Programming Paradigms
Lab 6

Jacek Wasilewski

Exercises

Case Classes & Pattern Matching

1. With the use of case classes, create a structure `Address` that holds street name, house number, city and country. Pick the most appropriate types.

2. Create an instance of `Address`. Check if you can create an instance using the `new` keyword, without the `new` keyword and with `apply()` method.

3. Can you create an instance of `Address` without specifying all the attributes?

4. Check if you can access member of `Address`. Check if you can modify them as well.

5. Check the default String representation of the instance you created.

6. Check the result of calling `unapply` method on you address instance.

7. Create another case class – `Person`. It should be made of first name, last name and address object.

8. Write a function that takes a string. Use pattern matching to check if the string is "A", "B", "C" or something else. Print different messages for these cases.

9. Write a function that takes an integer. Use pattern matching to check if the integer is odd or even. You can achieve that using pattern matching guards like in the following example:

```
case x if x > 5 => "x is greater than 5"
```

10. Write a function that takes a person. If that person is from Ireland, return "Hiya <name>!", otherwise return "Hello stranger from <country>!"

11. Add additional rules to the function you have just defined. The more complex the better.

12. Using the pattern matching mechanism, write a function that calculates the sum of a given list of numbers.

13. Using only the pattern matching mechanism, write a function that filters a given list to only keep even numbers

14. Write a function that takes a list of integers and using the pattern matching mechanism reorganises the list to first have all odd numbers, then even numbers. Numbers do not have to be sorted.

For-comprehensions

1. Generate a list of 10 elements. Write a for-comprehension to get a list of these elements multiplied by 2.

2. Write a for-comprehension that returns a square root of elements of the input list (use `Math.sqrt`).

3. Modify the previous code and add another generator inside the for-comprehension. Check the results.

4. Generate a list of elements from 50 to 100. Using the for-comprehensions, filter elements to keep only these elements where modulo 5 is 0.

5. Given is the following data structure and list:

```
case class Book(title: String, authors: List[String])

val books: List[Book] = List(
    Book("Structure and Interpretation of Computer Programs",
        List("Abelson, Harold", "Sussman, Gerald J.")),
    Book("Principles of Compiler Design",
        List("Aho, Alfred", "Ullman, Jeffrey")),
    Book("Programming in Modula-2",
```

```
    List("Wirth, Niklaus")),
Book("Introduction to Functional Programming",
    List("Bird, Richard")),
Book("The Java Language Specification",
    List("Gosling, James", "Joy, Bill", "Steele, Guy", "Bracha, Gilad")))
```

Write a for-comprehension that returns titles of books written by Ullman.

6. Write a for-comprehension that prints out to the screen titles of books where the "Program" string occurs.

7. Write a for-comprehension that returns a list of books that have more than 1 author. Use a definition in your solution.