JAVA

introduction

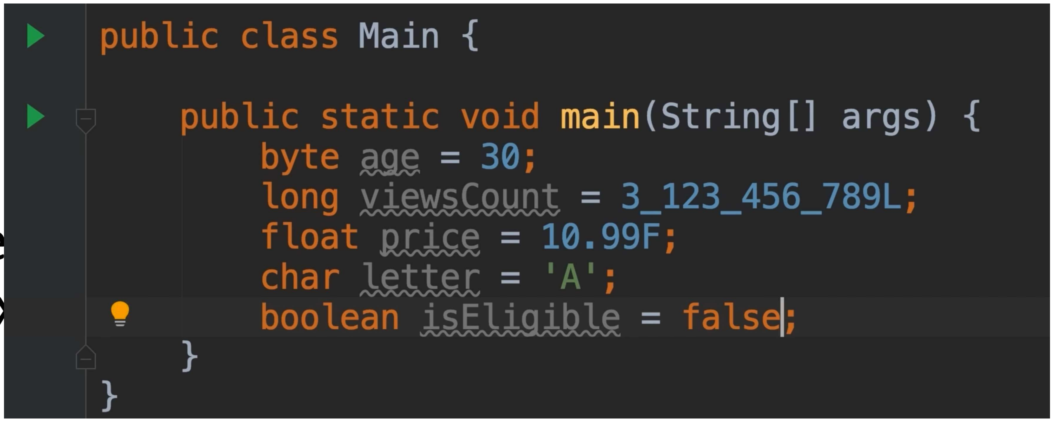
- Types
  - Variables and Constants
  - Primitive and Reference types
  - Casting
  - Numbers, Strings and Arrays
  - Read Input

Primitive types
simple values

Primitive Types

Type	Bytes	Range	
byte	1	[-128, 127]	
short	2	[-32K, 32K]	
int	4	[-2B, 2B]	
long	8		
float	4		
double	8		
char	2	A, B, C, …	
boolean	1	true / false	

- Why we use L after the long number? By default java assumes 3123456789 to be an integer so we have to add the suffix L to represent the number as a long
- When we use the decimal point java assumes it is a double, incase of a float we have to use the suffi F to represent a float number.



```java
public class Main {

    public static void main(String[] args) {
        byte age = 30;
        long viewsCount = 3_123_456_789L;
        float price = 10.99F;
        char letter = 'A';
        boolean isEligible = false;
    }
}
```

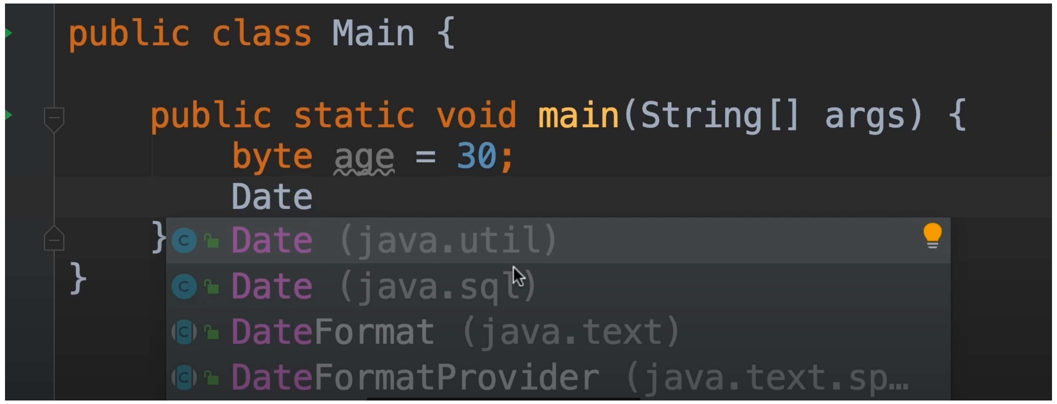
Reference types complex objects

- In other words, a variable of class type is called **reference data type**. It contains the address (or reference) of dynamically created objects.

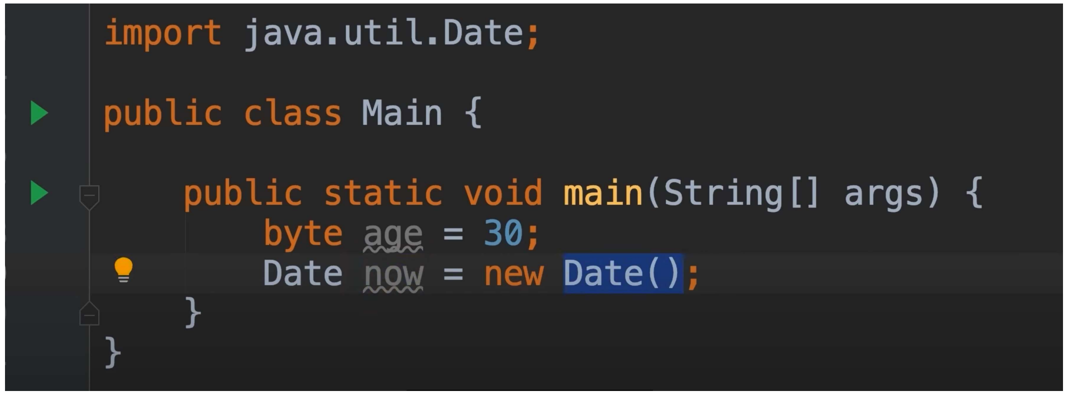
- Example

```
Demo d1 = new Demo("Atlanta");
//creating a reference of Demo class
```

- First we declare a primitive type
  - (int age=30;)
- Then we Declare and Initialize a reference type
  - Type Date and see what happens
    - It suggests classes with the name Date in different packages
    - To use a class from a different package you have to import it
    - We use new to allocate memory and repeat the name of our class
    - An object is an instance of a class

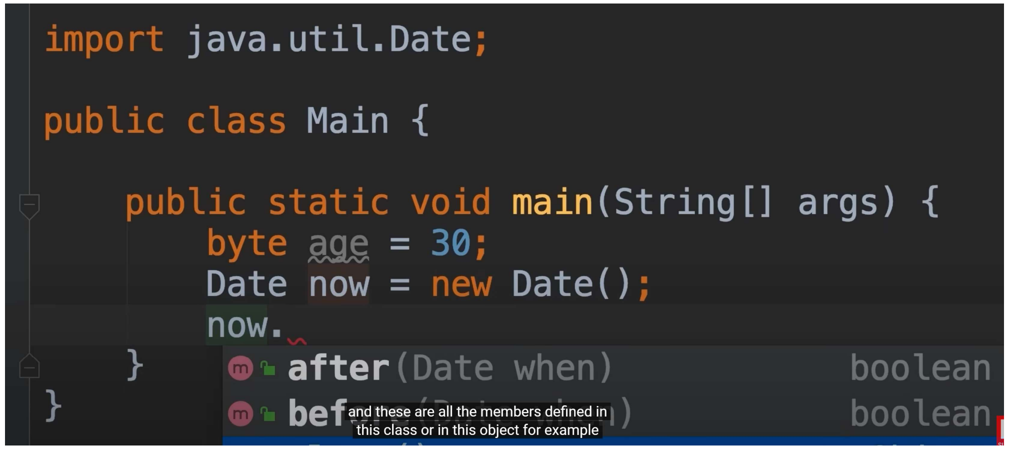


```
public class Main {

    public static void main(String[] args) {
        byte age = 30;
        Date
    }  Date (java.util)
}      Date (java.sql)
       DateFormat (java.text)
       DateFormatProvider (java.text.sp…
```



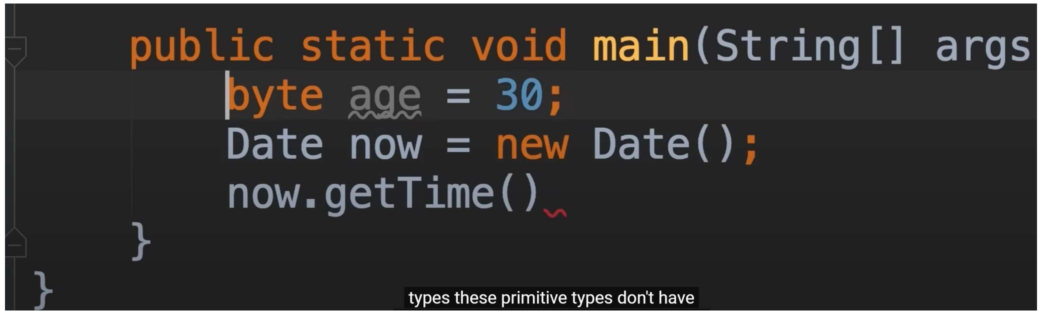
```
import java.util.Date;

public class Main {

    public static void main(String[] args) {
        byte age = 30;
        Date now = new Date();
    }
}
```

- A class has members that can be accessed by dot operator
- Can we use dot operator in primitive type (age.???)



```
import java.util.Date;

public class Main {

    public static void main(String[] args) {
        byte age = 30;
        Date now = new Date();
        now.
    }     after(Date when)              boolean
}     bef                    )          boolean
```

and these are all the members defined in this class or in this object for example

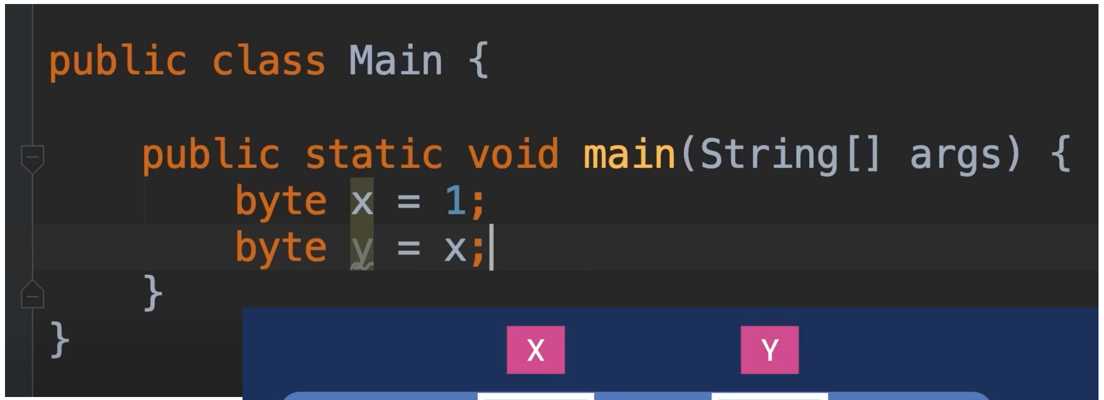


```
    public static void main(String[] args
        byte age = 30;
        Date now = new Date();
        now.getTime()
    }
}
```

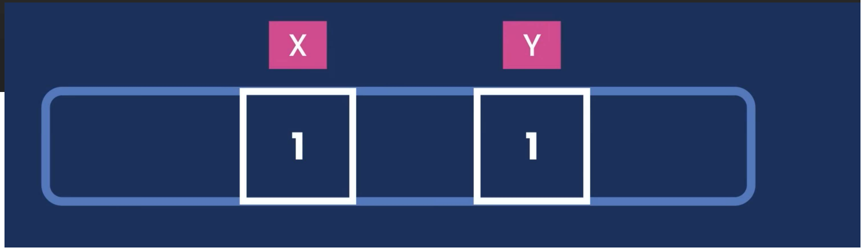
types these primitive types don't have

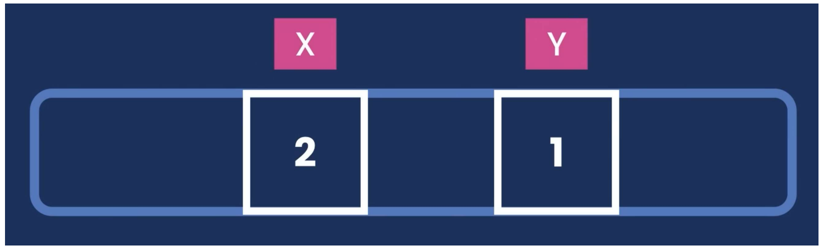
Differences

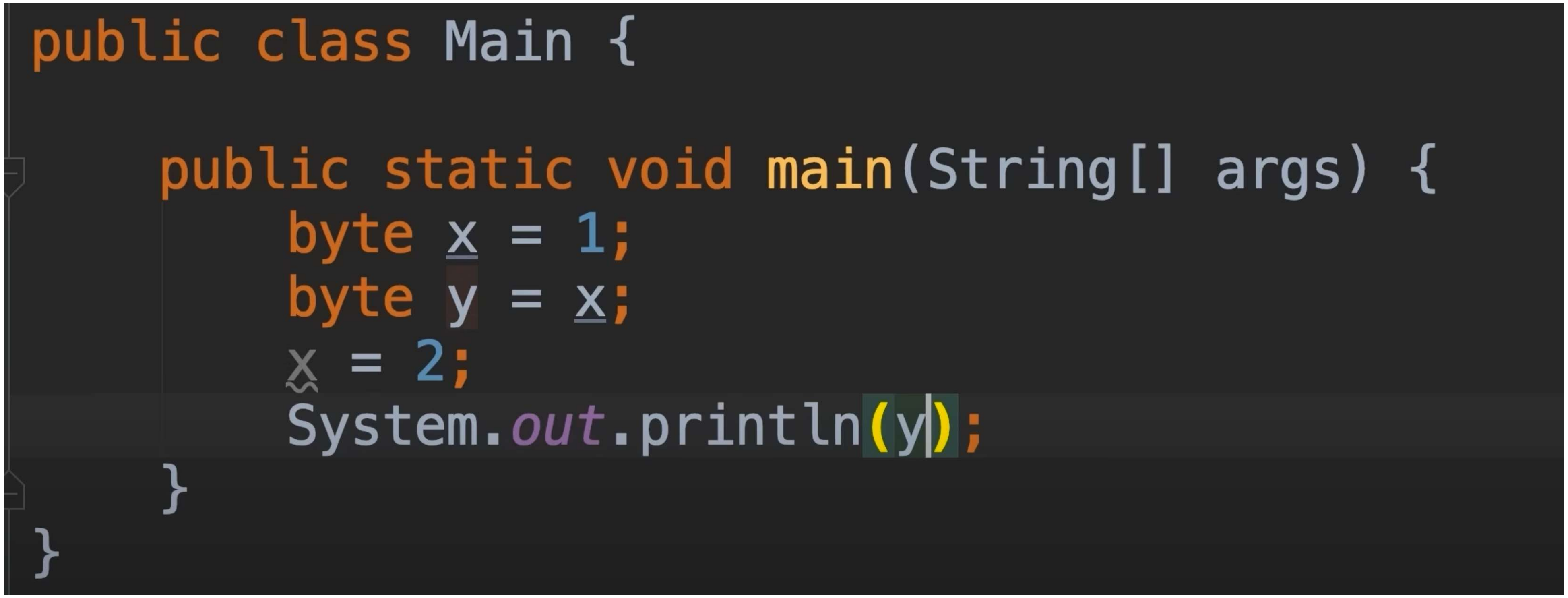
- We have to different variables x and y, in different memory locations, they are completely independent
- If we change the value of x, y won't be affected



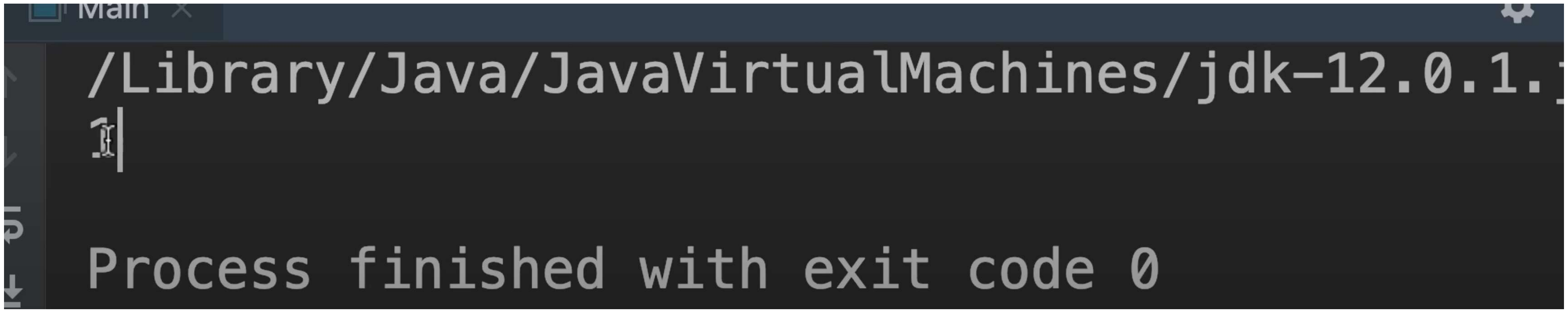
```
public class Main {

    public static void main(String[] args) {
        byte x = 1;
        byte y = x;
    }
}
```







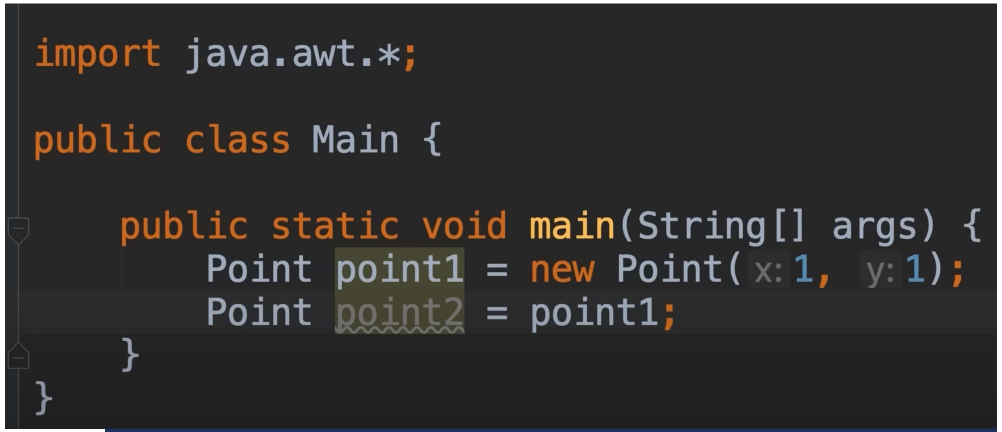
```java
public class Main {

    public static void main(String[] args) {
        byte x = 1;
        byte y = x;
        x = 2;
        System.out.println(y);
    }
}
```



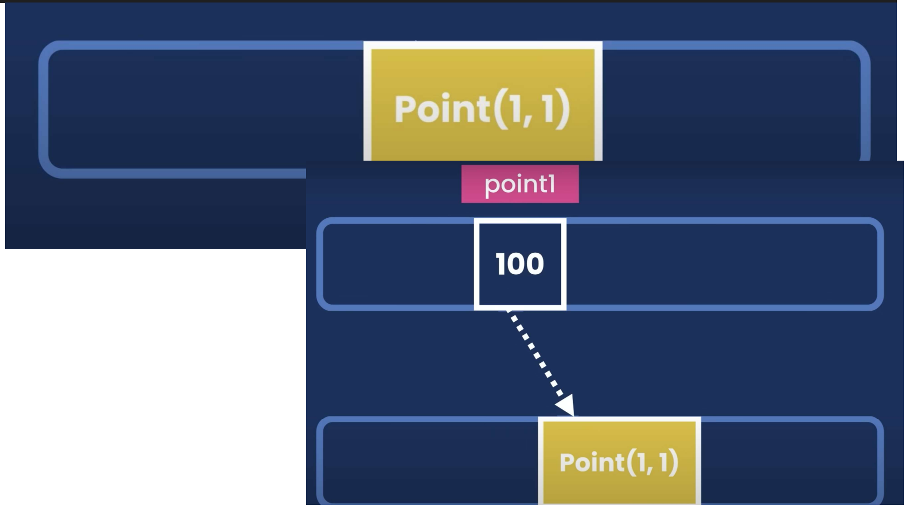
```
/Library/Java/JavaVirtualMachines/jdk-12.0.1.
1

Process finished with exit code 0
```

Memory allocation in reference types

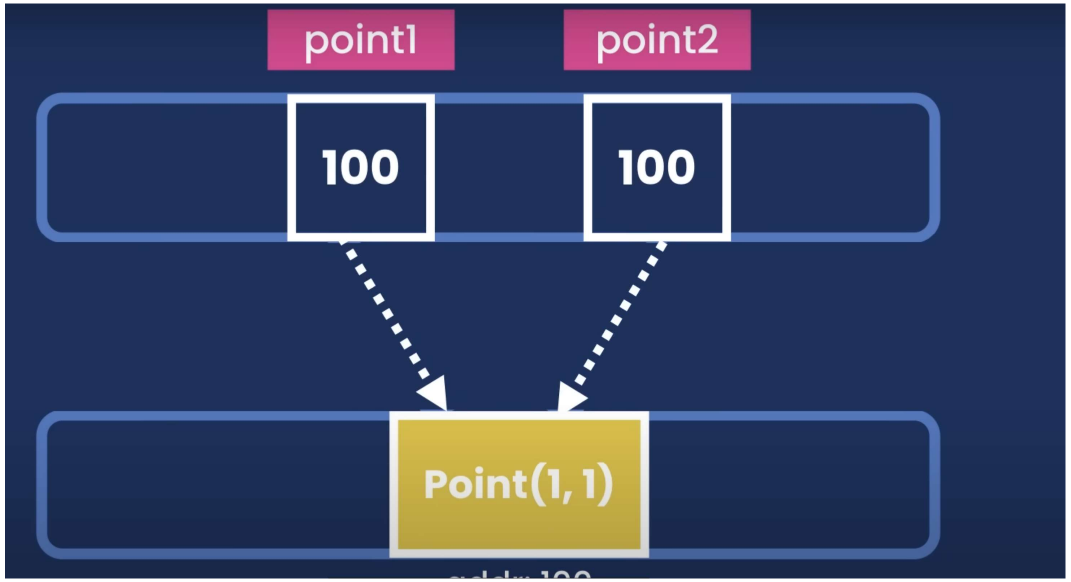
- It allocates memory for Point(1,1), assume the place is 100
  - Then it allocates a separate part of the memory and attach this label to that memory location point1 holds this
  - point2=point1, but point1 is the address of Point(1,1) not the actual value



```
import java.awt.*;

public class Main {

    public static void main(String[] args) {
        Point point1 = new Point( x: 1,  y: 1);
        Point point2 = point1;
    }
}
```

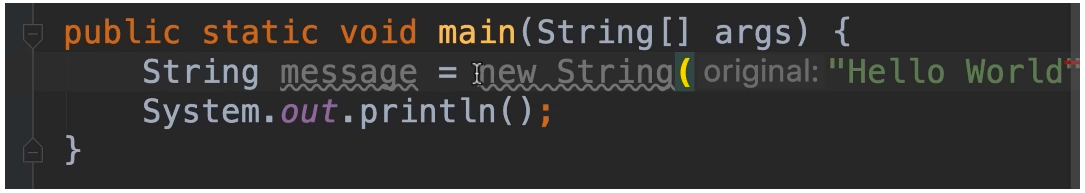


- If you update this Point object through either of these variables the changes will be visible to the other variable

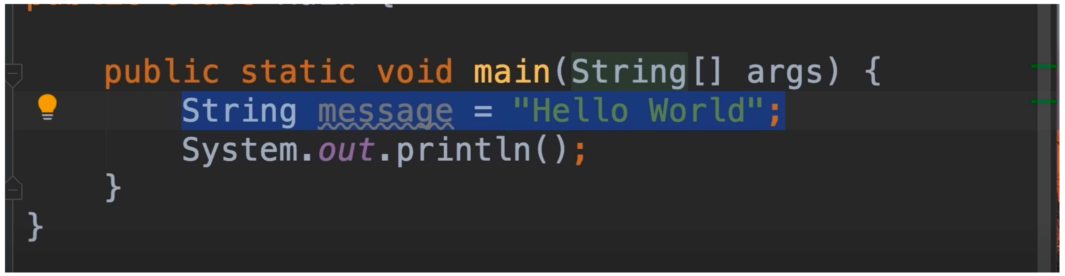


String
reference type

- Although String is a reference type, we use a short way to create them by using the notation of primitive type
- We can combine strings by using the "+" operator
- We can use the dot operator
- In Java strings are immutable, we cannot change them, so any method that modifies a string will always return a new string object.



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



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

Escape sequences

- \t -> It gives a tab between two words.
- The escape sequence \b is a backspace character
- // This \n escape sequence is for a new line.
- This \" escape sequence is for printing a double quotation mark on the text string

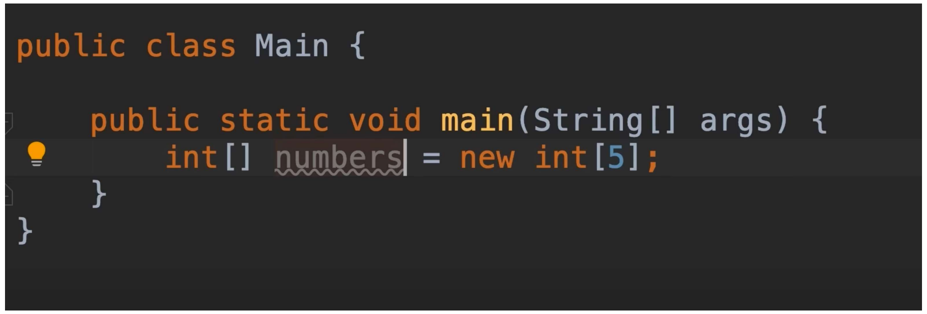
```
System.out.println("Good Morning \"Geeks!\" How are you all?  ");
```

```
Good Morning "Geeks!" How are you
all?
```

Arrays
reference types

Arrays

- Are used to store a list of items
- We specify the type of the items
  - Example for a list of integers we will write int[ ] arrayname = ???
  - Arrays are reference types so we use **new int [size_of_array]**

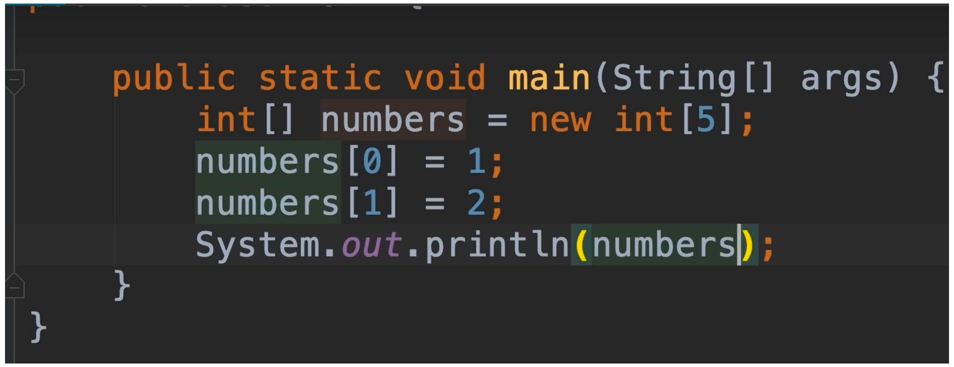


```
public class Main {

    public static void main(String[] args) {
        int[] numbers = new int[5];
    }
}
```

- Using index we can access individual items in the array
  - numbers[0]=2;
  - numbers[1]=3;
- In java indexing starts at 0

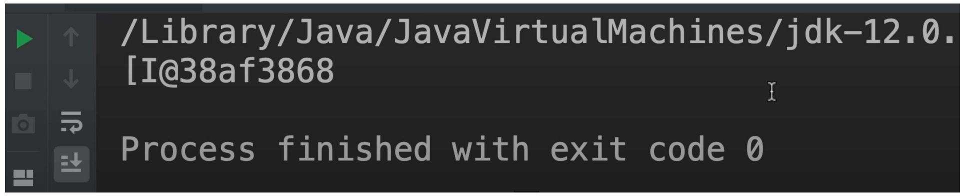
How to print an array

- Assume the following code



```
    public static void main(String[] args) {
        int[] numbers = new int[5];
        numbers[0] = 1;
        numbers[1] = 2;
        System.out.println(numbers);
    }
}
```

- We get a weird string

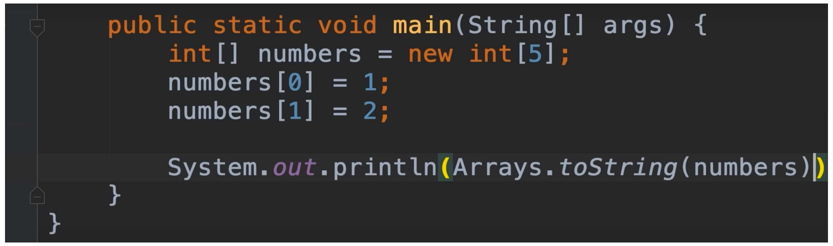


```
/Library/Java/JavaVirtualMachines/jdk-12.0.
[I@38af3868

Process finished with exit code 0
```

- When we print an array java returns a string which is calculated based on the address of this object
- We will use Array Class

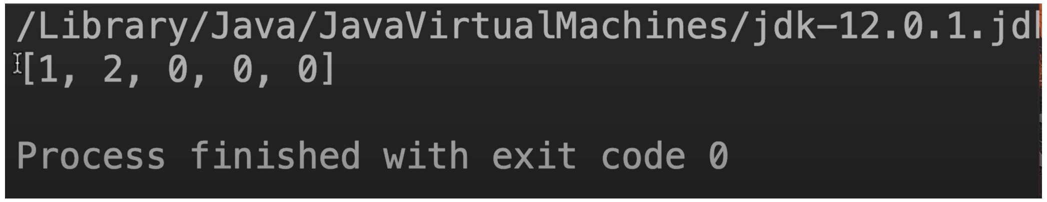
Arrays Class

- We have a class in java called Arrays, defined in java.util package
- We call the method Arrays.toString(primitive/reference type) this will return the string representation of the array



```
    public static void main(String[] args) {
        int[] numbers = new int[5];
        numbers[0] = 1;
        numbers[1] = 2;

        System.out.println(Arrays.toString(numbers));
    }
}
```

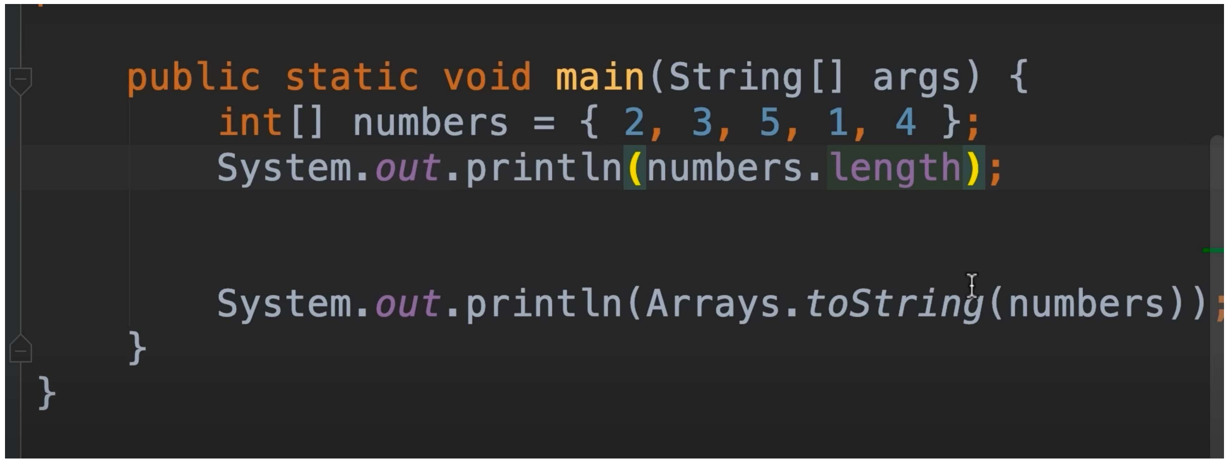
- The first two items have been initialized and the three others are set to zero by default.
- *If we had a Boolean array by default all items are set to false*



```
/Library/Java/JavaVirtualMachines/jdk-12.0.1.jd
[1, 2, 0, 0, 0]

Process finished with exit code 0
```

New way for initialization of array

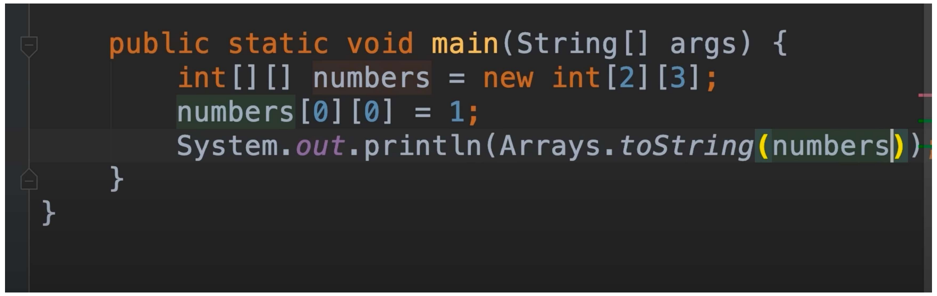
- If we already know the items of the array we can write
  - int [] numbers={2, 3 ,5, 1, 4}
  - numbers.length returns the size of the array
- An array has a fixed size and cannot be changed



```
    public static void main(String[] args) {
        int[] numbers = { 2, 3, 5, 1, 4 };
        System.out.println(numbers.length);


        System.out.println(Arrays.toString(numbers));
    }
}
```

Multi-dimensional Arrays

- To store a matrix we use a two dimensional array

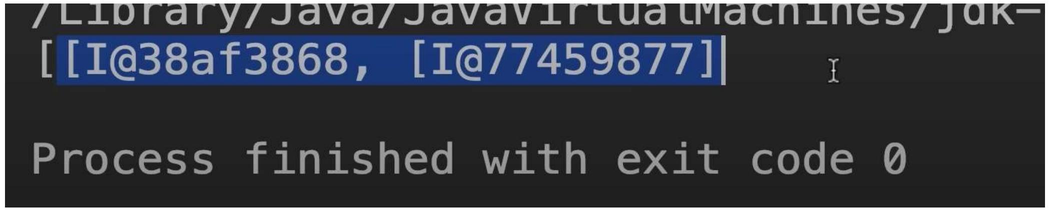


```
    public static void main(String[] args) {
        int[][] numbers = new int[2][3];
        numbers[0][0] = 1;
        System.out.println(Arrays.toString(numbers))
    }
}
```

- A three dimensional array to store data for a cube

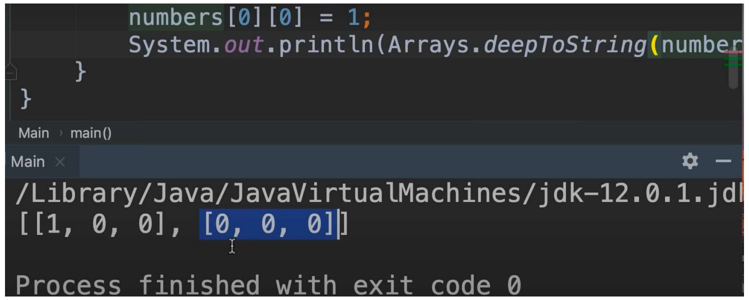
- The System.out.println(Arrays.toString(numbers))

Will return a weird string



```
/Library/Java/JavaVirtualMachines/jdk-
[[I@38af3868, [I@77459877]

Process finished with exit code 0
```

We have to use **the Arrays.deepToString(numbers)** to print the items of the array



```
        numbers[0][0] = 1;
        System.out.println(Arrays.deepToString(number
    }
}
```

Main › main()

Main

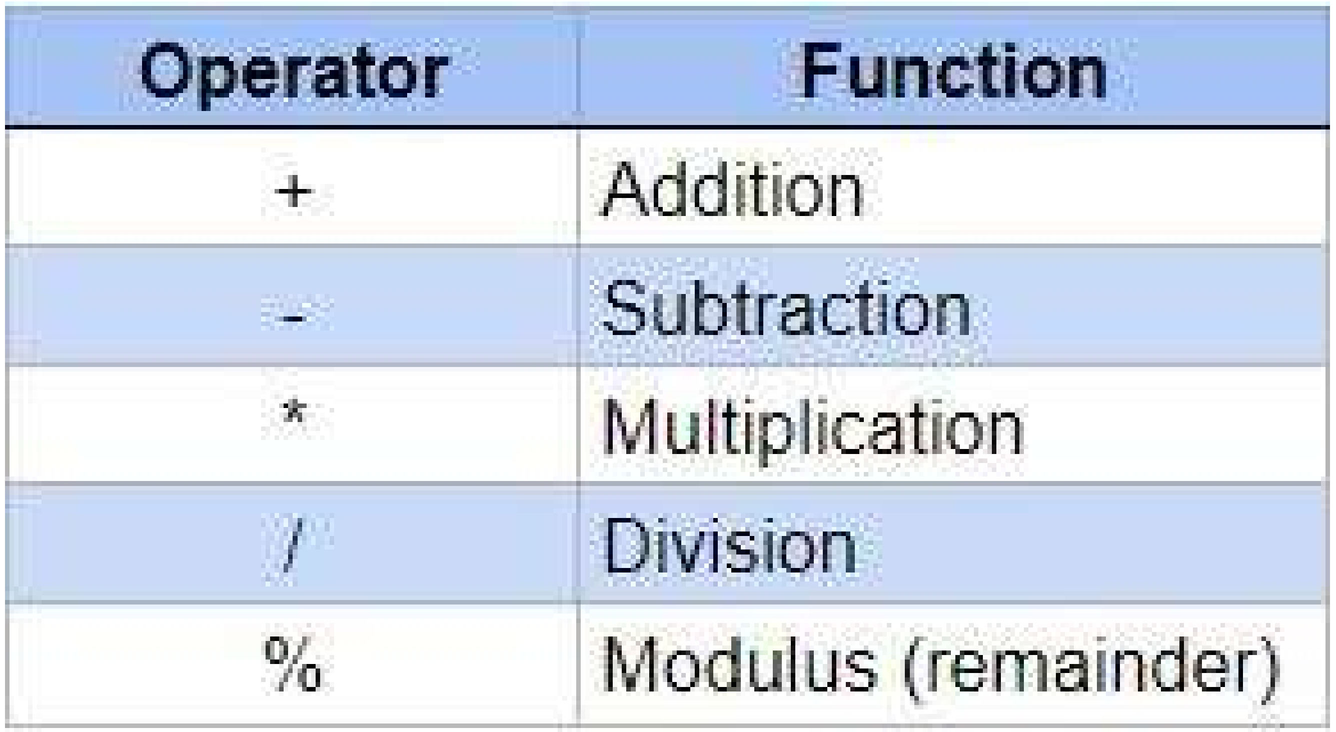
```
/Library/Java/JavaVirtualMachines/jdk-12.0.1.jd
[[1, 0, 0], [0, 0, 0]]

Process finished with exit code 0
```

- Create an array of int with 2 rows and 3 columns
  - Each row is an array itself because it's a list of items
  - int [ ][ ] numbers={{1,2,3},{4,5,6}}

Constant

- There are cases when we want to initialize a variable with a value and this value shouldn’t change.
- We have to write the word final in front of a specific type
- final float PI=3.14F

Arithmetic expressions



Operator	Function
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus (remainder)

```
int x = 5;
System.out.println("x = " + x);
 x += 6;
System.out.println("After x += 6, x = " + x);
x -= 7;
System.out.println("After x -= 7, x = " + x);
 x *= 10;
System.out.println("After x *= 10, x = " + x);
x /= 10;
 System.out.println("After x /= 5, x = " + x);
 x %= 2;
System.out.println("After x %= 2, x = " + x);
```

```
x= 5
After x += 6, x = 11
After x -= 7, x = 4
After x *= 10, x = 40
After x /= 5, x= 4
After x %= 2, x = 0
```

Casting
type conversion

Casting

- Automatic Casting

  short x=1;

  int y=x+2;

  The result is y=3, can we add short with int?

  Short x is first converted to an integer (a data type that is bigger) and then added to 2

  byte , short , int, long, float, double

  double x=1.1;

  double y=x+2;//2 is automatically converted to a double (2.0)

- Explicit Casting

  double x=1.1;

  int y=x+2;//if we don't care about the digits after the decimal point

  int y=(int)x +2// we explicitly cast the x to an integer

  It can happen only in compatible types, numbers you cannot convert a string to an integer with casting.

  In such a case you have to use a wrapper class Integer.parseInt(x)…

Convert String to Int

```
String x=“1”;
int y=Integer.parseInt(x)+2;
```

Math class

- Math.round(float/double) returns an int
  - int y=Math.round(1.1F)
- Math.ceil(float/double) returns a double
  - int y=(int)Math.round(1.1F)
- Math.max(int a, int b)
- Math.random() // value between 0 and 1

- If we want a random int number 0, 100 then we can write
  - (int) (Math.random()*100)

Reading Input

Scanner Class

- We create an object scanner and we specify from where we are going to read it from, a file, the terminal window or what. The Scanner class is found in the java.util package.

```
//reading from the terminal
Scanner scanner=new Scanner(System.in)
```

Method	Description
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads a int value from the user
nextLine()	Reads a String value from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user

Formatting Numbers

NumberFormat Class

NumberFormat helps you to format and parse numbers for any locale.

- Example:
  - Currency :You want to use the $ sign in front of price
  - Percent : you want to write 10% instead of 0.1

We Import java.text.NumberFormat

1. NumberFormat currency=NumberFormat.getCurrencyInstance() ;//we don't use **new** here because **NumberFormat** is an abstract class

   String result=currency.format(1234567.891);

   System.out.println(result); $1,234,567.89

2. NumberFormat percent=NumberFormat.getPercentInstance() ;//we don't use **new** here because **NumberFormat** is an abstract class

   String result= percent.format(0.1);

   System.out.println(result); 10%

Mortgage Calculator

Calculating Mortgage Payments

- r: is your monthly interest rate calculated by dividing your annual interest rate by 12
- n: number of payments
- *P: principle*
- Mortage $M = p\,\frac{r(1+r)^n}{(1+r)^n-1}$

- r: είναι το μηνιαίο επιτόκιο που υπολογίζεται διαιρώντας το ετήσιο επιτόκιο με το 12
- n: αριθμός πληρωμών
- p: ποσό δανείου
- M μηνιαία δόση

```
import java.util.Scanner;

import java.text.NumberFormat;

class MortageCalulator {

   public static void main(String[] args) {

      final byte MONTHS_IN_YEAR=12;

      final byte PERCENT=100;

      Scanner scanner=new Scanner(System.in);

      System.out.print("Principle:");

      int principle=scanner.nextInt();

       System.out.print("Annual interest rate (epitokio):");

       float interestRate=scanner.nextFloat();

       float monthlyInterestRate=interestRate/PERCENT/MONTHS_IN_YEAR;

        System.out.print("Number of years:");

        byte years=scanner.nextByte();

        int numberOfPayments=years*MONTHS_IN_YEAR;

        double mortage=principle*monthlyInterestRate*
                     Math.pow(1+monthlyInterestRate,numberOfPayments)/(Math.pow(1+monthlyInterestRate,numberOfPayments)-1);

        System.out.println("Mortage: "+mortage);

        String mortageFormatted=NumberFormat.getCurrencyInstance().format(mortage);

         System.out.println("Mortage: "+mortageFormatted);

      }

}
```

result

```
Principle:100,000
Annual interest rate (epitokio):3
Number of years:20
Mortage: 554.5883416606952
Mortage: $554.59
```