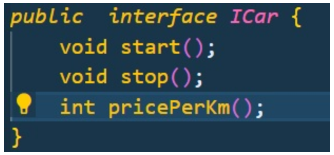
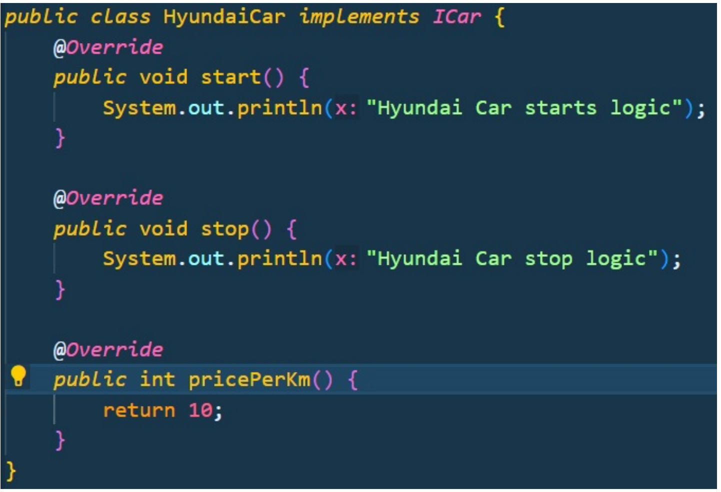
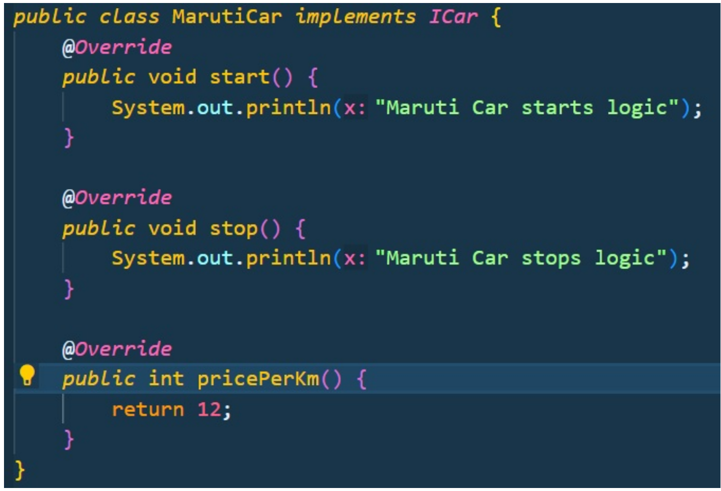


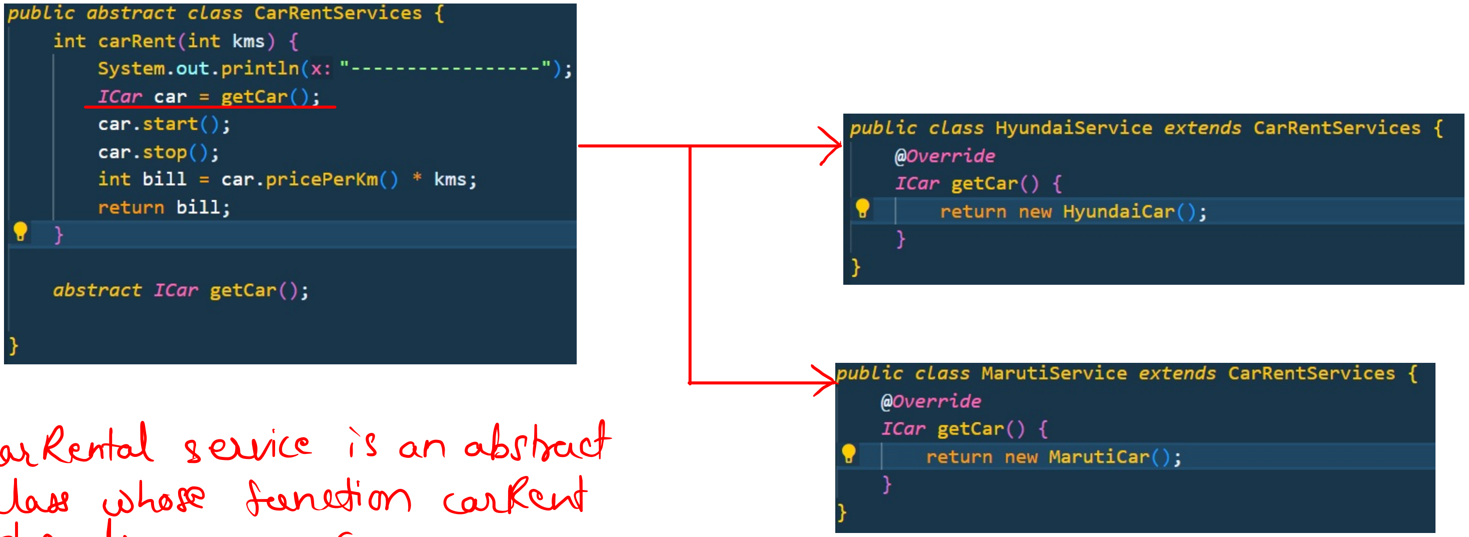
```
public  interface ICar {
    void start();
    void stop();
    int pricePerKm();
}
```



```
public class HyundaiCar implements ICar {
    @Override
    public void start() {
        System.out.println(x: "Hyundai Car starts logic");
    }

    @Override
    public void stop() {
        System.out.println(x: "Hyundai Car stop logic");
    }

    @Override
    public int pricePerKm() {
        return 10;
    }
}
```



```
public class MarutiCar implements ICar {
    @Override
    public void start() {
        System.out.println(x: "Maruti Car starts logic");
    }

    @Override
    public void stop() {
        System.out.println(x: "Maruti Car stops logic");
    }

    @Override
    public int pricePerKm() {
        return 12;
    }
}
```



```java
public abstract class CarRentServices {
    int carRent(int kms) {
        System.out.println(x: "----------------");
        ICar car = getCar();
        car.start();
        car.stop();
        int bill = car.pricePerKm() * kms;
        return bill;
    }

    abstract ICar getCar();

}
```

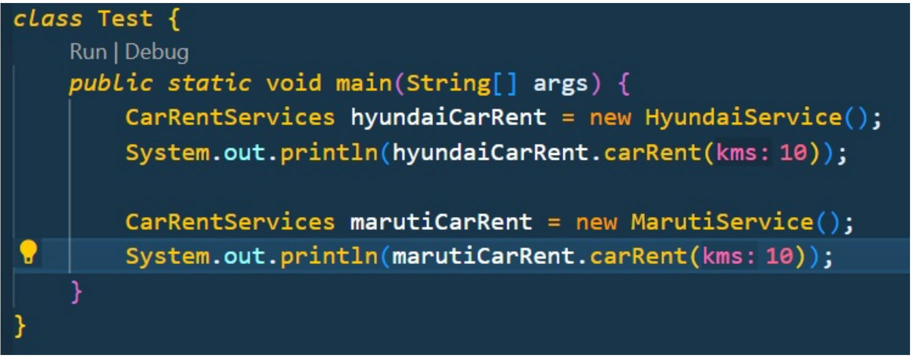
```java
public class HyundaiService extends CarRentServices {
    @Override
    ICar getCar() {
        return new HyundaiCar();
    }
}
```

```java
public class MarutiService extends CarRentServices {
    @Override
    ICar getCar() {
        return new MarutiCar();
    }
}
```

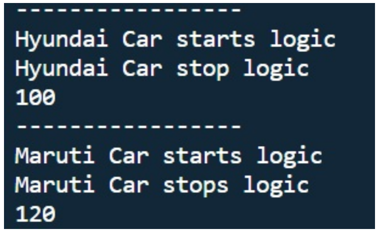
CarRental service is an abstract class whose function carRent depends upon a Car.
This car gets its object through getCar method which is abstract and unimplemented Factory Method.

HyundaiService & MarutiService classes extend CarRentServices and override this function to return the object of their own class.

This is called Factory method. design Pattern.

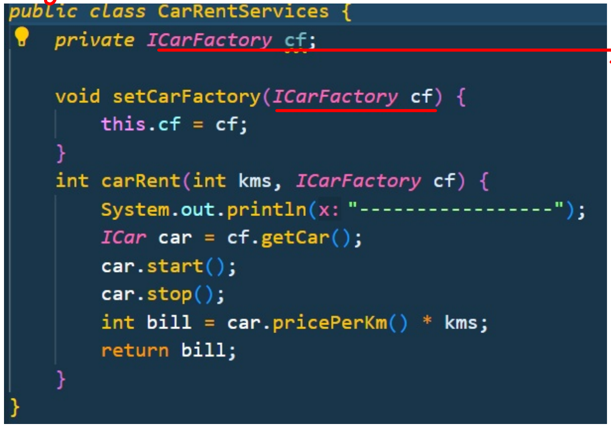


```
class Test {
    Run | Debug
    public static void main(String[] args) {
        CarRentServices hyundaiCarRent = new HyundaiService();
        System.out.println(hyundaiCarRent.carRent(kms: 10));

        CarRentServices marutiCarRent = new MarutiService();
        System.out.println(marutiCarRent.carRent(kms: 10));
    }
}
```

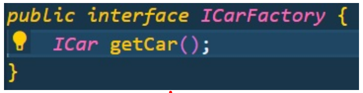


```
-----------------
Hyundai Car starts logic
Hyundai Car stop logic
100
-----------------
Maruti Car starts logic
Maruti Car stops logic
120
```

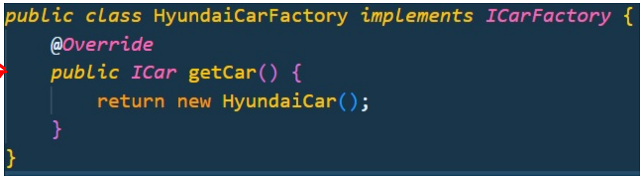
No longer abstract class



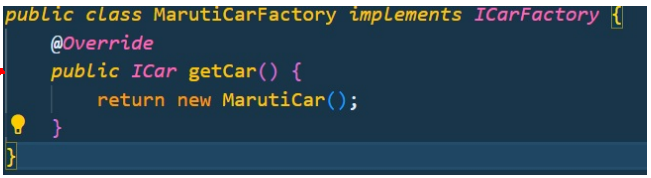
```java
public class CarRentServices {
    private ICarFactory cf;

    void setCarFactory(ICarFactory cf) {
        this.cf = cf;
    }
    int carRent(int kms, ICarFactory cf) {
        System.out.println(x: "----------------");
        ICar car = cf.getCar();
        car.start();
        car.stop();
        int bill = car.pricePerKm() * kms;
        return bill;
    }
}
```



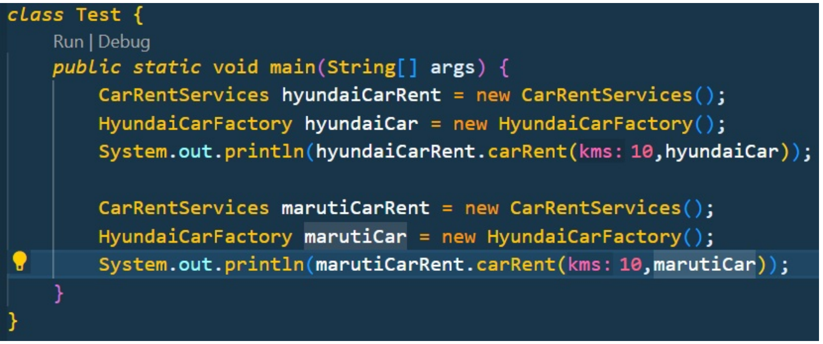
```java
public interface ICarFactory {
    ICar getCar();
}
```



```java
public class HyundaiCarFactory implements ICarFactory {
    @Override
    public ICar getCar() {
        return new HyundaiCar();
    }
}
```



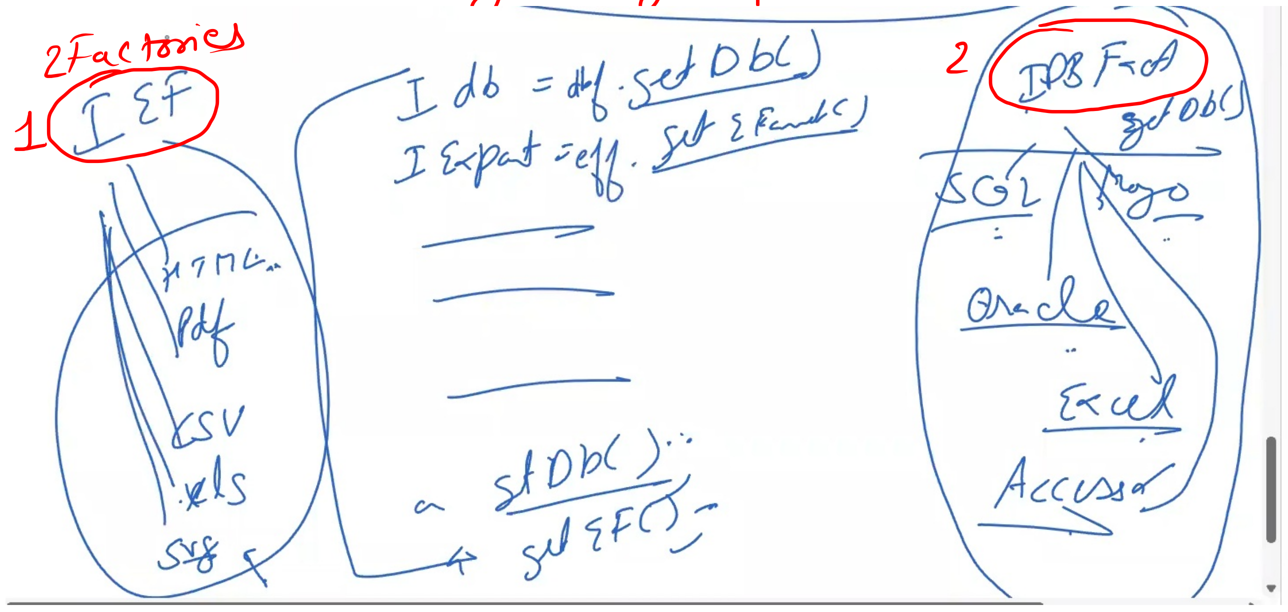
```java
public class MarutiCarFactory implements ICarFactory {
    @Override
    public ICar getCar() {
        return new MarutiCar();
    }
}
```



```java
class Test {
    Run | Debug
    public static void main(String[] args) {
        CarRentServices hyundaiCarRent = new CarRentServices();
        HyundaiCarFactory hyundaiCar = new HyundaiCarFactory();
        System.out.println(hyundaiCarRent.carRent(kms: 10,hyundaiCar));

        CarRentServices marutiCarRent = new CarRentServices();
        HyundaiCarFactory marutiCar = new HyundaiCarFactory();
        System.out.println(marutiCarRent.carRent(kms: 10,marutiCar));
    }
}
```

Car rent Services gets a car object from outside Factory method and sets it using setCarFactory. which it uses in the function of Car Rent.

Bridge Pattern applied over Factory Method Converts it into Abstract factory.
((Creating Interface socket for Factory classes so that multiple objects of Factory classes can be used with different implementation without burst of classes solved by bridge pattern.



5 × 5 = 25
but we only create
5 + 5 + 1 + 1 = 12
classes.