**CSC207 笔记和易错点**

**M0 Git入门**

**什么是版本控制（Version Control）：**

版本控制系统是一种控制我们代码版本的一种方式，它可以追踪我们代码的改动以及帮助我们进行版本复现。

想象一下，在一个基于代码的项目中，你会如何与他人（或跨多台机器与自己）合作：你会四处发送代码的最新版本 .zip 文件吗？你会将修改后的片段发送给拥有主副本的其他人吗？你会使用协作软件（例如 Google Docs）并在同一个地方更新代码吗？

所有提到的可能性都有缺点：要么很难跟踪当前版本或找出每一步的修订版本，要么很难独立工作。如果你犯了错误需要回滚更改，你要么必须找到旧版本或你的代码，要么手动回滚！这太繁琐了！

版本控制没有这些缺点：每个人都在自己独立的版本上工作，版本控制系统允许你根据需要合并更改，并跟踪所做的修订。

-来自教材提供例子

总结，版本控制的目有3：1.保证在多人合作的项目中，每个人的工作不会过多的影响到其他合作者；2.提供一种再多人合作项目下将不同人负责部分整合在一起的方法；3.当程序在一次更新后出现问题后，可以快速的回滚至旧版本

**版本控制如何工作：**

在一个版本控制系统中，将会有一个主储存库（master repository）：这是一个对于所有文件最后一次更新的复制文件。合作者须将这些文档（库）复制一份到他们的本地运行环境上，然后独立完成各自工作。

当合作者完成一些修改并希望将它们提供给其他合作者时，他们需要首先**push**他们的修改。相应的，如果需要获取最新的库存，则需要**pull**这些修改。

虽然各个版本控制系统有不同的特性，但是了解以上基础知识是学习如何使用版本控制系统的前提。

**Git入门**

Git是一种常用的版本控制系统

在git中，主存储库通常被称为**Origin**。我们常使用git clone <url>来获取一份本地库的复制。此命令类似与下载所有本地文档，但是提供了一些特殊的方法（git status）来进行文档修改的追踪。

1 修改初始文档

如果我们希望将一些修改后的文件更新至主存储库中，我们可以使用git add <files>来添加所有我们希望修改的文件。

在完成add命令后，我们需要紧接着使用git commit -m “<message>”来提供我们此次修改的描述（例如“修改了xxx”），方便后续版本回退时进行更快捷的查找。最后，当我们完成这些所有的工作后，使用git push进行最终提交

总结：如果你需要进行一次对主存储库的更新，请按照以下步骤完成：

git status:来查看那些文档被修改了（将会暂时工作路径和暂存区的情况，例如修改和暂存文件）

git add <file>:提供需要被提交的修改的文档（此时文档并没有被正式的更新至主存储库，此行为被称为暂存更改）

git commit -m “<message>”:为你的更新提供描述（将会被永久展示在主存储库的修改提交信息中）

git push:提交目标更改至主存储库

2 拉取新版本

git pull命令常用于获取主存储库的更新（将最新的主存储库内容复制到你的本地环境）

请注意，此操作可能会导致一些文档冲突或者其他错误。你需要通过adding/committing 本地修改或者手动处理冲突（常见于两个合作者对同一个文档进行修改的情况）

3 分支

有时你可能会在两个不同的库存来处理整个功能，此时如果将你的内容提交至主库存可能不符合你的需求（比如你的部分是完成库存中的某个子文件夹内容程序编写，你不应该将这些文档重新提交至主库存，这会改变库存的原始结构，同时对于这些子库存的修改也不应该在主库存中被看到）。

对于这种情况，你可以创建一个新分支，等待你将所有的工作完成后，再将他们整个提交至主库存中。

如果需要完成以上操作，首先使用git checkout -b <branch name>。或者git branch <branch name> + git checkout <branch name>（第一句命令用于创建新分支，第二句命令用于将你的修改传送至分支而不是主库存）。

同理，如果需要提交文档到其他的分支，使用git checkout <branch name>即可。

当完成分支任务，需要将文档融合至主库存时：git merge <branch name>。然而，更好的方法是发出拉取请求，以便人们可以看到和审查您尝试推送的更改。（？）

4 其他资源

可视化理解版本控制：<https://betterexplained.com/articles/a-visual-guide-to-version-control/>

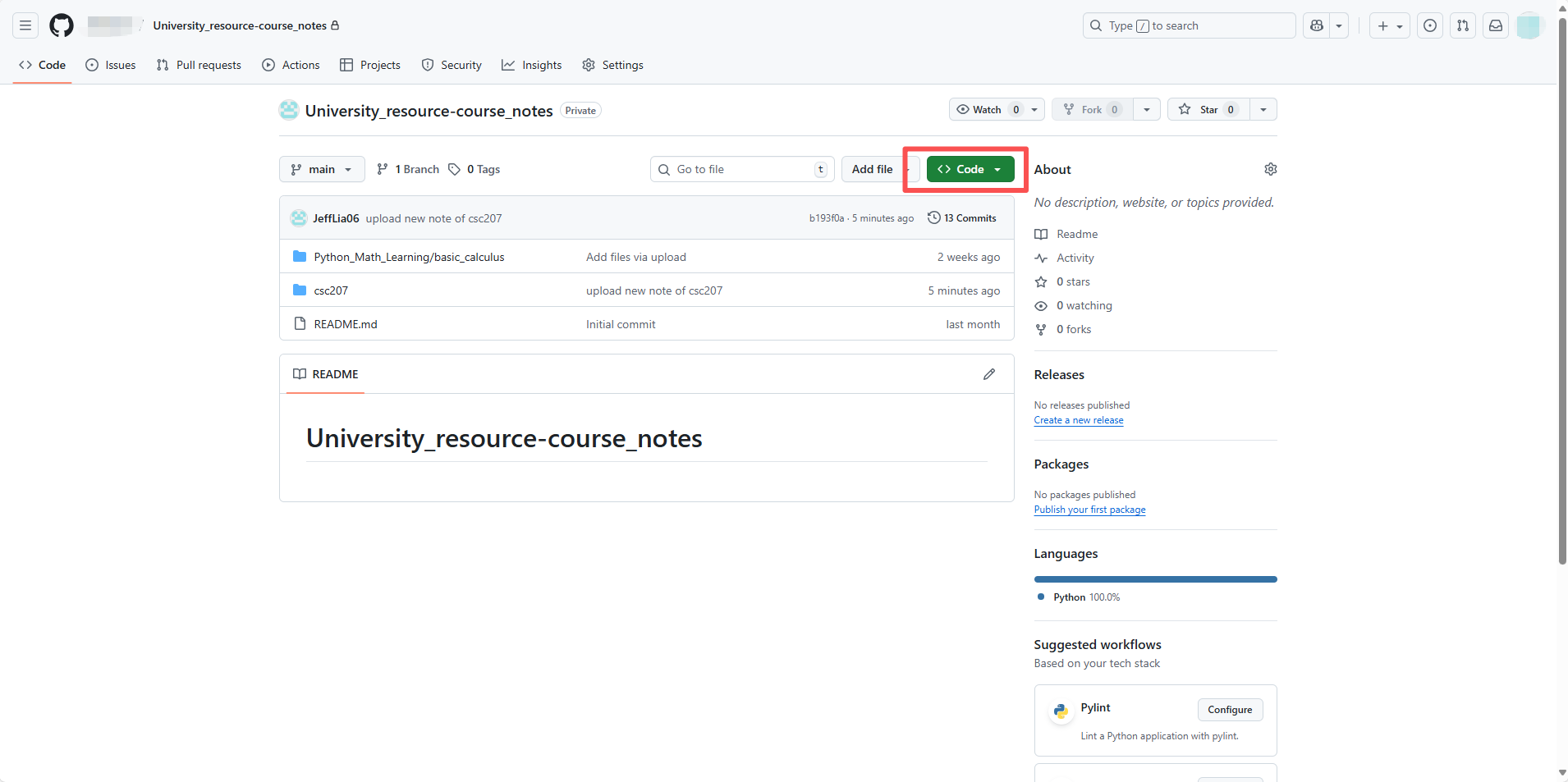
Github入门：<https://docs.github.com/en/get-started/start-your-journey>

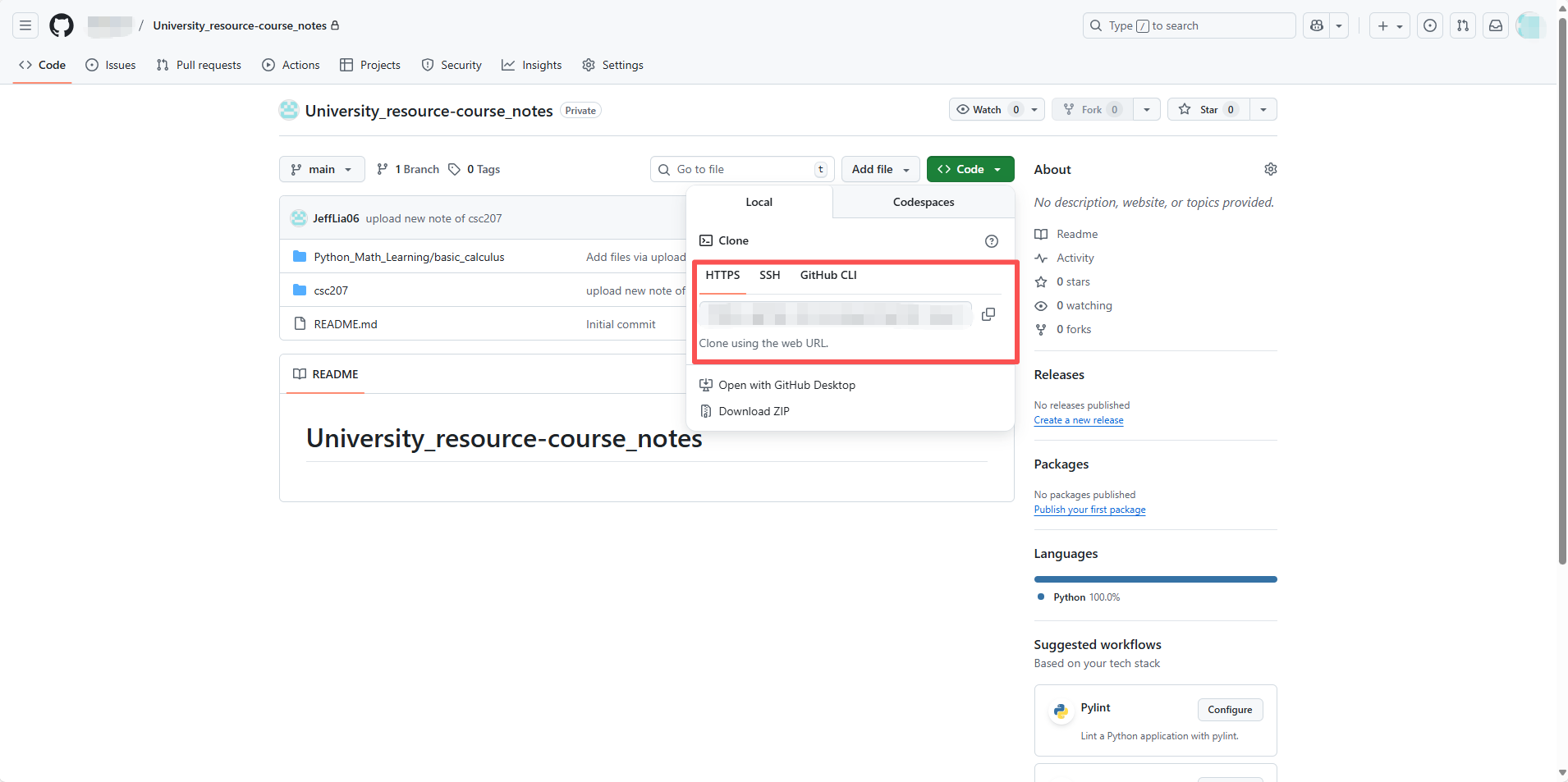
5 实战使用技巧

首先使用github来进行版本控制时，有几个额外步骤：

复制目标库存（若第一次使用库存）：git clone <url>：

url在此处：





完成库存复制后，之后所有的关于此库存的修改须cd至此文件夹进行。

若是此电脑第一次使用，需要通过网页端或者代码进行登录。同时在尝试commit时需要提供一个提交者名字和邮箱，具体代码如下：（此操作为个人电脑私人环境）

git config --global user.name "你的名字"

git config --global user.email "你的邮箱"

同时，git只能检测到主文件夹的更新，它无法对新创建的子文件夹的内容进行检查，须使用git add进行操作（首先使用git status查看具体哪个子文件夹未被添加，然后手动添加git add “文件夹”和commit + push）

需要注意一点，由于不适用git pull的话github不会自动更新本地库存，从而导致本地库存和云端有冲突文件。如果遇到了冲突文件的情况，则需要更具具体情况进行文件融合的工作。为了防止此类情况的发生，建议每次提交之前使用git pull对本地文档进行更新，从而避免单独处理冲突。

**M1 Java 入门**

从对象化软件设计的角度来讲，Java对比Python有以下几个优势：

1. Java现在依然是最广泛使用的计算机语言在工业中
2. 相比于Python的对象化编程，Java包含了更深入的对象化编程的概念，利于初学者
3. Java作为一个学习难度处于C和Python之间的语言，学习Java可以帮助理解其他更加复杂的语言

如何学习Java

1 Java如何运行一个程序

对于计算机来说，他们无法直接理解我们写的代码，通常需要将代码转化为一种机器语言从而进行运行。

这种转化过程通常来说有两种主要方法：

解释（Interpretation）：程序将被一行一行的进行阅读和运行（例如Python）

编译（Compilation）：整个程序会先被转化机械代码通过编译器，然后转化为一个可运行程序

对于Java来说，采取了一种混合的方法。Java的编译器（javac）会首先将源代码转化为字节码。然后Java虚拟机（Java Virtual Machine，JVM）将会解释和优化字节码在运行时。（在终端中，例如假设你有个程序 test.java，你可以使用javac test.java对它进行编译，生成一个test.class文件，随后就可以使用java test.java来运行代码）

为了运行一个Java程序，你首先需要编译它。

但是对于一个更大的项目，通常包含上百个类，使用javac进行一个个编译会变得非常麻烦。因此，现代Java程序通常会使用一些构建系统（Build System）例如Maven/Gradle。这些工具会帮助管理编译，依赖，和运行；从而让程序员可以专注于代码的写作上（这些功能通常已经被集成在现代的IDE上并自动运行）。但是作为计算机科学的学生，了解这些底层逻辑是必要的，并且可以有效的辅助后期C的学习。

2 Java的计算机架构介绍

为例理解一个Java程序是如何运行的，我们有必要了解一下计算机的架构：

应用（Application）

操作系统（Operation System，OS）

硬件（Hardware）

对于像C一类的语言，它让程序直接在OS上运行，因此代码并不是可移值的，这意味着每一次在不同的OS上都需要对程序进行重新编译。

而对于Java，则采取了一个虚拟机架构（Virtual Machine Architecture）：

一个虚拟机（Virtual Machine，VM）被置于应用和操作系统之间，充当一个固定的运行环境，而不是让程序直接在OS上运行。因此，使用VMs的编程语言所写的程序是可移值的，只要合适的Java VM被安装。

应用

虚拟机

操作系统

硬件

通过这个方法，Java的程序可以不用考虑操作系统进行运行。但是，Java的VM则是一种在OS上运行的一个程序，因此OS的一些特征依然会影响Java程序的运行。这些细节依然会被Java的程序自动处理，因此编程者无需担心这些问题。

Java 基础（所有代码演示请在csc207demo中查看）

Java的变量和类型

1 对比Python的区别

相比于Python，Java中的变量通常不能进行灵活的转换类型。在Java中变量需要强制规定它的数据类型在它定义时。这利于编译器在编译阶段发现部分漏洞并且让程序整体变得更加安全。

2 声明类型

对于Java来说，每个变量都需要一个明确的数据类型，例如 “int i；”定义了一个int类型的变量i。

3 声明与赋值

对于Java来说，并不需要在变量声明时就提供变量的值。声明变量值可以在声明变量后的任意一行出现。但是，需要注意的时，不提供变量值的变量会被赋予一些默认值，例如int类型的默认值为0，任意类的默认值是null

4 追踪Java变量

Java变量通常包含以下四个要素：

（在声明时提供的）变量名

（在声明时提供的）数据类型

用于存储变量数据的内容空间

（通过赋值语句提供的）变量值

以上四个要素只有变量值可以被直接修改

5 常见报错

变量未提供变量类型

错误赋值类型

使用相同变量名进行声明

Java的引用类型和原始类型

1 常用Java数据类型

原始类型

byte, int, float, short, long, double, char, booleam

引用类型

各种类（例如String, Integer），接口（List, Map, Runnable），数组（int[]），枚举（enum），泛型/接口实例化

2 引用类型与原始类型的区别

所有的原始类型都是不可变的，部分应用类型可变

原始类型变量直接存储在栈（stack），而引用类型在栈中存储引用对象的地址，实际数据存储在堆中（heap）

String类

1 String Pool/String Interning

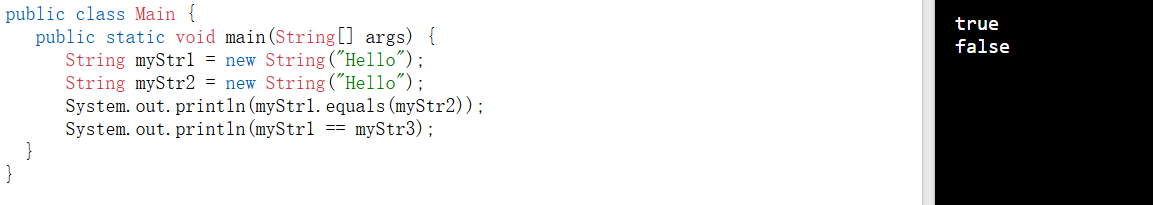
String类的变量基于不同的定义方法，会有不同的数据存储方法：

String s1 = “123”

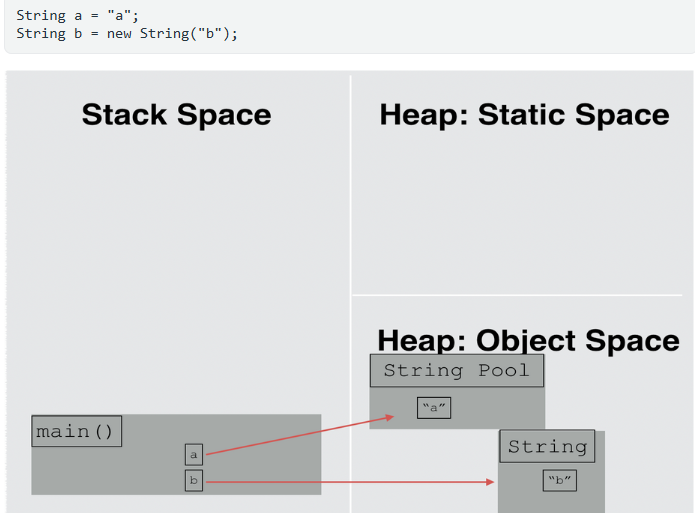
这种方法声明的String类变量会将“123”变量值存储在heap中的一个名为String Pool的空间中。此时如果有另外一个变量用此方法声明，则可以直接使用==进行字符值的比较。

String s1 = new String(“String”)

此方法定义了一个新的String类变量，类变量的值直接存储在heap中。使用==进行值比较会返回false，须使用equals()进行比较



下图为String Pool在内存空间的情况的可视化表达方法：

****

2 String是不可变的（immutable）

这意味着每次重新赋值都是创建一个新的内存空间

3 String的常用方法

通过+进行两个String变量的连接

.charAt(<index>)获取特定位置字符（类似于array的index方法）

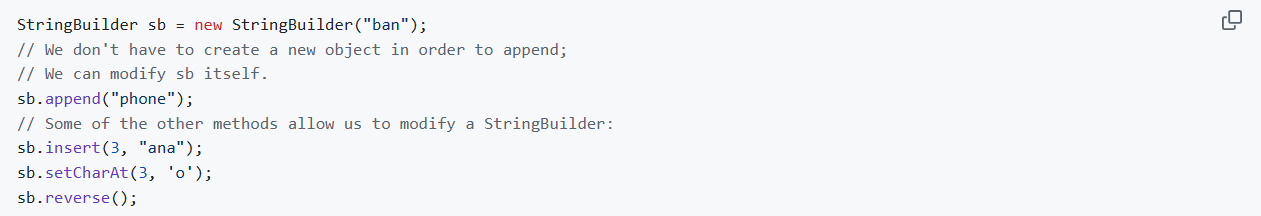
.subString(<start>, <end>)截取从start到end（不包含）的字符串

.trim()移除空白

.replace(<target>, <replacement>)

.splite(<target>)

4 创建可变String的方法：StringBuilder



5 String对比StringBuilder

通常来说，创建一个新的字符串会比修改一个已经存在的字符串更快（尤其是这种修改字符串的操作以循环等方式出现时）

Java的类

1 类初始化

当Java处理“new StringBuilder(“asdf”)”时

分配内存给新的对象

分析语句含义

调用合适的初始函数

将这个新的对象的索引返回到对象名上

2 APIs

告诉我们一个代码如何运行交互的文档被称为程序编程接口（API）。

3 类方法

类的方法简单分为两种：静态类方法和类方法

静态类方法（static）：方法的调用不需要对应类的实例化

类方法：方法的调用必须对应类实例化，且必须在对应实例化上的类实例进行使用

4 访问数据成员

在IDEA中，xx.会直接告诉编程者有哪些方法和参数可以调用。

但是关于参数和方法的调用会受到访问修饰符(Access Modifier)的影响：



5 高效管理对象

Java会自动追踪未使用的变量并删除他们，如果须手动删除某个变量，可以赋值为null（如何处理不需要使用的变量名）

Arrays（数组）

数组是最简单的一种表示一个集合的物品的数据类型。但是包含一些特点：

固定长度

所有的元素都为同一种数据类型

1 数组的声明

int[] numbers;

2 构造数组

int[] numbers = new int[5];

int[] numbers = {<element>};

3 Java数组的长度是不可变的

不可以修改已经声明的数组的长度，除非重新将此数组引用至一个新的数组

4 访问数组所有元素

numbers[0]：第一个

numbers[-1]：倒数第一个

numbers[0:2]：第一个到第二个

5 为什么Java的数组会有诸多限制

这种限制多的数据类型比较高效相较于python的list

Java同时也包含ArrayList等数据类型来实现相对灵活的数据集合方法

Extra：多数据类型的数组：

Object[] items;

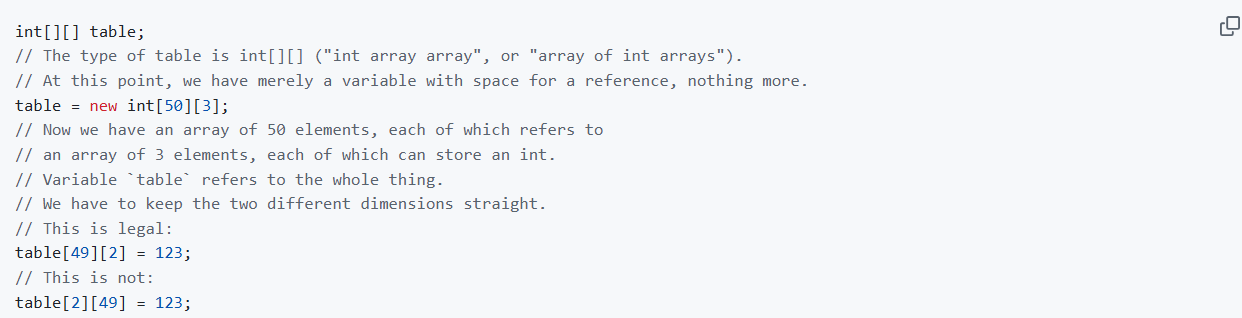
由于Java中所有的类都是Object的子类，因此可以使用Object来创建数组，用继承方法来将多种不同数据类型装在同一个数组中

6 二维/高维数组定义

int[row][column]

Extra：特殊的二维数组定义方法

可以后置定义每个行列的数据长度，而不是一开始确定全部长度



Aliases（别名）

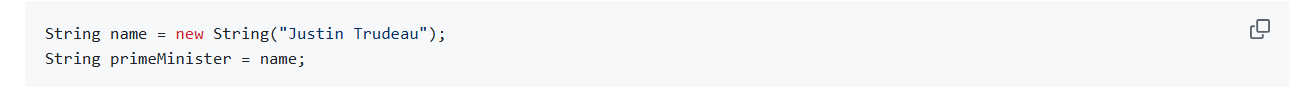
1 别名和它的作用

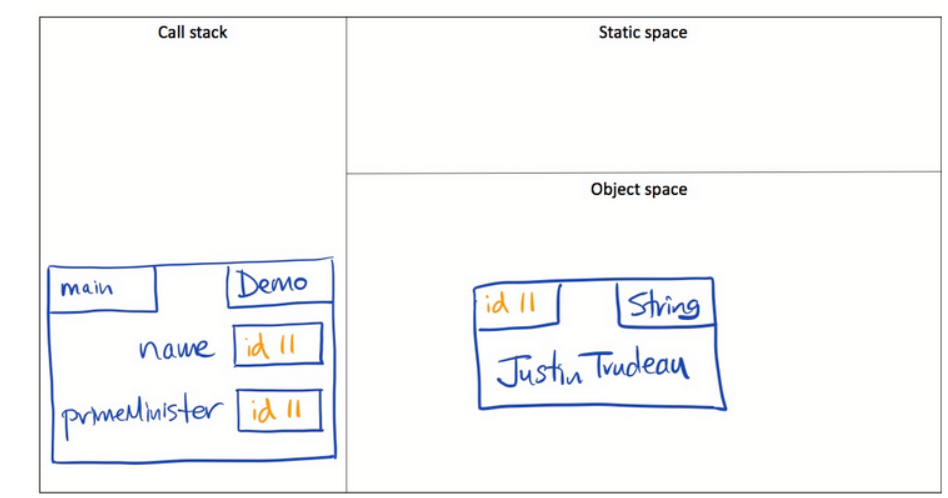
我们已经了解，Java 有两种类型：原始类型（例如 int），它直接存储值；引用类型（例如 String），它存储对对象的引用。我们必须通过引用才能找到对象中存储的值和方法。

为了了解代码的执行情况，我们必须始终注意变量是原始类型还是引用类型。这可能会彻底改变代码的执行情况。

2 通过引用来创建别名

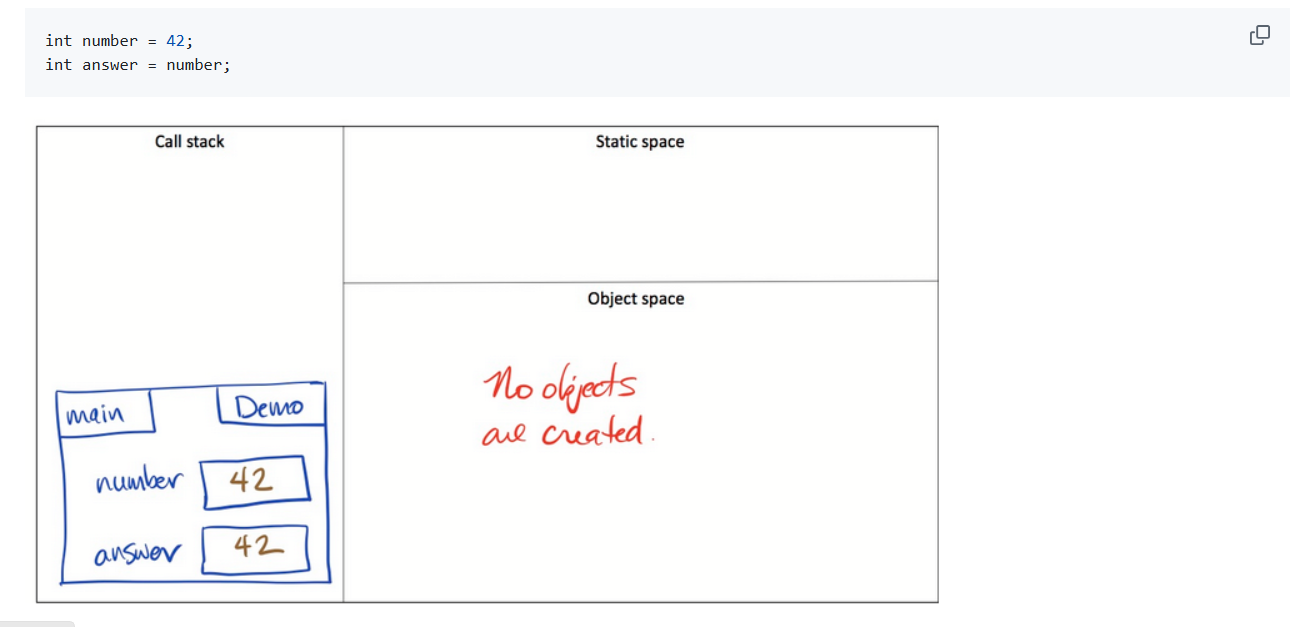
对于引用数据类型，别名相当于指向heap中同一个地址





3 别名无法用于原始类型

由于原始数据类型没有创建新的对象在heap中，闯将别名的行为类似于创建一个数据完全相同的新变量



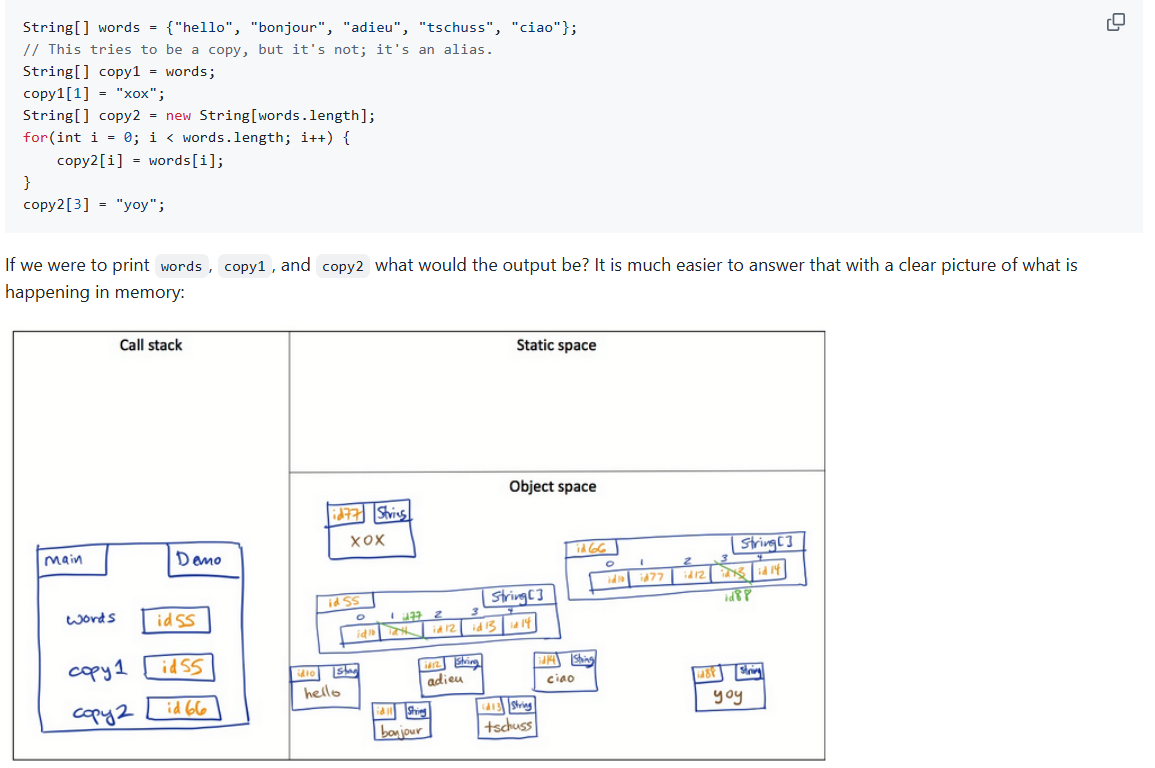
4 别名的作用

对于其中一个别名的修改会影响另外一个别名的数据。由于别名实际上是指向同一个被创建出来的对象

5 别名作用与不可变类

不可变类如String使用别名没有意义，由于类的实例无法被修改，因此无法通过别名修改heap中的对象。因此最好的解决方法是创建一个新的对象。

Extra：使用copy来避免别名的影响（如果不希望使用别名，使用new来创建新的变量）

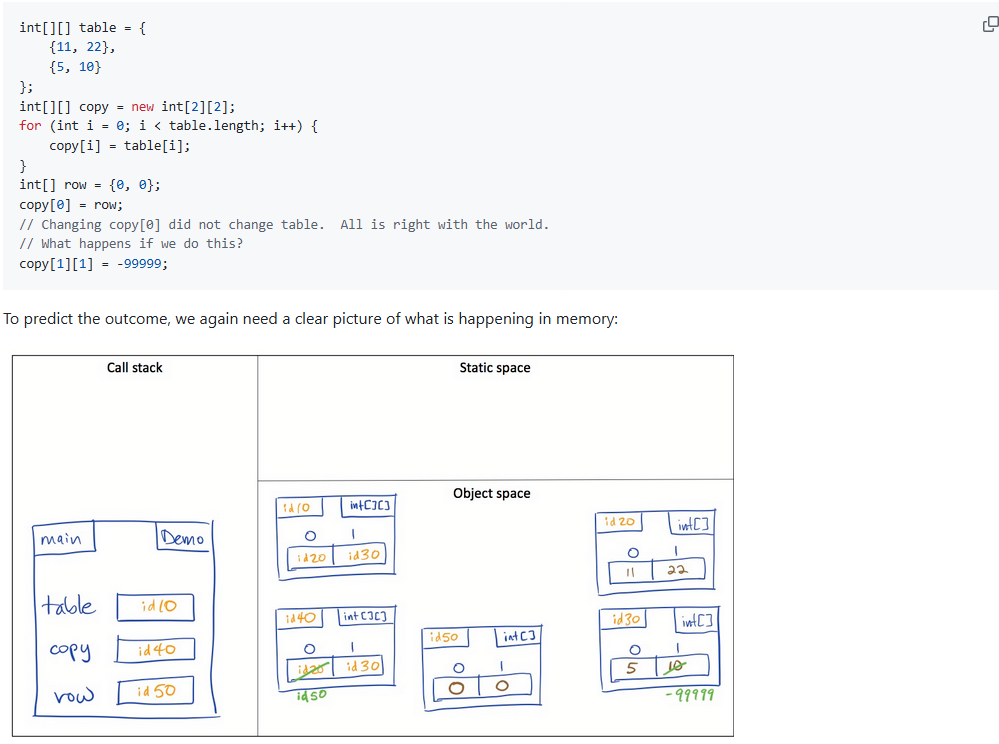


6 浅拷贝和深拷贝

将数据的引用进行赋值的行为被称为浅拷贝，例如上文通过for loop将words中的元素拷贝到copy2中

拷贝对象以及其所有的引用对象，从而让新旧对象完全独立的拷贝行为被称为深拷贝：

使用row对copy[0]进行赋值是一种深拷贝，因为这个行为让table和row完全独立了，反之对copy[1][1]则会影响原对象



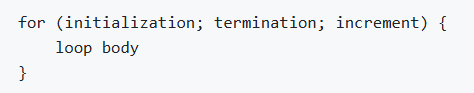
（进行拷贝时需要时刻注意哪些拷贝行为是在进行引用赋值，那些是在直接修改某个引用下的数据）

控制结构（Control Structure）

1 if判断结构组

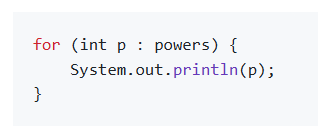
If, else if, else语句组，与python没有太大区别，须将boolean expression装入括号中

2 for循环组



所有的statement都可以不在括号中进行定义，只要符合顺序（没出现variable not declare的情况）

Extra：for each循环（for i in array）



++, ==, +=, =+:

x+=n 等价于 x = x + n

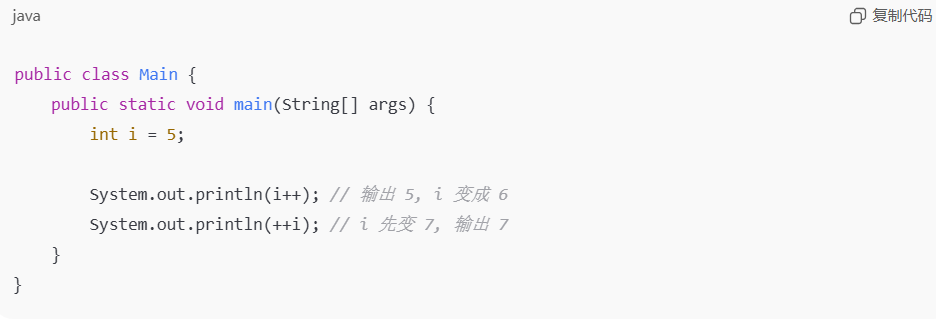
y-=n 等价于 y = y – n

i++ 等价于 i = i + 1

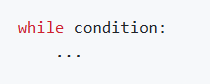
i – 等价于 i = i – 1

++i 在i使用前就增加

i ++ 在i使用后再增加

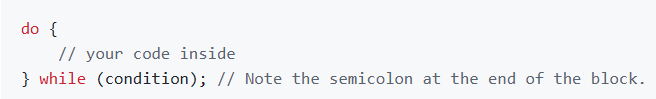


3 while循环组

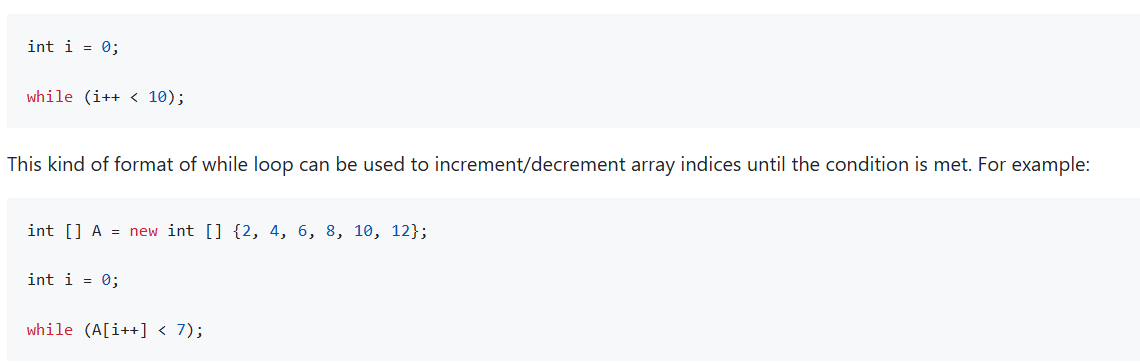


4 do…while循环组

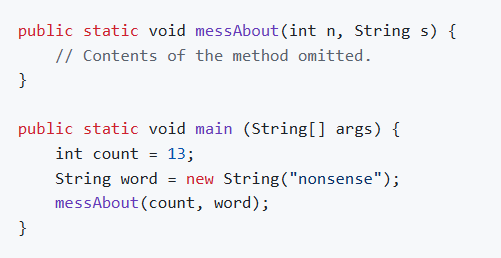
循环中的内容至少会运行一次



结合i++/++i的特殊用法：

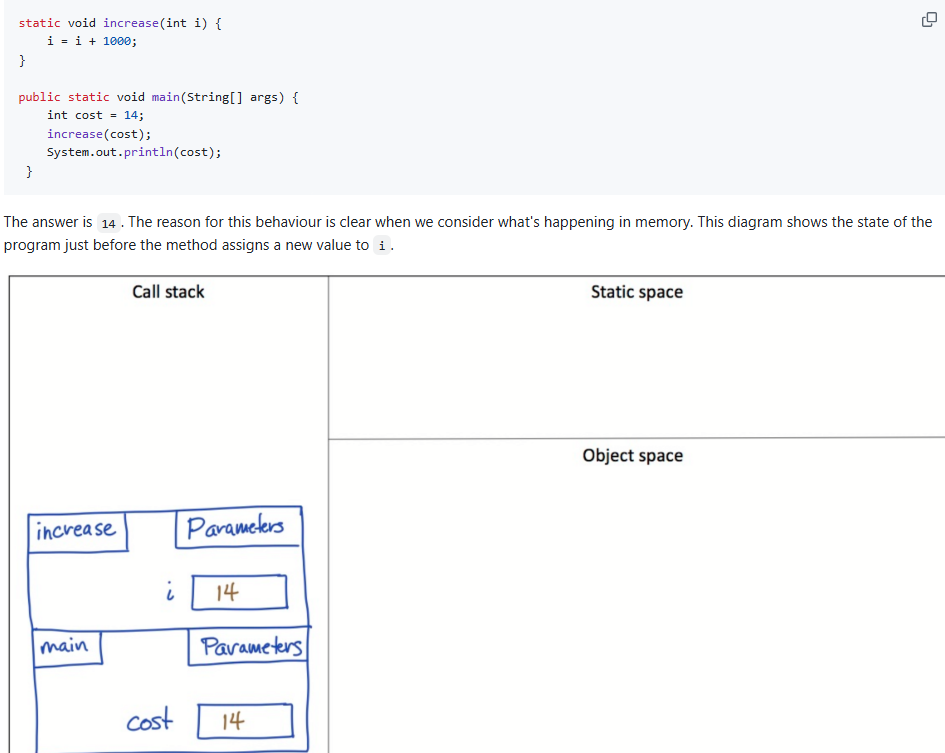


参数



括号内定义的变量就是参数parameter，所有的参数被称为argument

1 通过参数传递一个原始变量



注意事项：

1. 传递参数相当于是创建一个传递变量的别名，因此对于可变对象来说可能会导致不期望的修改
2. 如果传递的是一个不可修改对象，不用担心方法导致的不期望值修改

**M2 Java中的类Class**

类中的变量

对于Java来说，类中的变量主要有两类：

实例变量：属于实例的变量

类变量：属于类本身的变量，被所有实例所拥有。

可见与访问修饰符access modifier



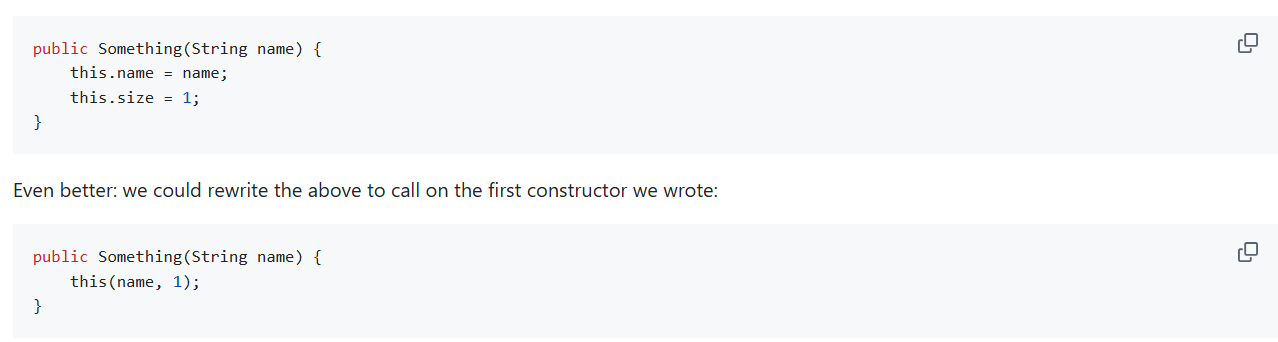
构造函数Constructor

类似于Python的\_\_init\_\_

此方法不会返回数据，在实例被创建时调用

在Java中，使用this.进行对此类/实例的变量的指代

使用实例与技巧：



对象的创建流程

分配对象的内存

实例变量初始化和默认赋值

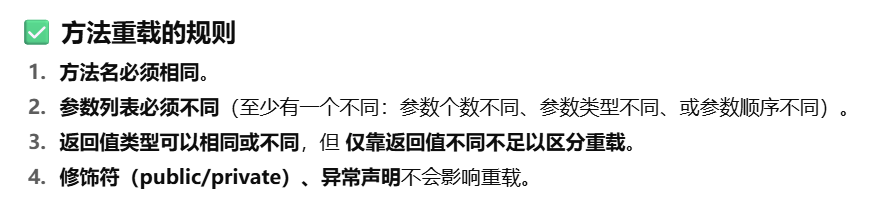
实例变量基于语句赋值（例如int i = 1）

调用构造函数

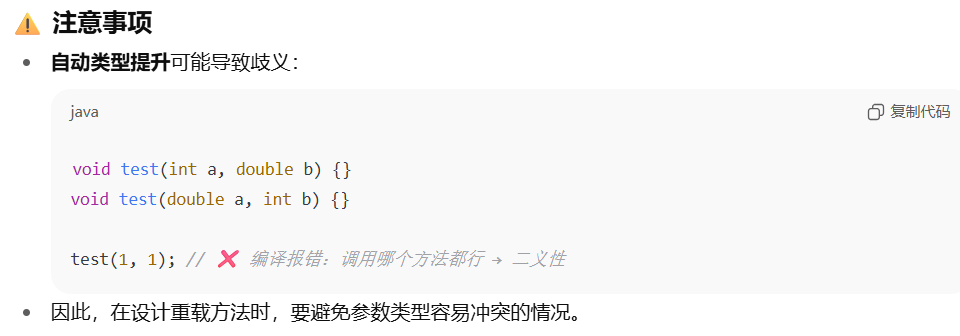
如果是继承类，优先调用父类的构造方法

方法重载Overloading（区别与Override）

可以定义同名但是不同argument的方法，此行为被定义为Overloading



注意，返回类型不包含在argument内



方法重写Override

使用@Override在类名前，可以让新的内覆盖赛旧内，常见于需要将父类方法替换为新的方法实现不同功能

常见的一些特殊的方法重写：

toString()

此方法返回一个String，通常用于返回某个方法的一些基本信息（注意，当变量被直接打印时，会调用该方法）

Equals(<object>)

没有重写，默认比较引用

此方法定义了一种与其他方法比较的方法，但是不是定义==符号作用于此对象的方法。（==的本质时计算两个变量的id是否相等，而equals可以被定义为任何比较，例如某个具体数值的比较）

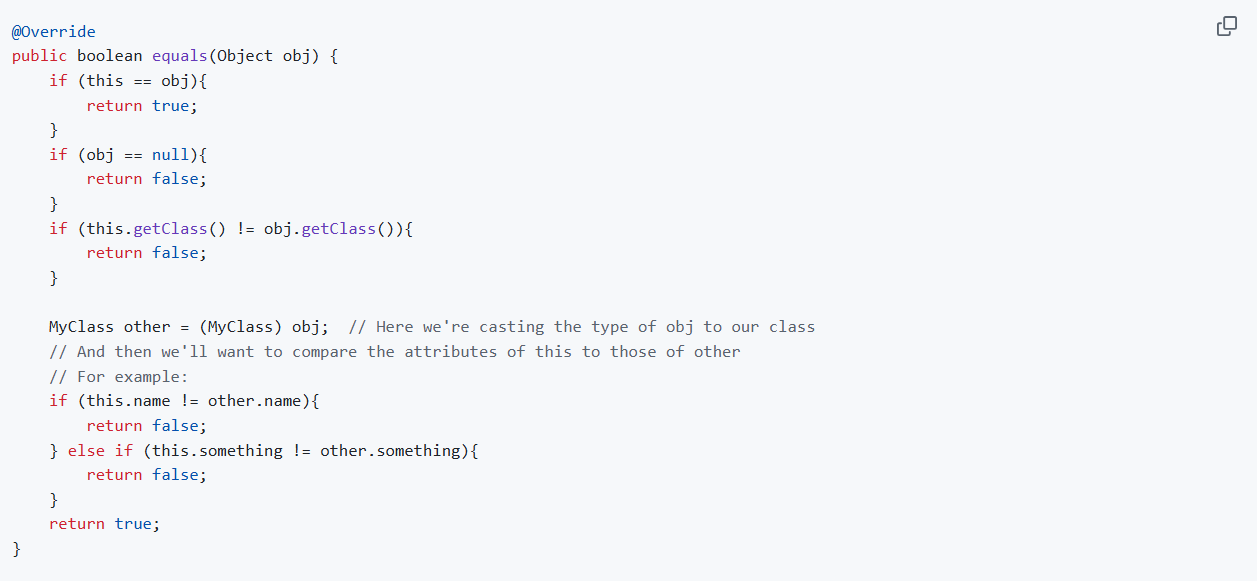
对此方法进行重写时一般需要注意一下三点：

Symmetric：确保a.equals(b)和b.equals(a)的输出一致

Reflexity: a.equals(a)一定正确

Transitivity: 如果a.equals(b)和b.equals(c)，那么a.equals(c)

以下为常见实例：



hashCode()

返回一个int，此方法通常与equals一起重写

对于两个相同（equals）的对象，他们需要有相同的hash code

但是对于两个有相同hash code的对象，它们不一定相同

此方法的使用是为了更高效的使用hash类结构进行数据查找，因此针对hashcode可能有一些优化方法

静态方法static

定义属于类的方法，不可以包含实例方法在方法主题中

关键词final

限制关键词，定义无法被重新赋值的变量。

如果用于定义一个对象，只可以保证对象引用id不变，不可以保证引用对象本身的内部变量不可变。

M2-1 类的关系

！以下为ai总结内容，非实际阅读

我来帮你总结一下你上传的内容（Chapter 3: Relationships between Classes）👇

**总结**

**1. 继承 (Inheritance)**

* Java 使用 extends 关键字定义子类。
* 子类继承父类的属性和方法，是父类的一种实例。
* **抽象类 (abstract class)**
  + 用 abstract 修饰，不能直接实例化。
  + 可以包含抽象方法（无方法体），子类必须实现。
* **方法重写 (Overriding)**
  + 子类用 @Override 重新定义父类方法。
  + 注解不是必须的，但能避免错误。

**2. 接口 (Interfaces)**

* 接口定义了一组行为契约，类可以 implements 一个或多个接口。
* 接口的特点：
  + 方法默认 public abstract。
  + 变量默认 public static final，不能有实例变量。
* 从 **Java 8** 开始：
  + 接口可包含 **default 方法**（带实现）。
  + 接口可包含 **static 方法**（实用工具方法）。
* 一个类可以实现多个接口；接口之间也能 extends。

**3. super 关键字**

* 调用父类构造器：super() 或 super(args)，必须写在子类构造器第一行。
* 如果子类构造器不写 super，编译器会默认插入 super()。
* 如果父类没有无参构造器，而子类没写 super(args)，则会报错。

**4. 多态 (Polymorphism)**

* 多态允许对象以不同形式存在。
* 一个对象既是其本身类型，也属于父类、接口类型。
* instanceof 检查对象是否可以安全转换为某个类型（类或接口）。
* 通过编程时面向父类或接口，提高代码的复用性和扩展性。

**5. 类型转换 (Casting)**

* **上转型 (Upcasting)**：子类对象赋值给父类引用（自动、安全）。
* **下转型 (Downcasting)**：父类引用强制转为子类（需小心，可能抛 ClassCastException）。
* 可以用 instanceof 检查后再下转型。
* **基本类型转换**：可能有精度丢失（如 double 转 int 会截断）。

**6. Comparable 接口**

* 使对象具备排序能力。
* 必须实现 compareTo(T o) 方法：返回负数 / 0 / 正数表示小于、等于、大于。
* 常用于 Arrays.sort() 或集合排序。
* 要求同一数组中的元素必须是“可相互比较”的。
* 作者自己写的类可实现 Comparable，例如 Review implements Comparable<Review>。

**7. Comparator 接口**

* 当一个类需要多种排序方式，或类定义不可改时使用。
* 实现 compare(T o1, T o2) 方法。
* 例如：
  + LikesComparator → 按点赞数排序。
  + TextComparator → 按文本长度排序。
* Arrays.sort(arr, new ComparatorClass()) 可使用比较器排序。

✅ **总结要点**

* extends 继承类，implements 实现接口。
* 抽象类可有抽象方法，接口方法默认抽象（Java 8 可有 default/static）。
* super 必须出现在子类构造器第一行。
* 多态使对象能被父类/接口引用。
* 上转型安全，下转型需 instanceof 检查。
* Comparable 用于定义“自然顺序”，Comparator 用于自定义/多样化排序。

要不要我帮你整理一张 **“抽象类 vs 接口 vs 普通类” 对比表格**，让你更快区分它们的特性？

M2-2 基于Swing的Java GUI介绍

！以下为ai总结内容，非实际阅读

我来帮你总结一下你上传的 **Chapter 4: Graphical user interfaces in Java** 的内容 👇

**总结**

**1. Swing 基础**

* Swing 是 Java 提供的 GUI 库，主要类都在 javax.swing 包下。
* 所有 UI 元素都是 **组件 (component)**，例如：
  + JFrame：窗口
  + JLabel：标签
  + JTextField：文本框
  + JButton：按钮
  + JPanel：容器（可以嵌套）

常见约定：大多数 Swing 组件类名以 **J** 开头。

**2. 创建窗口**

* 使用 JFrame 创建窗口：
* JFrame frame = new JFrame("Example");
* frame.setMinimumSize(new Dimension(300,200));
* frame.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);
* frame.pack();
* frame.setVisible(true);
* 必须用 SwingUtilities.invokeLater(...) 来启动 GUI 代码，保证线程安全。

**3. 组件与布局**

* JPanel 可作为容器，默认是水平布局。
* 可以用 BoxLayout 设置垂直排列：
* JPanel mainPanel = new JPanel();
* mainPanel.setLayout(new BoxLayout(mainPanel, BoxLayout.Y\_AXIS));
* 常用方式：通过嵌套 JPanel 来组织复杂界面。
  + 一个 JPanel 放 label 和 text field。
  + 一个 JPanel 放按钮。
  + 主 JPanel 嵌套所有子面板，再放入 JFrame。

**4. 事件驱动 (Event-driven)**

* GUI 是 **事件驱动** 的：组件发生事件时，Java 会创建事件对象并调用你定义的 **事件监听器**。
* 例如按钮点击：
* JButton submit = new JButton("Submit");
* submit.addActionListener(new ActionListener() {
* @Override
* public void actionPerformed(ActionEvent e) {
* String name = textField.getText();
* JOptionPane.showMessageDialog(null, "Hello " + name);
* }
* });
* 这里的 new ActionListener() {...} 定义了一个 **匿名类**，实现了 actionPerformed 方法。

**5. 实践练习**

* 给 **Cancel 按钮** 添加监听器，调用 setText("") 来清空文本框。
* JButton cancel = new JButton("Cancel");
* cancel.addActionListener(e -> {
* firstNameField.setText("");
* lastNameField.setText("");
* });

**🔑 核心要点**

* javax.swing 包含大部分可视化类。
* 用 JFrame 创建窗口，用 JPanel 组织组件。
* BoxLayout 或嵌套面板可以实现灵活布局。
* 事件监听器 (ActionListener) 实现交互逻辑。
* JOptionPane 可用于弹出对话框。

要不要我帮你画一张 **Swing 组件层级和事件流图**，把 JFrame → JPanel → JButton/JTextField 以及事件监听的流程直观展示出来？