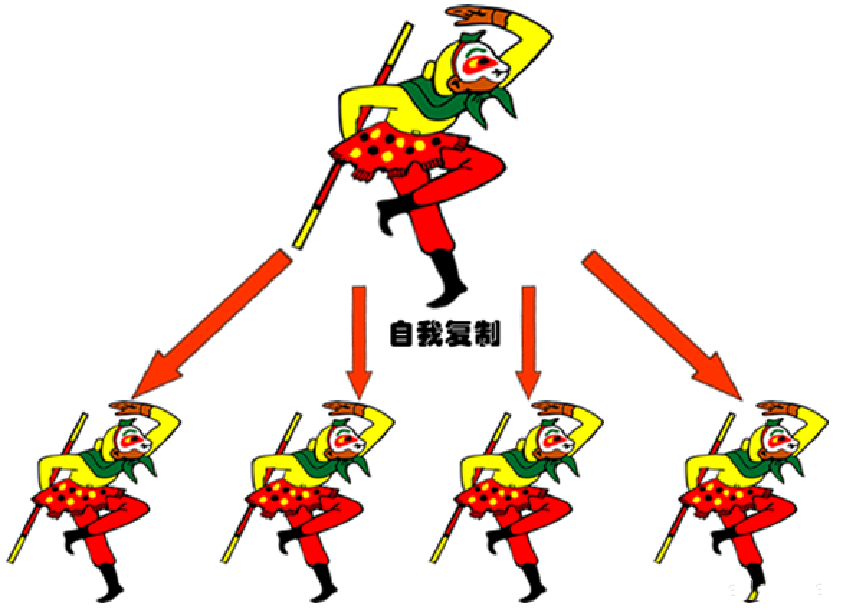
## **原型模式**

### **1、模式动机**



·在面向对象系统中，使用原型模式来复制一个对象自身，从而克隆出多个与原型对象一模一样的对象。在软件系统中，有些对象的创建过程较为复杂，而且有时候需要频繁创建，原型模式通过给出一个原型对象来指明所要创建的对象的类型，然后用复制这个原型对象的办法创建出更多同类型的对象，这就是原型模式的意图所在。

### **2、模式定义**

1.原型模式(Prototype Pattern)：原型模式是一种对象创建型模式，用原型实例指定创建对象的种类，并且通过复制这些原型创建新的对象。原型模式允许一个对象再创建另外一个可定制的对象，无须知道任何创建的细节。

2.原型模式的基本工作原理是通过将一个原型对象传给那个要发动创建的对象，这个要发动创建的对象通过请求原型对象拷贝原型自己来实现创建过程。

3.原型模式包含如下角色：

Prototype：抽象原型类

ConcretePrototype：具体原型类

Client：客户类

### **3、模式分析**

1.在原型模式结构中定义了一个抽象原型类，所有的Java类都继承自java.lang.Object，而Object类提供一个clone()方法，可以将一个Java对象复制一份。因此在Java中可以直接使用Object提供的clone()方法来实现对象的克隆，Java语言中的原型模式实现很简单。

2.能够实现克隆的Java类必须实现一个标识接口Cloneable，表示这个Java类支持复制。如果一个类没有实现这个接口但是调用了clone()方法，Java编译器将抛出一个CloneNotSupportedException异常。

3.通常情况下，一个类包含一些成员对象，在使用原型模式克隆对象时，根据其成员对象是否也克隆，原型模式可以分为两种形式：深克隆和浅克隆。

### C:\Users\29404\Documents\微信图片_20200610172722.jpg微信图片_20200610172722

Java语言提供的clone()方法将对象复制了一份并返回给调用者。一般而言，clone()方法满足：

(1) 对任何的对象x，都有x.clone() !=x，即克隆对象与原对象不是同一个对象。

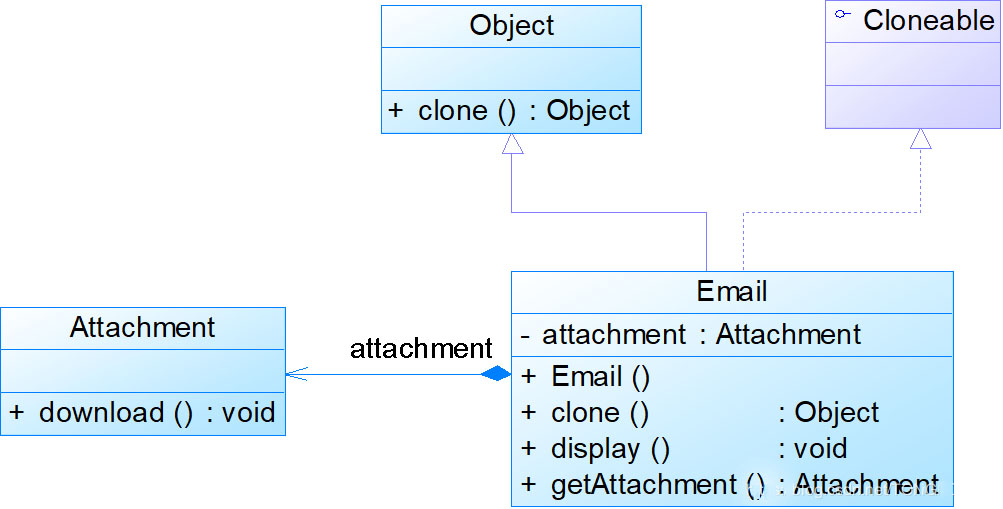
(2) 对任何的对象x，都有x.clone().getClass()==x.getClass()，即克隆对象与原对象的类型一样。

(3) 如果对象x的equals()方法定义恰当，那么x.clone().equals(x)应该成立。

### **4、模式案例与分析**

实例一：邮件复制（浅克隆）

由于邮件对象包含的内容较多（如发送者、接收者、标题、内容、日期、附件等），某系统中现需要提供一个邮件复制功能，对于已经创建好的邮件对象，可以通过复制的方式创建一个新的邮件对象，如果需要改变某部分内容，无须修改原始的邮件对象，只需要修改复制后得到的邮件对象即可。使用原型模式设计该系统。在本实例中使用浅克隆实现邮件复制，即复制邮件(Email)的同时不复制附件(Attachment)。



package 原型模式;

//抽象原型类

public class AttachMent {

public void download(){

System.out.println("下载邮件");

}

}

package 原型模式;

//具体原型类

public class Email implements Cloneable {

private AttachMent attachMent =null;

public Email(){

this.attachMent = new AttachMent();

}

public Object clone(){

Email clone =null;

try {

clone = (Email) super.clone();

} catch (CloneNotSupportedException e) {

System.out.println("Clone failture");

}

return clone;

}

public AttachMent getAttachMent(){

return this.attachMent;

}

public void display(){

System.out.println("查看邮件");

}

}

package 原型模式;

//客户类

public class 邮件复制\_浅克隆 {

public static void main(String[] args) {

Email email,copyEmail;

email = new Email();

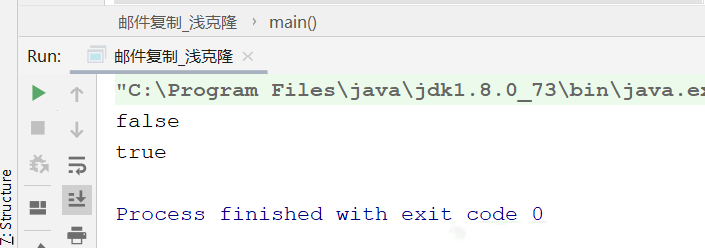
copyEmail = (Email) email.clone();

System.out.println(copyEmail==email);

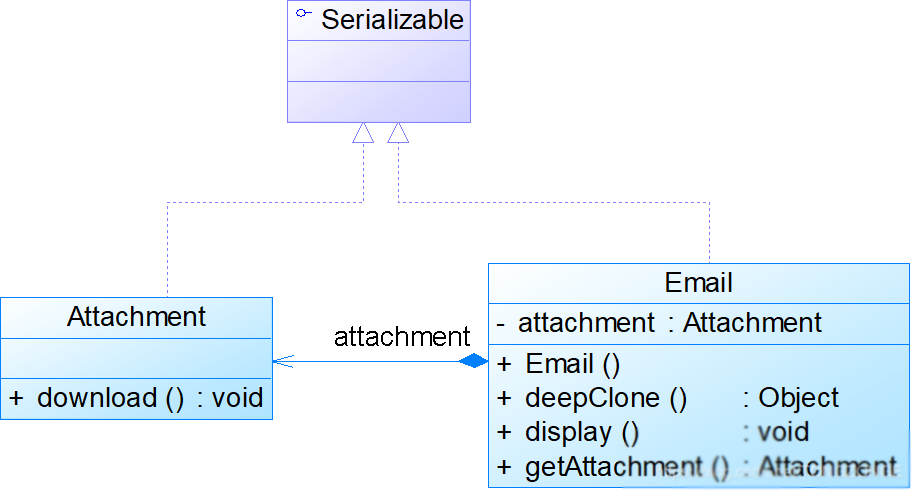
System.out.println(email.getAttachMent()==copyEmail.getAttachMent());

}

}



实例二：邮件复制（深克隆）  
使用深克隆实现邮件复制，即复制邮件的同时复制附件。



package 原型模式2;

import java.io.Serializable;

//抽象原型类

public class AttachMent implements Serializable {

public void download(){

System.out.println("下载附件");

}

}

package 原型模式2;

//具体原型类

import java.io.\*;

public class Email implements Serializable

{

private AttachMent attachMent =null;

public Email(){

this.attachMent = new AttachMent();

}

public Object deepclone() throws Exception {

// 将对象写入流中

ByteArrayOutputStream out = new ByteArrayOutputStream();

ObjectOutputStream ob = new ObjectOutputStream(out);

ob.writeObject(this);

//将对象从流中取出

ByteArrayInputStream in = new ByteArrayInputStream(out.toByteArray());

ObjectInputStream ois = new ObjectInputStream(in);

return (ois.readObject());

}

public AttachMent getAttachMent(){

return this.attachMent;

}

public void display(){

System.out.println("查看邮件");

}

}

package 原型模式2;

//客户端类

public class 邮件复制\_深克隆 {

public static void main(String[] args) throws Exception {

Email email,copyEmail;

email = new Email();

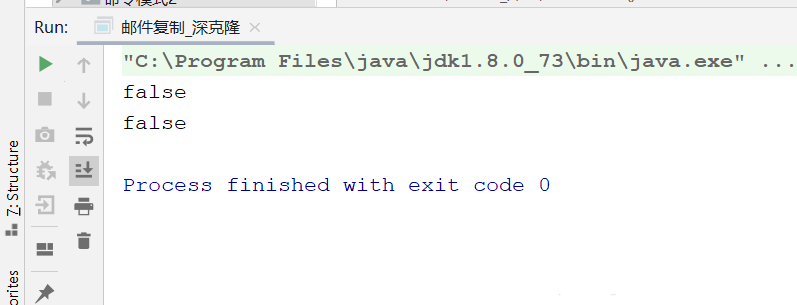
copyEmail = (Email) email.deepclone();

System.out.println(copyEmail==email);

System.out.println(email.getAttachMent()==copyEmail.getAttachMent());

}

}



* 相似对象的复制

package 原型模式3;

public class Student implements Cloneable {

private String name;

private String sex;

private int age;

private String Major;

private String college;

private String University;

public Student(String name, String sex, int age, String major, String college, String university) {

this.name = name;

this.sex = sex;

this.age = age;

Major = major;

this.college = college;

University = university;

}

public Student clone() throws CloneNotSupportedException {

Student cpStudent =null;

cpStudent = (Student) super.clone();

return cpStudent;

}

public String getName() {

return name;

}

public void setName(String name) {

this.name = name;

}

public String getSex() {

return sex;

}

public void setSex(String sex) {

this.sex = sex;

}

public int getAge() {

return age;

}

public void setAge(int age) {

this.age = age;

}

public String getMajor() {

return Major;

}

public void setMajor(String major) {

Major = major;

}

public String getCollege() {

return college;

}

public void setCollege(String college) {

this.college = college;

}

public String getUniversity() {

return University;

}

public void setUniversity(String university) {

University = university;

}

}

package 原型模式3;

public class 模式扩展 {

public static void main(String[] args) throws CloneNotSupportedException {

Student stu1,stu2,stu3;

stu1 =new Student("张无忌","男",24,"软件工厂","软件学院","中南大学");

stu2 =stu1.clone();

stu2.setName("杨过");

stu3 = stu1.clone();

stu3.setName("小龙女");

stu3.setSex("女");

System.out.println(stu1.getName() + " " + stu1.getSex() +" "+ stu1.getAge()+" "+stu1.getMajor()+

stu1.getCollege() +" "+ stu1.getUniversity());

System.out.println(stu2.getName() + " " + stu2.getSex() +" "+ stu2.getAge()+" "+stu2.getMajor()+

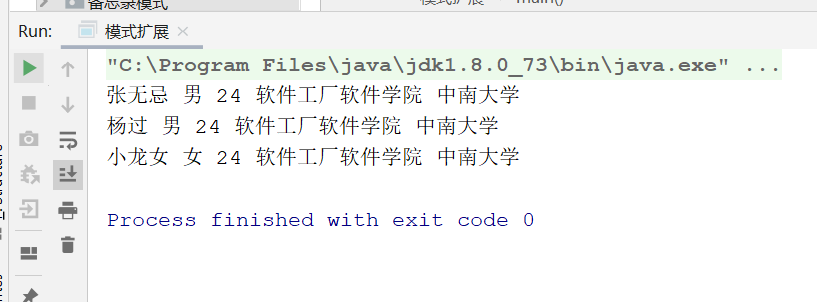
stu2.getCollege() +" "+ stu2.getUniversity());

System.out.println(stu3.getName() + " " + stu3.getSex() +" "+ stu3.getAge()+" "+stu3.getMajor()+

stu3.getCollege() +" "+ stu3.getUniversity());

}

}



5、模式的优缺点

优点

当创建新的对象实例较为复杂时，使用原型模式可以简化对象的创建过程，通过一个已有实例可以提高新实例的创建效率。

可以动态增加或减少产品类。

原型模式提供了简化的创建结构。

可以使用深克隆的方式保存对象的状态。

缺点

需要为每一个类配备一个克隆方法，而且这个克隆方法需要对类的功能进行通盘考虑，这对全新的类来说不是很难，但对已有的类进行改造时，不一定是件容易的事，必须修改其源代码，违背了“开闭原则”。

在实现深克隆时需要编写较为复杂的代码。

6、模式的适用环境

创建新对象成本较大，新的对象可以通过原型模式对已有对象进行复制来获得，如果是相似对象，则可以对其属性稍作修改。

如果系统要保存对象的状态，而对象的状态变化很小，或者对象本身占内存不大的时候，也可以使用原型模式配合备忘录模式来应用。相反，如果对象的状态变化很大，或者对象占用的内存很大，那么采用状态模式会比原型模式更好。

需要避免使用分层次的工厂类来创建分层次的对象，并且类的实例对象只有一个或很少的几个组合状态，通过复制原型对象得到新实例可能比使用构造函数创建一个新实例更加方便。

7、模式应用

原型模式应用于很多软件中，如果每次创建一个对象要花大量时间，原型模式是最好的解决方案。很多软件提供的复制(Ctrl + C)和粘贴(Ctrl + V)操作就是原型模式的应用，复制得到的对象与原型对象是两个类型相同但内存地址不同的对象，通过原型模式可以大大提高对象的创建效率。

### **8、总结**

1. 原型模式是一种对象创建型模式，用原型实例指定创建对象的种类，并且通过复制这些原型创建新的对象。原型模式允许一个对象再创建另外一个可定制的对象，无须知道任何创建的细节。原型模式的基本工作原理是通过将一个原型对象传给那个要发动创建的对象，这个要发动创建的对象通过请求原型对象拷贝原型自己来实现创建过程。

2. 原型模式包含三个角色：抽象原型类是定义具有克隆自己的方法的接口；具体原型类实现具体的克隆方法，在克隆方法中返回自己的一个克隆对象；让一个原型克隆自身从而创建一个新的对象，在客户类中只需要直接实例化或通过工厂方法等方式创建一个对象，再通过调用该对象的克隆方法复制得到多个相同的对象。在Java中可以直接使用Object提供的clone()方法来实现对象的克隆，能够实现克隆的Java类必须实现一个标识接口Cloneable，表示这个Java类支持复制。

1. 在浅克隆中，当对象被复制时它所包含的成员对象却没有被复制；在深克隆中，除了对象本身被复制外，对象包含的引用也被复制，也就是其中的成员对象也将复制。在Java语言中，通过覆盖Object类的clone()方法可以实现浅克隆；如果需要实现深克隆，可以通过序列化等方式来实现。

原型模式最大的优点在于可以快速创建很多相同或相似的对象，简化对象的创建过程，还可以保存对象的一些中间状态；其缺点在于需要为每一个类配备一个克隆方法，因此对已有类进行改造比较麻烦，需要修改其源代码，并且在实现深克隆时需要编写较为复杂的代码。

1. 原型模式适用情况包括：创建新对象成本较大，新的对象可以通过原型模式对已有对象进行复制来获得；系统要保存对象的状态，而对象的状态变化很小；需要避免使用分层次的工厂类来创建分层次的对象，并且类的实例对象只有一个或很少的几个组合状态，通过复制原型对象得到新实例可能比使用构造函数创建一个新实例更加方便。