



# Programming in Java

## Introduction



Carlos Kavka  
Head of Research and Development

# Course structure



## **Introduction to the language**

Covering most aspects of the language till last release

## **Many examples**

Most concepts will be explained through examples

## **Emphasis on functional programming**

One of the most exciting features of the language

# Recommended online reference

The screenshot shows a web browser window displaying the Oracle Java SE 13 & JDK 13 API Specification Overview page. The browser's address bar shows the URL `docs.oracle.com/en/java/javase/13/docs/api/index.html`. The page has a dark blue header with navigation links: OVERVIEW, MODULE, PACKAGE, CLASS, USE, TREE, DEPRECATED, INDEX, and HELP. The current page is 'OVERVIEW'. On the right side of the header, it says 'Java SE 13 & JDK 13'. Below the header is a search bar with the text 'SEARCH: Search'. The main content area has the title 'Java® Platform, Standard Edition & Java Development Kit Version 13 API Specification'. Below the title, it states: 'This document is divided into two sections: Java SE. The Java Platform, Standard Edition (Java SE) APIs define the core Java platform for general-purpose computing. These APIs are in modules whose names start with java. JDK. The Java Development Kit (JDK) APIs are specific to the JDK and will not necessarily be available in all implementations of the Java SE Platform. These APIs are in modules whose names start with jdk.' Below this text is a 'Window Snip' button. At the bottom, there is a table with four tabs: 'All Modules', 'Java SE', 'JDK', and 'Other Modules'. The 'All Modules' tab is selected. The table has two columns: 'Module' and 'Description'. The table lists the following modules and their descriptions:

Module	Description
<code>java.base</code>	Defines the foundational APIs of the Java SE Platform.
<code>java.compiler</code>	Defines the Language Model, Annotation Processing, and Java Compiler APIs.
<code>java.datatransfer</code>	Defines the API for transferring data between and within applications.
<code>java.desktop</code>	Defines the AWT and Swing user interface toolkits, plus APIs for accessibility, audio, imaging, printing, and JavaBeans.
<code>java.instrument</code>	Defines services that allow agents to instrument programs running on the JVM.
<code>java.logging</code>	Defines the Java Logging API.
<code>java.management</code>	Defines the Java Management Extensions (JMX) API.
<code>java.management.rmi</code>	Defines the RMI connector for the Java Management Extensions (JMX) Remote API.
<code>java.naming</code>	Defines the Java Naming and Directory Interface (JNDI) API.
<code>java.net.http</code>	Defines the HTTP Client and WebSocket APIs.
<code>java.prefs</code>	Defines the Preferences API.





# Recommended development tool



```
intellij-community [~/intellij-community] - .../platform/core-api/src/com/intellij/lang/folding/LanguageFolding.java [intellij.platform.core]
intellij-community platform core-api src com intellij lang folding LanguageFolding IDEA
Project core-api [intellij.platform.core]
  src
    com.intellij
      codeInsight
      concurrency
      core
      diagnostic
      formatting
      ide
      injected.editor
      lang
        CompositeFoldingBuilder
        CustomFoldingBuilder
        CustomFoldingProvider
        FoldingBuilder
        FoldingBuilderEx
        FoldingDescriptor
        LanguageFolding
      injection
        ASTNode
        CodeDocumentationAwareCo
        CodeDocumentationAwareCo
        Commenter
        CompositeLanguage
        CustomUncommenter
        DependentLanguage
        FCTSBackedLighterAST
        FileASTNode
        InjectableLanguage
        ITokenTypeRemapper
        Language
49  /**
50  * Only queries base language results if there are no extensions for originally requested
51  */
52  @NotNull
53  @Override
54  public List<FoldingBuilder> allForLanguage(@NotNull Language language) {
55  for (Language l = language; l != null; l = l.getBaseLanguage()) {
56  List<FoldingBuilder> extensions = forKey(l);
57  if (!extensions.isEmpty()) {
58  return extensions;
59  }
60  }
61  return Collections.emptyList();
62  }
63
64  @NotNull
65  public static FoldingDescriptor[] buildFoldingDescr
66  if (!DumbService.isDumbAware(builder) && DumbServ
67  return FoldingDescriptor.EMPTY;
68  }
69
70  if (builder instanceof FoldingBuilderEx) {
71  return ((FoldingBuilderEx)builder).buildFoldReg
72  }
73  final ASTNode astNode = root.getNode();
74  if (astNode == null || builder == null) {
75  return FoldingDescriptor.EMPTY;
76  }
77
78  return
79  }
80  }
81  builder.buildFoldRegions(ASTNode node, Document document) FoldingDescriptor[]
82  FoldingDescriptor.EMPTY (com.intellij.lang... FoldingDescriptor[]
83  Dot, space and some other keys will also close this lookup and be inserted into editor >>
```

Version Control

Local Changes Log

Commit Message	Author	Date
use psiTraverser() in CheckPsiReadAccessors	Daniil Ovchinnikov	21.08.18, 15:23
remove unused code	Dmitry Batkovich	21.08.18, 15:03
<a href="#">IDEA-59397</a> When caret is outside visible editor area, on Alt-En	Dmitry Batrak	21.08.18, 15:01
[gui-test] replace system properties with env variables	Vladislav Shishov	21.08.18, 14:12
add a test case for java same parameter value inspection	Dmitry Batkovich	21.08.18, 14:16
[groovy] parse new lines in type parameter list ( <a href="#">IDEA-197524</a> )	Daniil Ovchinnikov	21.08.18, 14:09
cleanup for code review <a href="#">IDEA-CR-36249</a>	Nikita Skvortsov	21.08.18, 14:01
<b>PyCharm 2018.1.5</b>	<b>origin/181.5540 Aleksey Rostovskiy</b>	<b>21.08.18, 14:06</b>
check for sa pid attach availability - <a href="#">IDEA-168185</a>	Egor Ushakov	21.08.18, 14:01
[groovy] wrap PsiFile#isValid check into a read action ( <a href="#">IDEA</a> )	Daniil Ovchinnikov*	20.08.18, 21:05
cleanup	Alexey Kudravnsev	21.08.18, 12:28

intellij.pycharm.community.resources 1 file  
resources/idea 1 file  
PyCharmCoreApplicationInfo.xml

PyCharm 2018.1.5

8935cc3 Aleksey Rostovskiy  
<aleksey.rostovskiy@jetbrains.com> on  
21.08.18 at 14:06

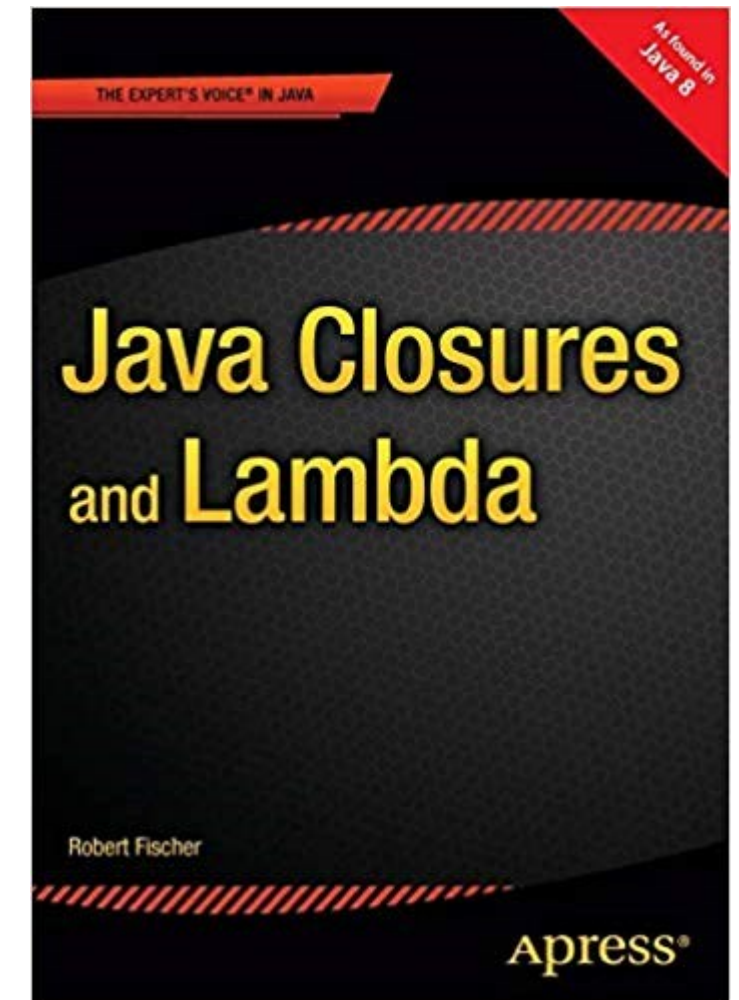
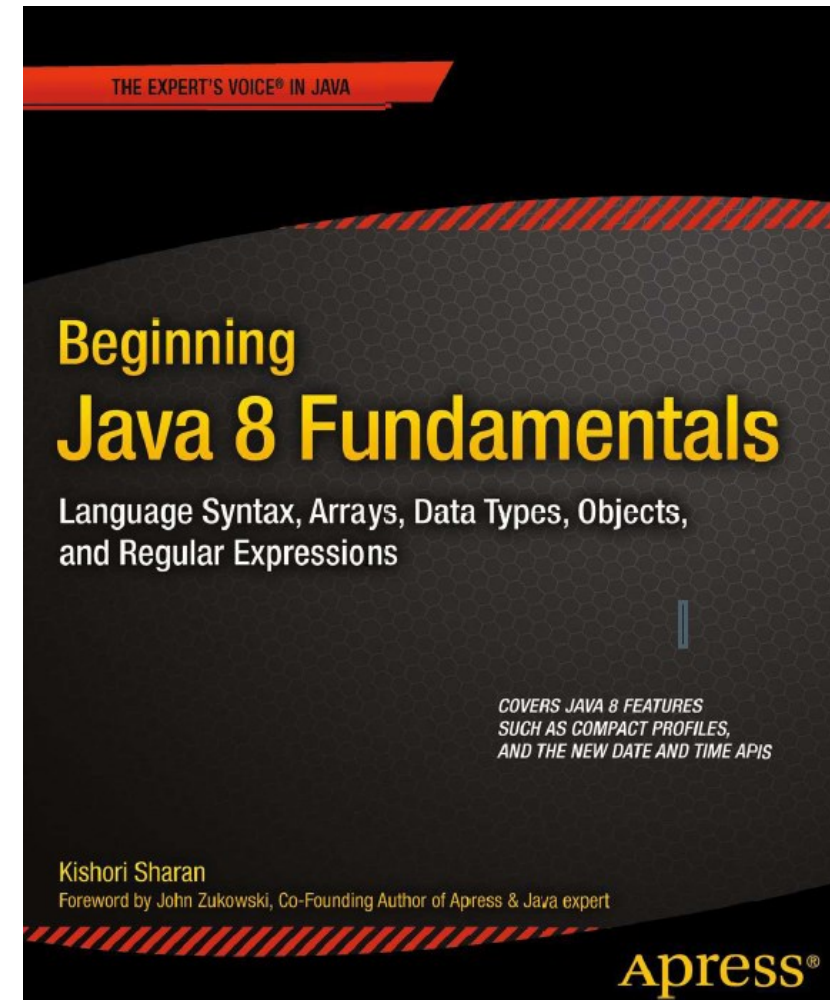
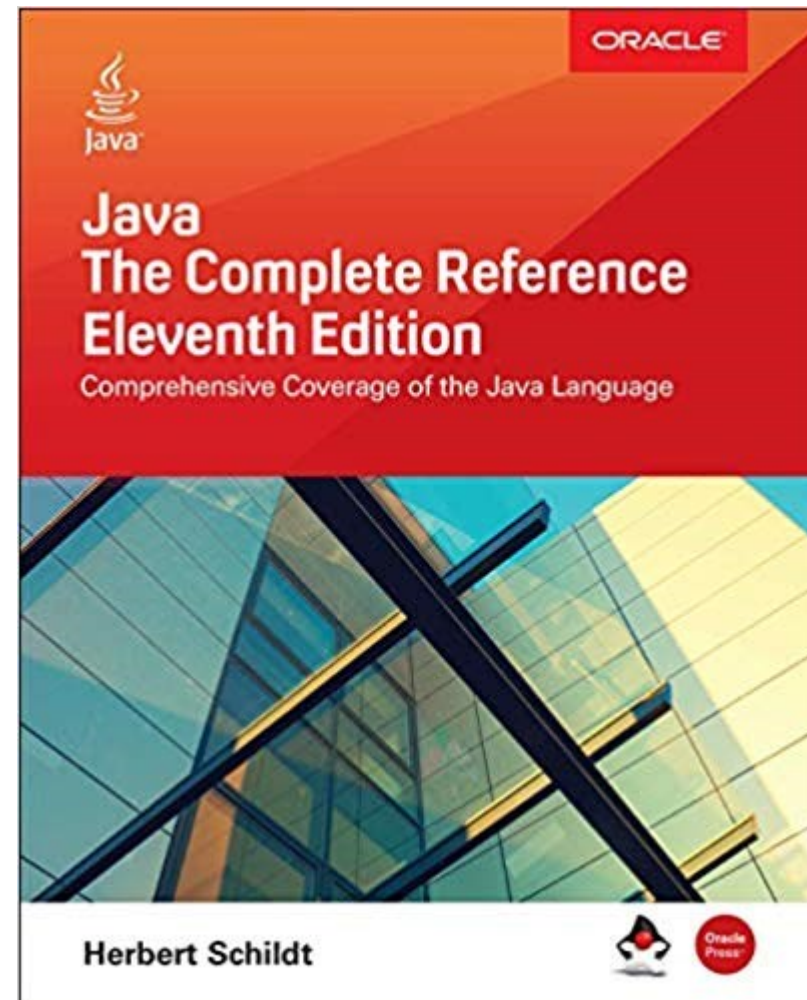
origin/181.5540  
pycharm/181.5540.17

In 1 branch: origin/181.5540





# Recommended books



Many good books available!





# Programming in Java

## Part I - basic concepts



Carlos Kavka  
Head of Research and Development



# Agenda



**A bit of history**

---

**Basic types**

---

**Basic expressions**

---

**Control structures**

---

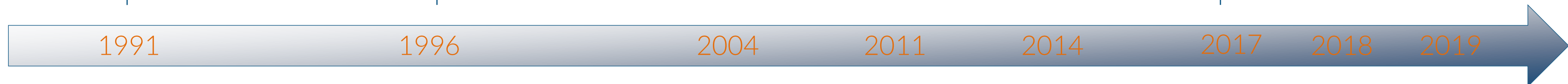
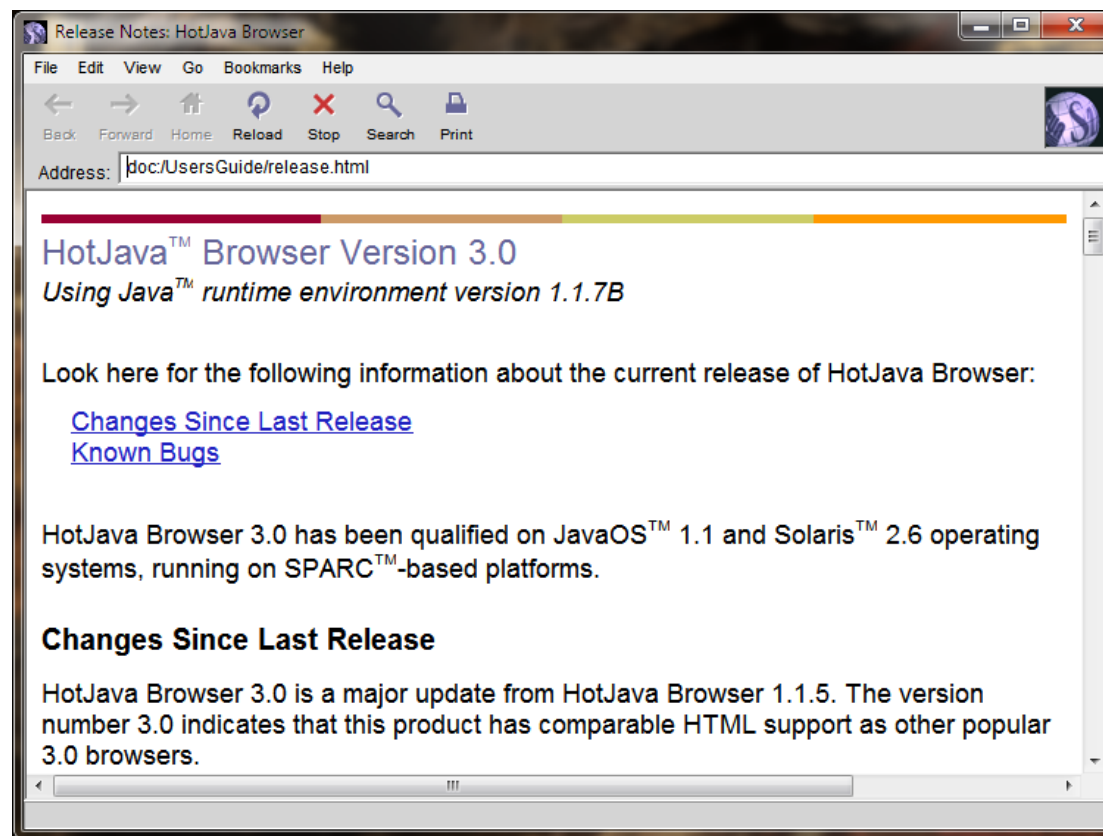
**Arrays**

# A bit of history



\*7

HotJava browser



1991

1996

2004

2011

2014

2017

2018

2019

Oak

Java 1.0

Java 5

Java 7

Java 8

Java 9

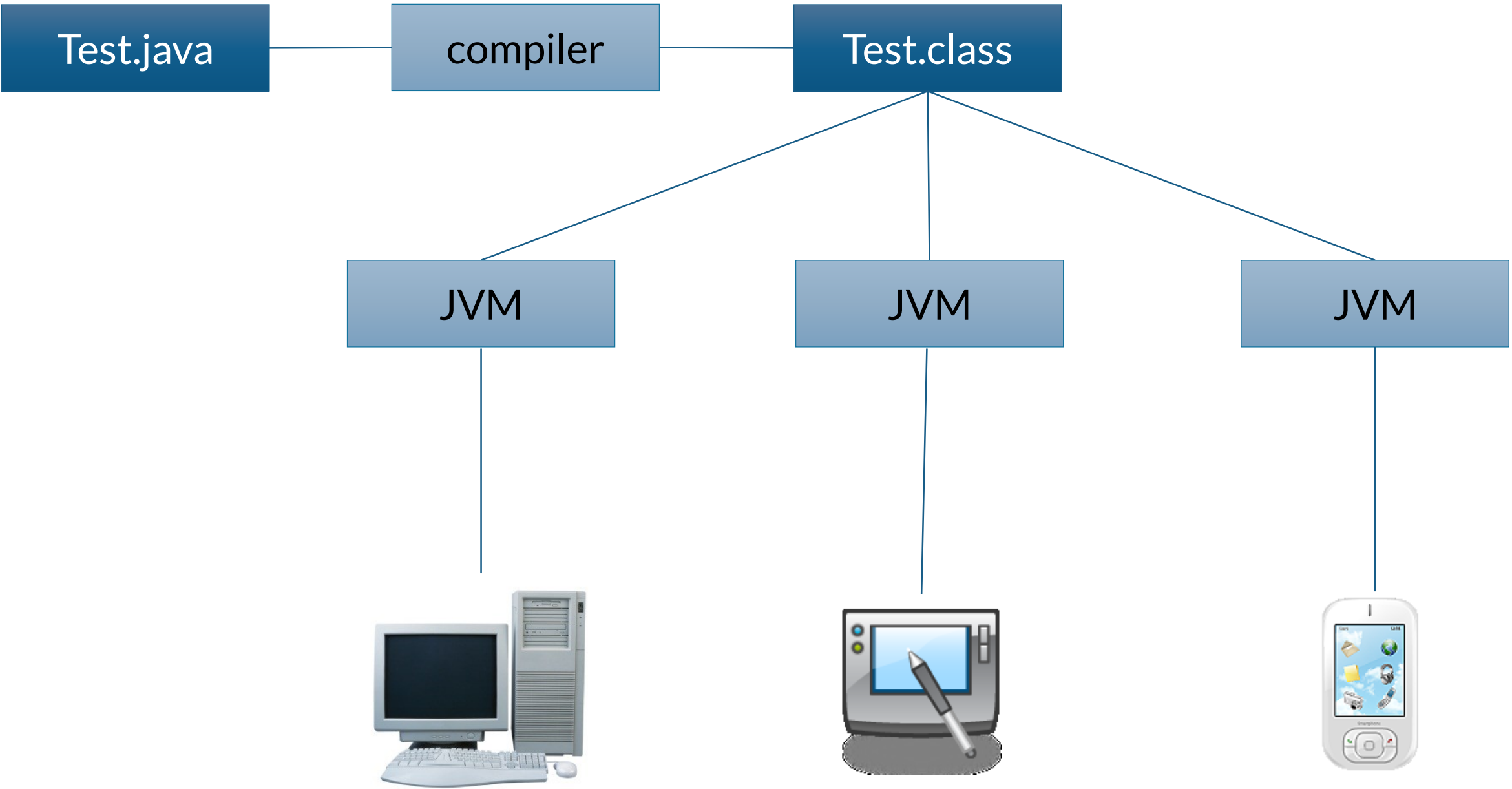
Java 10/11

Java 12/13





# The Java platform



The compiled code is **independent** of the architecture of the computer



# A first example

HelloWorld application 

```
/**
 * * Hello World Application
 */
public class HelloWorld {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // display output
    }
}
```

```
$ javac HelloWorld.java
```

```
$ ls
HelloWorld.class
HelloWorld.java
```

```
$ java HelloWorld
Hello World
```

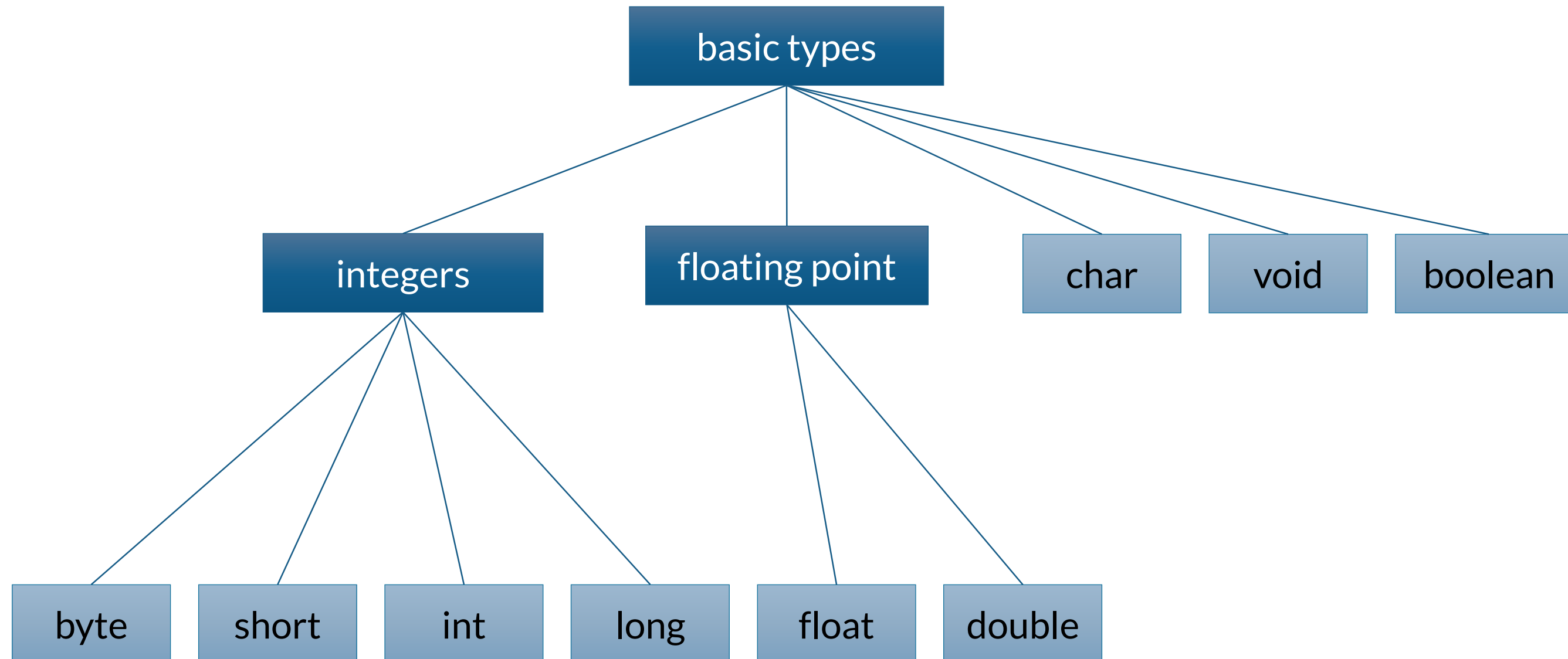
Only single source applications can be executed directly:

```
$ java HelloWorld.java
Hello World
```



# Basic types

Java provides the following primitive **types**





# Variables and constants definition

```
int x;  
double d = 0.33;  
float f = 0.22F;  
char c = 'a';  
boolean ready = true;  
  
x = 15;
```

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

**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**



# Strings

Strings are not a basic type, but defined as a class, more details later!

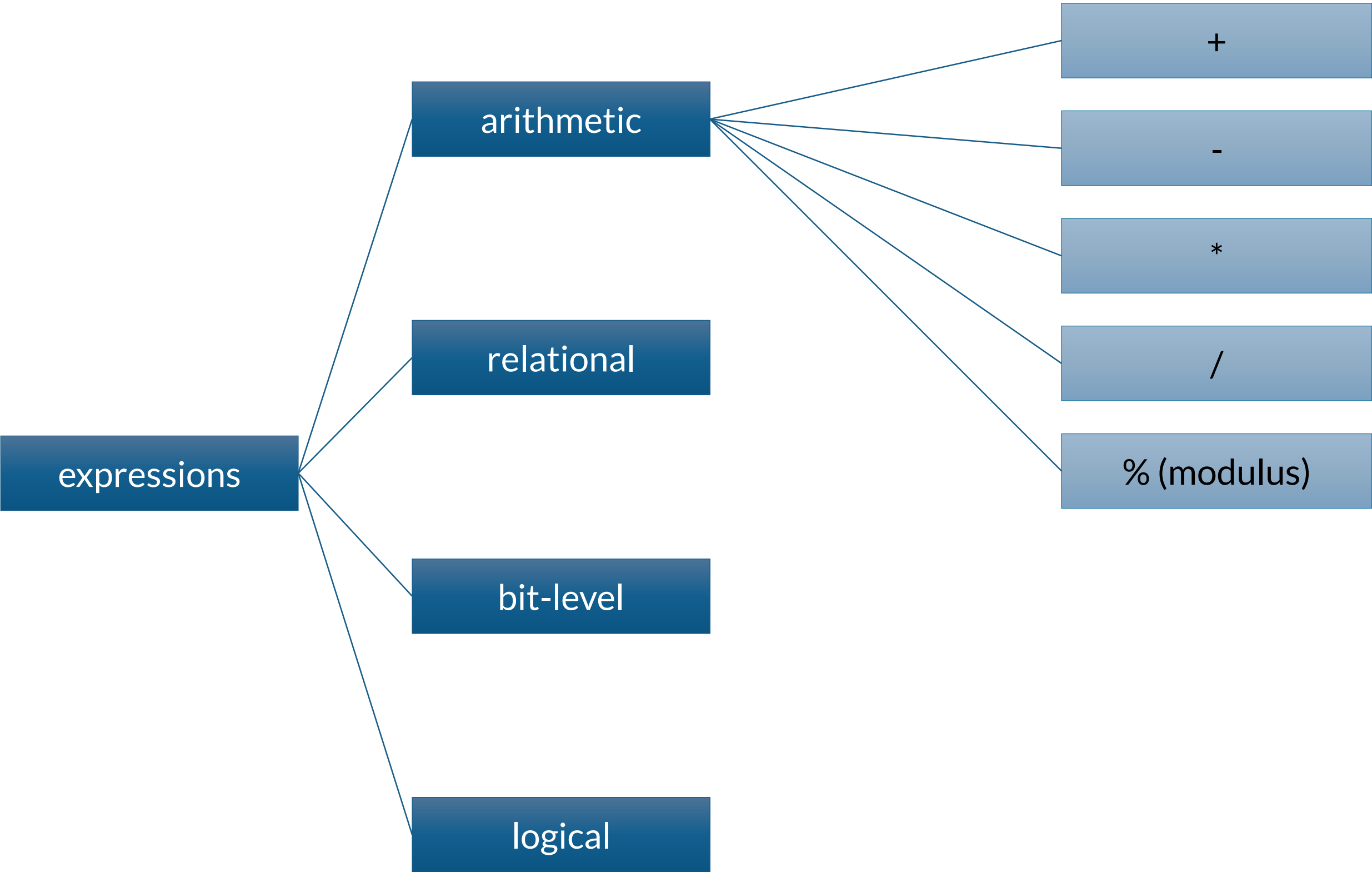
```
String a = "abc";
```

If the expression begins with a string and uses the + operator, then the next argument is **converted** to a string

```
int cost = 22;  
String b = "the cost is " + cost + " euro";
```



# Arithmetic expressions





# Example with arithmetic operators

Arithmetic class 

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

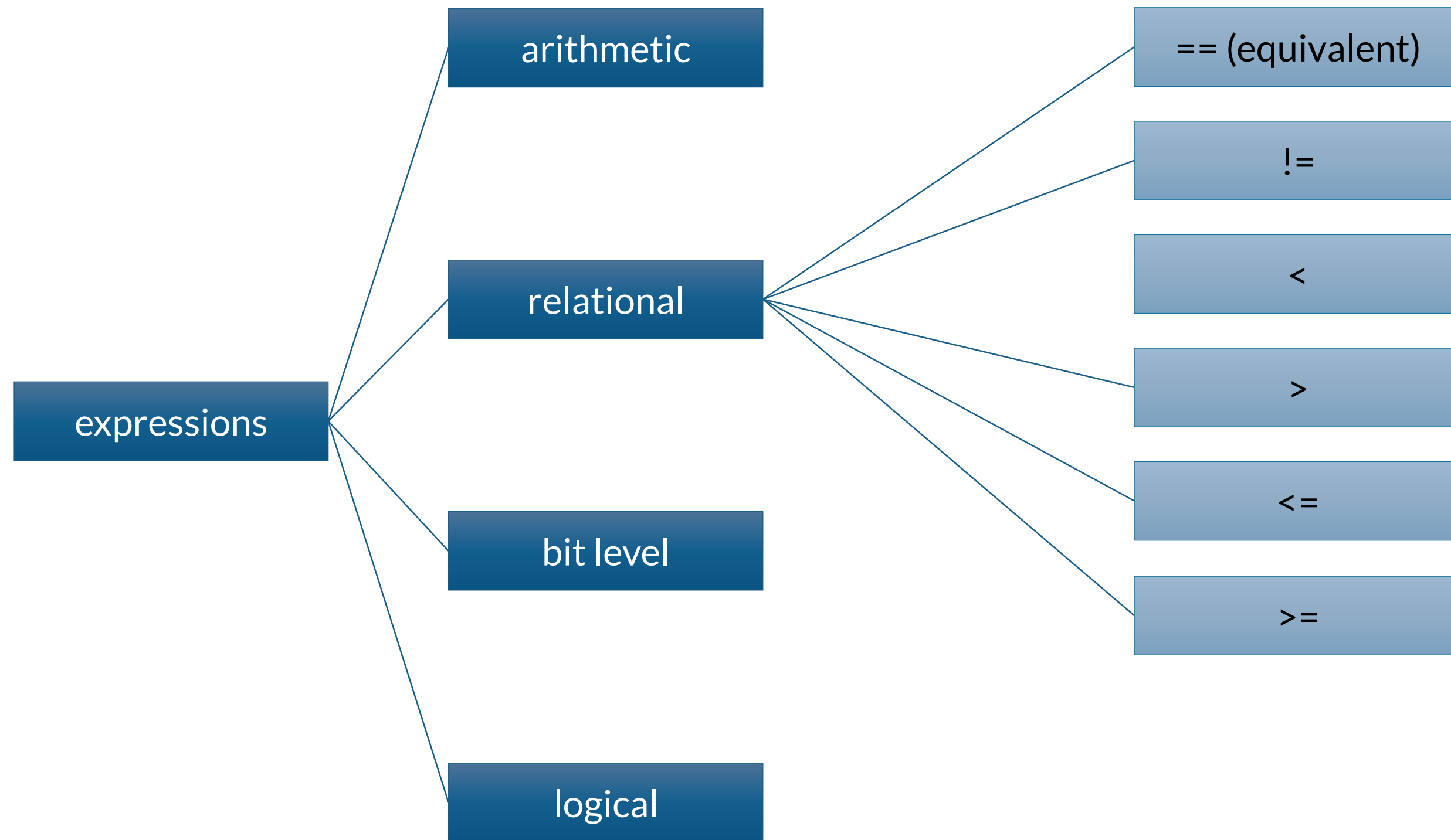
    int a = 12,b = 12;
    System.out.print(a++); // printed and then incremented
    System.out.print(a);

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

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



# Relational expressions



# Example with relational operators

Boolean class 

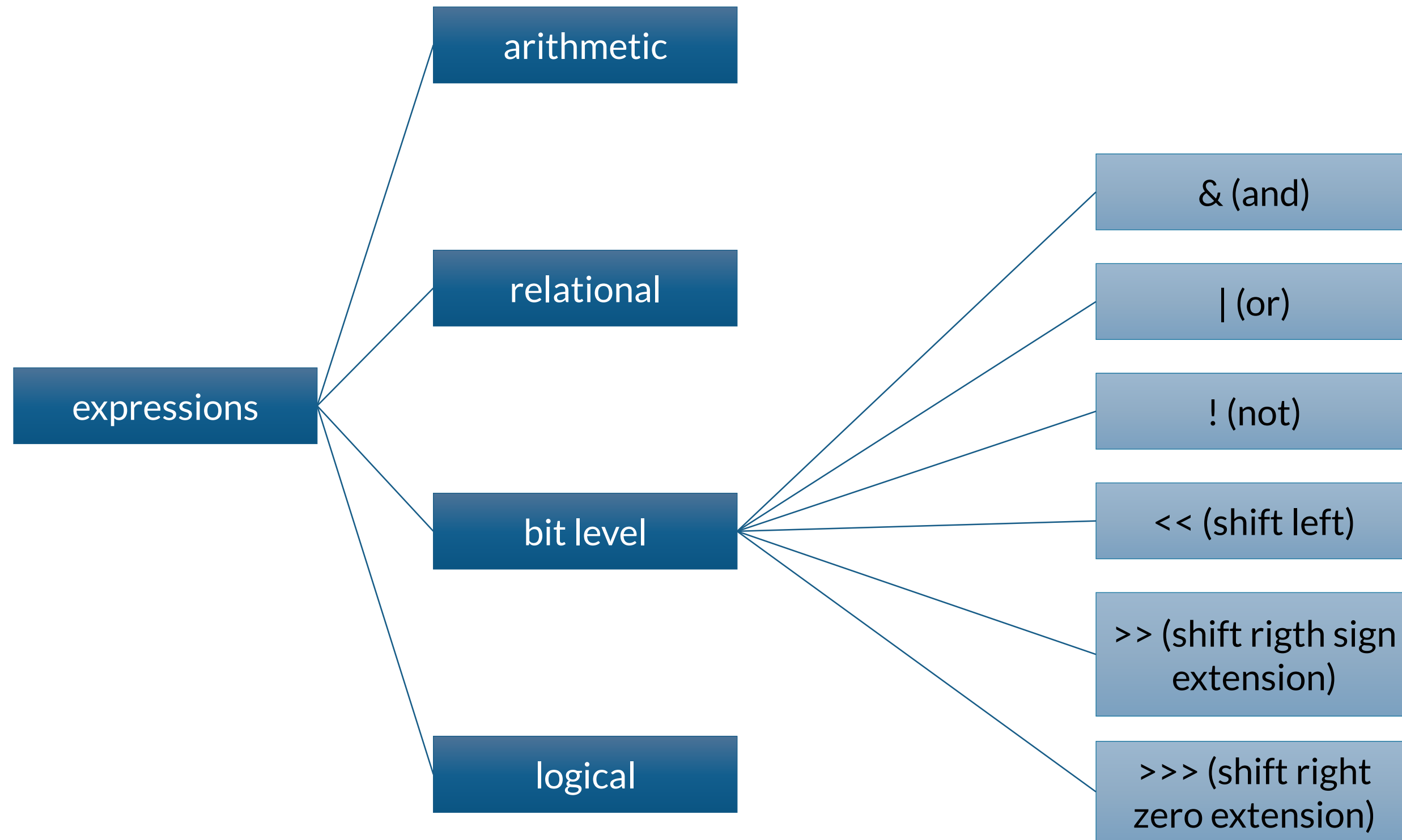
```
public class Boolean {  
    public static void main(String[] args) {  
        int x = 12,y = 33;  
  
        System.out.println(x < y);  
        System.out.println(x != y - 21);  
  
        boolean test = x >= 10;  
        System.out.println(test);  
    }  
}
```

```
$ java Boolean  
true  
false  
true
```





# Bit level expressions



# Example with bit-level operators

Boolean class 

```
public class Bits {
    public static void main(String[] args) {
        int x = 0b000000000000000000000000000000000000000010110;
        int y = 0b0000000000000000000000000000000000000000110011;

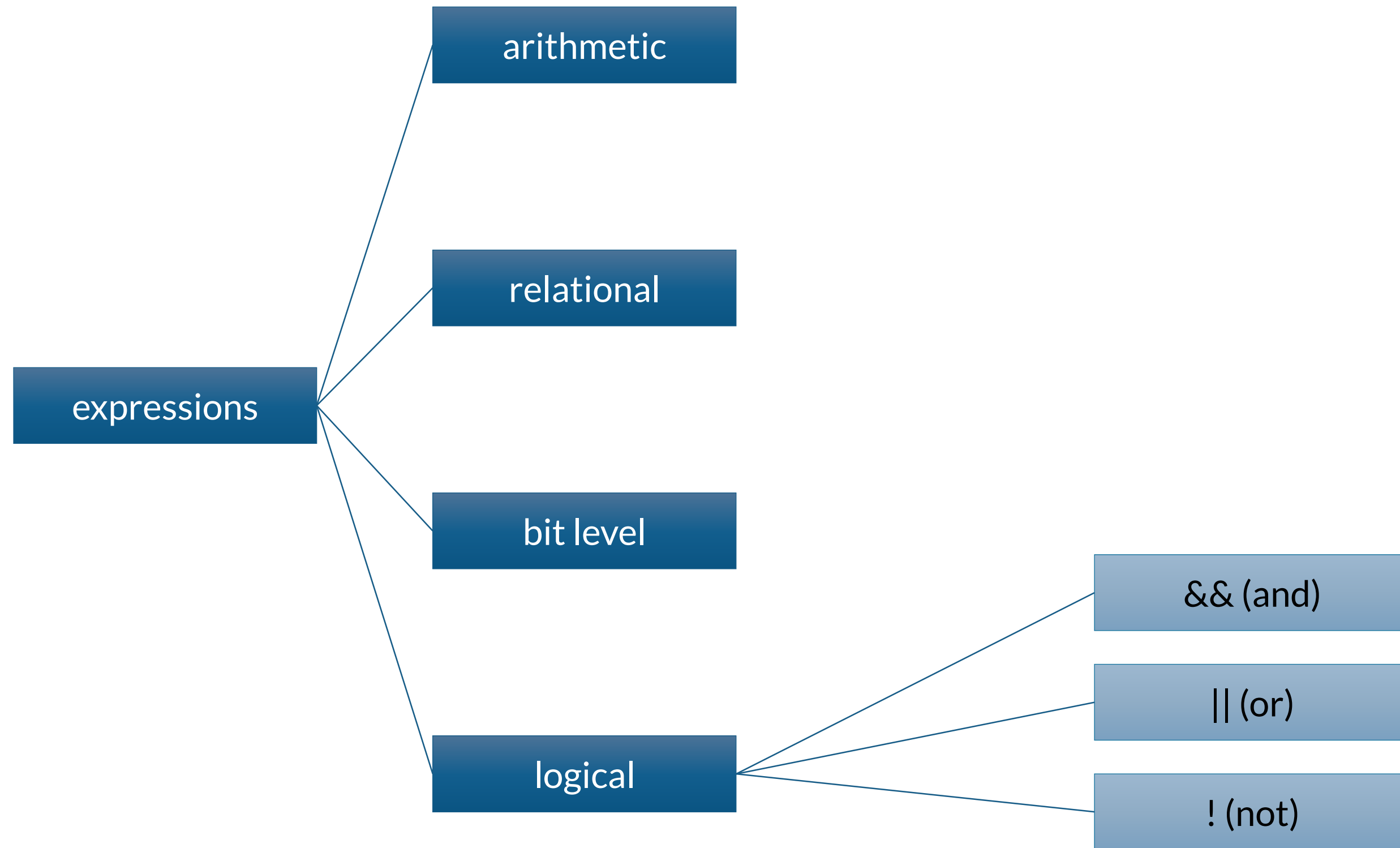
        System.out.println(x & y);           // 000000000000000000000000000000000000000010010
        System.out.println(x | y);          // 0000000000000000000000000000000000000000110111
        System.out.println(~x);             // 111111111111111111111111111111111111101001

        x = 0b00000000000000000000000000000000000000001001; // 9
        System.out.println(x >> 3);         // 000000000000000000000000000000000000000000000000000001
        System.out.println(x >>> 3);        // 0000000000000000000000000000000000000000000000000000001

        x = -9;                             // 111111111111111111111111111111111111110111
        System.out.println(x >> 3);         // 111111111111111111111111111111111111111110
        System.out.println(x >>> 3);        // 000111111111111111111111111111111111111110
    }
}
```



# Logical expressions



# Example with logical operators

Logical class 

```
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



# Casting

Java performs a **automatic** type conversion when there is no risk for data to be lost.

TestCast class 

In order to specify conversions where data can be lost it is necessary to use the **cast** operator.

```
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  
  
        long e = 34;  
        int f = (int)e;    // e is a long  
        double g = 3.45;  
        float h = (float)g; // g is a double  
    }  
}
```





# Control structures: if

If class 

```
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
```



# Control structures: while

While class 

```
public class While {  
    public static void main(String[] args) {  
        final float initialValue = 2.34F;  
        final float step = 0.11F;  
        final float limit = 4.69F;  
        float var = initialValue;  
  
        int counter = 0;  
        while (var < limit) {  
            var += step;  
            counter++;  
        }  
        System.out.println("Incremented " + counter + " times");  
    }  
}
```

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



# Control structures: for

For class 

```
public class For {  
    public static void main(String[] args) {  
        final float initialValue = 2.34F;  
        final float step = 0.11F;  
        final float limit = 4.69F;  
        int counter = 0;  
  
        for (float var = initialValue; var < limit; var += step)  
            counter++;  
        System.out.println("Incremented " + counter + " times");  
    }  
}
```

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



# Control structures: break and continue

BreakContinue class 

```
public class BreakContinue {  
    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; // abandon the loop if the counter is equal to 8  
  
            System.out.println(counter);  
        }  
        System.out.println("done.");  
    }  
}
```

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



# Control structures:switch

Switch class 

```
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 1:// months with 31 days
                case 3:
                case 5:
                case 7:
                case 8:
                case 10:
                case 12: days += 31;
                    break;
```

```
                case 2: // February is a special case
                    if (leapYear)
                        days += 29;
                    else
                        days += 28;
                    break;
                default: // a month with 30 days
                    days += 30;
                    break;
            }
        }
        System.out.println(days);
    }
}
```

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

```
$ java Switch
366
```





# Arrays

Arrays can be used to store a number of elements of the **same** type

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

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

Important: The declaration does not specify a **size**. However, it can be inferred when initialized

Other possibility to allocate space for arrays consists in the use of the operator **new**

```
int i = 3,j = 5;  
double[] d;  
  
d = new double[i+j];
```



# Arrays

Components can be accessed with an integer **index** with values from 0 to length minus 1.

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

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

Every array has a member called **length** that can be used to get the length of the array

Components of the arrays are initialized with **default** values

```
int []a = new int[3];  
for(int i = 0;i < a.length;i++)  
    System.out.println(a[i]);  
}
```

0  
0  
0



# Arrays

Arrays class 

```
public class Arrays {  
    public static void main(String[] args) {  
        int[] a = {2,4,3,1};  
  
        // compute the summation of the elements of a  
        int sum = 0;  
        for(int i = 0;i < a.length;i++) sum += a[i];  
  
        // create an array of the size computed before  
        float[] d = new float[sum];  
        for(int i = 0;i < d.length;i++) d[i] = 1.0F / (i+1);  
  
        // print values in odd positions  
        for(int i = 1;i < d.length;i += 2)  
            System.out.println("d[" + i + "]=" + d[i]);  
    }  
}
```

```
$ java Arrays  
d[1]=0.5  
d[3]=0.25  
d[5]=0.16666667  
d[7]=0.125  
d[9]=0.1
```



# The for-each iteration

ForEach class 

```
public class ForEach {  
    public static void main(String[] args) {  
        int[] a = {2,4,3,1};  
  
        // compute the summation of the elements of a  
        int sum = 0;  
        for(int x : a) sum += x;  
  
        // create an array of the size computed before  
        float[] d = new float[sum];  
        for(int i = 0; i < d.length; i++) d[i] = 1.0F / (i+1);  
  
        // print all values (note the use of type inference!!)  
        for(var f : d)  
            System.out.println(f);  
    }  
}
```



# Methods with variable number of arguments

A variable length argument list is specified with three periods:

```
int add(int ... values) {  
    int summation = 0;  
    for(int i = 0; i < values.length; i++) {  
        summation += values[i];  
    }  
    return summation;  
}
```

```
int sum = add(1, 2, 3, 4, 5);
```

The argument is implicitly declared as an **array**, however, the function can be called with a variable number of arguments







Thank you!

[esteco.com](http://esteco.com)

