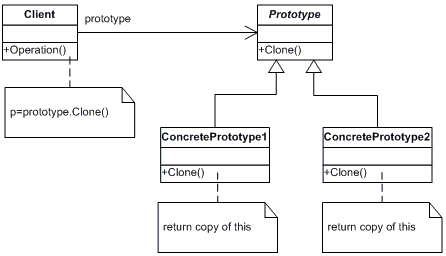
原型模式是一种创建型设计模式,它通过复制一个已经存在的实例来返回新的实例,而不是新建实例.被复制的实例就是我们所称的原型,这个原型是可定制的.  
原型模式多用于创建复杂的或者耗时的实例, 因为这种情况下,复制一个已经存在的实例可以使程序运行更高效,或者创建值相等,只是命名不一样的同类数据.

原型模式中的拷贝分为"浅拷贝"和"深拷贝":  
浅拷贝: 对值类型的成员变量进行值的复制,对引用类型的成员变量只复制引用,不复制引用的对象.  
深拷贝: 对值类型的成员变量进行值的复制,对引用类型的成员变量也进行引用对象的复制.

类图:



实例一:浅拷贝  
public class Prototype implements Cloneable {  
 private String name;

 public String getName() {  
  return name;  
 }

 public void setName(String name) {  
  this.name = name;  
 }

 public Object clone() {  
  try {  
   return super.clone();  
  } catch (CloneNotSupportedException e) {     
   e.printStackTrace();  
   return null;  
  }  
 }   
   
}

public class TestMain {

 public static void main(String[] args) {  
  testPrototype();  
 }  
   
 private static void testPrototype(){  
  Prototype pro = new Prototype();  
  pro.setName("original object");  
  Prototype pro1 = (Prototype)pro.clone();  
  pro.setName("changed object1");  
    
  System.out.println("original object:" + pro.getName());  
  System.out.println("cloned object:" + pro1.getName());  
    
 }

}  
结果:  
original object:changed object1  
cloned object:original object

实例二: 浅拷贝  
public class Prototype{  
 private String name;

 public String getName() {  
  return name;  
 }

 public void setName(String name) {  
  this.name = name;  
 }   
   
}  
public class NewPrototype implements Cloneable {  
   
 private String id;  
   
 public String getId() {  
  return id;  
 }

 public void setId(String id) {  
  this.id = id;  
 }

 private Prototype prototype;  
   
 public Prototype getPrototype() {  
  return prototype;  
 }

 public void setPrototype(Prototype prototype) {  
  this.prototype = prototype;  
 }

 public Object clone(){   
  try {  
   return super.clone();  
  } catch (CloneNotSupportedException e) {  
   e.printStackTrace();  
   return null;  
  }    
 }

}  
public class TestMain {

 public static void main(String[] args) {  
  // TODO Auto-generated method stub  
  testPrototype();  
 }  
   
 private static void testPrototype(){  
  Prototype pro = new Prototype();  
  pro.setName("original object");  
  NewPrototype newObj = new NewPrototype();  
  newObj.setId("test1");  
  newObj.setPrototype(pro);  
    
  NewPrototype copyObj = (NewPrototype)newObj.clone();  
  copyObj.setId("testCopy");  
  copyObj.getPrototype().setName("changed object");  
    
  System.out.println("original object id:" + newObj.getId());  
  System.out.println("original object name:" + newObj.getPrototype().getName());  
    
  System.out.println("cloned object id:" + copyObj.getId());  
  System.out.println("cloned object name:" + copyObj.getPrototype().getName());  
    
 }

}

结果:  
original object id:test1  
original object name:changed object  
cloned object id:testCopy  
cloned object name:changed object

实例三: 深拷贝  
public class Prototype implements Cloneable {  
 private String name;

 public String getName() {  
  return name;  
 }

 public void setName(String name) {  
  this.name = name;  
 }

 public Object clone() {  
  try {   
   return super.clone();  
  } catch (CloneNotSupportedException e) {     
   e.printStackTrace();  
   return null;  
  }  
 }   
   
}

public class NewPrototype implements Cloneable {  
   
 private String id;  
   
 public String getId() {  
  return id;  
 }

 public void setId(String id) {  
  this.id = id;  
 }

 private Prototype prototype;  
   
 public Prototype getPrototype() {  
  return prototype;  
 }

 public void setPrototype(Prototype prototype) {  
  this.prototype = prototype;  
 }

 public Object clone(){  
  NewPrototype ret = null;  
  try {  
   ret = (NewPrototype)super.clone();  
   ret.prototype = (Prototype)this.prototype.clone();  
   return ret;  
  } catch (CloneNotSupportedException e) {  
   e.printStackTrace();  
   return null;  
  }    
 }

}

public class TestMain {

 /\*\*  
  \* @param args  
  \*/  
 public static void main(String[] args) {  
  testDeepCopy();  
 }  
   
 private static void testDeepCopy(){  
  Prototype pro = new Prototype();  
  pro.setName("original object");  
  NewPrototype newObj = new NewPrototype();  
  newObj.setId("test1");  
  newObj.setPrototype(pro);  
    
  NewPrototype copyObj = (NewPrototype)newObj.clone();  
  copyObj.setId("testCopy");  
  copyObj.getPrototype().setName("changed object");  
    
  System.out.println("original object id:" + newObj.getId());  
  System.out.println("original object name:" + newObj.getPrototype().getName());  
    
  System.out.println("cloned object id:" + copyObj.getId());  
  System.out.println("cloned object name:" + copyObj.getPrototype().getName());  
    
 }

}

结果:  
original object id:test1  
original object name:original object  
cloned object id:testCopy  
cloned object name:changed object

实例四: 利用串行化来做深复制  
把对象写道流里的过程是串行化(Serilization)过程;把对象从流中读出来是并行化(Deserialization)过程. 写在流里的是对象的一个拷贝,然后再从流里读出来重建对象.  
public class PrototypeSe implements Serializable {

 private String name;

 public String getName() {  
  return name;  
 }

 public void setName(String name) {  
  this.name = name;  
 }

}

public class NewPrototypeSe implements Serializable {  
   
 private String id;  
   
 public String getId() {  
  return id;  
 }

 public void setId(String id) {  
  this.id = id;  
 }

 private PrototypeSe prototype;  
   
 public PrototypeSe getPrototype() {  
  return prototype;  
 }

 public void setPrototype(PrototypeSe prototype) {  
  this.prototype = prototype;  
 }  
   
 public Object deepClone(){  
  try {  
   ByteArrayOutputStream bo = new ByteArrayOutputStream();  
   ObjectOutputStream oo = new ObjectOutputStream(bo);     
   oo.writeObject(this);     
     
   ByteArrayInputStream bi = new ByteArrayInputStream(bo.toByteArray());  
   ObjectInputStream oi = new ObjectInputStream(bi);  
   return oi.readObject();   
  } catch (IOException | ClassNotFoundException e) {  
   // TODO Auto-generated catch block  
   e.printStackTrace();  
   return null;  
  }  
 }

}

public class TestDeepClone {

 public static void main(String[] args) {  
  // TODO Auto-generated method stub  
  PrototypeSe po = new PrototypeSe();  
  po.setName("test1");  
  NewPrototypeSe se = new NewPrototypeSe();  
  se.setPrototype(po);  
    
  NewPrototypeSe deepClone = (NewPrototypeSe)se.deepClone();  
  deepClone.getPrototype().setName("test2");  
    
  System.out.println("original name:" + se.getPrototype().getName());  
  System.out.println("cloned name:" + deepClone.getPrototype().getName());

 }  
}  
结果:  
original name:test1  
cloned name:test2