Programming in Java - Part 01 Introduction

Paolo Vercesi
ESTECO SpA

The Java platform

Data types, operators, flow control

Classes

Packages and libraries

The Java platform

«Hello, World!» explained

Java file → **HelloWorld.java**

Class declaration →

Method declaration →

```
public class HelloWorld {

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

The public class has the same name of the Java file

Java programs start from the main method

```
$ javac HelloWorld.java
$ ls
HelloWorld.class  HelloWorld.java
$ java HelloWorld
Hello, World!
```

We turn javac compiler Java source files into class files by using the javac compiler

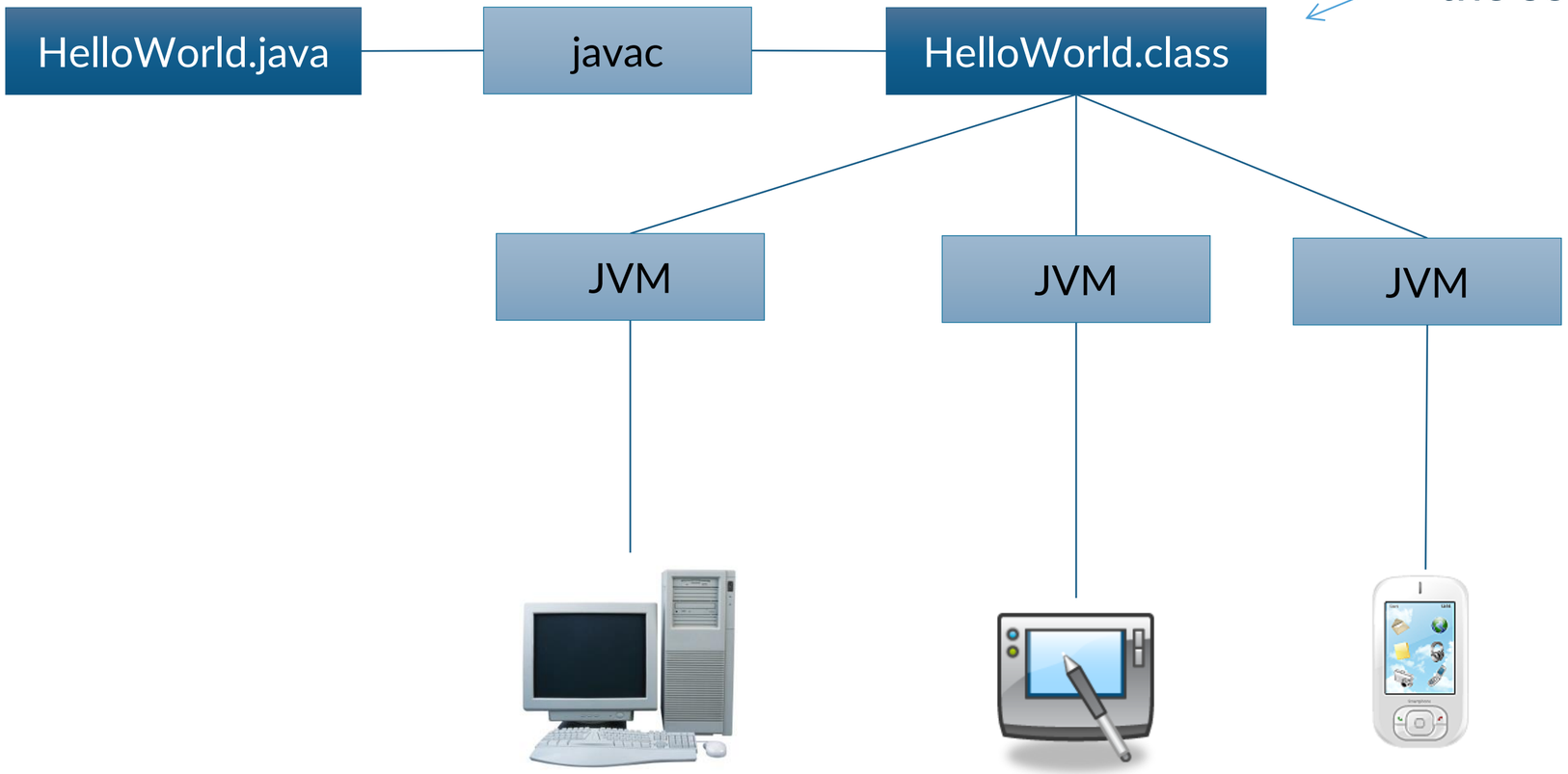
The javac compiler takes a list of source Java files and it compiles the corresponding class files

A class file is created for each class defined in the source files

To run a java main method we invoke the java virtual machine (JVM) on the class containing the main method

The Java platform

The output of the java compiler is not executable code but it is the so-called bytecode



The compiled code is independent of the processor of the device in which is running

Comparison with Python and C++

hello_world.py

```python
def main():
    print('Hello, World!')


if __name__ == "__main__":
    main()
```

HelloWorld.cpp

```cpp
#include <iostream>

int main()
{
    std::cout << "Hello, World!" << std::endl;
    return 0;
}
```

```
$ python hello_world.py
Hello, World!
```

```
$ g++ HelloWorld.cpp
$ a.out
Hello, World!
```

Python compiles scripts into .pyc files inside the __pycache__ directory

Which Java?

JRE or JDK

The Java Runtime Environment (JRE) contains all you need to launch a class file

The Java Development Kit (JDK) contains the JRE plus all you need to compile class files

Which version

The latest Long Term Support (LTS) Java version is Java 21

Java releases follow a 6 months cycle, but LTS are released every 2 years.
Latest version is Java 23

Which vendor

There are many “vendors”

- Oracle
- Amazon
- IBM
- openJDK
  https://adoptium.net/

Additional content - The Java documentation

The official documentation is available here
https://docs.oracle.com/en/java/javase/21/

The API documentation is here
https://docs.oracle.com/en/java/javase/21/docs/api/index.html

The Java language evolution is driven by the Java Community Process (JCP)
https://www.jcp.org/en/home/index

The JCP is the mechanism for developing standard technical specifications for Java technology. Anyone can register for the site and participate in reviewing and providing feedback for the Java Specification Requests (JSRs), and anyone can sign up to become a JCP Member and then participate on the Expert Group of a JSR or even submit their own JSR Proposals.

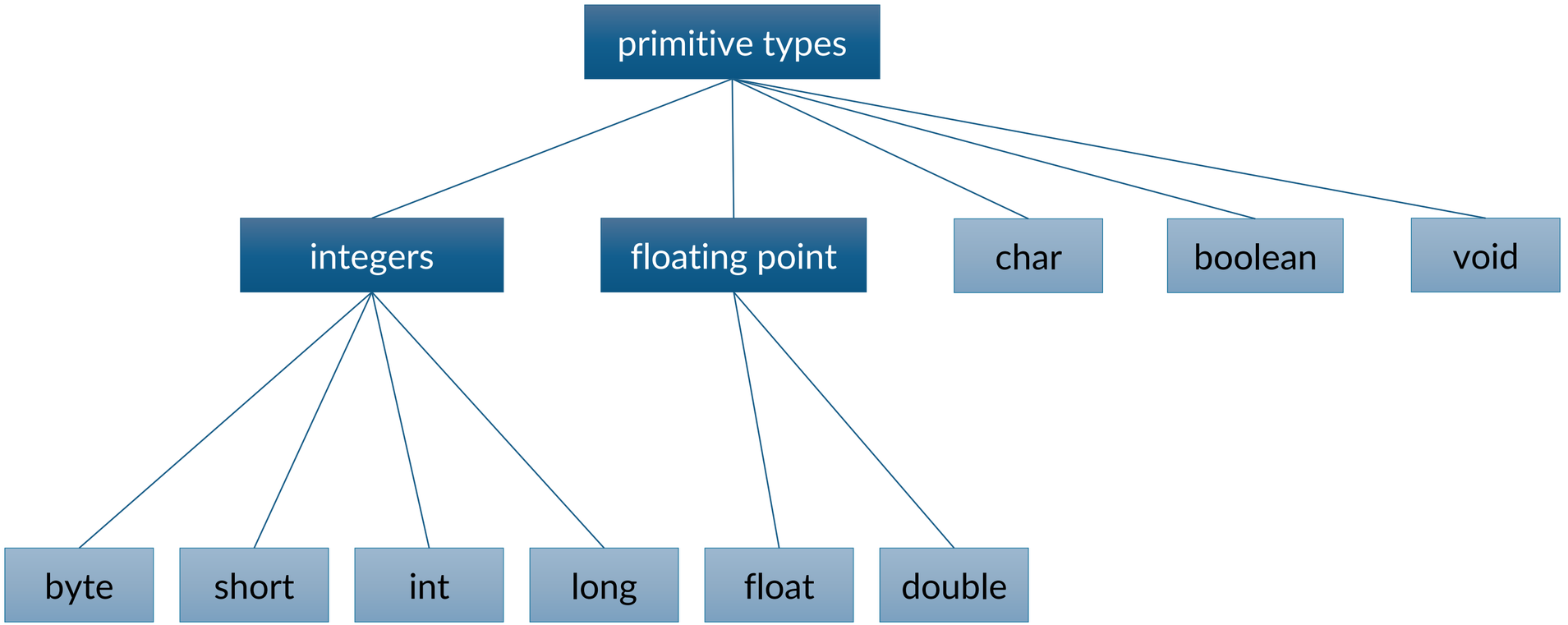
Informal place to discuss the new features of Java is the JDK Enhancement Proposals (JEP)
https://openjdk.java.net/jeps/0

Release roadmap
https://www.oracle.com/java/technologies/java-se-support-roadmap.html

Data types, operators, flow control

Primitive types

Java provides the following primitive types



Variable and constant definition

```
int x;
double d = 0.33;
float f = 0.22F;
char c = 'a';
boolean ready = true;

x = 15;
```

Variables are declared specifying their type and name, and initialized in the point of declaration, or later with the assignment expression

They cannot be used before initialization

Constants are declared with the word final in front, the specification of the initial value is compulsory

```
final double pi = 3.1415;
final int maxSize = 100_000;
final char lastLetter = 'z';
```

```
var f = 10.0; // a double variable
var i = 50;      // an int variable
```

Only local variables can be declared without an explicitly declared type by using the so-called type inference

Data type ranges

Integer type	Width [bits]	Range
byte	8	-128 to 127
short	16	-32,768 to 32767
int	32	-2,147,483,648 to 2,147,483,647
long	64	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
char	16	0 to 65535

Floating point type	Width [bits]	Range
float	32	1.4e-45 to 3.4e+38
double	64	4.9e-324 to 1.8e+308

Differences with Python and C++	
Python	No primitive data types, everything is an object, e.g., integers have unlimited precision
C++	The range of integer and floating-point type is implementation dependent with some constraints, e.g., minimum width for integer types. Furthermore, C++ allows integer types with the unsigned modifier

Type conversion and casting

Java performs automatic conversions when the magnitude of a numeric value is preserved, widening primitive conversion

- from int to long
- from long to double
- from float to double
- …

When the magnitude of a numeric values cannot be preserved, you must declare the explicit type conversion, narrowing primitive conversion or casting

Narrowing conversion	Rules
from integer to integer (e.g., long to int)	Integer component is reduced modulo the target type size
from floating point to integer (e.g., double to int)	Fractional component is truncated Integer component is reduced modulo the target type size
from double to float	The number is rounded to the closest float, including +Infinity and -Infinity

Explicit type conversion - Casting

TestCast.java

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

    int a = 'x';          // 'x' is a character
    long b = 34;           // 34 is an int
    float c = 1002;        // 1002 is an int
    double d = 3.45F;      // 3.45F is a float

    int f = (int) b;            // b is a long
    float h = (float) d;        // d is a double
  }
}
```

When the magnitude of a numeric value cannot be preserved, it is compulsory to use the cast operator

Strings

Strings are not a primitive type, but they are objects of the String class

```
String a = "abc";
```

The string concatenation operator '+' converts the argument on the right to a String

```
int cost = 2;
String b = "the cost is " + cost + " euro";
```

What's the output of?

```
String bb = "the cost is " + cost + cost + " euro";
System.out.println(bb);
```

Array declaration

Arrays are used to group elements of the same type

The declaration does not specify a size nor allocates any memory

```
int[] a;
double[] b;
String[] c;
```

```
int size = 3;
double[] d1, d2;

d1 = new double[3];
d2 = new double[size];
```

We allocate memory for arrays by using the new operator

Arrays can be allocated and initialized while declared by using an array initializer expression

```
int[] a = {13, 56, 2034, 4, 55};
double[] b = {1.23, 2.1};
String[] c = {"Java","is","great"};
```

Working with arrays

Java arrays are 0-based and every array has a field called length representing the length of the array

```
int len = a.length;
```

We access the array components by using the [] operator with an integer index from 0 to length-1

```
a[2] = 1000;
```

```
int[] a = new int[2];
System.out.println(a[0]);
System.out.println(a[1]);
```

```
0
0
```

Components of the arrays are initialized with the default value of the type

```
String[] b = new String[2];
System.out.println(b[0]);
System.out.println(b[1]);
```

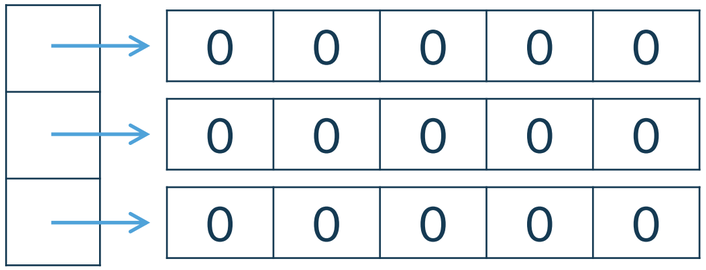
```
null
null
```

null is not the default values for objects, but components are initialized using null

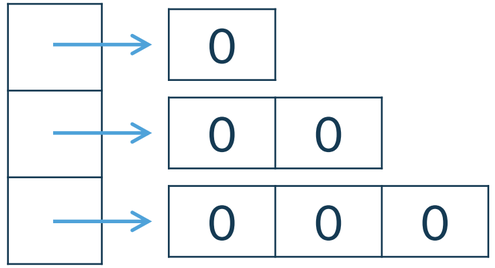
Multidimensional arrays

Multidimensional arrays are arrays of arrays

```
int[][] a = new int[3][5];
```




```
int[][] a = new int[3][];
a[0] = new int[1];
a[1] = new int[2];
a[2] = new int[3];
```



In Java is possible to define jagged arrays

When you allocate a multidimensional array only the first dimension is compulsory

Working with multidimensional arrays

Multidimensional arrays can be initialized by nesting array initializers

```
int[][] a = new int[][] {
                            {1},
                            {1, 1},
                            {1, 2, 1},
                            {1, 3, 3, 1}
                          };
```

```
int[][][] b = new int[2][][];

b[0] = new int[3][];

b[0][0] = new int[7];

b[0][1] = a[0];

b[0][2] = a[2];

b[1] = a;
```

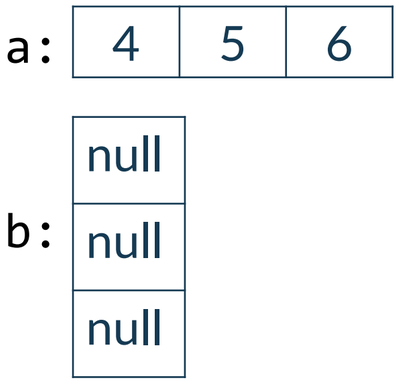
A multidimensional array is an array of arrays

In Java arrays are objects, so a multidimensional array is an array of references to other arrays

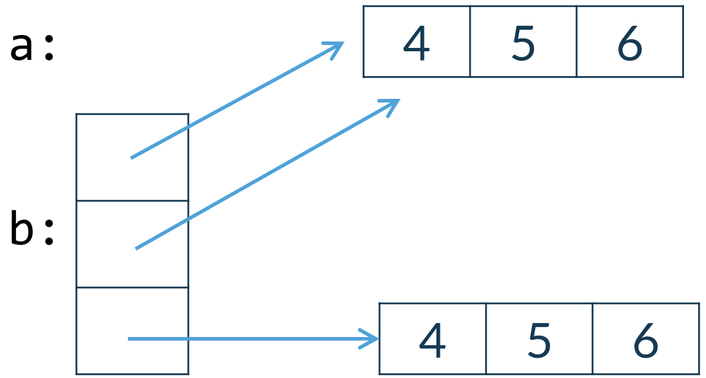
Objects, references, and the new operator

We've not yet formally introduced the concept of objects and references, but we have already used them in combination with the new operator

```
int[] a = new int[] {4, 5, 6};

int[][] b = new int[3][];
```

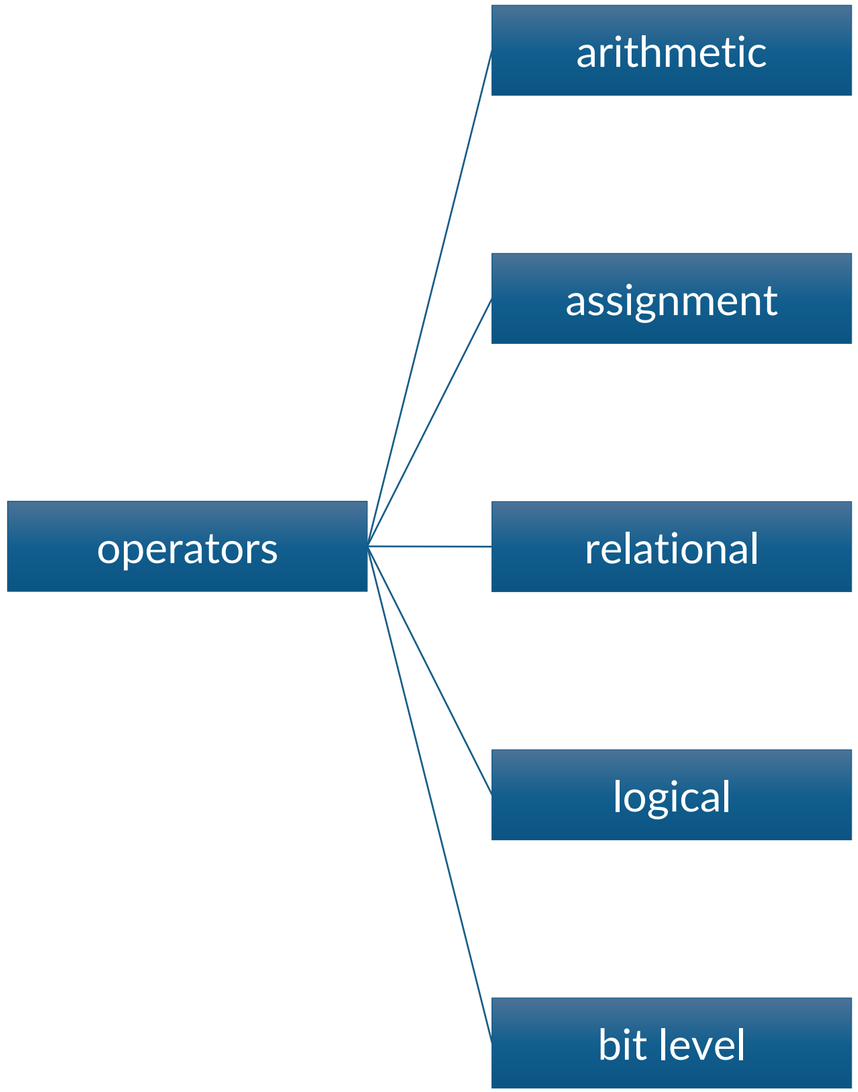



```
b[0] = a;

b[1] = a;

b[2] = new int[] {4, 5, 6};
```

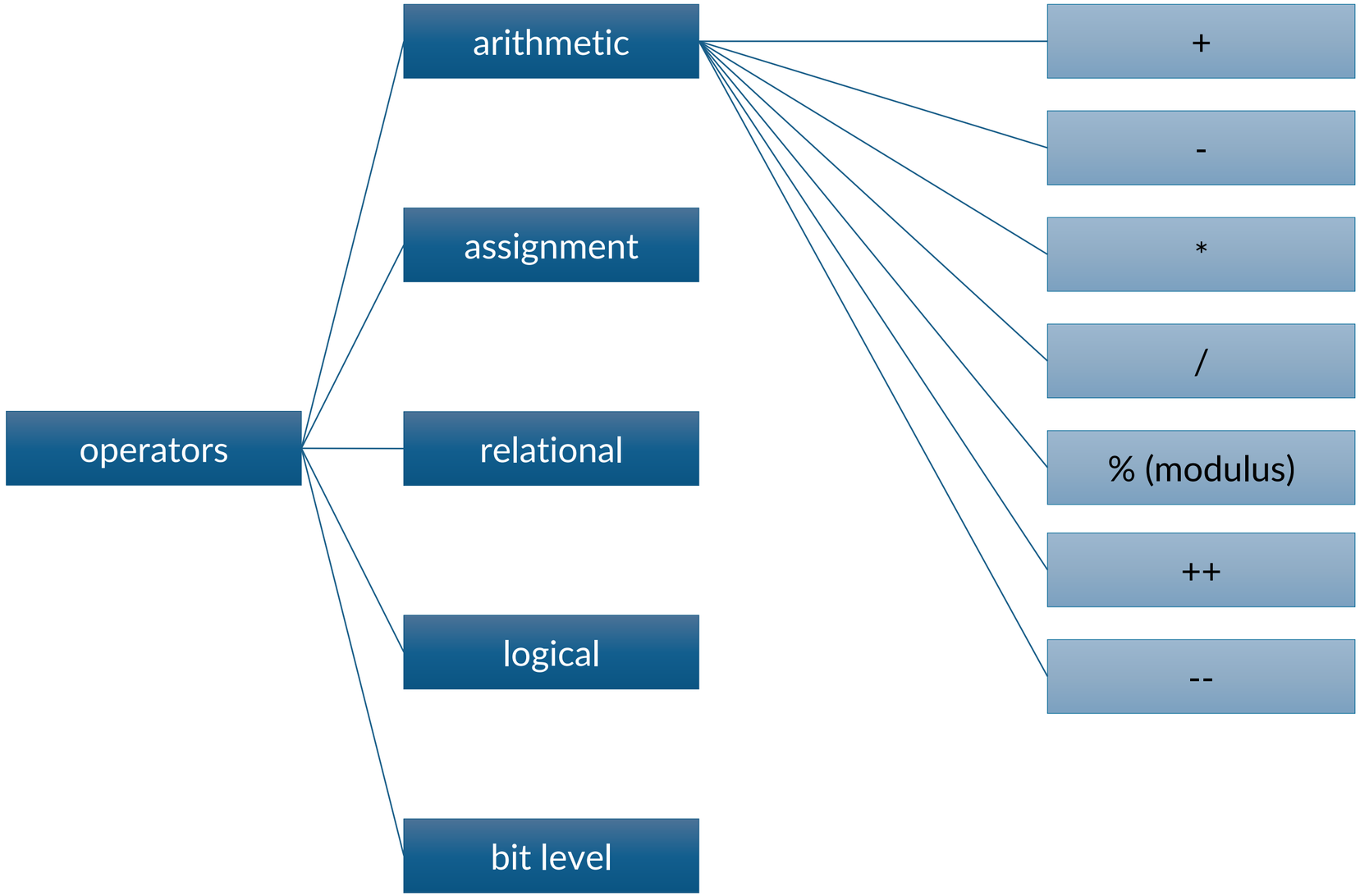


What does it happen when we set `b[1][1] = 3;` ?

Operators



Arithmetic operators



Type promotion in arithmetic expressions

byte, short and char operands are always converted to int in arithmetic expressions

If an operand is a long, the whole expression is converted to long

If one operand is a float, the whole expression is converted to float

If one operand is a double, the whole expression is converted to double

```
byte b1 = 3;
byte b2 = 4;
byte b3 = b1 * b2; // Incompatible types
byte b4 = (byte) (b1 * b2);
```

Can you explain this result?

```
double q = 3 / 2; // 1 !!!!
```

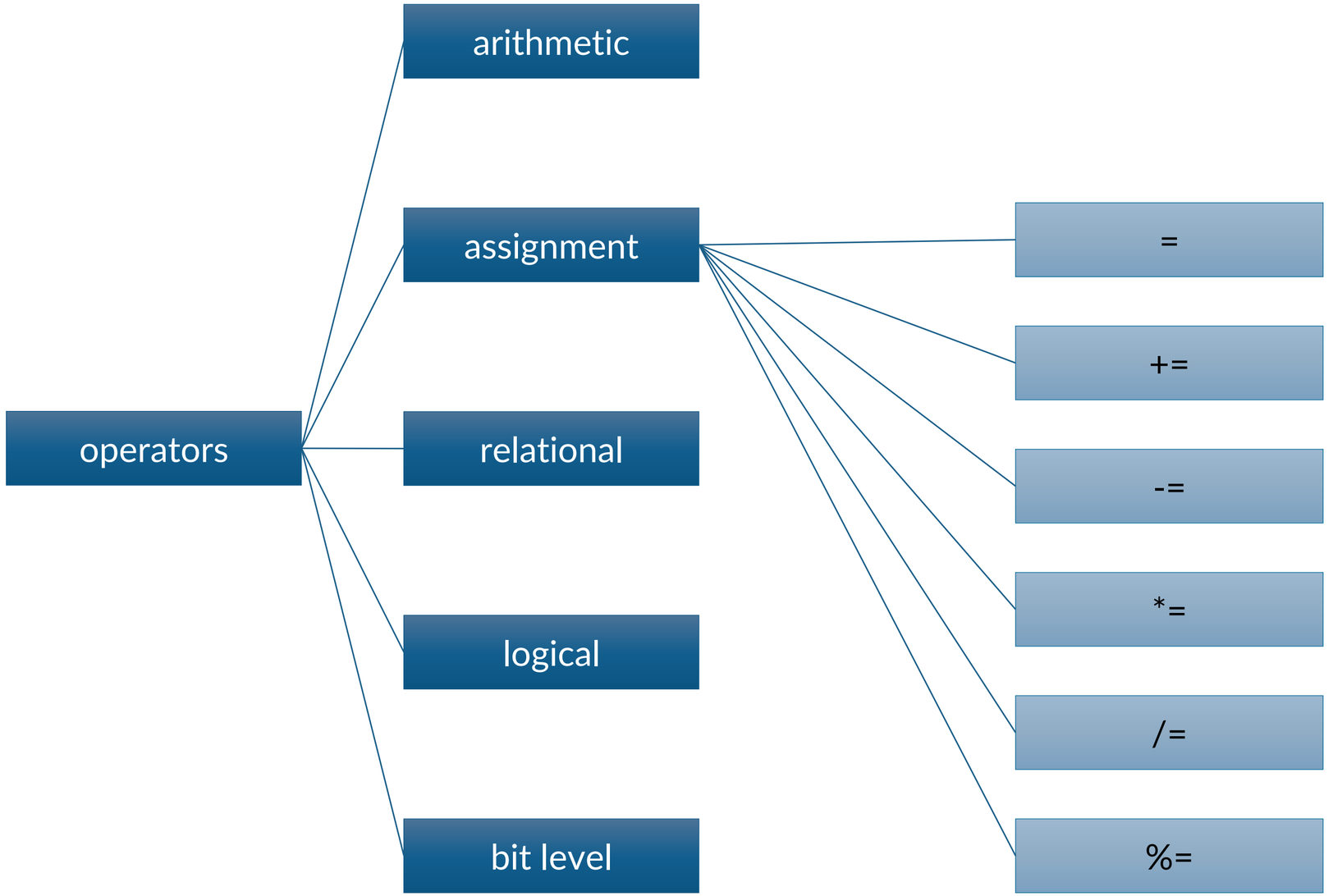
Example with arithmetic operators

Arithmetic.java

```
public class Arithmetic {
  public static void main(String[] args) {
    int x = 12;
    x += 5; // x = x + 5
    System.out.println(x);

    int a = 12, b = 12;
    System.out.print(a++);    // printed and then incremented
    System.out.print(a);

    System.out.print(++b);    // incremented and then printed
    System.out.println(b);
  }
}
```

```
$ java Arithmetic
17
12 13 13 13
```

Assignment operators



Example with assignment operators

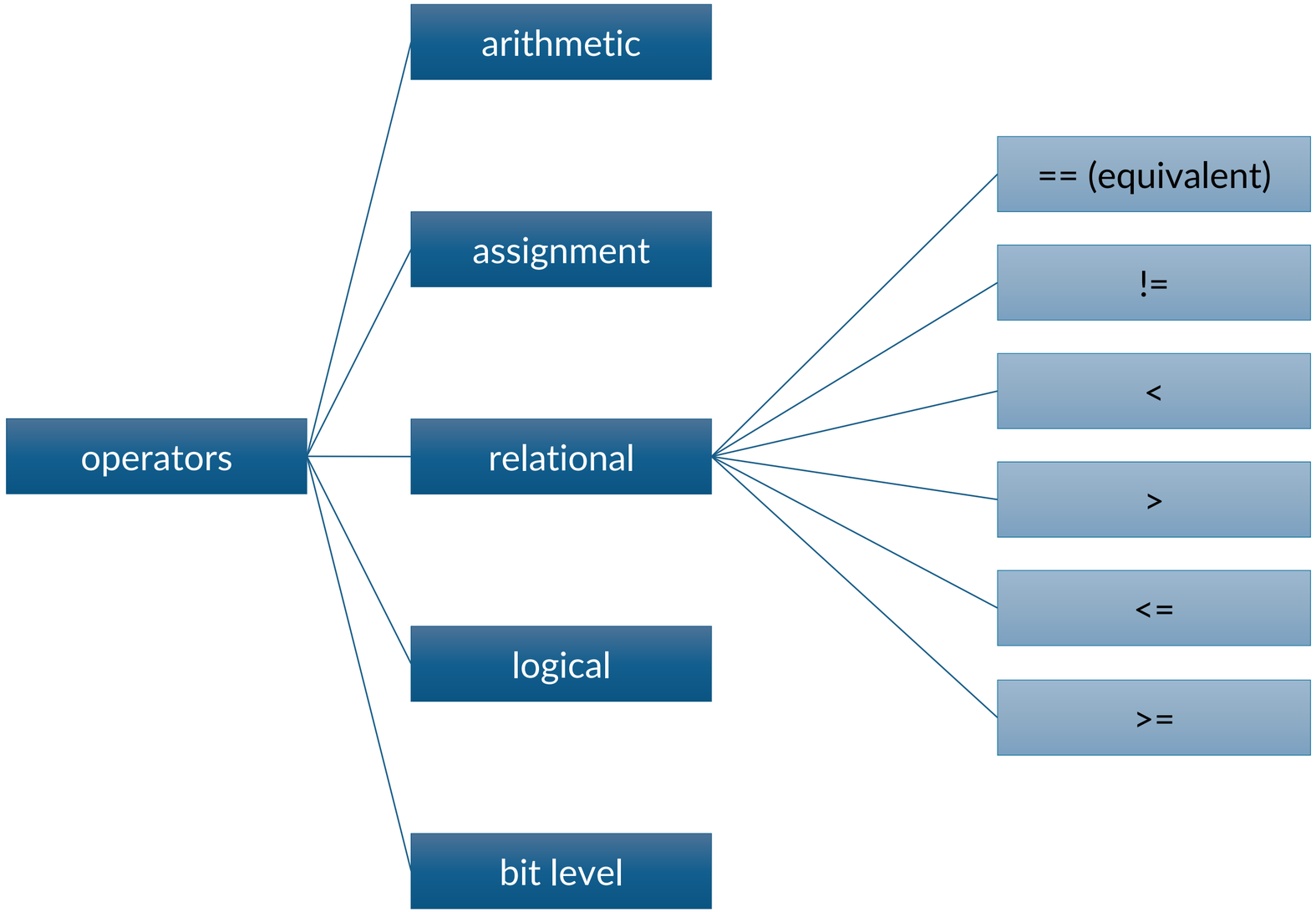
Assignment.java

```
public class Assignment {
  public static void main(String[] args) {
    int x = 12, y = 33;
    double d = 2.45, e = 4.54;

    y %= x;
    d -= e;
    System.out.println(y);
    System.out.println(d);
  }
}
```

a op= b is equivalent to
a = a op b

```
$ java Assignment
9
-2.09
```

Relational operators



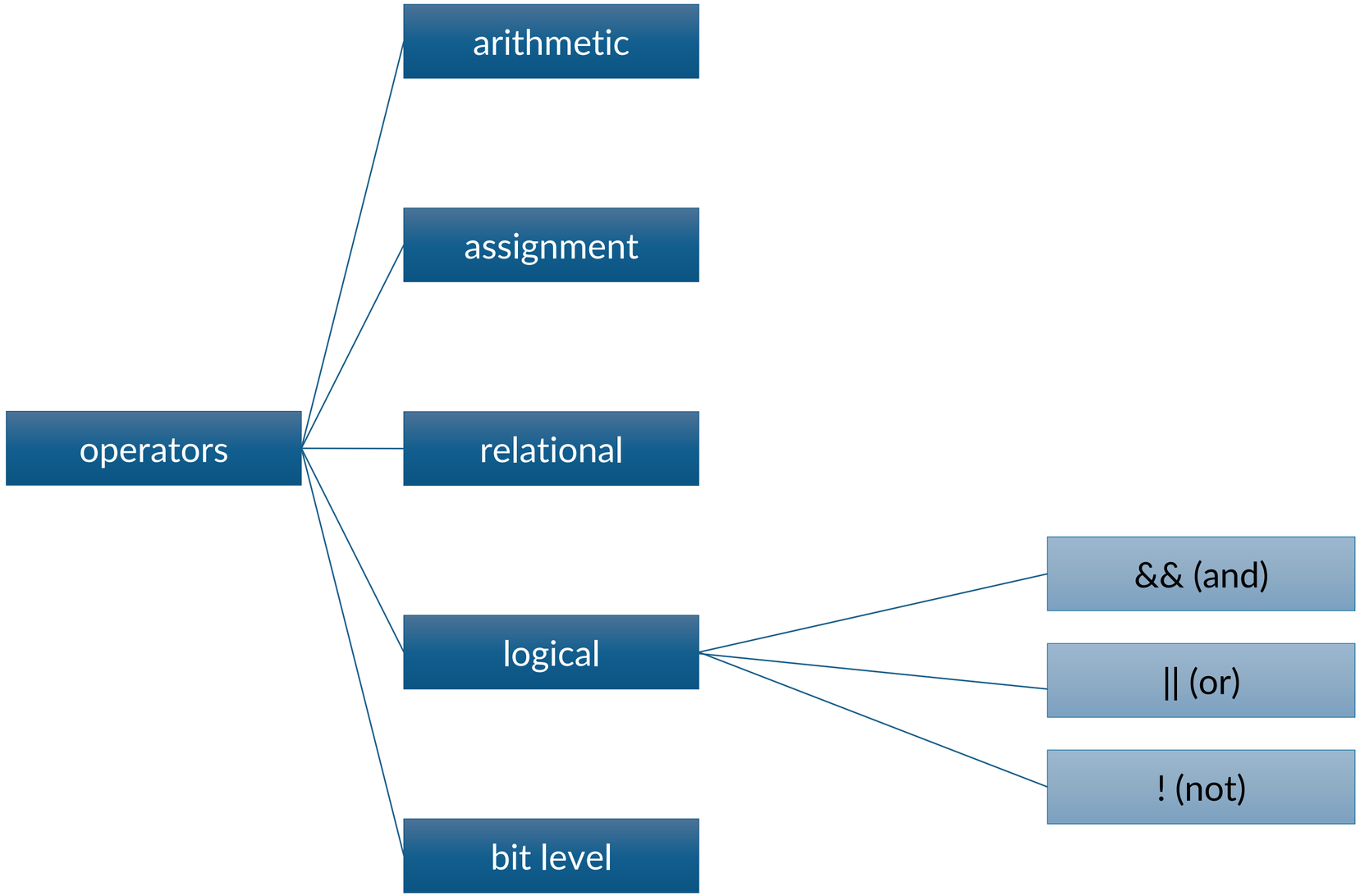
Example with relational operators

TestBoolean.java

```
public class TestBoolean {
  public static void main(String[] args) {
    int x = 12, y = 33;

    System.out.println(x < y);
    System.out.println(x != y - 21);

    boolean test = x >= 10;
    System.out.println(test);
  }
}
```

```
$ java TestBoolean
true
false
true
```

Logical operators



Example with logical operators

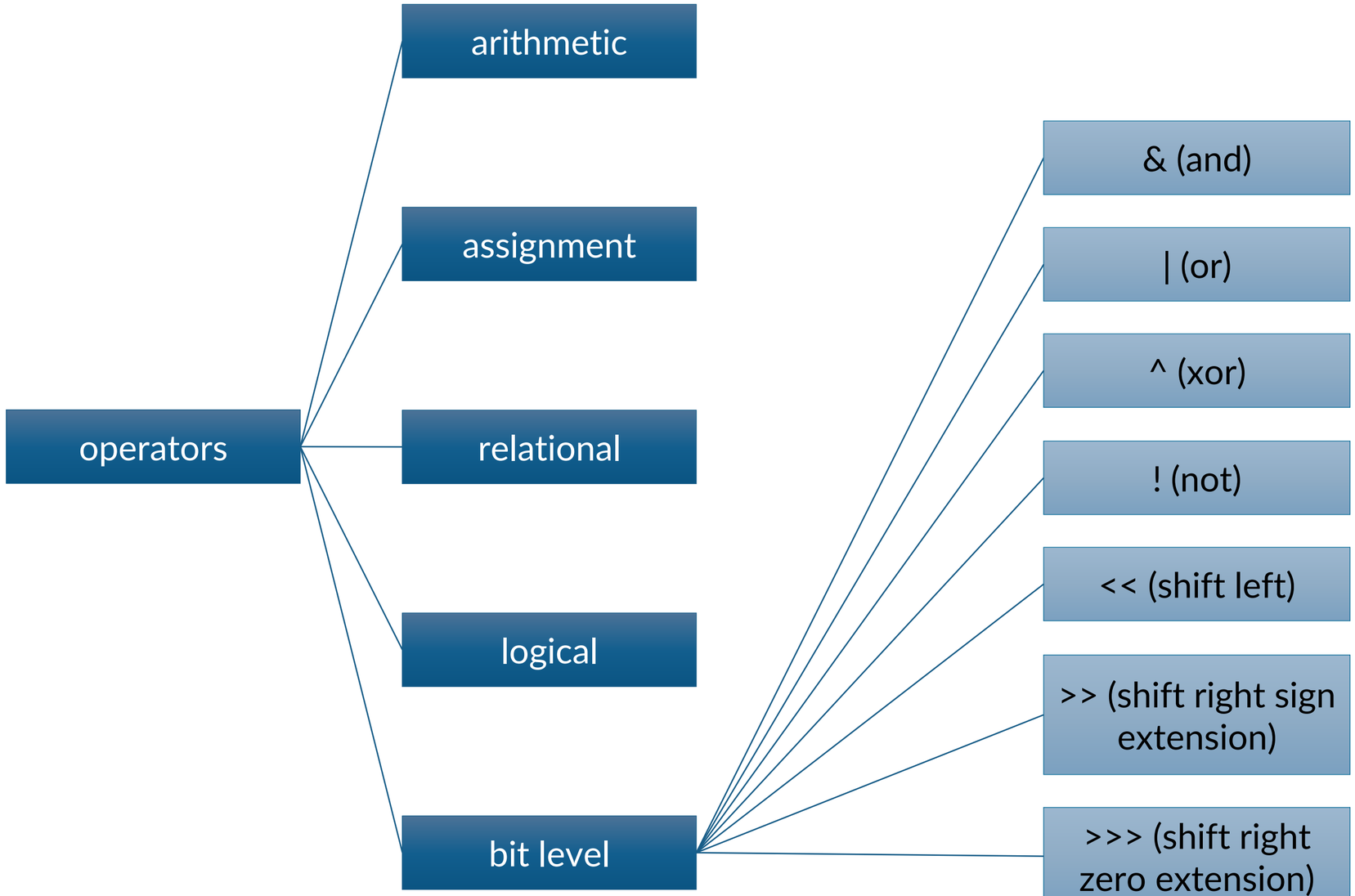
Logical.java

```
public class Logical {
  public static void main(String[] args) {
    int x = 12, y = 33;
    double d = 2.45, e = 4.54;

    System.out.println(x < y && d < e);
    System.out.println(!(x < y));

    boolean test = 'a' > 'z';
    System.out.println(test || d - 2.1 > 0);
  }
}
```

```
$ java Logical
true
false
true
```

Please note that there are also logical non-short circuit operators. Investigate about them

Bit level operators



Example with bit-level operators

Bits.java

```
public class Bits {
  public static void main(String[] args) {
    int x = 0b00000000000000000000000000010110;
    int y = 0b00000000000000000000000000110011;

    System.out.println(x & y);                  // 00000000000000000000000000010010
    System.out.println(x | y);                  // 00000000000000000000000000110111
    System.out.println(x ^ y);                  // 00000000000000000000000000100101
    System.out.println(~x);                     // 11111111111111111111111111101001

    x = 0b00000000000000000000000000001001; // 9
    System.out.println(x >> 3);                 // 00000000000000000000000000000001
    System.out.println(x >>>3);                 // 00000000000000000000000000000001

    x = -9;                                     // 11111111111111111111111111110111
    System.out.println(x >> 3);                 // 11111111111111111111111111111110
    System.out.println(x >>>3);                 // 00011111111111111111111111111110
  }
}
```

The ? operator

Sort of if-then-else that given a conditional expression chooses between two expressions

```
condition ? expression1 : expression2
```

If condition is true, expression1 is evaluated, otherwise expression2 is evaluated

The ?-expression assumes the result of the evaluated expression

```
System.out.println(expression ? "It rains" : "It doesn’t rain")
```

Flow control statements

Selection

- if-then-else
- switch

Iteration

- while
- do-while
- for
- for-each

Jump

- continue
- break
- return

If-then-else

If.java

```
public class If {
  public static void main(String[] args) {
    char c = 'x';

    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
      System.out.println("letter: " + c);
    else
      if (c >= '0' && c <= '9')
        System.out.println("digit: " + c);
      else {
        System.out.println("the character is: " + c);
        System.out.println("it is not a letter nor a digit");
      }
   }
}
```

```
$ java If
letter: x
```

Switch

Switch.java

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

    boolean leapYear = true;
    int days = 0;

    for (int month = 1; month <= 12; month++) {
      switch (month) {
        case 2: days += leapYear ? 29 : 28; break;
        case 4:
        case 6:
        case 9:
        case 11: days += 30; break;
        default: days += 31; break;
      }
    }
    System.out.println(days);
  }
}
```

"Thirty days has September,
April, June, and November,
All the rest have thirty-one,
Save February at twenty-eight,
But leap year, coming once in four,
February then has one day more."

```
$ java Switch
366
```

Grouped switch

GroupedSwitch.java

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

    boolean leapYear = true;
    int days = 0;

    for (int month = 1; month <= 12; month++) {
      switch (month) {
        case 2: days += leapYear ? 29 : 28; break;
        case 4, 6, 9, 11: days += 30; break;
        default: days += 31; break;
      }
    }
    System.out.println(days);
  }
}
```

```
$ java GroupedSwitch
366
```

By using the comma ‘,’ we can group together multiple cases

Enhanced switch

EnhancedSwitch.java

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

    boolean leapYear = true;
    int days = 0;

    for (int month = 1; month <= 12; month++) {
      switch (month) {
        case 2 -> days += leapYear ? 29 : 28;
        case 4, 6, 9, 11 -> days += 30;
        default -> days += 31;
      }
    }
    System.out.println(days);
  }
}
```

```
$ java EnhancedSwitch
366
```

The traditional colon ‘:’ and break can be replaced by the arrow ‘->’

Switch expression

SwitchExpression.java

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

    boolean leapYear = true;
    int days = 0;

    for (int month = 1; month <= 12; month++) {
      days += switch (month) {
        case 2: yield leapYear ? 29 : 28;
        case 4, 6, 9, 11: yield 30;
        default: yield 31;
      };
    }
    System.out.println(days);
  }
}
```

```
$ java SwitchExpression
366
```

The switch expression construct directly returns a value by using the yield keyword

Expressions terminate with a semicolon

Not exactly a flow control statement!

Enhanced switch expression

EnhancedSwitchExpression.java

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

    boolean leapYear = true;
    int days = 0;

    for (int month = 1; month <= 12; month++) {
      days += switch (month) {
        case 2 -> leapYear ? 29 : 28;
        case 4, 6, 9, 11 -> 30;
        default -> 31;
      };
    }
    System.out.println(days);
  }
}
```

```
$ java EnhancedSwitchExpression
366
```

The colon ‘:’ and yield can be replaced by the arrow ‘->’ providing a very compact syntax

Summary of switch

	Traditional ':'	Enhanced '->'
Switch	traditional use	no need to break the fall-through
Expression switch	use yield to return the case value	no need to break the fall-through no need to yield to return the case value

The case-expression must evaluate to
byte, short, char, int, enum, or String

Cases can be grouped together

Add a semicolon ';' after the switch expression

While

While.java

```
public class While {
  public static void main(String[] args) {
    final double initialValue = 2.34;
    final double step = 0.11;
    final double limit = 4.69;
    double var = initialValue;

    int counter = 0;
    while (var < limit) {
      var += step;
      counter++;
    }
    System.out.println("Incremented " + counter + " times");
  }
}
```

```
$ java While
Incremented 22 times
```

Do-while

DoWhile.java

```
public class DoWhile {
  public static void main(String[] args) {
    final double initialValue = 2.34;
    final double step = 0.11;
    final double limit = 4.69;
    double var = initialValue;

    int counter = 0;
    do {
      var += step;
      counter++;
    } while (var < limit);
    System.out.println("Incremented " + counter + " times");
  }
}
```

```
$ java DoWhile
Incremented 22 times
```

For

For.java

```
public class For {
  public static void main(String[] args) {
    final double initialValue = 2.34;
    final double step = 0.11;
    final double limit = 4.69;
    int counter = 0;

    for (double var = initialValue; var < limit; var += step)
      counter++;
    System.out.println("Incremented " + counter + " times");
  }
}
```

```
$ java For
Incremented 22 times
```

For-each

ForEach.java

```
public class ForEach {
    public static void main(String[] args) {
        int[] a = {2,4,3,1};

        int sum = 0;
        for (int x : a) {
            sum += x;
        }

        double[] d = new double[sum];
        for (int i = 0; i < d.length; i++) {
            d[i] = 1.0 / (i+1);
        }

        for (var f : d) {
           System.out.println(f);
        }
    }
}
```

Continue

Continue.java

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

    for (int counter = 0; counter < 10; counter++) {

      if (counter % 2 == 1) continue; // start a new iteration if the counter is odd

      System.out.println(counter);
    }
    System.out.println("done.");
  }
}
```

```
$ java Continue
0 2 4 6 8 done.
```

Break

Break.java

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

    for (int counter = 0; counter < 10; counter++) {

      if (counter % 2 == 1) continue; // start a new iteration if the counter is odd
      if (counter == 8) break; // exit the loop if the counter is equal to 8

      System.out.println(counter);
    }
    System.out.println("done.");
  }
}
```

```
$ java Continue
0 2 4 6 done.
```

Return

Return.java

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

    for (int counter = 0; counter < 10; counter++) {

      if (counter % 2 == 1) continue; // start a new iteration if the counter is odd
      if (counter == 8) return; // exit the method if the counter is equal to 8

      System.out.println(counter);
    }
    System.out.println("done.");
  }
}
```

```
$ java Return
0 2 4 6
```

Return

Sum.java

```
public class Sum {
    public static int sum(int[] array) {
        int s = 0;
        for (int i : array) {
            s = s + i;
        }
        return s;
    }

    public static void main(String[] args) {
        int[] a = {71, 4, 31, 53};
        System.out.println(sum(a));
    }
}
```

```
$ java Sum
159
```

Exercise - A revised «Hello, World!»

```
$ java Hello.java Paolo
Hello, Paolo!
```

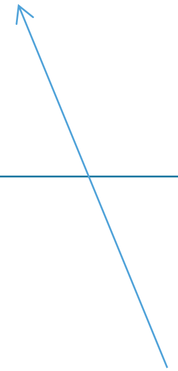
Hello.java

```java
public class Hello {

    public static void main(String[] args) {
       System.out.println("Hello, " + args[0] + "!" );
    }
}
```

Assignment 1

Implement a Hello class to say hello to multiple people

```
$ java Hello Paolo Dario
Hello Paolo and Dario!

$ java Hello Francesco Joe Arthur
Hello Francesco, Joe, and Arthur!

$ java Hello
Hello everybody!
```



Note the usage of
the Oxford comma

Classes

A sample class

```
class Television {
    String model;
    boolean on;
    int channel;
    int volume;
}
```

Definition of a class with four instance variables, fields

Creation of a new object of class Television new + constructor

Declaration of a variable of type Television

```
new Television();
Television tv;
tv = new Television();
```

Creation and assignment of an object of class Television. The variable tv holds a reference to the new object.

```
tv.model = "LG5464VX";
```

Instance variables are accessed using the dot notation

Classes

Classes are used to define new types, once a class is defined, we can then create new objects of that class, These objects are instances of that class, and that class is the type of these objects

As the words class and type can be used interchangeably, so object and instance can be used interchangeably too

A class is composed of

- (instance) methods, that define the operations that can be performed on its instances
- (instance) variables, that define the data that is associated to an object

Methods and instance variables are collectively known as members

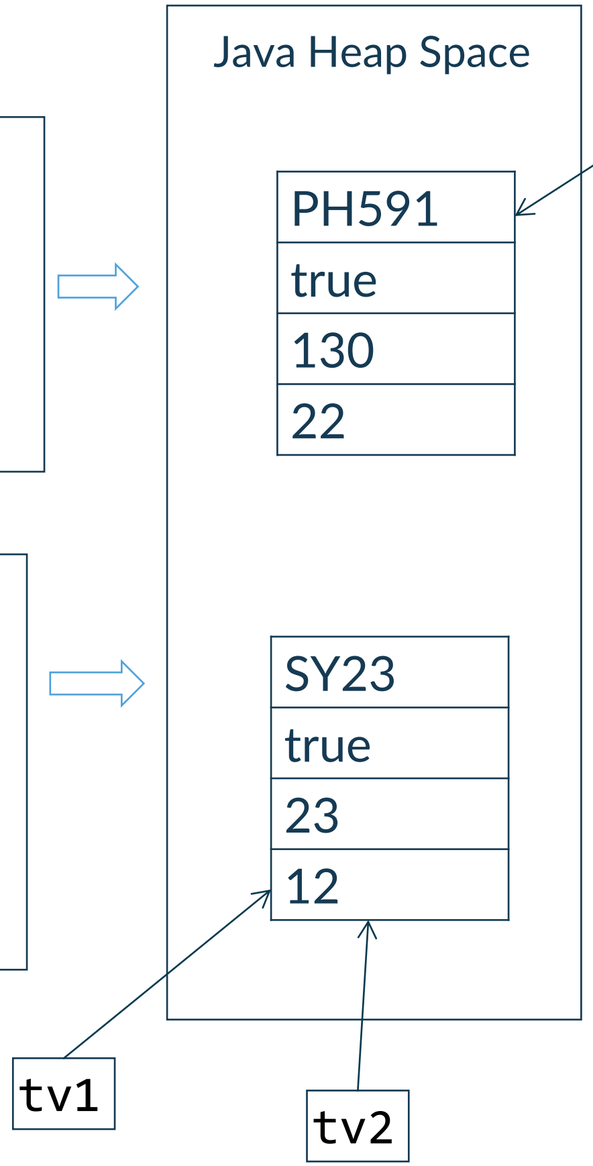
(Instance) methods define how other entities can interact with the objects of that class

(Instance) variables differentiate the behavior of different instances of the same class

Classes and references

```
var tv = new Television();
tv.model = "PH591";
tv.on = true;
tv.channel = 130;
tv.volume = 22;
```

```
Television tv1 = new Television();
Television tv2 = tv1;
tv1.model = "SY23";
tv1.on = true;
tv2.channel = 23;
tv2.volume = 12;
```



When we create an object, by using the new operator, we allocate the memory to hold the instance variables of this new object

Two, or more, references to the same object point to the same memory location.

Objects vs. primitive types or refences vs. values

```
int i1 = 10;
int i2 = i1; //we copy the value of i1 in i2
i1 = i1 + 1; //now i1 is 11, i2 is still 10
```

Variables of primitive types hold the value represented by the primitive

Variables of object references hold a reference to the object

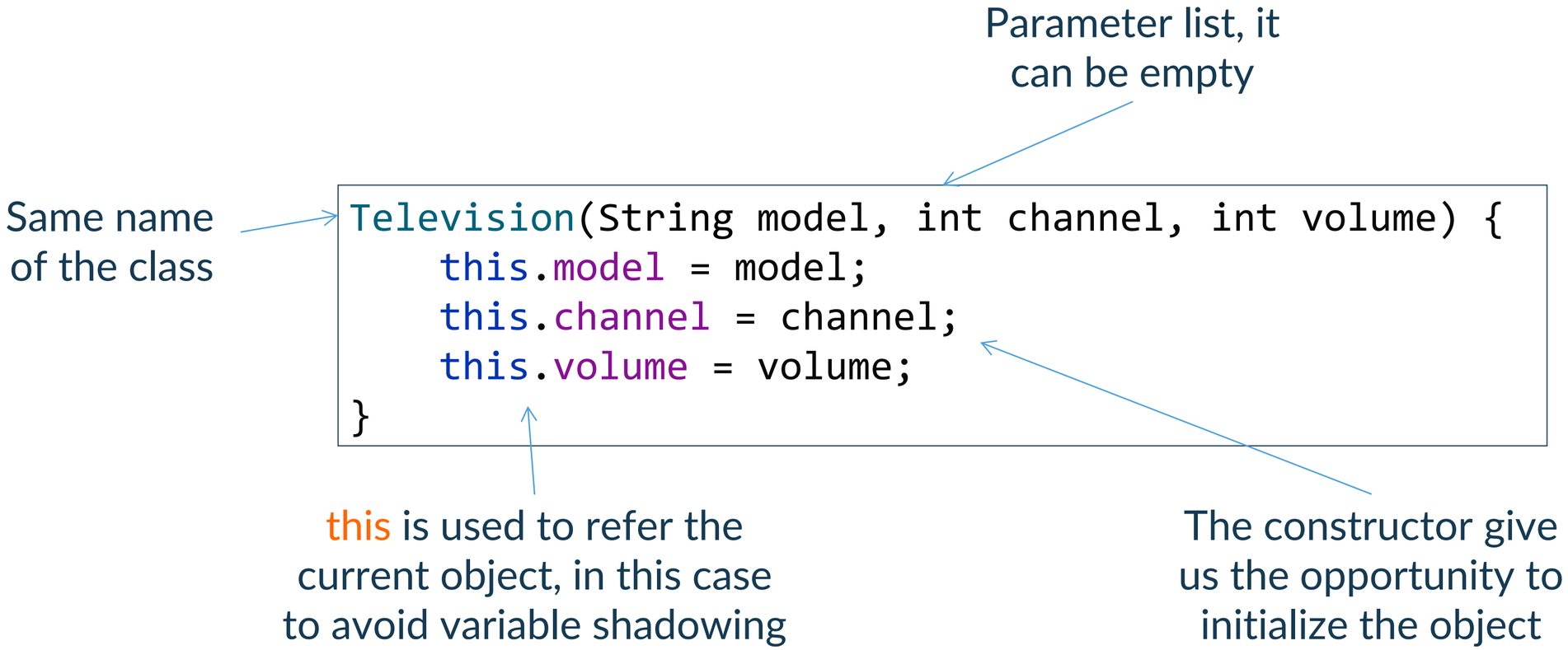
```
Television tv1 = new Television();
Television tv2 = tv1;
tv1.model = "SM2192"; //now both tv1 and tv2
                      //model property is “SM2192”
```

No new object creation nor object copy is involved when we copy one reference from one variable to another variable

Constructors

The creation of an object using the new operator is a two steps operation

1. Java allocates the memory for the object
2. the class constructor is invoked



Default constructor

```
class Television {
    String model;
    boolean on;
    int channel;
    int volume;
}
```

If we don't define a constructor for a class, Java automatically defines a default constructor

The Java default constructor has an empty parameter list

Once we define at least one constructor, the default one is no more available

```
class Television {
    String model;
    boolean on;
    int channel;
    int volume;

    Television(String model) {
        this.model = model;
    }
}
```

We can always override the default constructor

```
class Television {
    String model;
    boolean on;
    int channel;
    int volume;

    Television() {
        this.model = "SY23";
    }
}
```

Constructor overloading

```
class Television {
    String model;
    boolean on;
    int channel;
    int volume;

    Television(String model, int channel, int volume) {
        this.model = model;
        this.channel = channel;
        this.volume = volume;
    }

    Television(String model) {
        this.model = model;
    }
}
```

Java allows constructor overloading, a class can have multiple constructors, given their signatures, i.e., their parameter lists are different

Constructor chaining

It is possible to invoke one constructor from another one, using this(), possibly with an argument list.

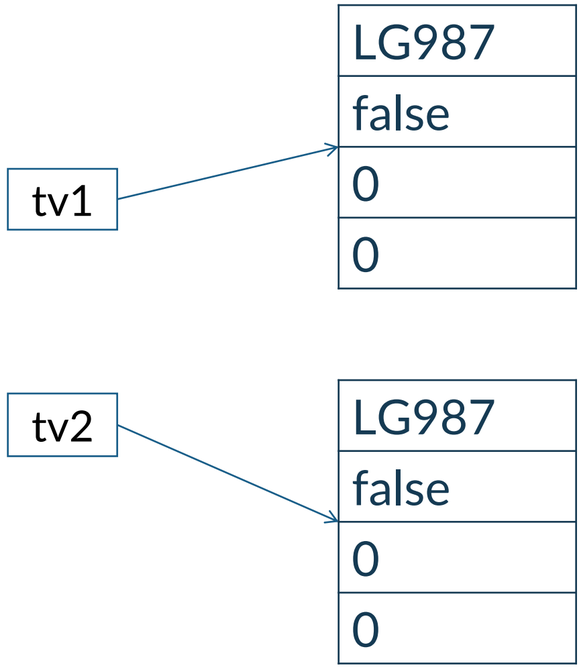
The call to this() must be the first statement within the constructor.

When this() is executed, the overloaded constructor that matches the parameter list is executed first. Then, if there are any statements inside the original constructor, they are executed.

```
class Television {
    String model;
    boolean on;
    int channel;
    int volume;

    Television(String model) {
        this.model = model;
    }

    Television(String model, int channel, int volume) {
        this(model);
        this.channel = channel;
        this.volume = volume;
    }

}
```

Referential equality 1/2

```
public static void main(String[] args) {
    Television tv1 = new Television("LG987");
    Television tv2 = new Television("LG987");

    if (tv1 == tv2) {
        System.out.println("Same");
    } else {
        System.out.println("Different");
    }
}
```

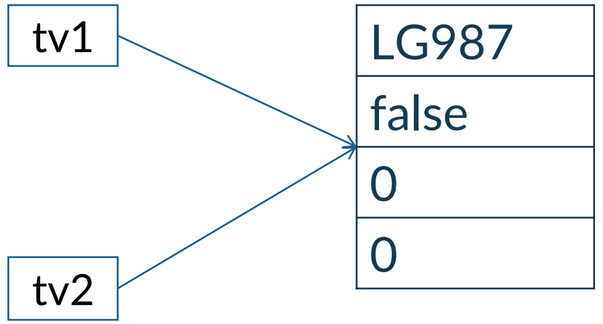



```
$ java Television
Different
```

tv1 and tv2 are references
to different objects

Referential equality 2/2

```
public static void main(String[] args) {
    Television tv1 = new Television("LG987");
    Television tv2 = tv1;

    if (tv1 == tv2) {
        System.out.println("Same");
    } else {
        System.out.println("Different");
    }
}
```




```
$ java Television
Same
```

tv1 and tv2 are references
to the same object

Equivalence of Strings

Equals.java

```
public class Equals {
  public static void main(String[] args) {
        String s1 = "Java is great!";
        String s2 = "Java" + " is great!";
        String a = "Java";
        String b = " is great!";
        String s3 = a + b;

        System.out.println("s1: " + s1);
        System.out.println("s2: " + s2);
        System.out.println("s3: " + s3);

        System.out.println("s1 == s2: " + (s1 == s2));
        System.out.println("s1.equals(s2): " + s1.equals(s2));
        System.out.println("s1 == s3: " + (s1 == s3));
        System.out.println("s1.equals(s3): " + s1.equals(s3));
    }
}
```

Strings are objects not primitive types. In general, to compare objects we use the function equals()

```
$ java Equals.java
s1: Java is great!
s2: Java is great!
s3: Java is great!

s1 == s2: true
s1.equals(s2): true

s1 == s3: false
s1.equals(s3): true
```

Methods

```
class Television {
    String model;
    boolean on;
    int channel;
    int volume;
}
```

The Television class we have defined is almost useless. Why?

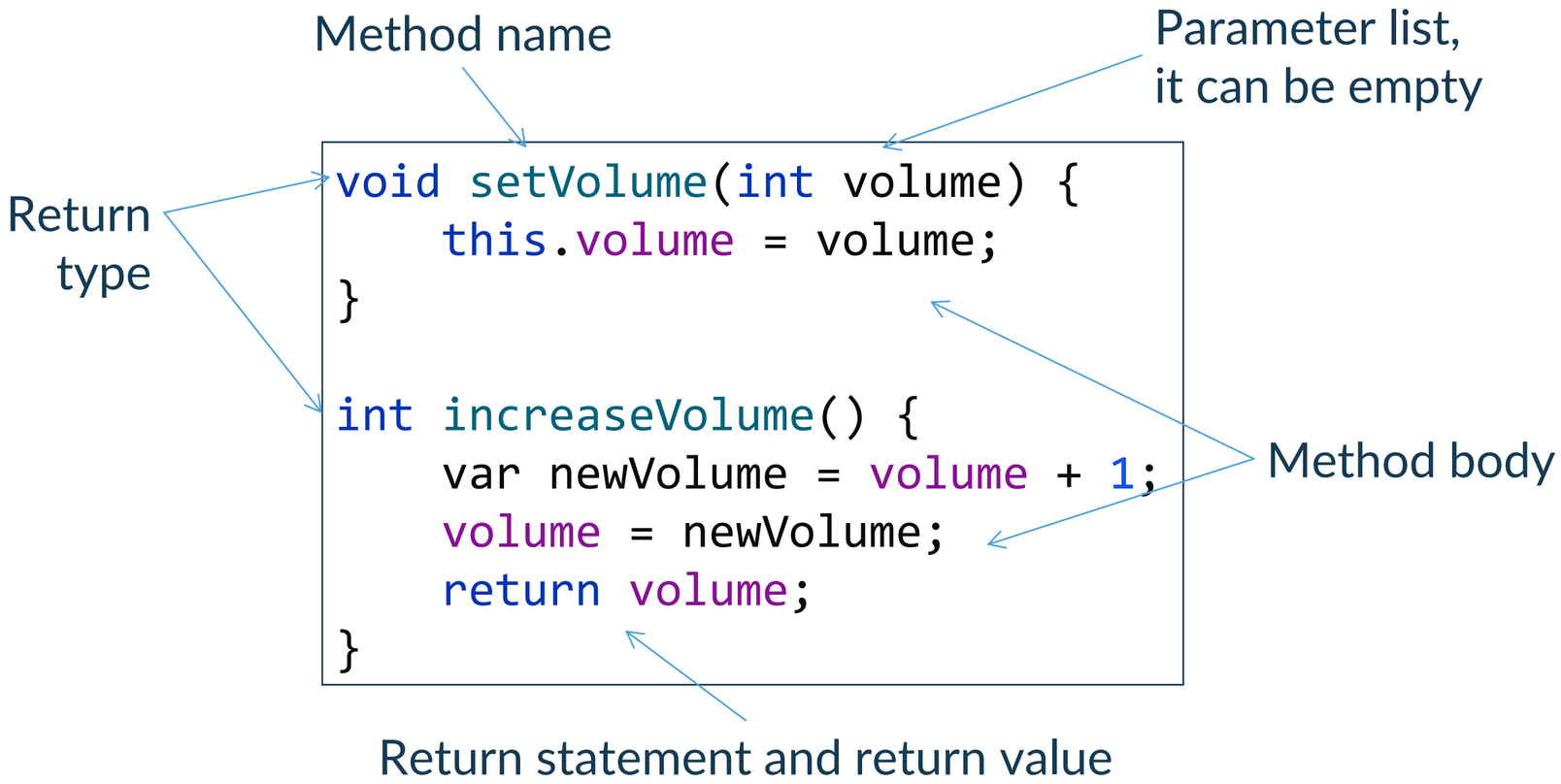
It has no methods, it doesn't expose any behavior, it is just a data object

Data objects need other classes to operate on their own data, this violates encapsulation and reduces code cohesion

Cohesion is the degree to which the elements inside a module belong together

"the code that changes together, stays together"

Method definition



Method invocation

Similar to instance variables, instance methods are invoked using the dot notation

```
Television tv = new Television("LG543");
tv.turnOn();
tv.setChannel(23);
tv.increaseVolume(5);
tv.setChannel(80);
tv.turnOff();
```

To invoke an instance method, you need to instantiate an object of the Television class

Method overloading

Like the constructor case, Java allows method overloading, a class can have multiple methods with the same name, if their parameter lists are different.

```
int increaseVolume() {
    return ++volume;
}

int increaseVolume(int delta) {
    return volume += delta;
}
```

Different parameter list

Different return type

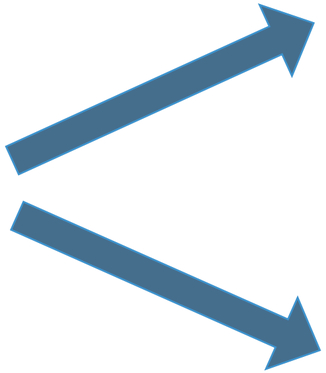
```
int increaseVolume() {
    return ++volume;
}

double increaseVolume() {
    return ++volume;
}
```

The return type cannot be used to differentiate overloaded methods

```
error: method increaseVolume()
is already defined in class
Television
```

The keyword “this”

The keyword this
has two main uses



to return a reference
to the current object

to call constructors
from other
constructors,
constructor chaining

It has also other uses that we’ll discover later

Static members

HelloWorld.java

```
public class HelloWorld {

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

Instance members (variables and methods) come to existence only when we create objects of their classes

On the contrary, when a member is declared static, it is a class member and it can be accessed without any reference to objects of its class

The keyword static must precede the member declaration

We have already met the main method that is declared static because it must be called before any object exists

Static variables

Variables declared as static are a sort of global variables

When objects of its class are declared, no copy of a static variable is made. Instead, all instances of the class share the same static variable

Fields that are both static and final can be used as global constants

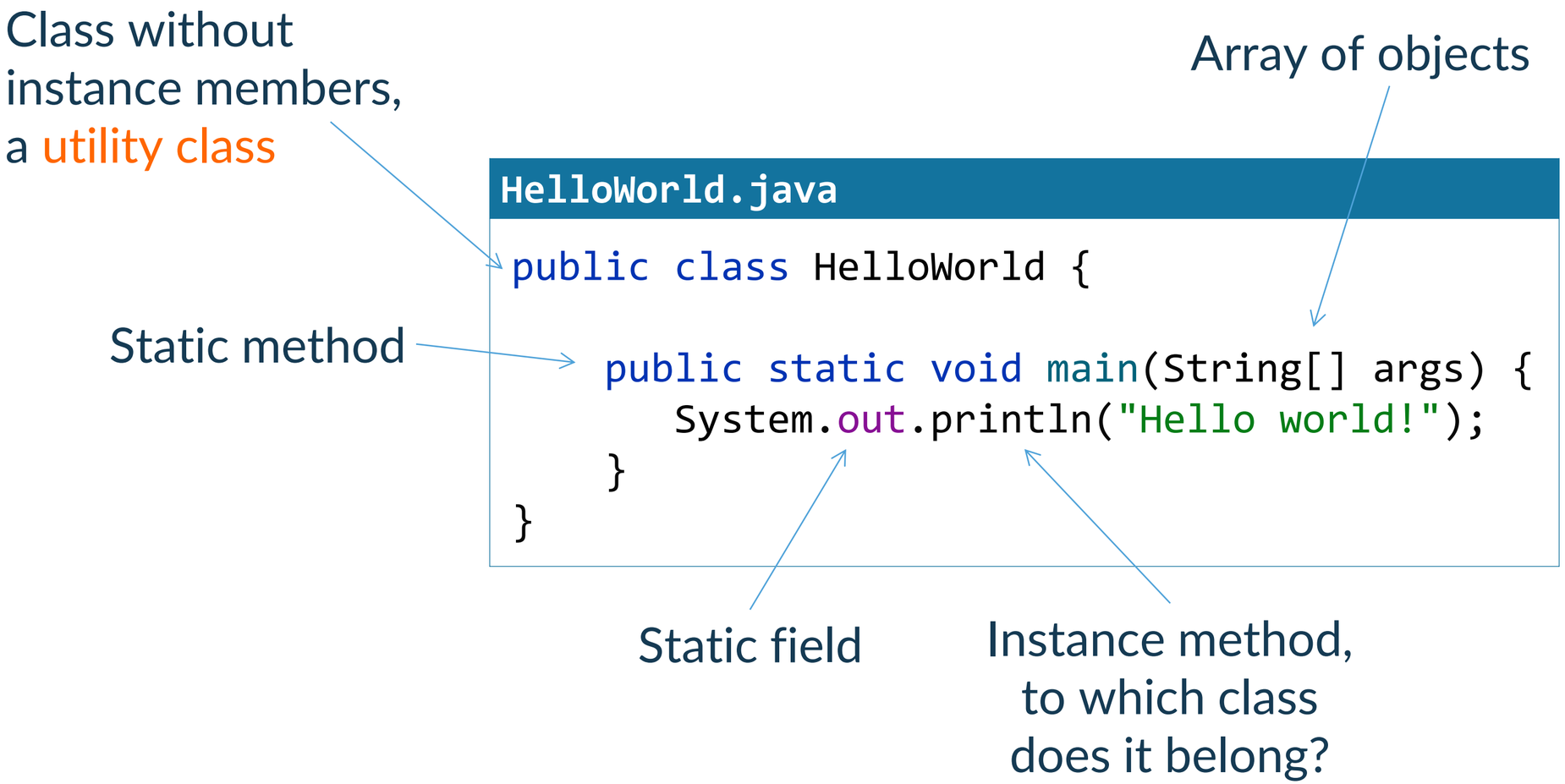
```
class Television {

    static int numberOfTelevisionsTurnedOn;

    String model;
    boolean on;
    int channel, volume;

    Television(String model) {
        this.model = model;
    }

    void turnOn() {
        on = true;
        numberOfTelevisionsTurnedOn++;
    }

    void turnOff() {
        on = false;
        numberOfTelevisionsTurnedOn--;
    }
}
```

Accessing static members

```
class TelevisionMain {
    public static void main(String[] args) {
        Television.setInitialVolume(5);
        System.out.println("Number of televisions turned on: " +
                Television.numberOfTelevisionsTurnedOn);
    }
}
```

Outside of the class in which they are defined, static methods and static variables can be used independently of any object. To do so, you need only to specify the name of their class followed by the dot operator.

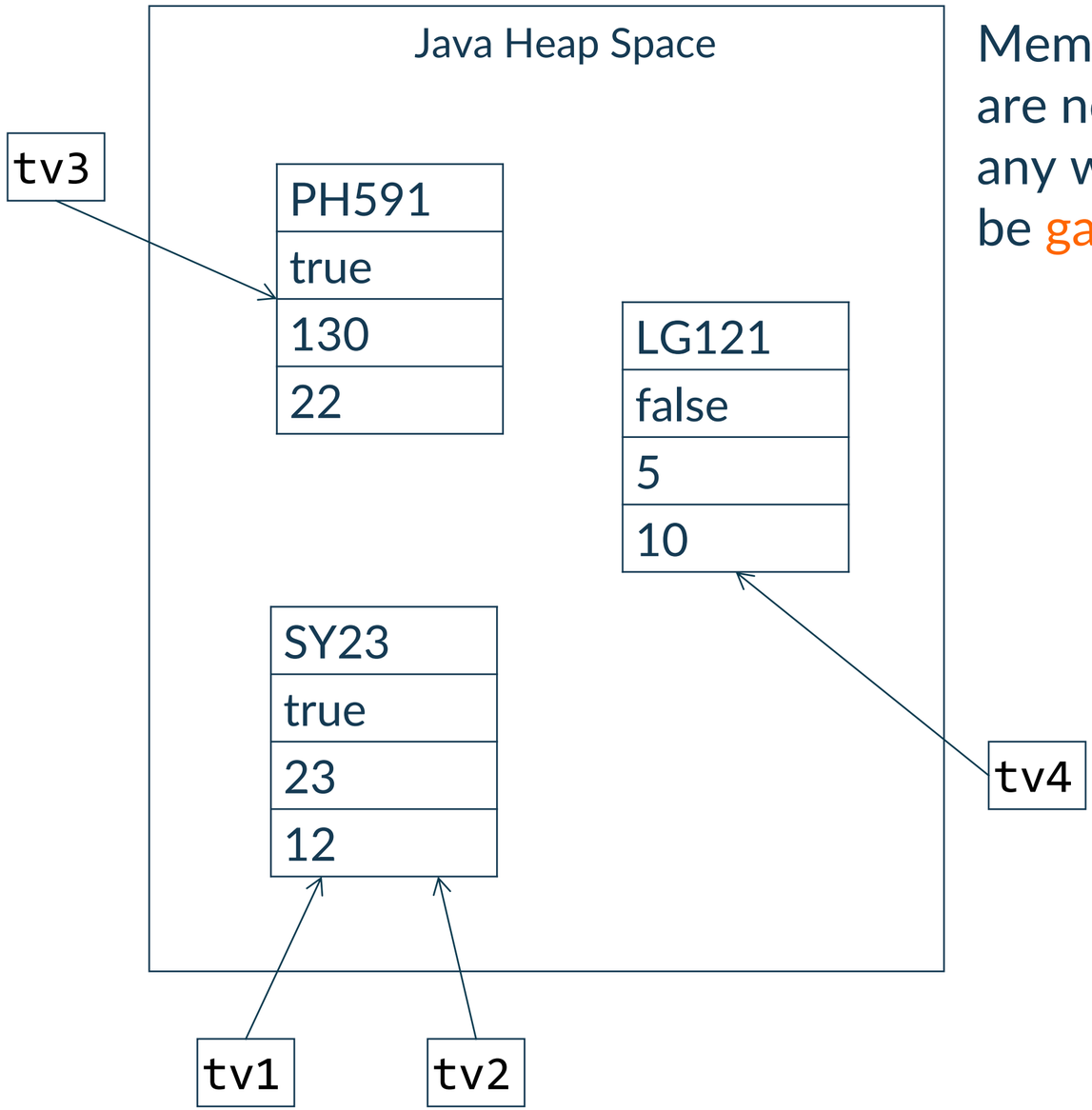
Revisiting Hello, World!



https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/System.html#out

A word about garbage collection

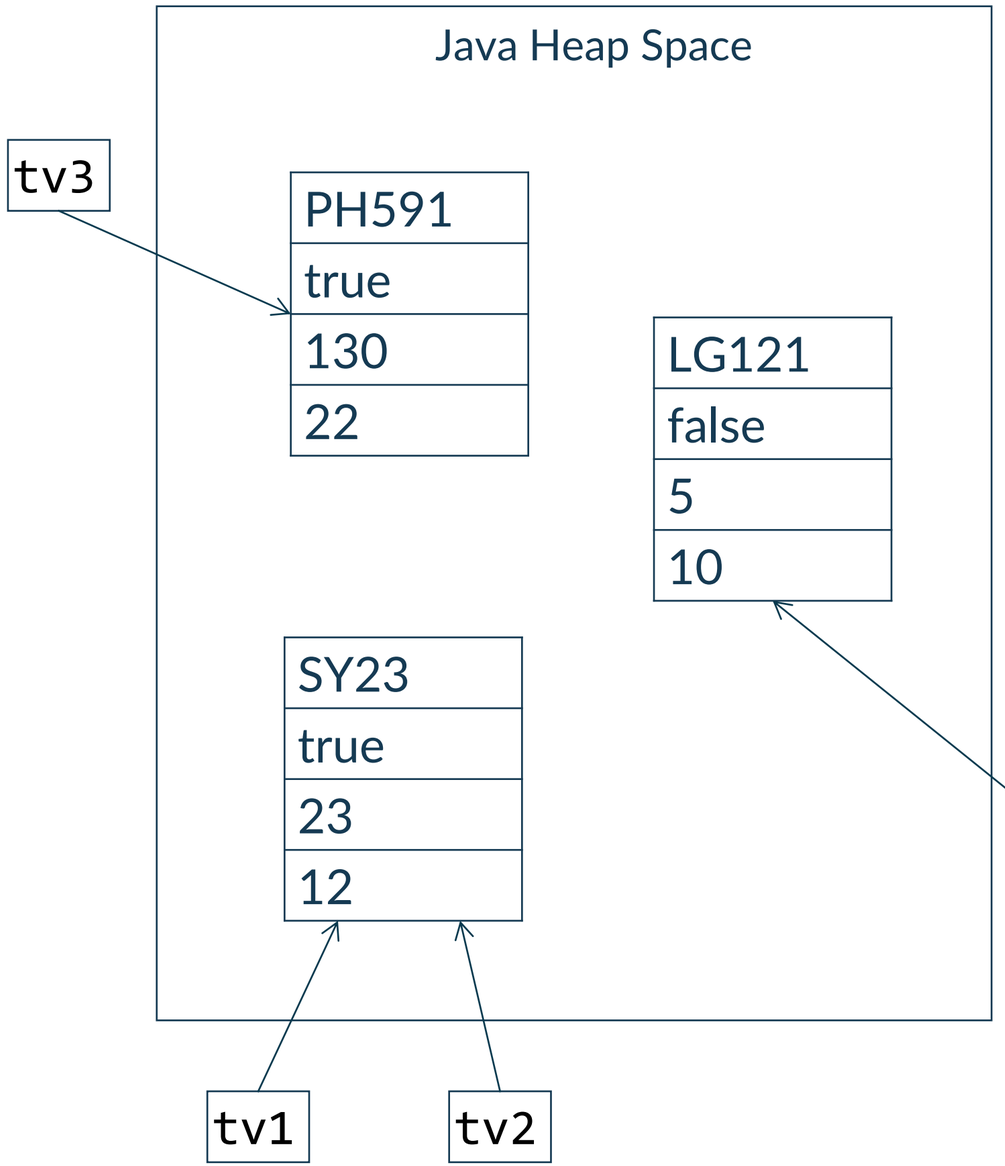
```
var tv1 = new Television();
tv1.model = "SY23"
tv1.on = true;
Television tv2 = tv1;
tv2.channel = 23;
tv2.volume = 12;

var tv3 = new Television();
tv3.model = "PH591"
tv3.on = true;
tv3.channel = 130;
tv3.volume = 22;

var tv4 = new Television();
tv4.model = "LG121"
tv4.on = false;
tv4.channel = 5;
tv4.volume = 10;
```



Memory locations that are not referenced in any way are eligible to be garbage collected

A word about garbage collection

```
var tv1 = new Television();
tv1.model = "SY23"
tv1.on = true;
Television tv2 = tv1;
tv2.channel = 23;
tv2.volume = 12;

var tv3 = new Television();
tv3.model = "PH591"
tv3.on = true;
tv3.channel = 130;
tv3.volume = 22;

var tv4 = new Television();
tv4.model = "LG121"
tv4.on = false;
tv4.channel = 5;
tv4.volume = 10;

tv4 = null;
```



Memory locations that are not referenced in any way are eligible to be garbage collected

The memory of garbage collected objects is freed and made again available in the Heap Space

Packages and libraries

Packages

esteco.Television

```
package esteco;

class Television {
    String model;
    boolean on;
    int channel;
    int volume;

    Television(String model) {
        this.model = model;
    }
}
```

units.Television

```
package units;

class Television {
    String model;
    boolean on;
    int channel;
    int volume;

    Television(String model) {
        this.model = model;
    }
}
```

Java classes can be organized in different
namespaces by defining different packages

Hierarchical packages

com.esteco.sdm.Television

```java
package com.esteco.sdm;

public class Television {
    String model;
    boolean on;
    int channel;
    int volume;

    public Television(String model) {
        this.model = model;
    }
}
```

it.units.sdm.Television

```java
package it.units.sdm;

public class Television {
    String model;
    boolean on;
    int channel;
    int volume;

    public Television(String model) {
        this.model = model;
    }
}
```

Packages can be organized in hierarchies.
Each package is separate from the parent using the dot notation

The package hierarchy must be reflected in the file system

The fully-qualified name

HelloTelevision.java

```
class HelloTelevision {

    public static void main(String args[]) {
        com.esteco.sdm.Television tv1 = new com.esteco.sdm.Television("LG121");
        it.units.sdm.Television tv2 = new it.units.sdm.Television("LG121");

        System.out.println("Hello tv1: " + tv1.getClass().getName());
        System.out.println("Hello tv2: " + tv2.getClass().getName());
    }
}
```

fully-qualified name = package name + '.' + simple name

To reference classes in other packages they must be declared as public. The constructor must be declared as public, too!

The import declaration

HelloTelevision.java

```
import com.esteco.sdm.Television;

class HelloTelevision {

    public static void main(String args[]) {
        Television tv1 = new Television("LG121");
        it.units.sdm.Television tv2 = new it.units.sdm.Television("LG121");

        System.out.println("Hello tv1: " + tv1.getClass().getName());
        System.out.println("Hello tv2: " + tv2.getClass().getName());
    }
}
```

Each class can be referenced by using its simple name or the fully-qualified name. The simple name can be used

1. when the referenced class is in the same package of the current class
2. when the referenced class has been imported by using the import declaration

The `java.lang` package

Classes in the java.lang package are imported by default

Notable classes in the java.lang package

```
Class
Exception
Math
Object
Process/ProcessBuilder
Runnable
Runtime
String/StringBuilder
System
Thread
```

The Java library – java.base module

Package	Description
java.io	I/O through data streams
java.lang	Fundamental classes & interaction with environment
java.math	Arbitrary-precision arithmetic
java.net/javax.net	Networking applications
java.nio	New I/O
java.text	Handling of text, date, numbers, and messages
java.time	API for times, dates, instances, and durations
java.util.concurrent	Classes for concurrency
java.util.function	Classes for functional programming
java.util.stream	Classes for streams
java.util.zip	Reading and writing ZIP and GZIP files

Using external libraries

- External libraries are organized in the so-called Jar-files
- Jar-files are collections of classes
- You can imagine a Jar file like a ZIP file with some metadata
- The classpath property is used by java to specify the places where to search for class files
- The classpath can be defined on the command line of java and javac, by using the –cp option
  - or in the environment variable CLASSPATH (not recommended)

Example JSON and JSON-java

https://www.json.org/json-en.html

Using JSONObject

JsonTest

```
import org.json.JSONObject;

public class JsonTest {
     public static void main(String[] args) {
          JSONObject jsonObject = new JSONObject();
          jsonObject.put("pi", 3.14);
          jsonObject.put("a", new int[] {1, 2, 3});
          System.out.println(jsonObject.toString());
     }
}
```

```
>javac -cp lib\json-20220320.jar JsonTest.java
>java -cp .;lib\json-20220320.jar JsonTest
{"a":[1,2,3],"pi":3.14}
```

Assignments

Assignment 2

Implement a Calculator class to perform arithmetic operations.

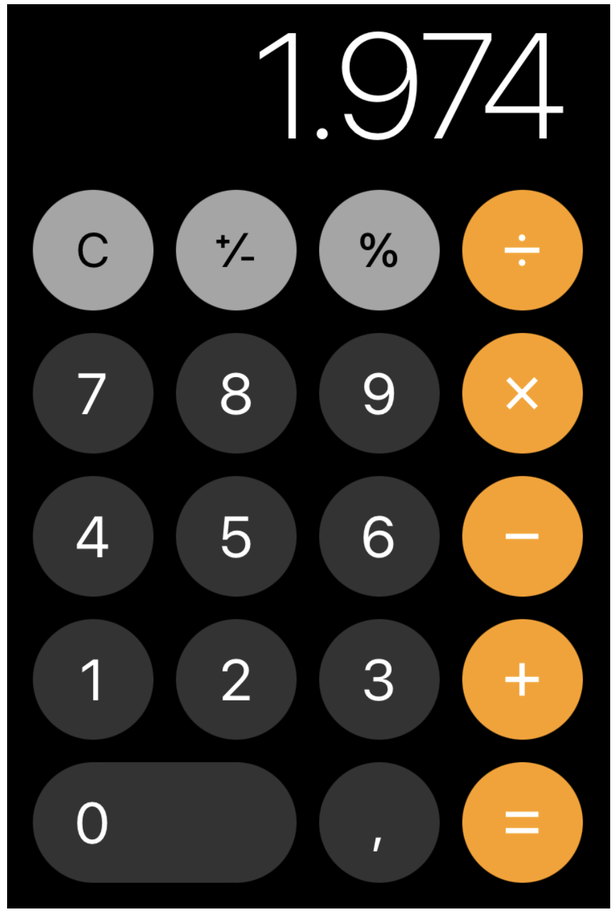
```
$ java Calculator 6 + 4.1
10.1
$ java Calculator 3.6 / -2
-1.8
$ java Calculator 8.5 * 9
76.5
$ java Calculator -3.14
-3.14
```

I let you discover how to convert strings to numbers

Enhance the calculator so that it can handle concatenated operations

```
$ java Calculator 6 + 4.1 * 3
10.1
30.3
$ java Calculator 3.6 / 2 + -0.3 / .5
1.8
1.5
3
```

Assignment 3



Define a calculator class that

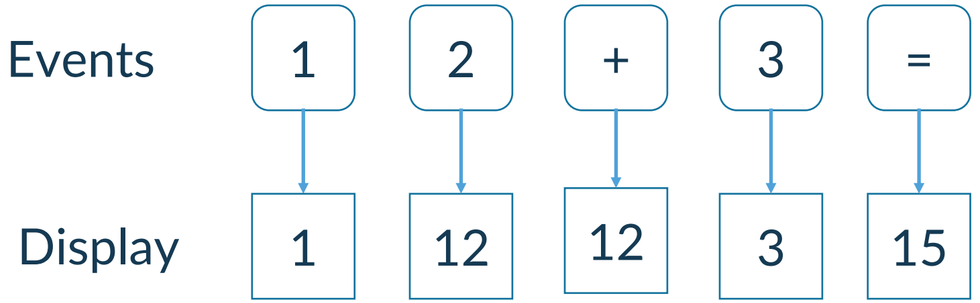
1. receives “events” from a calculator keyboard
2. sends the output to a Display object

```
class Display {
    void display(String text) {
        System.out.println(text);
    }
}
```

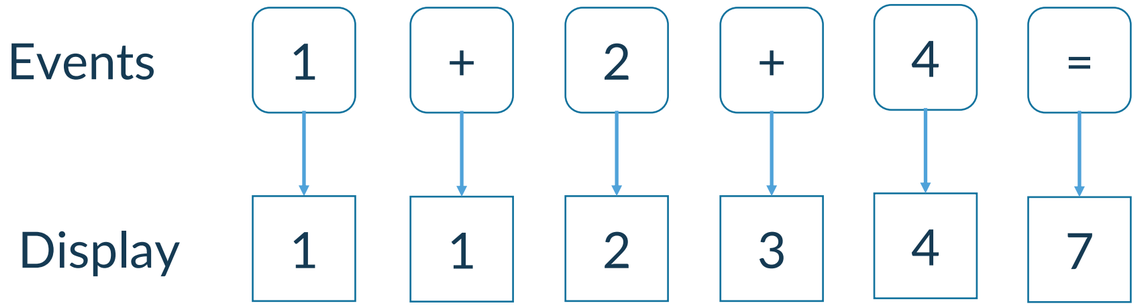
```
class Calculator {

    final Display display;
    //...

    Calculator(Display display) {
        this.display = display;
    }

    void plusPressed() {
        //...
    }

    void zeroPressed() {
        //...
    }

    //...
}
```

A couple of scenarios

Sample scenario



Operations chaining



Assignment 4

Implement a Java program to run scripts in a language of your choice , e.g., Python or JavaScript.

RunScript

```
public class RunScript {
    public static void main(String[] args) {
        PythonInterpreter interpreter = new PythonInterpreter();
        interpreter.runScript("print('Hello, World!')");
    }
}

class PythonInterpreter {
    void runScript(String script) {
      …
    }
}
```

Thank you!

esteco.com