基本数据类型的装箱 Boxing 与拆箱 Unboxing

- Boxing 和 Unboxing 用于桥接 **基本类型** 与 **引用类型** 之间的转换
- Wrapper classes provide a way to **use primitive data types as objects.**
- Sometimes you must use wrapper classes, for example when working with Collection objects, such as `ArrayList`, where primitive types cannot be used (the list can only store objects):

```
ArrayList<int> myNumbers = new ArrayList<int>(); // Invalid
```

```
ArrayList<Integer> myNumbers = new ArrayList<Integer>(); // Valid
```

> 装箱（Boxing）：把**基本类型**转换成对应的**包装类对象**
>
> 拆箱（Unboxing）：把**包装类对象**转换成对应的**基本类型值**

▼ 🧩 基本数据类型和对应包装类的对照表

基本类型	包装类（引用类型）
int	Integer
double	Double
boolean	Boolean
char	Character
byte	Byte
short	Short

基本类型	包装类（引用类型）
long	Long
float	Float

▼ ✅ 自动装箱 AutoBoxing

```
int i = 100;
Integer integer = i;
```

本质上发生的是：

```
Integer integer = new Integer(i)  // 这一步是Java自动做的
```

▼ **每当需要 int 类型的时候, 可以直接丢 Integer 给程序,因为 Java 会自动装拆箱**

```
int i = 100;
Integer integer = i;

add ( integer );      即使函数声明的时候参数是int,也可以直接传 Integer

public static void add ( int i ){
}
```

```
public static void int add ( int i ){
      return new Integer();
   }
```

▼ 自动拆箱 unboxing

```
int val = obj;
```

等价于：

```
int val = obj.intValue();
```

▼ 常见坑：null 拆箱会抛异常

```
Integer obj = null;
int x = obj;  // ❌ 报错 NullPointerException！
```

因为你试图对一个为 null 的包装类进行 `.intValue()`，就相当于：

```
int x = obj.intValue(); // ❌ NullPointerException
```

▼ 装箱的开销

- **装箱后是对象，分配在堆中，影响性能**
- Java 会缓存一部分常用数值，减少创建对象的次数（比如 Integer 会缓存 -128 到 127）

```
Integer a = 100;
Integer b = 100;
System.out.println(a == b);  // true（指向缓存）

Integer x = 1000;
Integer y = 1000;
System.out.println(x == y);  // false（不同对象）
```

> ⚠️ == 比较的是引用地址

```
Integer c = 1000;
Integer d = 1000;
if ( c == d ) {
        system.out.println(" c=d! ")   ❌
}

if ( c.equals(d) ) {
         system.out.println(" c equals d! ")   ✅
}
```

▼ 装箱的应用场景

场景	原因
Java 集合类（如 `ArrayList`）	集合只能存对象，不能存基本类型
泛型	泛型不支持基本类型
需要对象操作	装箱类有方法可用，如 `Integer.toString()`