Java Primitive and Reference Types

Table of contents

Two categories of types 1
Primitive types 1
Reference types 2
Why the distinction matters 2
Wrapper classes 3

Introduced in 2026-02-26-course-overview-and-java-basics[1] (Lecture 1, Week 1).

Two categories of types

Java distinguishes two sorts of types:

1. **Primitive types** — built-in, fixed representations.
2. **Reference types** — everything else (i.e. classes), including `String`, arrays, and any user-defined class.

Primitive types

Type	Size	Stores
`boolean`	not specified	`true` or `false`
`byte`	1 byte / 8 bits	whole numbers from -128 to 127
`short`	2 bytes / 16 bits	whole numbers from -32,768 to 32,767

[1] courses/csse2002/2026-02-26-course-overview-and-java-basics.pdf

Type	Size	Stores
`int`	4 bytes / 32 bits	whole numbers from -2,147,483,648 to 2,147,483,647
`long`	8 bytes / 64 bits	whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
`float`	4 bytes / 32 bits	fractional numbers, accurate to 6-7 decimal places
`double`	8 bytes / 64 bits	fractional numbers, accurate to 15 decimal places
`char`	2 bytes / 16 bits	a single character/letter

The JVM does not specify a width for `boolean`.

Reference types

Reference types are everything that isn't primitive — classes, including `String` and arrays. **A variable of a reference type stores a reference to an object, not the object's data directly.** That difference is visible in the runtime memory model: a primitive's value lives in the method's stack frame, while a reference type's frame slot holds only an arrow at a heap object.

Why the distinction matters

Assignment and equality mean different things on each side. `x == y` compares stored values only for primitive types:

	Primitive types	Reference types
`x = y`	make `x` store a copy of `y`'s value	make `x` refer to the same object `y` refers to
`x == y`	check if `x` stores the same value as `y`	check if `x` refers to the **same object** as `y`
`x != y`	check if the value stored in `x` differs from `y`'s	check if `x` and `y` reference different objects

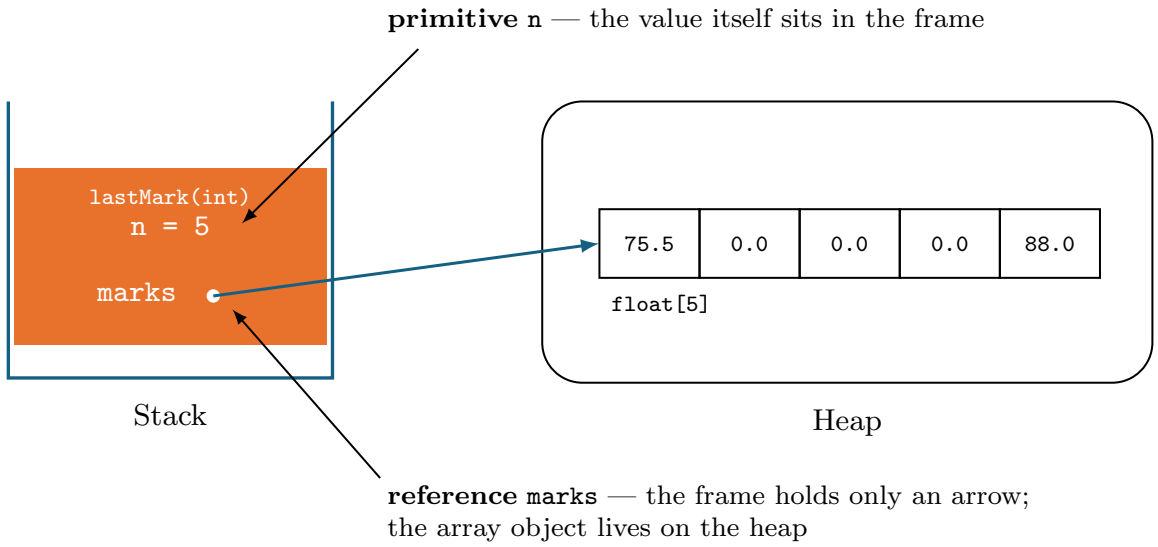

Figure 1: The difference that everything else follows from, in a trace of `lastMark(5)`: the primitive `int n` is stored *as a value* in the stack frame, while the reference `float[] marks` stores only an arrow at an object on the heap. The stack is the open-topped container, the heap is the rounded region, and the reference is the teal arrow.

So on reference types you almost always want `.equals()`, not `==` — see 2026-02-26-java-basics-part-02-collections-and-strings[2] for the `String` pooling example that makes this concrete, and for how the stack/heap split follows from it.

Wrapper classes

Collections (`Stack`, `List`, `Set`, `Map`) can only store objects/reference types, so Java provides a class for each primitive type:

Primitive	Wrapper class
`boolean`	`Boolean`
`byte`	`Byte`
`char`	`Character`
`double`	`Double`
`float`	`Float`
`int`	`Integer`
`long`	`Long`

[2] courses/csse2002/2026-02-26-java-basics-part-02-collections-and-strings.pdf

Primitive	Wrapper class
`short`	`Short`

`Stack<int>` is illegal; `Stack<Integer>` is what you write. See java-collections-framework[3].

[3] courses/csse2002/java-collections-framework.pdf