# 原型模式简介

## 概述

1. 原型模式通过复制一个已存在的对象来获取更多相同或者相似的对象，通过复制一个原型对象得到与原型对象一模一样的新对象，这就是原型模式的动机。
2. 通过原型对象复制出来的对象是全新的对象，它们在内存中拥有新的地址，通常对克隆所产生的对象进行修改不会对原型对象造成任何影响，每个克隆对象都是相互独立的。

## 结构

原型对象的结构包含以下角色：



1. Prototype（抽象原型类）：它是声明克隆方法的接口，是原型对象所属类的父类，它可以是一个抽象类也可以是接口、或者是一个普通类。
2. ConcretePrototype（具体原型类）：原型对象所属类，也就是原型模式最终要创建的对象，它是抽象原型类的派生类，它实现或拥有了在抽象原型类中声明的克隆方法，在克隆方法中返回自己的一个克隆对象。
3. Client（客户类）：需要使用原型模式进行克隆获取对象的所在类。

# 实现

根据在复制原型对象的同时是否复包含在原型对象中引用类型的成员变量，原型模式的可控分为两种，即浅克隆（Shallow Clone）和深克隆（Deep Clone）。

1. 浅克隆：在浅克隆中，如果原型对象的成员变量是值类型（int、double、float、boolean、

char等基本数据类型），将其复制一份给克隆对象；如果原型对象的成员变量是引用类型（如类、接口、或数组），则将其引用地址复制一份给克隆对象，原型对象和克隆对象的成员变量指向相同的内存地址；在浅克隆中，原型对象和克隆对象两者始终不会指向相同的内存地址，只有两者的引用类型成员变量才会指向相同的内存地址。

1. 深克隆：在深克隆中，无论原型对象的成员变量时值类型还是引用类型，都将复制一份给克隆对象，原型对象和克隆对象的成员变量没有共用某个内存地址，原型对象和克隆对象两者始终不会指向相同的内存地址。

## 浅克隆

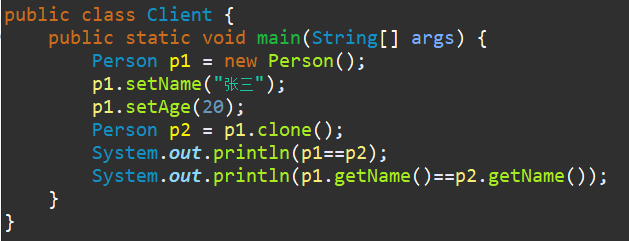
在Java语言中，所有的类默认继承了java.lang.Object类，Object类提供了一个clone( )方法，可以将一个Java对象复制一份，实现浅克隆；需要注意的是原型类必须实现一个标识接口Cloneable，表示这个类的实例（对象）可以被复制，如果没有实现这个接口而调用了clone( )方法，Java编译时将抛出CloneNotSupportedException异常。

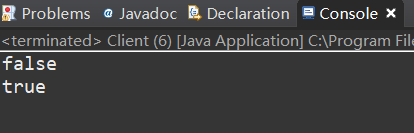
1. Person（原型类）：充当具体原型类



成员变量name是引用数据数据类型，age是基本数据类型；clone( )方法用于

1. 由于Java中所有的类都默认继承了Object类，所以Object类充当了抽象原型类的角色，无需再单独定义抽象类。
2. 编写客户端，测试原型模式的浅克隆



 运行结果如下：

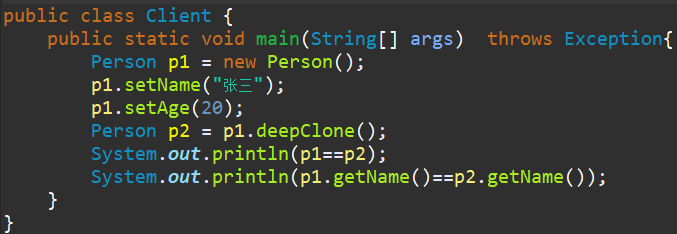
## 深克隆

在Java中可以通过序列化（Serialization）来实现深克隆。通过序列化实现的复制不仅可以复制对象本身，而且可以复制其引用类型的成员对象，因此通过序列化将对象写到一个流中，再从流里将其读出来，实现深克隆。需要注意的是原型类必须实现序列化接口Serializable，否则无法实现序列化操作。

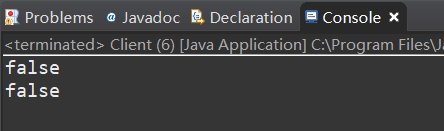
1）Person（原型类）：



1. 由于Java中所有的类都默认继承了Object类，所以Object类充当了抽象原型类的角色，无需再单独定义抽象类。
2. 编写客户端测试类，测试原型模式的深克隆



运行结果如下：



# 优缺点及适用环境

原型模式作为一种快速出创建大量相同或相似对象的方式，在软件开发中的应用较为广泛，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典型应用。

## 优点

原型模式的优点如下：

1. 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，通过复制一个已有实例可以提高实例的创建效率。
2. 扩展性较好，由于在原型模式中提供了抽象原型类，在客户端可以针对抽象原型类进行编程，而将具体原型类写在配置文件中，增加或减少产品类对原有系统没有任何影响，
3. 原型模式提供了简化的创建结构，工厂方法模式常常需要一个与产品类等级结构相同的工厂等级结构，而原型模式就不需要这样，原型模式中产品的复制是通过封装在原型类中的克隆方法实现的，无须专门的工厂类来创建产品。
4. 可以使用深克隆的方式保存对象的状态，使用原型模式将对象复制一份并将其状态保存起来，以便在需要的时候使用（例如恢复到某一历史状态），可辅助实现撤销操作。

## 缺点

原型模式的缺点主要如下：

1. 需要在原型类中定义克隆方法；
2. 在实现深克隆时需要编写较为复杂的代码，而且当对象之间存在多重的嵌套引用时，为了实现深克隆，每一层对象对应的类都必须支持深克隆。

## 适用环境

在以下情况下可以考虑适用原型模式：

1. 创建新对象成本较大（例如初始化需要占用较长的时间、占用太多的CPU资源或网络资源），此时可以通过复制已有对象来获得新对象，如果是相似对象，则可以对其成员变量稍作修改。
2. 系统要保存对象的状态，而对象的状态变化很小。
3. 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的实例可能比使用构造函数创建一个新实例更加方便。