# Learning goals

Before the next day, you should have achieved the following learning goals:

- Be able to create new methods, with or without parameters.
- Be able to create new methods, free or inside classes.
- Be able to use methods to avoid repeating code.
- Understand the scope of a variable in a piece of code.
- Understand how memory changes (both stack and heap) when a method is called and executed, in particular when several methods call each other.

Remember that star exercises are more difficult. **Do not try star-exercises unless the other ones are clear to you**.

# 1 Scope

Look at the following code (with line numbers for clarity) and say where each of the following variables is visible: i, j, newSize, size.

```
01  class UnitMatrix {
02    int size;
03
04    void setSize(int newSize) {
05      this.size = newSize;
06    }
07
08    void print() {
09      for (int i = 0; i < size; i++) {
10        for (int j = 0; j < size; j++) {
11          if (i == j) {
12            println("1 ");
13          } else {
14            println("0 ");
15          }
16        }
17        println ""
18      }
19    }
20  }
```

Check your answers with the faculty members in the lab.

# 2 Pointer, arrows...

## 2.1 a)

Take the example code from the notes:

```
class Point {
    int x;
    int y;
}
// This method increments the int by 1 and
// moves the point to the right
```

```
void increment(Point point, int n) {
    n = n + 1;
    point.x = point.x + 1;
    point = null;
    println "  At the end of the method..."
    println "  The integer is " + n;
    println "  The point is " + point;
}
// Program execution starts here
Point myPoint = new Point();
myPoint.x = 0;
myPoint.y = 0;
int myInt = 0;
println "The integer is now " + myInt;
println "The point is now " + myPoint.x + "," + myPoint.y;
println "Calling method increment(Point, int)..."
increment(myPoint, myInt);
println "The integer is now " + myInt;
println "The point is now " + myPoint.x + "," + myPoint.y;
```

Write detailed diagrams that show what variables are there in the "stack" and what objects they point to in the "heap" (if they are complex types).

## 2.2 b)

Now do the same for this example code from Day 3:

```
class Person {
    int age;
    String name;
    Person father;
    Person mother;
}
Person john = new Person();
john.name = "John Smith";
john.age = 35;
Person mary = new Person();
mary.name = "Mary Smith";
mary.age = 32;
Person student = new Person();
student.name = "John Smith, Jr.";
student.age = 5;
student.father = john
student.mother = mary
println "TEACHER: How old are you, " + student.name + "?"
println "LITTLE JOHN: I am " + student.age + " years old, sir.";
println "TEACHER: Who is your mother?"
println "LITTLE JOHN: " + student.mother.name + ", sir.";
```

Check your diagrams with the faculty members in the lab.

## 2.3 Flow of execution

Look at the following code (with line numbers for clarity):

```
1   String requestUser() {
2       String result = System.console().readLine();
3       return result;
```

```
 4  }
 5
 6  void createUser() {
 7      String user = requestUser();
 8      while (!isValidUser(user)) {
 9          println("That name is not valid. Please try again.);
10          user = requestUser();
11      }
12      insertUserInDB(user);
13  }
14
15  void deleteUser() {
16      String user = requestUser();
17      while (!isValidUser(user)) {
18          println("That name is not valid. Please try again.);
19          user = requestUser();
20      }
21      if (existsInDB(user) {
22          deleteUserInDB(user);
23      } else {
24          println "That user does not exist."
25      }
26  }
27
28  void insertUserInDB(String user) {
29      // do things with DB that we will cover in later weeks
30  }
31
32  void deleteUserFromDB(String user) {
33      // do things with DB that we will cover in later weeks
34  }
35
36  boolean existsInDB(String user) {
37      // do things with DB that we will cover in later weeks
38  }
39
40  boolean isValidUser(String login) {
41      boolean loginIsValid = true;
42      for (int i = 0; i < login.length(); i++) {
43          char c = login.charAt(i);
44          if (!Character.isLetter(c) || !Character.isLowerCase(c)) {
45              loginIsValid = false;
46          }
47      }
48      return true;
49  }
50
51  boolean running = true;
52  while (running) {
53      println "What would you like to do?";
54      println "1 - Enter a new user";
55      println "2 - Delete a user";
56      println "0 - Exit";
57      print   "> ";
58      String str = System.console().readLine();
59      int option = Integer.parseInt(str);
```

```
60        switch (option) {
61        case 0: running = false;
62                break;
63        case 1: createUser();
64                break;
65        case 2: deleteUser();
66                break;
67        default: println "Invalid option. Please try again."
68        }
69    }
```

Follow the execution of the code as the user enters the following sequences of inputs:

- 4, 0
- 1, john, 0
- 1, john smith, johnsmith, 0
- 2, userNotInDB, 0
- 1, john, 2, john, 0

Hint: all of them start at 40.

# 3 Binary and decimal

Create a program in which you define the following methods:

**power(int, int):** Takes a base $b$ and an exponent $e$ from the user, and returns the result of $b^e$.

**power2(int):** Takes an exponent $e$ from the user and returns the result of $2^e$. This method must call the previous one to find out the result.

**binary2decimal(String):** Takes from the user a binary number (with digits 0 and 1) and returns the corresponding number in decimal (base-10, with digits between 0 and 9). Hint: in the same way that you know that $35 = 3 \cdot 10^1 + 5 \cdot 10^0$, you can find that $100011 = 1 \cdot 2^5 + 1 \cdot 2^1 + 1 \cdot 2^0$. This method must call the previous one to find out the result.

**decimal2binary(int):** The opposite of the previous one: takes a decimal number and returns the corresponding binary number. Hint: instead of multiplying by 2, you will need to divide by two this time (the quotients and the last remainder will give you the binary number).

The program must offer a menu to the user with two options. The first one takes a binary number from the user and returns the corresponding decimal number. The second one does the opposite: takes a decimal number and returns a binary number. The program should use the methods defined.

# 4 Binary and hexadecimal (*)

Binary numbers can be quite long. A 32-bit memory address looks like 1001 0101 0110 1010 1011 0010 1001 1010, which is difficult to handle. That is why memory addresses and other binary numbers are usually written as *hexadecimal* numbers. An hexadecimal number is a base-16 number, with digits between 0 and f (f is equivalent to decimal 15, e is equivalent to decimal 14, and so on). An hexadecimal number is equivalent to a four-digit binary number, which makes them quite compact. The former address reads 956ab29a, which is easier to read and write. To prevent confusion with decimal numbers, hexadecimal numbers are usually prefixed by "0x", as in 0x95 (which is 149) or 0xff (which is 255).

Write a program that takes a String. The string can be a decimal or a hexadecimal number (starting by "0x"). Your program must recognise the number as what it is and convert it to the other base. Use two methods for conversion as in the former exercise.

## 5 More on points

Write a program in which you create and use a class called Point, with two fields of type double (e.g. x, y) and the following methods:

**distanceTo(Point):** calculates the distance to another point.

**distanceToOrigin():** calculates the distance to the origin. Implement it by calling the first method.

**moveTo(double x, double y):** changes the coordinates of this point to be the given parameters x and y.

**moveTo(Point):** changes the coordinates of this point to move where the given point is.

**clone():** returns a copy of the current point with the same coordinates.

**opposite():** returns a copy of the current point with the coordinates multiplied by $-1$.

Two methods in a class can have the same name (identifier) as long as their parameters are different. This is called *method overloading* and we will see more of that in the future.

## 6 Integer

Create your own version of boxed `int`! Create a class Integer2 with only one field (`int value`) and the following methods:

**getValue():** returns the value of this number as an `int`, a getter.

**setValue(int):** a setter.

**isEven():** returns true if the number is even, false otherwise.

**isOdd():** the opposite.

**prettyPrint():** prints the value of the integer on the screen.

**toString():** returns a String equivalent to the number.

Check that it works by using the following program:

```
Integer2 i2 = new Integer2();
print "Enter a number: ";
String str = System.console().readLine();
int i = Integer.parseInt(str);
i2.setValue(i);
print "The number you entered is "
if (i2.isEven()) {
    println "even.";
} else if (i2.isOdd()) {
    println "odd.";
} else {
    println "undefined!! Your code is buggy!";
}
int parsedInt = Integer.parseInt(i2.toString());
if (parsedInt == i2.getValue()) {
    println("Your toString() method seems to work fine.");
}
```

# 7 A bit more practice with doubles

The goal of this exercise is providing additional practice with double-precision numbers.

Write a program that asks the user for the total amount borrowed for a mortgage, the number of years to pay it back, and the interest rate (in this exercise, we assume it is a fixed rate). The program can then calculate how much must be paid at the end

$$t = c \cdot \left(1 + \frac{r}{100}\right)$$

where $t$ is the total, $c$ is the cost, and $r$ is the rate as a percentage. The program should print:

- The total amount to be paid
- The money to be paid every year
- The number of years before the interest is paid and only the initial capital remains

Write a method to calculate each piece of data.