Lexical Aspects

- Comments

```
// comment                    until end of line
/* comment  */                more than one line
```

```
/**
  Comment for javadoc
*/
```

- White spaces: separators and for clarity
- Declarations and instructions separated by “;”
- Case sensitive in identifiers:
  - A letter, followed by letters or digits, including $ or _
  - Naming conventions: aVariable, AClass, ACONSTANT, aMethod(anotherVariable)

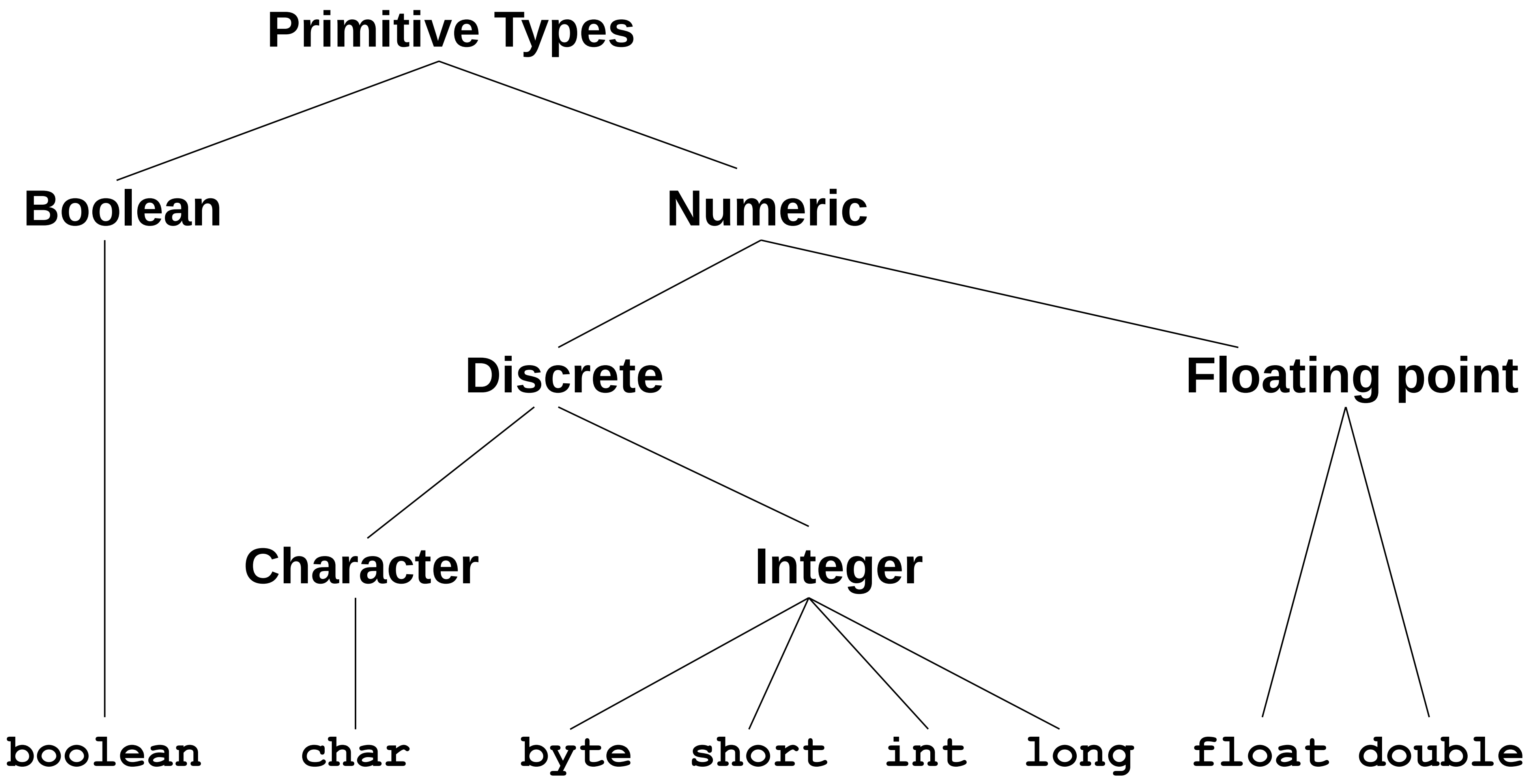
3.1. Introduction to Java (Appendix)

- Introduction, origins, environment
- Introduction by examples
- **Basic elements of the language**
  - **Primitive data types**
  - **Non-primitive data types**
  - **Control statements**
  - **Web applications**

3.1. Introduction to Java (Appendix)

- Introduction, origins, environment
- Introduction by examples
- **Basic elements of the language**
  - **Primitive data types**
  - **Non-primitive data types**
  - **Control statements**
  - **Web applications**

Primitive data types



Primitive data types

Basic or primitive types (fixed sizes, portability**)**

`byte`	1 byte	values between -128 y 127
`char`	2 bytes	(unsigned, Unicode characters, including ASCII and more …)
`short`	2 bytes	values between -32768 and 32767
`int`	4 bytes	values between -2^{31} y $2^{31}-1$
`long`	8 bytes	values between -2^{63} y $2^{63}-1$
`float`	4 bytes	rationals with 6 significative decimal digits
`double`	8 bytes	with 15 decimal digits
`boolean`		`true` and `false` (not numeric)

Literals

`-81` `12345678901`**`L`** `0xBEBA` `010` (`010` is 8 in octal)

`2.5` `1.72F` `11.03125D` `'A'` `'\t'` `'\u005B'`

Variables

```
char aLetter;          // declaration
// A local variable is not initialized with a default value,
// System.out.println(aLetter) raises an error

aLetter = 'a';         // assignment

short x, y, z;         // multiple declaration

double myMontlySalary;                // declaration
double myYearlySalary = 15000;        // declaration + initializ.
final int MONTHSPERYEAR = 12;         // constant
// assignment to a previously declared variable

myMontlySalary = myYearlySalary / MONTHSPERYEAR;
boolean lowSalary;   // declaration between statements
lowSalary = myMontlySalary < 1000;  // assignment
```

Compatibility between numeric types

```
byte b = -15;
//byte b = -152;        // Error: values between -128 y 127
char c = 'a';           // also valid: c = 97;  but less clear
short s = 1024;
int i = 50000;
long l = 120000;
float f = 5.67f;        // the f is needed
double d = .1234;       // same as 0.1234
double result = (f*b) + (i/s) - (d*s);
System.out.println ((f*b) + " + " + (i/s) + " - " + (d*s));
System.out.println ("result = " + result);
```

- Automatic conversion

```
i = s;
d = b;
d = f;
d = l; //beware!
```

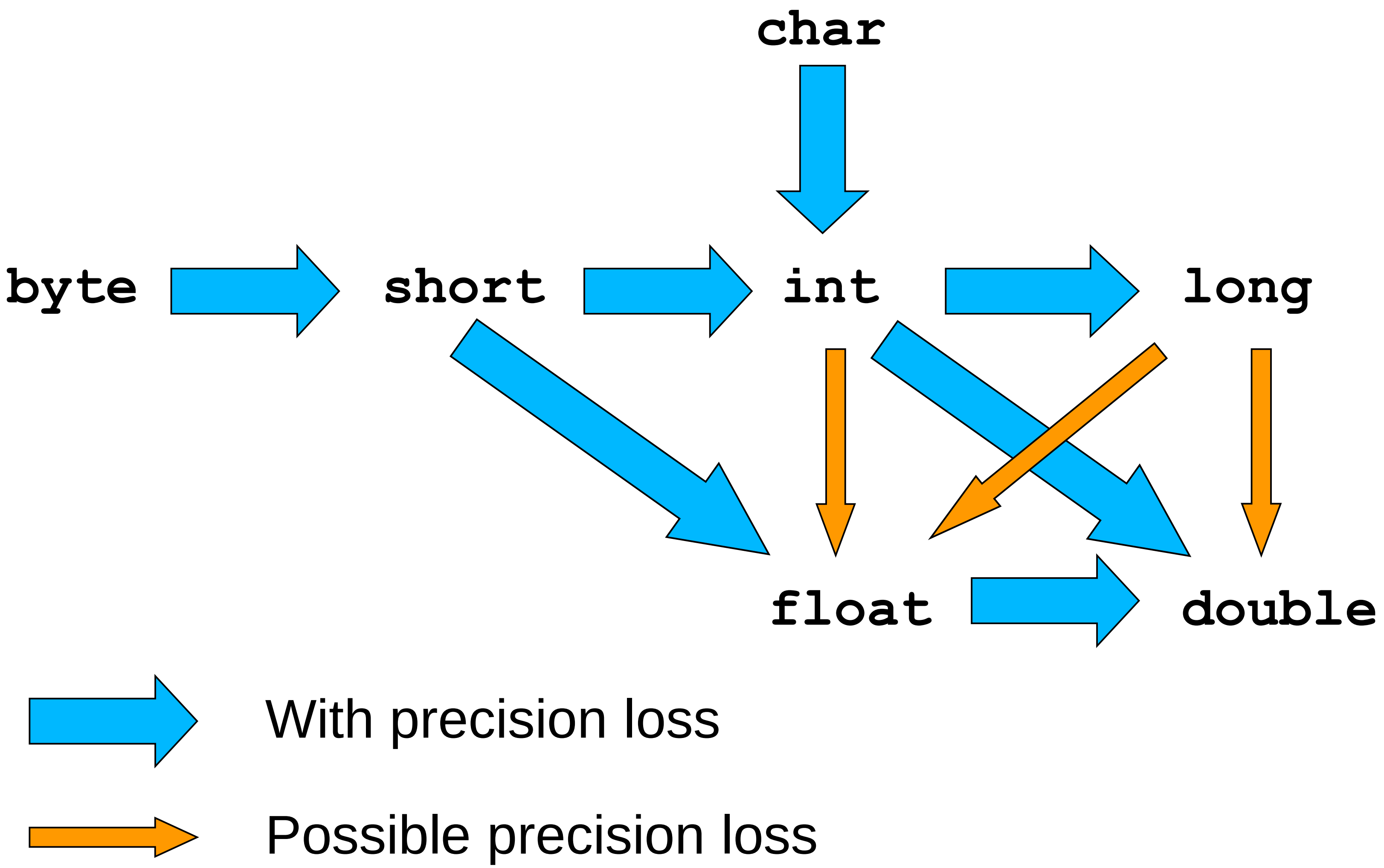
- Explicit casting

```
s = (short) i;
f = (float) d;
i = (int) d;
b = (byte) f;
```

- char: *explicit casting*

```
s = (short) c;
c = (char) s;
c = (char) b;
```

Assignments with loss of precision



Operators

- 46 operatos
- Numeric

  ```
  +     -     *     /     %
  +=    -=    *=    /=    %=     --    ++
  ```
- Logical

  ```
  &     |     ^     !     &&     ||
  ```
- Bit operators

  ```
  &     |     ^     ~     <<     >>    >>>
  ```
- Relational
  - Any type: `==` `!=`
  - Numeric types: `>` `<` `>=` `<=`
- Conditional expression

  `(`<u>*condition*</u>`)?` <u>*action1*</u> `:` <u>*action2*</u>

Operator precedence

Precedence level	Associativity
`[]` `new` `.` `(`*parameters*`)`	LEFT to RIGHT
`!` `~` `++` `--` `+`*expr* `-`*expr* `+` (*unary*) `-` (*unary*) `(`*type_or_class*`)` *expr*	**RIGHT to LEFT**
`*` `/` `%`	LEFT to RIGHT
`+` `-`	LEFT to RIGHT
`<<` `>>` `>>>`	LEFT to RIGHT
`<` `<=` `>` `>=` `instanceof`	LEFT to RIGHT
`==` `!=`	LEFT to RIGHT
`&`	LEFT to RIGHT
`^`	LEFT to RIGHT
`|`	LEFT to RIGHT
`&&`	LEFT to RIGHT
`||`	LEFT to RIGHT
condition `?` *expr1* `:` *expr2*	**RIGHT to LEFT**
`=` `+=` `-=` `*=` `/=` `%=` `&=` `^=` `|=` `<<=` `>>=`	**RIGHT to LEFT**

Non-Primitive types (Reference Types)

- Defined by means of ***classes***

  Built-in classes of the Java language

  `String, Array, Enum, Thread, Exception,…`

  Classes of the standard libraries

  `BigInteger, BigDecimal, File, List, Hashtable,…`

  Classes of additional libraries

  `JMenu, JWindow, SQLData, ImageIO, KeyGenerator,…`

  Classes defined in the program being written

  `Account, SavingsAccount, FixedTermAccount,`
  `CreditAccount, Card, CreditCard,`
  `DebitCard, CashCard, Client,…`

- http://docs.oracle.com/javase/8/docs/api/index.html

3.1. Introduction to Java (Appendix)

- Introduction, origins, environment
- Introduction by examples
- **Basic elements of the language**
  - **Primitive data types**
  - **Non-primitive data types**
  - **Control statements**
  - **Web applications**

Strings

- ***String*** variables and literals are objects of the class `java.lang.String` (class instances)
- They are not primitive types, are not terminated by \0, and are not a char array
- Being objects, they can be invoked methods defined in the String class:

  length() charAt(int) concat(String) compareTo(String) …

Strings

- Declaration of the object of type String

```
String title;                    // only declared
String name, surname1, surname2;
```

- Creation, memory allocation, initialization and assignment

```
String author = "Saramago";
title = "";                           // created, with empty string
surname1 = author;                    // content is not copied
```

- Access methods

```
char first = author.charAt(0);        // first value is 'S'
int t = title.length();               // t value is 0
```

- Errors

```
int e = name.length();           // Error: string is not created
author[0]                        // Error: strings are not arrays
```

Arrays

- Arrays are *objects*.
  - In addition to the array content, they have the `length` attribute
- Indexed collection of homogeneous elements
  - Primitive types or references
  - The first index of an array `A` is 0, the last one is `A.length-1`
- Multidimensional arrays

- Declaration of the Array object

  int[] a;   // just declared;
  int other[];   // this syntax is also valid

- Creation, memory allocation, initialization

  int[] b = **new int**[7];   // created, with 7 zeroes
  char[] c = **{**'U', 'S', 'A'**}**; // created and initialized, {} ≈ new
  byte[][] x = {{1,2},{},{3},{4,4,4,4}}; // bidimensional array

- Access to length:

  int k = c.length;   // k value is 3

Arrays

- Content access

```
char n = c[0];                      // n value is 'U'
int m = x[2][0];                    // m value is 3, m[2,0] is not allowed
```

- Errors:

```
int e1 = a.length;                  // Error: array is not created
byte e2 = x[1][0];                  // ArrayIndexOutOfBoundsException
```

- Array assignment

```
int[] power;
int[] even = {2,4,6,8};
power = even;                       // Content is not copied
power[2] = 1;                       // Also changes the even array
```

- Can be copied

```
int[] copy = new int[4];
System.arraycopy(even,0,copy,0,even.length);
int otther[] = even.clone();
```

- Can be converted to strings

```
char[] c = {‘J', ‘a', ‘v‘, ‘a’};
String language = new String(c);     // language is “Java"
```

More utilities in java.util.Arrays

3.1. Introduction to Java (Appendix)

- Introduction, origins, environment
- Introduction by examples
- **Basic elements of the language**
  - **Primitive data types**
  - **Non-primitive data types**
  - **Control statements**
  - **Web applications**

Basic statements in Java

- Very similar to C … including new ones
- **Blocks**:   { … }
  - can be nested, can have local variables, static scope, also with labels
- **Conditionals**:   if/else   switch/case/break
- **Loops**:   while do/while for   and enhanced for
- **Structured jumps**: continue, break (both can accept labels)
- Termination and return values: **return**

Conditional: if

if (*condition*) *action*$_1$ [**else** *action*$_2$]

```
if ( a>b )
   if ( a>c ) { maximo = a; }  // optional brackets { }
   else { max = c; }
else
   if ( b>c ) { max = b; }
   else { max = c; }
```

```
if ( n == 0 ) {               // { } needed
   if ( m == 0 ) System.out.println("indeterminacy");
}
else
   System.out.println("Result = "  + m/n);
```

Conditional: swith/case/break

```
switch ( expresion ) {// types byte, short, char, int, enum and String
    case ec_1: [ case ec_2: … case ec_i: ] {
            statements
            break;
    }
    …
    case ec_j: … {
            statements
            break;
    }
    default:
            statements
            break;
}
```

The values of the cases ec_i are constant expresions without repetitions

Loops

```
while (condition) {
            ...
}
```

```
do {
   ...
} while (condition)
```

```
for (inicialization; condition; loop) {
            ...
}
for (type variable : colection/array) {
  ...
}
```

(a collection can also be iterated with .forEach(<l-exp>), as we will see later).

```
String[] words= {"hi","hello","hola","eh!"};
for (String element : words)
  System.out.println(element);
```

Labelled Break

```
boolean cond = true;
a: {
b:   {
c:     {
          System.out.println(”Before break”);
          if (cond) break c; else break b;
          // System.out.println(”Never executed”);
       }
       System.out.println(”Executed if cond true”);
     }
     System.out.println(”After b, always executed”);
   }
```

- Could be used for error handling…
- But using exceptions is much better

Labelled Continue

```
for (int i = 0; i<10; i++) {                // 0 1
        System.out.print( i + " ");         // 2 3
        if (i % 2 == 0) continue;           // 4 5
        System.out.println();               // 6 7
}                                           // 8 9
```

```
extern: for (int i = 0; i<10; i++) {             // 0
             for (int j = 0; j<10; j++) {        // 0 1
               if (i < j) {                      // 0 2 4
                  System.out.println();          // 0 3 6 9
                  continue extern;               // 0 4 8 12 16
               }                                 // …
               System.out.print(" " + (i * j));
             }
}
```

Termination and return value

Return is used in ***methods*** because there are no procedures or functions (we will soon see the difference)

```
public class MainClass {
   public static void main(String[] args) {
      System.out.println("3! = " + factorial(3));
      return;
   }

   static long factorial(long n) {
      if (n == 0) { return 1; }
      else { return n * factorial(n - 1); }
   }
}
```

3.1. Introduction to Java (Appendix)

- Introduction, origins, environment
- Introduction by examples
- **Basic elements of the language**
  - **Primitive data types**
  - **Non-primitive data types**
  - **Control statements**
  - **Web applications**

Web applications: applets

Java can be executed within web browsers

```
<HTML>
<HEAD>   <TITLE> Check date </TITLE> </HEAD>
<BODY>
   <APPLET CODE=“CheckDate.class">
   </APPLET>
</BODY>
</HTML>
```

The browser downloads the file `CheckDate.class` and executes it

The code is in the server, but executed in the client

Applets is an old technology, rarely used today (Java EE instead)

Web appications: **`applets`**

Subclass of **`JApplet`**, with entry point **`init()`**

```
import java.util.*;
import javax.swing.*;
public class CheckDate extends JApplet {
   public void init() {
      JLabel label = new JLabel( (new Date()).toString() );
      add(label); // adds the element to the panel
   }
}
```