# PROTOTYPE PATTERN

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 way 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, a 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 as and when needed thus reducing database calls.

Implementation

We're going to create an abstract class *Shape* and concrete classes extending the *Shape* class. A class *ShapeCache* is defined as a next step which stores shape objects in a *Hashtable* and returns their clone when requested.

PrototypPatternDemo, our demo class will use *ShapeCache* class to get a *Shape* object.
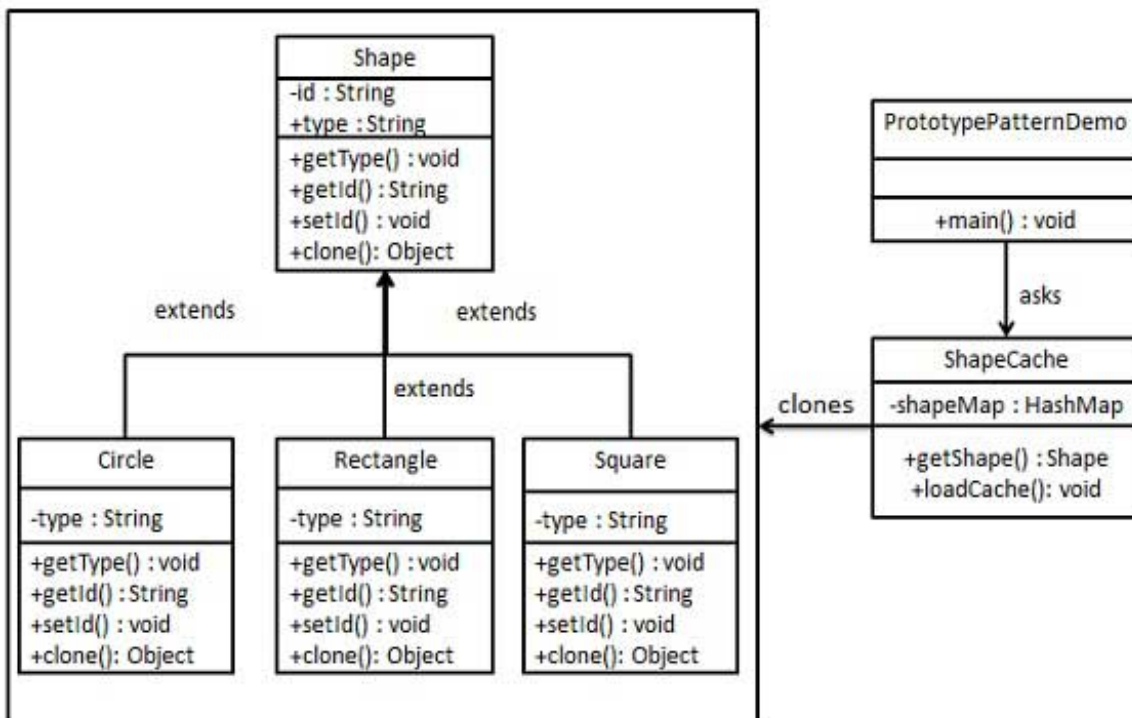
Step 1

Create an abstract class implementing *Clonable* interface.

Shape.java

```java
public 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.

Rectangle.java

```
public class Rectangle extends Shape {

   public Rectangle(){
     type = "Rectangle";
   }

   @Override
   public void draw() {
      System.out.println("Inside Rectangle::draw() method.");
   }
}
```

Square.java

```
public class Square extends Shape {

   public Square(){
     type = "Square";
   }

   @Override
   public void draw() {
      System.out.println("Inside Square::draw() method.");
   }
}
```

Circle.java

```
public class Circle extends Shape {

   public Circle(){
     type = "Circle";
   }

   @Override
   public 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*.

ShapeCache.java

```java
import java.util.Hashtable;

public class ShapeCache {

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

   public static Shape getShape(String shapeId) {
      Shape cachedShape = shapeMap.get(shapeId);
      return (Shape) cachedShape.clone();
   }

   // for each shape run database query and create shape
   // shapeMap.put(shapeKey, shape);
   // for example, we are adding three 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);
   }
}
```

Step 4

PrototypePatternDemo uses *ShapeCache* class to get clones of shapes stored in a *Hashtable*.

PrototypePatternDemo.java

```java
public class PrototypePatternDemo {
   public static void main(String[] args) {
      ShapeCache.loadCache();

      Shape clonedShape = (Shape) ShapeCache.getShape("1");
      System.out.println("Shape : " + clonedShape.getType());

      Shape clonedShape2 = (Shape) ShapeCache.getShape("2");
      System.out.println("Shape : " + clonedShape2.getType());

      Shape clonedShape3 = (Shape) ShapeCache.getShape("3");
      System.out.println("Shape : " + clonedShape3.getType());
   }
}
```

Step 5

Verify the output.

```
Shape : Circle
Shape : Square
Shape : Rectangle
```