Marmara University – Faculty of Engineering – Department of Computer Engineering

Fall 2021 – CSE1141 Computer Programming I
Homework #5

Due: 16.01.2022.Sun 23.59

In this homework, you will implement a factory simulator program with object-oriented approach. In the UML diagrams below, access (getter) and modifier (setter) methods are not shown for simplicity. However, you are supposed to implement these methods for each class as well.

1) Implement an *Item* class using the following UML diagram.

	Item
-	id: int
+	<u>numberOfItems</u>: int
+	Item(id: int)

- An *Item* object represents an item generated by the employees working for the factory.
- Data field *id* keeps the id of the Item object created.
- Static data field *numberOfItems* keeps the number of item objects created.
- An item is created with an *id* data field.
  - You have to use *this* keyword in the implementation of the constructor.
  - In the constructor, you also need to increment *numberOfItems* data field.
  - Consider the following example which demonstrates the creation of *Item* object.

```
Item item = new Item(10);
```

2) Implement a *Payroll* class using the following UML diagram.

	Payroll
-	workHour: int
-	itemCount: int
+	Payroll(workHour: int, itemCount: int)
+	calculateSalary(): int
+	toString(): String

- A *Payroll* object represents the payroll of the employees.
- Data field *workHour* keeps the number of hours an employee has worked.
- Data field *itemCount* keeps the number of items an employee has produced.
- A *Payroll* object is created with a given *workHour* and *itemCount*.
  - You have to use *this* keyword in the implementation of the constructor.
  - Consider the following example which demonstrates the creation of *Payroll* object.

```
Payroll payroll = new Payroll(8,15);
```

  - Here, the payroll will be created for the 8 hours of working and 15 produced items.

- *calculateSalary()* method calculates salary of the employee according to the number of hours an employee has worked and the number of items s/he produced. For each hour an employee works, s/he earns 3 liras and for each item s/he produces s/he additionally earns 2 liras.
  - As an example, an employee will earn 8*3=24 liras for the working hours and 15*2=30 liras for the items produced with the sample *Payroll* above; hence a total of 54 liras should be calculated.
- *toString()* method returns a String containing information about the work hour and the item count of the payroll. An example string is as following:
  - *The work hour is 8 and the produced item count is 15.*

3) Implement an *Employee* class using the following UML diagram.

	Employee
-	id: int
-	name: String
-	surname: String
-	workHour: int
-	speed: int
-	items: Item[]
-	payroll: Payroll
+	Employee(id: int, name: String, surname: String, workHour: int, speed: int)
+	startShift(): Item[]
+	endShift(): Payroll
+	toString(): String

- An *Employee* object represents an employee working for the factory.
- Data field *id* keeps the id number of the *Employee* object created.
- Data fields *name* and *surname* keeps the name and the surname of the employee, respectively.
- Data field *workHour* keeps the number of hours an employee will work.
- Data field *speed* keeps the number of items that the employee can produce in an hour.
- Data field *items* array holds the items produced by the employee.
- Data field *payroll* keeps the payroll of the employee.
- An employee is created with an *id*, *name*, *surname*, *workhour*, and *speed* values.
  - You have to use *this* keyword in the implementation of the constructor.
  - Consider the following example which demonstrates the creation of *Employee* object.

```
Employee employee = new Employee(1, "Ahmet", "Yilmaz", 8, 3);
```

  - Here, an employee has an id of 1, name of "Ahmet", surname of "Yilmaz", working hour of 8, and speed of 3. Here, speed represents that the employee can produce 3 items in an hour.

- *startShift()* method finds how many items this employee should produce according to *speed* and *workHour* values. After that, it creates appropriate number of items with randomly generated ids between 1-100 and put them into *items* array. It then returns *items* array.
- *endShift()* method creates a *Payroll* object with employee's work hour and the number of items s/he creates. It assigns this object to payroll data field. It then returns *payroll* object.
- *toString()* method returns a String with employee's id and the return value of the payroll object's *toString()* method. An example string is as following:
  - *This is the employee with id 1 speed 3. The work hour is 8 and the produced item count is 24.*

4) Implement a *Storage* class using the following UML diagram.

	Storage
-	capacity: int
-	Items: Item[]
+	Storage(capacity: int)
+	addItem(item: Item): void

- A *Storage* object represents a storage area for the factory.
- Data field id *capacity* keeps the capacity (the maximum number of items that can be stored) of the *Storage* object created.
- Data fields *items* array keeps the items put inside the storage.
- A storage is created with a given capacity value.
  - You have to use *this* keyword in the implementation of the constructor.
  - Consider the following example which demonstrates the creation of *Storage* object.

```
Storage storage = new Storage(100);
```

- *addItem(item:Item)* method adds the item passed as the parameter to the items data field. Note that you need to resize the items array for this.

5) Implement a *Factory* class using the following UML diagram.

	Factory
-	name: String
-	employees: Employee[]
-	storage: Storage
-	payrolls: Payroll[]
-	itemPrice: double
+	Factory(name: String, capacity: int, itemPrice: double)
+	getRevenue(): double
+	getPaidSalaries(): double
+	addEmployee(employee: Employee): void
+	removeEmployee(id: int): Employee
-	addPayroll(payroll: Payroll): void

- A *Factory* object represents a factory with employees.
- Data field *name* represents the name of the factory.
- Data field *employees* represents employees working for the factory.
- Data field *storage* represents the storage area of the factory.
- Data field *payrolls* represents the payrolls belonging to the employees.
- Data field *itemPrice* represents the price for one item (Suppose that all items have the same price).
- A factory is created with a given *name*, *capacity*, and *itemPrice* value.
  - You have to use *this* keyword in the implementation of the constructor.
  - In the constructor, you also need to create a *Storage* object with the given *capacity* and assign it to *storage* data field.
  - Consider the following example which demonstrates the creation of *Factory* object.

    ```
    Factory factory = new Factory("My Factory", 100, 5);
    ```
- *getRevenue()* method returns the revenue according to the number of items in the *storage* data field and *itemPrice* data field.
- *getPaidSalaries()* method calculates the paid salaries of the employees according to the *payrolls* array. Note that you need to use *calculateSalary()* method of payroll objects in *payrolls* array.
- *addEmployee(employee:Employee)* method adds an employee to the *employees* array.
  - Note that you need to resize the *employees* array for this.
  - Then, you need to call *startShift()* method of the newly added employee and add all the items returned from *startShift()* method to *storage*, using *addItem(item:Item)* method of *storage* data field.
- *removeEmployee(id:int)* method removes the employee from *employees* array.
  - If there are no employees, it prints an appropriate error message.
  - If the employee with a given id is not found, it prints an appropriate error message.
  - If the employee is found, it removes employee from the *employees* array.
    - Note that you need to resize the *employees* array for this.
  - Then, you need to call *endShift()* method of the newly removed employee and call *addPayroll(payroll:Payroll)* method with the returned payroll by the *endShift()* method.
  - At the end, you need to return the removed employee.
- *addPayroll(payroll:Payroll)* method adds the payroll passed as the parameter to the *payrolls* data field. Note that you need to resize the *payrolls* array for this.

You are given a simple scenario (Test.java) to test your class implementations. Note that the output of the example test class is as the following:

```
There are no employees!

Employee does not exist!

This is the employee with id 1 speed 3. The work hour is 8 and the produced
item count is 24.

This is the employee with id 2 speed 2. The work hour is 6 and the produced
item count is 12.

This is the employee with id 3 speed 1. The work hour is 10 and the produced
item count is 10.

Revenue: 230.0

Paid salaries: 164.0

Total item produced: 46
```

IMPORTANT NOTES

1) There might be other test cases too. Therefore, please pay attention to use the same class, method and variable names in your implementations. You are allowed to increase the number of methods in the classes; however, you cannot decrease the number of them. You are not allowed to use ArrayLists in the homework! You can only use Arrays.

2) Write a comment at the beginning of each program to explain the purpose of the program. Write your name and student ID as a comment. Include necessary comments to explain your actions.

3) You are allowed to use the materials that you have learned in lectures and labs. Do not use the ones that you have not learned in the course.

4) Please be sure that your programs run properly on any computer.

5) Your program will be tested with an auto-grader. So, it should take the inputs exactly the same in the sample runs and it should print the outputs exactly the same in the sample runs. Otherwise, your program may fail.

6) Your program should execute correctly for different test cases.

7) Please do not write any package name at the beginning of your code.

8) Please zip all your files into a single zip file using file naming convention StudentID_HW5.zip, e.g., 150120123_HW5.zip. Your zip file should contain the followings:

a) 5 Java source files: Item.java, Payroll.java, Employee.java, Storage.java, Factory.java.

b) 5 Java class files: Item.class, Payroll.class, Employee.class, Storage.class, Factory.class.

9) Submit your zip file to http://ues.marmara.edu.tr before deadline.

10) You are responsible for making sure you are turning in the right file, and that it is not corrupted in anyway. We will not allow resubmissions if you turn in the wrong file, even if you can prove that you have not modified the file after the deadline.

11) Each student should submit his/her own homework. You can discuss with your peers about the homework, but you are not allowed to exchange codes or pseudocodes. This also applies to material found on the web. If some submitted homework assignments are found to be identical or suspected to be identical, all involved parties will get a grade of ZERO from all homework. You should submit your own work. In case of any forms of cheating or copying, both giver and receiver are equally culpable and suffer equal penalties. All types of plagiarism will result in FF grade from the course.

12) No late submission will be accepted.