Java 中的对象变量本质上是地址（引用）

在 Java 中：

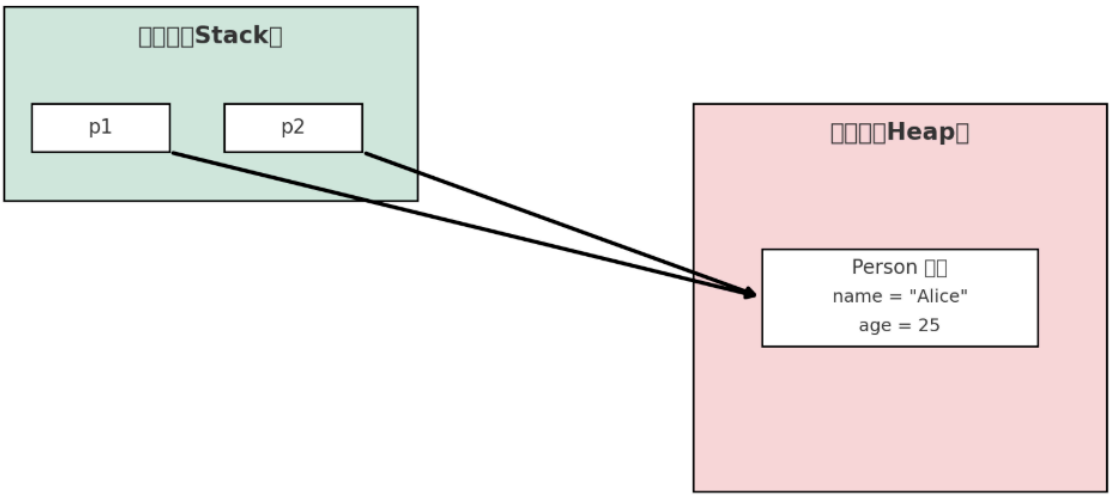
> ✅ 对象变量存的是“地址”，而不是实际的对象本身

这个地址指向👉真正的对象在 **堆（Heap）内存** 中的位置

```
public class Person {
    String name;
    int age;

    public Person(String n, int a) {
        name = n;
        age = a;
    }

    public void sayHi() {
        System.out.println("Hi, I'm " + name);
    }
}

public class Main {
    public static void main(String[] args) {
        Person p1 = new Person("Alice", 25);
        Person p2 = p1;  👇
    }                在堆中开辟空间放对象
}
```

🧩 重点！！！

步骤	内存中发生的事	说明
`new Person`	在堆内存中开辟空间，创建对象	真正的对象在堆（Heap）中
`Person p1 =`	在栈（Stack）中创建一个变量 `p1`，它**保存的是这个对象的在堆中的地址**	`p1` **不是对象本身，而是一个引用**
`Person p2 = p1;`	新建一个变量 `p2`，复制了 `p1` 的地址	`p2` 和 `p1` 指向同一个对象



✅ 所以说：

- 在 Java 中，**类的对象变量保存的是地址（引用）**
- 对象本体在 **堆内存（heap）** 中
- 变量在 **栈内存（stack）** 中保存一个地址，指向堆上的对象
- 当你写 `p2 = p1;`，其实是把地址复制了一份，不是复制了一份对象

🧨 验证

```
p1.name = "Bob";
System.out.println(p2.name);  // 输出是 "Bob"
```

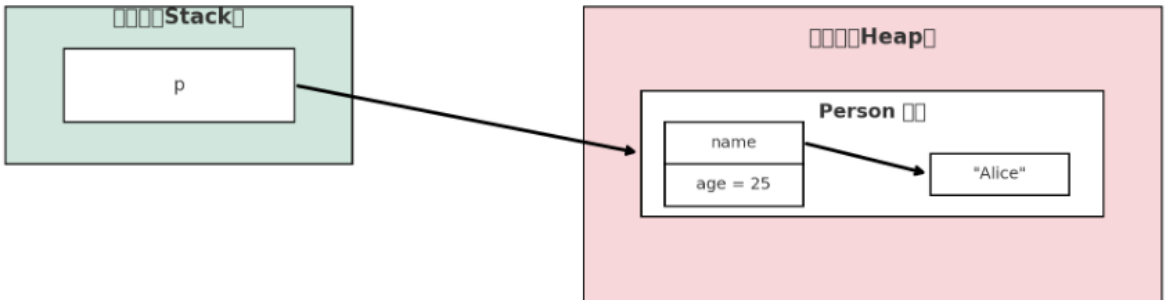
因为 `p1` 和 `p2` 指向的是同一个对象，所以修改 `p1` 的内容会影响 `p2`

❗成员变量不一定是地址

成员变量的类型	是否存地址？
基本数据类型（primitive type）	❌不是地址，直接存**值**
引用类型（reference type）	✅是地址，存的是对象的“地址“

❗❗❗❗❗内存图解

```
public class Person {
    String name;    // String是引用类型，存的是地址
    int age;        // int是基本类型，存的是值
}

    Person p = new Person();
    p.name = "Alice";
    p.age = 25;
```



- **栈内存（Stack）：**
  - 保存的是变量 `p`（一个引用变量）
  - `p` 存的是地址，指向右侧堆内存中的 `Person` 对象

- **堆内存（Heap）：**
  - `Person` 对象本体存放在这里
  - 里面的成员变量：
    - `age = 25`：是基本类型，直接存值
    - `name`：是引用类型（`String`），存的是 `"Alice"` 字符串对象的地址
  - `"Alice"` 字符串是另一个对象，也存在堆中，和 `Person` 是两个独立对象

成员变量	类型	存的是什么	举例内容
`age`	`int`	直接存的 25（值）	`age: 25`
`name`	`String`	存的 "Alice" 这个字符串的地址	`name: 0x123456`

📌 String 也是引用类型

虽然你写 `p.name = "Alice"` 看起来像是直接赋值，其实 `"Alice"` 会在堆中作为一个 `String` 对象存在，`p.name` 存的是这个对象的地址

🧠 name 是引用类型的成员变量，那它是“地址”
既然地址通常存在 stack 中，那为什么它会在 heap中

✅ 核心回答：

> 因为 name 是 成员变量，属于对象（Object）的一部分
>
> **对象是分配在 heap 中的，所以它的成员变量（无论是值还是地址）都在堆中**

> ✔️ 成员变量都跟着对象走，在堆里
>
> ✔️ **局部变量都在栈中，包括对象引用变量本身**