Data Types and its Ranges in Java

There are two types of data types they are

- Primitive data type
- Non primitive data type

Primitive data type

Data Type	Size (bits)	Range (Min → Max)	Example
byte	8-bit	-128 to 127	byte b = 100;
short	16-bit	-32,768 to 32,767	short s = 20000;
int	32-bit	-2,147,483,648 to 2,147,483,647	int i = 100000;
long	64-bit	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	long l = 10000000000L;
float	32-bit (single precision)	~ ±3.40282347E+38 (6–7 decimal digits precision)	float f = 3.14f;
double	64-bit (double precision)	~ ±1.7976931348623157E+308 (15–16 decimal digits precision)	double d = 3.141592653589793;
char	16-bit (Unicode)	0 to 65,535 (represents characters)	char c = 'A';

boolean	JVM-dependent (usually 1 bit used in arrays, 8 bits in memory)	true or false	boolean flag = true;

Non-Primitive data type:

◇ Java Non-Primitive Data Types

These are **objects (reference types)**, not stored directly as values like primitives. They store a **reference (address)** pointing to memory.

1. **String**

- A sequence of characters.
- Immutable (once created, cannot change).

String name = "Bhuvi";
System.out.println(name.toUpperCase()); // BHUVI

2. **Arrays**

- Fixed-size collection of same type.

```
int[] nums = {10, 20, 30};
System.out.println(nums[1]); // 20
```

3. **Classes**

- Blueprint for creating objects.
- Can have fields (variables) and methods.

```
class Student {
   String name;
   int age;
}
Student s = new Student();
s.name = "Bhuvi";
s.age = 21;
System.out.println(s.name + " - " + s.age);
```

4. **Objects**

- Instance of a class.

```
class Car {
   void drive() { System.out.println("Car is driving"); }
}
Car c = new Car();
c.drive();  // Car is driving
```

5. **Interfaces**

- Similar to a contract → defines abstract methods.
- Classes implement them.

```
interface Animal {
   void sound();
}
class Dog implements Animal {
   public void sound() { System.out.println("Bark"); }
}
new Dog().sound();
```

6. **Enums**

- Special type for fixed constants.

```
enum Day { MON, TUE, WED }
Day d = Day.MON;
System.out.println(d);
```

7. **Wrapper Classes**

- Object versions of primitive types (useful for Collections, Generics).

- Example: Integer, Double, Character, Boolean.

```
Integer x = 10; // wraps int
System.out.println(Integer.MAX_VALUE); // 2147483647
```

Key Difference: Primitive vs Non-Primitive

Primitive	Non-Primitive
Predefined by Java	Can be user-defined (classes, objects)
Store values directly	Store reference (address)
Always lowercase (e.g., int)	Always start with uppercase (e.g., String, Integer)
Faster, memory-efficient	More powerful & flexible