SWE3005

Principles of Design Patterns

Submitted by

ANKAM SRINIVAS

(21MIS7009)

To

DR. B.V.GOKULNATH

VIT-AP UNIVERSITY

VIT-AP UNIVERSITY , AMARAVATHI

ANDHRA PRADESH

LAB – 05

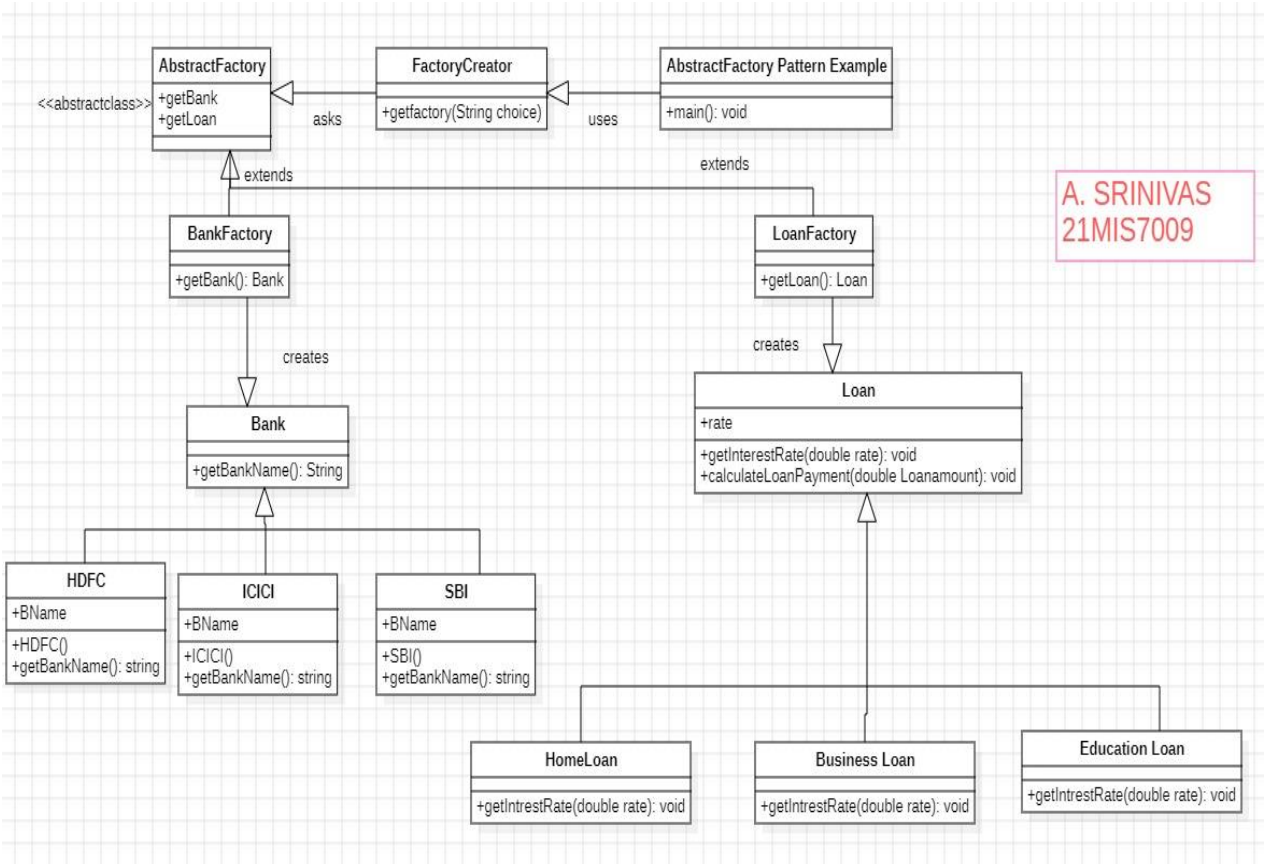
Question :

1. Calculate the loan payment for different bank using Abstract Factory design pattern.
   Document should include aim, uml diagram, code and input output.

Aim:

Abstract Factory Pattern says that just define an interface or abstract class for creating families of related (or dependent) objects but without specifying their concrete sub-classes.That means Abstract Factory lets a class returns a factory of classes. So, this is the reason that Abstract Factory Pattern is one level higher than the Factory Pattern.An Abstract Factory Pattern is also known as Kit. Now , Calculate the loan payment for different bank using Abstract Factory design pattern.

UML diagram:



Code:

```
import java.io.*;

interface Bank {
    String getBankName();
}

class HDFC implements Bank {
    private final String BNAME;

    public HDFC() {
        BNAME = "HDFC BANK";
    }

    public String getBankName() {
        return BNAME;
    }
}

class ICICI implements Bank {
    private final String BNAME;

    public ICICI() {
        BNAME = "ICICI BANK";
    }

    public String getBankName() {
        return BNAME;
    }
}

class SBI implements Bank {
    private final String BNAME;

    public SBI() {
        BNAME = "SBI BANK";
    }

    public String getBankName() {
        return BNAME;
    }
}

abstract class Loan {
    protected double rate;

    abstract void getInterestRate(double rate);
```

```
    public void calculateLoanPayment(double loanamount, int years) {
        double EMI;
        int n;

        n = years * 12;
        rate = rate / 1200;

        EMI = ((rate * Math.pow((1 + rate), n)) / ((Math.pow((1 + rate), n)) -
1)) * loanamount;

        System.out.println("your's monthly EMI is " + EMI + " for the amount "
+ loanamount + " you have borrowed ");
    }
}

class HomeLoan extends Loan {
    public void getInterestRate(double r) {
        rate = r;
    }
}

class BussinessLoan extends Loan {
    public void getInterestRate(double r) {
        rate = r;
    }
}

class EducationLoan extends Loan {
    public void getInterestRate(double r) {
        rate = r;
    }
}

abstract class AbstractFactory {
    public abstract Bank getBank(String bank);

    public abstract Loan getLoan(String loan);
}

class BankFactory extends AbstractFactory {
    public Bank getBank(String bank) {
        if (bank == null) {
            return null;
        }

        if (bank.equalsIgnoreCase("HDFC")) {
            return new HDFC();
```

```
        } else if (bank.equalsIgnoreCase("ICICI")) {
            return new ICICI();
        } else if (bank.equalsIgnoreCase("SBI")) {
            return new SBI();
        }
        return null;
    }

    public Loan getLoan(String loan) {
        return null;
    }
}

class LoanFactory extends AbstractFactory {
    public Bank getBank(String bank) {
        return null;
    }

    public Loan getLoan(String loan) {
        if (loan == null) {
            return null;
        }

        if (loan.equalsIgnoreCase("Home")) {
            return new HomeLoan();
        } else if (loan.equalsIgnoreCase("Business")) {
            return new BussinessLoan();
        } else if (loan.equalsIgnoreCase("Education")) {
            return new EducationLoan();
        }
        return null;
    }
}

class FactoryCreator {
    public static AbstractFactory getFactory(String choice) {
        if (choice.equalsIgnoreCase("Bank")) {
            return new BankFactory();
        } else if (choice.equalsIgnoreCase("Loan")) {
            return new LoanFactory();
        }
        return null;
    }
}

class AbstractFactoryPatternExample
{
    public static void main(String args[]) throws IOException
```

```
    {
        System.out.println("Name: Ankam Srinivas");
        System.out.println("21MIS7009");

        BufferedReader br=new BufferedReader(new
InputStreamReader(System.in));

        System.out.print("Enter the name of Bank from where you want to take
loan amount: ");
        String bankName=br.readLine();

        System.out.print("\n");
        System.out.print("Enter the type of loan you want to take, like home
loan or business loan or education loan: ");
        String loanName=br.readLine();

        AbstractFactory bankFactory = FactoryCreator.getFactory("Bank");

        Bank b=bankFactory.getBank(bankName);

        System.out.print("\n");
        System.out.print("Enter the interest rate for "+b.getBankName()+ ":
");
        double rate=Double.parseDouble(br.readLine());

        System.out.print("\n");
        System.out.print("Enter the loan amount you want to take: ");
        double loanAmount=Double.parseDouble(br.readLine());

        System.out.print("\n");
        System.out.print("Enter the number of years to pay your entire loan
amount: ");
        int years=Integer.parseInt(br.readLine());

        System.out.print("\n");
        System.out.println("You are taking the loan from "+ b.getBankName());

        AbstractFactory loanFactory = FactoryCreator.getFactory("Loan");

        Loan l=loanFactory.getLoan(loanName);

        l.getInterestRate(rate);
        l.calculateLoanPayment(loanAmount,years);
    }
}
```

OUTPUT:

1. HDFC Bank

```
Enter the name of Bank from where you want to take loan amount: hdfc

Enter the type of loan you want to take, like home loan or bussiness loan or education loan : education

Enter the interest rate for HDFC BANK: 12.9

Enter the loan amount you want to take: 5000000

Enter the number of years to pay your entire loan amount: 12

you are taking the loan from HDFC BANK
your's monthly EMI is 68422.27719284763 for the amount 5000000.0 you have borrowed
PS C:\Users\ankam>
```

2. SBI Bank

```
Enter the name of Bank from where you want to take loan amount: sbi

Enter the type of loan you want to take, like home loan or bussiness loan or education loan : business

Enter the interest rate for SBI BANK: 10

Enter the loan amount you want to take: 600000

Enter the number of years to pay your entire loan amount: 14

you are taking the loan from SBI BANK
your's monthly EMI is 6649.216124211667 for the amount 600000.0 you have borrowed
PS C:\Users\ankam>
```

3. ICICI Bank

```
Enter the name of Bank from where you want to take loan amount: icici

Enter the type of loan you want to take, like home loan or bussiness loan or education loan : business

Enter the interest rate for ICICI BANK: 12.9

Enter the loan amount you want to take: 5000000

Enter the number of years to pay your entire loan amount: 12

you are taking the loan from ICICI BANK
your's monthly EMI is 68422.27719284763 for the amount 5000000.0 you have borrowed
PS C:\Users\ankam>
```

2. Significance of Abstract Factory design pattern

The Abstract Factory design pattern is significant in software engineering for several reasons:

1. Abstraction: It provides an abstraction layer for creating families of related or dependent objects without specifying their concrete classes. This promotes loose coupling between the client code and the concrete implementations, making the system more flexible and easier to maintain.

2. Encapsulation: It encapsulates the object creation code within the factory, allowing clients to use the factory interface to create objects without needing to know the details of their instantiation. This enhances encapsulation and information hiding principles, contributing to better modularization and code organization.

3. Flexibility and Extensibility: Abstract Factory allows you to swap entire families of related objects at runtime by switching the concrete factory implementation. This makes the system highly flexible and extensible, facilitating changes or additions to the product line with minimal impact on existing code.

4. Consistency: It ensures that objects created by a factory are compatible and consistent with each other. This is particularly useful when dealing with complex systems composed of multiple interrelated objects or when enforcing specific constraints among objects.

5. Testing and Dependency Injection: Abstract Factory facilitates unit testing and dependency injection by enabling the substitution of concrete factory implementations with mock or stub factories. This helps in isolating components for testing purposes and promoting test-driven development practices.

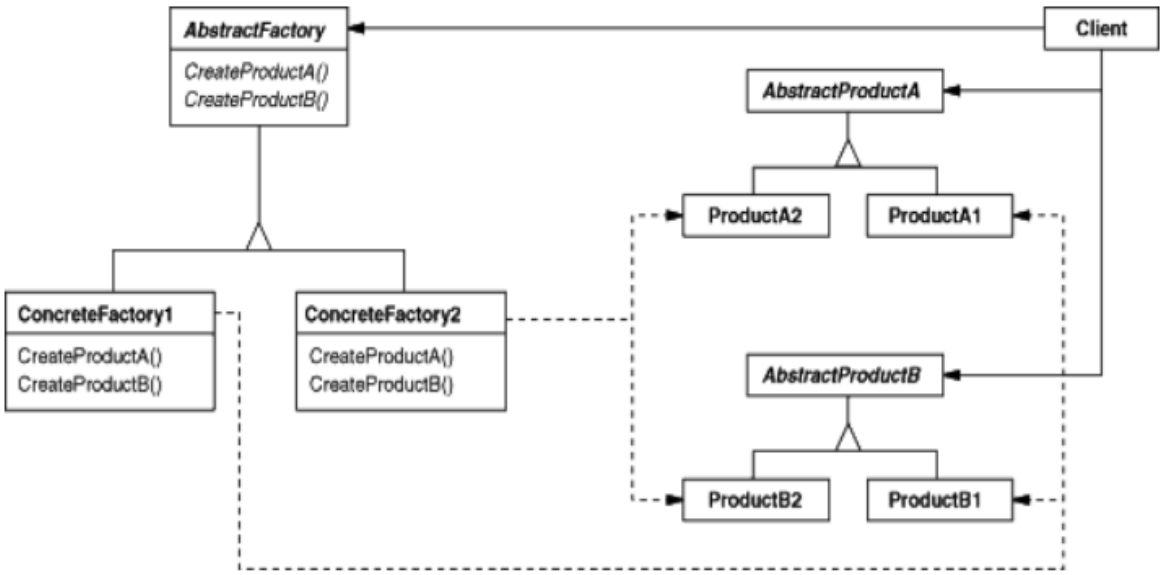
6. Separation of Concerns: It promotes separation of concerns by separating the responsibility of object creation from the client code. This leads to better code organization and improves maintainability by adhering to the Single Responsibility Principle (SRP).

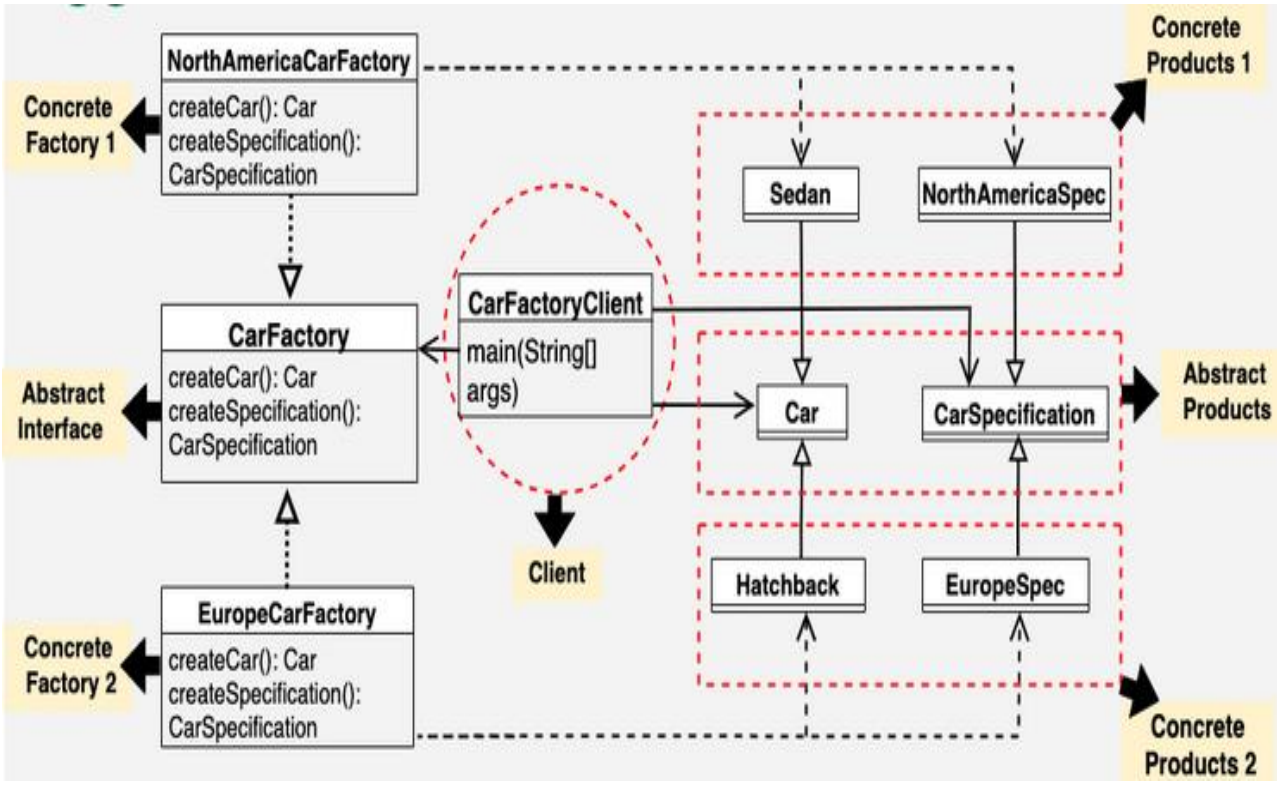
7. Design for Interface, not Implementation: Abstract Factory encourages designing against interfaces rather than concrete implementations, which aligns with the principle of programming to interfaces rather than implementations. This makes the code more adaptable to changes in requirements or technology choices.

Participants:

- ✓ **AbstractFactory** (WidgetFactory)
  - declares an interface for operations that create abstract product objects.
- ✓ **ConcreteFactory** (MotifWidgetFactory, PMWidgetFactory)
  - implements the operations to create concrete product objects.
- ✓ **AbstractProduct** (Window, ScrollBar)
  - declares an interface for a type of product object.
- ✓ **ConcreteProduct** (MotifWindow, MotifScrollBar)
  - defines a product object to be created by the corresponding concrete factory.
  - implements the AbstractProduct interface.
- ✓ **Client:**
  - uses only interfaces declared by AbstractFactory and AbstractProduct classes.

Structure :


AbstractFactory
CreateProductA()
CreateProductB()
Client
AbstractProductA
ProductA2
ProductA1
ConcreteFactory1
CreateProductA()
CreateProductB()
ConcreteFactory2
CreateProductA()
CreateProductB()
AbstractProductB
ProductB2
ProductB1


NorthAmericaCarFactory
createCar(): Car
createSpecification(): CarSpecification
Concrete Factory 1
Concrete Products 1
Sedan
NorthAmericaSpec
CarFactoryClient
main(String[] args)
CarFactory
createCar(): Car
createSpecification(): CarSpecification
Abstract Interface
Car
CarSpecification
Abstract Products
Client
Hatchback
EuropeSpec
EuropeCarFactory
createCar(): Car
createSpecification(): CarSpecification
Concrete Factory 2
Concrete Products 2


Thank You