Factory Design Pattern

This is a *creational design pattern* that delegates the task of creating objects of a certain class to its subclasses. Alternatively, the objects could be interface types, created by implementing classes. For example, a Car factory may be able to create different types of cars; a Food factory may be able to create different kinds of foods depending on input parameters. This pattern is iniquitous in the OOP world.

The ***intent of the pattern*** is to:

1. Create objects without exposing the instantiation logic to the client.
2. Refer to the newly created object through a common interface

The ***implementation of the pattern*** is generally as follows:

- The client needs a product, but instead of creating it directly using the ***new*** operator, it asks the factory object for a new product, providing the information about the type of object it needs.
- The factory instantiates a new concrete product and then returns to the client the newly created product(cast to abstract product class or interface type).

The client uses the products as abstract products or interface types without being aware about their concrete implementation.

```
/**
  A Factory Design Pattern example
*/
interface Car{
   public String origin();
}

class  lamborghini implements Car{
   String name;
   public lamborghini(){
         name="Lombarghini";
         System.out.println("I am a Lamborghini and I am very expensive");
   }
   public String origin(){
      return "I am from Italy";
   }
}

class BMW implements Car{
   String name;
   public BMW(){
         name="BMW";
         System.out.println("I am a BMW and I am moderately expensive");
   }
   public String origin(){
      return "I am from Germany";
   }
}
```

```
class Kia implements  Car{
   String name;
   public Kia(){
         name="Kia";
         System.out.println("I am a Kia and I am not very expensive");
   }
   public String origin(){
      return "I am from S. Korea";
   }
}

class CarFactory{
   public static Car makeCar(String criteria) {
         if ( criteria.equals("very expensive") )
            return new lamborghini();
         else if ( criteria.equals("moderately expensive") )
            return new BMW();
         else if ( criteria.equals("inexpensive") )
            return new Kia();
         return null;
   }
}

/**
 * A driver program to demonstrate our Car factory.
 */
public class factory{
   public static void main(String[] args)  {
         // expensive car
         Car myCar1 = CarFactory.makeCar("very expensive");
         System.out.println(myCar1.origin());
         // moderately expensive
         Car myCar2 = CarFactory.makeCar("moderately expensive");
         System.out.println(myCar2.origin());
         // inexpensive car
         Car myCar3 = CarFactory.makeCar("inexpensive");
         System.out.println(myCar3.origin());
   }
}
```