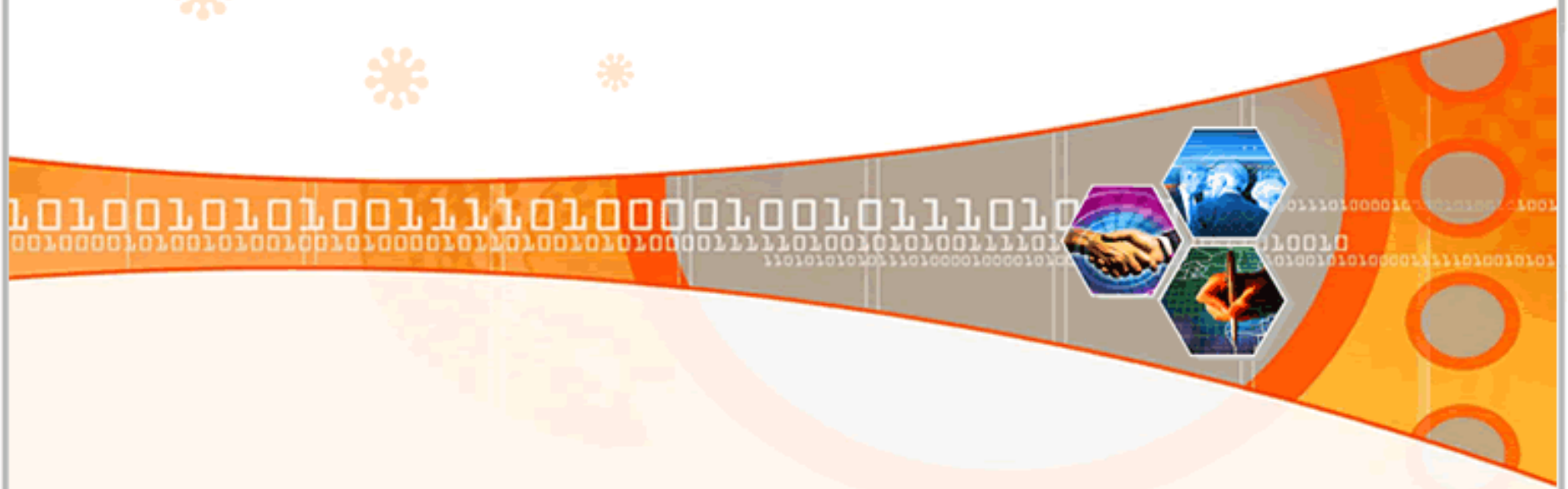
第8章

原型模式



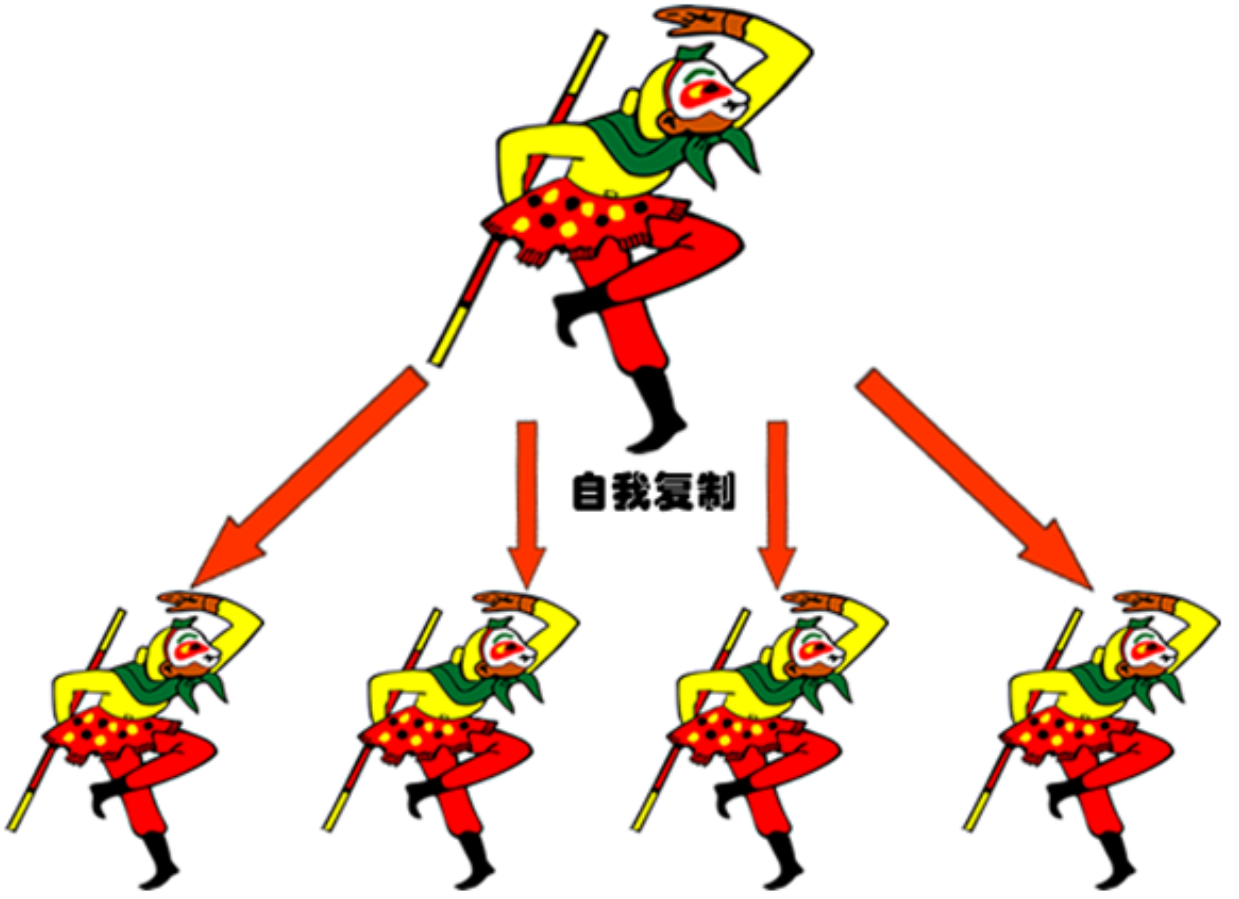
主讲教师：程细柱　韶关学院计算机系
本书主编：刘　伟　　清华大学出版社

本章教学内容

- 原型模式
  - ✓ 模式动机与定义
  - ✓ 模式结构与分析
  - ✓ 模式实例与解析
  - ✓ 模式效果与应用
  - ✓ 模式扩展

原型模式

- 模式动机



原型模式

- **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。

原型模式

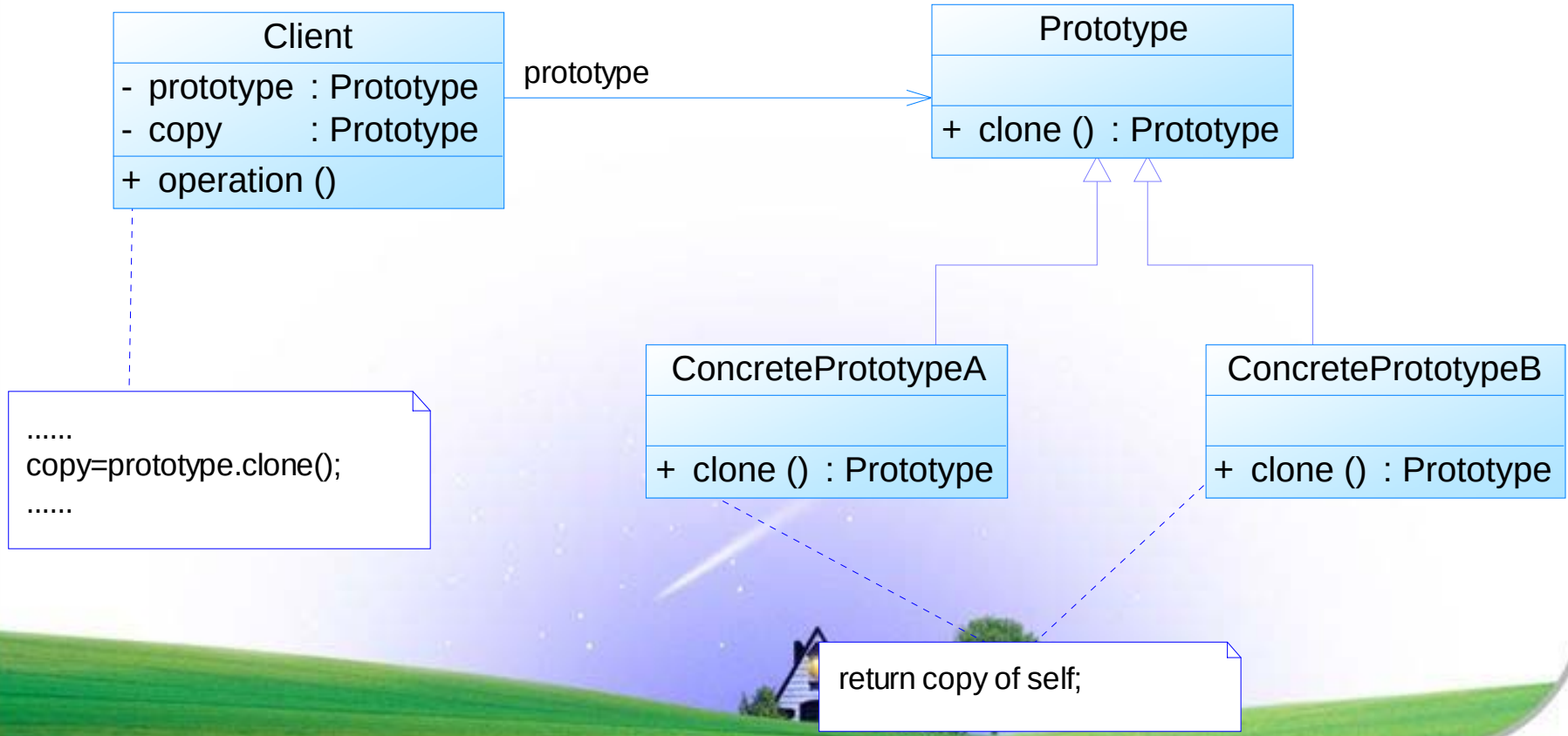
- 模式定义
  - ✓ 原型模式(Prototype Pattern)：原型模式是一种对象创建型模式，用原型实例指定创建对象的种类，并且通过复制这些原型创建新的对象。原型模式允许一个对象再创建另外一个可定制的对象，无须知道任何创建的细节。
  - ✓ 原型模式的基本工作原理是通过将一个原型对象传给那个要发动创建的对象，这个要发动创建的对象通过请求原型对象拷贝原型自己来实现创建过程。

原型模式

- 模式定义
  - ✓ **Prototype Pattern: Specify the kind of objects to create using a prototypical instance, and create new objects by copying this prototype.**
  - ✓ **Frequency of use: medium**

原型模式

- 模式结构


Client
- prototype : Prototype
- copy : Prototype
+ operation ()
prototype
Prototype
+ clone () : Prototype
ConcretePrototypeA
+ clone () : Prototype
ConcretePrototypeB
+ clone () : Prototype
.....
copy=prototype.clone();
.....
return copy of self;

原型模式

- **模式结构**
  - ✓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**

原型模式

- 模式分析
  - ✓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。
  - ✓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。

原型模式

- 模式分析
  - ✓ 示例代码:

```
public class PrototypeDemo implements Cloneable
{
    ……
        public Object clone()
        {
                Object object = null;
                try {
                        object = super.clone();
                } catch (CloneNotSupportedException exception) {
                        System.err.println("Not support cloneable");
                }
                return object;
    }
    ……
}
```

原型模式

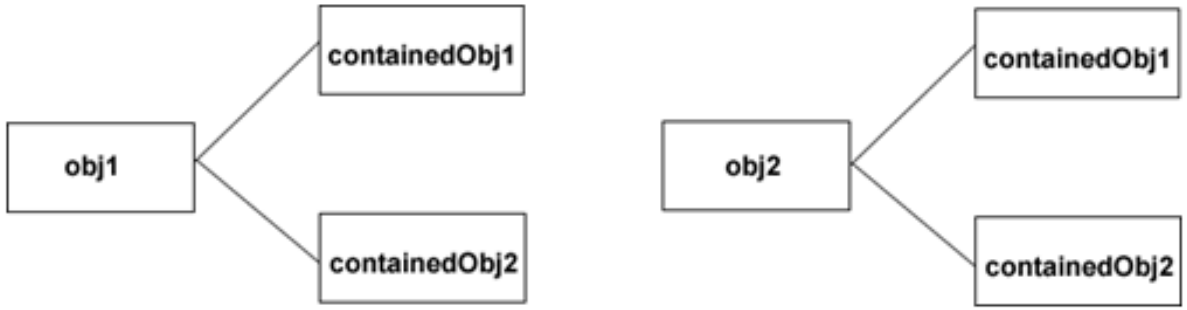
- 模式分析
  - ✓通常情况下，一个类包含一些成员对象，在使用原型模式克隆对象时，根据其成员对象是否也克隆，原型模式可以分为两种形式：深克隆和浅克隆。

原型模式

- 模式分析
  - ✓浅克隆：仅仅复制所考虑的对象，不复制它所引用的成员对象。
  - ✓深克隆：两者都复制。

浅克隆

深克隆



原型模式

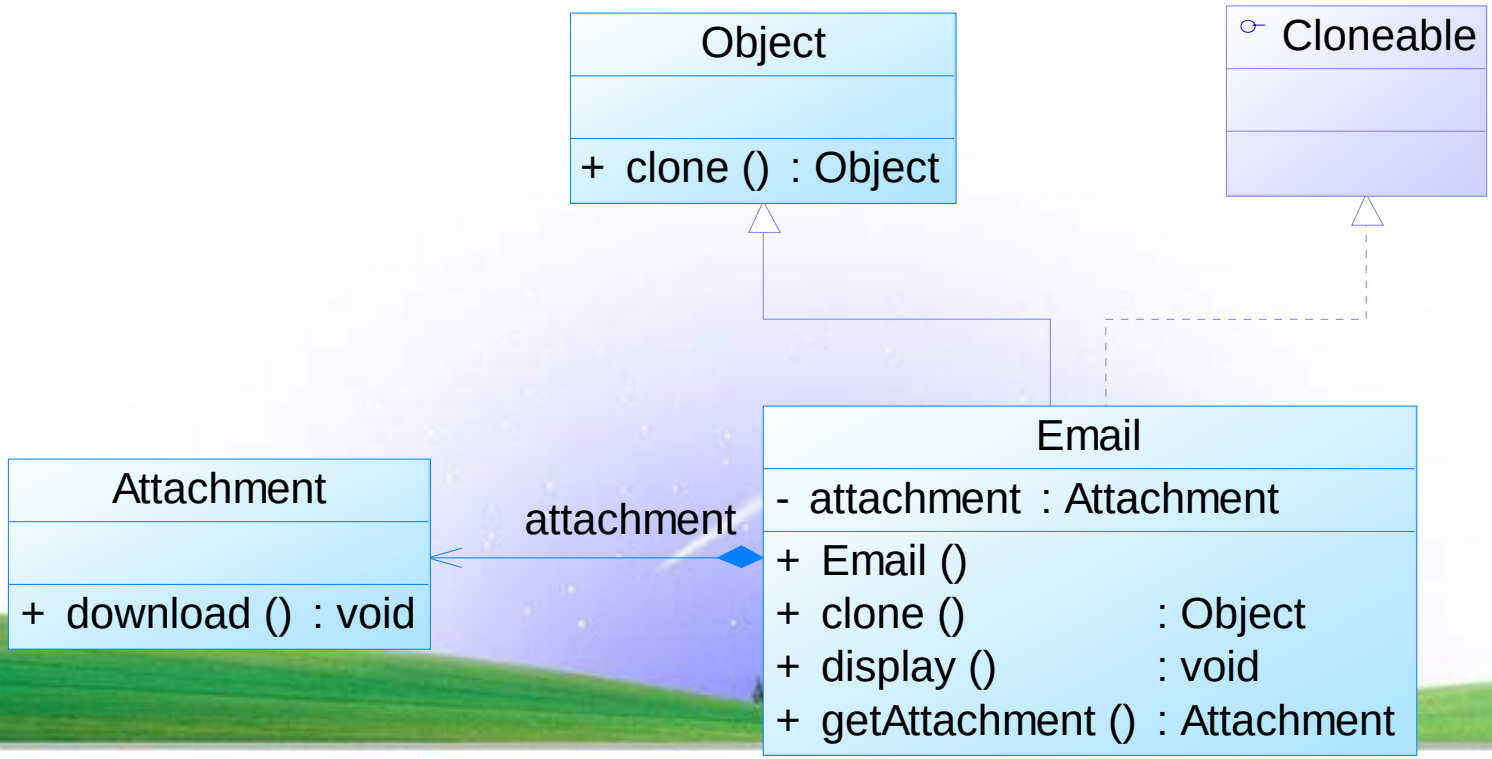
- **模式分析**
  - ✓ Java语言提供的clone()方法将对象复制了一份并返回给调用者。一般而言，clone()方法满足:
    - (1) 对任何的对象x，都有x.clone() !=x，即克隆对象与原对象不是同一个对象。
    - (2) 对任何的对象x，都有x.clone().getClass()==x.getClass()，即克隆对象与原对象的类型一样。
    - (3) 如果对象x的equals()方法定义恰当，那么x.clone().equals(x)应该成立。

原型模式

- **原型模式实例与解析**
  - ✓ 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)。

原型模式

- **原型模式实例与解析**
  - ✓ 实例一：邮件复制（浅克隆）： 实现接口**Cloneable**



原型模式

- **原型模式实例与解析**
  - ✓ 实例一：邮件复制（浅克隆）
    - 参考代码 (Chapter 08 Prototype\sample01)

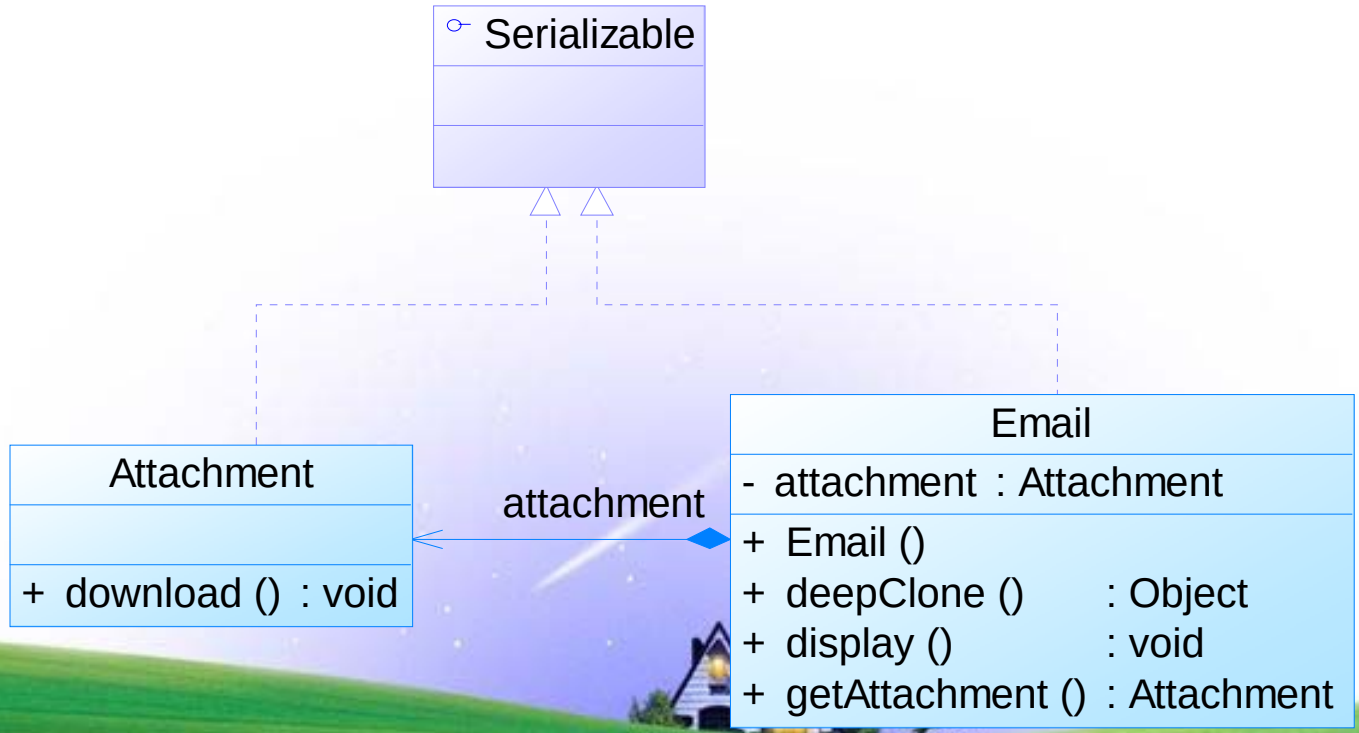
演示.....

原型模式

- **原型模式实例与解析**
  - ✓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。

原型模式

- **原型模式实例与解析**
  - ✓ 实例二：邮件复制（深克隆）：通过序列化实现。



原型模式

- **原型模式实例与解析**
  - ✓ 实例二：邮件复制（深克隆）
    - 参考代码 (Chapter 08 Prototype\sample02)

演示.....

原型模式

- **模式优缺点**
  - ✓ 原型模式的优点
    - 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。

原型模式

- **模式优缺点**
  - ✓原型模式的缺点
    - 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。

原型模式

模式适用环境

✓ 在以下情况下可以使用原型模式：

- 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。

原型模式

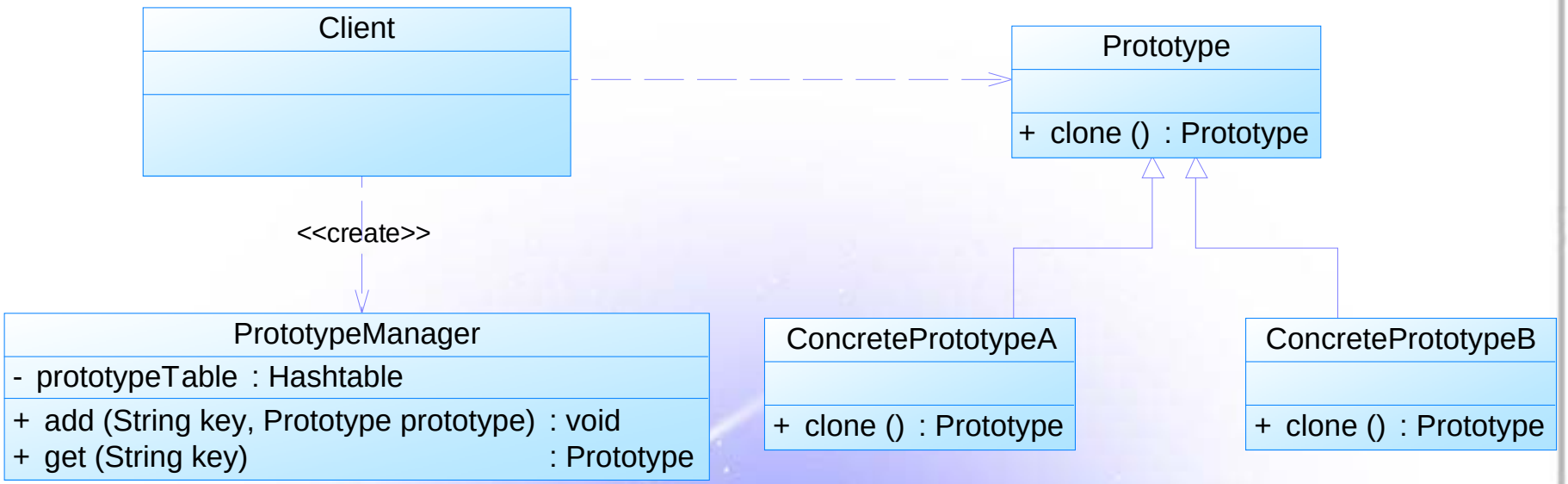
- 模式应用
  - ✓ (1) 原型模式应用于很多软件中，如果每次创建一个对象要花大量时间，原型模式是最好的解决方案。很多软件提供的复制(Ctrl + C)和粘贴(Ctrl + V)操作就是原型模式的应用，复制得到的对象与原型对象是两个类型相同但内存地址不同的对象，通过原型模式可以大大提高对象的创建效率。

原型模式

- 模式应用
  - ✓ (2) 在Struts2中为了保证线程的安全性，Action对象的创建使用了原型模式，访问一个已经存在的Action对象时将通过克隆的方式创建出一个新的对象，从而保证其中定义的变量无须进行加锁实现同步，每一个Action中都有自己的成员变量，避免Struts1因使用单例模式而导致的并发和同步问题。
  - ✓ (3) 在Spring中，用户也可以采用原型模式来创建新的bean实例，从而实现每次获取的是通过克隆生成的新实例，对其进行修改时对原有实例对象不造成任何影响。

原型模式

- 模式扩展
  - ✓带原型管理器的原型模式：管理器中定义了一个集合用于存储原型对象。



原型模式

- **模式扩展**
  - ✓ 带原型管理器的原型模式
    - 参考代码 (Chapter 08 Prototype\原型管理器)

演示.....

原型模式

- **模式扩展**
  - ✓相似对象的复制
    - 很多情况下，复制所得到的对象与原型对象并不是完全相同的，它们的某些属性值存在异同。通过原型模式获得相同对象后可以再对其属性进行修改，从而获取所需对象。如多个学生对象的信息的区别在于性别、姓名和年龄，而专业、学院、学校等信息都相同，为了简化创建过程，可以通过原型模式来实现相似对象的复制。

原型模式

- **模式扩展**
  - ✓相似对象的复制
    - 参考代码 (Chapter 08 Prototype\相似对象的复制)

演示.....

本章小结

- 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。
- 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。

本章小结

- 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。
- 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。

本章小结

- 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。
- 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。

END

Thanks!