Factory Design Pattern

Definition: - Factory Design pattern is a creational design pattern of GOF design pattern. It will hide the process of object creation and return the same class object is called factory pattern.

Problem:- Creating object for knowly its creational process and dependencies is very complex and creating multiple object for multiple classes and utilizing one of the object based on the user supplied data, because the remaining objects unnecessarily created, it will wasted the heap memory and also effect the performance of the application.

Solution:- use factory design pattern, here method of the factory class instantiate or create the object one of the subclass that is based on the data that is supplied by the user at runtime, that means objects for remaining subclasses/other classes will not be created.

Problem:-

Car.java

```
package com.nt.car;

public abstract class Car {

	public abstract void assemble();

	public abstract void roadTest();

	public abstract void transportation();
}
```

LuxuryCar.java

```
package com.nt.car;

public class LuxuryCar extends Car {

	@Override
	public void assemble() {
		System.out.println("LuxuryCar: Luxury Car assembling");
	}

	@Override
```

```
	public void roadTest() {
		System.out.println("LuxuryCar: Luxury Car roadTesting");
	}

	@Override
	public void transportation() {
		System.out.println("LuxuryCar: Luxury Car Transportation");
	}

}
```

SimpleCar.java

```
package com.nt.car;

public class SimpleCar extends Car {

	@Override
	public void assemble() {
		System.out.println("SimpleCar: Simple car assembling");
	}

	@Override
	public void roadTest() {
		System.out.println("SimpleCar: Simple car roadTesting");
	}

	@Override
	public void transportation() {
		System.out.println("SimpleCar: Simple car transportation");
	}
}
```

SportsCar.java

```
package com.nt.car;

public class SportsCar extends Car {

	@Override
	public void assemble() {
		System.out.println("SportsCar.assemble()");

	}

	@Override
	public void roadTest() {
		System.out.println("SportsCar.roadTest()");

	}
```

```
	@Override
	public void transportation() {
		System.out.println("SportsCar.transportation()");
	}

}
```

FactoryTest1.java

```
package com.nt.test;

import com.nt.car.Car;
import com.nt.car.SimpleCar;

public class FactoryTest1 {

	public static void main(String[] args) {
		Car car = new SimpleCar();
		car.assemble();
		car.roadTest();
		car.transportation();
		System.out.println("Simple car ready.....");
	}

}
```

FactoryTest2.java

```
package com.nt.test;

import com.nt.car.Car;
import com.nt.car.LuxuryCar;

public class FactoryTest2 {

	public static void main(String[] args) {
		Car  car=null;
		car=new LuxuryCar();
		car.assemble();
		car.roadTest();
		car.transportation();
		System.out.println("Luxory Car Ready ......");
	}

}
```

FactoryTest3.java

```
package com.nt.test;

import com.nt.car.Car;
import com.nt.car.SportsCar;

public class FactoryTest3 {

	public static void main(String[] args) {
		Car car = null;
		car = new SportsCar();
		car.assemble();
		car.roadTest();
		car.transportation();
		System.out.println("Sports Car Ready.....");
	}

}
```

Solution:-

All the classes and abstract classes is same we just create one CarFactory class that hide the object creation to others.

CarFactory.java

```
package com.nt.factory;

import com.nt.car.Car;
import com.nt.car.LuxuryCar;
import com.nt.car.SimpleCar;
import com.nt.car.SportsCar;

public class CarFactory {

	public static Car getInstance(String type) {
		Car car = null;
		if (type.equalsIgnoreCase("simple")) {
			car = new SimpleCar();
		}

		else if (type.equalsIgnoreCase("luxury")) {
			car = new LuxuryCar();
		}

		else if (type.equalsIgnoreCase("sports")) {
			car = new SportsCar();
		}
```

```
                else {
                        throw new IllegalArgumentException("Invalid Car type");
                }
                car.assemble();
                car.roadTest();
                car.transportation();

                return car;
        }
}
```

FactoryTest1.java

```
package com.nt.test;

import com.nt.car.Car;
import com.nt.car.SimpleCar;
import com.nt.factory.CarFactory;

public class FactoryTest1 {

        public static void main(String[] args) {
                Car car = CarFactory.getInstance("simple");
                System.out.println("Simple car ready.....");
        }

}
```