

Overview and Language Fundamentals

- Java Overview
- Writing Your first Java Program
- Language Fundamentals (Summary)

Claudia Niederée, Joachim W. Schmidt
Software Systems Institute
OOAD 1999/2000

c.niederee@tu-harburg.de
[http:// www.sts.tu-harburg.de](http://www.sts.tu-harburg.de)

Java: History

- Rapid development
- 1990: developed for use in electronic devices
- about 1995: gained popularity because of possible use in the Internet
- by now: accepted as scalable programming language for different purposes

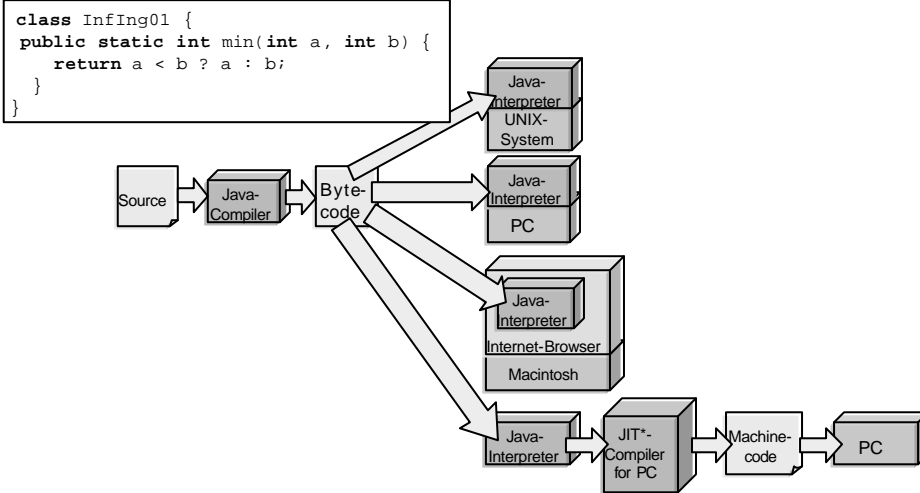
Java is Designed For:

- A fresh start (no backwards compatibility)
- Pure OOP: C++ syntax, Smalltalk style
- Improvements over C++, near guarantee that you can't run a bad program
- Internet programming: probably can't create viruses, programmability at the browser end, easier client/server programming
- Larger, more complex programs; smaller teams, less time
- There is the speed issue

How does Java generally work?

- Compilation creates bytecode
- Platform-independent
- Interpreted
- Bytecode is either:
 - interpreted statement by statement => "slow"
 - once compiled to a program in machine language => "fast"

From Source Code to Execution



* *JIT = Just In Time*

OOAD99/00-STS-Overview and Language Fundamentals

5

Writing Your First Java Class

```

class Cell {
    private int value = 0; } Data member

    void setValue(int newValue){
        this.value = newValue;
    }
    int getValue( ) {
        return this.value;
    }
    ...
}

```

Methods

Class definition

OOAD99/00-STS-Overview and Language Fundamentals

6

Naming Conventions

- Words run together, no underscores
- Intermediate words capitalized (`getValue`)
- Classes: first word capitalized (`Cell`)
- Methods and variables: first word lower case (`value`, `getValue`)
- Constants: all caps with underscores to separate words (like `C`).

Writing a Java Program

```
import java.util.*;
class Cell {
    int value = ...
    void printValue(){
        System.out.println("Date: " + new Date());
        System.out.println("Value: " + this.value);
    }
    public static void main(String [] args){
        Cell cell1 = new Cell();
        cell1.setValue(4);
        cell1.printValue();
    }
}
```

The Keyword *static*

- Normally each object gets its own data
- What if you want only one piece of data shared between all objects of a class? ("class data")

```
class StaticTest {  
    static int i = 47;  
}
```

See also
`System.out.println(...)`

- What if you want a method that can be called for the class, without an object? ("class method")

```
class StaticFun {  
    static void incr() { StaticTest.i++; }  
}  
StaticFun.incr();
```

See also: `main`

Using Other Components

- Bring in a library of components using **import** keyword
- Can specify specific element in library:

```
import utility.MyTools;
```

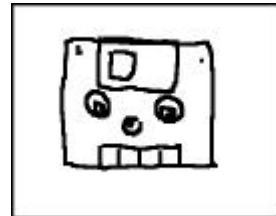
- Can specify entire library:

```
import java.util.*;
```

Java Applications - Important Steps

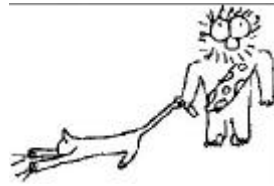
- Create some source code within your text editor like the class *Cell*
- Save it as *.java* - file, use the name of the (primary) class plus the extension *.java* (e.g. *Cell.java*)
- Compile the source file:
`javac Cell.java`
- You get bytecode within a *.class* - files, for all the classes included (*Cell.class*)
- Run your application by interpreting the primary class' bytecode
`java Cell`
- The Java interpreter looks for the main-method and executes it.

Data



- In Java, almost everything is an object.
- Objects are described by reference types like the type *String*, classes or arrays
- Some simple data types, however, are frequently used as programming basis.
- These so-called primitive types are built-in types, implying a behaviour different from reference types

Primitives



- Built-in types: not object handles, but variables on the stack like C.

boolean, char (Unicode), byte, short, int, long, float, double

- Consequences for bindings and comparisons.
- Size of each data type is machine independent.

Binding and Comparing Primitives

- Binding a variable to a primitive value means copying this value into the variable's storage cell.

```
int a = 100;
int b = 75;    a 100    b 75
```

- Assigning a new value to an existing variable means replacing the original value by a copy of the new value.

```
a = b;
```

- A comparison is done by comparing the bits of the copied values.

```
a 75    b 75
a == b;
```

Booleans

- The primitive type used to express that something can have exactly two different states is called boolean.
- The two literals are true and false.
- Operations on booleans:
 - Comparison: ==, !=
 - Negation: !
 - binary logic: &, |, ^
 - short-circuit-evaluation logic: &&, ||

Ternary if-else

```
int a,b;  
...  
b = a >= 0 ? a : -a;
```

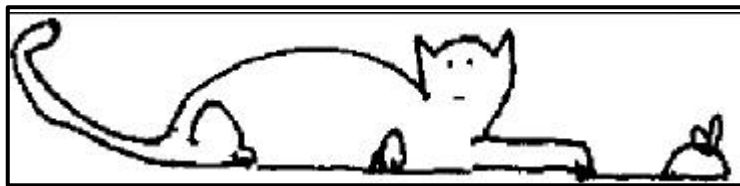
- Does the same as

```
int a,b;  
...  
if (a >= 0) b = a; else b = -a;
```

- A ternary if-else always results in a value
- Can also be used for side-effects
- Shouldn't be used too often for the sake of readable code

Controlling Program Flow

- Variable declaration and assignment
- Using operations on primitive data types
- Iteration
- Conditionals



Statements

- There are two general kinds of statement
 - simple statements
 - compound statements
- Simple statements end with a semicolon.
- Compound statements encompass several statements and are enclosed in brackets.
- Compound statements are also called blocks.
- Blocks are important for scoping.

Variables

- Variables have to be declared before they are used.

```
int a;
```

- They can be assigned changing values according to their declared type.

```
a = 3; a = a + 2; a = "Oh no!";
```

- Declaration and assignment can be done within a single statement.

```
int b = 8;
```

- Some operations involve implicit assignments

```
b = ++a;
```

```
a 6 b 6
```

- Variables that are used but not initialized cause compile-time errors.

Scoping

```
{ /* <-- Beginning of scope 1 */  
  int x = 12;  
  /* Only x available */  
  { /* <-- Beginning of scope 2 */  
    int q = 96;  
    /* Cannot redefine x here! */  
    /* Both x & q available */  
  } /* <-- End of scope 2 */  
  /* Only x available */  
  /* q out of scope */  
} /* <-- End of scope 1 */
```

if-else

```
int a;  
...  
if (a >= 0)  
    System.out.println("Already positive");  
else {  
    a = -a;  
    System.out.println("Turned positive");  
}
```

- The statements can either be simple (terminated by semicolon) or compound in braces.
- else is optional

switch

```
char c;  
...  
switch(c) {  
    case 'a':  
    case 'o':  
    case 'u':  
        System.out.println("dark vowel");  
        break;  
    case 'e':  
    case 'i':  
        System.out.println("bright vowel");  
        break;  
    default:  
        System.out.println("no vowel");  
}
```

What does switch do?

- Tests all case-conditions and executes the statement behind the colon, if the condition is true.
- If a condition is true and the attached statement is followed by a **break**, switch is finished.
- If no match occurs, the **default**-statement is executed.
- switch only works on integral values like **int** or **char**.

for

```
int [] a = ...;
int sum = 0;

for (int i = 0; i < 10; i ++) {
    sum = sum + a[i];
}
```

- performs initialization before first iteration, conditional testing at the beginning of each iteration and some form of stepping at the end of each iteration
- all three parts are optional

while

```
int i = 0;
while (i < 10) {
    System.out.println(i);
    ++i;
}
```

- Tests a condition and executes the loop-statements as long as the condition is true.

Nobody is perfect ...

- The following program code contains the typical syntax errors a programmer has to cope with.
- Some appear time and again :- (
- Training to find them is useful ...

```

import java.util.*;

public class goodMorning {
public void main(String[] args) {
    int counter;
    System.out.println(new Date());
    System.out.println("Could do some sports...");
    for(int i = 1; i <= 20; i++) {
        if (i % 3 = 0) {System.out.println("Too tired!");continue;};
        counter = (i % 2 != 0 ? counter + i : counter);
        if (i % 2 == 0) System.out.print("Bowing my knees, ");
        else System.out.print("Sitting up, ");
        switch(i)
            case 1: System.out.println("once!"); break;
            case 2: System.out.println("twice!"); break;
            default: System.out.println(i + " times!");
        };
    System.out.println("Wow, I am great!");
    System.out.println("I did " + counter + " sit-ups!");
};
}

```