# 《Java 程序设计》实验报告

**班级名称：**

**学号：**

**姓名：**

## 实验一 java 程序的编辑、编译和运行

### 实验目的

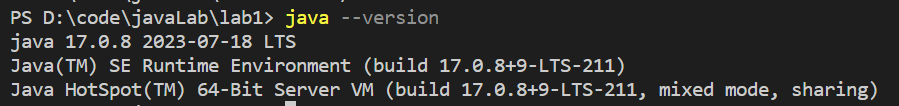
理解环境变量，了解环境变量的设置；掌握命令行下运行java程序的过程、需要的软件环境、生成的结果等。

### 实验内容

* 用记事本编写含有至少两个类的java源程序（扩展名为Java），理解和领会源程序名称和类名的关系
* 在命令行下编译(javac)源程序，生成类文件，即目标文件（扩展名为class），认识和理解生成的目标文件和源程序中类的关系、类文件的数量
* 在命令行下运行(java)程序，理解各个类文件之间的关系：是不是每个类文件都可以运行，删除某些类文件的运行情况
* 领会Java的类型和程序执行过程，理解java跨平台的原理

### 实验过程和结果分析：

1. 配置环境变量，使用java –version命令检测是否配置成功

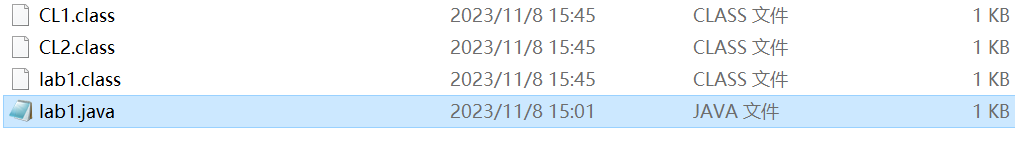


1. 创建文件lab1.java。在其中创建三个类分别为

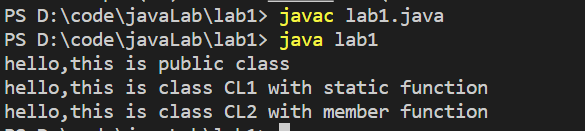
Public class lab1,class CL1,class CL2

在lab1中创建main函数，在CL1和CL2中分别创建一个显示信息的成员函数

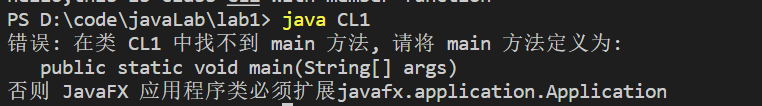
1. 在命令行中执行javac lab1.java命令生成3个文件



1. 可用java命令执行lab1.class,程序正常运行



但是CL1和CL2是无法执行的



原因是一个.java文件中只能有一个public修饰的类，而只有这个类中可以有main函数，这个函数将作为程序的入口

### 总结

#### 实验内容总结：

本次实验旨在让学生理解环境变量的设置，掌握命令行下运行Java程序的过程，了解生成的结果。通过使用记事本编写Java源程序，编译源程序生成类文件，然后在命令行下运行程序，学生能够领会Java的类型和程序执行过程，并理解Java的跨平台原理。

#### 心得体会：

通过这次实验，我深刻理解了环境变量的重要性，以及如何配置和使用它们。我也学会了如何编写Java程序、编译源代码，生成类文件，并在命令行下运行程序。同时，我明白了Java中每个.java文件只能有一个public修饰的类，并且只有该类中可以有main函数作为程序的入口。这个实验帮助我建立了对Java编程环境和基本操作的初步认识，为今后的学习和实践奠定了基础。

## 实验二 数据类型和数组

### 实验目的

学会使用各种数据类型和数组存储数据和处理数据。

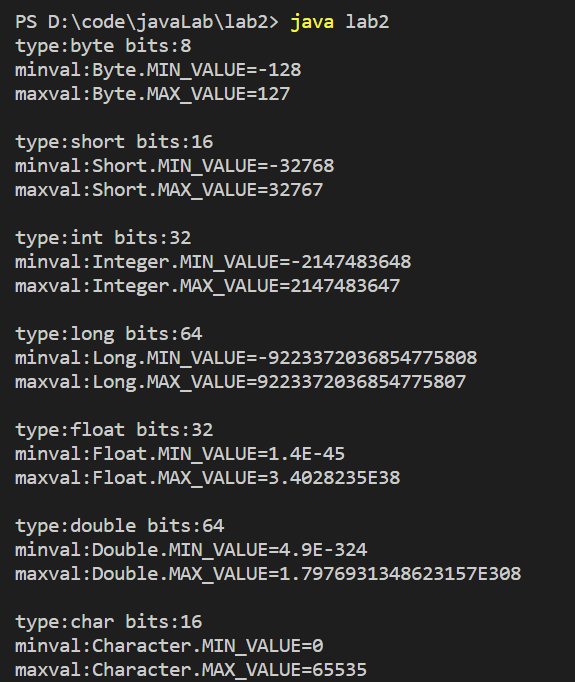
### 实验内容

* 定义各种基本类型，输出各种类型的占用内存大小，数值型数据的最大、最小值
* 分别定义一个基本类型的数组和对象类型的数组
* 对数组元素进行初始化
* 数组元素存取
* 输出满足一定条件的数组元素

### 实验过程和结果分析

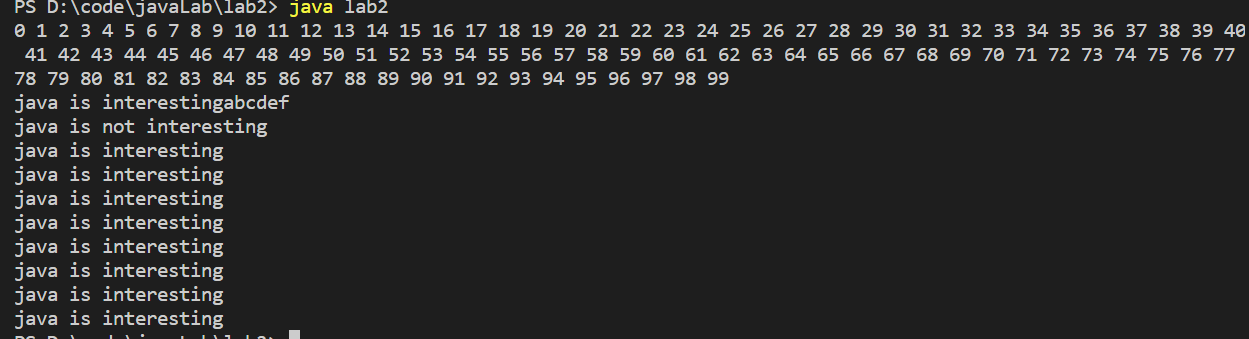
1. 在java中为每一个基本数据类型封装了一个类，例如Byte,Interger,Long等，这些类中描述了该类型的相关信息，因此我们可以直接使这些类中的成员常量SIZE，MIN\_VALUE，MAX\_VALUE打印出该类型占多少bit,以及十进制下的最小值和最大值

运行结果如下：



1. 分别选取int类型和StringBuilder类型作为基本数据类型和对象类型创建数组并初始化。可以通过[]数组下标直接访问数组元素。最后通过for each遍历数组中的每一个元素打印数组元素

运行结果如下



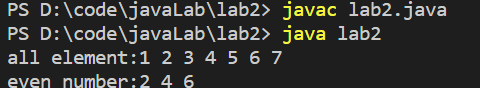
1. 对于要求5，定义如下函数

public static void ques5(int []arr)

然后遍历数组打印出其中的偶数

在c/c++中，当将一个数组作为函数参数传递时，数组将会退化为指针，失去原来数组的大小信息，而在java中，数组在底层进行了进一步的封装，随时拥有length常量维护数组大小。因此在java中传递数组时无需传递数组的大小信息

运行结果如下：



### 总结

#### 实验内容总结：

本次实验的目的是让学生掌握各种数据类型和数组的使用，包括定义基本数据类型、输出其占用内存大小和数值范围，创建基本类型和对象类型的数组，对数组元素进行初始化和存取，以及输出满足特定条件的数组元素。

#### 心得体会：

通过这次实验，我学会了如何操作各种数据类型和数组。我了解了如何获取各种基本数据类型的占用内存大小和数值范围，以及如何创建不同类型的数组并初始化它们。我还学会了使用循环和条件语句来遍历数组元素，并输出符合条件的元素。这个实验让我更加熟悉Java中的数据类型和数组操作，为以后的编程和数据处理提供了基础知识。

## 实验三 java程序的内存分配

### 实验目的

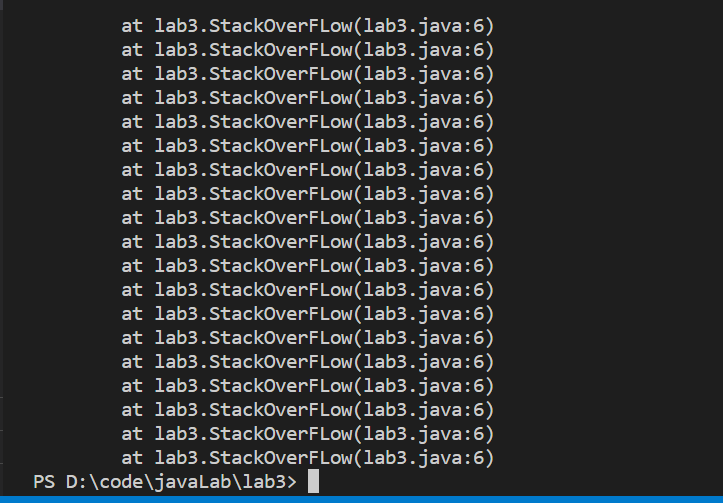
理解java程序的内存分配，写出高效率的程序。

### 实验内容

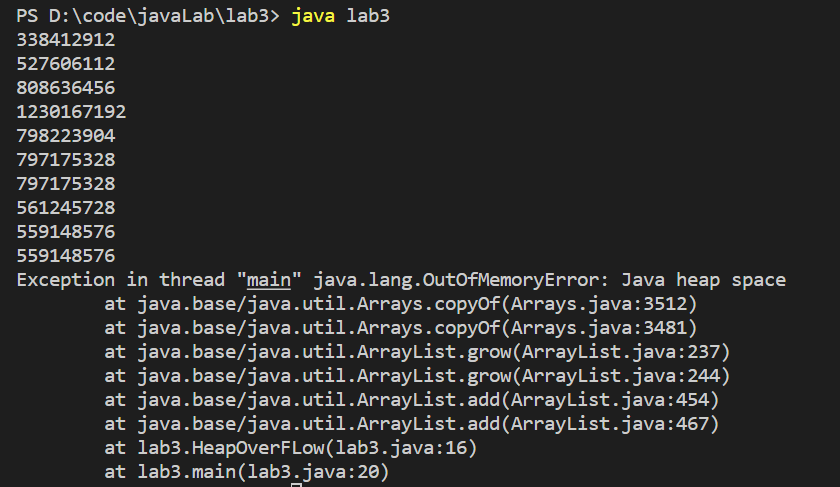
* 编写一个简单的栈内存溢出的程序，理解stack 内存和java的stack 内存的分配
* 编写一个简单的堆内存溢出的程序，理解heap 内存和java的heap 内存的分配
* 编写一个简单的程序，单步执行程序，验证内存数量的变化

### 实验过程和结果分析

1. 栈溢出最经典的案例就是递归的层数过多。因此对于要求1只需定义一个没有递归头的递归函数即可



1. 对于堆溢出，使用一个死循环无限创建StringBuilder对象，同时使用Runtime.getRuntime().freeMemory()函数监视内存的使用情况，每运行50000000次打印一次剩余内存。运行代码得到如下结果



可以看到内存并不是想象中的一直减小，而是有时增加有时减小。分析原因为，java存在Garbage Collection机制，即垃圾回收机制。JVM中存在一套算法检测一个对象是否还会被需要，当JVM认为该内存不被需要时会被自动销毁。

### 总结

#### 1.实验内容总结：

本次实验旨在让学生理解Java程序的内存分配，包括栈内存和堆内存的分配。学生需要编写程序来触发栈内存溢出和堆内存溢出，以理解这些内存区域的特点。还需要单步执行程序，验证内存数量的变化，以更好地了解Java程序的内存管理。

#### 2.心得体会：

通过这次实验，我深入了解了Java程序的内存分配机制。我学会了如何触发栈内存溢出和堆内存溢出，以及如何监视内存的使用情况。我也了解了Java中的垃圾回收机制，它会自动销毁不再需要的对象，导致内存占用量的波动。这个实验让我更好地理解了Java程序的内存管理，为编写高效的程序提供了重要的知识。

## 实验四 静态fields and methods

### 实验目的

理解和掌握Java的static fields and methods.

### 实验内容

根据现实世界的某个应用，创建一个java类，类中包含static fields 和 methods,验证static field 和 methods 的特点。

* 编写程序验证static fields 和methods的存取方法
* 编写程序验证 static field ,类变量，所有对象共享一个内存，创建对象static field不占用内存

### 实验过程和结果分析

定义一个Student类含有如下成员变量

class Student{

public static LocalDate timeOfEnrollment = LocalDate.of(2023, 9, 1);

public static void getTimeOfEnrollment(){}

private String name;

private int uid;

public Student(String name,int uid){}

public String toString(){}

}:

静态成员变量timeOfEnrollment表示入学时间

非静态成员变量name和uid分别表示姓名和学号

创建两个对象alice 1001和clara1002

静态字段timeOfEnrollment被所有Student类的实例共享，因此通过任何一个对象修改了timeOfEnrollment，所有对象都会受到影响。

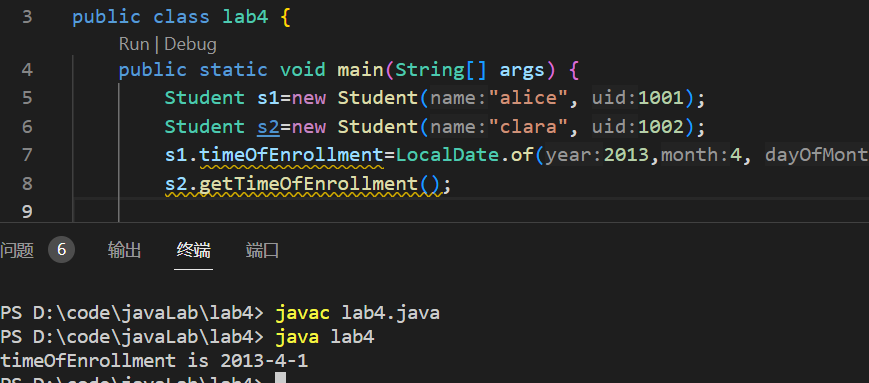
Student s1=new Student("alice", 1001);

Student s2=new Student("clara", 1002);

s1.timeOfEnrollment=LocalDate.of(2013,4, 1);

s2.getTimeOfEnrollment();

在主函数中做如下调用发现当修改alice的入学时间后,clara的入学时间也改变，说明了静态对象的共享。



### 实验总结：

静态字段的共享特性： 实验证明，静态字段是类级别的，所有对象共享同一个内存空间。当通过一个对象修改了静态字段的值后，其他对象也能看到这个变化。

静态方法的调用： 静态方法属于类而不是对象，可以通过类名直接调用，不需要创建类的实例。在本实验中，通过s2.getTimeOfEnrollment()调用了静态方法，获取并输出了入学时间信息。

静态字段不占用对象内存： 实验证明，创建多个Student对象并不会导致静态字段timeOfEnrollment占用更多的内存，因为它是与类关联的，而不是与对象关联的。

### 心得体会：

通过这个实验，我深入理解了Java中静态字段和静态方法的特点。静态字段的共享性使得它适用于表示类级别的信息，而静态方法的调用方式也与对象实例无关。这些特性在编写大型应用程序时非常有用，可以在不创建对象实例的情况下访问和修改类级别的信息。同时，对于静态字段不占用对象内存的特性，也为优化内存使用提供了一种方式。在实际编程中，合理使用静态字段和方法可以提高代码的可读性和性能。

## 实验五 java 的access control

## 实验目的

理解和掌握Java的access control,特别是protected 级别的存取控制

### 实验内容

根据现实世界的某个应用，抽象创建一个java类，类中使用 public 、protected、private等fields 和 methods

* 体会同一个类中访问private, protected, public 域和方法的情况
* 体会一个类访问另一个类中private, protected, public 域和方法的情况

### 实验过程和结果分析

创建一个student类定义如下。

class Student {

private double score;

protected String name;

protected int age;

protected int id;

public Student(int id,String name,int age,double score){

this.name=name;

this.age=age;

this.id=id;

this.score=score;

}

public String showStudentInfo(){

String msg=showInfo()+"成绩:"+this.score+"\n";

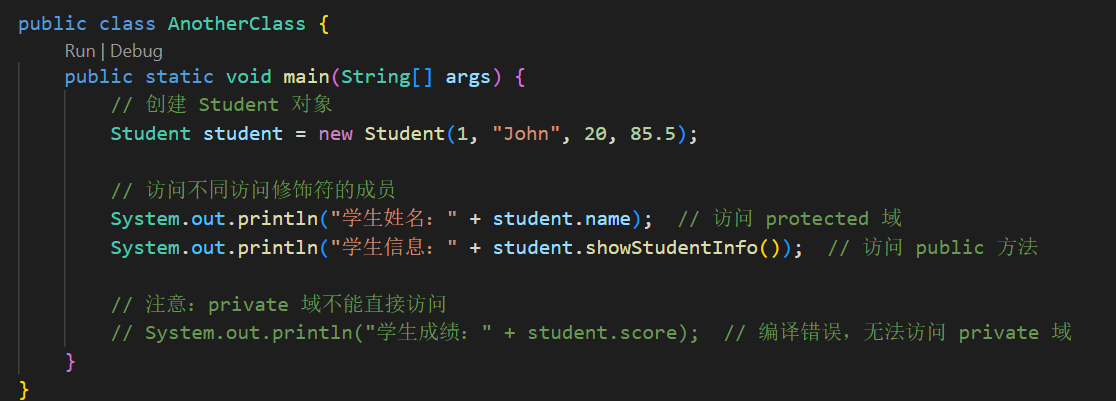
return msg;

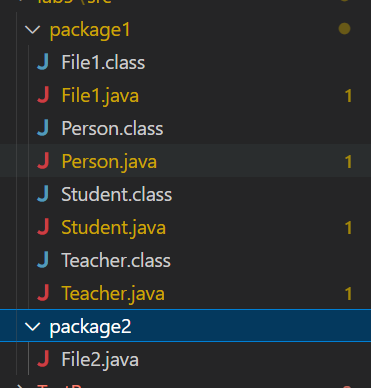
}

}

将学生的名字，年龄，学生号设置为protected或public，将score设置为priavte可以得到，在类内可以直接访问任何修饰符的属性。

我们定义另一个类去访问student的protected域name和public方法showStudentInfo。而private域score在AnotherClass类中无法直接访问，因为它只能在Student类内部访问。



针对protected先创建了如下目录结构：

Package1的包下有Person，Student类。Package1包中的类可以自由访问类中其他protected级别的属性。

而在另一个包package2中则不能访问package1中的protected成员属性。

### 实验总结：

Java中的访问修饰符用于控制成员的可见性。private表示只能在类内部访问，protected表示在同一个包内或子类中可访问，public表示任何地方都可访问。在另一个类中，只有public的成员才能直接访问，而private和protected的成员需要通过公共方法或继承关系进行访问。

### 心得体会：

通过这个实验，我深入理解了Java中的修饰符控制成员可见性的操作，进一步理解了对代码的封装。

## 实验六 java 面向对象编程1

## 实验目的

了解和掌握Java的OOP的基本概念并灵活运用

### 实验内容

根据现实世界的某个应用，抽象创建一个java类，类中使用 public 、private、static、final 等定义fields和methods

* 体现继承 Object对象
* 包含overloaded method（构造函数、一般函数）的使用
* 定义该类的若干对象，存取对象的属性和运行行为

### 实验过程和结果分析

创建一个Person类定义如下。

public class Person extends Object{

。。。。。。省略部分代码

protected String showInfo(){

String msg="姓名:"+this.name+"\n年龄:"+this.age+"\n学号:"+this.id+'\n';

return msg;

}

@Override

public String toString() {

// TODO Auto-generated method stub

return showInfo();

}

}

Person类继承自Object,同时我们重写了Object的toString方法实现了打印Person信息的成员函数

同时定义如下重载函数

public Person(){

this.name="NULL";

this.age=-1;

this.id=-1;

}

public Person(int id,String name,int age){

this.name=name;

this.age=age;

this.id=id;

}

这样实现了用户可以通过多种方式构造函数进行对象信息的存储。

Person p1=new Person();

p1.name="张三";

System.out.println(p1.showInfo());

Person p2=new Person(10, "李四", 12);

System.out.println(p2.toString());

然后创建两个对象，应该用无参构造函数一个用有参构造函数构造对象，并使用toString()打印他们的信息

### 实验总结：

在本次实验中，通过创建Java类展现面向对象编程的核心概念，深化了对public、private、static、final等关键字的理解。方法的重载和继承Object对象的使用提升了代码灵活性，成功创建多个对象演示了类的属性和行为的存取。

### 心得体会：

通过这个实验，我深入理解了Java中的oop的概念，熟悉了java中oop的操作，学习了在写代码的过程中如何更好地加入面向对象的思想封装代码

## 实验七 java 面向对象编程2

## 实验目的

了解和掌握Java的OOP中的继承、多态、覆盖、接口

### 实验内容

根据现实世界的抽象，创建一个多层继承关系的java类，至少两层（不包括继承Object）类中使用 public 、private、static、final 等定义fields和methods

* 体现组合的重用
* 体现类之间继承的关系重用
* 体现override函数和多态的特性

### 实验过程和结果分析

基于Java的线性表库，我设计了一个LoggingList，LoggingList类继承自ArrayList，SpecializedLoggingList类继承自LoggingList，体现了继承和多态的特性。

同时使用@Override覆盖了log方法，同时定义了Loggable接口，

然后 在LoggingList类中使用了ArrayList的功能，通过组合实现了对ArrayList的重用和功能的拓展。

SpecializedLoggingList类在继承LoggingList的基础上扩展了新的功能，体现了类之间继承关系的重用。

interface Loggable {

void log(String message);

}

class LoggingList<E> extends ArrayList<E> implements Loggable {

private static final String LOG\_TAG = "LoggingList";

@Override

public void log(String message) {

System.out.println(LOG\_TAG + ": " + message);

}

public void addAndLog(E element) {

super.add(element);

log("Added: " + element.toString());

}

}

class SpecializedLoggingList<E> extends LoggingList<E> {

@Override

public void log(String message) {

System.out.println("Specialized " + LOG\_TAG + ": " + message);

}

public void addSpecial(E element) {

addAndLog("Special: " + element.toString());

}

}

### 实验总结：

在本次实验中，通过创建一个多层继承关系的Java类，使用List、ArrayList、LinkedList等集合类进行演示，体现了面向对象编程（OOP）的核心概念。

### 心得体会：

通过这个实验，我深入理解了Java中的oop的概念，通过多层继承，如LoggingList和SpecializedLoggingList，展示了Java中OOP的继承与多态概念。使用接口和方法覆盖实现了灵活的功能扩展，而组合重用的思想在类设计中提供了灵活性。

## 实验八 String类的不变性

## 实验目的

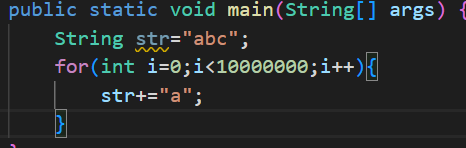
理解String类不变性的目的，解决不变性带来的问题。

### 实验内容

* 简历一个初始的String类，初始化；
* 在初始化值的基础上，不断增加字符串长度，改变String 类的值，验证String类的不变性；
* 用StringBuffer, StringBuilder解决String类带来的消耗内存的问题。

### 实验过程和结果分析

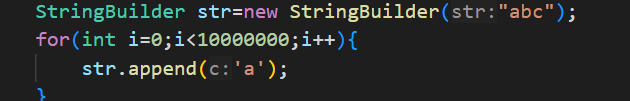
我们首先使用String类型自带的方法进行大量的字符串拼接执行程序



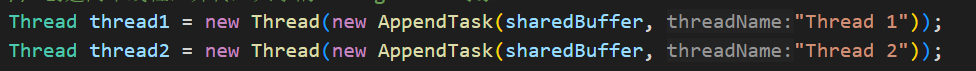
发现程序运行时间超过了20秒，并且cpu占用率和内存开销都很高。这说明String类型是不可变的，每次进行字符串拼接其实是新生成了一个String对象，又由于垃圾回收机制的存在大大降低了性能



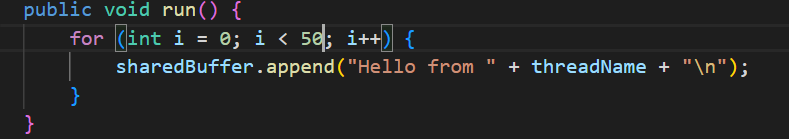
之后我们使用StringBuilder进行同样的操作，可以发现这次程序不到1秒就运行结束。



用StringBuffer可以看到同样的效果，不过StringBuffer更适用于多线程环境。我们创建两个线程

每个线程的任务是给字符串sharedBuffer拼接字符串。





演示发现并没有出现线程安全问题，说明stringBuffer是线程安全的。

### 实验总结：

### 在实验中，我们学习了StringBuilder和StringBuffer两个类，它们都是用于处理可变字符串的工具类，StringBuilder和StringBuffer都允许字符串的内容被修改，因此属于可变类。相比之下，String类是不可变的，一旦创建就不能被修改。

### 心得体会：

通过这个实现我了解到在使用java时更好的操作字符串的方法。