1. We can use static method in abstract class

public class abstract ShapeFactory {

public static Shape getShape(String shapeType){

if(shapeType == null){ return null; }

if(shapeType.equalsIgnoreCase(“SQR")){ return new Square(); } else

}

}

1. We can use singleton design pattern

class ShapeFactory {

private static ShapeFactory instance;

private ShapeFactor(){}

public statc getShapeFactory(){

if (instance==null){

instance=new ShapeFactory)();

}

Return instance;

public Shape getShape(String shapeType){

if(shapeType == null){ return null; }

if(shapeType.equalsIgnoreCase(“SQR")){ return new Square(); } else

}

}

1. We can use protected method in same package

Package ShapeFactory;

public interface Shape {

void draw();

}

public class Rectangle implements Shape {

public void draw() {

System.out.println("Inside Rectangle::draw() method.");

}

}

public class Square implements Shape {

public void draw() {

System.out.println("Inside Square::draw() method.");

}

}

public class ShapeFactory {

protected ShapeFactor(){}

public Shape getShape(String shapeType){

if(shapeType == null){ return null; }

if(shapeType.equalsIgnoreCase(“SQR")){ return new Square(); } else

}

}

1. We use singleton design pattern to create only one factory object

We use template design pattern

1. Polymorphism – product object put into interface instances

Encapsulation – use private objects and methods

Abstraction – use interface method

1. Class A{

Private static A instance;

Private A(){}

Public static A getInstance(){

If(instance==null){

Instance=new A();

}

Return instance;

}

Class B{

Private A a;

Protected B(){

{

Public A getA(){

a=a.getInstance();

}

}

Class C{

}