3

Java 的高级特性

【本章导读】

Java 编程语言之所以受到广大开发人员的追捧，并经久不衰，除了具有面向对象编程语言的基本功能外，还具有如多线程、丰富的流功能、抽象类、接口等高级功能。本章在上一章的 Java 基础知识之上更深入一步，讲述 Java 的高级特性。

本章开篇小节讲述 Java 中的继承，包括单继承的实现、访问控制、方法重载、方法覆盖、多态实现和隐藏技术等；接着讲述 Java 中的线程技术，包括线程的概念、如何创建线程、线程的状态和调度、线程的中断和恢复，以及线程同步等知识；接着讲述高级 I/O 流，让读者使用 Java 中的流时能得心应手；最后详细讲述更多 Java 的高级特性，包括 this、super、static 和 final 等关键字的使用，以及内部类、抽象类和接口等高级知识。

通过本章，不但可以使读者重温 Java 中的一些高级知识的理论，而且可以使读者更熟练地应对 Java 高级知识中各种各样的面试题。

3.1 Java 的继承

3.1.1 继承

Java 继承是使用已存在的类的定义作为基础建立新类的技术，新类的定义可以增加新的数据或新的功能，也可以用父类的功能，但不能选择性地继承父类。这种技术使得复用以前的代码非常容易，能够大大缩短开发周期，降低开发费用。

Java 不支持多重继承，单继承使 Java 的继承关系很简单，一个类只能有一个父类，易于管理程序，同时一个类可以实现多个接口，从而克服单继承的缺点。将被继承的类称为父类（基类），继承类称为子类（派生类）。在 Java 中用关键字 extends 来实现单继承，语法如下：

```
class subclass extends superclass{...}
```

实现继承关系的类之间有着必然的联系，不能将不相关的类实现继承。当类 A 和类 B 之间有着共同的属性和行为时，那么类 A 和类 B 之间就可能是继承关系或者有着共同的父类。

在面向对象程序设计中运用继承原则，就是在由一般类和特殊类形成的“一般—特殊结构”中，

把一般类和特殊类的对象实例所共同具有的属性和操作在一般类中进行显式地定义，在特殊类中不再重复地定义这些东西，但是在语义上，特殊类却自动地、隐含地拥有它的一般类（以及所有更上层的一般类）中定义的属性和操作。特殊类的对象拥有一般类的全部或部分属性与方法，称为特殊类对一般类的继承。

3.1.2 访问控制

在 Java 中，通过各种访问区分符来实现数据封装，共分为四种访问级别（由高到低）：

（1）private（私有）：被 private 修饰的所有变量和方法只能在所属类中被访问。即类的私有成员和变量只能在当前类中被访问。

（2）friendly（默认）：当类的变量和方法没有显式地被任何访问区分符修饰时，该变量和方法的访问级别是默认的。默认的变量和方法只能在同包的类中访问。

（3）protected（受保护）：当类的变量或方法被 protected 修饰时，只可以在同包中的任何类、不同包中的任何当前类的子类中所访问，不同包中的不是该类的子类不可访问。

（4）public（公共）：当变量或方法被 public 修饰时，该变量和方法可以在任何地方（指的是任何包中）的任何类中被访问。

以上四种访问修饰符可以作用于任何变量和方法，类只可以定义为默认或公共级别（嵌套类除外）。

3.1.3 方法重载

在 Java 中，每一个方法都有自己的特征，其特征主要是指方法名和方法的参数。当参数相同但方法名不同或者方法名相同但参数不同时，都认为这两个方法的特征不一样。对于 Java 编译器来说，它只依据方法的名称、参数列表的不同来判断两个方法是否相同，如果出现两个名称相同、参数也完全一致的方法，那么编译器就认为这两个方法是完全一样的，也就是说方法被重复定义，会提示编译错误。

需要注意的是，方法的重载都是基于同一个类的。

当一个类中的多个同名方法满足以下条件之一时，即实现了方法重载：

（1）不同的参数个数。

（2）不同的参数类型。

（3）不同的参数顺序。

如果有一个类带有几个构造函数，那么也许会想复制其中一个构造函数的某些功能到另一个构造函数中，可以通过使用关键字 this 作为一个方法调用来达到这个目的。参考代码如下：

```
public class Employee {
    private String name;
    private int salary;

    public Employee(String n, int s) {
        name = n;
        salary = s;
    }
```

```
    public Employee(String n) {
        this(n, 0);
    }

    public Employee() {
        this(" Unknown ");
    }
}
```

对于 this 的任何调用，如果出现，在任何构造函数中必须是第一个语句，否则在编译时会出现“Constructor call must be first statement in a constructor”错误。另外，构造函数中调用另一构造函数，其调用（this()、super()）有且只能有一次，并不能同时出现调用。

3.1.4 方法覆盖

覆盖是基于继承的，没有继承就没有覆盖。在 Java 中，覆盖的实现是在子类中对从父类中继承过来的非私有方法的内容进行修改或扩展的一个动作。不能违反访问级别的限制，即子类方法的访问级别不能低于父类方法的访问级别。

实现方法的覆盖必须满足以下所有条件：

（1）覆盖方法的返回类型必须与父类中被覆盖方法的返回类型相同。

（2）覆盖方法的参数列表类型、次序和方法名称必须与被覆盖方法的参数列表类型、次序和方法名称相同，否则会被认为是子类中自己新定义的方法。

（3）覆盖方法的访问级别不能比被覆盖方法的访问级别低。

（4）覆盖方法不能比它所覆盖的方法抛出更多的异常。

3.1.5 多态实现

一个名字可以表示许多不同类（这些不同类必须拥有一个共同的超类）的对象，从而实现以不同的方式来响应某个共同的操作集。

Java 中的多态就体现在一个变量可以引用多个不同类对象，前提是这些不同类必须有着共同的父类，从而该变量可以且只能调用每个不同对象之间的公共操作集（方法）。多态的实现是基于继承的。

在 Java 中，父类变量可以引用子类对象，即 Employee e = new Manager()是合法的，但是变量 e 只能调用共同的成员属性与方法，即 e 只能访问子类从父类中继承过来的成员。父类引用可以引用子类对象，同时该父类引用只能访问所有子类的公有操作集（从父类继承过来的成员）；当子类中已覆盖继承方法时，父类变量调用的将是子类中已覆盖的方法。

可以创建具有共同类的对象的收集（如数组），这种收集被称为同类收集。有了 Java 的这种多态机制，因而可以实现异类收集（不相同的对象的收集）。

Java 拥有一个顶层父类 java.lang.Object，该类是所有类的顶级父类，在平常定义各种类时，虚拟机会自动在类的声明语句后加上继承 Object 类，如 class User 与 class User extends Object 是等同的。

有了异类收集，Object 数组就能收集所有种类的元素，例如如下收集了三种不同类型的对象：

```
Object[] obj = {"123", new Employee(), new Manager()};
```

3.1.6 隐藏技术

类的私有方法对于子类来说是不可见的，但是不可见不等于没有，子类仍旧继承了父类所有的成员。实际上，这些私有方法都被隐藏，对子类来说，这些父类的私有成员都被隐藏了起来，从而导致在子类中的不可见。

3.1.7 面试题

1．问：Java 继承具有哪些特征？

答案：Java 继承具有以下特征：

（1）继承关系是传递的。若类 C 继承类 B，类 B 继承类 A，则类 C 既有从类 B 那里继承下来的属性和方法，也有从类 A 那里继承下来的属性和方法，还可以有自己新定义的属性和方法。继承来的属性和方法尽管是隐式的，但仍是类 C 的属性和方法。

（2）继承简化了人们对事物的认识和描述，能清晰体现相关类间的层次结构关系。

（3）继承提供了软件复用功能。若类 B 继承类 A，那么建立类 B 时只需要再描述与基类 A 不同的少量特征。这种做法能减少代码和降低数据的冗余度，大大增加程序的重用性。

（4）继承通过增强一致性来减少模块间的接口和界面，大大增加了程序的易维护性。

（5）提供多重继承机制。Java 出于安全性和可靠性的考虑，仅支持单重继承，而通过使用接口机制来实现多重继承。

2．问：请举例说明日常生活中有哪些常见的关系可以使用继承来实现？

答案：例如公司的经理类（Manager）与职员类（Employee）之间的关系，Manager 具有职员的基本特征，但是又具有一些一般职员没有的权利或属性。

另外父亲与儿子、鱼类与草鱼类之间的关系等都可以使用继承关系来体现。

3．问：在下面的代码中有一个 Dog 的基类：

```
package com.itjob.javaadvanced;

public class Dog {
    private String getName(){
        return "Dog";
    }

    public String bark(){
        return "wang-wang";
    }

    public void call(){
        System.out.println("It's " + getName() + " " + bark());
    }
}
```

MiniDog 类继承 Dog 类，该类的代码如下：

```
package com.itjob.javaadvanced;

public class MiniDog extends Dog {
```

```
    private String getName(){
        return "Mini";
    }

    public String bark(){
        return "WOO";
    }
}
```

测试类代码如下，请回答打印出的结果信息，并给出原因。

答案：打印的结果如下：

```
It's Dog wang-wang
It's Dog WOO
It's Dog WOO
```

第一句打印结果没有异议，第一句为什么 getName(…)打印的是 Dog 呢？原因在于 getName 是私有方法，而 MiniDog 类又没有实现 call(…)方法，所以打印出来的是父类的 getName 的值。

4．问：请问方法重载和方法重写（方法覆盖）分别表示什么？两者的区别是什么？分别使用什么英文单词表示？

答案：方法重载：使用英文单词 Overloading 表示。

在一个类中创建多个方法，它们具有相同的名字，但具有不同的参数和不同的定义。调用方法时通过传递给不同的参数个数和参数类型来决定具体使用哪个方法。重载的时候，方法名要一样，但是参数类型和个数可以不一样，返回值类型可以相同也可以不相同，无法以返回类型作为重载函数的区分标准。

方法重写：使用英文单词 Overriding 表示。

父类与子类之间的多态性，子类对父类的函数进行重新定义。如果在子类中定义某方法与其父类有相同的名称和参数，称该方法被重写。在 Java 中，子类可继承父类中的方法，而不需要重新编写相同的方法。但有时子类并不想原封不动地继承父类的方法，而是想作一定的修改，这就需要采用方法重写。方法重写又称为方法覆盖。

若子类中的方法与父类中的某一方法具有相同的方法名、返回类型和参数表，则新方法将覆盖原有的方法。如果需要父类中原有的方法，可使用 super 关键字。

另外，子类函数的访问修饰权限不能少于父类的。

5．问：什么是 Java 的多态？如何实现多态？请举例说明多态的使用。

答案：简单来说，多态是具有表现多种形态的能力的特征。

实现多态的步骤如下：

（1）子类重写父类的方法。

（2）编写方法时，重写父类定义的方法。

（3）运行时，根据实际创建的对象类型动态决定使用哪个方法。

例如，Manager 类和 Engineer 类都是 Employee 的子类，并且都重写了 getEmpDetails 方法，如下展示了 Java 中使用继承实现多态性的方法：

```
Employee e = new Manager();
System.out.println(e.getEmpDetails());
e = new Engineer();
System.out.println(e.getEmpDetails());
```

6．问：在 Java 中提供了进行访问控制的机制，方法具有不同的作用域，方法的作用域分为哪四种？若方法前不加作用域修饰符，表示的是什么作用域？四者对于当前类、同一 package、子孙类和其他 package 的访问权限是怎样的？

答案：Java 作用域分为 public、protected、friendly 和 private，不写表示是 friendly。

作用域与访问权限的对应关系如下：

作用域	当前类	同一 package	子孙类	其他 package
public	√	√	√	√
protected	√	√	√	×
friendly	√	√	×	×
private	√	×	×	×

7．问：如下代码是否会出现编译错误？如果有，原因是什么？

```
public class Something {
    void doSomething () {
        private String s = "";
        int l = s.length();
    }
}
```

答案：会出现编译错误。原因在于局部变量前不能放置任何访问修饰符（private、public 和 protected）。final 可以用来修饰局部变量，但它不是访问修饰符。

3.2 Java 的线程

3.2.1 线程的概念

线程是 Java 的一大特色，从语言上直接支持线程，线程对于进程来说，优势在于创建的代价很小、上下文切换迅速，当然其他的优势还有很多，但缺点也是有的，例如对于开发人员的要求比较高、不容易操作等。

一个线程（执行上下文）由三个主要部分组成：

（1）一个虚拟 CPU。

（2）CPU 执行的代码：代码可以由多个线程共享，它不依赖数据。如果两个线程执行同一个类的实例的代码时，则它们可以共享相同的代码。

（3）代码操作的数据：数据可以由多个线程共享，而不依赖代码。如果两个线程共享对一个公共对象的访问，则它们可以共享相同的数据。

3.2.2 创建线程

创建线程有两种方式：

（1）实现 Runnable 接口：从面向对象的角度来看，Thread 类是一个虚拟处理机的严格封装，因此只有当处理机模型修改或扩展时，才应该继承类。由于 Java 技术只允许单一继承，所以如果已经继承了 Thread，就不能再继承其他任何类了。

（2）继承 Thread 类：当一个 run()方法体出现在继承 Thread 的类中时，用 this 指向实际控制

运行的 Thread 实例。因此，代码不再需要使用如下控制：

```
Thread.currentThread().join();
```

而可以简单地用：

```
join();
```

3.2.3 线程的状态与调度

线程的调度是基于时间片基础上的优先级优先原则。抢占式调度模型（优先级优先）是指可能有多个线程是可运行的，但只有一个线程在实际运行。这个线程会一直运行，直至它不再是可运行的（运行时间到、时间片原则、被具有更高优先级的线程抢占、优先级优先原则）。

线程的代码可能执行了一个 Thread.sleep()调用，要求这个线程暂停一段固定的时间。这个线程可能在等待访问某个资源，而且在这个资源可访问之前这个线程无法继续运行。

所有可运行线程根据优先级保存在池中。当一个被阻塞的线程变成可运行时，它会被放回相应的可运行池。优先级最高的非空池中的线程会得到处理机时间（被运行）。

一个 Thread 对象在它的生命周期中会处于各种不同的状态。状态转移图如图 3-1 所示。

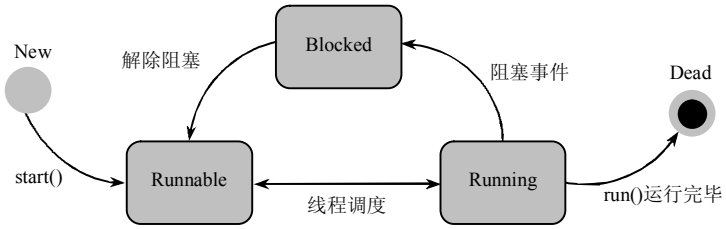

图 3-1 线程的状态转移图

线程进入 Runnable（可运行）状态，并不意味着它立即开始运行。在一台只有一个 CPU 的机器上，在一个时刻只能进行一个动作。

因为 Java 线程是抢占式的，所以开发人员必须确保代码中的线程会不时地给其他线程运行的机会，可以通过在各种时间间隔中发出 sleep()调用来做到。sleep()方法对当前线程操作，它是一个静态方法，使用 Thread.sleep(x)调用。sleep()的参数指定以毫秒为单位的线程最小休眠时间。除非线程因为中断而提早恢复执行，否则它不会在这段时间之前恢复执行。使用该方法只是使当前线程中断多少毫秒，并不是创建多线程。

Thread.sleep()和其他使线程暂停一段时间的方法是可中断的。线程可以调用另外一个线程的 interrupt()方法，这将向暂停的线程发出一个 InterruptedException。

3.2.4 线程的中断与恢复

一个线程可能因为各种原因而不再是可运行的：

（1）该线程调用 Thread.sleep() 进入中断状态时，必须经过规定的毫秒数才能从中断状态进入可运行状态。

（2）该线程进行 I/O 操作而进入中断状态，必须等待 I/O 操作完成，才能进入可运行状态。

（3）该线程调用了其他线程的 join()方法，而使自己进入中断状态，必须等待调用的线程执行完，才能进入可运行状态。

（4）该线程试图访问被另一个线程锁住的对象而进入中断状态，必须等待另一个线程释放对象锁，该线程才能进入可运行状态，例如该线程调用 wait()方法而进入中断状态，则必须通过其他线程调用 notify()或 notifyAll()方法才能进入可运行状态。

3.2.5 线程的同步

为了保证共享数据在任何线程使用它完成某一特定任务之前是一致的，Java 使用关键字 synchronized，允许程序员控制共享数据的线程。

synchronized 关键字使线程能和锁标志（每个对象都有一个和它相关联的标志）交互，即允许独占地存取对象。当线程运行到 synchronized 语句，它检查作为参数传递的对象，并在继续执行之前试图从对象获得锁标志。

持有锁标志的线程执行到 synchronized()代码块末尾时将释放锁。若出现中断或异常而使得执行流跳出 synchronized()代码块，锁也会自动返回。此外，如果一个线程对同一个对象两次发出 synchronized 调用，则在跳出最外层的块时，标志会正确地释放，而最内层的将被忽略。

3.2.6 面试题

1．问：下面关于线程的说法正确的是（多选题）（　　）。

A．支持多线程机制

B．一个线程创建并启动后，它将执行自己的 run()方法，如果通过派生 Thread 类实现多线程，则需要在子类中重新定义 run()方法，把需要执行的代码写入 run()方法中；如果通过实现 Runnable 接口实现多线程，则要编写接口中的抽象方法——run()方法的方法体

C．要在程序中实现多线程，必须导入类：import java.lang.Thread;

D．一个程序中的主类不是 Thread 的子类，该类也没有实现 Runnable 接口，则这个主类运行不能控制主线程的休眠

解析：本题考查的知识点是线程的使用，控制线程的类可以不是 Thread 类的子类，也可以不必实现 Runnable 接口，但它也可以调用创建对象的休眠方法进行休眠，所以选项 D 是错误选项。

答案：A B C

2．问：什么是 Java 的线程？它与进程的区别是什么？

答案：线程是程序运行的基本执行单元。当操作系统（不包括单线程的操作系统，如微软早期的 DOS）在执行一个程序时，会在系统中建立一个进程，而在这个进程中必须至少建立一个线程（这个线程被称为主线程）来作为这个程序运行的入口点。因此，在操作系统中运行的任何程序都至少有一个主线程。

在操作系统中可以有多个进程，这些进程包括系统进程（由操作系统内部建立的进程）和用户进程（由用户程序建立的进程），一个进程中可以有一个或多个线程。

进程和进程之间不共享内存，也就是说系统中的进程是在各自独立的内存空间中运行的。而一个进程中的线程可以共享系统分派给这个进程的内存空间。线程不仅可以共享进程的内存，而且还拥有一个属于自己的内存空间，这段内存空间也叫做线程栈，是在建立线程时由系统分配的，主要用来保存线程内部所使用的数据，如线程执行函数中所定义的变量。

3．问：创建线程的两种方式是什么？两者分别有哪些优点？

答案：第一种方式是使用 Runnable 接口创建线程，该种方式的优点如下：

（1）可以将 CPU、代码和数据分开，形成清晰的模型。

（2）线程体 run()方法所在的类可以从其他类中继承一些有用的属性和方法。

（3）有利于保持程序的设计风格一致。

第二种方式是直接继承 Thread 类创建对象，该种方式的优点如下：

（1）Thread 子类无法再从其他类继承（Java 语言单继承）。

（2）编写简单，run()方法的当前对象就是线程对象，可直接操作。

在实际应用中，几乎都采取第一种方式。

4．问：线程的生命周期是怎样的？

答案：线程的生命周期为：新建→就绪（阻塞）→运行→死亡。

（1）当用 new 创建完一个线程对象后，该线程处于新建状态。

（2）当线程对象调用了 start()后，该线程处于就绪状态。

（3）如果处于就绪状态的线程获得 CPU 时间片，开始执行 run 方法的线程执行体，该线程处于运行状态。

（4）如果线程调用了 sleep()或者调用了一个阻塞式 IO 方法等，该线程处于阻塞状态。

（5）如果线程的 run()执行完成或者抛出一个未捕获的异常等，该线程处于死亡状态。

5．问：如何终止一个进程？请举例说明。

答案：当一个线程结束运行并终止时，它就不能再运行了。可以用一个标志来指示 run()方法，必须退出一个线程。例如线程运行类的代码如下：

```
public class Runner implements Runnable {
    private boolean timeToQuit = false; //终止标志
    public void run() {
        while(! timeToQuit) {
            //当结束条件为假时运行
            ...
        }
    }

    //停止运行的方法
    public void stopRunning() {
        timeToQuit = true;
    }
}
```

线程控制类的参考代码如下：

```
public class ControlThread {
    private Runnable r = new Runner();
    private Thread t = new Thread(r);
    public void startThread() {
        t.start();
    }
    public void stopThread() {
        r.stopRunning();
    }
}
```

6．问：使用什么方法获取线程的优先级？使用什么方法设置线程的优先级？

答案：使用 getPriority 方法获取线程的当前优先级，使用 setPriority 方法设定线程的当前优先级。线程优先级是一个整数（1～10）。

7．问：线程死锁一般发生在什么情况下？避免线程死锁的通用经验法则是什么？

答案：当一个线程等待由另一个线程持有的锁，而后者正在等待已被第一个线程持有的锁时，就会发生死锁。

避免死锁的一个通用的经验法则是：决定获取锁的次序并始终遵照这个次序。按照与获取相反的次序释放锁。

8．问：调用 Thread 类的 destroy()方法会有什么后果？

答案：Thread 的 destroy()方法从来就没有被实现过，不能用它来销毁线程：

```
public void destroy() {
    throw new nosuchmethoderror();
}
```


该方法最初用于破坏该线程，但不作任何清除，它所保持的任何监视器都会保持锁定状态。不过，该方法绝不会被实现。即使实现也极有可能以 suspend() 方式被死锁。如果目标线程被破坏时保持一个保护关键系统资源的锁，则任何线程在任何时候都无法再次访问该资源。如果另一个线程曾试图锁定该资源，则会出现死锁。这类死锁通常会证明它们自己是“冻结”的进程。

9．问：sleep() 和 wait() 有什么区别？

答案：sleep 是线程类（Thread）的方法，导致此线程暂停执行指定时间，将执行机会给其他线程，但是监控状态依然保持，到时后会自动恢复。调用 sleep 不会释放对象锁。

wait 是 Object 类的方法，对此对象调用 wait 方法导致本线程放弃对象锁，进入等待此对象的等待锁定池，只有针对此对象发出 notify 方法（或 notifyAll）后，本线程才进入对象锁定池，准备获得对象锁进入运行状态。

10．问：Java 中的管道流是做什么用的？它与一般的输入输出流有什么不同之处？

答案：管道用来把一个程序、线程和代码块的输出连接到另一个程序、线程和代码块的输入。java.io 中提供了类 PipedInputStream 和 PipedOutputStream 作为管道的输入/输出流。管道输入流作为一个通信管道的接收端，管道输出流则作为发送端。管道流必须是输入输出并用，即在使用管道前，两者必须进行连接，可以在各自的构造方法中进行连接，或者使用各自的 connect()方法进行连接。

11．问：请使用 Java 中的管道输入输出流完成如下功能：管道输出流发送消息"Hello! Amigo. "，管道输入流接收到信息后打印出来。

答案：本题需要三个类进行写作：管道输出流（用于进行数据发送）类 PipeSender、管道输入流（用于接收数据）类 PipeReceiver、测试类 PipeTest。主要用到的类为 PipedOutputStream 和 java.io.PipedInputStream 类。

测试类 PipeTest 类的参考代码如下：

```
package com.itjob.javaadvanced;
import java.io.IOException;
public class PipeTest {
    public static void main(String[] args) {
        PipeSender sender = new PipeSender();
        PipeReceiver receiver = new PipeReceiver();
        try {
```

```
                // 连接管道
                sender.getPos().connect(receiver.getPis());
            } catch (IOException e) {
                e.printStackTrace();
            }

            // 启动线程
            new Thread(sender).start();
            new Thread(receiver).start();
        }
}
```

管道输出流（用于进行数据发送）类 PipeSender 的参考代码如下：

```
package com.itjob.javaadvanced;
import java.io.IOException;
import java.io.PipedOutputStream;
class PipeSender implements Runnable {
    private PipedOutputStream pos = null;
    public PipeSender() {
        this.pos = new PipedOutputStream();
    }

    public void run() {
        String str = "Hello! Amigo.";
        try {
            // 发送信息
            this.pos.write(str.getBytes());
        } catch (IOException e) {
            e.printStackTrace();
        }
        try {
            // 关闭输出流
            this.pos.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
    public PipedOutputStream getPos() {
        return pos;
    }
}
```

管道输入流（用于接收数据）类 PipeReceiver 的参考代码如下：

```
package com.itjob.javaadvanced;
import java.io.IOException;
import java.io.PipedInputStream;
class PipeReceiver implements Runnable {
    private PipedInputStream pis = null;
    public PipeReceiver() {
        this.pis = new PipedInputStream();
    }
```