to the class:

- `public void add(Message message)` - adds a message passed as a parameter to the messaging service as long as the message content is at most 280 characters long.
- `public ArrayList<Message> getMessages()` - returns the messages added to the messaging service.

Tip! You can find out the length of the string using the `length()` method associated with the string.

Exercise submission instructions

How to see the solution

Printing an Object from a List

Let's now modify the `toString` method so that the string returned by the method contains the name of each and every person on the ride.

```
public class AmusementParkRide {
    private String name;
    private int minimumHeight;
    private int visitors;
    private ArrayList<Person> riding;

    // ...

    public String toString() {
        // let's form a string from all the people on the list
        String onTheRide = "";
        for (Person person: riding) {
            onTheRide = onTheRide + person.getName() + "\n";
        }

        // we return a string describing the object
        // including the names of those on the ride
        return this.name + ", minimum height requirement: " + this.minimumHeigh
            ", visitors: " + this.visitors + "\n" +
            "riding:\n" + onTheRide;
    }
}
```

Let's test out the extended amusement park ride:

```
Person matti = new Person("Matti");
matti.setWeigth(86);
matti.setHeight(180);

Person juhana = new Person("Juhana");
juhana.setWeigth(34);
juhana.setHeight(132);

AmusementParkRide hurjakuru = new AmusementParkRide("Hurjakuru", 140);
System.out.println(hurjakuru);

System.out.println();

if (hurjakuru.isAllowedOn(matti)) {
    System.out.println(matti.getNimi() + " is allowed on the ride");
} else {
    System.out.println(matti.getNimi() + " is not allowed on the ride");
```

```
}

if (hurjakuru.isAllowedOn(juhana)) {
    System.out.println(juhana.getNimi() + " is allowed on the ride");
} else {
    System.out.println(juhana.getNimi() + " is not allowed on the ride");
}

System.out.println(hurjakuru);
```

The program's output is:

Sample output

```
Hurjakuru, minimum height requirement: 140, visitors: 0
riding:

Matti is allowed on the ride
Juhana is not allowed on the ride
Hurjakuru, minimum height requirement: 140, visitors: 1
riding:
Matti
```

Even though there is no one on the ride, the string `riding:` is on the print output. Let's modify the `toString` method so that if there is no one on the ride, the string returned by the method informs of it.

```
public class AmusementParkRide {
    private String name;
    private int minimumHeight;
    private int visitors;
    private ArrayList<Person> riding;

    public AmusementParkRide(String name, int minimumHeight) {
        this.name = name;
        this.minimumHeight = minimumHeight;
        this.visitors = 0;
        this.riding = new ArrayList<>();
    }

    // ...

    public String toString() {
```

```
        String printOutput = this.name + ", minimum height requirement: " + thi
            ", visitors: " + this.visitors + "\n";

        if (riding.isEmpty()) {
            return printOutput + "no one is on the ride.";
        }

        // we form a string from the people on the list
        String peopleOnRide = "";

        for (Person person: riding) {
            peopleOnRide = peopleOnRide + person.getName() + "\n";
        }

        return printOutput + "\n" +
            "on the ride:\n" + peopleOnRide;
    }
```

The print output has now been improved.

```
Person matti = new Person("Matti");
matti.setWeight(86);
matti.setHeight(180);

Person juhana = new Person("Juhana");
juhana.setWeight(34);
juhana.setHeight(132);

AmusementParkRide hurjakuru = new AmusementParkRide("Hurjakuru", 140);
System.out.println(hurjakuru);

System.out.println();

if (hurjakuru.isAllowedOn(matti)) {
    System.out.println(matti.getName() + " is allowed on the ride");
} else {
    System.out.println(matti.getName() + " is not allowed on the ride");
}

if (hurjakuru.isAllowedOn(juhana)) {
    System.out.println(juhana.getName() + " is allowed on the ride");
} else {
    System.out.println(juhana.getName() + " is not allowed on the ride");
}

System.out.println(hurjakuru);
```

The program's output is:

Sample output

```
Hurjakuru, minimum height requirement: 140, visitors: 0
no one is on the ride.

Matti is allowed on the ride
Juhana is not allowed on the ride
Hurjakuru, minimum height requirement: 140, visitors: 1
on the ride:
Matti
```

Programming exercise:

Printing a Collection

Points 1/1

The exercise template has a predefined `SimpleCollection` class, which is used to represent a group of values. The class is missing the `toString` method used for printing.

Implement a `toString` method for the class that will perform as demonstrated in the following examples.

```
SimpleCollection s = new SimpleCollection("alphabet");
System.out.println(s);

System.out.println();

s.add("a");
System.out.println(s);

System.out.println();

s.add("b");
System.out.println(s);

System.out.println();

s.add("c");
System.out.println(s);
```

Sample output

```
The collection alphabet is empty.

The collection alphabet has 1 element:
a

The collection alphabet has 2 elements:
a
b

The collection alphabet has 3 elements:
a
b
c
```

```
SimpleCollection s = new SimpleCollection("characters");
System.out.println(s);

System.out.println();

s.add("magneto");
System.out.println(s);

System.out.println();

s.add("mystique");
System.out.println(s);

System.out.println();

s.add("phoenix");
System.out.println(s);
```

Sample output

```
The collection characters is empty.

The collection characters has 1 element:
magneto

The collection characters has 2 elements:
magneto
mystique
```

```
The collection characters has 3 elements:
magneto
mystique
phoenix
```

Exercise submission instructions

How to see the solution

Clearing an Object's List

We'll next add a `removeEveryoneOnRide` method to the amusement park ride, which removes each and every person currently on the ride.The list method `clear` is very handy here.

```
public class AmusementParkRIde {
    // ..

    public void removeEveryoneOnRide() {
        this.riding.clear();
    }

    // ..
}
```

```
Person matti = new Person("Matti");
matti.setWeight(86);
matti.setHeight(180);

Person juhana = new Person("Juhana");
juhana.setWeight(34);
juhana.setHeight(132);

AmusementParkRide hurjakuru = new AmusementParkRide("Hurjakuru", 140);
System.out.println(hurjakuru);

System.out.println();
```

```
if (hurjakuru.isAllowedOn(matti)) {
    System.out.println(matti.getName() + " is allowed on the ride");
} else {
    System.out.println(matti.getName() + " is not allowed on the ride");
}

if (hurjakuru.isAllowedOn(juhana)) {
    System.out.println(juhana.getName() + " is allowed on the ride");
} else {
    System.out.println(juhana.getName() + " is not allowed on the ride");
}

System.out.println(hurjakuru);

hurjakuru.removeEveryoneOnRide();

System.out.println();
System.out.println(hurjakuru);
```

The program's output is:

Sample output

```
Hurjakuru, minimum height requirement: 140, visitors: 0
no one is on the ride.

Matti is allowed on the ride.
Juhana is not allowed on the ride
Hurjakuru, minimum height requirement: 140, visitors: 1
on the ride:
Matti

Hurjakuru, minimum height requirement: 140, visitors: 1
no one is on the ride.
```

Calculating a Sum from Objects on a List

Let's now create a method for the amusement park ride that calculates the average height of the people currently on it. Average height can obtained by calculating the average from the persons on the ride — the average is calculated by adding up the individual values and dividing that sum by the number of values.

The implementation underneath returns `-1` if not a single person is on the ride. The result of `-1` is impossible in a program that calculates averages. Based on that, we can determine that the average could not have been calculated.

```
public class AmusementParkRide {
    private String name;
    private int minimumHeight;
    private int visitors;
    private ArrayList<Person> riding;

    // ..

    public double averageHeightOfPeopleOnRide() {
        if (riding.isEmpty()) {
            return -1;
        }

        int sumOfHeights = 0;
        for (Person per: riding) {
            sumOfHeights += per.getHeight();
        }

        return 1.0 * sumOfHeights / riding.size();
    }

    // ..
}
```

```
Person matti = new Person("Matti");
matti.setHeight(180);

Person juhana = new Person("Juhana");
juhana.setHeight(132);

Person awak = new Henkilo("Awak");
awak.setHeight(194);

AmusementParkRide hurjakuru = new AmusementParkRide("Hurjakuru", 140);

hurjakuru.isAllowedOn(matti);
hurjakuru.isAllowedOn(juhana);
hurjakuru.isAllowedOn(awak);
```

```
System.out.println(hurjakuru);
System.out.println(hurjakuru.averageHeightOfPeopleOnRide());
```

The program's output is:

Sample output

```
Hurjakuru, minimum height requirement: 140, visitors: 2
on the ride:
Matti
Awak
187.0
```

Programming exercise:

Santa's Workshop (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.

We'll practise wrapping gifts in this exercise. Let's create the classes `Gift` and `Package`. The gift has a name and weight, and the package contains gifts.

Part 1: Gift-class

Create a `Gift` class, where the objects instantiated from it represent different kinds of gifts. The information that's recorded is the name and weight of the item (kg).

Add the following methods to the class:

- Constructor for which the name and weight of the gift are given as parameters
- Method `public String getName()`, which returns the name of the gift

- Method `public int getWeight()`, which returns the weight of the gift
- Method `public String toString()`, which returns a string in the form "name (weight kg)"

The following is an example of the class in use:

```
public class Main {
    public static void main(String[] args) {
        Gift book = new Gift("Harry Potter and the Philosopher's Stone", 2)

        System.out.println("Gift's name: " + book.getName());
        System.out.println("Gift's weight: " + book.getWeight());

        System.out.println("Gift: " + book);
    }
}
```

The program's print output should be as follows:

Sample output

```
Gift's name: Harry Potter and the Philosopher's Stone
Gift's weight: 2
Gift: Harry Potter and the Philosopher's Stone (2 kg)
```

Part 2: Package-class

Create a `Package` class to which gifts can be added, and that keeps track of the total weight of the gifts in the package. The class should contain:

- A parameterless constructor
- Method `public void addGift(Gift gift)`, which adds the gift passed as a parameter to the package. The method returns no value.
- Method `public int totalWeight()`, which returns the total weight of the package's gifts.

It's recommended to store the items in an `ArrayList` object.

```
ArrayList<Gift> gifts = new ArrayList<>();
```

An example use case of the class is as follows:

```
public class Main {
    public static void main(String[] args) {
        Gift book = new Gift("Harry Potter and the Philosopher's Stone", 2)

        Package gifts = new Package();
        gifts.addGift(book);
        System.out.println(gifts.totalWeight());
    }
}
```

The program's output should be the following:

Sample output

```
2
```

Exercise submission instructions

How to see the solution

Retrieving a Specific Object from a List

We'll now create a method for the amusement park ride that returns the tallest person on the ride. As such, the method should both retrieve the tallest person from the list and return it.

Methods that retrieve objects from a list should be implemented in the following way. First off, we'll check whether or not the list is empty - if it is, we return a `null` reference or some other value indicating that the list had no values. After that, we create an object reference variable that

describes the object to be returned. We set the first object on the list as its value. We then go through the values on the list by comparing each list object with the object variable representing the object to be returned. If the comparison finds a better matching object, its assigned to the object reference variable to be returned. Finally, we return the object variable describing the object that we want to return.

```
public Person getTallest() {
    // return a null reference if there's no one on the ride
    if (this.riding.isEmpty()) {
        return null;
    }

    // create an object reference for the object to be returned
    // its first value is the first object on the list
    Person returnObject = this.riding.get(0);

    // go through the list
    for (Person prs: this.riding) {
        // compare each object on the list
        // to the returnObject -- we compare heights
        // since we're searching for the tallest,

        if (returnObject.getHeight() < prs.getHeight()) {
            // if we find a taller person in the comparison,
            // we assign it as the value of the returnObject
            returnObject = prs;
        }
    }

    // finally, the object reference describing the
    // return object is returned
    return returnObject;
}
```

Finding the tallest person is now easy.

```
Person matti = new Person("Matti");
matti.setHeight(180);

Person juhana = new Person("Juhana");
juhana.setHeight(132);

Person awak = new Person("Awak");
awak.setHeight(194);
```

```
AmusementParkRide hurjakuru = new AmusementParkRide("Hurjakuru", 140);

hurjakuru.isAllowedOn(matti);
hurjakuru.isAllowedOn(juhana);
hurjakuru.isAllowedOn(awak);

System.out.println(hurjakuru);
System.out.println(hurjakuru.averageHeightOfPeopleOnRide());

System.out.println();
System.out.println(hurjakuru.getTallest().getName());
Person tallest = hurjakuru.getTallest();
System.out.println(tallest.getName());
```

Sample output

```
Hurjakuru, minimum height requirement: 140, visitors: 2
on the ride:
Matti
Awak
187.0

Awak
Awak
```

Programming exercise:

Longest in collection

Points
1/1

The exercise template comes with the class `SimpleCollection` that's familiar from previous exercises. Implement the method `public String longest()` for the class, which returns the longest string of the collection. If the collection is empty, the method should return a `null` reference.

```
SimpleCollection j = new SimpleCollection("characters");
System.out.println("Longest: " + j.longest());

j.add("magneto");
j.add("mystique");
```

```
j.add("phoenix");

System.out.println("Longest: " + j.longest());
```

Sample output

```
Longest: null
Longest: mystique
```

Exercise submission instructions

How to see the solution

Programming exercise:

Height Order (3 parts)

Points 3/3

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.

A `Person` class is included in the exercise template. A person has a name and a height. In this exercise, we'll implement a `Room` class, which can be used to add people and order them according to their height — taking a person out of the room always returns the shortest person.

The class should eventually work in the following way.

Part 1: Room

Create `Room` class. The class should contain a list of persons as an instance variable, and it should have a parameterless constructor. In addition, add the following methods to the class:

- `public void add(Person person)` - add the person passed as a paramter to the list.
- `public boolean isEmpty()` - returns a `boolean`-type value `true` or `false`, that tells whether the room is empty or not.
- `public ArrayList<Person> getPersons()` - returns a list of the persons in the room.

```
Room room = new Room();
System.out.println("Empty room? " + room.isEmpty());
room.add(new Person("Lea", 183));
room.add(new Person("Kenya", 182));
room.add(new Person("Auli", 186));
room.add(new Person("Nina", 172));
room.add(new Person("Terhi", 185));
System.out.println("Empty room? " + room.isEmpty());

System.out.println("");
for (Person person : room.getPersons()) {
    System.out.println(person);
}
```

Sample output

```
Empty room? true
Empty room? false

Lea (183 cm)
Kenya (182 cm)
Auli (186 cm)
Nina (172 cm)
Terhi (185 cm)
```

Part 2: Shortest person

Add a `public Person shortest()` method to the `Room` class, which returns the shortest person added to the room. If the room is empty, a null reference is returned. The method should not remove a person from the room.

```
Room room = new Room();
System.out.println("Shortest: " + room.shortest());
System.out.println("Empty room? " + room.isEmpty());
room.add(new Person("Lea", 183));
room.add(new Person("Kenya", 182));
room.add(new Person("Auli", 186));
room.add(new Person("Nina", 172));
room.add(new Person("Terhi", 185));
System.out.println("Empty room? " + room.isEmpty());

System.out.println("");
for (Person person : room.getPersons()) {
    System.out.println(person);
}

System.out.println();
System.out.println("Shortest: " + room.shortest());
System.out.println("");
for (Person person : room.getPersons()) {
    System.out.println(person);
}
```

Sample output

```
Shortest: null
Empty room? true
Empty room? false

Lea (183 cm)
Kenya (182 cm)
Auli (186 cm)
Nina (172 cm)
Terhi (185 cm)

Shortest: Nina (172 cm)

Lea (183 cm)
Kenya (182 cm)
Auli (186 cm)
Nina (172 cm)
Terhi (185 cm)
```

Part 3: Taking from a room

Add a `public Person take()` method to the `Room` class, which takes the shortest person in the room. When a room is empty, it returns a `null` reference.

```
Room room = new Room();
room.add(new Person("Lea", 183));
room.add(new Person("Kenya", 182));
room.add(new Person("Auli", 186));
room.add(new Person("Nina", 172));
room.add(new Person("Terhi", 185));

System.out.println("");
for (Person person : room.getPersons()) {
    System.out.println(person);
}

System.out.println();
System.out.println("Shortest: " + room.take());
System.out.println("");
for (Person person : room.getPersons()) {
    System.out.println(person);
}
```

Sample output

```
Lea (183 cm)
Kenya (182 cm)
Auli (186 cm)
Nina (172 cm)
Terhi (185 cm)

Shortest: Nina (172 cm)

Lea (183 cm)
Kenya (182 cm)
Auli (186 cm)
Terhi (185 cm)
```

It's now possible to print the persons in height order.

```
Room room = new Room();
room.add(new Person("Lea", 183));
room.add(new Person("Kenya", 182));
```

```
room.add(new Person("Auli", 186));
room.add(new Person("Nina", 172));
room.add(new Person("Terhi", 185));

while (!room.isEmpty()) {
    System.out.println(room.take());
}
```

Sample output

Nina (172 cm)
Kenya (182 cm)
Lea (183 cm)
Terhi (185 cm)
Auli (186 cm)

Exercise submission instructions

How to see the solution

Programming exercise:

Cargo hold (7 parts)

Points
7/7

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.

In this exercise, we create the classes `Item`, `Suitcase` and `Hold` to practise the use of objects containing other objects.

Part 1: Item-class

Create an `Item` class from which objects can be instantiated to represent different items. The information to store is the name and weight of the item (kg).

Add the following methods to the class:

- Constructor that takes the name and the weight of the item as parameters
- Method `public String getName()`, which returns the item's name
- Method `public int getWeight()`, which returns the item's weight
- Method `public String toString()`, which returns the string "name (weight kg)"

The following is an example of the class in use:

```
public class Main {
    public static void main(String[] args) {
        Item book = new Item("The lord of the rings", 2);
        Item phone = new Item("Nokia 3210", 1);

        System.out.println("The book's name: " + book.getName());
        System.out.println("The book's weight: " + book.getWeight());

        System.out.println("Book: " + book);
        System.out.println("Phone: " + phone);
    }
}
```

The program's print output should be the following:

Sample output

```
The book's name: Lord of the rings
The book's weight: 2
Book: Lord of the rings (2 kg)
Phone: Nokia 3210 (1 kg)
```

Part 2: Suitcase-class

Create a `Suitcase` class. The suitcase has items and a maximum weight that determines the maximum total weight of the items.

Add the following methods to the class:

- Constructor, to which the maximum weight is provided
- The method `public void addItem(Item item)`, which adds the item passed as a parameter to the suitcase. The method does not return a value.
- The method `public String toString()`, which returns the string "x items (y kg)"

It's advisable to store the items in an `ArrayList` object:

```
ArrayList<Item> items = new ArrayList<>();
```

The class `Suitcase` should ensure that the total weight of the items within it does not exceed the maximum weight limit. If that limit would be exceeded as a result of the item to be added, the method `addItem` should not add the new item to the suitcase.

The following is an example use case of the class:

```
public class Main {
    public static void main(String[] args) {
        Item book = new Item("Lord of the rings", 2);
        Item phone = new Item("Nokia 3210", 1);
        Item brick = new Item("brick", 4);

        Suitcase suitcase = new Suitcase(5);
        System.out.println(suitcase);

        suitcase.addItem(book);
        System.out.println(suitcase);

        suitcase.addItem(phone);
        System.out.println(suitcase);

        suitcase.addItem(brick);
        System.out.println(suitcase);
    }
}
```

The program's output should be the following:

Sample output

```
0 items (0 kg)
1 items (2 kg)
2 items (3 kg)
2 items (3 kg)
```

Part 3: Language Formatting

The statement "1 items" is not exactly proper English — a better form would be "1 item". The lack of items could also be expressed as "no items". Implement this change to the toString method of the `Suitcase` class.

The output of the previous program should now look as follows:

Sample output

```
no items (0 kg)
1 item (2 kg)
2 items (3 kg)
2 items (3 kg)
```

Part 4: All items

Add the following methods to the `Suitcase` class:

- a `printItems` method, which prints all the items in the suitcase
- a `totalWeight` method, which returns the total weight of the items

The following is an example use case of the class:

```
public class Main {
    public static void main(String[] args) {
        Item book = new Item("Lord of the rings", 2);
        Item phone = new Item("Nokia 3210", 1);
        Item brick = new Item("brick", 4);

        Suitcase suitcase = new Suitcase(10);
        suitcase.addItem(book);
```

```
        suitcase.addItem(phone);
        suitcase.addItem(brick);

        System.out.println("The suitcase contains the following items:");
        suitcase.printItems();
        System.out.println("Total weight: " + suitcase.totalWeight() + " kg
    }
```

The program's output should be the following:

Sample output

```
The suitcase contains the following items:
Lord of the rings (2 kg)
Nokia 3210 (1 kg)
Brick (4 kg)
Total Weight: 7 kg
```

Make a further modification to your class so that you only use two instance variables. One holds the maximum weight, the other is the list of items in the suitcase.

Part 5: Heaviest item

Add to the `Suitcase` class a `heaviestItem` method, which returns the largest item based on weight. If several items share the heaviest weight, the method can return any one of them. The method should return an object reference. If the suitcase is empty, return the value *null*.

The following is an example of the class in use:

```
public class Main {
    public static void main(String[] args) {
        Item book = new Item("Lord of the rings", 2);
        Item phone = new Item("Nokia 3210", 1);
        Item brick = new Item("Brick", 4);

        Suitcase suitcase = new Suitcase(10);
        suitcase.addItem(book);
        suitcase.addItem(phone);
        suitcase.addItem(brick);
```

```
        Item heaviest = suitcase.heaviestItem();
        System.out.println("Heaviest item: " + heaviest);
    }
}
```

The program should print the following:

Sample output

```
Heaviest item: Brick (4 kg)
```

Part 6: Hold-class

Make a `Hold` class with the following methods:

- a constructor, to which the maximum weight is given
- method `public void addSuitcase(Suitcase suitcase)` that adds the specified luggage to the hold
- method `public String toString()` that returns the string "x suitcases (y kg)"

Store your suitcases in a suitable `ArrayList` structure.

The class `Hold` has to ensure that the total weight of the suitcases it contains does not exceed the maximum weight. Should the maximum weight be exceeded due to the addition of new luggage, the `addSuitcase` method should not add the new suitcase.

The following is an example of the class in use:

```
public class Main {
    public static void main(String[] args) {
        Item book = new Item("Lord of the rings", 2);
        Item phone = new Item("Nokia 3210", 1);
        Item brick = new Item("brick", 4);

        Suitcase adasCase = new Suitcase(10);
        adasCase.addItem(book);
        adasCase.addItem(phone);

        Suitcase pekkasCase = new Suitcase(10);
```

```
        pekkasCase.addItem(brick);

        Hold hold = new Hold(1000);
        hold.addSuitcase(adasCase);
        hold.addSuitcase(pekkasCase);

        System.out.println(hold);
    }
}
```

The program's output should be the following:

Sample output

```
2 suitcases (7 kg)
```

Part 7: The Hold's Contents

Add to the `Hold` class the method `public void printItems()` that prints all the items contained in the hold's suitcases.

The following is an example of the class in use:

```
public class Main {
    public static void main(String[] args) {
        Item book = new Item("Lord of the rings", 2);
        Item phone = new Item("Nokia 3210", 1);
        Item brick = new Item("brick", 4);

        Suitcase adasCase = new Suitcase(10);
        adasCase.addItem(book);
        adasCase.addItem(phone);

        Suitcase pekkasCase = new Suitcase(10);
        pekkasCase.addItem(brick);

        Hold hold = new Hold(1000);
        hold.addSuitcase(adasCase);
        hold.addSuitcase(pekkasCase);

        System.out.println("The suitcases in the hold contain the following
        hold.printItems();
```

```
    }
}
```

The program's output should be as follows:

Sample output

```
The suitcases in the hold contain the following items:
Lord of the rings (2 kg)
Nokia 3210 (1 kg)
brick (4 kg)
```

Exercise submission instructions

How to see the solution

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

→ 2. Separating the user interface from program logic

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

In this part:

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.

HELSINGIN YLIOPISTO
HELSINGFORS UNIVERSITET
UNIVERSITY OF HELSINKI

MASSIIVISET AVOIMET VERKKOKURSSIT
MASSIVE OPEN ONLINE COURSES · MOOC.FI