Java - Basic Concepts

Table of Content

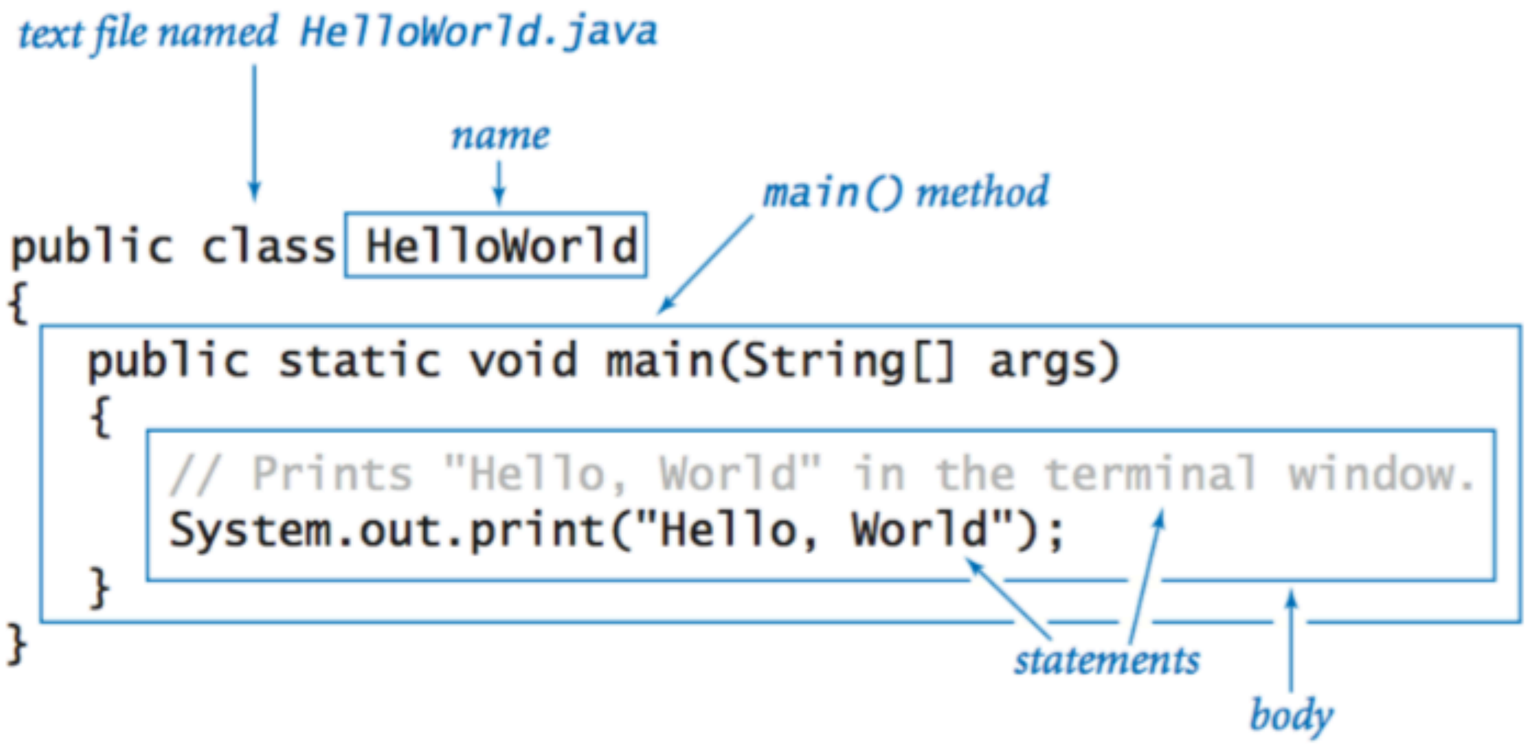
What is Java?

- High-level, class-based, object-oriented programming language
- One of the most popular programming languages
- Java applications are compiled to bytecode and can be run on every platform that has Java on it
- Android applications, Desktop GUI applications, Web-Based applications,…

Basic syntax

```java
public class Main {
  public static void main(String[] args) {
    System.out.println("Hello World");
  }
}
```

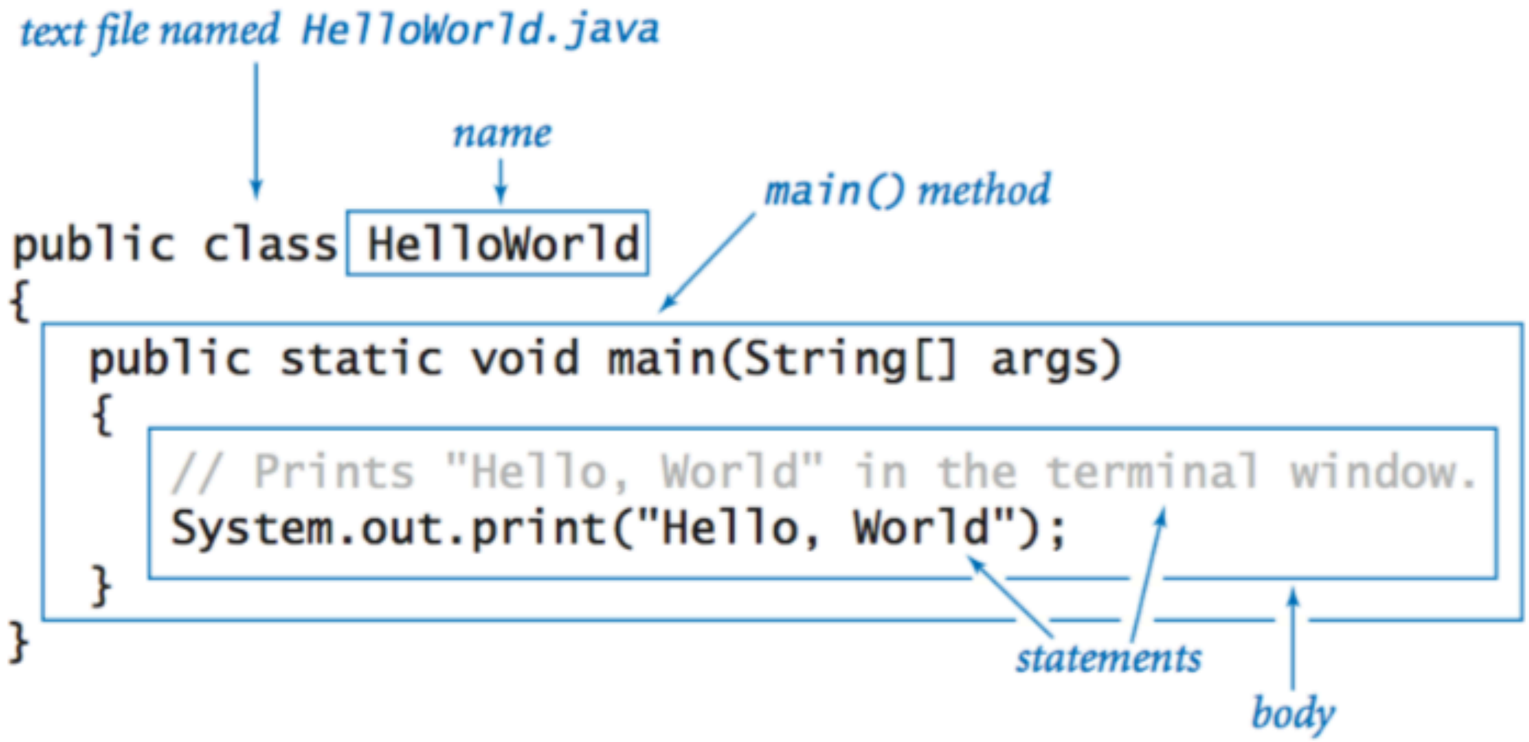
Basic syntax



The main method

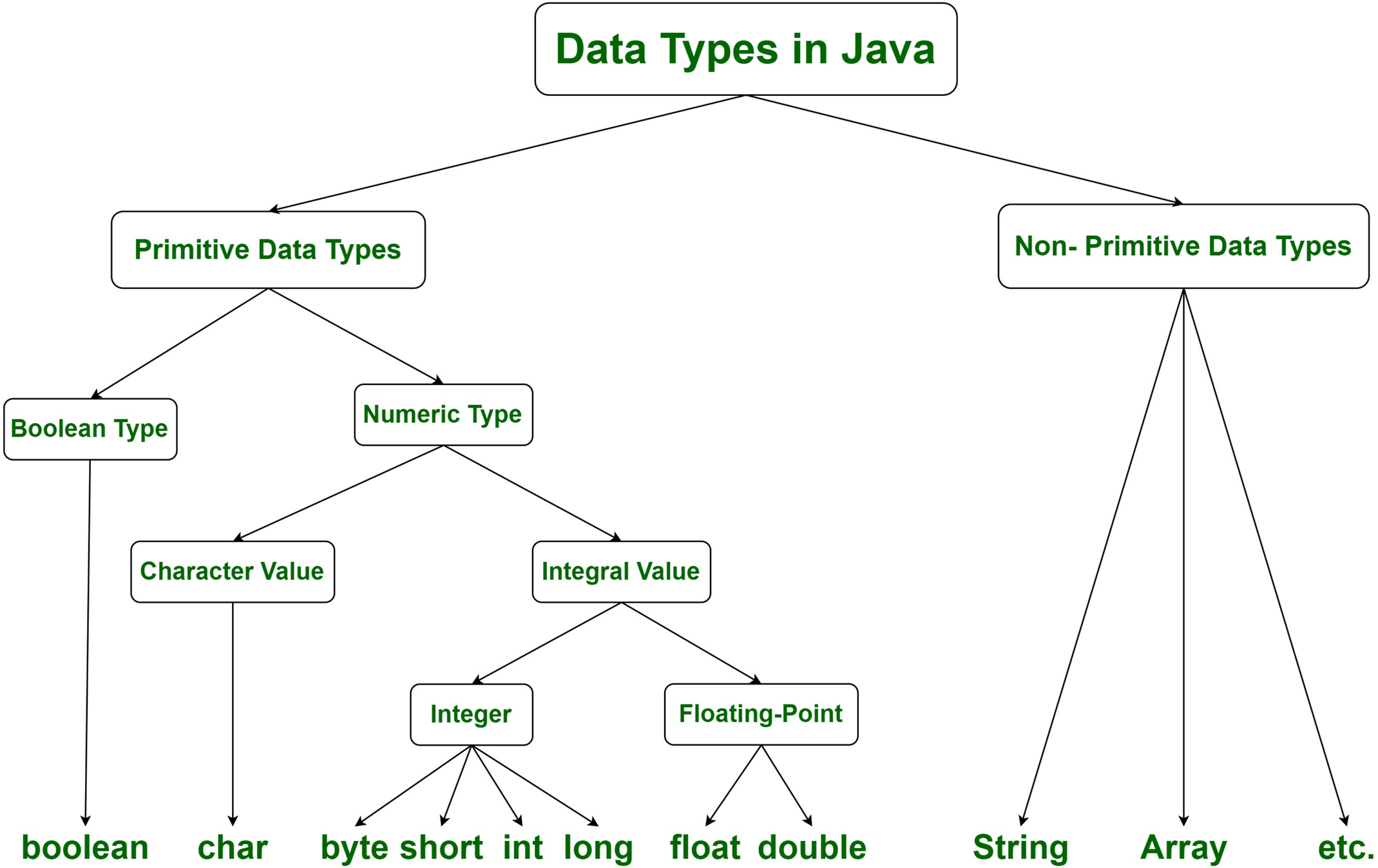
- Every line of code that runs in Java must be inside a class
- Class always starts with an uppercase first letter
- **Note**: Java is case sensitive, therefore “MyClass” and “myclass” have different meaning
- The name of the java file **must match** the class name
- When saving the file, save it using the class name and add “.java” to the end of the file name
- **main** method is the entry point for code execution

Basic syntax



Data Types

- Data types specify the different sizes and values that can be stored in the variable
- Primitive types (*value* types)
- Non-Primitive types (*reference* types)


Data Types in Java
Primitive Data Types
Non- Primitive Data Types
Boolean Type
Numeric Type
Character Value
Integral Value
Integer
Floating-Point
boolean
char
byte
short
int
long
float
double
String
Array
etc.

Primitive Types

8 primitive types in Java

Type	Size (bits)	Minimum	Maximum	Example
byte	8	-2^{7}	$2^{7}-1$	*byte b = 100;*
short	16	-2^{15}	$2^{15}-1$	*short s = 30_000;*
int	32	-2^{31}	$2^{31}-1$	*int i = 100_000_000;*
long	64	-2^{63}	$2^{63}-1$	*long l = 100_000_000_000_000;*
float	32	-2^{-149}	$(2-2^{-23})\cdot 2^{127}$	*float f = 1.456f;*
double	64	-2^{-1074}	$(2-2^{-52})\cdot 2^{1023}$	*double f = 1.456789012345678;*
char	16	0	$2^{16}-1$	*char c = 'c';*
boolean	1	–	–	*boolean b = true;*

Boolean

- Boolean data type is used to store only two possible values: ***true*** or ***false***
- `bool isValid = true;`
- `bool isValid = false;`

int

- 32-bit integer value (from -2,147,483,648 to +2,147,483,647)
- `int a = 589000;`
- `int b = -243;`

long

- 64-bit integer value (from -2^63 to +2^63 - 1)
- `long x = 10000L;`
- `long y = -23500L;`

float

- From **float**ing point values
- `float x = 234.5f;`

double

- From **double**-precision floating point
- `double num = 123.34;`

char

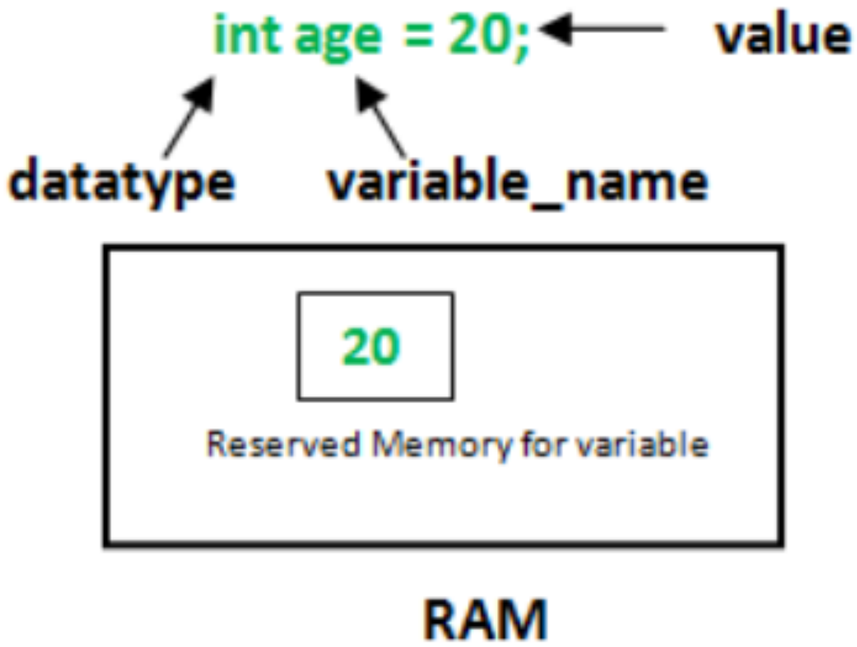
- Single 16-bit Unicode character
- char values are enclosed with **single** quotes (important!)
- `char letterA = 'A'`
- `char letterB = 'B'`

Type Casting

- Type casting occurs when a value of one primitive data type is assigned to another type
- Two types of casting:
  - *automatically* - converting a smaller type to a larger type
    - `byte -> short -> char -> int -> long -> float -> double`
  - *manually* - converting larger type to a smaller type
    - `double -> float -> long -> int -> char -> short -> byte`

Variables

- Variable is a container which holds the value while Java program is executed
- It’s the name of a reserved area allocated in the memory



Variable classification

- 3 types of variables in Java
- Local Variables
- Instance Variables
- Static Variables

Local Variables

- A variable defined within a *block* or *method* or class *constructor* is called a local variable
- Local variables are created when the block is entered, or the method is called and destroyed after exiting from the block or when the call returns from the function
- Local variables can be accessed only within the block they were declared
- ```
  class Example {
        public static void main(String[] args) {
              int age = 10;
              System.out.println(“I am “ + age “ years old”);
        }
  }
  ```
  ```

Instance Variables

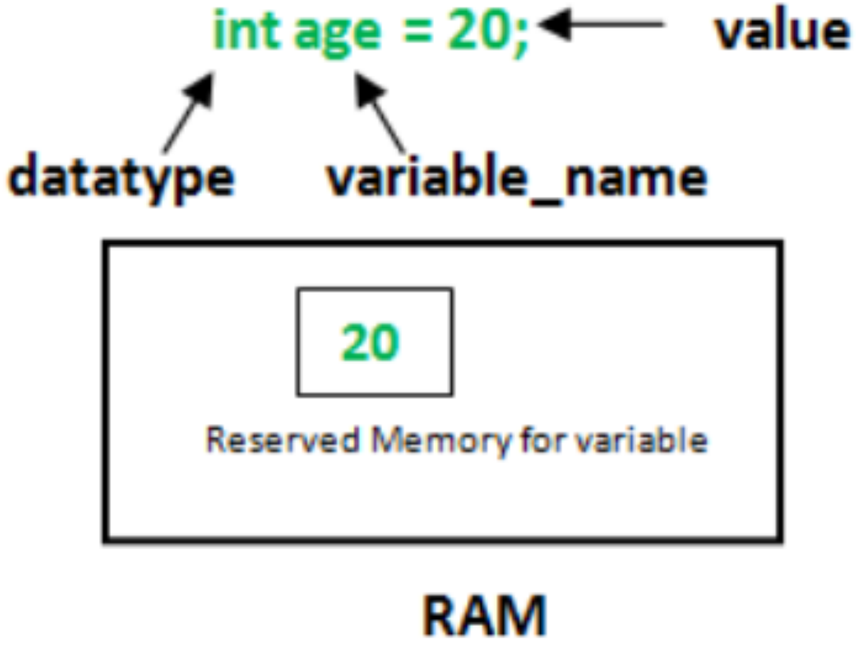
- Instance variables are non-static variables, declared in a class outside any method, constructor or block
- Instance variables live as long as the object they are created lives.
- More on this later

Static Variables

- Also known as Class variables
- Static variables are declared in the same way as Instance variables, but using the **static** keyword.
- Created at the start of program execution and destroyed automatically when execution ends.
- More on this later

Variable declaration

- To declare a variable, we need:
  - Data type
  - Variable name
- Rules for naming variables:
  - Starts with a letter, underscore or a dollar sign ($)
  - Followed by letters, digits, $, underscore
  - Cannot contain any keyword



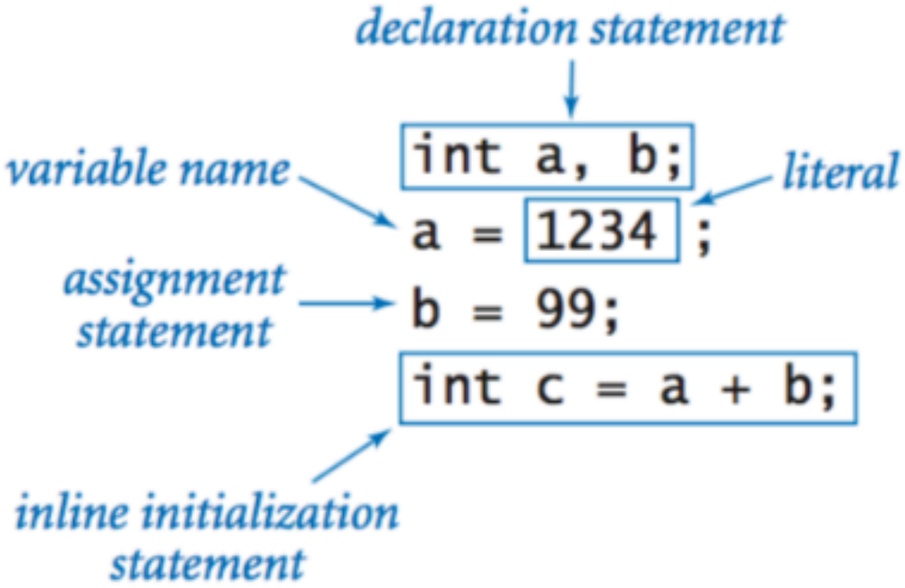
Variable declaration

Naming convention - best practice

- Variable name should start with a lower-case letter
- Java variables should be written in `camelCase` notation
- Variable names should be self-explanatory

Purpose of Variable	Good Names, Good Descriptors	Bad Names, Poor Descriptors
Running total of checks written to date	runningTotal, checkTotal	written, ct, checks, CHKTTL, x, x1, x2
Velocity of a bullet train	velocity, trainVelocity, velocityInMph	velt, v, tv, x, x1, x2, train
Current date	currentDate, todaysDate	cd, current, c, x, x1, x2, date
Lines per page	linesPerPage	lpp, lines, l, x, x1, x2

Declaration statement vs assignment statement



Operators

- Arithmetic operators
- Logical operators
- Comparison operators

Arithmetic Operators

Used for mathematical or computational logic

- + (addition; also used for string concatenation)
- - (subtraction)
- * (multiplication)
- / (division)
- % (modulus or remainder)

Logical Operators

Used for evaluating boolean expressions

- && (AND)
- || (OR)
- ! (NOT)

A	B	A AND B	A OR B	NOT A
False	False	False	False	True
False	True	False	True	True
True	False	False	True	False
True	True	True	True	False

```
if (number % 2 == 0 || number % 5 == 0) {
System.out.println(number + " is divisible by 2 OR 5");
}
```

Comparison Operators

Used for comparing values

- < (less than)
- <= (less than or equal to)
- > (greater than)
- >= (greater than or equal to)
- == (equal to)
- != (not equal to)

```
public boolean canVote(int age) {
    if(age < 18) {
        return false;
    }
    return true;
}
```

Literals
(*constants)

```
// Here 100 is a constant/literal.
int x = 100;
```

- Literal is any constant value which can be assigned to the variable
- It’s a representation of boolean, numeric (integral and floating-point), character or string data
- 4 types of literals:
  - Numeric literals
  - Character literals
  - String literals
  - Boolean literals

Numeric Literals

Integral Literals

- For integral data types (byte, short, int, long), we can specify literals in 4 ways:
  - Decimal literals (base 10) - In this form, the allowed digits are 0-9

```
int x = 101;
```

Numeric Literals

Integral Literals

- Octal literals (base 8) - In this form, the allowed digits are 0-7

```
// The octal number should be prefix with 0.
int x = 0146;
```

Numeric Literals

Integral Literals

- Hexa-Decimal literals (base 16) - In this form, allowed digits are 0-9 and characters a-f

```
// The hexa-decimal number should be prefix
// with 0X or 0x.
int x = 0X123Face;
```

Numeric Literals

Integral Literals

- Binary literals - In this form, the allowed digits are 0 and 1

```
int x = 0b1111;
```

Numeric Literals

Floating-Point Literals

- Floating-Point literals we can specify only in decimal form

```
double d = 123.456;
```

Character Literals

Single quote character literals

- We can specify character literals in 4 ways:
  - Single quote - only a single character

```
char ch = 'a';
```

Character Literals

Char literal as Integral literal

- Decimal, octal and hexa-decimal forms
- Allowed range is 0 to 65535

```
char ch = 062;
```

Character Literals

Unicode Representation

- We can specify char literals in Unicode representation `\uxxxx`
  - xxxx represents 4 hexa-decimal numbers

```
char ch = '\u0061';// Here /u0061 represent a.
```

Character Literals

Escape Sequence

- Every escape character can be specified as char literal (‘\n’, ‘\t’, …)

```
char ch = '\n';
```

String Literals

- String literal represents any sequence of characters within **double quotes**

```
String s = "Hello";
```

Boolean Literals

- Boolean literal allows only two values: **true** and **false**

```
boolean b = true;
boolean c = false;
```

String

```
// string variable
String courseName = "Java Course"
```

- In Java, String is a sequence of characters
- Strings are represented with sequence of characters within **double quotes**
- String is a non-primitive type in Java
- All string variables in Java are instances (*objects*) of `String` class
- Java Strings are **immutable** - this means that once we create a string, we cannot change that string

Operations with strings

- As we mentioned, strings in Java are actually instances of `String` class
- String class in Java has a bunch of built-in methods that are useful, and we can use them out-of-the-box

Get length of a String

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

        // create a string
        String greet = "Hello! World";
        System.out.println("String: " + greet);

        // get the length of greet
        int length = greet.length();
        System.out.println("Length: " + length);

    }
}
```

Join two Strings

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

        // create first string
        String first = "Java ";
        System.out.println("First String: " + first);

        // create second
        String second = "Programming";
        System.out.println("Second String: " + second);

        // join two strings
        String joinedString = first.concat(second);
        System.out.println("Joined String: " + joinedString);

    }
}
```

```
First String: Java
Second String: Programming
Joined String: Java Programming
```

Comparing Strings

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

        // create 3 strings
        String first = "java programming";
        String second = "java programming";
        String third = "python programming";

        // compare first and second strings
        boolean result1 = first.equals(second);
        System.out.println("Strings first and second are equal: " + result1);

        // compare first and third strings
        boolean result2 = first.equals(third);
        System.out.println("Strings first and third are equal: " + result2);
    }
}
```

```
Strings first and second are equal: true
Strings first and third are equal: false
```

Escaping characters in Strings

- Let’s say we want to create string variable that has double quotes inside string value

```
String quote = "He said: "Hi, how are you?" to me.";
```

- Problem? Let’s escape inner double quotes:

```
String quote = "He said: \"Hi, how are you?\" to me.";
```

String format() method

- Similar to backticks in JS
- Method receives a string where all of the format specifiers will be replaced by arguments that are passed in to the method

```
String str1 = String.format("%d", 101);           // Integer value
String str2 = String.format("%s", "Amar Singh"); // String value
String str3 = String.format("%f", 101.00);        // Float value
```

Format Specifier	Data Type	Output
%a	floating point (except *BigDecimal*)	Returns Hex output of floating point number.
%b	Any type	"true" if non-null, "false" if null
%c	character	Unicode character
%d	integer (incl. byte, short, int, long, bigint)	Decimal Integer
%e	floating point	decimal number in scientific notation
%f	floating point	decimal number
%g	floating point	decimal number, possibly in scientific notation depending on the precision and value.
%h	any type	Hex String of value from hashCode() method.
%n	none	Platform-specific line separator.
%o	integer (incl. byte, short, int, long, bigint)	Octal number
%s	any type	String value
%t	Date/Time (incl. long, Calendar, Date and TemporalAccessor)	%t is the prefix for Date/Time conversions. More formatting flags are needed after this. See Date/Time conversion below.
%x	integer (incl. byte, short, int, long, bigint)	Hex string.

```java
String myName = "Pera";

String formattedString = String.format("My name is %s", myName);
System.out.println(formattedString);

int myAge = 25;
String formattedString2 = String.format("My name is %s and I am %d years old", myName, myAge);
System.out.println(formattedString2);
```

Exercises

- https://www.hackerrank.com/challenges/java-strings-introduction/problem
- https://www.hackerrank.com/challenges/java-substring/problem
- https://www.hackerrank.com/challenges/java-string-reverse/problem