Programming 1

Aryobarzan Atashpendar (aryobarzan.atashpendar@uni.lu)

Lab 5 – Arrays, Methods

HEARTSTEEL © *Riot Games*

Exercise 1 – HEARTSTEEL

As seen in class, it is possible to declare blocks of code that can be re-used in Java. Such blocks of code, referred to as methods, can use their own variables and can optionally return a value that can be used by the rest of your code. Here are two example methods:

```
public class MyClass{
    // We consider static methods for now. Non-static methods will be covered later during the course!
    // The syntax of a method is as follows: access_modifier {static} return_type name(parameters) { body }
    // - access_modifier: you will see this in a later lecture.
    // - static: optional
    // - return_type: the type of the value that will be returned by the method.
    //                If no value will be returned, use the special type 'void'.
    // - name: the name of your method.
    // - parameters: optionally, indicate parameters. These act as inputs to the method, allowing you to pass values from outside
    //               of your method to the method, which can be used in the method's body.
    // - body: the code to be executed when the method is called.

    public static void methodVoid(){
    // Notice that this method does not return a value because its return type is 'void'.
    }
    public static int methodInt(int number){
        // This method has a parameter called "number", which is a value passed to the method when it is called.
        int internalVariable = number * 2;
        return internalVariable; // At the end of its body's execution, the method will return some value of type int.
    }
}
```

In this exercise, you'll be applying everything you've learned up till now, as well as arrays and methods, to implement a slightly more complex program.

Specifically, you are managing the fictional musical boy group "HEARTSTEEL", who have released their new song "PARANOIA". Your program should allow you to keep track of and modify several properties of the group and their song:

- *leader*: the current leader of the group can be changed by entering their name in the console. The initial leader is "yone".
- *bpm*: the "beats-per-minute" of the song can be changed by entering a new integer value in the console. If the user enters a BPM less than or equal to 20, ask them for a new value. If the user enters a BPM that is lower than the previous BPM, confirm if the user wants to use the new, lower BPM by asking them to enter true or false in the console. The initial bpm is 240.
- *sale*: the percentage that the song's price is on sale for can be changed by entering a new float value in the console. Only values in the range [0.0, 1.0] are valid. Also, if the new sale percentage is lower than the previous percentage, keep the old percentage. The initial sale is 0.
- *chartings*: this array keeps track of the last four chart rankings of the song, which you should initialize with [4, 2, 5, 1]. The user can indicate the latest charting by entering an integer value in the console. The value must be in the range [1, 100]. If it is valid, update the chartings by firstly shifting the existing values by 1 position to the right. Then, set the first element to the new value. For example, if the user enters the new charting 20, the new chartings would be [20, 4, 2, 5].

When the program is launched, have an input process where you repeatedly ask the user what value they want to update:

- (1) Change current leader
- (2) Update BPM
- (3) Update sale
- (4) Update charting
- (0) Exit

Your solution should define the blocks of code that read the user's input for the various properties as the followings methods:

```
public static String changeLeader(Scanner sc) {
    // Return the new leader's name that was input in the console.
}

public static int inputBPM(int previousBPM, Scanner sc) {
   // Validate the new BPM that was input in the console and return it.
}

public static float inputSale(float previousSale, Scanner sc) {
  // Validate the new sale percentage that was input in the console and return it.
}

public static int[] inputCharting(int[] previousChartings, Scanner sc) {
   // Validate the new charting integer that was input in the console and return the new chartings array containing it.
}
```

Finally, when the user decides to exit the program, print the current value of all the properties (leader, bpm, leader, chartings) before exiting.

Exercise 2 – TicTacToe

Create a basic Tic-Tac-Toe game.

1° Write a program **"TicTacToe3X3.java"** that uses a (two-dimensional) array to represent the 3x3 game field. The type you use for the array is left to your choice but keep in mind you need to represent 3 different game field states: empty, cross & circle.

2° Extend the previous program and add the game logic that runs for 9 rounds. (i.e. until the game field is completely filled up) Player "circle" is starting and players will switch at each round. Each round, read from standard input the X and Y coordinates the player wants to place its mark. After the round is finished, print the updated game board to standard output (use tabs "\t" to align the columns).

3° Now extend your program to check after each round whether a player has won already. Modify the game logic such that the game ends once a player has won instead of completing all 9 rounds. Additionally, announce the winner to standard output.

Exercise 3 – Palindrome

A Palindrome is a word that is read forwards the same as backwards, e.g. noon, radar, level, ...

1° Write a program **"Palindrome.java"** that reads a sequence of chars from standard input, one-by-one, until the user enter 0 (zero, which is not a part of the char sequence!). After reading the sequence, the program verifies and outputs whether the input sequence represents a palindrome or not. (i.e. NoOn is not considered a palindrome!). For simplicity we consider that the user will not enter a sequence bigger than 256 characters.

2° The ASCII value of the characters `A, Z, a, z` are respectively 65, 90, 97, 122. Write a function `toLowerCase(char letter)` that returns the lowercase value of a character if it's a character between A-Z and returns the unmodified letter otherwise. Use this function to make your program not case-sensitive (i.e. NoON would be considered a palindrome!).

Exercise 4 – Vectors and Matrices

In this exercise we consider an N-dimensional euclidean vector space (the vector space you know from high-school). Each of the following programs will **first read the dimension N** of our vector space.

1° Write a program **"Inverse.java"** that reads an N-dimensional vector v from standard input and calculates and outputs the inverse of the vector $(-v)$.

2° Write a program **"Norm.java"** that reads a vector and outputs the euclidean norm of the vector.

3° Write a program **"DotProduct.java"** that reads two vectors and outputs their dot product.

4° Write a program **"MatrixVectorMultiplication.java"** that reads an NxN-dimensional matrix m and an N-dimensional vector v and outputs their multiplication $(m \times v)$.

Exercise 5 – TicTacToe AI

This time we will extend the TicTacToe game from the corresponding exercise in two ways. Create a new program **"TicTacToeNXM.java"** based on the **TicTacToe3X3.java** you wrote earlier.

1° Extend your solution of the previous exercise such that it requests to input the size of the playing field at the start of the program, rather than fixing it to 3×3. Note that the game field does not have to be square. The smaller dimension will define the number of signs in a line needed to win - alternatively you can also request the number of wining signs upon start.

2° Now extend the program such that the second player (cross) is played by an AI instead of a human player. To do that you can use the `nextInt(int bound)` method from the `Random` class provided by Java (see JDK documentation). You can let the AI play randomly or try to make it smarter (e.g. checking if it can finalize a winning line or blocking you from not finalizing yours).