

Abstract Factory Pattern

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**.

Advantage of Abstract Factory Pattern

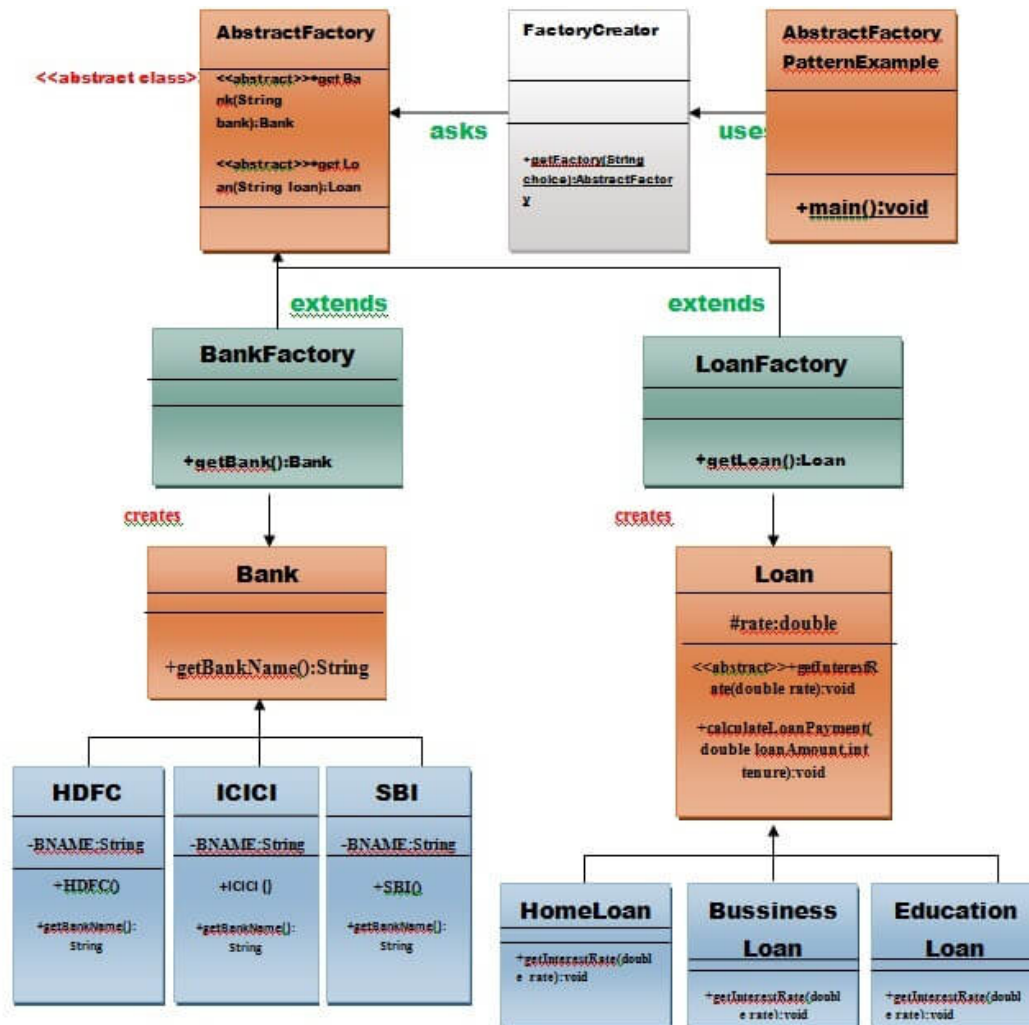
- Abstract Factory Pattern isolates the client code from concrete (implementation) classes.
- It eases the exchanging of object families.
- It promotes consistency among objects.

Usage of Abstract Factory Pattern

- When the system needs to be independent of how its object are created, composed, and represented.
- When the family of related objects has to be used together, then this constraint needs to be enforced.
- When you want to provide a library of objects that does not show implementations and only reveals interfaces.
- When the system needs to be configured with one of a multiple family of objects.

UML for Abstract Factory Pattern

- We are going to create a **Bank interface** and a **Loan abstract class** as well as their sub-classes.
- Then we will create **AbstractFactory** class as next step.
- Then after we will create concrete classes, **BankFactory**, and **LoanFactory** that will extends **AbstractFactory class**
- After that, **AbstractFactoryPatternExample** class uses the **FactoryCreator** to get an object of **AbstractFactory** class.
- See the diagram carefully which is given below:



Example of Abstract Factory Pattern

Here, we are calculating the loan payment for different banks like HDFC, ICICI, SBI etc.

Step 1: Create a Bank interface

```
import java.io.*;

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

Step 2: Create concrete classes that implement the Bank interface.

```
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;
    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;
    }
}
```

```

    }
}

```

Step 3: Create the Loan abstract class.

```

abstract class Loan{
    protected double rate;
    abstract void getInterestRate(double rate);
    public void calculateLoanPayment(double loanamount, int years)
    {
        /*
            to calculate the monthly loan payment i.e. EMI

            rate=annual interest rate/12*100;
            n=number of monthly installments;
            1year=12 months.
            so, n=years*12;

        */

        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 monthly EMI is "+ EMI +" for the amount"+loanamount+" you have borrowed");
    }
} // end of the Loan abstract class.

```

Step 4: Create concrete classes that extend the Loan abstract class..

```

class HomeLoan extends Loan{
    public void getInterestRate(double r){
        rate=r;
    }
} //End of the HomeLoan class.

```

```

class BussinessLoan extends Loan{
    public void getInterestRate(double r){
        rate=r;
    }
} //End of the BusssinessLoan class.

```

```

class EducationLoan extends Loan{
    public void getInterestRate(double r){

```

```
        rate=r;
    }
} //End of the EducationLoan class.
```

Step 5: Create an abstract class (i.e AbstractFactory) to get the factories for Bank and Loan Objects.

```
abstract class AbstractFactory{
    public abstract Bank getBank(String bank);
    public abstract Loan getLoan(String loan);
}
```

Step 6: Create the factory classes that inherit AbstractFactory class to generate the object of concrete class based on given information.

```
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;
    }
} //End of the BankFactory class.
```

```
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;
}

}

```

Step 7: Create a FactoryCreator class to get the factories by passing an information such as Bank or Loan.

```

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;
    }
}
} //End of the FactoryCreator.

```

Step 8: Use the FactoryCreator to get AbstractFactory in order to get factories of concrete classes by passing an information such as type.

```

import java.io.*;

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

        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 e.g. 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: ");
    }
}

```

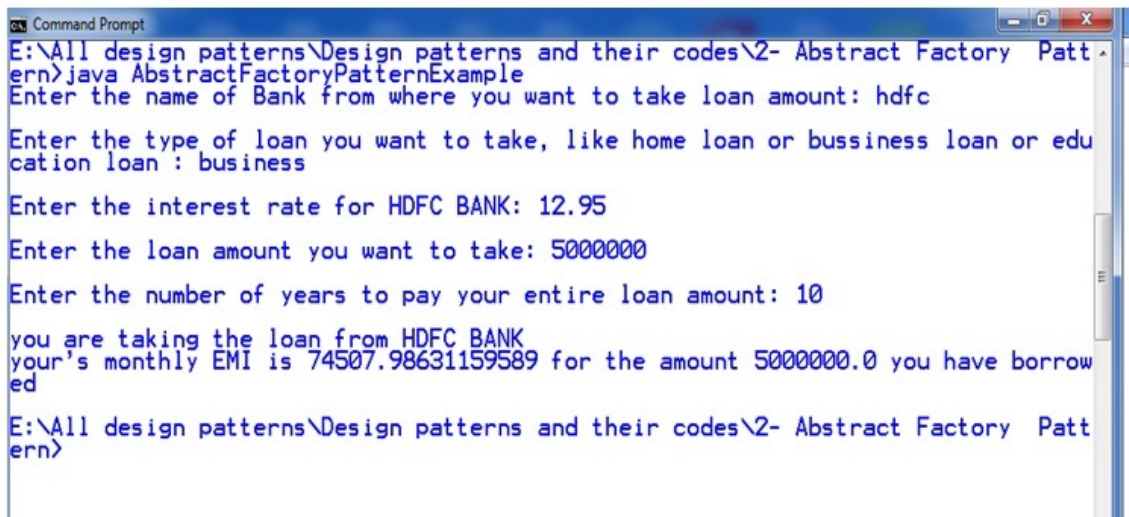
```
int years=Integer.parseInt(br.readLine());

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);
}
} //End of the AbstractFactoryPatternExample
```

download this Abstract Factory Pattern Example

Output



```
Command Prompt
E:\All design patterns\Design patterns and their codes\2- Abstract Factory Pattern>java AbstractFactoryPatternExample
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 : business
Enter the interest rate for HDFC BANK: 12.95
Enter the loan amount you want to take: 5000000
Enter the number of years to pay your entire loan amount: 10
you are taking the loan from HDFC BANK
your's monthly EMI is 74507.98631159589 for the amount 5000000.0 you have borrowed
E:\All design patterns\Design patterns and their codes\2- Abstract Factory Pattern>
```

← prev

next →













Help Others, Please Share








Join Javatpoint Test Series

| | | | |
|------------------|---------------|----------------|---------|
| Placement Papers | AMCAT | Bank PO/Clerk | GATE |
| TCS | eLitmas | UPSSSC | NEET |
| HCL | Java | Government | CAT |
| Infosys | Python | Exams | Railway |
| IBM | C Programming | SSC | CTET |
| Accenture | Networking | Civil Services | IIT JEE |
| | | SBI | |













Learn Latest Tutorials

| | | | |
|---|---|---|--|
|  Ansible Tutorial Ansible |  Mockito Tutorial Mockito |  Talend Tutorial Talend |  Microsoft Azure Tutorial Azure |
|  Sharepoint Tutorial SharePoint |  Powershell Tutorial Powershell |  Kali Linux Tutorial Kali Linux |  OpenCV Tutorial OpenCV |
|  Kafka Tutorial Kafka |  Pandas Tutorial Pandas |  Joomla Tutorial Joomla |  Reinforcement Learning Tutorial Reinforcement |





















Preparation

| | | | |
|--|--|---|--|
|  Aptitude Aptitude |  Logical Reasoning Reasoning |  Verbal Ability Verbal A. |  Interview Questions Interview |
|  Company Interview Questions Company | | | |

Trending Technologies

| | | | |
|--|---|---|---|
|  Artificial Intelligence Tutorial AI |  AWS Tutorial AWS |  Selenium tutorial Selenium |  Cloud tutorial Cloud |
|  Hadoop tutorial Hadoop |  ReactJS Tutorial ReactJS |  Data Science Tutorial D. Science |  Angular 7 Tutorial Angular 7 |
|  Blockchain Tutorial Blockchain |  Git Tutorial Git |  Machine Learning Tutorial ML |  DevOps Tutorial DevOps |

B.Tech / MCA

| | | | |
|---|---|---|---|
|  DBMS tutorial DBMS |  Data Structures tutorial DS |  DAA tutorial DAA |  Operating System tutorial OS |
|  Computer Network tutorial C. Network |  Compiler Design tutorial Compiler D. |  Computer Organization and Architecture COA |  Discrete Mathematics Tutorial D. Math. |
|  Ethical Hacking Tutorial E. Hacking |  Computer Graphics Tutorial C. Graphics |  Software Engineering Tutorial Software E. |  html tutorial Web Tech. |
|  Cyber Security tutorial Cyber Sec. |  Automata Tutorial Automata |  C Language tutorial C |  C++ tutorial C++ |
|  Java tutorial Java |  .Net Framework tutorial .Net |  Python tutorial Python |  List of Programs Programs |



Control
Systems
tutorial

Control S.



Data Mining
Tutorial

Data Mining