Sample programming tasks (MAS)

version 2020-03-26

Lecturer: Mariusz Trzaska, PhD (mtrzaska@pjwstk.edu.pl, http://www.mtrzaska.com)

These programming tasks are only intended to guide programming material (learned during previous courses) and their solutions will not be assessed as part of the subject MAS (Modeling and Analysis of Information Systems). They can be used during the "Selected constructions of object-oriented programming languages" class. Students entering the MAS course should be able to solve the vast majority of the following tasks.

For each task, prepare a console application, e.g. in Java (GUI is not needed) that implements / illustrates the following requirements. Each solution must contain examples of use, and hence, the right amount of test data. Take care of proper quality / readability / modifiability of the code, including proper element names, division into smaller units (methods, classes, etc.). Also make sure the code is formatted correctly.

1. How do you store many `int` numbers when you know how many there are? How do you do it when they are added and / or removed while the program is running? How will you display them on the console?
2. Create a random string generator for the given text length.
3. Create a Lorem Ipsum text generator.
4. Write a program that remembers the variable number of strings and then removes random ones. Display pre and post operation data on the console.
5. Generate a set of random numbers from the given range. Then find the largest and smallest among them.
6. Remember the data characterizing the right prism. Develop a solution that stores data about the fixed and variable number of prisms. Display information about them on the console.
7. Remember the data characterizing any prism. Develop a solution that stores data about the fixed and variable number of prisms. Display information about them on the console.
8. Prepare a solution to modify the value of the given numbers (eg increasing by 1).
9. Create a program that calculates the number of days from the beginning of the semester and to its end. Use the classes (e.g. `LocalDateTime`) in the `java.time` package added to Java 8.
10. Practice several refactoring techniques available in modern IDEs, e.g. in IntelliJ IDEA, e.g. *Safe Delete, Extract Method, Extract Constant, Extract Field, Extract Parameter, Introduce Variable, Rename.*
11. Give examples of the use of `break` and `continue` constructs.
12. Create a short program using the `switch` structure. Remember to use `break` and `default` properly.
13. Create a short program using the `while` and `do / while` loops. How are they different?

14. We want to store information about different types of devices, each of them has a different set of features. How to do it? How to put them in a common collection and display them on the console?
15. We have completely different entities, but each of them has a common skill, e.g. movement. How to remember it in the program, iterate over them and activate this skill? Do not use a common superclass.
16. Give an example of your own error handling class and using it together with the `throw` construct.
17. Suppose we need to store information, e.g. about engines. They have some common characteristics, and one of them is the *type of engine*. How to do it? What the method will look like, which, depending on the *type of engine*, will do different things. Give two ways to solve this problem and their advantages / disadvantages.
18. Give examples of business requirements that can be met with different types of containers. Create such implementations.
    a. a native table,
    b. a list,
    c. a set,
    d. a map.

    In each case, add a few objects, remove a few that meet certain requirements, search and display the contents of the containers.
19. Why is *generics* used in Java? Create sample programs that use this concept to implement classes and methods.
20. Give an example of using a *for-each* loop.
21. Save random *strings* to the file. Then read them and display them on the console.
22. Save / read from the file sample data describing a person. Try to take up as little space as possible. Consider using the `ZipOutputStream` class from the Java library.
23. Display the list of files in the given directory. Check if there is a file whose name contains the given text.
24. Give an example of using the *try-with-resources* Java language construct.
25. Give an example of using Java diamond <> notation. What are its advantages / disadvantages?
26. Create an example that uses code of method in the interface (requires Java 8 or later).
27. Create an example using lambda expressions in Java.
28. Create a collection of business objects (e.g. Product), then:
    a. choose from them those that meet the sample requirements (e.g. price higher than the amount provided),
    b. count their total value,

c. determine if it is less than the specified threshold.

Try to make the code as concise as possible, but also legible. Consider using function interfaces in the `java.util.function` package (requires Java 8+).

29. Give examples of the use of the *var* keyword in Java (requires Java 10+). How does it relate to strong typological control and similar word in JavaScript?

30. Implement a program comparing performance of different containers (a list, a map, a set) using operations such as adding, inserting, retrieving, removing items.