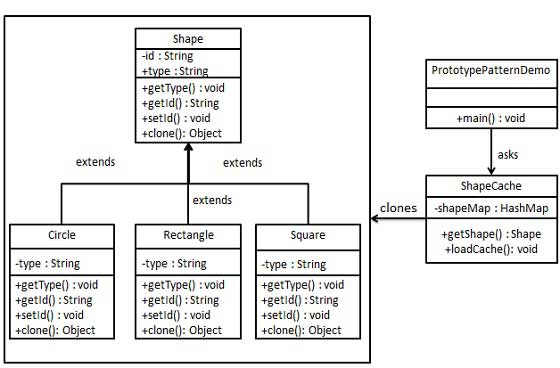
Prototype pattern refers to creating duplicate object while keeping performance in mind. This type of design pattern comes under creational pattern as this pattern provides one of the best ways to create an object.

This pattern involves implementing a prototype interface which tells to create a clone of the current object. This pattern is used when creation of object directly is costly. For example, an object is to be created after a costly database operation. We can cache the object, returns its clone on next request and update the database as and when needed thus reducing database calls.

Source : wikipedia

This pattern creates the kind of object using its prototype. In other words, while creating the object of Prototype object, the class actually creates a clone of it and returns it as prototype. You can see here, we have used Clone method to clone the prototype when required.

*// Prototype pattern*  
**public** **abstract** **class** **Prototype** **implements** Cloneable {  
 **public** Prototype clone() **throws** CloneNotSupportedException{  
 **return** (Prototype) **super**.clone();  
 }  
}  
   
**public** **class** **ConcretePrototype1** **extends** Prototype {  
 @Override  
 **public** Prototype clone() **throws** CloneNotSupportedException {  
 **return** (ConcretePrototype1)**super**.clone();  
 }  
}  
  
**public** **class** **ConcretePrototype2** **extends** Prototype {  
 @Override  
 **public** Prototype clone() **throws** CloneNotSupportedException {  
 **return** (ConcretePrototype2)**super**.clone();  
 }  
}



import java.util.Hashtable;

//step 1

//create abstract class implementing cloneable interface

abstract class Shape implements Cloneable{

private String id;

protected String type;

abstract void draw();

public String getType() {

return type;

}

public String getId() {

return id;

}

public void setId(String id) {

this.id = id;

}

public Object clone() {

Object clone = null;

try {

clone = super.clone();

}catch(CloneNotSupportedException e) {

e.printStackTrace();

}

return clone;

}

}

//step 2

//create concrete classes extending the above class

class Rectangle extends Shape{

public Rectangle() {

type = "Rectangle";

}

@Override

void draw() {

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

}

}

class Square extends Shape{

public Square() {

type = "Square";

}

@Override

void draw() {

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

}

}

class Circle extends Shape{

public Circle() {

type = "Circle";

}

@Override

void draw() {

System.out.println("Inside Circle draw method");

}

}

//Step 3

//Create a class to get concrete classes from database and store them in a Hashtable.

class ShapeCache{

private static Hashtable<String, Shape> shapeMap = new Hashtable();

public static Shape getShape(String shapeId) {

Shape cachedShape = shapeMap.get(shapeId);

return (Shape) cachedShape.clone();

}

//for each shape run database query and create shape

//in this example we are manually creating shapes

public static void loadCache() {

Circle circle = new Circle();

circle.setId("1");

shapeMap.put(circle.getId(), circle);

Square square = new Square();

square.setId("2");

shapeMap.put(square.getId(), square);

Rectangle rectangle = new Rectangle();

rectangle.setId("3");

shapeMap.put(rectangle.getId(), rectangle);

}

}

public class PrototypePatternDemo {

public static void main(String[] args) {

ShapeCache.loadCache();

Shape clonedShape = (Shape) ShapeCache.getShape("1");

//System.out.println("shape = " + clonedShape.getType());

Shape clonedShaped = (Shape) ShapeCache.getShape("1");

//System.out.println("shape = " + clonedShape.getType());

clonedShape.draw();

clonedShaped.draw();

Shape clonedShape1 = (Shape) ShapeCache.getShape("2");

System.out.println("shape = " + clonedShape1.getType());

Shape clonedShape2 = (Shape) ShapeCache.getShape("3");

System.out.println("shape = " + clonedShape2.getType());

}

}