Root

🔷 计算机存储信息的方式
🔷 The Golden Rule of Equals (GRoE)

Given variables y and x:
y = x copies all the bits from x into y.

所有数据都存储为二进制（0 和 1）。
计算机内存中，每个 数值、字符、布尔值 都是以 二进制位（bit） 存储的。
📝 示例
数据类型 具体值 二进制表示
整数 72 01001000
浮点数 205.75 01000011 01001101 11000000 00000000
字符 'H' 01001000 （与 72 的二进制相同）
布尔值 true 00000001

📌 关键点：
字符 'H' 和数字 72 的二进制表示一样，因为 'H' 在 ASCII 码中对应 72。
浮点数的存储更复杂，遵循 IEEE 754 浮点数标准（涉及指数、尾数等部分）。
布尔值 true 可能存储为 00000001，但不同机器可能有不同实现。

🔷 Java 变量存储原理

计算机会分配固定大小的空间
整数 (int): 分配 32 位（4 字节）
浮点数 (double): 分配 64 位（8 字节）
Java 维护一个 内部映射表（symbol table），将变量名称映射到内存地址。
Java 不会自动初始化局部变量
Java 不会在内存中填充默认值（不像 Python 那样）。
使用未初始化的变量会导致编译错误（安全性机制）。

Java 的 "Pass by Value"（按值传递）！🐶✨

这是 Java 里非常重要的一个概念，很多初学者会误解 Java 是 Pass by Reference，但其实它完全是 Pass by Value！
🐶 什么是 "Pass by Value"？
在 Java 里，所有方法调用时，参数都是 "按值传递"，这意味着：
基本数据类型（int, double, char, boolean, etc.）
直接传递它们的"值"，方法内部的修改不会影响原变量。
对象（Object, Arrays, Lists, etc.）
对象变量存的是"引用地址"，传递的是这个地址的值，而不是对象本身！
方法内部的修改会影响对象的内容，但不会改变对象的引用地址。
🐶 为什么 walrus.weight 变了，而 x 没变？
传递类型 传递内容 方法内修改影响？ 例子
基本类型 (int, double, boolean) 直接传值（副本） ❌ 不影响原始变量 x = x - 5;
对象 (Object, Array, List) 传递对象引用的值（地址） ✅ 影响对象的内容 W.weight = W.weight - 100;
对象引用（但重新赋值） 传递对象引用的值（地址） ❌ 不影响原始对象地址 W = new Walrus(5000, 12.0);
🐶 Java 不是 "Pass by Reference"！
很多人会误解 Java 是 "Pass by Reference"，但其实不是！
在 C++（真正的 Pass by Reference 语言），如果 W 是一个对象引用，我们可以在 doStuff() 里直接修改 W 指向新的对象，main() 里的 walrus 也会变！
但在 Java 里，W = new Walrus(5000, 12.0); 只会修改 W 的本地副本，不会影响 main() 里的 walrus！🚫

🔷 Java 的 8 种基本数据类型

数据类型 关键字 描述
字节 byte 8-bit 整数 (-128 到 127)
短整型 short 16-bit 整数 (-32,768 到 32,767)
整型 int 32-bit 整数（常用，默认）
长整型 long 64-bit 整数
单精度浮点 float 32-bit 浮点数
双精度浮点 double 64-bit 浮点数 (默认)
布尔型 boolean true / false (占 1 bit，但存储可能是 1 字节)
字符型 char 16-bit Unicode 字符
📌 关键点：
整数类型 默认是 int，浮点数默认是 double。
boolean 只有 true/false，但实际存储可能占 1 字节（实现相关）。
char 使用 Unicode 编码，占 2 个字节，可以表示中文等字符。

数组（Array）作为对象 在 Java 里的 声明（Declaration） 和 实例化（Instantiation）

在 Java 里，数组（Array）本质上是一个对象！

1. 声明（Declaration）： 只是创建了一个"盒子"来存数组的引用，并没有真正分配内存。

什么是"声明"？ int[] a;
🚀 这一步只是告诉 Java：
"我要一个 int 类型的数组变量 a！"
但Java 还没有为这个数组分配内存！
a 只是个 "64 位的引用"，用来存放未来数组的地址。(🚫 但它目前是 null！)
🔴 注意：
int[] a;
System.out.println(a[0]); // ❌ 会报 NullPointerException，因为 a 还没有被实例化！

2. 实例化（Instantiation）： 通过 new 关键字，==真正创建数组对象，并分配内存。

int[] a; // 声明 (Declaration)
a = new int[]{0, 1, 2, 95, 4}; // 实例化 (Instantiation)
什么是"实例化"？
a = new int[]{0, 1, 2, 95, 4};
💡 这时候 Java 真的分配了一块内存！
new int[]{0, 1, 2, 95, 4} 创建了一个包含 5 个元素的数组。
a 现在指向这个数组的地址！

3. Assignment

int[] a = new int[]{0, 1, 2, 95, 4};
● Creates a 64 bit box for storing an int array address. (declaration)
● Creates a new Object, in this case an int array. (instantiation)
● Puts the address of this new Object into the 64 bit box named a. (assignment)
Note: Instantiated objects can be lost!
● If we were to reassign a to something else, we'd never be able to get the original Object back!

一步完成"声明 + 实例化"：省略new

int[] x = new int[]{0, 1, 2, 95, 4};
💡 简化版（更常用）：int[] y = {0, 1, 2, 95, 4}; // 省略 new int[]
💖 注意：
只有在声明时，{} 这种简写方式才有效！
如果分开赋值，必须显式写 new int[]：
int[] z;
z = {0, 1, 2, 95, 4}; // ❌ 错误！
z = new int[]{0, 1, 2, 95, 4}; // ✅ 正确！

💥 数组的核心特点：
固定长度（创建后不能变长或变短）。

连续的内存空间，每个元素占用相同的字节大小。
索引从 0 开始，最后一个索引是 length - 1。
存储相同类型的值（例如整数数组、字符数组等）。
💡 你可以把 数组 想象成 N 个排列整齐的储物柜，每个柜子只能放一种类型的东西，而且柜子的数量不能改变！ 🚪🚪🚪
📌 数组的组成
固定整数 length → 一旦创建，长度就无法更改！ （不像 ArrayList 可以动态扩展）
一串连续的 N 个内存单元 → N = length
每个单元存相同类型的数据
索引从 0 到 length-1
比喻： 假设你在超市租了一排 5 个冰箱，每个只能存放饮料，你不能突然增加冰箱或者减少冰箱的数量，每个格子都要放同种类型的饮料。🥤🥤🥤🥤🥤

🔥 "数组的数组"

也就是 Java 里的二维数组（jagged arrays/稀疏数组），重点是：
Java 的二维数组是"数组的数组"，即 数组中的每个元素本质上是一个引用，指向另一个数组。
数组的行可以有不同的列数（jagged array），不像普通的 `int[][] matrix = new int[4][4];` 那样严格的二维数组。 理解 new int[4][] 和 new int[4][4] 的区别：
new `int[4][]` 只创建了存放 4 个数组引用的"框架"。 new int[4][4] 则完整创建了 4 行 4 列的数组。
int[][] pascalsTriangle;
`pascalsTriangle = new int[4][];`
📌 这一行代码的作用：
创建了 pascalsTriangle （一个可以存放 4 个数组引用的变量）。
但每一行的数组还没有初始化，它们都只是 null。
这意味着 pascalsTriangle 的列数是不固定的，可以每行有不同的元素数量。
🔹 2.1 初始化不同行的数组
pascalsTriangle[0] = new int[]{1}; // 第一行 1 个元素
pascalsTriangle[1] = new int[]{1, 1}; // 第二行 2 个元素
pascalsTriangle[2] = new int[]{1, 2, 1}; // 第三行 3 个元素
pascalsTriangle[3] = new int[]{1, 3, 3, 1}; // 第四行 4 个元素
📌 这里做了什么？
pascalsTriangle[0] = new int[]{1};
创建一个只有 1 个元素的数组 {1}，然后把它存到 pascalsTriangle[0] 里。
pascalsTriangle[2] = new int[]{1, 2, 1};
创建一个 {1, 2, 1} 的数组，存到 pascalsTriangle[2]。
💡 这个数组并不是一个严格的矩形，而是"锯齿状"的（jagged array），每一行的长度可以不同！
🔹 2.2 访问和修改元素
int[] rowTwo = pascalsTriangle[2]; // rowTwo 现在指向 {1, 2, 1}
rowTwo[1] = -5; // 把 2 改成 -5
🔹 2.3 另一种二维数组的创建方式
int[][] matrix;
`matrix = new int[4][]; // 创建一个 4x? 的数组，列数未定义` matrix = new int[4][4]; // 重新创建一个 4x4 的完整矩阵
📌 区别：
`new int[4][]; 只创建了一个可以存放 4 行的数组框架，但每一行的数组还没有初始化。` new int[4][4];
创建了完整的 4x4 矩阵，每行都有 4 个元素，默认值为 0。
🔹 2.4 另一种初始化方式
`int[][] pascalAgain = {`
{1},
{1, 1},
{1, 2, 1},
{1, 3, 3, 1}
};
📌 作用：
直接创建并初始化一个 jagged array，等价于前面 pascalsTriangle 的手动赋值过程。
适用于已知数组内容的情况，更加简洁。

数组 vs. 类（Arrays vs. Classes）

1. 访问方式
数组（Array） 使用 [] 访问元素。例如：
int[] x = {100, 101, 102, 103};
int value = x[2]; // 访问索引 2 处的元素，得到 102
类（Class） 使用 .（dot notation）访问属性。例如：
Planet p = new Planet(6e24, "earth");
double massValue = p.mass; // 访问 `mass` 属性

2. 存储的数据类型
数组：只能存储相同类型的数据（同一个数组中的所有元素必须是相同的数据类型）。
类：可以存储不同类型的数据。例如，一个 Planet 对象可以同时存储 double 类型的 mass 和 String 类型的 name。

3. 数据结构
数组：是一组相同类型的值的集合，每个元素占据一个固定大小的存储空间，索引从 0 开始。
类：类是对象的模板，可以包含多个属性，每个属性可以是不同的数据类型（例如 int、double、String）。

4. 存储方式
数组：数组变量存储的是数组的引用，数组元素存储的是数据本身。
类：类的对象存储的是对象的引用，对象内部存储的是各个属性的数据。

5.数组索引可以在运行时计算
Array indices can be computed at runtime.
数组索引可以在运行时计算。
int[] x = new int[]{100, 101, 102, 103};
int indexOfInterest = askUser(); // 运行时获取索引值
int k = x[indexOfInterest]; // 使用用户输入的索引访问数组
System.out.println(k);
类的成员变量各不能在运行时计算
Class member variable names CANNOT be computed and used at runtime.
类的成员变量名不能在运行时计算和使用。
String fieldOfInterest = "mass";
Planet earth = new Planet(6e24, "earth");
double mass = earth.fieldOfInterest; // ❌ 错误！
System.out.println(mass);
📌 代码的错误
String fieldOfInterest = "mass";
试图用变量存储属性名。但 Java 不支持这样访问属性。
double mass = earth.fieldOfInterest;
错误！earth 没有 fieldOfInterest 这个字段，它只有 mass 和 name。
Java 不允许 用字符串变量来动态访问对象的字段。
📌 运行时报错
ClassDemo.java:5: error: cannot find symbol
double mass = earth.fieldOfInterest;
^
symbol: variable fieldOfInterest
location: variable earth of type Planet
Java 不允许 earth.fieldOfInterest 这样动态访问属性，必须明确写出 earth.mass 或 earth.name。