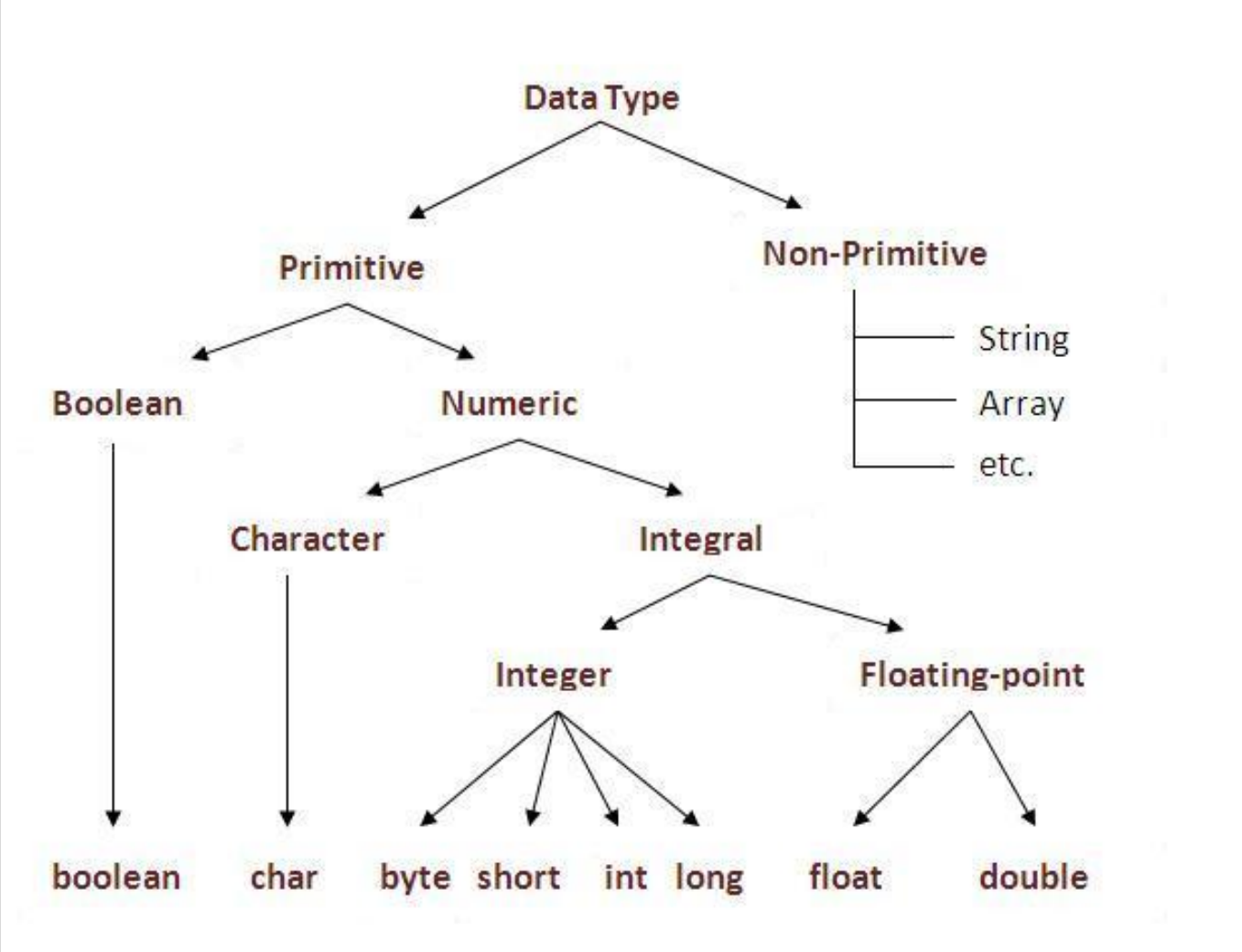

LETS LEARN JAVA
BEYOND THE BASIC TYPES

Training Plan - Basics

Topics to be covered
Reference Data Types Value vs. Reference, Heap vs. Stack Wrapper Classes. Why they are used, Auto-Boxing and Unboxing Example of Operations on an Integer Wrapper class String – String data type, how its an array of characters, methods on Strings Immutability of String Alternatives to String – StringBuilder (and StringBuffer) Dates - Java 8 time APIs. LocalDate, LocalDateTime. How to get current date, time. How to operate on dates Duration, Period and ChronoUnits

Data Types recap



We have already discussed about **primitive** data types. But what about non-primitive ones

Non primitive data types are also called Reference types.

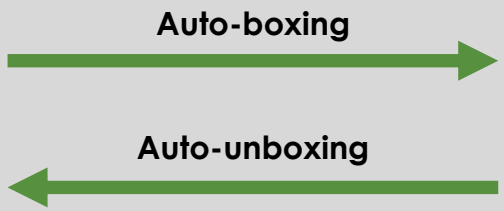
There are three broad categories of reference data types –

- Wrapper classes/types
  - e.g. `Integer`
- Java provided Reference Types
  - e.g. `String`, `LocalDate`
- User Defined Reference Types (classes)
  - e.g. `Student`, `Animal`

Wrapper Classes

- In Java, every primitive data type has a corresponding Wrapper class.
- Wrapper data types are better as they have lots of utility methods.
- Wrapper classes also allow to be used in collections.
- Wrapper objects are immutable
- Except Character and Integer, all other cases, the name of the data type is same, but follows Pascal casing (as they are classes)
- Conversion from primitive to wrapper is called Boxing and reverse is called Unboxing

Primitive Data Type	Wrapper Classes
boolean	Boolean
char	**Character**
byte	Byte
short	Short
int	**Integer**
long	Long
float	Float
double	Double



Wrapper Classes

```
public static void main(String[] args) {
    Integer number = 10;
    Integer anotherNumber = Integer.valueOf(20);

    System.out.println("Max range of Integer is : " + Integer.MAX_VALUE);
    System.out.println("Min range of Integer is : " + Integer.MIN_VALUE);

    String scannedValue = "12034";
    Integer scannedNumberValue = Integer.valueOf(scannedValue);

    System.out.println("Next value of the scanned number is : " + ++scannedNumberValue);
}
```

How data is stored in memory

STACK - Method Level		
Variables		
Location	Variable Name	Value
1001		
1002	score	10
1003		
1004	grade	A
1005		
1006	name	107
1007		
1008		

HEAP - Global	
Objects	
Location	Reference Object
101	
102	
103	
104	
105	
106	
107	StringObject11
108	


```
int score = 10;
char grade = 'A';
String name = "Pritam";
```

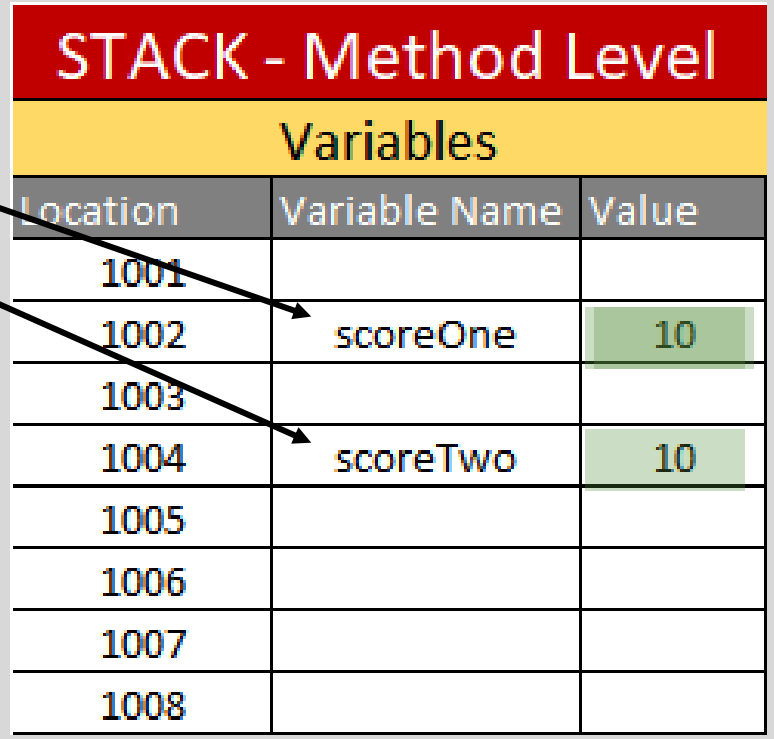
Primitive data types directly store values in Stack.

Reference data types store Object address/reference in Stack, and actual Object is stored in the HEAP

Comparison of primitive data

```java
int scoreOne = 10;
int scoreTwo = 10;

if (scoreOne == scoreTwo) {
    System.out.println("Equal");
}else {
    System.out.println("Not Equal");
}
```



So far we have used == to check if two variables are storing same value

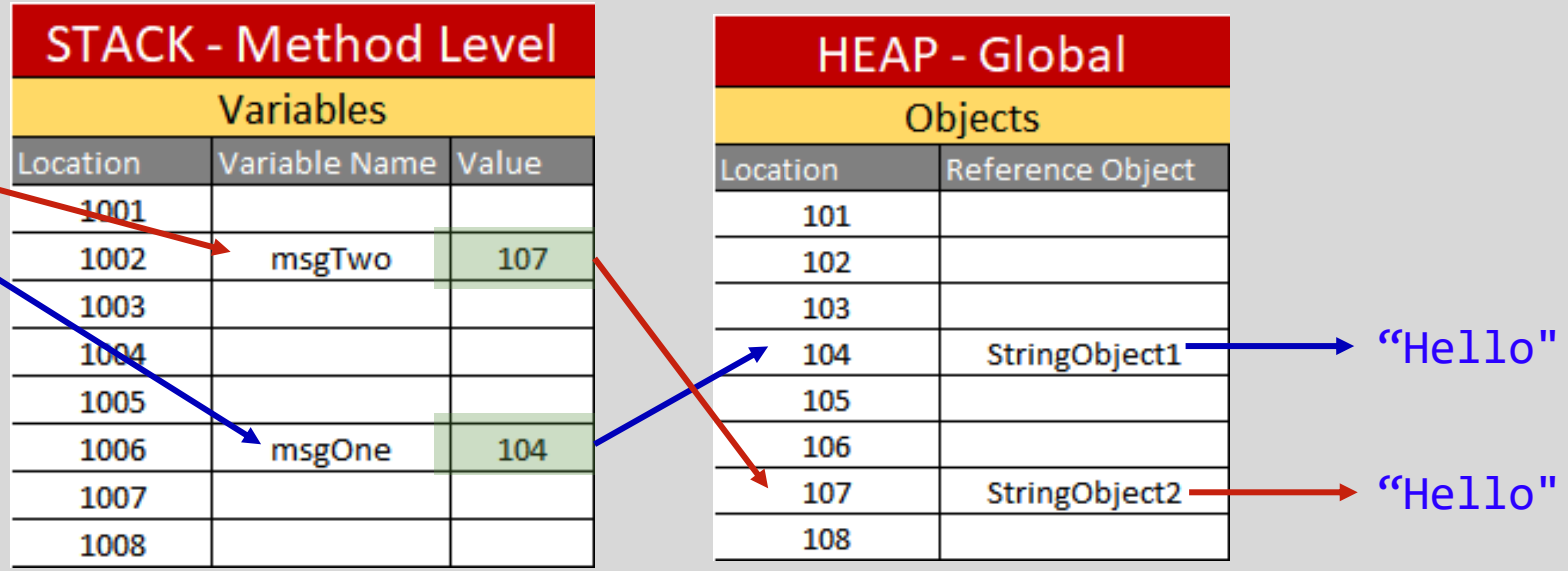
== checks the value in Stack.

For primitive data types since values are stored in Stack, == returns true

Comparison of reference data

```
String msgOne = "Hello";
String msgTwo = "Hello";

if (msgOne == msgTwo) {
    System.out.println("Equal");
}else {
    System.out.println("Not Equal");
}
```



If we use == for Strings, it compares the values on Stack, which will be different and hence returns false.

To check equality of values of actual Objects, we use .equals() method. Correct implementation is

```
if (msgOne.equals(msgTwo))
```

Special reference Types - String

Strings are internally array of characters

```
String message = "Hello";
char[] msgArray = {'H', 'e', 'l', 'l', ‘o'};
```

Operations on Strings

charAt – find the character at a specific index location
toCharArray – converts the string into a character array
indexOf – find the position of the first occurrence of a character/string
contains – checks if a string is contained within
substring – standard substring functionality

Special reference Types - String

Strings are internally array of characters

```
String message = “  We are learning Java  ";
```

Operations on Strings

trim – remove spaces at beginning and end of string
concat – concatenate two strings
toUpperCase – converts string to UPPER case
toLowerCase – converts string to lower case

Strings are immutable

```
String message = "Learning";
message.concat(" Java");

System.out.println(message);
```


"Learning"

What happened here…
Why the concat did not change the value of `message`

String Alternative – StringBuilder

```
StringBuilder message = new StringBuilder("Learning");
message.append(" Java");

System.out.println(message);
```

"Learning Java"

Operations on `StringBuider` (additional from String)

reverse – reverses the string
setCharAt – updates the character at a position to the one passed
insert – insert characters/string at a position
delete – delete characters/string at a position, or between one position to another

There is another alternative called `StringBuffer` which we will discuss post multithreading

Special reference Types - LocalDate

```
public static void main(String[] args) {
    LocalDate today = LocalDate.now();
    System.out.println(today);

    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MMM-yyyy");
    System.out.println(today.format(formatter));
}
```

2021-03-21

21-Mar-2021

LocalDate – for dates
LocalDateTime – for Date and Time

Date with Time

```
public static void main(String[] args) {
    LocalDate today = LocalDate.now();
    System.out.println(today);

    DateTimeFormatter formatter = DateTimeFormatter.ofPattern("dd-MMM-yyyy");
    System.out.println(today.format(formatter));

    LocalDateTime now = LocalDateTime.now();
    System.out.println(now);

    DateTimeFormatter timeFormatter = DateTimeFormatter.ofPattern("dd-MMM-yyyy hh:mm:ss a");
    System.out.println(now.format(timeFormatter));
}
```

→ 2021-03-27T18:23:24.923820500

→ 27-Mar-2021 06:23:24 PM

Operations on Dates

```
public static void main(String[] args) {

    // Set to specific date
    LocalDate someDay = LocalDate.of(1999, 12, 31);
    System.out.println(someDay);

    LocalDate someOtherDay = LocalDate.parse("01-Dec-2020", DateTimeFormatter.ofPattern("dd-MMM-yyyy"));
    System.out.println(someOtherDay);

    LocalDate today = LocalDate.now();

    System.out.println("Today : " + today);
    System.out.println("20 days back : " + today.minusDays(20));
    System.out.println("2 years 3 months from now : " + today.plusYears(2).plusMonths(3));
}
```

Difference between dates

```
public static void main(String[] args) {

    LocalDate startDate = LocalDate.of(1999, 12, 31);
    LocalDate endDate = LocalDate.of(2021, 03, 27);

    Period period = Period.between(startDate, endDate);

    System.out.println(
            period.getYears() + " Years, " + period.getMonths() + " Months, and " + period.getDays() + " Days");

    System.out.println(ChronoUnit.DAYS.between(startDate, endDate));

}
```

Similar to Period, for calculating difference between two `LocalDateTime`, we use `Duration`.

ZonedDateTime for working with Time zones

```
public static void main(String[] args) {

    LocalDateTime now = LocalDateTime.now();

    ZonedDateTime nowinIndia = now.atZone(ZoneId.of("Asia/Kolkata"));
    System.out.println(nowinIndia);

    ZonedDateTime nowInLondon = nowinIndia.withZoneSameInstant(ZoneId.of("Europe/London"));
    System.out.println(nowInLondon);

}
```