Assignment 3: Design pattern I

//Bài tập CÁ NHÂN

Note: Refer to the text book for Exercise 1,2

Exercise 1: Using interface for designing a program for calculating salaries of employees of various categories

Exercise 2: Using abstract or interface for designing Employee class with methods

- Save employee data
- Display employee data
- Access employee attributes such as name and ID
- Calculate compensation

Exercise 3: Factory method

1. Running the code given at the end of this file
2. Adjusting code to add 2 services (TechcomBank, ViettinBank, ANZ…..). //adding interfaces and adjusting factory
3. Running the adjusted code

Exercise 4: Refer to Internet to **(give link)**

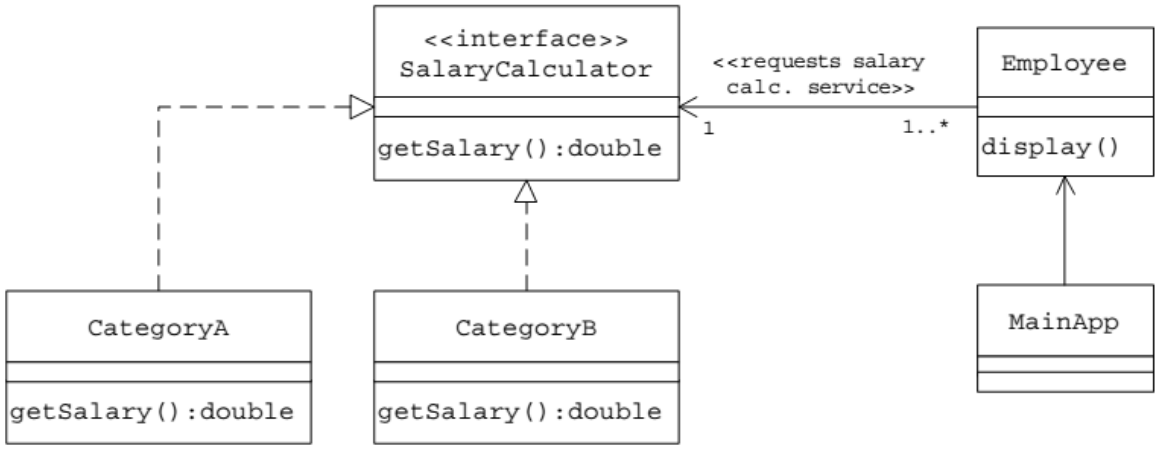
1. Draw abstract factory diagram and code an application
2. Draw builder diagram and code an application

Exercise 5: Applying patterns (factory, abstract factory, builder) for your Project

1. Team discussion: which pattern should be applied for what services?
2. Draw diagram and code (each student/one application)

===Exercise 1==========

Designations	Category
Programmer, Designer and Consultant	Category-A
Sales Rep, Sales Manager, Account Rep	Category-B
...	...
C-Level Executives	Category-n
...	...



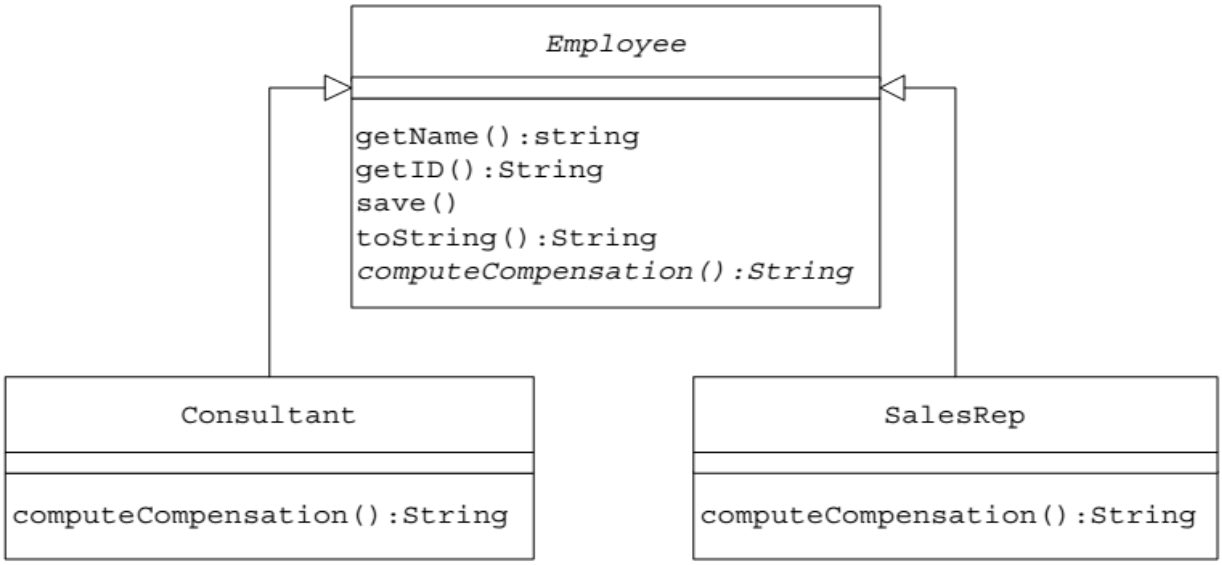
Code

```
public interface SalaryCalculator {
public double getSalary();
}

public class CategoryA implements SalaryCalculator {
double baseSalary;
double OT;

public CategoryA(double base, double overTime) {
baseSalary = base;
OT = overTime;
}
public double getSalary() {
return (baseSalary + OT);
}
}
public class CategoryB implements SalaryCalculator {
```

```
double salesAmt;
double baseSalary;
final static double commission = 0.02;
public CategoryB(double sa, double base) {
baseSalary = base;
salesAmt = sa;
}
public double getSalary() {
return (baseSalary + (commission * salesAmt));
}
}

public class Employee {
SalaryCalculator empType;
String name;

public Employee(String s, SalaryCalculator c) {
name = s;
empType = c;
}
public void display() {
System.out.println("Name=" + name);
System.out.println("salary= " + empType.getSalary());
}
}

public class MainApp {
public static void main(String [] args) {
SalaryCalculator c = new CategoryA(10000, 200);
Employee e = new Employee ("Jennifer”,c);
e.display();
c = new CategoryB(20000, 800);
e = new Employee ("Shania”,c);
e.display();
}
}
```

===Exercise 2==========================


<<interface>>
Employee
getName():string
getID():String
save()
toString():String
computeCompensation():String
Consultant
getName():string
getID():String
save()
toString():String
computeCompensation():String
SalesRep
getName():string
getID():String
save()
toString():String
computeCompensation():String


Employee
getName():string
getID():String
save()
toString():String
computeCompensation():String
Consultant
computeCompensation():String
SalesRep
computeCompensation():String

=======Exercise 3==========

Supper Class:

```
public interface Bank {
    String getBankName();
}
```

Sub Classes:

```
package com.gpcoder.patterns.creational.factorymethod;

public class TPBank implements Bank {

    @Override
    public String getBankName() {
        return "TPBank";
    }

}
```

```
package com.gpcoder.patterns.creational.factorymethod;

public class VietcomBank implements Bank {

    @Override
    public String getBankName() {
        return "VietcomBank";
    }

}
```

Factory class:

```
public class BankFactory {

    private BankFactory() {
    }

    public static final Bank getBank(BankType bankType) {
        switch (bankType) {

        case TPBANK:
            return new TPBank();

        case VIETCOMBANK:
            return new VietcomBank();

        default:
```

```
                throw new IllegalArgumentException("This bank type is unsupported");
            }
        }

    }
```

Bank type:

```
public enum BankType {

    VIETCOMBANK, TPBANK;

}
```

Client:

```
public class Client {

    public static void main(String[] args) {
        Bank bank = BankFactory.getBank(BankType.TPBANK);
        System.out.println(bank.getBankName()); // TPBank
    }
}
```