原生数据类型（基本类型）和引用数据类型（对象）

- 只要能找到对应的类，就是引用数据类型
- 否则就是原生数据类型

🧠 Java 中的数据类型分为 两类

- ✅ 原生数据类型 Primitive Types ：直接存值，性能高，位于 stack
- ✅ 引用数据类型 Reference Types ：存的是地址，指向 **heap** 中的对象

🧩 Java 的 8 种 primitive types

类型	占用字节	示例值	用途
byte	1 字节	-128 ~ 127	整数，节省空间
short	2 字节	-32,768 ~ 32,767	整数
int	4 字节	-21亿 ~ 21亿	默认整数类型
long	8 字节	极大整数	大数据计算
float	4 字节	3.14f	小数，低精度
double	8 字节	3.1415926	默认小数类型，高精度
char	2 字节	'A', '中'	存储单个字符（使用 Unicode 编码）
boolean	1 字节*	true / false	逻辑判断

> 注意：Java 虽然说 boolean 是 1 bit，本质还是以 1 字节对齐存储

✅ 原生类型的特点

特性	说明
存储位置	**stack/ 对象字段中**
存储内容	真实的数值
速度	快，效率高
是否支持 null	❌ 不可以为 null
是否有方法	❌ 没有方法（非对象）

🧩 引用数据类型 Reference Types

包括：

- **对象**
- **数组**
- **接口**
- **枚举**
- **String（字符串）**
- **包装类**（如 `Integer`，`Double`，`Boolean` ）

```
String name = "Alice";    // String 是引用类型
int[] numbers = {1, 2, 3}; // 数组是引用类型
Person p = new Person();   // Person 是引用类型
```

✅ 引用类型的特点

特性	说明
存储位置	引用变量在栈，实际对象在堆
存储内容	栈里是**地址**（引用），指向堆中的对象
速度	比原生慢一些（需要间接访问）
是否支持 null	✅ 可以为 null
是否有方法	✅ 是对象，有方法可调用

🖼️ 图示理解：

```
int x = 5;
String name = "Alice";
```

```
 栈 Stack                堆 Heap

x: 5
name:0xA1A1      👉      name对象: "Alice"
```

- `x` 存的是数值 5，直接在栈中
- `name` 是引用变量，存的是地址（0xA1A1），这个**地址指向堆中的 `"Alice"` 字符串对象**

🔁 包装类型 Wrapper Classes

Java 为每个原生类型都提供了一个**对应的类**

原生类型	包装类
`int`	`Integer`
`double`	`Double`
`char`	`Character`
`boolean`	`Boolean`
`...`	`...`

这些包装类是引用类型，有很多方法，比如：

```
Integer x = 5;
x.toString();  // "5"
```

计算机底层 float 存储原理

```
public class FloatTest {
    public static void main(String[] args) {
```

```
        float a = 0.1f;
        System.out.println(a);
        // 输出不是精确的 0.1，而是 0.10000000149...
    }
}
```

🧠 0.1 转成二进制时发生了什么？

```
0.1 十进制 = 0.0001100110011001100110011001100...(无限循环) 二进制
```

它是一个 **无限循环的二进制小数**，就像 1/3 在十进制中是 0.3333... 一样

🚫 但 float 的尾数位只有 23 位，必须截断：

这意味着：

- 计算机只能存下 **“0.1”最前面那几位二进制**
- 所以只能存一个 **接近 0.1** 的值
- 存储结果 ≈ 0.10000000149011612 （不是 0.1）

✅ float double 只能“近似”表示小数，不是精确表示

不能期待 `double` 精确地表示 任意小数，只是它误差更小

📌 `double` 只是比 float 更接近（精度更高而已）：

类型	位数	精度	示例
float	32 位	~6–7 位有效数字	0.1f ≈ 0.10000000149
double	64 位	~15–16 位有效数字	0.1d ≈ 0.1000000000000001

🔍 浮点误差影响程序

很多人会写：

```
if (a == 0.1f) { ... }
```

这是非常危险的写法，因为：

- `a` 存的其实是 0.10000000149
- `0.1f` 是编译时转的另一个近似值
- 两个“看似一样”的值，其实不一样 ❌

✅ 正确做法：

```
if (Math.abs(a - 0.1f) < 1e-6) {
...
}
```