KHOUILID Abdelkbir & EL KADIRI Soufiane

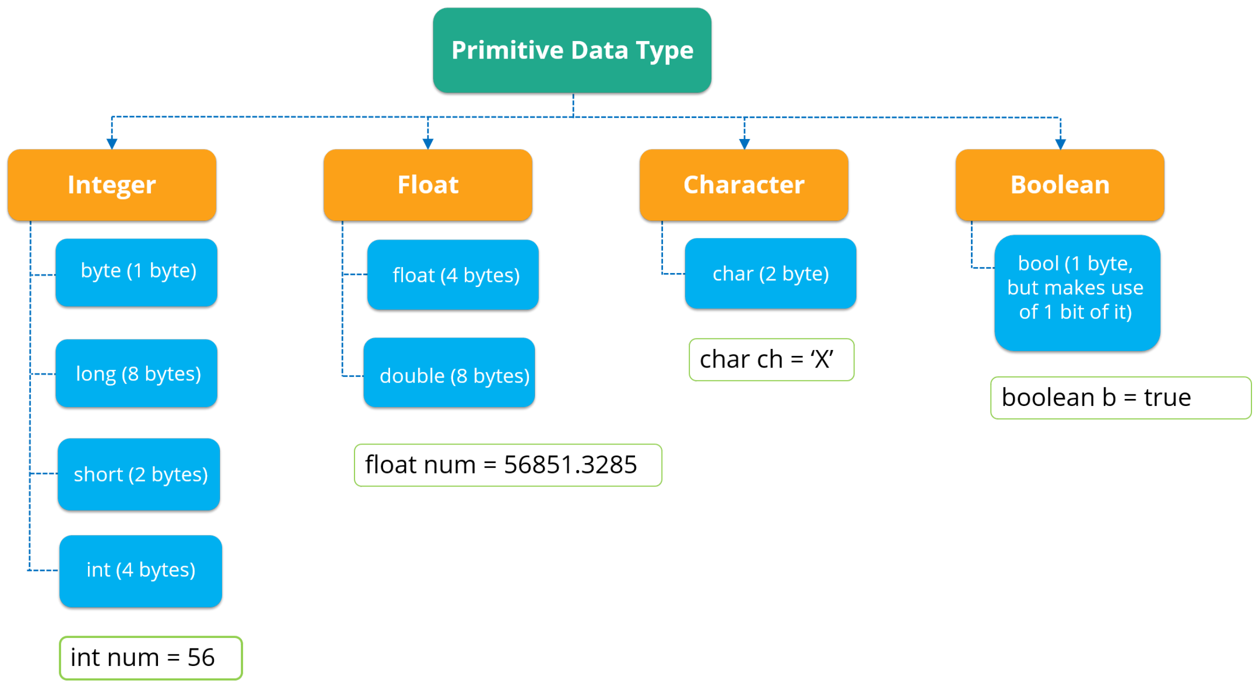
Déclaré variable:

Pour déclaré un variable nous utilisons la forme suivent:

```
int age = 21;
```

Java variables types:

il y a des autre type :



Les opérations :

Copy of Opératios

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 by another	x / y
%	Modulus	Returns the division remainder	x % y

les types de Reference variables:

Nous utilisions cette type de variable pour stocké des complexe values,

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

Examples of non-primitive(reference) types:

- Strings

```
String SayHello = "Hello";
```

- Arrays

```
  int[] ids = { 1,2,3,45,1 };
```

Parceque Array est un variable de type reference it accepte multiple method like **sort()** :

```
int[] Ids = {1,2,34,5,2};
Arrays.sort(Ids);
System.out.println(Arrays.toString(Ids));
```

Conditions:

In java we have two type of condition : switch case & id statement.

Its looks exactly like condition in JavaScript, so we don't need to explain it.

Switch case:

```
System.out.print("Please enter the operation type! : ");
//get the operation simbole
String operation = calc.nextLine();
switch (operation) {
            case "+" -> System.out.println(number_1 + number_2);
            case "-" -> System.out.println(number_1 - number_2);
            case "/" -> System.out.println(number_1 / number_2);
            case "*" -> System.out.println(number_1 * number_2);
            case "%" -> System.out.println(number_1 % number_2);
            default -> System.out.println("Sorry we don't provide this operation!!");
        }
```

If statement:

```
if ("+".equals(operation)) {
            System.out.println(number_1 + number_2);
        } else if ("-".equals(operation)) {
            System.out.println(number_1 - number_2);
        } else if ("/".equals(operation)) {
            System.out.println(number_1 / number_2);
        } else if ("*".equals(operation)) {
            System.out.println(number_1 * number_2);
        } else if ("%".equals(operation)) {
            System.out.println(number_1 % number_2);
        } else {
            System.out.println("Sorry we don't provide this operation!!");
        }
```

Loops:

For loops:

We can use this loop when we know the end of the loop;

```
int[] myArray = {1,2,42,3,534};
for (int i = 0; i < myArray.length; i++){
    System.out.println("Index of " + myArray[i] + " is " + i);
}
```

For Each loop:

this loop maybe its look like for loop but it's not :

```
String[] cars = {"ford", "ferari", "bmw"};
for (String car:cars){
    System.out.println(cars);
}
```

While loops:

The opposite of for loop, we can use while loop when we don't know the end of the loop.

```
Scanner input = new Scanner(System.in);
String input_value = "";
while(!input_value.equals("exit")){
      System.out.print("input : ");
      input_value = input.nextLine().toLowerCase().trim();
      System.out.println("Your input is :" +input_value);
}
```

Also we have **do ... while loop** id like while loop but execute at list one time, and the resent behind this is do while loop we check the condition last:

```
do{
      System.out.print("input : ");
      input_value = input.nextLine().toLowerCase().trim();
      System.out.println("Your input is :" +input_value);
}while(!input_value.equals("exit"))
```

Methods:

in java we have two type methods: normal methods(that we known) & method overloading.

Normal methods;

this methods we already know it, like in JavaScript, but in java is the methods return something we should declare the type that thing.

```
static int plusMethodInt(int x, int y) {
  return x + y;
}
```

Otherwise, if the methods doesn't return anything we declare it **void :**

```
static void plus(int x, int y) {
   System.out.println(x + y;);
}
```

if the methods accept parameters we should also declare its type like in this example ☝🏼☝🏼☝🏼☝🏼☝🏼

Method Overloading:

With **method overloading**, multiple methods can have the same name with different parameters:

```
int myMethod(int x)
float myMethod(float x)
double myMethod(double x, double y)
```

Scopes(*this article coped from w3school*)

In Java, variables are only accessible inside the region they are created,This is called scope.

Block scopes

block of code refers to all of the code between curly braces {}. Variables declared inside blocks of code are only accessible by the code between the curly braces, which follows the line in which the variable was declared:

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

    // Code here CANNOT use x

    { // This is a block

      // Code here CANNOT use x

      int x = 100;

      // Code here CAN use x
      System.out.println(x);

   } // The block ends here

  // Code here CANNOT use x

  }
}
```

Collections:

ArrayList:

Array list is an array but:

→ normal array in java we can't its size.

→ otherwise, **ArrayList** we let us do that.

```
ArrayList<String> cars = new ArrayList<String>(); // Create an ArrayList object
```

Add new item to car listArray we use add():

```
cars.add("Volvo");
```

And if we wanna access this items we use get();

```
Cars.get(0);
```

there is other methods that we can use : set(), remove() …

LikedList:

this type of collections its almost like ArrayList, but there is a deferent. (you can see the deferent between them *here*) .

lets see when we can use each one:

It is best to use an `ArrayList` when:

- You want to access random items frequently
- You only need to add or remove elements at the end of the list

It is best to use a `LinkedList` when:

- You only use the list by looping through it instead of accessing random items
- You frequently need to add and remove items from the beginning or middle of the

hashMap:

this type is like object in JavaScript, we can store items and get them a key to access later.

we should declare both type of the key and his value.

```
HashMap<String, int> Cars = new HashMap<String, int>();
```

to add new items we use put() method.

```
Cars.put("BMW", 200_000);
```

if we wanna access this items we use get():

```
Cars.get("BMW");
```

and we have remove() & clear() methods.

New thing that we can loop thought the hasList by using for each.

```
for (String i : Cars.keySet()) {
  System.out.println(i);
}
```

Use the **keySet()** method if you only want the keys, and use the values() method if you only want the values.

if we want access the value we use **value()**;

```
for (String i :Cars.values()) {
  System.out.println(i);
}
```

hashSet:

this type is like an Arraylist & linkedList,but its collection of items where every item is unique.

we can also use **add()** & **get() & remove()** methods,and ...

contains() : To check whether an item exists in a HashSet.