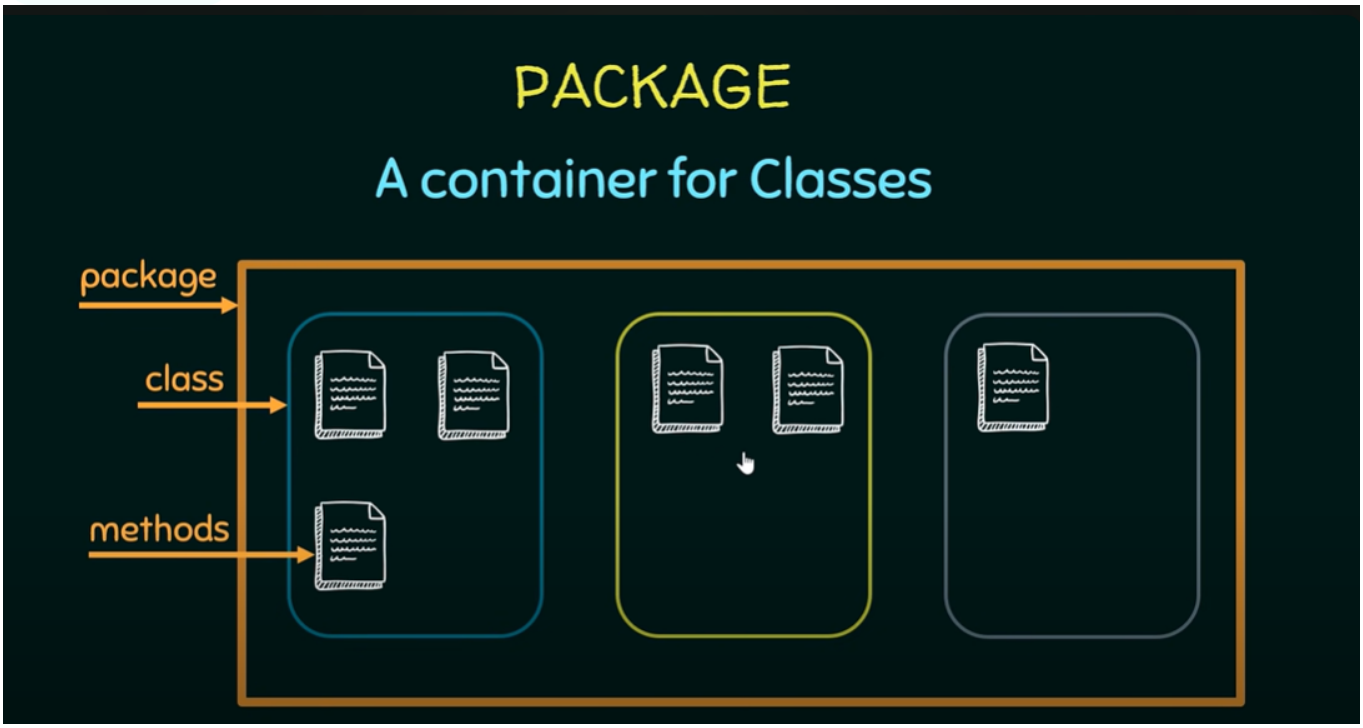
BASICS

#Packages



Class , Object and Method

class

#class

- blueprint for a program

Object

#Object

- an instance of class

initiating Object

```
String course=new String("Neso academy")
```

- String(class name) course(Object name) .
- new is the keyword the word to create object .
- "String("Neso academy)" is a special method called CONSTRUCTOR.

Method

#Method

- Group of instruction to do some specific task.
- every Method inside a class
- class is container of methods
-
```
public static void main(String[] args) {

}
```

Public(access modifier) Static void(return type) name

- Is an Method
- Entry for a program.

Access modifier of class, method

Specific how to access class ,methods and fields

#Accessmodifier

Public

- Access level in everywhere(Inside and outside packages ,inside and outside classes)

Private

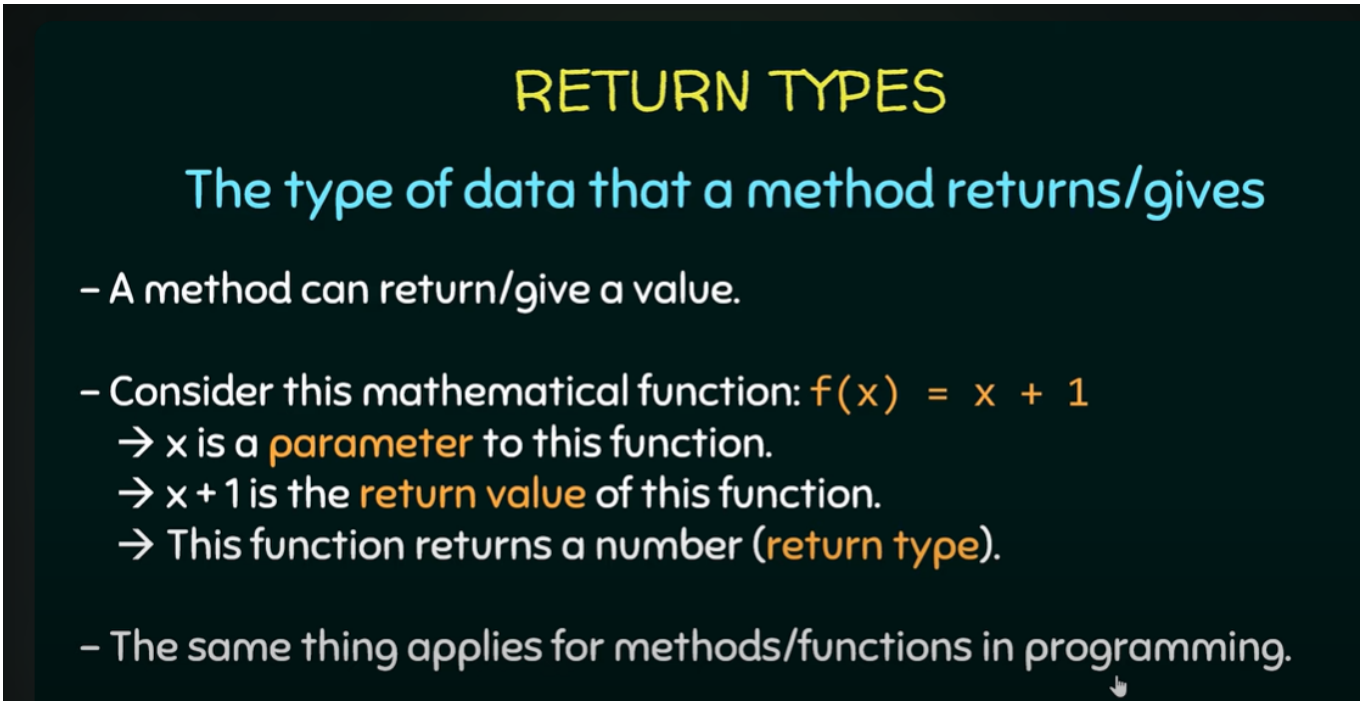
- Access level is only inside class

Static keyword

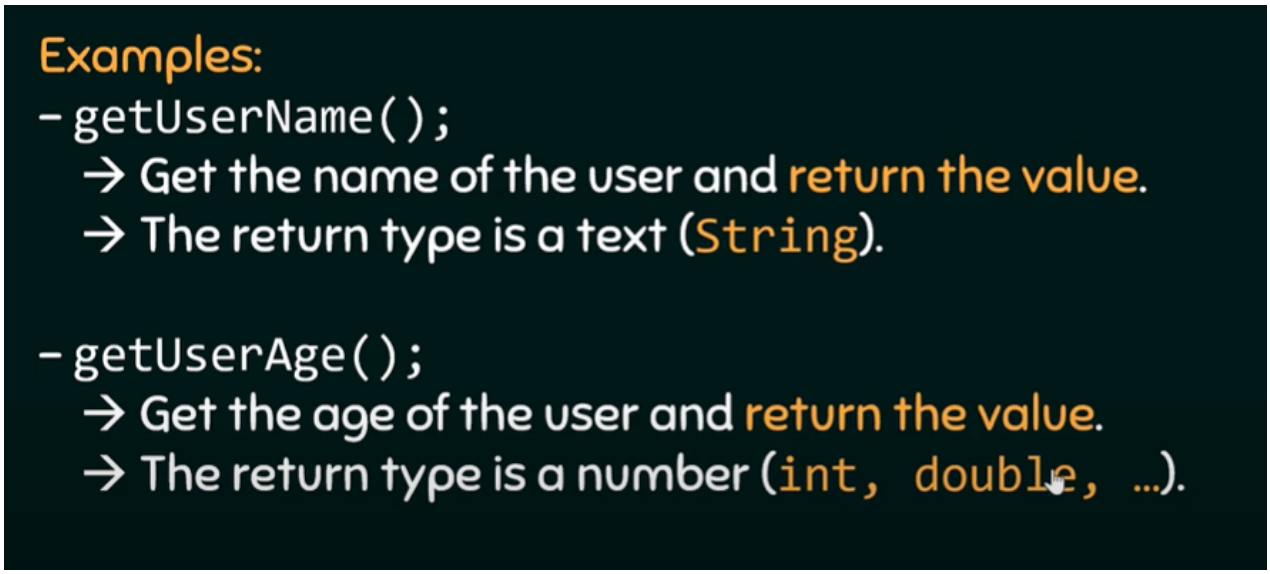
#Statickeyword

Public static or private static where static is allows the access methos and class by ".".we can access only by static keyword.

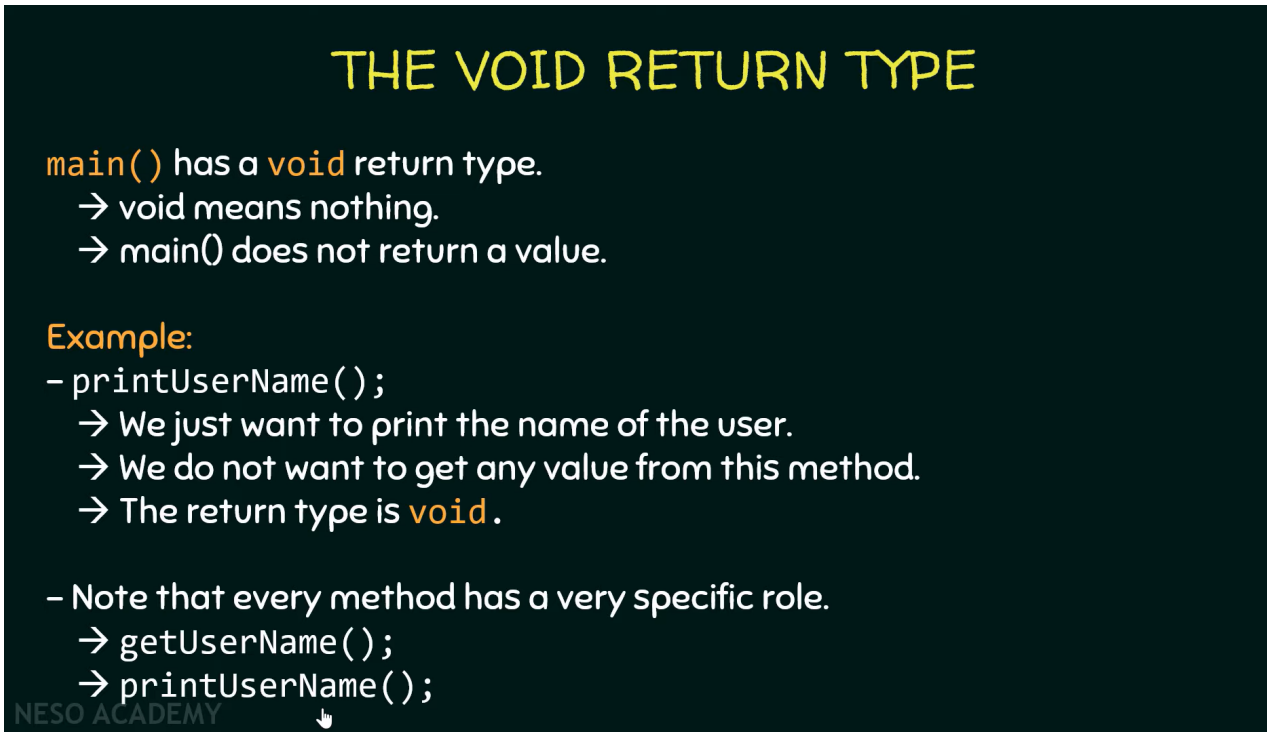
Return type



Return type



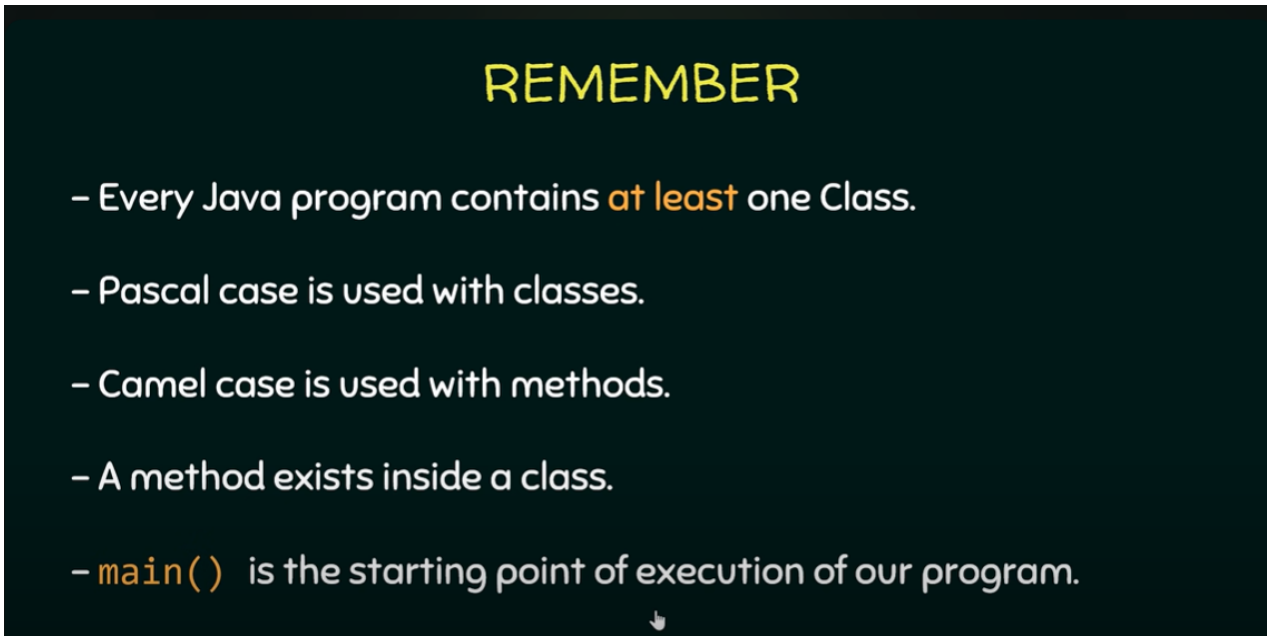
Void Return type



Naming

- pascal case->This Is A Name

- Camel case->this Is A Name



Structure

```
1  package com.prakii;
2
3  //TIP To <b>Run</b> code, press <shortcut actionIc
4  // click the <icon src="AllIcons.Actions.Execute"/
5  public class Main {
6      public static void main(String[] args) {
7          System.out.println("HELLO WORLD");
8      }}
```

System.out

#Systemout

- 'out' is an object of Print Stream class.
- out is an camel case

- print and println are methods in print Stream class where it can be called by using ".out".
- System is class(Pascal case)
- out is in system field. where out can access by "." System.
- System.out.println->println method can access by out. where it can access by system.

Types of errors

#errors

Syntax error

whenever you error related to java syntax.

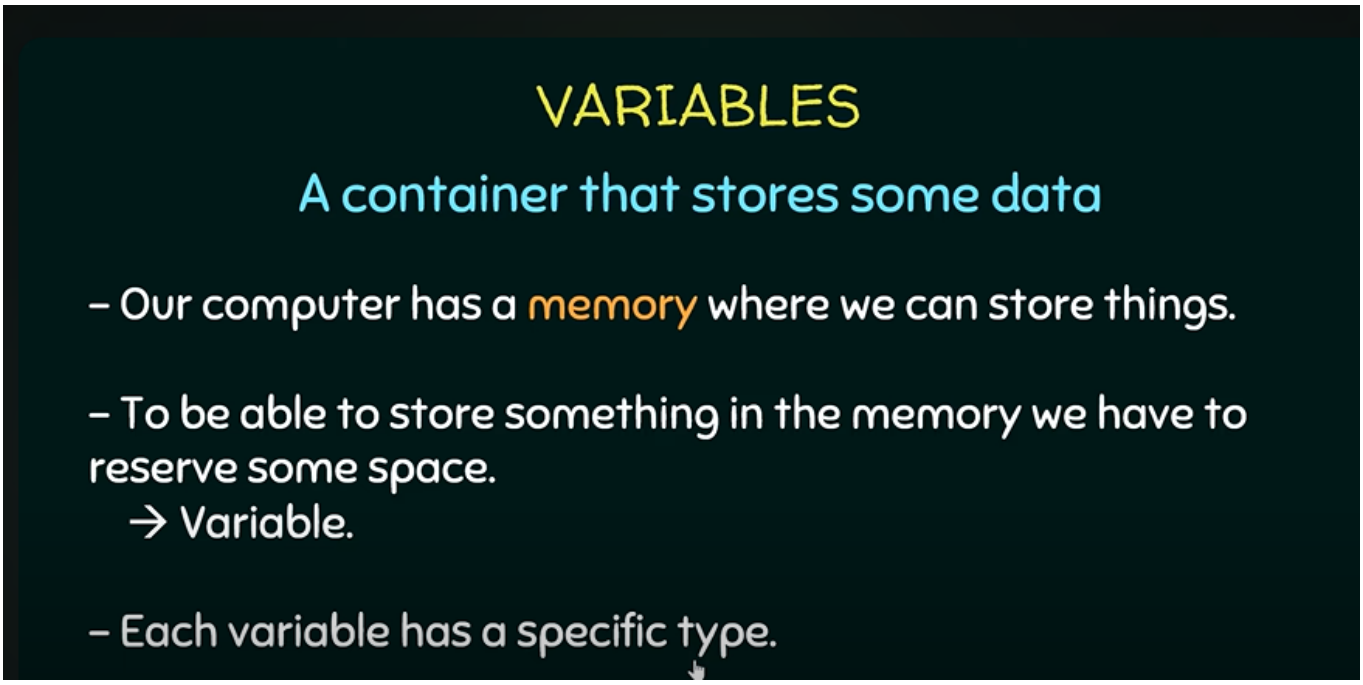
Run time error

we have identify during the execution of code.

Logical Error

error arise by user.it can be avoid by careful while coding

VARIABLES AND ITS SIZE



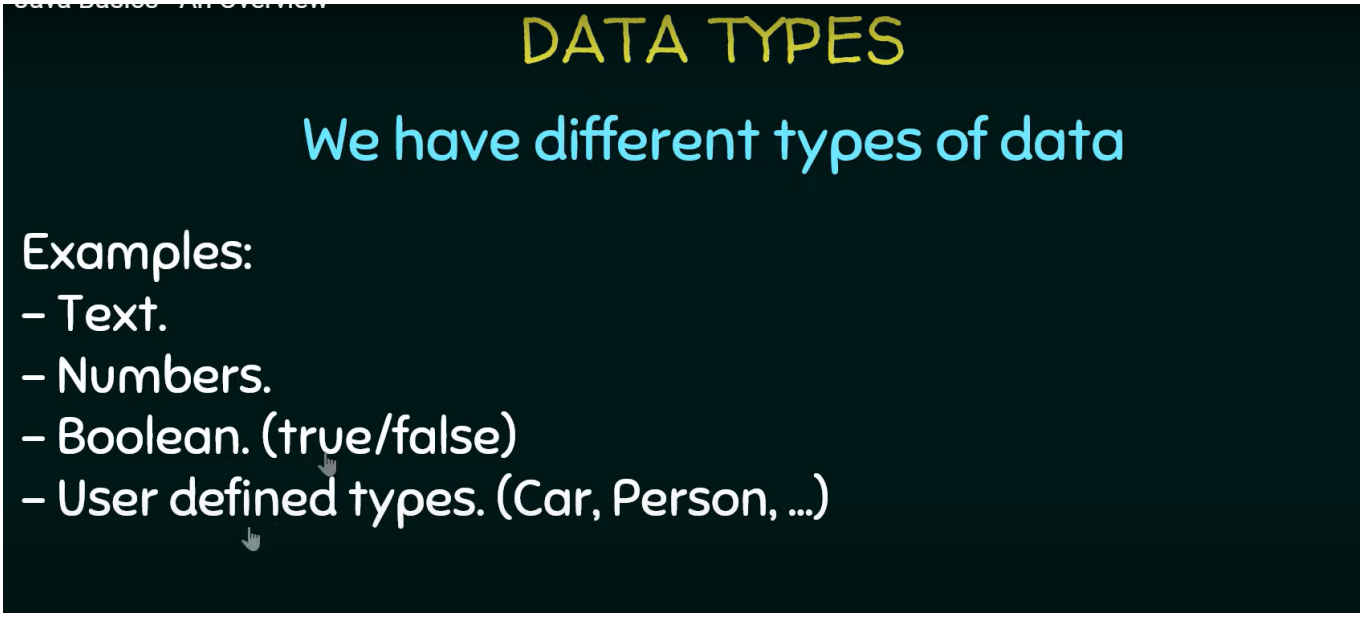
Type	Size (bits)	Minimum	Maximum	Example
byte	8	-128	127	byte b = 100;
short	16	-32,768	32,767	short s = 30_000;
int	32	-2147483648	2147483647	int i = 100_000_000;
long	64	-9223372036854775808	9223372036854775807	long l = 100_000_000_000_000;
float	32	-2-149	(2-2-23)·2127	float f = 1.456f;
double	64	-2-1074	(2-2-52)·21023	double f = 1.456789012345678;
char	16	0	216– 1	char c = 'c';
boolean	1			boolean b = true;

Variable Declaration

```java
public class Main {
    public static void main(String[] args) {
        // primitives
        int number1 = 20;
        int number2 = 20;
        int result = number1 + number2;
        System.out.println(result);
        double pi = 3.14;
        boolean isAdult = true;
        char a = 'A';
    }
```

Datatypes

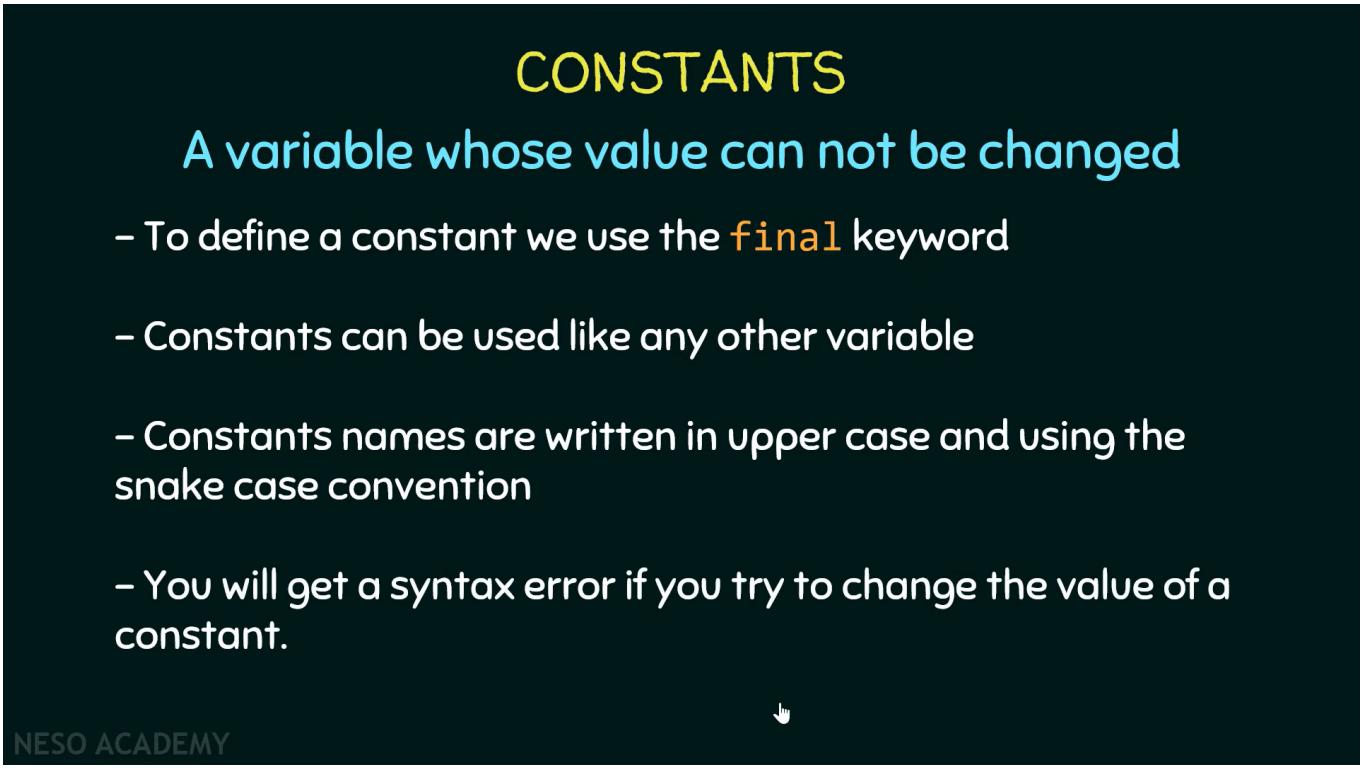
#Datatypes

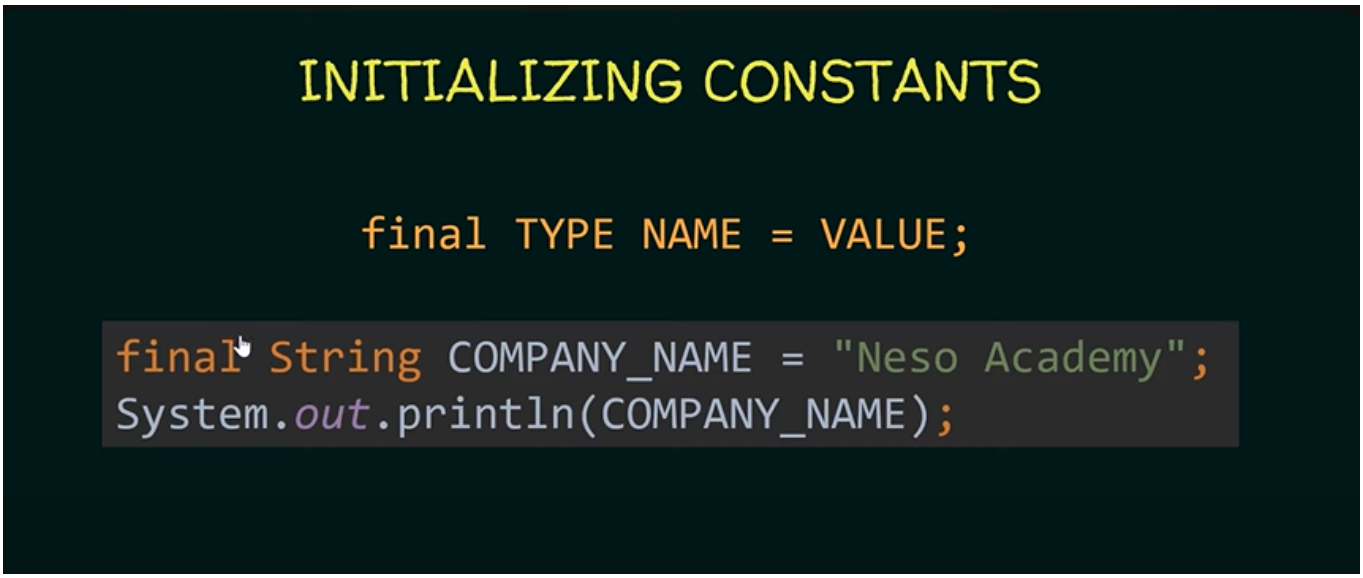


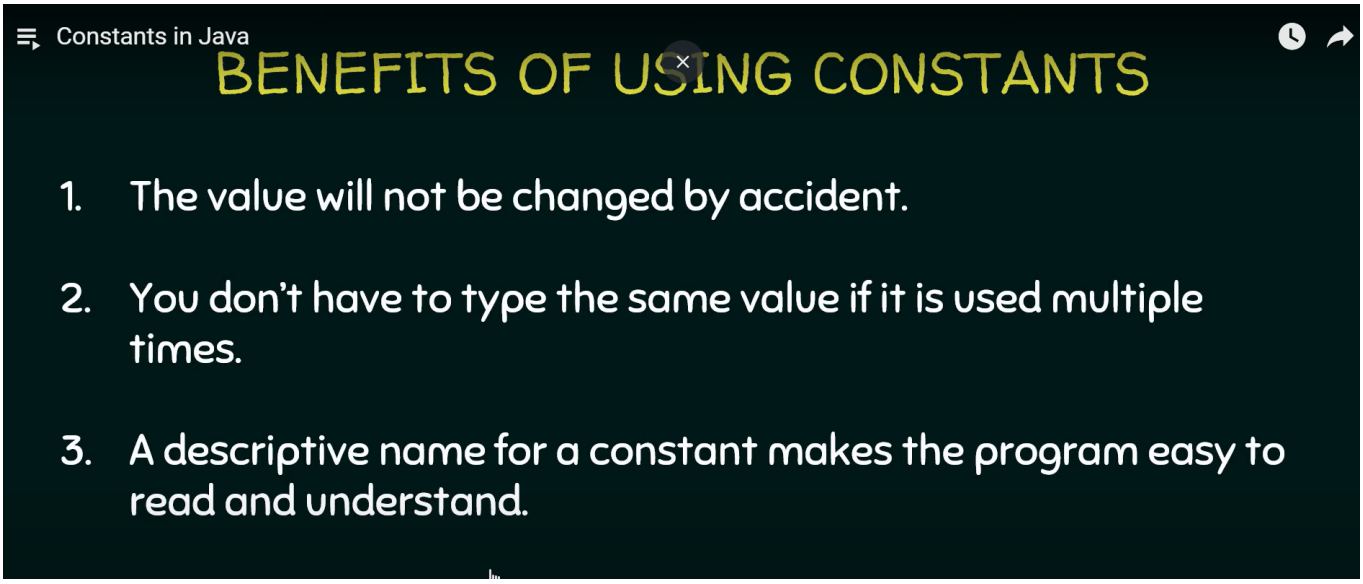
CONSTANTS

#CONSTANTS

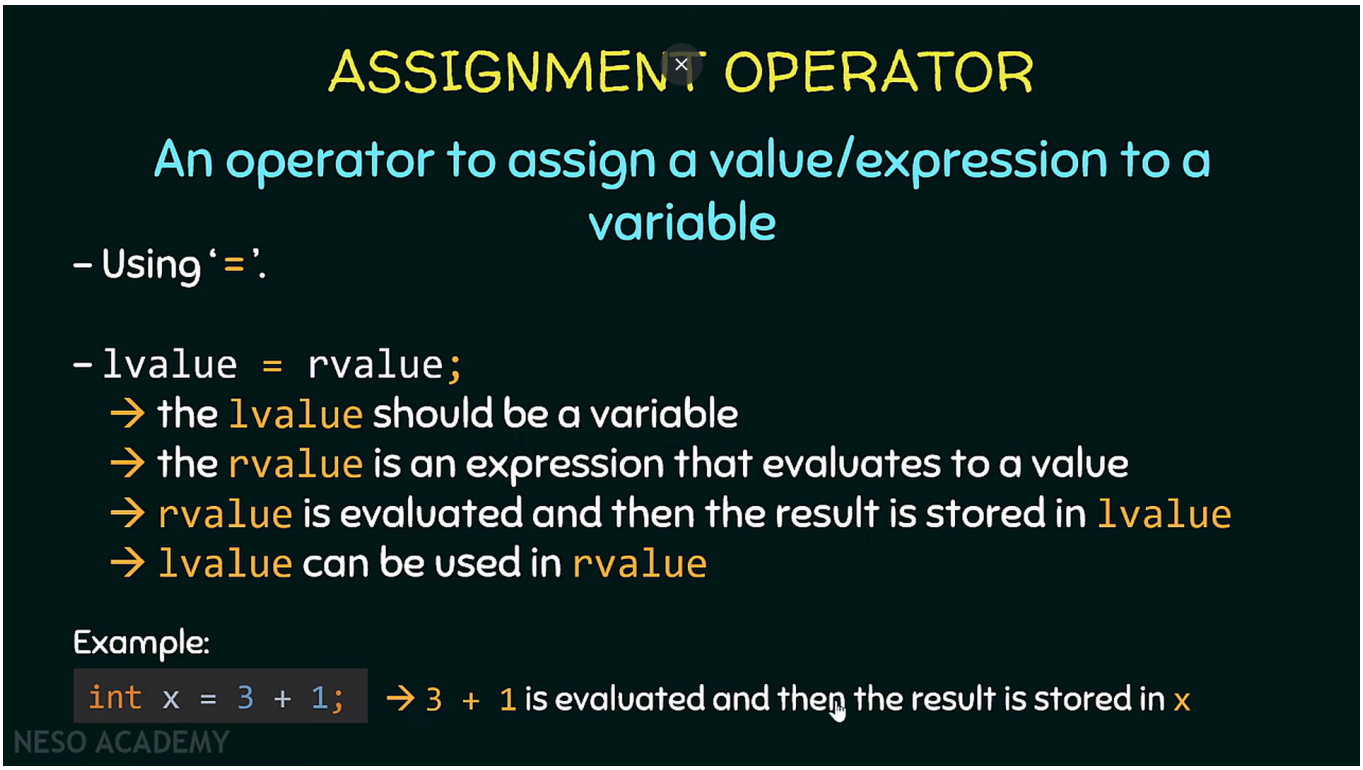
KEYWORD "final".



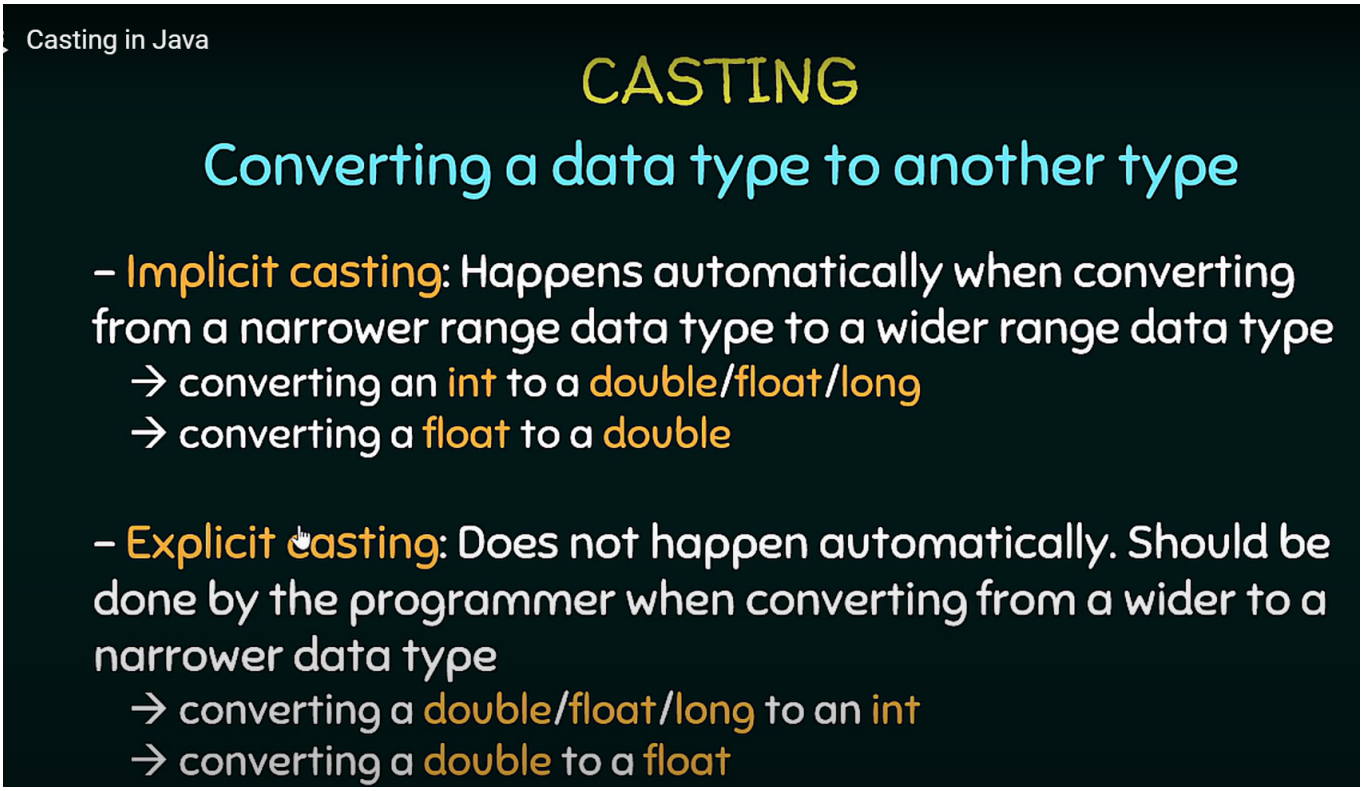




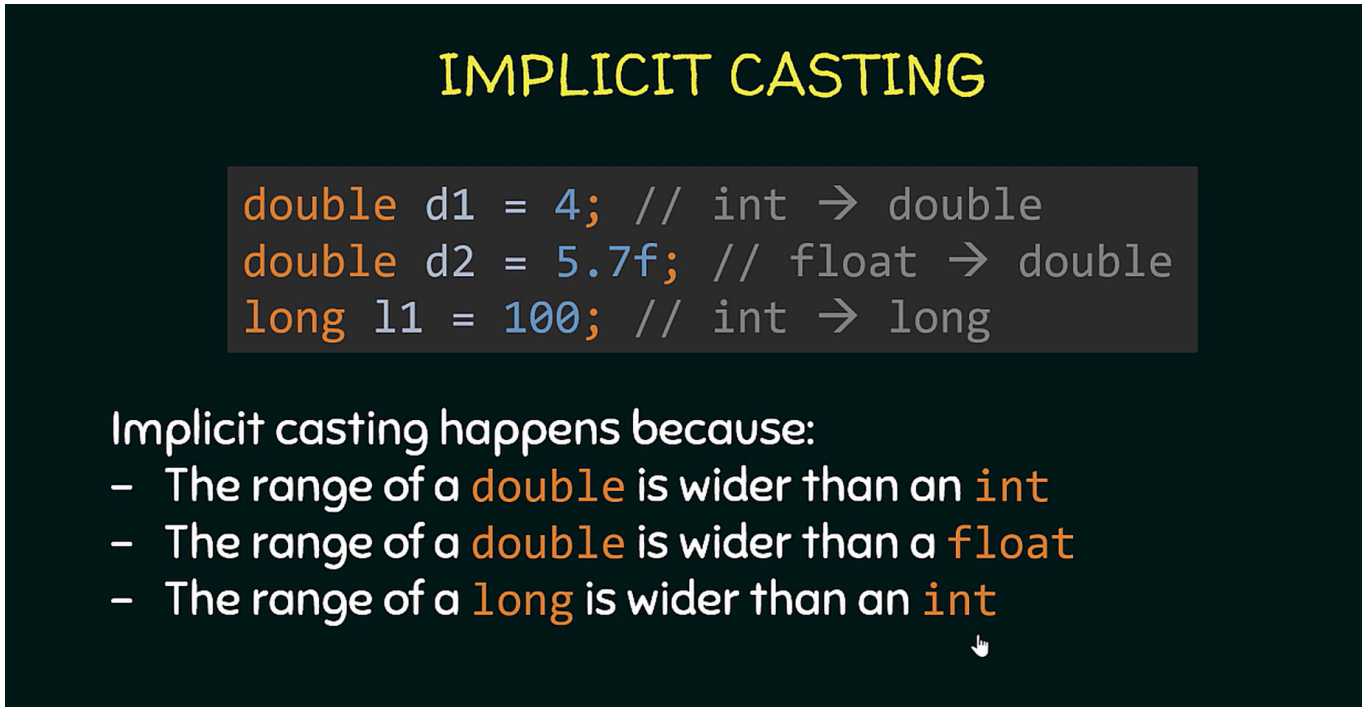
Assignment Operator


ASSIGNMENT OPERATOR
An operator to assign a value/expression to a variable
– Using '='.
– lvalue = rvalue;
→ the lvalue should be a variable
→ the rvalue is an expression that evaluates to a value
→ rvalue is evaluated and then the result is stored in lvalue
→ lvalue can be used in rvalue
Example:
int x = 3 + 1;
→ 3 + 1 is evaluated and then the result is stored in x
NESO ACADEMY

Casting


Casting in Java
CASTING
Converting a data type to another type
– Implicit casting: Happens automatically when converting from a narrower range data type to a wider range data type
→ converting an int to a double/float/long
→ converting a float to a double
– Explicit casting: Does not happen automatically. Should be done by the programmer when converting from a wider to a narrower data type
→ converting a double/float/long to an int
→ converting a double to a float

IMPLICIT CASTING



EXPLICIT CASTING

EXPLICIT CASTING

```
int i1 = 4.5; // ERROR
int i2 = 8L; // ERROR
float f1 = 4.5; // ERROR
```

Implicit casting can not happen because:

- The range of an `int` is narrower than a `double`
- The range of an `int` is narrower than a `long`
- The range of a `float` is narrower than a `double`

→ Explicit casting is used

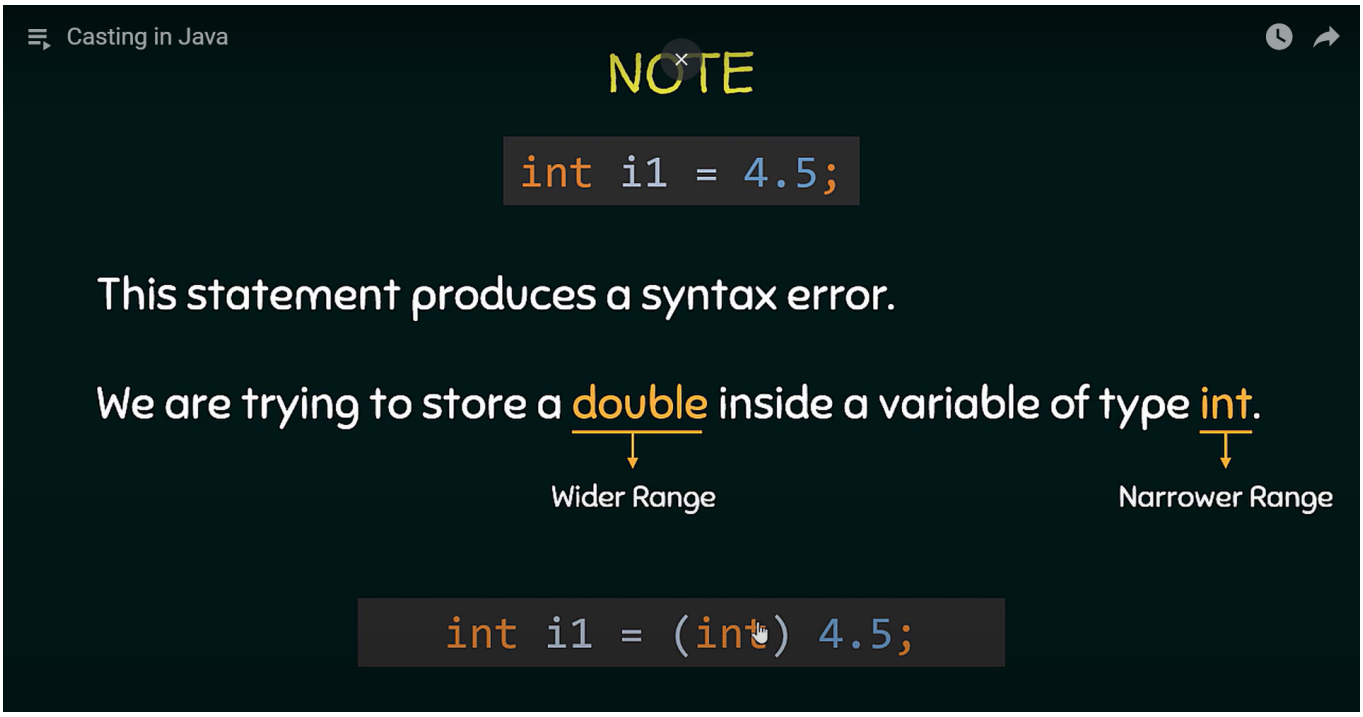
EXPLICIT CASTING

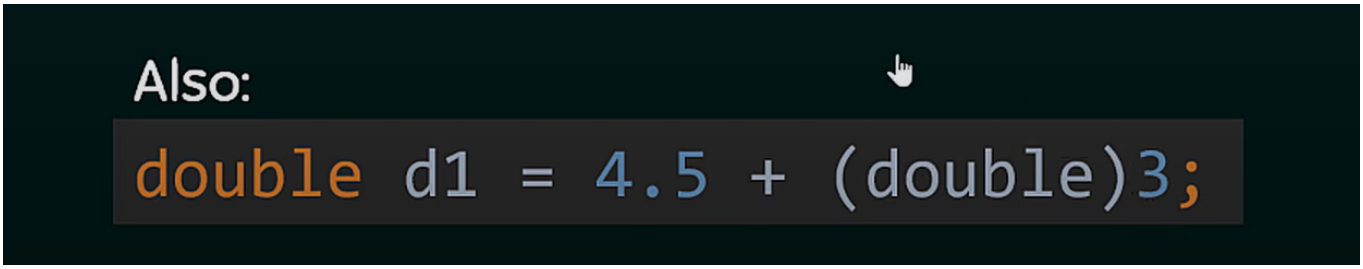
(new data type) expression

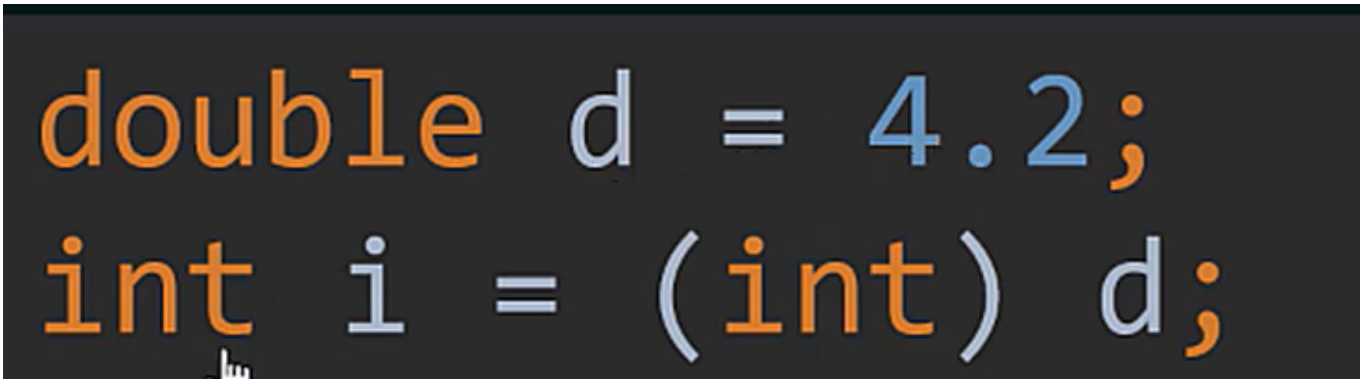
```
int i1 = (int) 4.5;
int i2 = (int) 8L;
float f1 = (float) 4.5;
```

The programmer tells Java to do the casting:

- (int) 4.5 → 4 (data loss)
- (int) 8L → 8
- (float) 4.5 → 4.5F







OPERTORS

ARITHMETIC OPERTORS(+,-,*,%,/)

```java
public class Main {
    public static void main(String[] args) {
        int numberOne = 10;
        int numberTwo = 3;
        System.out.println(numberOne + numberTwo);
        System.out.println(numberOne - numberTwo);
        System.out.println(numberOne * numberTwo);
        System.out.println(numberOne / numberTwo);
        System.out.println(numberOne % numberTwo);
        // S-Subtraction
        System.out.println((2+2) * (3/1) * 2);
    }
```

**Increment

```java
public static void main(String[] args) {
    int number = 0;
    System.out.println(number++);
    System.out.println(number);
    int numberTwo = 0;
    System.out.println(++numberTwo);
    System.out.println(numberTwo);
```

Decrement

```java
public class Main {
    public static void main(String[] args) {
        int number = 0;

        System.out.println(number--);
        System.out.println(number);
        int numberTwo = 0;
        System.out.println(--numberTwo);
        System.out.println(numberTwo);
```

```
/Users/amigoscode/Library/Java/JavaVirtualMachines/oper
0
-1
-1
-1
```

- x+=1->x=x+1
- x-=1->x=x-1

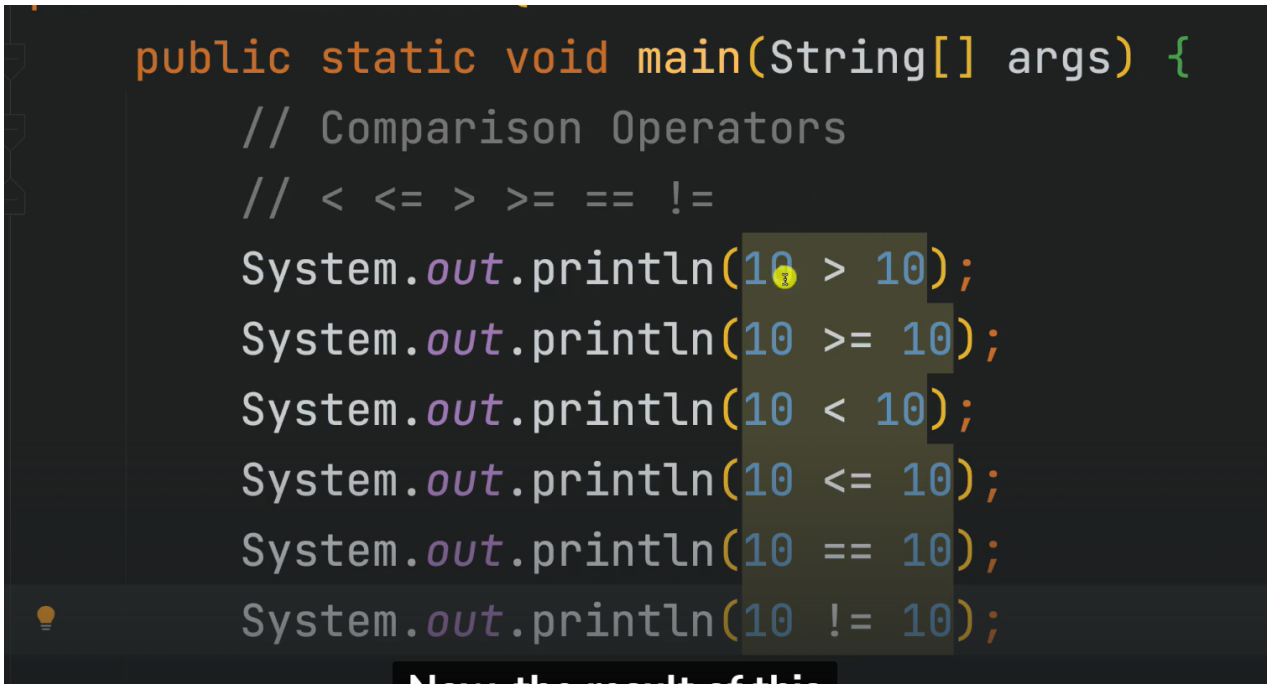
Division Operator I/P & O/P

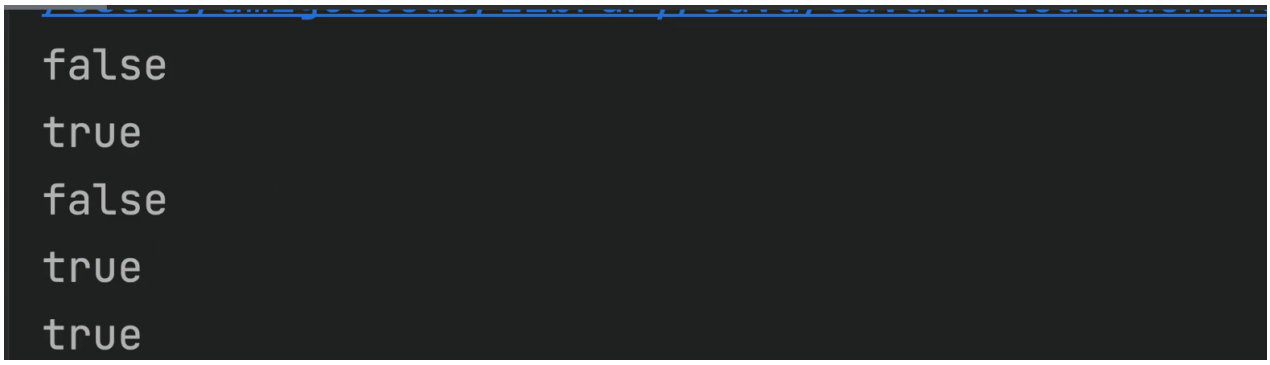
```java
int i1 = 1 / 2; // OK, int = int
int i2 = 1.0 / 2; // ERROR, int = double
int i3 = (int) (1.0 / 2); // OK, int = int
int i4 = (int) (1.0f / 2); // OK, int = int
int i5 = (int) 1.0f / 2; // OK, int = int
int i6 = 1.0f / (int) 2; // ERROR, int = float
double i7 = 1.0 / 2; // OK, double = double
double i8 = 1 / 2; // OK, double = int
double i9 = (double) 1 / 2; // OK, double = double
float i10 = 1.0f / 2.0f; // OK, float = float
float i11 = 1 / 2; // OK, float = int
```

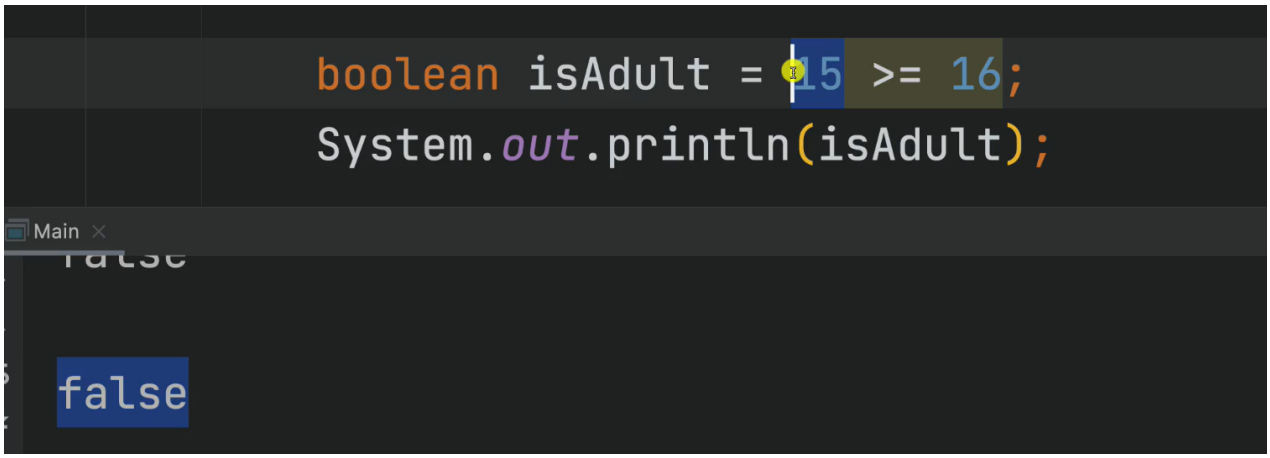
```java
int i1 = 3;
int i2 = 2;
double d1 = 2;

System.out.println(i1 / i2); // 1
System.out.println(i1 / d1); // 1.5
System.out.println((double) i1 / i2); // 1.5
System.out.println(i1 / (double) i2); // 1.5
System.out.println((double) i1 / (double) i2); // 1.5
System.out.println((double) (i1 /i2)); // 1.0
```

RELATIONAL OPERATOR(COMPARISON OPERATORS)

- it retrieves only Boolean Values as output.
- it compares to values

- 

- 

- 

LOGICAL OPERATORS(AND,OR,NOT,.....)

```
public static void main(String[] args) {
    boolean Prakii= true;
    boolean prakash = false;
    System.out.println(Prakii && prakash);
```

```
    System.out.println(Prakii && !prakash);
    System.out.println(Prakii || prakash);
    System.out.println(!Prakii || !prakash);
    System.out.println(10>=2||2>=3);
}
```

Ternary Operator

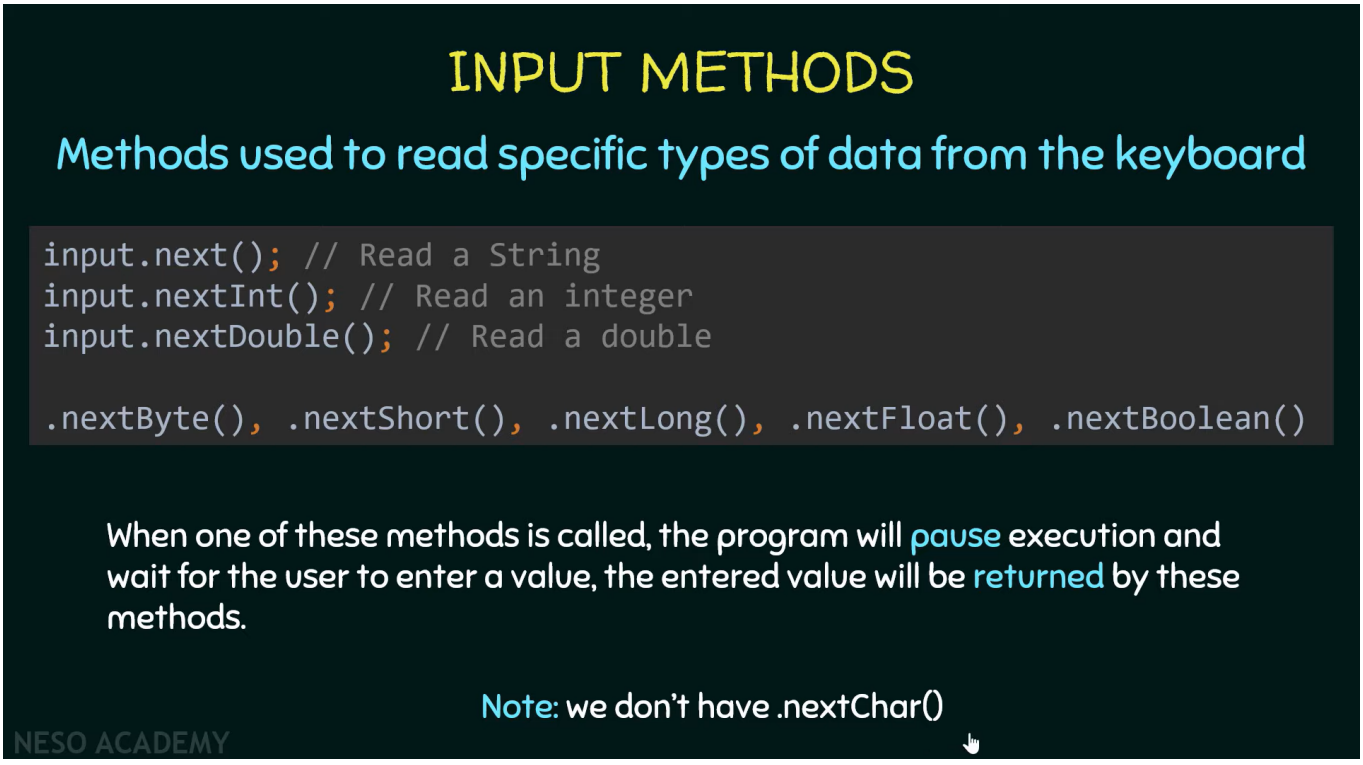
```
public class Main {
    public static void main(String[] args) {
        Scanner Age = new Scanner(System.in);
        int age = Age.nextInt();
     String Adult = age>18 ? "Iam an Adult":"Iam an child";
        System.out.println(Adult);
    }
}
```

Null value

#Nullvalue

Scanner Class

"Scanner is class which reads input from keyboard".




```java
public class Main {
    public static void main(String[] args) {
        Scanner Name = new Scanner(System.in);
        System.out.println(Name.next());
        Scanner name = new Scanner(System.in);
```

```
        System.out.println(name.nextLine());
        Scanner No = new Scanner(System.in);
        System.out.println(No.nextInt());
        Scanner no = new Scanner(System.in);
        System.out.println(no.nextDouble());
    }
}
```

```
ish prakii
ish
 prakii
3434
3434
234
234.0
```

String

```
{
    public static void main(String[] args) {
        String I = "Its";
        String P = "_Prakii";
        String Prakii = I + P;
        System.out.println(I);
        System.out.println(P);
        System.out.println(Prakii);
        System.out.println(Prakii.toUpperCase());
        System.out.println(Prakii.toLowerCase());
        System.out.println(Prakii.substring(4));
        System.out.println("A".isBlank());
        System.out.println("  ".isBlank());
        System.out.println(" ".isEmpty());
        System.out.println("".isEmpty());
        System.out.println("    a    ");
        System.out.println("    a    ".trim());
```

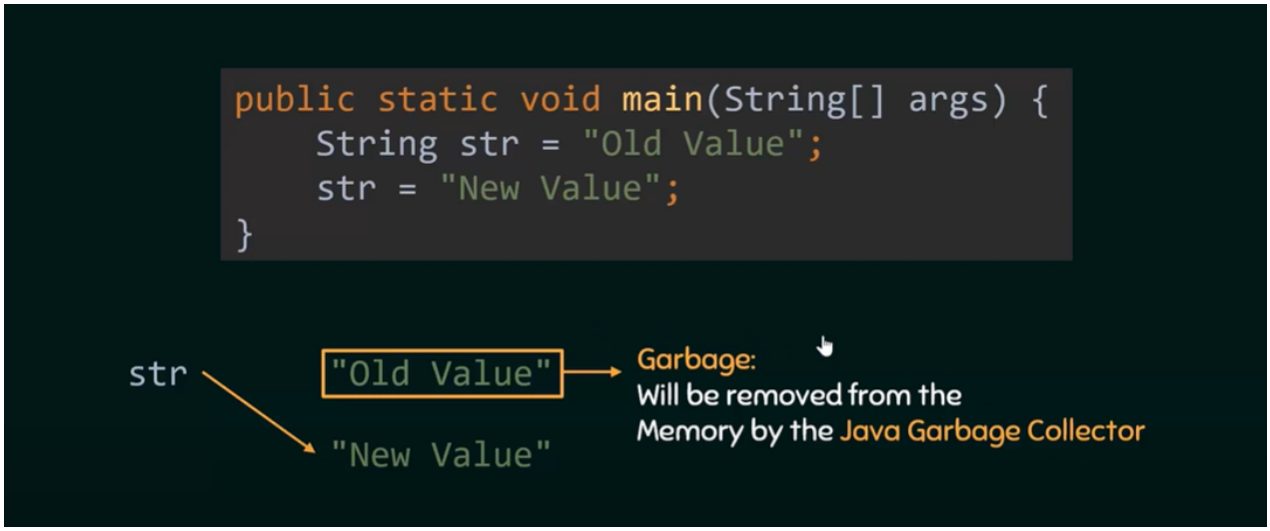
```
        System.out.println(I.charAt(2));
        System.out.println(I.indexOf('t'));
        String op = Prakii.concat("0888");
        System.out.println(op);
        String its=new String ("Its");
        System.out.println(Prakii.contains("Its"));
        System.out.println(Prakii.contains("its"));

    }
}
```

o/p

```
Its
_Prakii
Its_Prakii
ITS_PRAKII
its_prakii
Prakii
false
true
false
true
    a
a
s
1
Its_Prakii0888
true
false
```

new vs =

Strings are Immutable

- Immutable means it cannot be changed.
- that Means reference Type are immutable.
- string is an Immutable.
- 

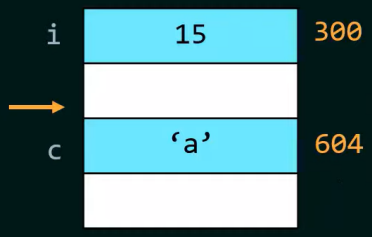
Primitive and Reference Type

PRIMITIVE TYPES

Types that hold simple values

byte, short, int, long, float, double, and char are primitive types

```
public static void main(String[] args) {
    int i = 15;
    char c = 'a';
}
```



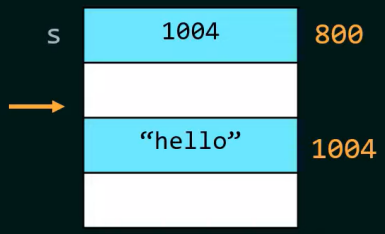
The variable contains the value

REFERENCE TYPES

Types that hold complex values (Objects)

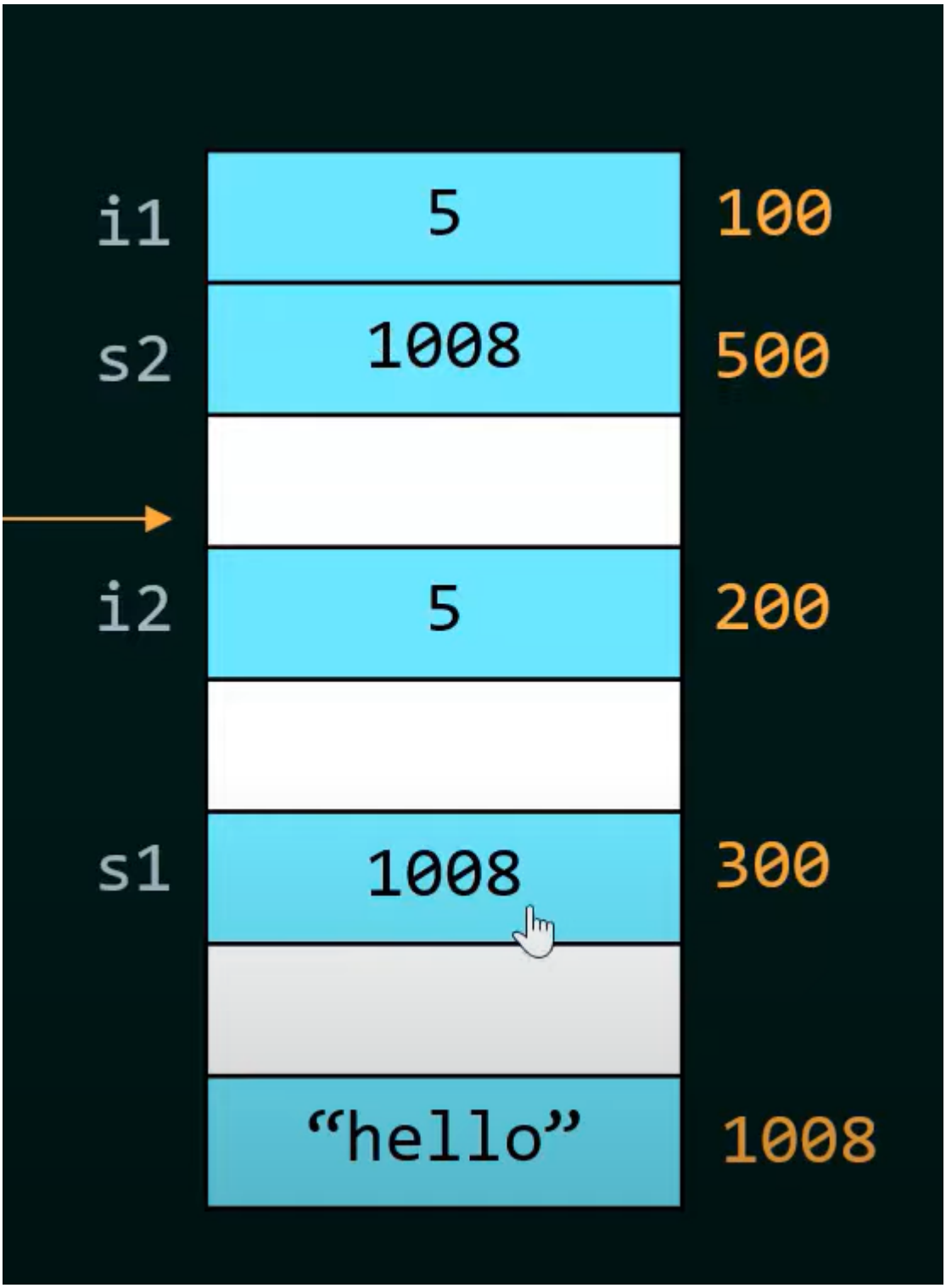
String is a reference types

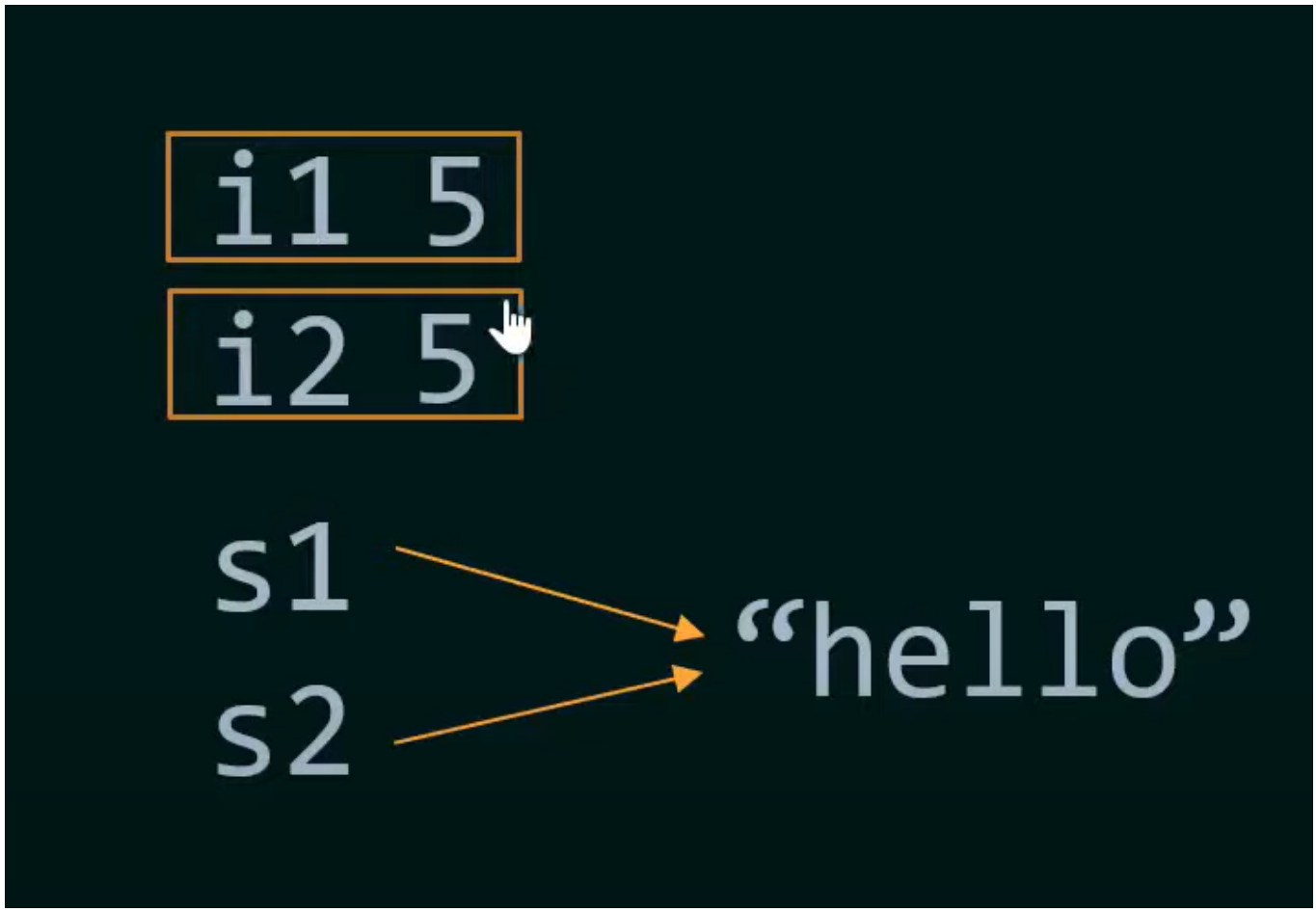
```
public static void main(String[] args) {
    String s = "hello";
}
```



The variable contains the address of the value.
The variable references the value.
s → "hello"

```java
public static void main(String[] args) {
    int i1 = 5;
    int i2 = i1;

    String s1 = "hello";
    String s2 = s1;
}
```


i1
5
100
s2
1008
500
i2
5
200
s1
1008
300
“hello”
1008



String Builder

In Java, `StringBuilder` is a mutable sequence of characters, which is used to create and manipulate strings in a more efficient manner compared to using `String`. The `String` class in Java is immutable, meaning once a `String` object is created, it cannot be changed. Each modification of a `String` creates a new `String` object, which can be inefficient in terms of memory and performance when there are many string manipulations.

`StringBuilder` addresses this by allowing modifications without creating new objects, making it more efficient for scenarios where strings are frequently appended, inserted, or modified.

Here are some key points about `StringBuilder`:

1. **Mutable**: Unlike `String`, `StringBuilder` objects can be modified after they are created.
2. **Performance**: It is faster and more memory-efficient when performing multiple string manipulations.
3. **Methods**: `StringBuilder` provides various methods to manipulate strings, such as `append()`, `insert()`, `delete()`, `reverse()`, and more.
4. **Thread Safety**: `StringBuilder` is not synchronized, which means it is not thread-safe. For thread-safe operations, you should use `StringBuffer`.

Example Usage

Here's a basic example demonstrating the use of `StringBuilder`:

```
public class StringBuilderExample {
    public static void main(String[] args) {
        // Creating a StringBuilder object
        StringBuilder sb = new StringBuilder("Hello");

        // Appending a string to the StringBuilder
        sb.append(" World");
        System.out.println(sb.toString()); // Output: Hello
World

        // Inserting a string at a specific position
        sb.insert(6, "Java ");
        System.out.println(sb.toString()); // Output: Hello
Java World

        // Reversing the contents of the StringBuilder
        sb.reverse();
        System.out.println(sb.toString()); // Output: dlroW
```

```
avaJ olleH

        // Deleting a portion of the string
        sb.delete(0, 6);
        System.out.println(sb.toString()); // Output: avaJ
olleH
    }
}
```

Key Methods

- `append(String str)`: Appends the specified string to this character sequence.
- `insert(int offset, String str)`: Inserts the specified string at the specified position.
- `delete(int start, int end)`: Removes the characters in a substring of this sequence.
- `reverse()`: Reverses the sequence of characters.
- `toString()`: Converts the `StringBuilder` object to a `String`.

When to Use

- Use `StringBuilder` when you need to perform a lot of modifications to a string, such as in loops or when constructing a string dynamically.
- Avoid `StringBuilder` in multi-threaded environments unless external synchronization is provided, as it is not thread-safe.

Using `StringBuilder` can significantly improve the performance of your Java applications when dealing with extensive string manipulations.

Break And Continue

```java
public static void main(String[] args) {
        String[] Name ={"prakii","ish","Lakshmi"};
        for (String s : Name) {
            if(s.equal("ish")){
                break;
            }
            System.out.println(s);

        }
    }
```

```
prakii
```

```java
public static void main(String[] args) {
    String[] Name ={"prakii","ish","Lakshmi"};
    for (String s : Name) {
        if(s.equals("ish")){
            continue;
        }
        System.out.println(s);

    }
}
```

```
prakii
Lakshmi
```

```java
  public static void main(String[] args) {
        String[] Name ={"prakii","ish","Lakshmi"};
        for (String s : Name) {
            if(s.startsWith("L")){
```

```
                break;
            }System.out.println(s);
        }
    }
```

```
prakii
ish
```

```
  public static void main(String[] args) {
        String[] Name ={"prakii","ish","Lakshmi"};
        for (String s : Name) {
            if(s.startsWith("i")){
                continue;
            }System.out.println(s);
        }
    }
```

```
prakii
Lakshmi
```

LOOPS

FOR LOOPS

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

   for(int i=0;i<3;i++)
   {
       System.out.println(i+" : Hello");
   }
    System.out.println("Reverse");
   for(int i=3;i>=0;i--)
```

```
    {
        System.out.println(i+" : Hello");
    }
}
```

```
0 : Hello
1 : Hello
2 : Hello
Reverse
3 : Hello
2 : Hello
1 : Hello
0 : Hello
```

While LOOP

```
public static void main(String[] args) {
    int no=0;
    while(no>10){
        System.out.println("Count"+no);
        no++;
    }
}
```

```
Count0
Count1
Count2
Count3
Count4
Count5
Count6
Count7
Count8
Count9
```

Do While LOOP

```
public static void main(String[] args) {
    int no=3;
    do{
        System.out.println("Count"+no);
        no++;
    }
    while(no<0);
}
```

```
Count3
```

If statements

```
  public static void main(String[] args) {
if(true){
    System.out.println("Its True");
}else{
    System.out.println("Its False");
}
        }
```

```
Its True
```

With condition

```
   public static void main(String[] args) {
        int age=17;
        boolean adult= age>=18;
if(adult){
    System.out.println("Is adult");
}else{
```

```
    System.out.println("Is not adult");
}
        }
```

Elseif

```
public static void main(String[] args) {
        int mark=99;
        boolean adult= mark>=90;
if(adult){
    System.out.println("excellent");
}else if(mark>=50){
    System.out.println("AVerage");
}else{
    System.out.println("Fail");
}
        }
```

```
excellent
```

Switch case

```
public static void main(String[] args) {
    Scanner gender = new Scanner(System.in);
    String Gender = gender.next();
    switch(Gender.toUpperCase()) {
        case "MALE":
            System.out.println("Iam an Male");
            break;
        case "FEMALE":
            System.out.println("Iam an Female");
            break;
```

```
        case "TRANSGENDER":
            System.out.println("Iam an Trans");
            break;
        default:
            System.out.println("Unknown Gender");
    }
    }
```

```
male
Iam an Male
```

```
public static void main(String[] args) {
    Scanner in = new Scanner(System.in);
    String fruit = in.next();

    switch (fruit) {
        case "Mango" -> System.out.println("King of fruits");
        case "Apple" -> System.out.println("A sweet red fruit");
        case "Orange" -> System.out.println("Round fruit");
        case "Grapes" -> System.out.println("Small fruit");
        default -> System.out.println("please enter a valid fruit");
    }
```

Math class

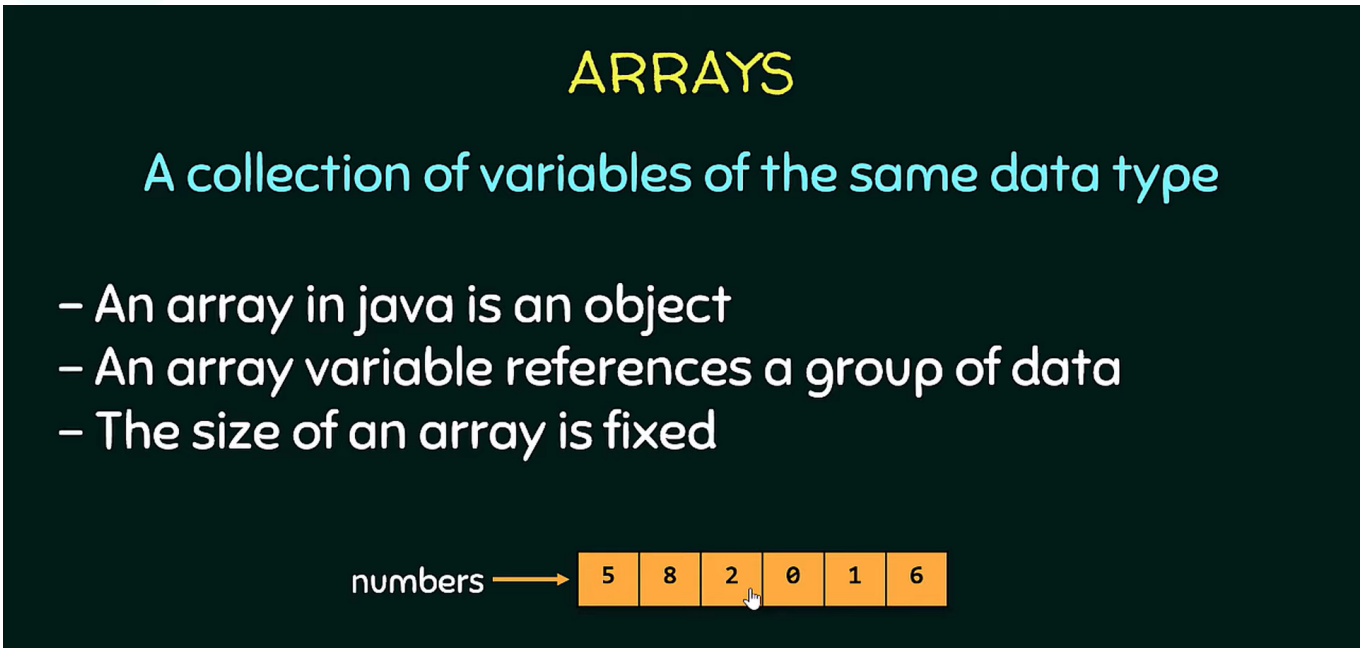
```
public static void main(String[] args) {
    System.out.println(Math.abs(-100));
    System.out.println(Math.max(2,424));
    System.out.println(Math.min(2,223));
    System.out.println((Math.pow(4,4)));
    System.out.println((int)Math.pow(4,4));
    System.out.println(Math.sqrt(25));
    System.out.println((int)Math.sqrt(25));
    System.out.println(Math.PI);
```

```
        int n =  Math.floor(4.06565)//4
}
```

```
100
424
2
256.0
256
5.0
2
3.141592653589793
```

Arrays

#Arrays



PRINTING ARRAYS

```
int[] numbers = {5, 0, 8, 0, 10};

System.out.println(numbers); // ADDRESS

for (int i = 0; i < numbers.length; i++)
    System.out.print(numbers[i] + " ");
```

0 and Null in array

DEFAULT VALUES

When an array is created, its elements are assigned the following default values:

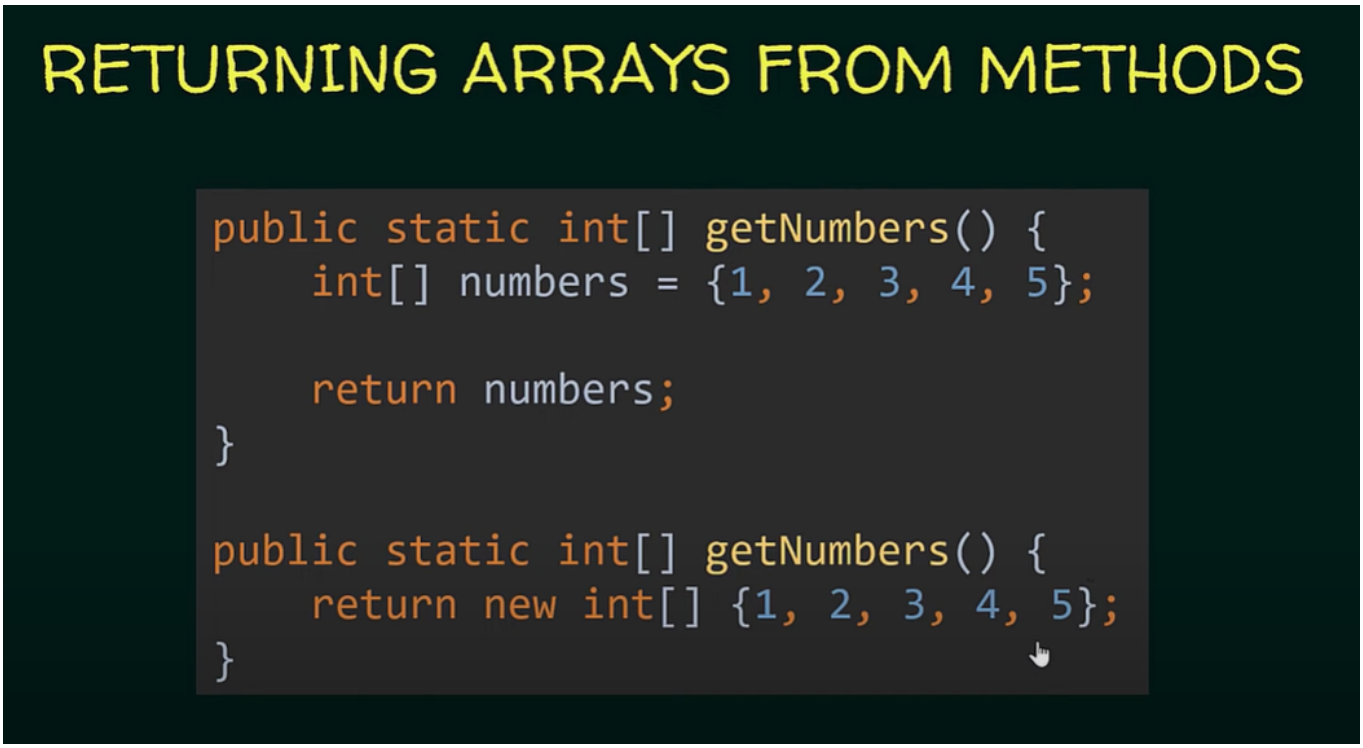
- `0` for the numeric primitive data types
- `\u0000` for char types
- `false` for boolean types
- `null` for reference types

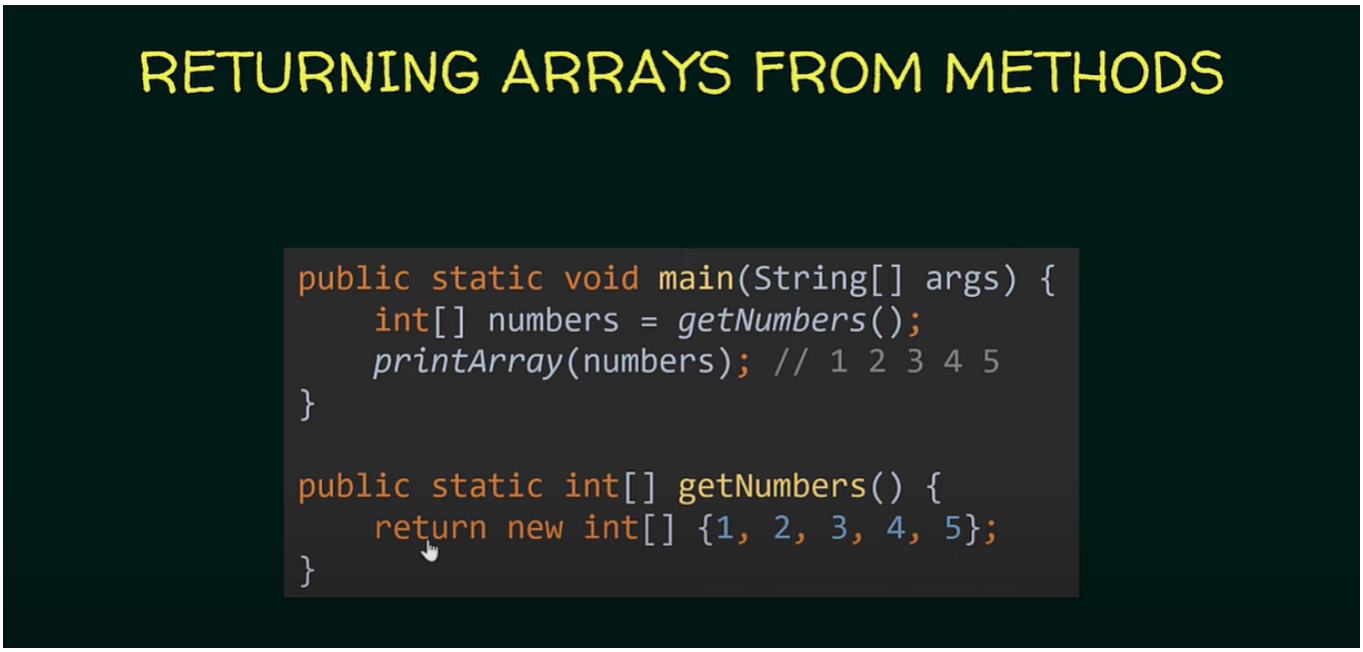
Returning Array in method

NOTE

```
public static int getNumber() {
    int x = 1;

    return x;
}
```

x is destroyed after this method is executed





- Array in getNumber() will not destroy after execution because it always referred by a array numbers.

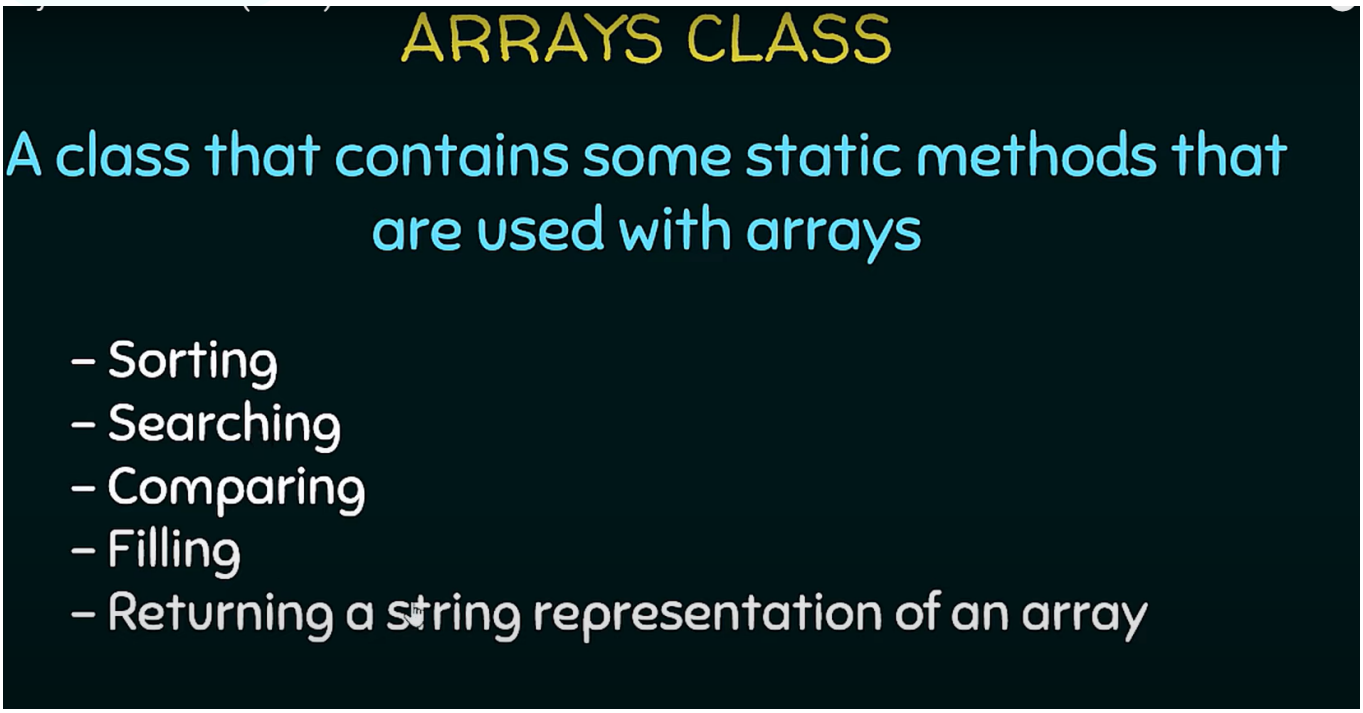
```
public static void main(String[] args) {
    int[] num1= new int[3];
    num1[0]=1;
    System.out.println(Arrays.toString(num1));
    Arrays.fill(num1,-1);
```

```
        System.out.println(Arrays.toString(num1));
        String[] names=new String[4];
        names[3]="Prakii";
        System.out.println(Arrays.toString(names));
        Arrays.fill(names,"Hey");
        System.out.println(Arrays.toString(names));
    }
```

```
[1, 0, 0]
[-1, -1, -1]
[null, null, null, Prakii]
[Hey, Hey, Hey, Hey]
```

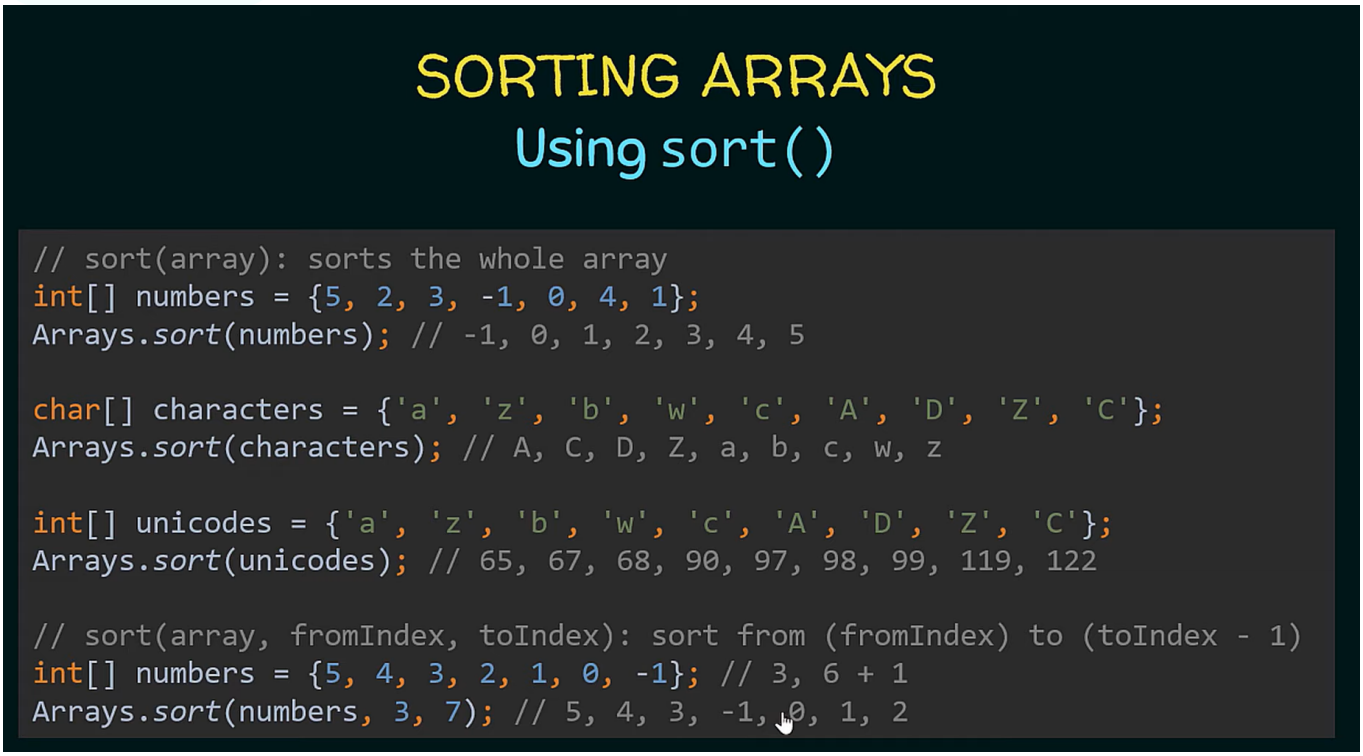
Array class

#Arrayclasses



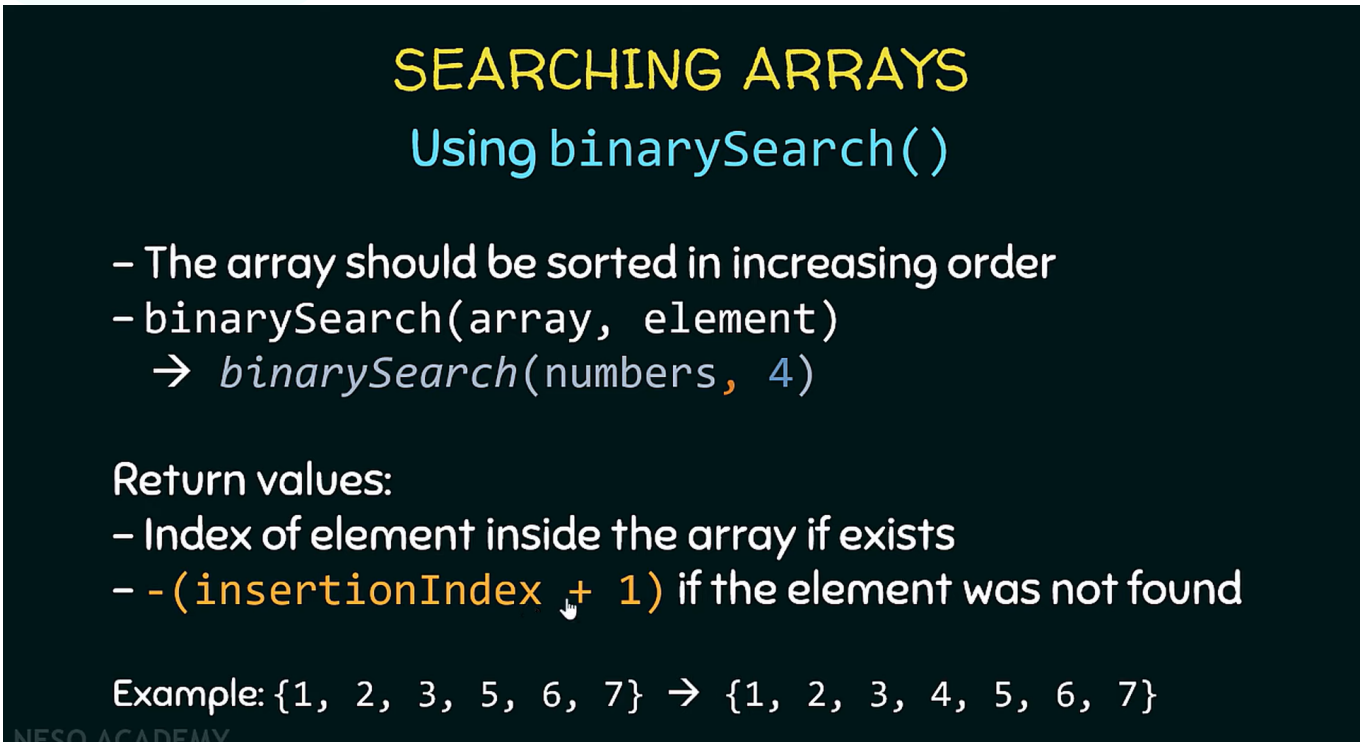
Sorting Array

#SortingArray



Searching Array

#searchingArray



SEARCHING ARRAYS

```
int[] numbers = {5, 4, 3, 2, 1, 0, -1};
Arrays.sort(numbers); // -1, 0, 1, 2, 3, 4, 5

System.out.println( Arrays.binarySearch(numbers, 4) ); // 5
System.out.println( Arrays.binarySearch(numbers, 3) ); // 4
System.out.println( Arrays.binarySearch(numbers, -3) ); // -1
System.out.println( Arrays.binarySearch(numbers, 6) ); // -8

String[] strings = {"a", "b", "c"};
System.out.println(Arrays.binarySearch(strings, "a")); // 0
System.out.println(Arrays.binarySearch(strings, "c")); // 2
System.out.println(Arrays.binarySearch(strings, "A")); // -1
System.out.println(Arrays.binarySearch(strings, "d")); // -4
```

Comparing Arrays

#ComparingArrays

COMPARING ARRAYS

Using equals()

```
int[] numbers1 = {5, 4, 3, 2, 1, 0, -1};
int[] numbers2 = {5, 4, 3, 2, 1, 0, -1};
int[] numbers3 = {1, 2, 3, 7, 7, 8, 1};

System.out.println(numbers1 == numbers2); // false
System.out.println(Arrays.equals(numbers1, numbers2)); // true
System.out.println(Arrays.equals(numbers1, numbers3)); // false
```

COMPARING ARRAYS

```
String[] strings1 = {"a", "b", "c"};
String[] strings2 = {"a", "b", "c"};

System.out.println(strings1 == strings2); // false
System.out.println(Arrays.equals(strings1, strings2)); // true

Point[] points1 = {new Point(1, 2), new Point(3, 4)};
Point[] points2 = {new Point(1, 2), new Point(3, 4)};
Point[] points3 = {new Point(0, 0), new Point(3, 4)};

System.out.println(points1 == points2); // false
System.out.println(Arrays.equals(points1, points2)); // true
System.out.println(Arrays.equals(points1, points3)); // false
```

EQUALS

```
String str1 = new String("hello");
String str2 = new String("hello");

System.out.println( str1 == str2 ); // false
System.out.println( str1.equals(str2) ); // true

Point point1 = new Point(1, 2);
Point point2 = new Point(1, 2);

System.out.println( point1 == point2 ); // false
System.out.println( point1.equals(point2) ); // true
```

Filling Arrays

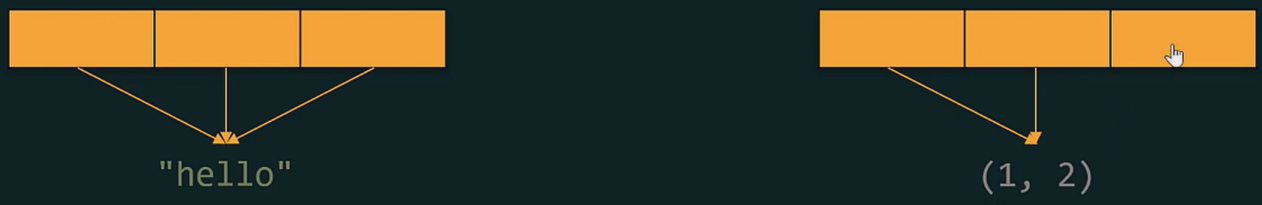
#FillingArray

FILLING ARRAYS
Using fill()

```
// fill(array, value): fill whole array
int[] numbers1 = new int[8]; // {0, 0, 0, 0, 0, 0, 0, 0}
Arrays.fill(numbers1, 3); // {3, 3, 3, 3, 3, 3, 3, 3}

// fill(array, fromIndex, toIndex, value)
int[] numbers2 = new int[8]; // {0, 0, 0, 0, 0, 0, 0, 0}
Arrays.fill(numbers2, 3, 7, 5); // {0, 0, 0, 5, 5, 5, 5, 0}
```

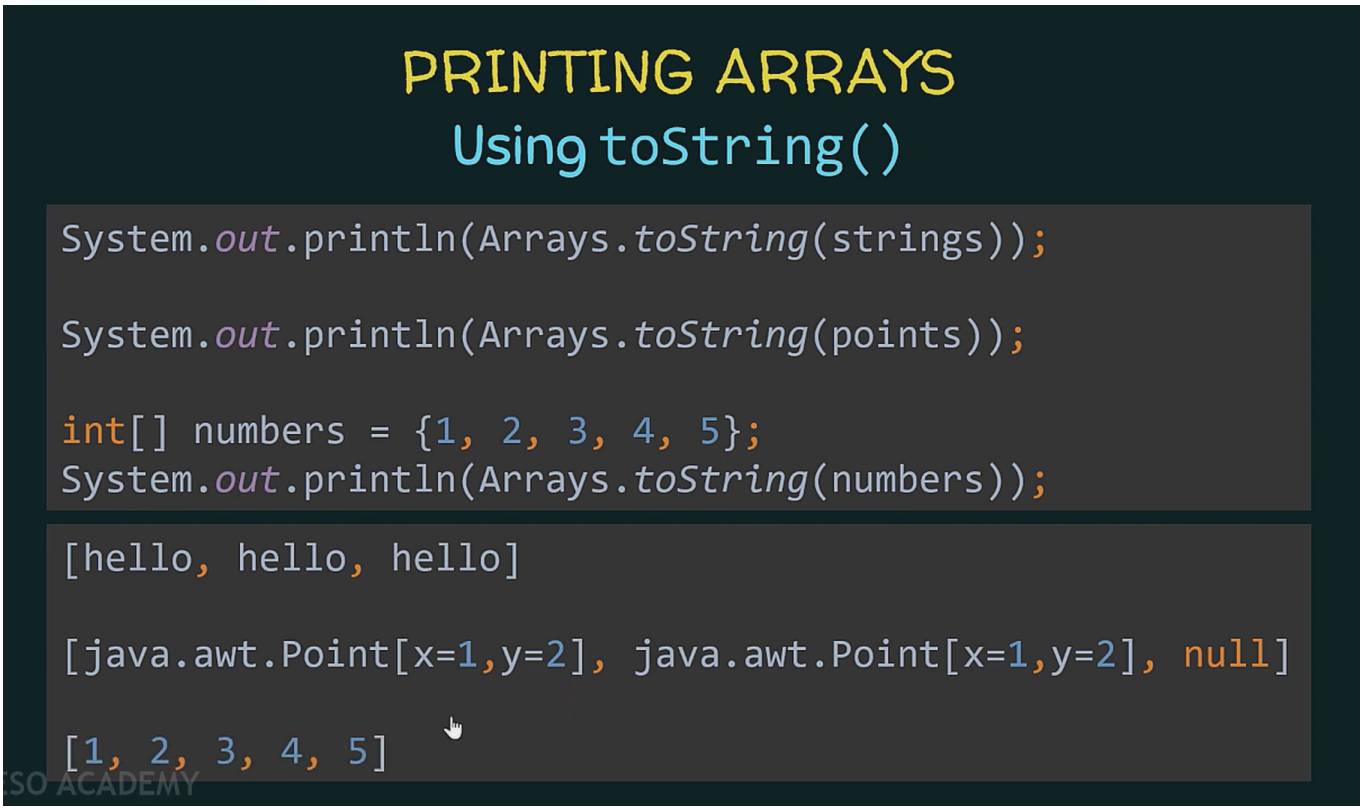
FILLING ARRAYS

```
String[] strings = new String[3]; // {null, null, null}
Arrays.fill(strings, "hello"); // {hello, hello, hello}

Point[] points = new Point[3]; // {null, null, null}
Arrays.fill(points, 0, 2, new Point(1, 2)); // {(1, 2), (1, 2), null}
```



Printing Arrays

#PrintingArray



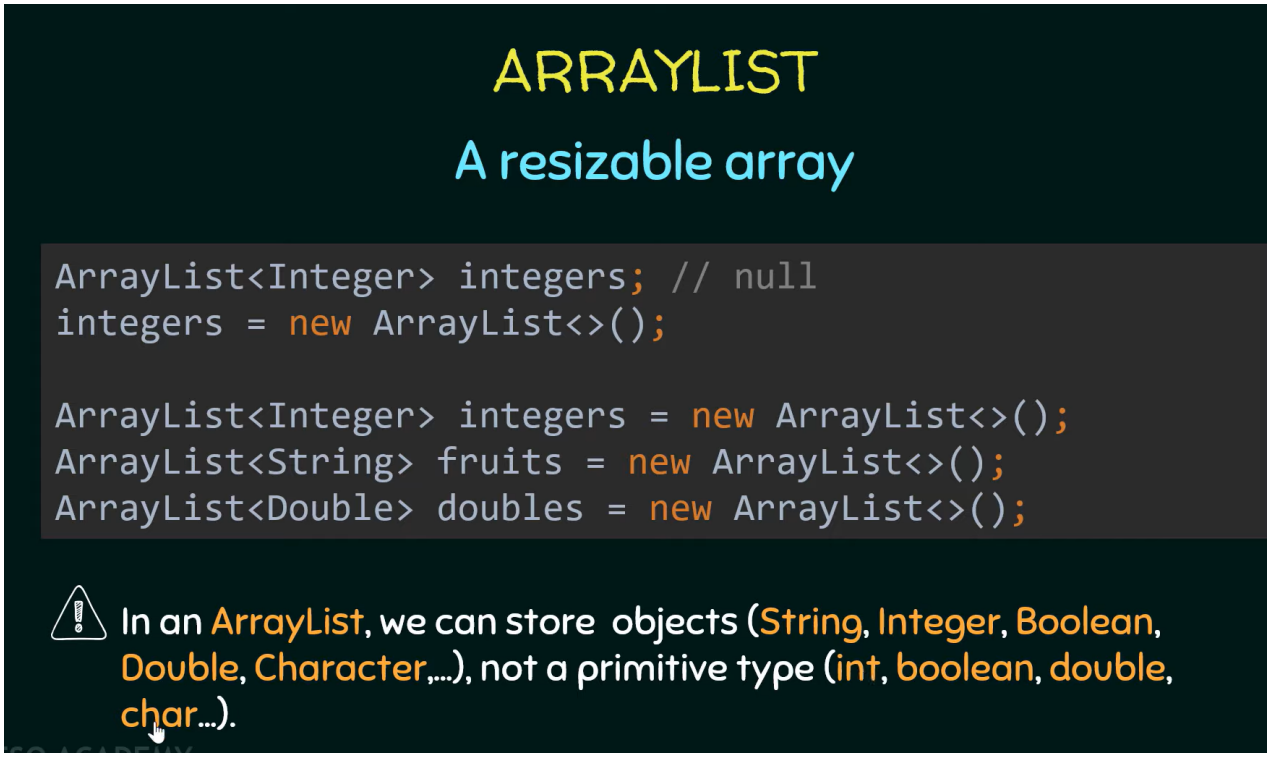
Array using For Loop

```java
public class Practice
{
    public static void main(String[] args) {
        String[] Name ={"prakii","ish","Lakshmi"};
        for (String s : Name) {
            System.out.println(s);
        }
    }
}
```

```
prakii
ish
Lakshmi
```

Array List

- In Array size is fixed. But in "Array list " the size is resizable and dynamic.
- Here Array List are objects.



ACCESS AN ITEM

– Using the get() method.

```
System.out.println(fruits.get(0)); // Apple
System.out.println(fruits.get(1)); // Banana
System.out.println(fruits.get(2)); // Strawberry
```

CHANGE AN ITEM

– Using the set() method.

```
fruits.set(2, "Orange"); // change Strawberry to Orange
System.out.println(fruits); // [Apple, Banana, Orange]
```

REMOVE AN ITEM

– To remove an element, use the remove() method.

→ Removing by index :

```
fruits.remove(1); //remove the element at index 1
System.out.println(fruits); //[Apple, Orange]
```

→ Removing by value :

```
fruits.remove("Banana"); // remove "Banana"
System.out.println(fruits); // [Apple, Orange]
```

– To remove all elements, use the clear() method.

```
fruits.clear(); // remove all elements
System.out.println(fruits); // []
```

Sorting An Arraylist

SORT AN ARRAYLIST

– Use the sort() method of the Collections class for sorting lists alphabetically or numerically.

```
System.out.println(fruits); // [Apple, Strawberry, Banana]
Collections.sort(fruits);
System.out.println(fruits); // [Apple, Banana, Strawberry]
```

SORT AN ARRAYLIST

```
ArrayList<Integer> numbers = new ArrayList<>();

numbers.add(1);
numbers.add(5);
numbers.add(7);
numbers.add(0);
numbers.add(-1);

System.out.println(numbers); // [1, 5, 7, 0, -1]
Collections.sort(numbers);
System.out.println(numbers); // [-1, 0, 1, 5, 7]
```

For Each Loop(Array list)

- used for iteration in Array/Array list

syntax of For Each

FOR EACH

```
for ( TYPE VAR_NAME : ArrayList/Array ) {
    ...
}
```

- In each iteration, the variable VAR_NAME will hold the value of an element inside the ArrayList/Array, starting from the first element.

- There is no index

- Safe (Boundaries)

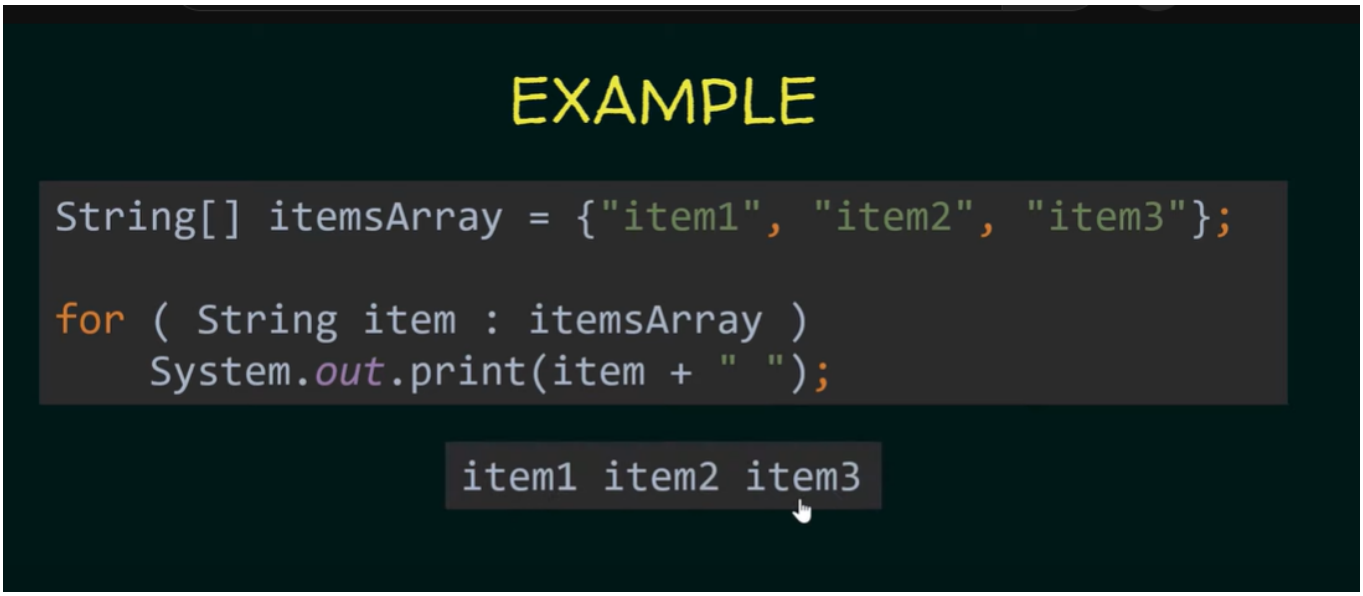
In Array list

EXAMPLE

```
ArrayList<String> itemsArrayList = new ArrayList<>();
itemsArrayList.add("item1");
itemsArrayList.add("item2");
itemsArrayList.add("item3");

for ( String item : itemsArrayList )
    System.out.print(item + " ");
```

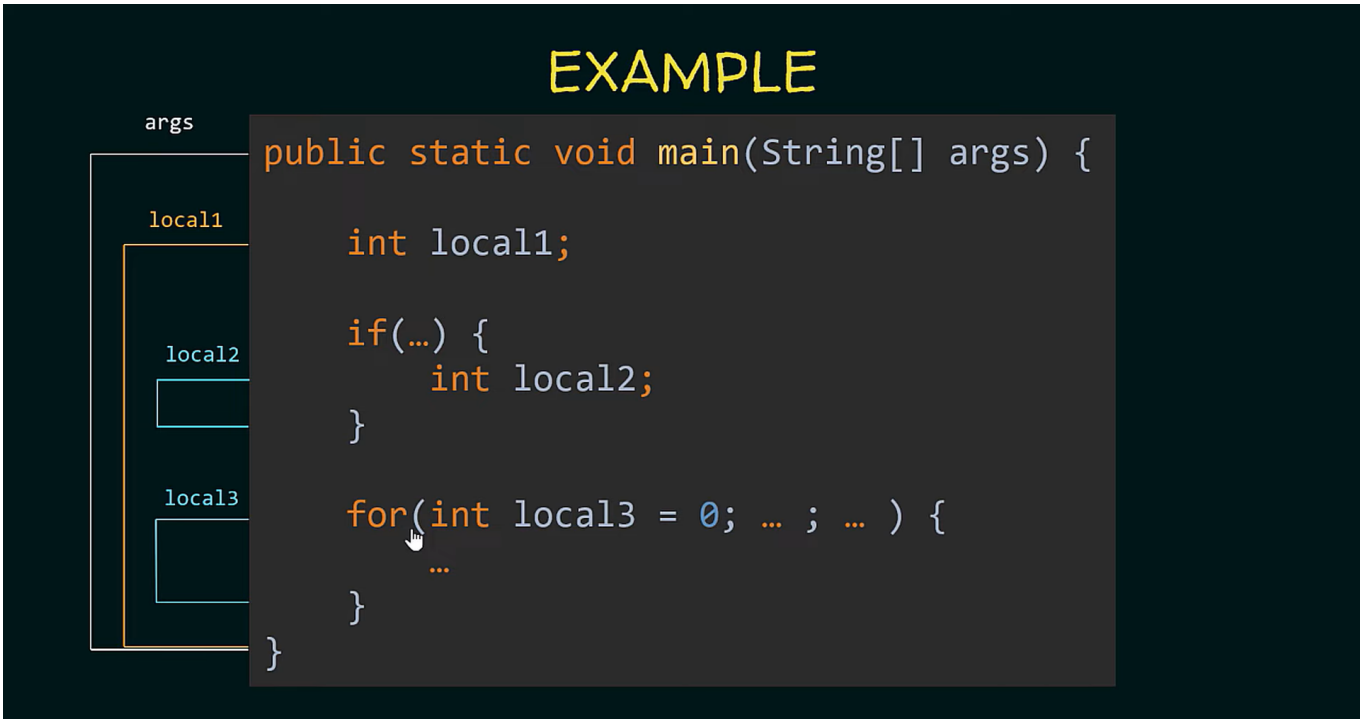
```
item1 item2 item3
```

In Array



Scope And Local Variables!

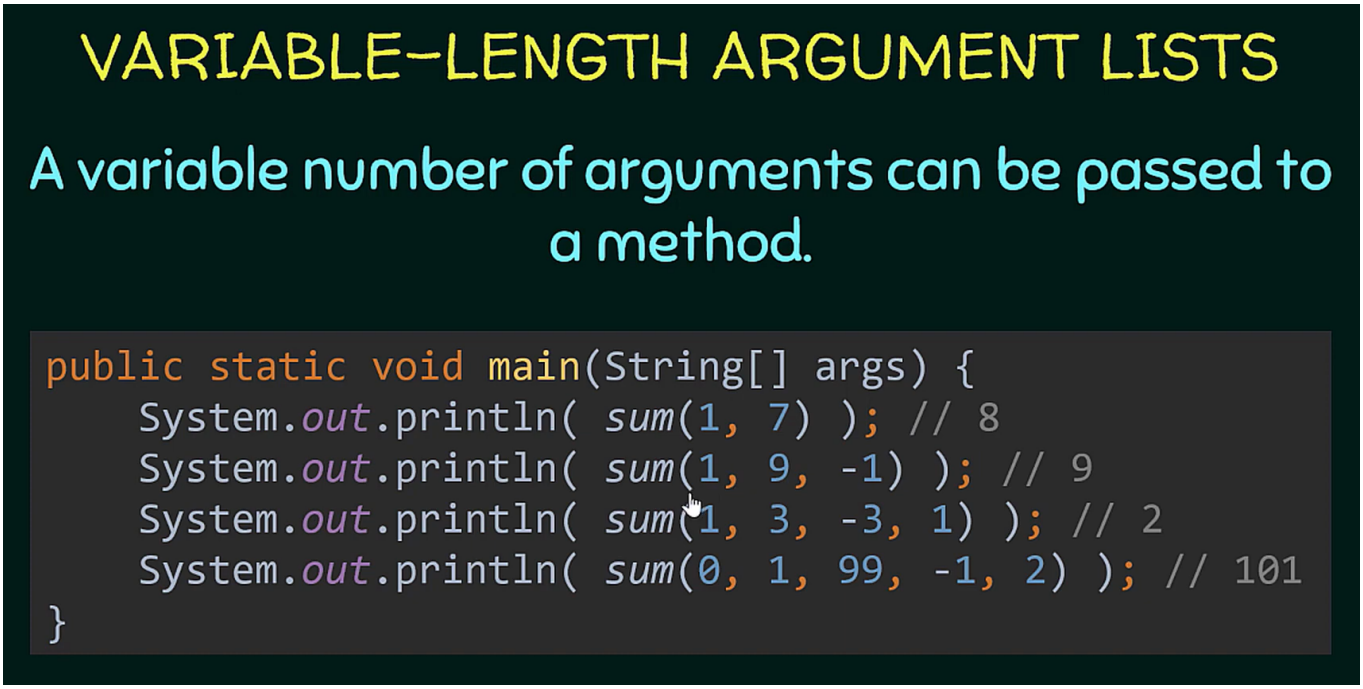
Screenshot 2024-02-17 213246.png

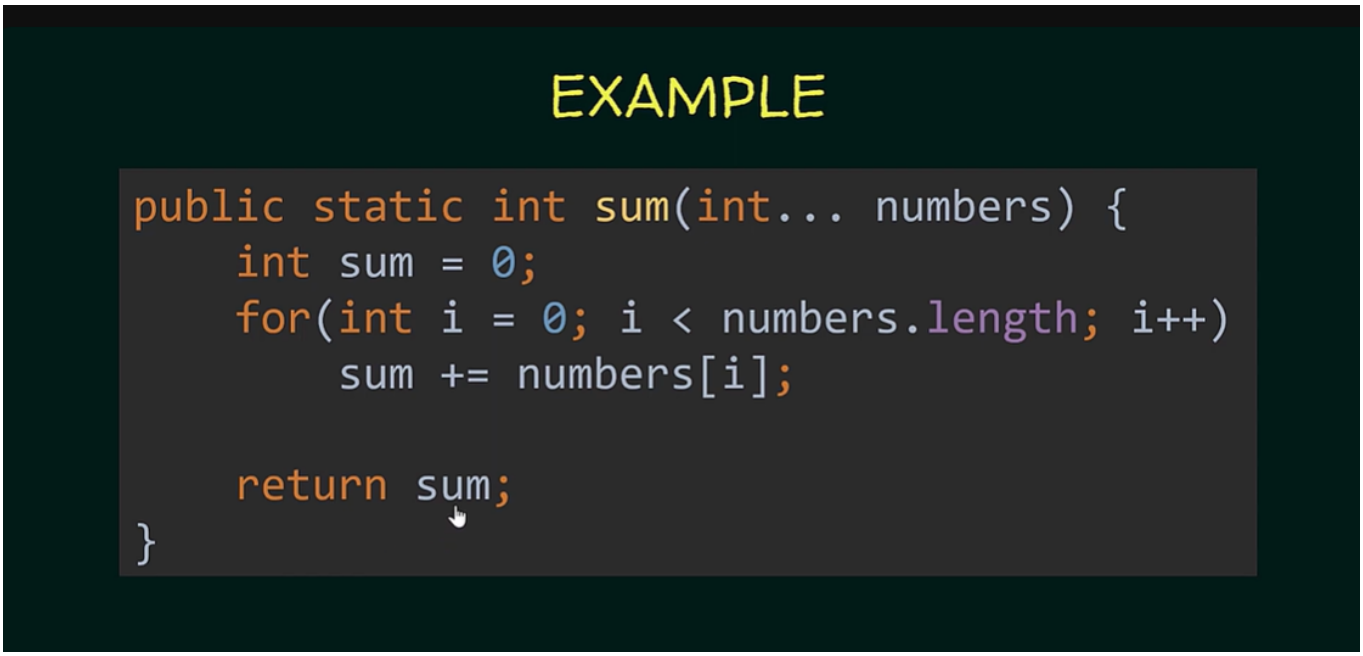


Method

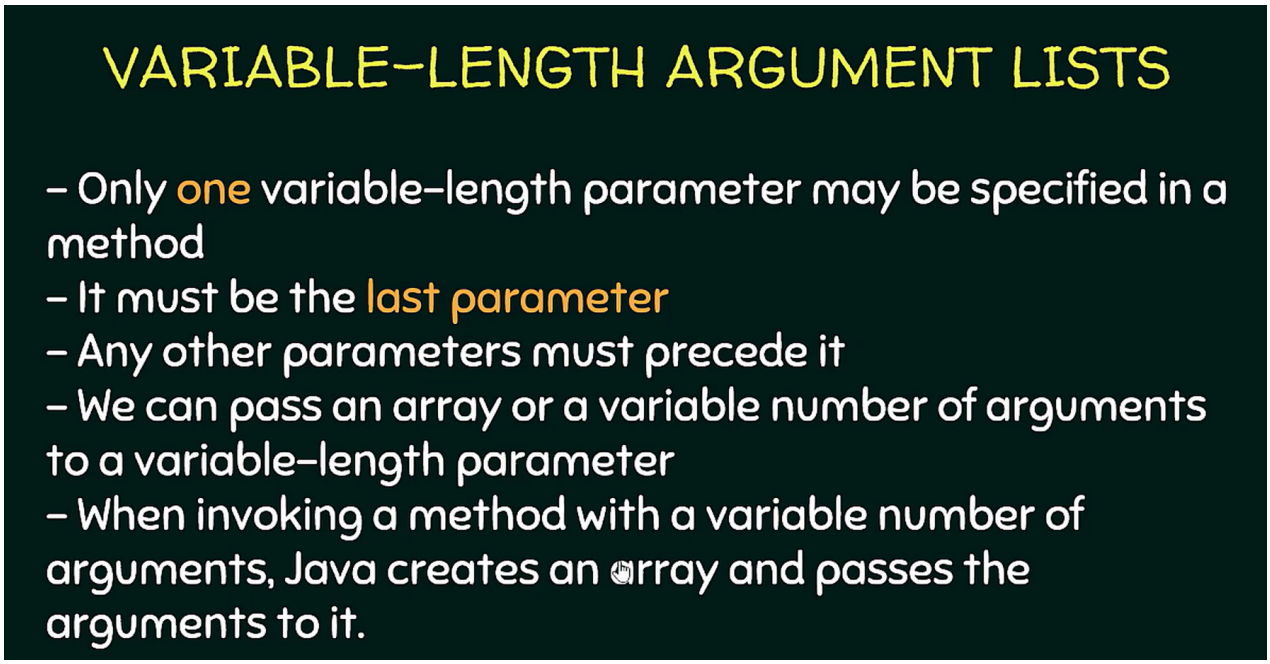
Variable-Length argument List

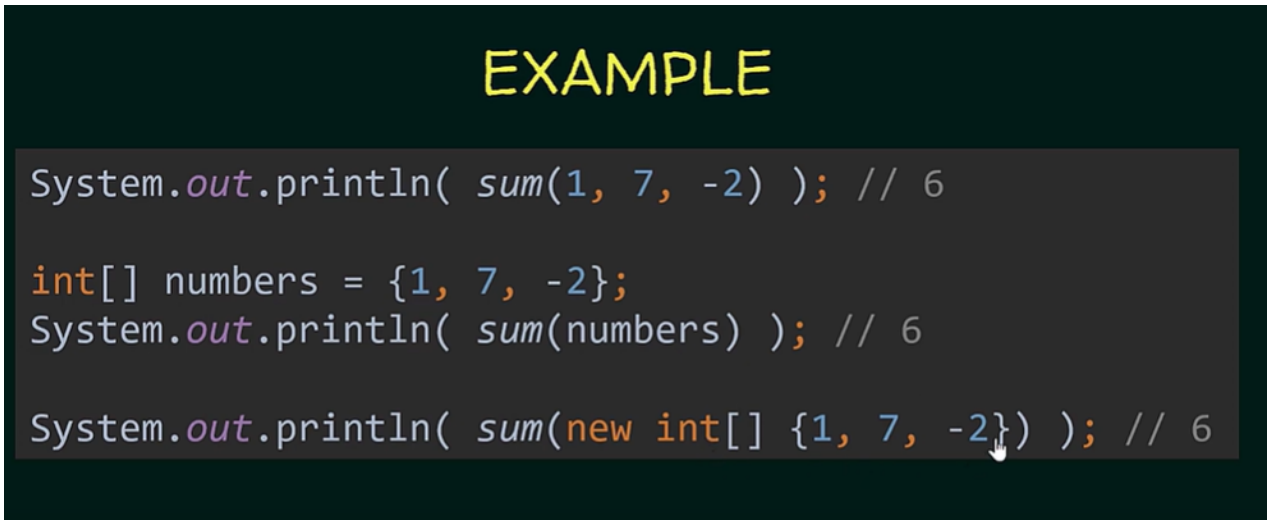
- ***it means variable number of arguments can be passed through a method

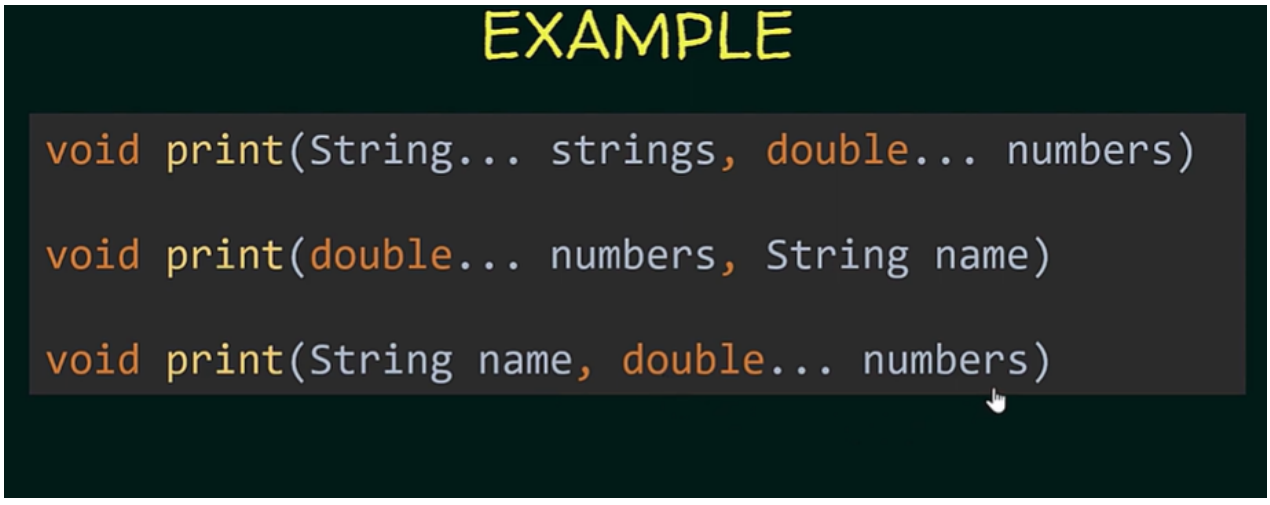




- In Above content that "Int . . ." means ellipse.it want to variable number of argument that continue calling by the parameter "numbers". here the parameter is act as an Array.



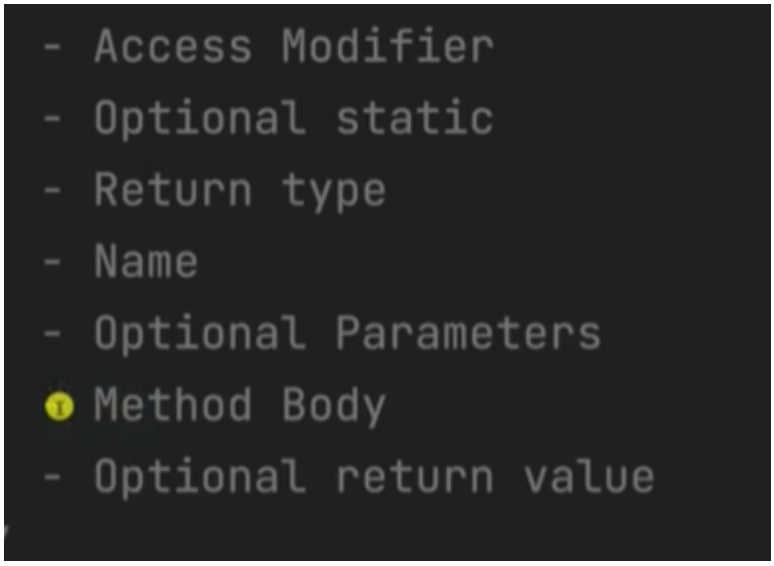




- In Above content 1st line have error because we have passes only one variable length parameter("Double . . .) is to be present.

- in 2nd also error because we need to pass variable length parameter as last parameter.
- Last line is correct ,because all the parameters are pass before the variable length parameter(it must be last).
- "void print(String name ,double . . . numbers)".

Format

- 

- Access modifier -> Public ,Private ,Protected.
- optional Static ->static
- return type -> double , int, void.
- Name -> method Name.
- optional parameter -> string[],double.
- method -> content in body within{};
- optional return value -> return.

Method Overloading

METHOD OVERLOADING

Writing the same method with different parameters

```java
public static int sum(int x, int y) {
    return x + y;
}

public static int sum(int x, int y, int z) {
    return x + y + z;
}
```

NOTE

Overloaded methods can have different return types

```java
public static void main(String[] args) {
    sayHi(); // Hi
    System.out.println( sayHi("NesoAcademy") );  // Hi NesoAcademy
}

public static void sayHi() {
    System.out.println("Hi");
}

public static String sayHi(String name) {
    return "Hi " + name;
}
```

NOTE

Overloaded methods must have different parameters

- two methods with same name leads to error.

- To avoid this overload we must have different parameter and different return type.
- same name but each have Different arguments for each method

Method creation

```
public static void main(String[] args) {
    System.out.print("Enter your name and Age :");
    System.out.println("Hello! "+Name()+" Your are "+age()+"
Old.");
}

public static int age() {
return new Scanner(System.in).nextInt();
}
public static String Name() {
    return new Scanner(System.in).nextLine();
```

```
Enter your name and Age :Prakii
19
Hello! Prakii Your are 19 Old.
```

Example 1

```
public class Main {
    public static void main(String[] args) {
        double percetage=markpercentage(75,43,90,300);
        System.out.println(percetage);
    }
    private static double markpercentage(int sub1,int
sub2,int sub3,int totalmark)
    {
```

```
        double markscored = sub1 + sub2 + sub3;
        double percentage = (markscored/totalmark)*100;
        return percentage;
    }

    }
```

```
69.33333333333334
```

EXAMPLE 2

```
public static void main(String[] args) {
    char[] Alphapets = {'A', 'A', 'B', 'C', 'D', 'D', 'D'};
    int count = Alpha(Alphapets, 'D');
    System.out.println(count);
}
public static int Alpha(char[] Alphapets, char searchletter)
{
    int count = 0;
    for (char alphapet : Alphapets) {
        if (alphapet == searchletter) {
            count++;
        }
    }
    return count;
}
```

Class And Object Creation

Class Structure

```
public static void main(String args) {
}
static class lens{
    String company;
    String Focus;
    Double Price;
    String Colour;
    boolean isprime;
    lens(String company,String Focus ,Double Price,String
colour,boolean isprime){
        this.company=company;
        this.Focus=Focus;
        this.Price=Price;
        this.Colour=Colour;
        this.isprime=isprime;
    }

}
```

Object Creation

```
public static void main(String[] args) {
    lens L1=new lens("Sony","30MM",3234.99,"RED",true);
    lens L2=new lens("Canon","40MM",40.00,"BLACK",true);
    lens L3=new lens("Gopro","10MM",3500.00,"BLACK",false);

}
```

final Code

```
public static void main(String[] args) {
    lens L1=new lens("Sony","30MM",3234.99,"RED",true);
    lens L2=new lens("Canon","40MM",40.00,"BLACK",true);
```

```
    lens L3=new lens("Gopro","10MM",3500.00,"BLACK",false);
    System.out.println("Lens 1->"+L1.company+" FocalLength->
"+L1.Focus+",  Price-> "+L1.Price+",  Colour->
"+L1.Colour+",  isPrime-> "+L1.isprime);
    System.out.println("Lens 2->"+L2.company+" FocalLength->
"+L2.Focus+",  Price-> "+L2.Price+",  Colour->
"+L2.Colour+",  isPrime-> "+L2.isprime);
    System.out.println("Lens 3->"+L3.company+" FocalLength->
"+L3.Focus+",  Price-> "+L3.Price+",  Colour->
"+L3.Colour+",  isPrime-> "+L3.isprime);

}
static class lens{
    String company;
    String Focus;
    Double Price;
    String Colour;
    boolean isprime;
    lens(String company,String Focus ,Double Price,String
colour,boolean isprime){
        this.company=company;
        this.Focus=Focus;
        this.Price=Price;
        this.Colour=Colour;
        this.isprime=isprime;
    }
}
```

```
Lens 1->Sony FocalLength-> 30MM,  Price-> 3234.99,  Colour-> null,  isPrime-> true
Lens 2->Canon FocalLength-> 40MM,  Price-> 40.0,  Colour-> null,  isPrime-> true
Lens 3->Gopro FocalLength-> 10MM,  Price-> 3500.0,  Colour-> null,  isPrime-> false
```

Example(Passport Problem)

```
public static void main(String[] args) {
  Passport ukPassport = new
Passport("323","UK",LocalDate.of(2025,02,24));
  Passport usPassport = new
Passport("123","USA",LocalDate.of(2026,9,26));
  Passport indiaPassport = new
Passport("123","INDIA",LocalDate.of(2029,12,23));
    System.out.println("UK passport");
    System.out.println(ukPassport.Number);
    System.out.println(ukPassport.Country);
    System.out.println(ukPassport.Expirydate);
    System.out.println();
    System.out.println("USA Passport");
    System.out.println(usPassport.Number);
    System.out.println(usPassport.Country);
    System.out.println(usPassport.Expirydate);
    System.out.println();
    System.out.println("INDIA passport");
    System.out.println(indiaPassport.Number);
    System.out.println(indiaPassport.Country);
    System.out.println(indiaPassport.Expirydate);
}
static class Passport{
    String Number;
    String Country;
    LocalDate Expirydate;
    Passport(String Number ,String Country,LocalDate
Expirydate){
      this.Number=Number;
      this.Country=Country;
      this.Expirydate=Expirydate;
    }

}
```

```
UK passport
323
UK
2025-02-24

USA Passport
123
USA
2026-09-26

INDIA passport
123
INDIA
2029-12-23
```