Objects in a list

Learning Objectives

- You can add objects to a list
- You can go through objects in a list

The type parameter used in creating a list defines the type of the variables that are added to the list. For instance, `ArrayList<String>` includes strings, `ArrayList<Integer>` integers, and `ArrayList<Double>` floating point numbers

In the example below we first add strings to a list, after which the strings in the list are printed one by one.

```
ArrayList<String> names = new ArrayList<>();

// string can first be stored in a variable
String name = "Betty Jennings";
// then add it to the list
names.add(name);

// strings can also be directly added to the list:
names.add("Betty Snyder");
names.add("Frances Spence");
names.add("Kay McNulty");
names.add("Marlyn Wescoff");
names.add("Ruth Lichterman");

// several different repeat statements can be
// used to go through the list elements

// 1. while loop
int index = 0;
while (index < names.size()) {
    System.out.println(names.get(index));
    index = index + 1;
}

// 2. for loop with index
for (int i = 0; i < names.size(); i++) {
    System.out.println(names.get(i));
}

System.out.println();
// 3. for each loop (no index)
for (String name: names) {
    System.out.println(name);
}
```

Sample output

```
Betty Jennings
Betty Snyder
Frances Spence
Kay McNulty
Marlyn Wescoff
Ruth Lichterman

Betty Jennings
Betty Snyder
Frances Spence
Kay McNulty
Marlyn Wescoff
Ruth Lichterman

Betty Jennings
Betty Snyder
Frances Spence
Kay McNulty
Marlyn Wescoff
Ruth Lichterman
```

Adding object to a list

Strings are objects, so it should come as no surprise that other kinds of objects can also be found in lists. Next, let's examine the cooperation of lists and objects in more detail.

Let's assume we have access to the class defined below, describing a person.

```java
public class Person {

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

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

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

    public int getAge() {
        return this.age;
    }

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

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

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

    public double bodyMassIndex() {
        double heightDivByHundred = this.height / 100.0;
        return this.weight / (heightDivByHundred * heightDivByHundred);
    }

    @Override
    public String toString() {
        return this.name + ", age " + this.age + " years";
    }
}
```

Handling objects in a list is not really different in any way from the previous experience we have with lists. The essential difference is only to define the type for the stored elements when you create the list.

In the example below we first create a list meant for storing Person type object, after which we add persons to it. Finally the person objects are printed one by one.

```
ArrayList<Person> persons = new ArrayList<>();

// a person object can be created first
Person john = new Person("John");
// and then added to the list
persons.add(john);

// person objects can also be created "in the same sentence" that they are adde
persons.add(new Person("Matthew"));
persons.add(new Person("Martin"));

for (Person person: persons) {
    System.out.println(person);
}
```

Sample output

```
John, age 0 years
Matthew, age 0 years
Martin, age 0 years
```

Adding user-inputted objects to a list

The structure we used earlier for reading inputs is still very useful.

```
Scanner scanner = new Scanner(System.in);
ArrayList<Person> persons = new ArrayList<>();

// Read the names of persons from the user
while (true) {
    System.out.print("Enter a name, empty will stop: ");
    String name = scanner.nextLine();
    if (name.isEmpty()) {
        break;
    }


    // Add to the list a new person
    // whose name is the previous user input
    persons.add(new Person(name));
}

// Print the number of the entered persons, and their individual information
System.out.println();
System.out.println("Persons in total: " + persons.size());
System.out.println("Persons: ");

for (Person person: persons) {
    System.out.println(person);
}
```

Sample output

```
Enter a name, empty will stop: Alan Kay
Enter a name, empty will stop: Ivan Sutherland
Enter a name, empty will stop: Kristen Nygaard

Persons in total: 3
Persons:
Alan Kay, age 0 years
Ivan Sutherland, age 0 years
Kristen Nygaard, age 0 years
```

Programming exercise:

Items

Points

1/1

Implement the class `Items` described here. **NB!** Don't modify the class `Item`.

Write a program that reads names of items from the user. If the name is empty, the program stops reading. Otherwise, the given name is used to create a new item, which you will then add to the `items` list.

Having read all the names, print all the items by using the `toString` method of the `Item` class. The implementation of the `Item` class keeps track of the time of creation, in addition to the name of the item.

An example of the working program is given below:

Sample output

```
Name: Hammer
Name: Collar
Name:

Hammer (created at: 06.07.2018 12:34:56)
Collar (created at: 06.07.2018 12:34:57)
```

Exercise submission instructions

How to see the solution

Multiple constructor parameters

If the constructor demands more than one parameter, you can query the user for more information. Let's assume we have the following constructor for the class `Person`.

```java
public class Person {

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

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

    // methods
}
```

In this case, an object is created by calling the two-parameter constructor.

If we want to query the user for this kind of object, they must be asked for each parameter separately. In the example below, name and age parameters are asked separately from the user. Entering an empty name will end the reading part.

The persons are printed after they have been read.

```java
Scanner scanner = new Scanner(System.in);
ArrayList<Person> persons = new ArrayList<>();

// Read person information from the user
while (true) {
    System.out.print("Enter name, empty will end: ");
    String name = scanner.nextLine();
    if (name.isEmpty()) {
        break;
    }

    System.out.print("Enter the age of the person " + name + ": ");

    int age = Integer.valueOf(scanner.nextLine());

    // We add a new person to the list.
    // The person's name and age were decided by the user
    persons.add(new Person(name, age));
}

// We'll print the number of the inputted persons, and the persons themselves
System.out.println();
System.out.println("Total number of persons: " + persons.size());
System.out.println("Persons: ");

for (Person person: persons) {
    System.out.println(person);
}
```

Sample output

```
Enter name, empty will end: Grace Hopper
Enter the age of the person Grace Hopper: 85
Enter name, empty will end:

Total number of persons: 1
Persons:
Grace Hopper, age 85 years
```

Programming exercise:

Personal information

Points 1/1

The program described here should be implemented in the class `PersonalInformationCollection`. **NB!** Do not modify the class `PersonalInformation`.

After the user has entered the last set of details (they enter an empty first name), exit the repeat statement.

Then print the collected personal information so that each entered object is printed in the following format: first and last names separated by a space (you don't print the identification number). An example of the working program is given below:

Sample output

```
First name: Jean
Last name: Bartik
Identification number: 271224
First name: Betty
Last name: Holberton
Identification number: 070317
First name:

Jean Bartik
Betty Holberton
```

Exercise submission instructions

How to see the solution

Reading input in a specific format

In the example and exercise below, the required information was entered line by line. By no means is it impossible to ask for input in a specific format, e.g. separated by a comma.

If the name and age were separated by a comma, the program could work in the following manner.

```
Scanner scanner = new Scanner(System.in);
ArrayList<Person> persons = new ArrayList<>();

// Read person information from the user
System.out.println("Enter the person details separated by a comma, e.g.: Ra
while (true) {
    System.out.print("Enter the details, empty will stop: ");
    String details = scanner.nextLine();
    if (details.isEmpty()) {
        break;
    }

    String[] parts = details.split(",");
    String name = parts[0];
    int age = Integer.valueOf(parts[1]);
    persons.add(new Person(name, age));
}

// Print the number of the entered persons, and the persons themselves
System.out.println();
System.out.println("Total number of persons: " + persons.size());
System.out.println("Persons: ");

for (Person person: persons) {
    System.out.println(person);
}
```

Sample output

```
Enter the person details separated by a comma, e.g.: Randall,2
Enter the details, empty will stop: Leevi,2
Enter the details, empty will stop: Anton,2
Enter the details, empty will stop: Sylvi,0
Enter the details, empty will stop:

Total number of persons: 3
Persons:
Leevi, age 2 years
Anton, age 2 years
Sylvi, age 0 years
```

Filtered printing from the list

You can also examine the objects on the list as you go through it. In the example below, we first ask the user for an age restriction, after which we print all the objects whose age is at least the number given by the user.

```java
// Assume we have a 'persons' list
// that consists of person objects

System.out.print("What is the age limit? ");
int ageLimit = Integer.valueOf(scanner.nextLine());

for (Person person: persons) {
    if (person.getAge() >= ageLimit) {
        System.out.println(person);
    }
}
```

Programming exercise:

Television programs

Points

1/1

In the exercise template there is a ready-made class TelevisionProgram, representing a television program. The class has object variables name and duration, a constructor, and a few methods.

Implement a program that begins by reading television programs from the user. When the user inputs an empty string as the name of the program, the program stops reading programs.

After this the user is queried for a maximum duration. Once the maximum is given, the program proceeds to list all the programs whose duration is smaller or equal to the specified maximum duration.

Sample output

```
Name: Rick and Morty
Duration: 25
Name: Two and a Half Men
Duration: 30
Name: Love it or list it
Duration: 60
Name: House
Duration: 60

Program's maximum duration? 30
Rick and Morty, 25 minutes
Two and a Half Men, 30 minutes
```

Exercise submission instructions

How to see the solution

Programming exercise:

Books

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.

Write a program that first reads book information from the user. The details to be asked for each book include the title, the number of pages and the publication year. Entering an empty string as the name of the book ends the reading process.

After this the user is asked for what is to be printed. If the user inputs "everything", all the details are printed: the book titles, the numbers of pages and the publication years. However, if the user enters the string "name", only the book titles are printed.

It is probably worthwhile to implement a class called `Book` to represent a book. There are two points in total available for this exercise.

Sample output

```
Title: To Kill a Mockingbird
Pages: 281
Publication year: 1960
Title: A Brief History of Time
Pages: 256
Publication year: 1988
Title: Beautiful Code
Pages: 593
Publication year: 2007
Title: The Name of the Wind
Pages: 662
Publication year: 2007
Title:

What information will be printed? everything
To Kill a Mockingbird, 281 pages, 1960
A Brief History of Time, 256 pages, 1988
Beautiful Code, 593 pages, 2007
The Name of the Wind, 662 pages, 2007
```

Sample output

```
Title: To Kill a Mockingbird
Pages: 281
Publication year: 1960
Title: A Brief History of Time
Pages: 256
Publication year: 1988
Title: Beautiful Code
Pages: 593
Publication year: 2007
Title: The Name of the Wind
Pages: 662
Publication year: 2007
Title:

What information will be printed? name
To Kill a Mockingbird
A Brief History of Time
Beautiful Code
The Name of the Wind
```

Exercise submission instructions

How to see the solution

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

→ 3. Files and reading data

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

In this part: