Data Types in Java

Definition

> A **data type** defines the kind of data a variable can store in its allocated memory.

> *(Previous version: "Every data has some type, and that type is known as a Data Type." — outdated)*

Types of Data Types in Java

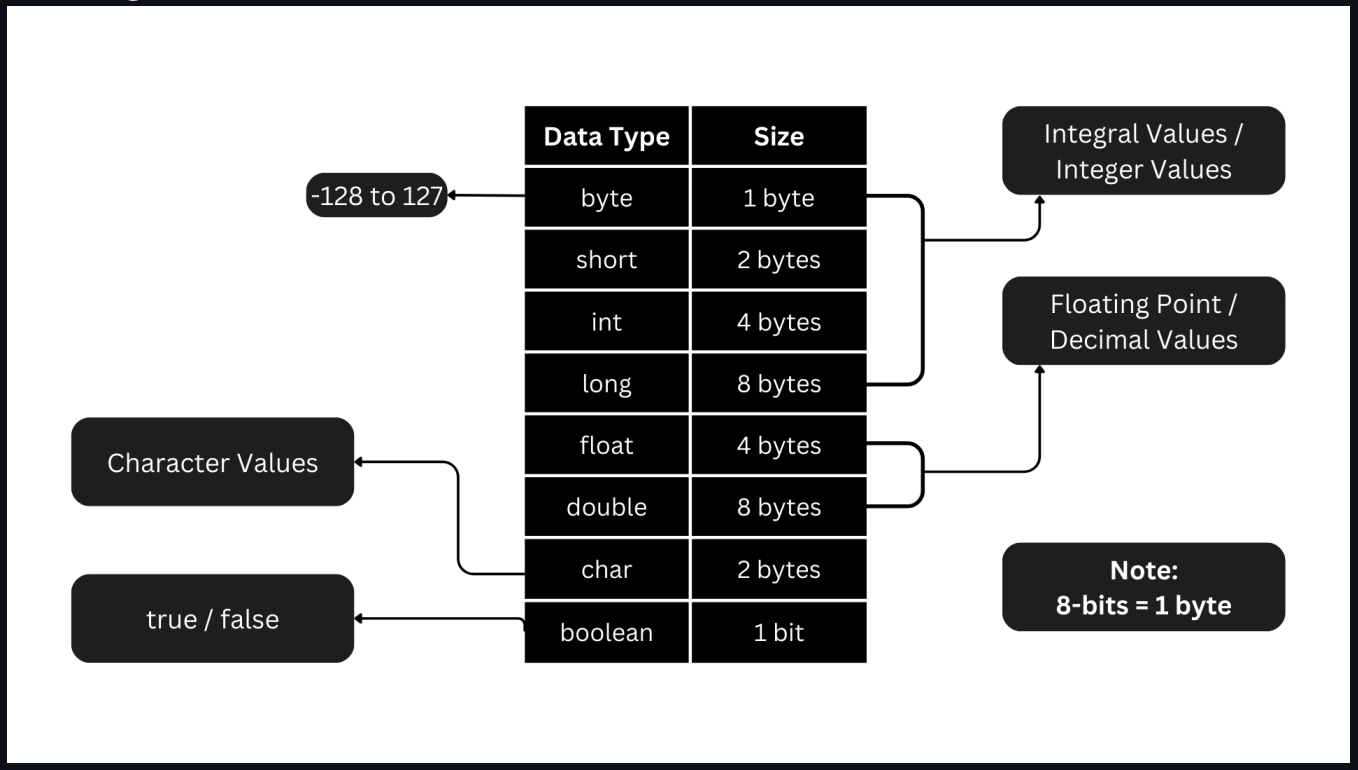
Java supports two broad categories of data types:

- **Primitive Data Types**
- **Non-Primitive Data Types**

Primitive Data Types

- These types store **only values**, not references.
- Java defines **8 primitive types**:
    - `byte`, `short`, `int`, `long`
    - `float`, `double`
    - `char`
    - `boolean`

Diagram:



Example:

```java
class PrimitiveDataTypes {
    public static void main(String[] args) {
        byte b = 123;
        short s = 12345;
        int i = 12;
        long l = 12L;          // L denotes long
        float f = 1.1F;        // F denotes float
        double d = 2.2D;       // D denotes double
        char c = 'a';
        boolean bo = true;

        System.out.println(b + "\n" + s + "\n" + i + "\n" + l);
        System.out.println(f + "\n" + d);
        System.out.println(c);
        System.out.println(bo);
    }
}
```

Non-Primitive Data Types

- Also known as **reference types** because they store **addresses** to objects in memory.

Object

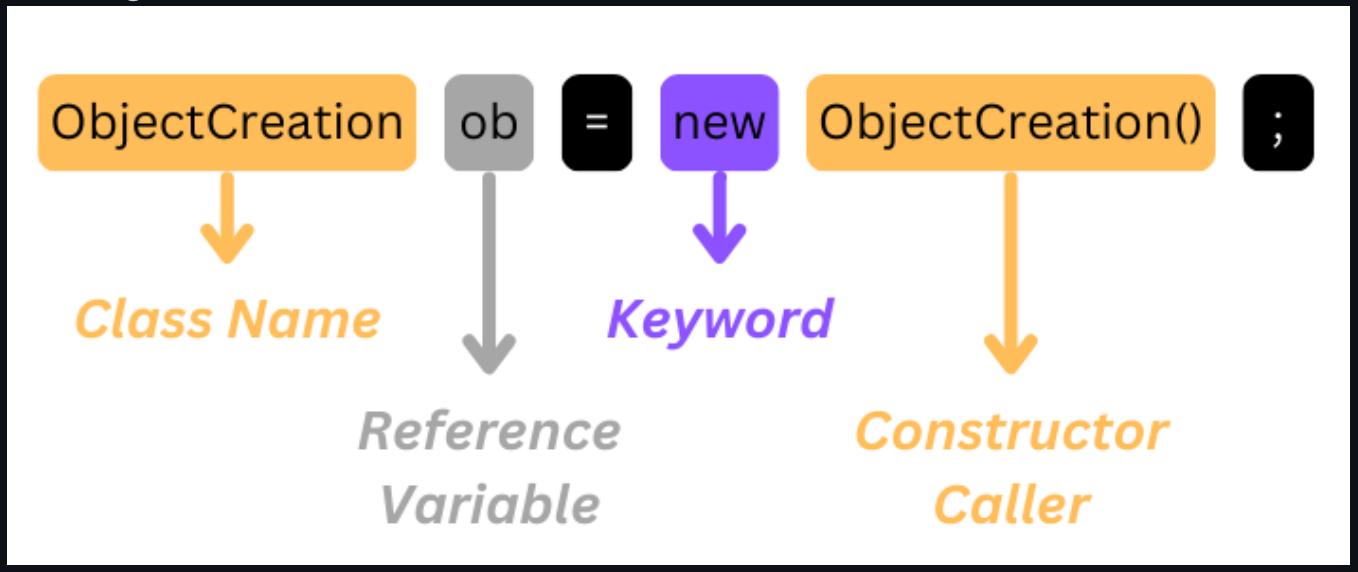
An **object** is a real-world entity characterized by:

- **State** – represented by variables
- **Behavior** – defined by methods

Object Creation Example:

```java
class ObjectCreation {
    public static void main(String[] args) {
        ObjectCreation ob = new ObjectCreation();
    }
}
```

Diagram:



Multiple Objects Example:

```
class MultipleObjects {
    public static void main(String[] args) {
        MultipleObjects ob1 = new MultipleObjects();
        MultipleObjects ob2 = new MultipleObjects();
        MultipleObjects ob3 = new MultipleObjects();
        MultipleObjects ob4 = new MultipleObjects();
    }
}
```

Reference Variable Address Example:

```
class ObjectAddress {
    public static void main(String[] args) {
        ObjectAddress ob1 = new ObjectAddress();
        ObjectAddress ob2 = new ObjectAddress();
        System.out.println(ob1);
        System.out.println(ob2);
    }
}
```

> *Note: The reference variable stores the address of the object in the **heap memory**.*

Miscellaneous Example

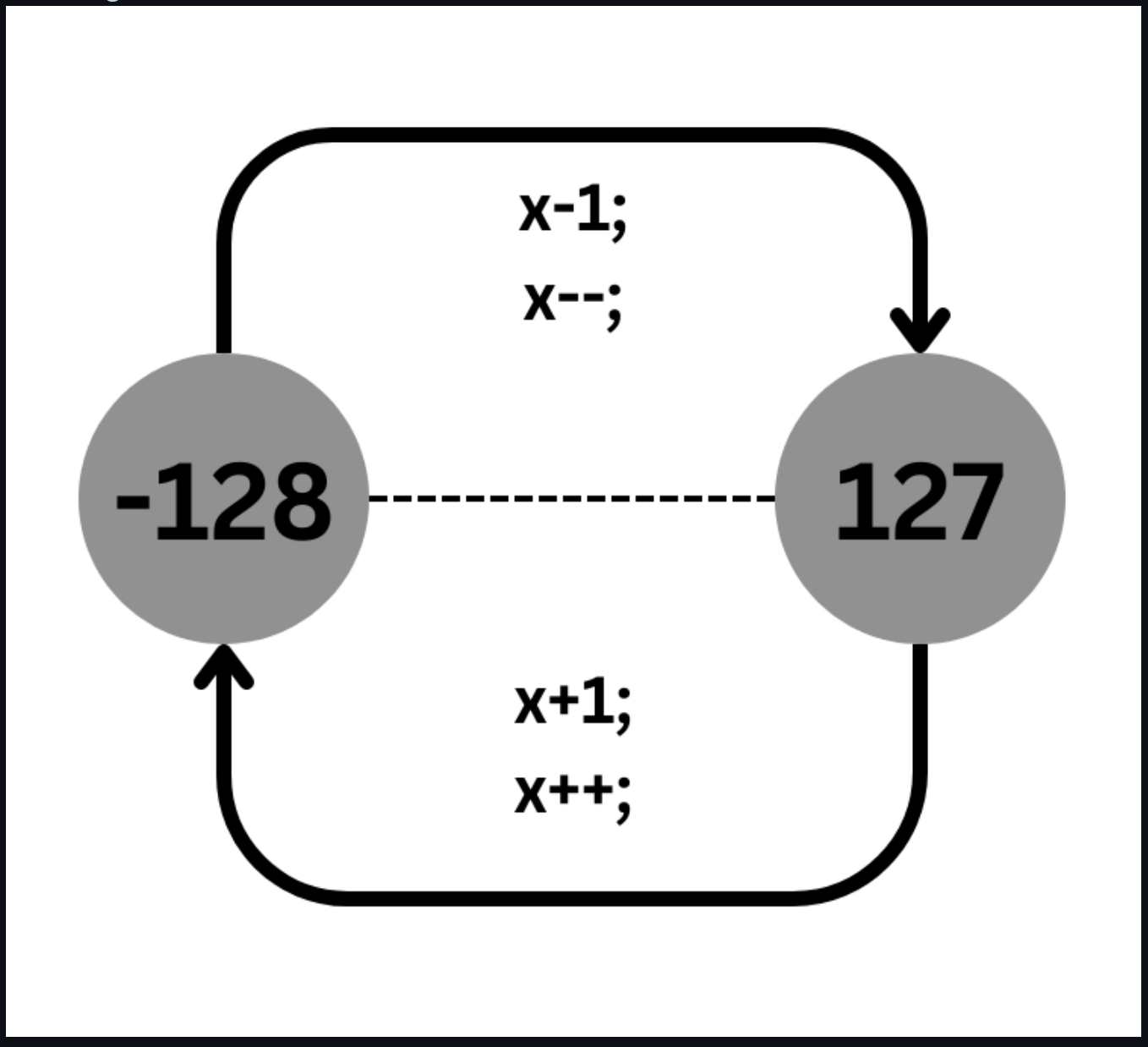
```
class ByteOverflow {
    public static void main(String[] args) {
        byte b = 127;
        b++;
        System.out.println(b);
    }
}
```

Output:

```
1  -128
```

> *The value overflows and wraps around the range for the `byte` data type (−128 to 127).*

Diagram:



Method Resolution

Diagram: