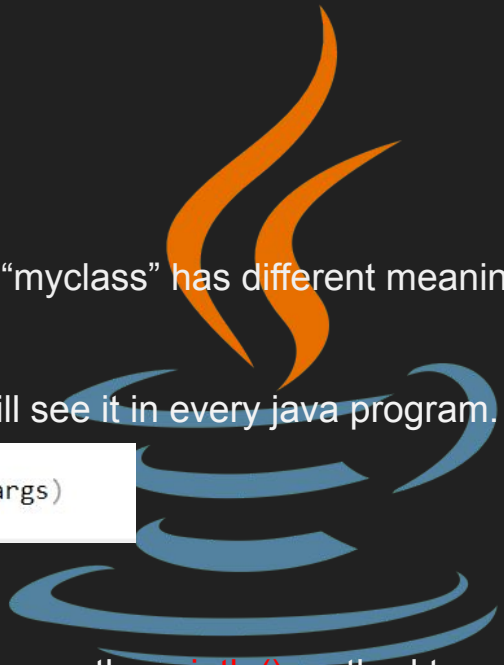
Java Basic Syntax

Java Syntax



Java is case-sensitive: “MyClass” and “myclass” has different meaning.

The **Main** method

the main() is required and you will see it in every java program.

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

System.out.println()

Inside the main() method, we can use the println() method to print a line of text in the screen.

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

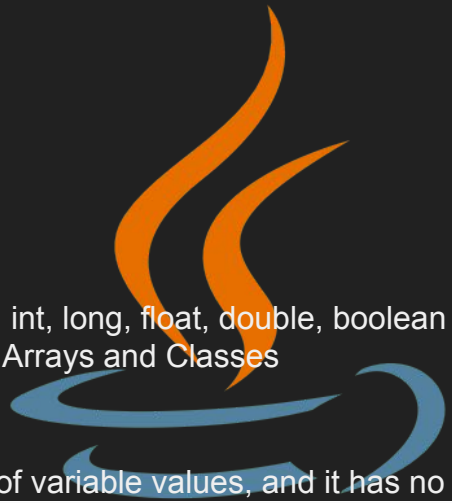
Java Variables

- String - stores text, such as “Hello”. String values are surrounded by double quotes
- int - stores integers (whole numbers), without decimals. Such as 123 or -123
- float - stores floating point numbers, with decimals. Such as 1.23 or -1.23
- char - stores single characters, such as ‘a’ or ‘B’. Char values are surrounded by sing quotes.
- boolean - stores value with two states: true or false

Example:

```
int myNum = 5;
float myFloatNum = 5.99f;
char myLetter = 'D';
boolean myBool = true;
String myText = "Hello";
```

Java Data Types



Data types are divided into two groups:

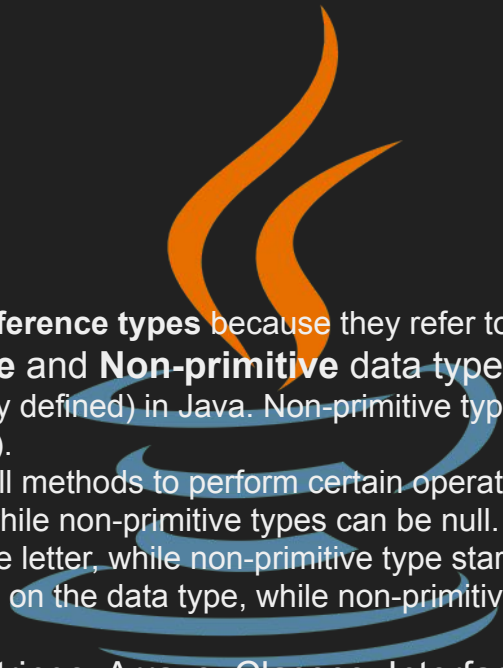
- Primitive data types - includes byte, short, int, long, float, double, boolean and char.
- Non-primitive data types - such as String, Arrays and Classes

Primitive Data types

A primitive data type specifies the size and type of variable values, and it has no additional methods.

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

Java Data Types



Non-primitive data types

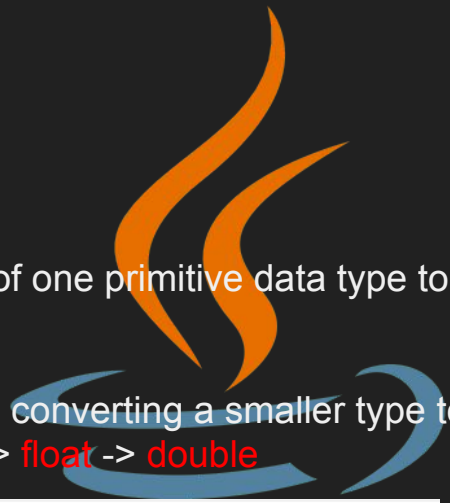
Non-primitive data types are called **reference types** because they refer to objects.

The main difference between **Primitive** and **Non-primitive** data types are:

- Primitive types are predefined (already defined) in Java. Non-primitive types are created by the programmer and is not defined by Java (except for String).
- Non-primitive types can be used to call methods to perform certain operations, while primitive types cannot.
- A primitive type has always a value, while non-primitive types can be null.
- A primitive type starts with a lowercase letter, while non-primitive type starts with an uppercase letter.
- The size of a primitive types depends on the data type, while non-primitive types have all the same size.

Examples of non-primitive types are Strings, Arrays, Classes, Interface, etc.

Java Type Casting



Type casting is when you assign a value of one primitive data type to another type.
2 types of type casting in Java:

- **Widening Casting** (automatically) - converting a smaller type to a larger type size.
  byte -> short -> char -> int -> long -> float -> double

```
public class MyClass {
  public static void main(String[] args) {
    int myInt = 9;
    double myDouble = myInt; // Automatic casting: int to double

    System.out.println(myInt);      // Outputs 9
    System.out.println(myDouble);   // Outputs 9.0
  }
}
```

Java Type Casting

- **Narrowing Casting** (manually) - converting a larger type to a smaller type size.
  double-> float-> long-> int -> char-> short-> byte

```java
public class MyClass {
  public static void main(String[] args) {
    double myDouble = 9.78;
    int myInt = (int) myDouble; // Manual casting: double to int

    System.out.println(myDouble);   // Outputs 9.78
    System.out.println(myInt);      // Outputs 9
  }
}
```

Java Operators

Operators are used to perform operations on variables and values.

- Arithmetic Operators
- Assignment Operators
- Comparison Operators
- Logical Operators

Java Operators

Arithmetic Operators

Operator	Name	Description	Example
+	Addition	Adds together two values	x + y
-	Subtraction	Subtracts one value from another	x - y
*	Multiplication	Multiplies two values	x * y
/	Division	Divides one value from another	x / y
%	Modulus	Returns the division remainder	x % y
++	Increment	Increases the value of a variable by 1	++x
--	Decrement	Decreases the value of a variable by 1	--x

Java Operators

Assignment Operators

Operator	Example	Same As
=	x = 5	x = 5
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Java Operators

Comparison Operators

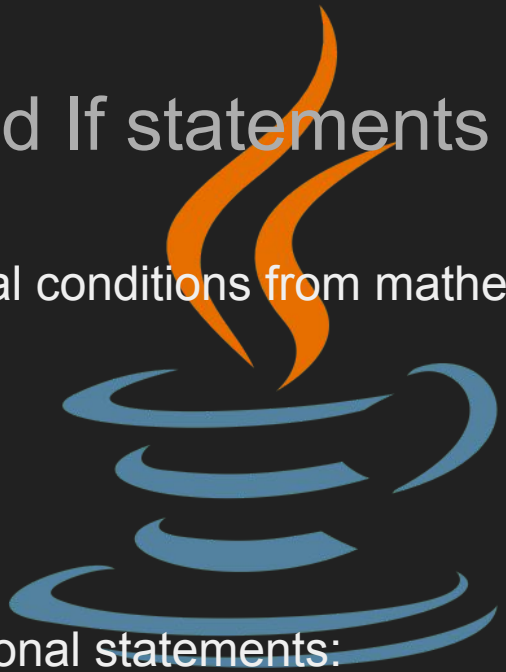
Operator	Name	Example
==	Equal to	x == y
!=	Not equal	x != y
>	Greater than	x > y
<	Less than	x < y
>=	Greater than or equal to	x >= y
<=	Less than or equal to	x <= y

Java Operators

Logical Operators

Operator	Name	Description	Example
&&	Logical and	Returns true if both statements are true	x < 5 && x < 10
||	Logical or	Returns true if one of the statements is true	x < 5 || x < 4
!	Logical not	Reverse the result, returns false if the result is true	!(x < 5 && x < 10)

Java Conditions and If statements



Java supports the usual logical conditions from mathematics:

- Less than: a < b
- Less than or equal to: a <= b
- Greater than: a > b
- Greater than or equal to: a >= b
- Equal to: a == b
- Not equal to: a != b

Java has the following conditional statements:

- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false
- User switch to specify many alternative blocks of code to be executed

Java Conditions and If statements

Example of if and else if statement:

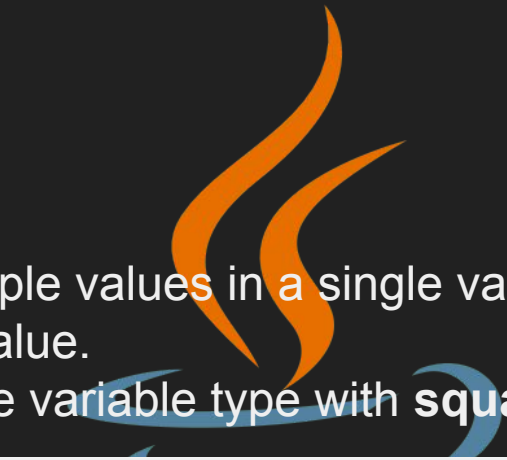
```
int time = 22;
if (time < 10) {
  System.out.println("Good morning.");
} else if (time < 20) {
  System.out.println("Good day.");
} else {
  System.out.println("Good evening.");
}
// Outputs "Good evening."
```

Java Switch

Use the switch statement to select one of many code blocks to be executed.

```
int day = 4;
switch (day) {
  case 1:
    System.out.println("Monday");
    break;
  case 2:
    System.out.println("Tuesday");
    break;
  case 3:
    System.out.println("Wednesday");
    break;
  case 4:
    System.out.println("Thursday");
    break;
  case 5:
    System.out.println("Friday");
    break;
  case 6:
    System.out.println("Saturday");
    break;
  case 7:
    System.out.println("Sunday");
    break;
}
// Outputs "Thursday" (day 4)
```

Java Arrays



Arrays are used to store multiple values in a single variable, instead of declaring separate variables for each value.
To declare an array, define the variable type with **square brackets:**

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
System.out.println(cars[0]);
// Outputs Volvo
```

Java ArrayList

The ArrayList class is a resizable array, which can be found in the java.util package.
The difference between a built-in array and an ArrayList in Java, is that size of an array cannot be modified (if you want to add or remove elements from array, you have to create a new one). While elements can be added and removed from an ArrayList whenever you want.

```java
import java.util.ArrayList;

public class MyClass {
  public static void main(String[] args) {
    ArrayList<String> cars = new ArrayList<String>();
    cars.add("Volvo");
    cars.add("BMW");
    cars.add("Ford");
    cars.add("Mazda");
    System.out.println(cars);
  }
}
```

To remove element in the array

```java
cars.remove(0);
```

Java While loop

Loops

- Can execute a block of code as long as a specified condition is reached
- Are handy because they save time, reduce errors, and they make code more readable.

While Loop

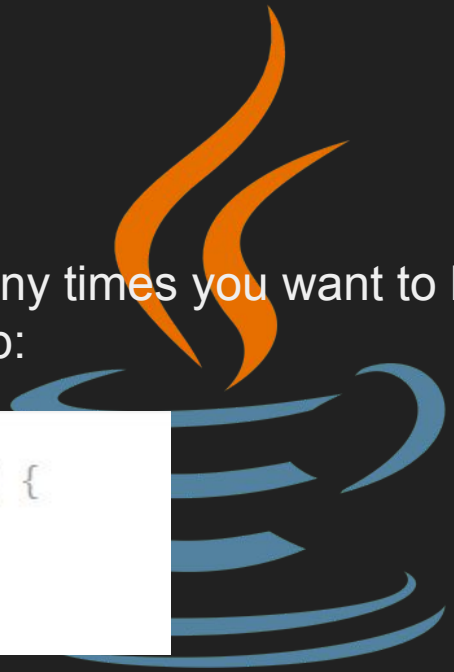
- While loops through a block of code as long as a specified condition is true:

```
int i = 0;
while (i < 5) {
  System.out.println(i);
  i++;
}
```

Java For loop

When you exactly know how many times you want to loop through a block of code, use for loop instead of while loop:

```
for (int i = 0; i < 5; i++) {
  System.out.println(i);
}
```



Java For-each loop

For-each

- Is exclusively to loop through elements in an array:

```
String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};
for (String i : cars) {
  System.out.println(i);
}
```