4、The Singleton Pattern
（单例模式）

组员：汤仲喆　王凯　李义冬
主讲：汤仲喆
代码：王凯
答辩：李义冬

SINGLETON—俺有6个漂亮的老婆，她们的老公都是我，我就是我们家里的老公Sigleton，她们只要说道"老公"，都是指的同一个人，那就是我(刚才做了个梦啦，哪有这么好的事)
单例模式：单例模式确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例单例模式。单例模式只应在有真正的"单一实例"的需求时才可使用。

解决方案---单例模式

- 什么是单例模式

  顾名思义，单例模式的意思就是只有一个实例。单例模式确保某一个类只有一个实例，而且自行实例化并向整个系统提供这个实例。这个类称为单例类。

- 单例模式的要点

  （1）某个类只能有一个实例；

  （2）它必须自行创建这个实例；

  （3）它必须自行向整个系统提供这个实例。

单例模式的关键特征

- 意图：希望类只有一个实例，但没有控制类实例化的全局变量（对象）。同时希望确保所有客体对象使用该类的相同实例，而无需将引用传给它们。

- 问题：几个不同的客户对象需要引用同一个对象，而且希望确保这种类型的对象数目不超过一个。

- 解决方案：保证一个实例
- 参与者与协作者：客户对象只能通过getInstance()方法创建单例类的实例。
- 效果：客户对象无需操心是否存在单例类的实例，实例化有单例类自己控制。
- 实现：
  - 一个引用单例对象的静态私有成员变量
  - 一个公共静态方法，负责实现一次性的实例化并返回对单例对象的引用
  - 设置为保护或私有的构造方法

单例模式设计会产生什么问题？

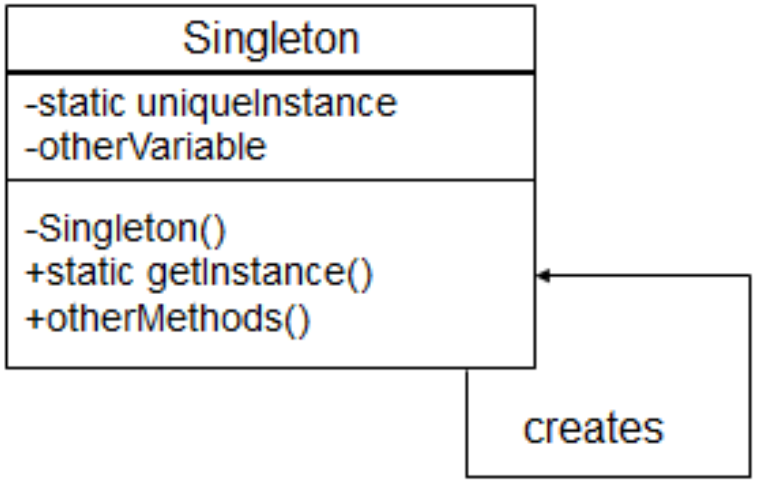
- 在多线程程序中，Singleton模式可能会出现一个问题。

- 假设对getInstance()方法的两个调用几乎同时发生，这种情况可能非常糟糕。此时会发生什么？

  1.第一个线程检查实例是否存在。因为实例不存在，该线程执行创建第一个实例的代码部分。

  2.然而，假设在实例化完成之前，另一个线程也来检查实例成员变量是否为null。因为第一个线程还什么都没有创建，实例成员变量仍然等于null，所以第二个线程也执行了创建一个对象的代码。

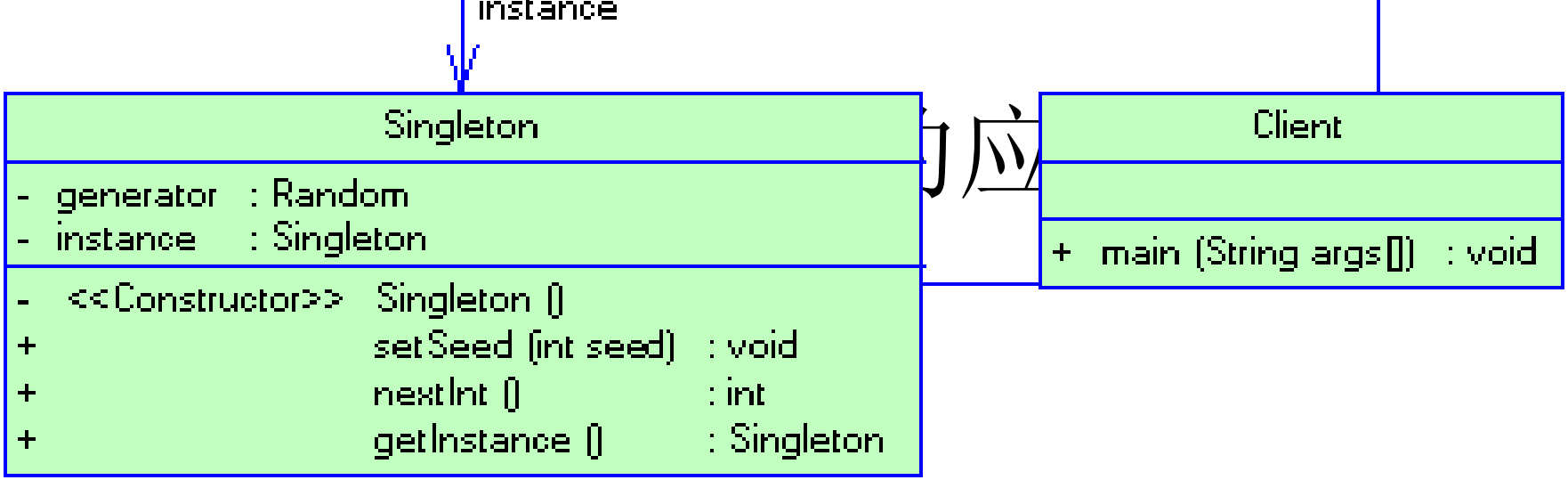
  3.现在，两个线程都执行了Singleton对象的new操作，因此创建了两个重复的对象。



懒汉式 vs 饿汉式

- 饿汉式：静态初始化方式，在启动加载单例类时就实例化对象，只实例化一次，以后用到的时候就不需要再去实例化了，加载类的时候速度比较慢，但以后获得对象时的速度比较快，该对象从加载到应用结束一直占用资源。
- 懒汉式：相当于一个延迟加载机制，即你需要这个对象时候才去实例化，加载类的时候速度比较快，但以后获得对象时的速度比较慢，该对象在整个应用的生命周期只有一部分时间占用资源。面临多线程访问的安全性问题，需要做双重锁定处理才可以保证安全。

所以，到底使用哪一种方式，要看实际的需求。



在整个应用程序中只需要一个类的实例来产生随机数，客户端程序从类中获取这个实例，调用这个实例的方法nextInt()，公用的方法访问需要进行同步，这是单例模式需要解决的同步问题。

参与者：Singleton定义一个Instance操作，允许客户访问它的唯一实例，Instance是一个类操作，负责创建自己的唯一实例。

协作关系：客户只能通过Singleton的Instance操作访问一个Singleton的实例。

第十七章　单件模式

单件模式

保证一个类仅有一个实例，并提供一个访问它的全局访问点。

Prototype Pattern

Ensure a class only has one instance, and provide a global point of access to it.

一、概述

单件模式是关于怎样设计一个类，并使得该类只有一个实例的成熟模式，该模式的关键是将类的构造方法设置为private权限，并提供一个返回它的唯一实例的类方法。

二、单件模式的结构与使用

模式的结构中只包括一个角色：

- 单件类（Singleton）

模式的UML类图

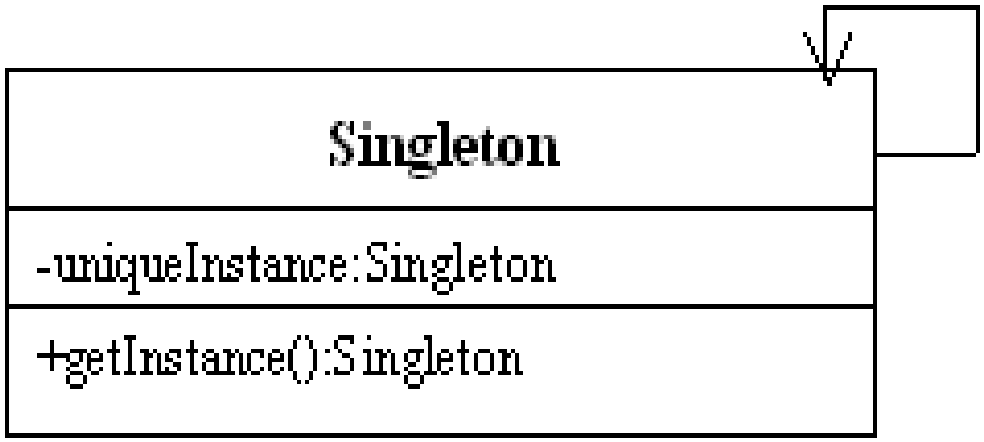


图 17.2 单件模式的类图

模式的结构的描述与使用

1. 单件类（Singleton）：Moon.java

```
public class Moon{
      private static Moon  uniqueMoon;
      double radius;
      double distanceToEarth;
      private Moon(){
            uniqueMoon=this;
            radius=1738;
            distanceToEarth=363300;
      }
      public static synchronized Moon getMoon(){
             if(uniqueMoon==null){
                   uniqueMoon=new Moon();
             }
             return uniqueMoon;
      }
      public String show(){
            String s="月亮的半径是"+radius+"km,距地球是"+distanceToEarth+"km";
            return s;
      }
}
```

模式的结构的描述与使用

2. 应用 Application.java

```
import javax.swing.*;
import java.awt.*;
public class Application{
    public static void main(String args[]){
          MyFrame f1=new MyFrame("张三看月亮");
          MyFrame f2=new MyFrame( "李四看月亮");
          f1.setBounds(10,10,360,150);
          f2.setBounds(370,10,360,150);
          f1.validate();
          f2.validate();
    }
}
class  MyFrame extends JFrame{
    String str;
    MyFrame(String title){
        setTitle(title);
        Moon moon=Moon.getMoon();
        str=moon.show();
        setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        setVisible(true);
        repaint();
   }
    public void paint(Graphics g){
          super.paint(g);
          g.setFont(new Font("宋体",Font.BOLD,14));
          g.drawString(str,5,100);
    }
}
```

三、单件模式的优点

- 单件类的唯一实例由单件类本身来控制，所以可以很好地控制用户何时访问它。