

### Prototype Design Pattern

In today's world, one of the largest concerns related to programming is costs.

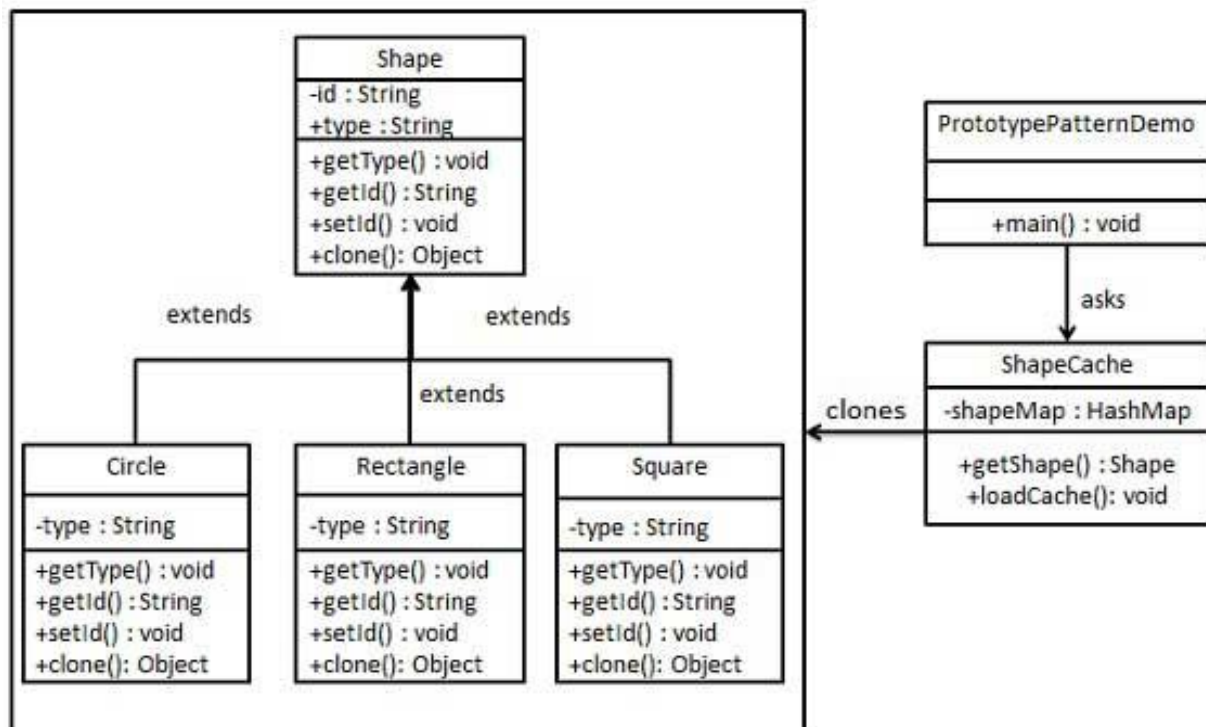
Programmers are striving to find ways to improve the performance and costs of their computer resources. This cost saving effort can be accomplished through the prototype design pattern. The prototype design pattern is used for creating new objects (instances) by cloning (copying) other objects. This pattern provides a mechanism to copy the original object to a new object and then modifying it according to our needs. One reason for using the prototype design pattern is if the creation an instance of a class is incredibly time-consuming or complex in some way. Also, if the cost of creating a new object is expensive or if the creation is resource intensive, then programmers could clone the object instead.

An example of the use of the prototype design pattern would be in the case where we have an Object that loads data from a database. Suppose we need to modify the data in our program multiple times. It is not efficient and practical to create the Object using a new keyword in order to load all the data from database. The better approach would be to clone the existing object into a new object. Then we can proceed with the data manipulation more efficiently through the prototype design.

The prototype design pattern uses three different types of classes. Suppose these classes are called the Prototype, ConcretePrototype, and Client. The Prototype class declares an interface for cloning itself. Then the Client class will create a new object by asking a prototype to clone itself. The ConcretePrototype will implement the cloning operation through the Clone() method,

which will making a new instance of itself. Thus, it is incredibly simple and efficient to apply the prototype design pattern.

An example of the prototype design pattern would be a Shape application. This calls for an abstract class called Shape and three concrete classes that extend Shape. These classes are Circle, Rectangle, and Square. Then the clone is returned in the ShapeCache class. Here is the UML diagram of the Shape application example:



The code for the **Shape.java** class:

```
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;
}
}
```

The code for the **Rectangle.java** class:

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

The code for the **Square.java** class:

```
public class Square extends Shape {
    public Square(){
        type = "Square";
    }
}
```

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

The code for the **Circle.java** class:

```
public class Circle extends Shape {
    public Circle(){
        type = "Circle";
    }
    @Override
    public void draw() {
        System.out.println("Inside Circle::draw() method.");
    }
}
```

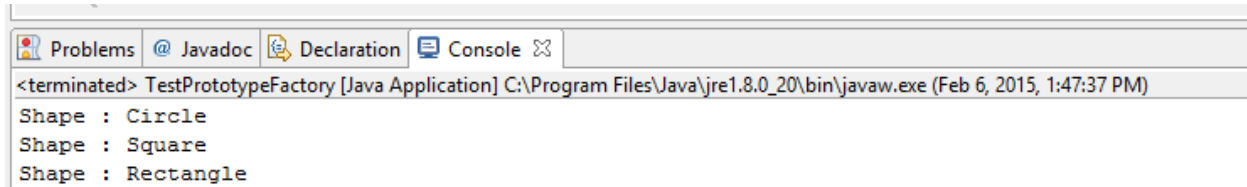
The code for the **ShapeCache.java** class:

```
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();
    }
}
```

The code for the **TestPrototypeFactory.java** class:

```
public class TestPrototypeFactory {
    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());
    }
}
```

Here is the output of the program:



```
<terminated> TestPrototypeFactory [Java Application] C:\Program Files\Java\jre1.8.0_20\bin\javaw.exe (Feb 6, 2015, 1:47:37 PM)
Shape : Circle
Shape : Square
Shape : Rectangle
```

The Shape application uses the prototype design pattern in order to clone the shapes needed. The class Shape is an abstract class that implements the Cloneable interface. Then the Rectangle, Circle, and Square are concrete classes that extend the abstract class, Shape. The class ShapeCache returns the clone of the shape objects, which is stored in a hashtable, when requested. This application requires the instances of Rectangle, Square and Circle classes to be used frequently. It would be extremely costly to create their instances every time. Therefore, the prototype design pattern is more practical in this case. First, we will create the prototype instances. Then when there is a need for a new instance, the program will simply clone the prototype. This will create a set of identical objects, whose type is determined at runtime. Thus, the Shape program is an example of how the prototype design pattern can be used efficiently.

A real life example can be used to better understand the prototype design pattern. A photocopier is a machine that makes paper copies of documents and other visual images quickly and cheaply. In designing a photocopier, we can use the prototype pattern. You can get exact copies of the original document instead of asking the original source to repeat and send the same document again. Making a copy is cheaper and more efficient way of getting the object, which in this example is the document.

Another example of the prototype design pattern can be seen through Minecraft, which is one of the most popular games of all time. It has a unique Block like art. Every object and terrain is represented by combining different colored or textured block. This is implemented using prototype pattern. All the prototype objects are actually of the same class Block, but each object has had different properties set on it so that it looks and behaves differently, for example:

```
prototypes.dirt = new Block;  
prototypes.dirt.texture = new Image("dirt.jpg");  
prototypes.dirt.hardness = 1;  
  
prototypes.stone = new Block;  
prototypes.stone.texture = new Image("stone.jpg");  
prototypes.stone.hardness = 9;
```

So instead of sub classing where you would write new DirtBlock or new StoneBlock, you would instead write prototypes.dirt.clone() or prototypes.stone.clone().

There are many advantages to using the prototype design pattern. First, it has a simplified object copy process. It reduces sub-classing and the load of initialization. It makes it easier to add and remove products at runtime. It also hides the complexities of making new instances from the client. The prototype design will allow the client to generate objects whose type is unknown. Lastly in some circumstances, copying an object is more efficient than creating a new object. The only disadvantage of this design pattern is that classes that have circular references to other classes cannot really be cloned. Otherwise, the prototype design pattern can be an incredibly efficient and inexpensive tool to be used in programs.

Works Cited

"Design Patterns Prototype Pattern." Tutorials Point. N.p., n.d. Web. 09 Feb. 2015.

"Prototype Design Pattern: Creating Another Dolly." *Java Code Geeks RSS*. N.p., 02 Nov. 2012.  
Web. 09 Feb. 2015.

"Prototype Design Pattern in Java - How To Do In Java." *How To Do In Java*. N.p., 04 Jan.  
2013. Web. 09 Feb. 2015.

"Prototype Pattern." Object Oriented Design, n.d. Web. 09 Feb. 2015.

"Prototype Pattern in Java." *JournalDev*. N.p., n.d. Web. 09 Feb. 2015.

"Sneha's Blog." : Prototype Pattern in Java. N.p., n.d. Web. 09 Feb. 2015.