SWE 4301 - OBJECT ORIENTED CONCEPTS II

OOC-2 LAB 11 HANDOUT AND TASKS

Prepared by:
Md. Jubair Ibna Mostafa
Assistant Professor, IUT CSE

Department of Computer Science and Engineering
Islamic University of Technology

Contents

1 Accessors & Mutators **3**

2 Pure & Impure Function **5**

3 Lambda Expression **8**

4 Tasks and Instructions **8**

1 ACCESSORS & MUTATORS

Accessor and Mutator are two types of methods that deals with fields of a class.

Accessor: An Accessor method is commonly known as a get method or simply a getter. A property of the object is returned by the accessor method.

Example:

```
public class Employee {
    private int ID;
    public int getID() {
        return ID;
    }
}
```

From, the above point of view, we can say a overriden *toString* method is also an accessor or getter method which return all the fields information.

Example:

```
public class Employee {
    private int ID;
    private String name;

    public Employee(int ID, String name){
        this.ID = ID;
        this.name = name;
    }
    public int getID() {
        return ID;
    }
    public String getName() {
        return name;
    }

    @Override
    public String toString() {
        return "Employee{" + "ID='" + ID + '\'' + ", name='" + name + '
```

```
    }';
19   }
20 }
```

If you think about the above class it has read-only property (i.e., the properties or fields' values can only be read, no change is allowed or possible.

Mutator: A Mutator method is commonly known as a set method or simply a setter. A Mutator method mutates things, in other words change things. They are also known as modifiers. Mutator methods may have any return type or not. However, they accept a parameter and change the field of an object.

Example:

```
1  public class Employee {
2      private int ID;
3      private String name;
4
5      public Employee(int ID, String name){
6          this.ID = ID;
7          this.name = name;
8      }
9
10     public void setID(int newID) {
11         this.ID = newID;
12     }
13     public void setName(String name) {
14         this.name = name;
15     }
16 }
```

Here, setID() and setName() methods are mutator or setter or modifier methods that change the state or value of an object.

If you observe the above class it has write-only property (i.e., the properties or fields' values can only be written but access of those properties are not allowed.

Normally, these methods are used to hide the data of the object and to prevent illegal access to these objects either willful or not. Validation logic can also be imposed by these

methods on the values that are being set or returned.

Example 2: Now, recall Liskov Substitution Principle (LSP)'s example, there is a part like below-

```
public class RectangleChanger {
    public void changeHeight(Rectangle rectangle, int newHeight) {
        rectangle.setHeight(newHeight);
    }
}
```

This example is also an mutator method which change the property (i.e., Height of a rectangle) of an object.

2 PURE & IMPURE FUNCTION

Impure Function Characterisitcs of Impure functions:

1. The return value of the impure functions does not solely depend on its arguments Hence, if you call the impure functions with the same set of arguments, you might get the different return values For example, Math.random(), Date.now()
2. They may have any side effects like network or database calls
3. They may modify the arguments which are passed to them

Example 1:

```
public class Employee {
    private int ID;
    private String name;
    private int experience;

    public Employee(String name) {
        this.name = name;
    }

```

```
    public void setID(int newID) {
        this.ID = (int) (Math.random() * newID);
    }
    public void setExperience(Date joining) {
        Calendar calendar = Calendar.getInstance(Locale.US);
        calendar.setTime(joining);
        LocalDateTime now = LocalDateTime.now();
        int years = now.getYear() - calendar.get(Calendar.YEAR);
        this.experience = years;
    }

    @Override
    public String toString() {
        return "Employee{" +
                "ID=" + ID +
                ", name='" + name + '\'' +
                ", experience=" + experience +
                '}';
    }
}
```

In this example, the setID() and setExperience() methods uses Math.random() and LocalDateTime.now() inside the method respectively. If you pass the same argument the methods will not generate same results. That is why above two methods are impure functions.

Example 2:

```
public class EmployeeDB {
    public void save(Employee employee) {
        FileWriter fw = null;
        try {
            fw = new FileWriter("myfile.txt", true);
            fw.write(employee.toString() + "\n");
        } catch (IOException e) {
            e.printStackTrace();
        } finally {
            try {
```

```
                fw.close();
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }

    // this code will give error if there is no file
    public void anotherSave(){
        try {
            Files.write(Paths.get("myfile.txt"), "the text".getBytes(),
    StandardOpenOption.APPEND);
        }catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Secondly, in this example, the save() method stored the argument in a file as a string. Suppose, the file is corrupted or the user do not have write access to the file. Then the above method give an error or different result from the successful one. That's why it has side effects and considered as impure function.

Example 3:

```
public class RectangleChanger {
    public void changeHeight(Rectangle rectangle, int newHeight) {
        rectangle.setHeight(newHeight);
    }
}
```

Finally, in this example you can see the rectangle object passed as an argument in the method is changed. As the method change the state of its argument, it is an impure function as well.

Pure Function Characteristics of Pure Function:

1. The return value of the pure functions solely depends on its arguments Hence, if you call the pure functions with the same set of arguments, you will always get the same

return values.

2. They do not have any side effects like network or database calls
3. They do not modify the arguments which are passed to them

Advantage of Pure Functions:

- Pure functions do not affect any external state, and they are also not affected by external codes.
- Pure functions are easier to read and debug than their impure alternatives.

3 LAMBDA EXPRESSION

Example: Normally, we use the below code get employees with high salaries.

```
public List<Employee> employeeWithHighSalaries(double salary) {
    List<Employee> res = new ArrayList<>();
    for(Employee e: employees) {
        if(e.getSalary() > salary) res.add(e);
    }
    return res;
}
```

Below code represents the previous code using lambda expression.

```
public List<Employee> employeeWithHighSalaries(double salary) {
    return employees.stream()
        .filter(e -> e.getSalary() > salary)
        .collect(Collectors.toList());
    }
```

4 TASKS AND INSTRUCTIONS

1. Create another method in the EmployeeDB class that will return the List of employees previously saved in the file.

2. Create a method addEmployee which will add an employee to a list of employees. This employee list is an attrribute of EmployeeDB. Specify whether it is a pure or impure function.

3. consider below method in EmployeeDB class and mention why is it a pure function or not.

```
1    public List<Employee> add(List<Employee> employees, Employee
   employee){
2        employees.add(employee);
3        return employees;
4    }
5
```

4. add age and salary field to employee and corresponding accessor and mutator methods.

5. In EmployeeDB class write one method ageFilter that takes an argument of age and return a list of employees who have the same age as passed argument. Use lambda expression.

6. In EmployeeDB class write one method salaryFilter that takes an argument of minimum salary and return a list of employees who have more salary than the passed argument. Use lambda expression.

7. In EmployeeDB class write one method incrementMapper that takes an argument of increment (e.g., 5%) and print the list of values. Use lambda expression.

8. In EmployeeDB class write one method totalSalaryofFilteredEmployee that takes an argument of minimum salary and total salary of filtered employees. Use lambda expression.