EXERCISES - PART 1

1. We have a class Company that stores two types of employees: HourlyEmployee and MonthlyEmployee. Out of the next two versions of class Company which one would you like to use? Explain your decision.

```
class CompanyA {
 private ArrayList<HourlyEmployee> hourlyEmployees;
 private ArrayList<MonthlyEmployee> monthlyEmployees;

 public void add(Employee e) { . . .
 public String toString() { // return all the stored employees as a String
}

class CompanyB {
 private ArrayList<Employee> employees;

 public void add(Employee e) { . . .
 public String toString() { // return all the stored employees as a String
}
```

2. a) Which rule/principle/pattern is violated in the next fragment of code? Refactor to get rid of the detected problem.

```
class BuildingTaxCalculator {
	private BuildingTaxTable table;
	public BuildingTaxCalculator(BuildingTaxTable table) { this.table = table; }

	public int compute(Person person, Building building) {
		return building.getSquareFeetNumber() *
			table.getTax(person.getAddress());
}}
```

b) What is wrong in the next source code?

```
class Engine { . . . }
class Car {
	private Engine e;
	public Car () {
		this.e = new Engine();
}}
```

3. Name the design principle that is violated in the next source code. Modify the code in order to remove the problem.

```
interface Toy {
	// postcondition - returns a String that is not null
	public String play();
}

class Car implements Toy {
	public String play() {
		return "Playing with the car";
}}
```

```
class ElectricCar implements Toy {
	private boolean isTurnedOn = false;

	public void turnOn() { isTurnedOn = true; }

	public String play() {
		if(isTurnedOn)
			return "Playing with an electric car";
		return null;
}}

class ToyTest{
	public static void main(String[] args){
		Toy t = new Car();
		t.play();
		Toy o = new ElectricCar();
		o.play();
}}
```

4. Given the next class

```
class Employee {
	private String firstName, lastName;
	public Employee(String firstName, lastName) {
	// . . .
	}}
```

Instantiate an object that contains Employee objects and print the kept employees. Inside the instantiated object we can store more than once the same employee. If needed, add new services inside the given class.

5. We have a class Company that stores two types of employees: HourlyEmployee and MonthlyEmployee. Out of the next two versions of class Company which one would you like to maintain? Explain your decision.

```
class CompanyA {
 private ArrayList<HourlyEmployee> hourlyEmployees;
 private ArrayList<MonthlyEmployee> monthlyEmployees;

 public void add(Employee e) { . . .
 public String toString() { // return all the stored employees as a String
}

class CompanyB {
 private ArrayList<Employee> hourlyEmployees;

 public void add(Employee e) { . . .
 public String toString() { // return all the stored employees as a String
}
```

6. a) Is Demeter's law violated in the next fragment of code? Explain the reasons behind your answer.

```
class Colada {
	private Blender myBlender;
	private ArrayList<Ingredient> ingredients;
```

```
	public Colada(ArrayList<Ingredient> ingredients) {
		myBlender = new Blender();
		this.ingredients = ingredients;
	}

	public void mix() {
		myBlender.addIngredients(ingredients);
	}}
```

b) It seems the programmer who wrote the code above does not know an important SOLID principle – name the principle the programmer forgot to comply with and refactor the code in order to remove the existing negative consequences.

7. Does the Service interface comply/not comply with the Interface Segregation Principle?

```
interface Service {
	public void firstService();
	public int computeService();
	public int multiplyService(); }
```

8. Given the next class

```
class Employee {
	private String firstName, lastName;
	public Employee(String firstName, lastName) { // . . . }}
```

Instantiate an object that contains Employee objects and print the kept employees. Define the needed services that guarantee an existing person is stored only once. If needed, add new services inside the given class.

9. Based on the next Vehicle interface

```
interface Vehicle {
	public void enableHeatInChair();
	public void disableHeatInChair();
	public void startEngine();
	public void accelerate();
```

}, we intend to define various types of vehicles and some of them provide all the presented services and some of them do not (e.g. implement the service that does not exist with a No Operation – an empty method). Some clients of this interface use all the provided services and some do not. Identify at least two problems related to this implementation, name the principles this approach does not comply with and show the necessarily changes that have to be performed in order to get rid of the problems.

10. a) Does the following class violate the Single Responsibility Principle? If so, refactor it!

```
class Student {
	. . .
	public void saveStudent() { . . . //save into a file
	public Double computeGrade() { . . .
	public String getStudentProfile() { . . . //return firstName + lastName +
grade
}
```

b) Does the Student class comply/not comply with the Interface Segregation Principle?

11. We have a class University that stores two types of persons: Teachers and Students. Out of the next two versions of class University which one would you like to use? Explain your decision.

```
class UniversityA {
 private Collection<Teacher> teachers;
 private Collection<Student> students;

 public void add(Teacher t) { . . .
 public void add(Student s) { . . .

 public String toString() { // return the stored teachers + students as a String
}
class UniversityB {
 private Collection<Person> persons; //both Teacher and Student extend Person

 public void add(Person e) { . . .
 public String toString() { // return the stored teachers + students as a String
}
```

12. Based on the next Vehicle interface

```
interface Vehicle {
      public void startEngine();
      public void accelerate();
      public void stopRadio();
      public void ejectCD();
```

}, we intend to define various types of vehicles and some of them provide all the presented services and some of them do not (e.g. implement the service that does not exist with a No Operation – an empty method). Some clients of this interface use all the provided services and some do not. Identify at least two problems related to this implementation, name the principles this approach does not comply with and show the necessarily changes that have to be performed in order to get rid of the problems.

13. Comment the next statements:

a) The Interface Segregation Principle refers to the fact that each class should implement a small number of interfaces, usually not more than four.

b) We should always create classes that do not have accessor methods (getters and setters).

c) If a class violates the Single Responsibility Principle it should provide more than two services.

d) Following Demeter‘s Law means accessing data from a restricted set of objects.

14. a)What is a God Class?

b) Which design principle you think is always violated by God Classes? Why?

c) How do you think you can capture the God Classes inside a system by looking only at its class diagram?

15. Considering a Base class, a Derived class that is derived from Base and the next methods, you have to specify whether or not the Liskov substitution principle is violated. Justify your answer by relating the pre and post conditions of the Derived class to the ones from the Base class.

Base: int service(int number) //PRECOND: x > 50 //POSTCOND: returns 1 OR 2
Derived: int service(int number) //PRECOND: x > 30 //POSTCOND: returns 1 OR 2 OR 3

16. Define the Interface Segregation Principle.

17. Comment the next statements:
a) We can detect the classes that violates The Interface Segregation Principle by looking at the interfaces that are implemented by each class.
b) If we add getters to a class then using the returned objects means always violating Demeter's Law.
c) We should always create classes that do not have accessor methods (getters and setters).

18. a) What is a hook method?
b) Define Fragility and Immobility.
c) Define Rigidity and Viscosity.