# Assignment . 10

**Aim:** Design and implement Factory design pattern for the given context. Consider Car building process, which requires many steps from allocating accessories to final makeup. These steps should be written as methods and should be called while creating an instance of a specific car type. Hatchback, Sedan, SUV could be the subclasses of Car class. Car class and its subclasses, CarFactory and TestFactoryPattern should be implemented.

**Objectives:** To learn the concept of Design pattern

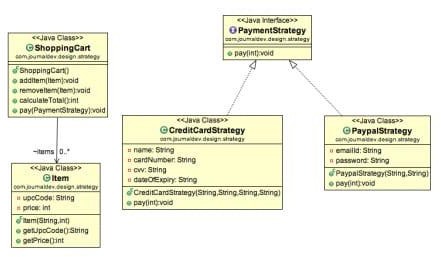
## Theory:

1. Design pattern
2. Factory design pattern diagram with example
3. Advantages of factory design pattern
4. Usage and the application where factory design patterns can be applied .

## Sample Code:

* Draw the class diagram for given context
* Crate classes such as **Car.java ,CarFactory.java ,**CarType .java
* LuxuryCar.java ,SedanCar.java ,SmallCar.java ,TestFactorypatern.java

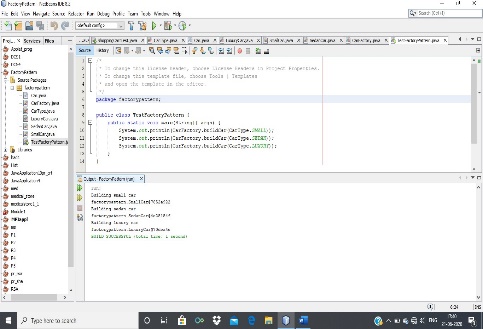
**Input:** Design and implement Factory design pattern for the given context. Consider Car building process, which requires many steps from allocating accessories to final makeup. These steps should be written as methods and should be called while creating an instance of a specific car type. Hatchback, Sedan, SUV could be the subclasses of Car class. Car class and its subclasses, CarFactory and TestFactoryPattern should be implemented.



**Output :**. Factory pattern –

Building small car factorypattern.SmallCar@7852e922 Building sedan car factorypattern.SedanCar@4e25154f Building luxury car factorypattern.LuxuryCar@70dea4e

BUILD SUCCESSFUL (total time: 1 second)



## Frequently Ask Question:

1. Give the applications where design patterns can be applied ?
2. Why factory pattern?
3. Explain factory pattern?
4. Draw the design Pattern with a context.
5. Give examples of creational design pattern .
6. Design application by applying the factory pattern .
7. Represent and implement a *Shape* interface which implements Circle ,Square, rectangle using Fatory pattern

Represent and implement for bill generation using GetPlanFactory to get a Plan object. Pass information (Domestic / commercial/ institutional) to get the type of object it needs.

ALGORITHM:

1 START

2 GET THE CUSTOMER DETAILS

3 MAKE A CLASS ACCOUNT FUNCTION

4 MAKE A SAVING ACCOUNT FUNCTION

5 IN CLASS BANK CALL ALL THE FUNCTIONS.

6 END

CODE:

package com.company;  
  
import java.util.Scanner;  
  
class Customer{  
 private String customerName;  
 private int customerAge;  
  
 public void setCustomerName(String customerName) {  
 this.customerName = customerName;  
 }  
  
 public String getCustomerName(){  
 return customerName;  
 }  
  
 public void setCustomerAge(int customerAge){  
 this.customerAge=customerAge;  
 }  
 public int getCustomerAge(){  
 return customerAge;  
 }  
}  
  
abstract class Account{  
 protected double balance ;  
 protected int accountId;  
 protected String accountType; *// here is account type* protected Customer custObj;  
  
 void setBalance(double balance){  
 this.balance = balance;  
 }  
 double getBalance(){  
 return balance;  
 }  
  
 void setAccountId(int accountId) {  
 this.accountId = accountId;  
 }  
 int getAccountId(){  
 return accountId;  
 }  
  
 void setAccountType(String accountType) {  
 this.accountType = accountType;  
 }  
  
 String getAccountType(){*//accounttype* return accountType;  
 }  
  
 void setCustObj(Customer custObj) {  
 this.custObj = custObj;  
 }  
 Customer getCustObj(){  
 return custObj;  
 }  
  
 public abstract boolean withdraw(double amount);  
}  
  
class SavingAccount extends Account{  
 private double minBalance;  
 public void setMinBalance(double minBalance){  
 this.minBalance = minBalance;  
 }  
 public double getMinBalance(){  
 return minBalance;  
 }  
  
 public boolean withdraw(double amount){  
 if((balance-amount)>minBalance){  
 balance-=amount;  
 return true;  
 }  
 else  
 return false;  
 }  
}  
  
  
class Bank{  
 Scanner in = new Scanner(System.*in*);  
  
 public SavingAccount a = new SavingAccount();  
 public Customer c =new Customer();  
  
 public SavingAccount createAccount(){  
 in.nextLine();  
  
 System.*out*.print("Enter your name = ");  
 String customername = in.nextLine();  
 c.setCustomerName(customername);  
  
 System.*out*.print("Enter your age = ");  
 int customerAge = in.nextInt();  
 if(customerAge<18){  
 do{  
 System.*out*.println("Minimum age should be 18 to create a bank account !");  
 customerAge = in.nextInt();  
 }while(customerAge<18);  
 }  
 c.setCustomerAge(customerAge);  
  
 a.setCustObj(c);  
 System.*out*.print("Enter your account ID = ");  
 int accountID = in.nextInt();  
 a.setAccountId(accountID);  
  
  
 System.*out*.print("Enter your account type = ");  
 String accountype = in.next();  
 a.setAccountType(accountype);  
  
  
 System.*out*.print("\nEnter the balance = ");  
 double balance = in.nextDouble();  
 a.setBalance(balance);  
  
 System.*out*.print("Enter the minimum balance = ");  
 double minbalance = in.nextDouble();  
 a.setMinBalance(minbalance);  
  
 return a;  
 }  
void getWithdrawAmount() {  
 System.*out*.print("Enter the amount you want to withdraw = ");  
 double amount = in.nextDouble();  
 if (amount > 20000) {  
 System.*out*.println("Withdrawal failed . Maximum limit of one withdrawal is Rs. 20000.");  
 } else {  
 if (a.withdraw(amount)) {  
 System.*out*.println("Withdrawal successfully done . " + a.getBalance());  
 } else {  
 System.*out*.println("Sorry !! Not enough balance .");  
 }  
 }  
}  
 public void depositAmount(double amount){  
 double bal = a.getBalance()+amount;  
 a.setBalance(bal);  
 System.*out*.println("Amount deposited successfully . Balance is = "+a.getBalance());  
 }  
 public void checkBalance(){  
 System.*out*.println("Balance is = "+a.getBalance());  
 }  
  
 public void displayAccountInfo(){  
 System.*out*.println("Welcome account holder :) "+c.getCustomerName()+" Following are your account details .");  
 System.*out*.println("Age = "+c.getCustomerAge());  
 System.*out*.println("Account ID = "+a.getAccountId());  
 System.*out*.println("Account type = "+a.getAccountType());  
 System.*out*.println("Account Balance = "+a.getBalance());  
 System.*out*.println("Account Minimum Balance = "+a.getMinBalance());  
 }  
  
}  
  
public class Assignmnetn09 {  
 public static void main(String[] args) {  
 Scanner in = new Scanner(System.*in*);  
 SavingAccount a ;  
 Bank bm = new Bank();  
  
 do{  
 System.*out*.println("\n\t1.Create Account\n\t2.Display Account\n\t3.Check Balance\n\t4.Deposit Amount\n\t5.Withdraw Amount\n\t6.Exit");  
 System.*out*.print("Enter your choice = ");  
 int choice = in.nextInt();  
 System.*out*.println("");  
  
 switch (choice) {  
 case 1 -> {  
 a = bm.createAccount();  
 System.*out*.println("------------------------------------------------");  
 }  
 case 2 -> {  
 bm.displayAccountInfo();  
 System.*out*.println("------------------------------------------------");  
 }  
 case 3 -> {  
 bm.checkBalance();  
 System.*out*.println("------------------------------------------------");  
 }  
 case 4 -> {  
 System.*out*.print("Enter the amount you deposit = ");  
 double amt = in.nextDouble();  
 bm.depositAmount(amt);  
 System.*out*.println("------------------------------------------------");  
 }  
 case 5 -> {  
 bm.getWithdrawAmount();  
 System.*out*.println("------------------------------------------------");  
 }  
 case 6 -> {  
 System.*out*.println("------------------------------------------------");  
 return;  
 }  
 default -> {  
 System.*out*.println("Invalid Input !");  
 System.*out*.println("-------------------------------------------------");  
 }  
 }  
 }while(true);  
  
  
 }  
}





CONCLUSION

IN THIS ASSIGNMENT WE HAVE SUCCESSFULLY LEARNED THE CONCEPT OF FACTORY DESIGN PATTERN IN JAVA.