Prototype Pattern

OODP, 2022

Copy and change
rather than creating with new;
Creating with **new** is costly;
Copy general properties, and change necessary part;

```
import java.util.*;
public class Car implements Cloneable {
  private String str;
  private Date date;

  public Car(String str) {
     this.str = str;
  }
  public String getStr(){
    return str;
  }
```

```
 public void setDate(Date date) {
    this.date = new Date(date.getTime());
  }
  public Date getDate() {
    return date;
  }
  @Override
  public Object clone() throws CloneNotSupportedException {
    Car tmp = (Car)super.clone();
    return tmp;
  }
}
```

```
import java.util.*;
public class Test {
  public static void main(String[] args) {
    Car car = new Car("캡티바");
    try{
      Car car1 = (Car)car.clone();
      car1.setDate(new Date(2015,0,1));
      Car car2 = (Car)car.clone();
      car2.setDate(new Date(2014,2,1));
      System.out.println(car1.getDate());
      System.out.println(car2.getDate());
    }catch(CloneNotSupportedException e) {
      e.printStackTrace();
    }
  }
}
```

```
Fri Jan 01 00:00:00 KST 3915
Sun Mar 01 00:00:00 KST 3914
```

```
Car car1 = (Car)car.clone();
      car1.setDate(new Date(2015,0,1));
      Car car2 = (Car)car.clone();
      car2.setDate(new Date(2014,2,1));
      System.out.println(car1.getStr());
      System.out.println(car1.getDate());
      System.out.println(car2.getStr());
      System.out.println(car2.getDate());
```

Copy general properties, and change necessary part;

```
Car car = new Car("캡티바");
    try{
      Car car1 = (Car)car.clone();
      car1.setDate(new Date(2015,0,1));
      Car car2 = (Car)car.clone();
      car2.setDate(new Date(2014,2,1));
      System.out.println(car1.getDate());
      System.out.println(car2.getDate());
   System.out.println(car1.hashCode());
   System.out.println(car2.hashCode());
    }catch(CloneNotSupportedException e) {
      e.printStackTrace();
    }
```

```
Fri Jan 01 00:00:00 KST 3915
Sun Mar 01 00:00:00 KST 3914
118352462
1550089733
```

Compare Prototype with Singleton

```
import java.util.*;
public class MySingleton {
    //the static singleton object
    private static MySingleton theObject;
    private MySingleton() {
    }
    public static MySingleton createMySingleton() {
        if (theObject == null)
            theObject = new MySingleton();
        return theObject;
    }
}
```

```
public class Main {
        public void createSingleton() {
      MySingleton ms1 =       MySingleton.createMySingleton();
      MySingleton ms2 =       MySingleton.createMySingleton();
      System.out.println( ms1 == ms2 );
    }
    public static void main(String[] args) {
      new  Main().createSingleton();
    }
}
```

```
true
```

```
abstract class Prototype implements Cloneable {
        @Override
        public Prototype clone()
                throws CloneNotSupportedException {
        return (Prototype)super.clone();
        }
       public abstract void setX(int x);
        public abstract void printX();
        public abstract int getX();
}
```

```
class PrototypeImpl extends Prototype {
        int x;
        public PrototypeImpl(int x) {
                this.x = x;
        }
        public void setX(int x) {
                this.x = x;
        }
        public void printX() {
                System.out.println("Value :" + x);
        }
        public int getX() {
                return x;
        }
}
```

```
public class PrototypeTest {
    public static void main(String args[])
        throws CloneNotSupportedException {
        Prototype prototype =
                new PrototypeImpl(1000);
        for (int i = 1; i < 10; i++) {
     Prototype tempotype =  prototype.clone();
     tempotype.setX( tempotype.getX() * i);
       tempotype.printX();
      }
   }
}
```

Value :1000
Value :2000
Value :3000
Value :4000
Value :5000
Value :6000
Value :7000
Value :8000
Value :9000

Yuki Book Example

```
"Hello, world."
 ~~~~~~~~~~~~~~~
*****************
* Hello, world. *
*****************
/////////////////
/ Hello, world. /
/////////////////
```

```
import framework.*;
public class Main {
    public static void main(String[] args) {
    Manager manager = new Manager();
    UnderlinePen upen = new UnderlinePen('~');
    MessageBox mbox = new MessageBox('*');
    MessageBox sbox = new MessageBox('/');
    manager.register("strong message", upen);
    manager.register("warning box", mbox);
    manager.register("slash box", sbox);
```

```
Product p1 = manager.create("strong message");
 p1.use("Hello, world.");
Product p2 = manager.create("warning box");
p2.use("Hello, world.");
Product p3 = manager.create("slash box");
p3.use("Hello, world.");
```

```
package framework;
import java.lang.Cloneable;

public interface Product extends Cloneable {
    public abstract void use(String s);
    public abstract Product createClone();
}
```

```
package framework;
import java.util.*;
public class Manager {
    private HashMap showcase = new HashMap();
    public void register(String name, Product proto) {
        showcase.put(name, proto);
    }
    public Product create(String protoname) {
        Product p = (Product)showcase.get(protoname);
        return p.createClone(); //copy
    }
}
```

```
import framework.*;
public class UnderlinePen implements Product {
    private char ulchar;
    public UnderlinePen(char ulchar) {
        this.ulchar = ulchar;
    }
    public void use(String s) {
        int length = s.getBytes().length;
        System.out.println("₩""  + s + "₩"");
        System.out.print(" ");
        for (int i = 0; i < length; i++) {
            System.out.print(ulchar);
        }
        System.out.println("");
    }
```

```
 public Product createClone() {
        Product p = null;
        try {
            p = (Product)clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return p;
    }
}
```

```
import framework.*;
public class MessageBox implements Product {
    private char decochar;
    public MessageBox(char decochar) {
        this.decochar = decochar;
    }
    public void use(String s) {
        int length = s.getBytes().length;
        for (int i = 0; i < length + 4; i++) {
            System.out.print(decochar);
        }
        System.out.println("");
```

```
 System.out.println(decochar + " "  + s + " " + decochar);
        for (int i = 0; i < length + 4; i++) {
            System.out.print(decochar);
        }
        System.out.println("");
    }
    public Product createClone() {
        Product p = null;
        try {
            p = (Product)clone();
        } catch (CloneNotSupportedException e) {
            e.printStackTrace();
        }
        return p;
    }
}
```

```
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());
   }
}
```

```
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();
    }
    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);
    }
}
```

What is the role of loadCache()?

```
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;
    }
}
```

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

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