326.041 (2015S) – Practical Software Technology

(Praktische Softwaretechnologie)

The Java Programming Language

Alexander Baumgartner
Alexander.Baumgartner@risc.jku.at

Research Institute for Symbolic Computation (RISC)
Johannes Kepler University, Linz, Austria

History of Java I

James Arthur Gosling is the “father” of the Java programming language. He is a Canadian computer scientist, born in 1955.

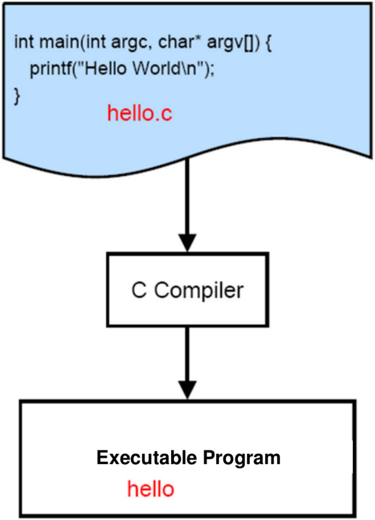
Figure: James Gosling 2008, by Peter Campbell. Licensed under GFDL via Wikimedia Commons.

History of Java II

- It began as “Oak”, created by **James Gosling** in 1991.
- 1995: **Sun Microsystems** releases the first public version, Java 1.0.
- 1995: Integration into Netscape.
- 1996: Definition of the language by Gosling, Bill Joy, Guy Steele.
- 1998: Java 2 (J2SE 1.2).
- 2006 – 2007: Sun makes all of Java’s core code **open-source**.
- 2009 – 2010: **Oracle Corporation** acquired Sun Microsystems.

Java has never been formally standardized. It is a **de facto standard**.

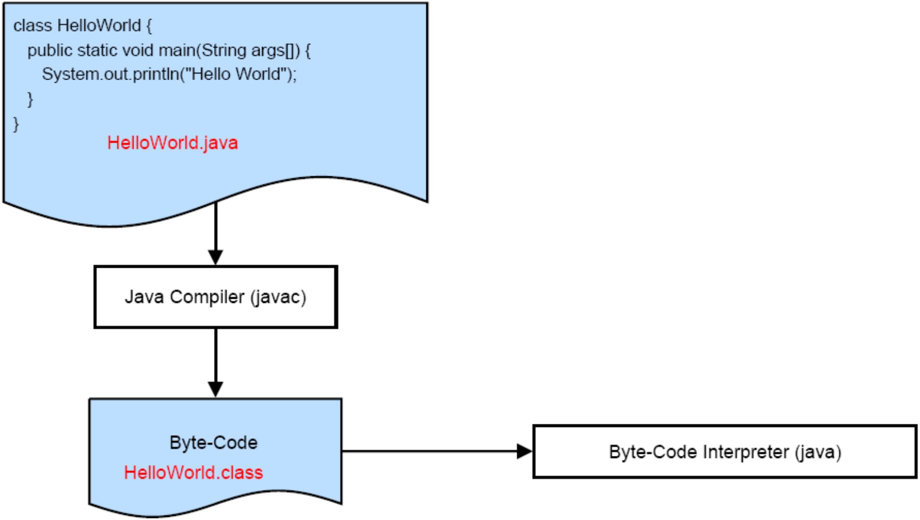
Machine-Code



```
int main(int argc, char* argv[]) {
  printf("Hello World\n");
}
```

Figure: Compiling and Running a C Program.

Byte-Code



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

HelloWorld.java

Java Compiler (javac)

Byte-Code

HelloWorld.class

Byte-Code Interpreter (java)

Figure: Compiling and Running a Java Program.

Byte-Code – Implications

- .class files are platform independent:
  - Write/Compile once, run anywhere.
- Byte-code is very compact:
  - Useful for network transfer.
- The interpreter is able to control access rights:
  - It is not necessary to trust in foreign codes.
- It is slower than machine code, but it is fast with JIT.

Data Types

- **Primitive types:** int, char, float,. . .
  Like in C but in Java they are **machine independent.**
- **Reference types:** Objects (vs. Pointer in C)
  Arrays, String,. . . are object types.

Primitive Data Types

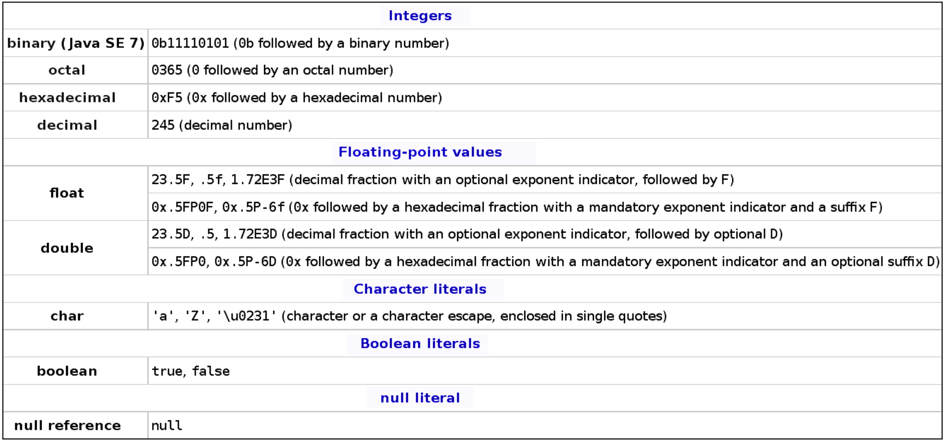
- **byte:** 8-bit signed integer. Range: $[-128, 127]$.
- **short:** 16-bit signed integer. Range: $[-32768, 32767]$.
- **int:** 32-bit signed integer. Range: $[-2^{31}, 2^{31} - 1]$.
- **long:** 64-bit signed integer. Range: $[-2^{63}, 2^{63} - 1]$.
- **float:** single-precision 32-bit IEEE 754 floating point.
- **double:** single-precision 64-bit IEEE 754 floating point.
- **boolean:** two possible values: true and false.
- **char:** 16-bit Unicode character.

Primitive types have so called Wrapper Classes:
Byte, Short, Integer, Long, Float, Double, Boolean, Character.
The Java compiler automatically converts (autoboxing) between primitive types and their wrapper classes.

- **void:** the "empty type", and its Wrapper **Void** (not instantiable).

Java 8 introduces unsigned interpretation of int and long.

Literals



Integers	
binary (Java SE 7)	`0b11110101` (`0b` followed by a binary number)
octal	`0365` (`0` followed by an octal number)
hexadecimal	`0xF5` (`0x` followed by a hexadecimal number)
decimal	`245` (decimal number)
Floating-point values	
float	`23.5F`, `.5f`, `1.72E3F` (decimal fraction with an optional exponent indicator, followed by `F`)
	`0x.5FP0F`, `0x.5P-6f` (`0x` followed by a hexadecimal fraction with a mandatory exponent indicator and a suffix `F`)
double	`23.5D`, `.5`, `1.72E3D` (decimal fraction with an optional exponent indicator, followed by optional `D`)
	`0x.5FP0`, `0x.5P-6D` (`0x` followed by a hexadecimal fraction with a mandatory exponent indicator and an optional suffix `D`)
Character literals	
char	`'a'`, `'Z'`, `'\u0231'` (character or a character escape, enclosed in single quotes)
Boolean literals	
boolean	`true`, `false`
null literal	
null reference	`null`

- **Fields** have default values 0, false, '\u0000', null. (Like calloc() in C.)
- **Local variables** don't have default values.

Operators

Operators in Java are similar to those in C:

- Arithmetik: $+, -, *, /, \%$
- Bind of Variables: $=, +=, -=, \ldots$
- Comparison: $==, !=, <, >, <=, >=$
- Incrementing/Decrementing: $++, --$
- Logical Operations: $\&\&, ||, !$
- Logical Operations on Bits: $\&, |, \hat{}$
- Bit Shift Operators: $<<, >>, >>>$
- Conditional Structures: $? \ :$
- Object Operators: **new, instanceof**

Comments

In Java there are 3 different kinds of comments:

- Comment in one line: // ...
- Comment in more lines: /* ...*/
- JavaDoc comment: /** ...*/
  - The command **javadoc** automatically generates a documentation.
  - JavaDoc comments are displayed inside the documentation.
  - Many IDEs display JavaDoc comments as tooltips.

Access Modifiers

The access of fields and methods can be controlled.

- **public:** Access from everywhere.

  public int age; **public String** getName() {. . . }

- **protected:** Access from subclasses and within the same package.

  protected int age; **protected String** getName() {. . . }

- **package private:** Access within the same package.

  int age; **String** getName() {. . . }

- **private:** Access within the enclosing block/class.

  private int age; **private String** getName() {. . . }

Local variables only exist temporarily (inside of a register or on the stack).

- **static:** The following method/variable is independent of any instances created for the class. It exists without/besides the objects. (E.g. System.out, public static void main,... See introduction slides.)

Strings

- Concatenation of Strings:

```java
String a = "This␣is␣a␣row";
String b = "This␣is␣another␣row";
String twoRows = a + "\n" + b;
```

- Concatenation of a String with another type:

```java
String s = "The␣answer␣is:␣" + 42;
```

The result is the same as:

```java
String s = "The␣answer␣is:␣" + String.valueOf(42);
```

Which is:

```java
String s = "The␣answer␣is:␣42";
```

- The operator + is left associative:

```java
System.out.println(2+3 + "␣Test"); // Prints 5 Test
System.out.println("TEST␣" + 2+3); // Prints Test 23
```

Strings are object/reference types

- a and b are references to the same object:

```
String a = "World";
String b = a;
```

- A new String object will be created:

```
a = "Hello␣" + a;
```

  The reference of the new string is stored in a.
  b still refers to the old string "World".

- String objects are immutable. They never change after their creation.

Arrays

- Creating (instantiating) an array:

```
int[] a = {1, 0, 4};
int[] b = new int[len];
```

- **a.length** gives the number of elements.
- **a[0]** is the first element.
- The class **Arrays** offers some static utility methods. E.g.:
  - Arrays.toString(a) returns a String representation of a.
  - Arrays.copyOfRange(a, 1, a.length) returns a copy of a, starting from its 2. element (which has index 1).
- There is **no pointer arithmetic** in Java:
  In C, a+1 is a pointer to the array from its 2. element.

If–Then–Else

```java
public static int abs(int x) {
    if (x < 0) {
        return -x;
    } else {
        return x;
    }
}
```

```java
public static int abs(int x) {
    if (x < 0)
        return -x;
    return x;
}
```

```java
public static int abs(int x) {
    return x < 0 ? -x : x;
}
```

Switch–Case

```java
String monthString;
switch (month) {  // month is of type int
    case 1:   monthString = "January";
              break;
    ...
    case 12:  monthString = "December";
              break;
    default:  monthString = "Invalid␣month";
              break;
}
```

```java
public String monthStr(int month) {
  switch (month) {
    case 1:   return "January";
    ...
    case 12:  return "December";
    default:  return "Invalid␣month";
  }
}
```

Fallthrough Switch–Case

```
String countDown = "";
switch (countFrom) {  // countFrom is of type int
    case 10: countDown += "10,␣";
    case 9:  countDown += "9,␣";
    ...
    case 1:  countDown += "1,␣";
    default: countDown += "Takeoff";
}
```

While

```java
public int digitSum(int number) {
    int sum = 0;
    while(number != 0) {
        sum += number % 10;
        number /= 10;
    }
    return sum;
}
```

Do–While

```java
...
String pwd;
do {
    printMessage("Enter␣a␣new␣password:");
    pwd = readInput();
    if (pwd.length() < 8) {
        printMessage("At␣least␣8␣characters.");
        pwd = null;
    }
} while(pwd == null);
...
private void printMessage(String msg) {
...
private String readInput() {
...
```

For Loops

- Suppose we have an array, which contains some int values:

```java
int[] a = new int[capacity];
...
```

- We compute the sum of all the values:

```java
int sum = 0;
for (int i = 0; i < a.length; i++)
    sum += a[i];
```

- Alternatively:

```java
int sum = 0;
for (int i = a.length; --i >= 0; sum += a[i]);
```

- Alternatively:

```java
int sum = 0;
for (int val : a)
    sum += val;
```

Return, Break, and Continue

- **return:** Terminates the current method and returns to the caller.
  - A return value might be passed to the caller.
- **break:** Terminates the execution of the (inner) loop.
  - A label might be given to break through more loops.
- **continue:** Jumps to the next iteration of a loop.

```java
outer: for(...) {
    for(...) {
        ...
        if (...)
            continue;
        if (...)
            break;
        if (...)
            break outer;
        ...
        // continue jumps to this position
    }
    // break jumps to this position
    ...
}
// break outer jumps to this position
```

Inheritance and Instantiation

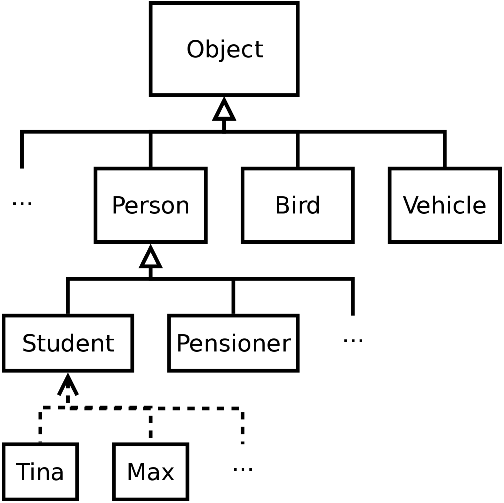


Figure: Every reference/object type is a subtype of Object.

Constructors and the new-operator I

- Empty default constructor: **new** CLASS_NAME();

```java
public class Person {
    String name;
    int age;
    boolean woman;
}
```

- Creating an object of type Person:

```java
Person p = new Person();
Object o = new Person();
System.out.println(p); // Prints type@id   Person@d273c8fc
System.out.println(o); // Prints type@id   Person@6d172f8f
```

```java
public class Object {
    ...
    public String toString() {
        return getClass().getName() + ”@”
               + Integer.toHexString(hashCode());
    }
```

Constructors and the new-operator II

- Override the method toString() from Object:

```
public class Person {
    public String name;
    public int age;
    public boolean woman;

    public String toString() {
        return "I␣am␣a␣" + (woman ? "woman" : "man") +
          "␣of␣" + age + "␣years␣and␣my␣name␣is␣" + name;
    }
}
```

```
System.out.println(new Person());
// I am a man of 0 years and my name is null
Person p = new Person();
p.name = "Tina";
p.woman = true;
System.out.println(p);
// I am a woman of 0 years and my name is Tina
// Is she really a newborn?
```

Constructors and the new-operator III

- Replace default Constructor:

```java
public class Person {
    private String name;
    private int age;
    private boolean woman;

    public Person(String nameArg, int age, boolean man) {
        // super();      // constructor from super class
        name = nameArg;
        this.age = age; // this is the self-reference
        woman = !man;
    }

    public String toString() {
        return "I␣am␣a␣" + (woman ? "woman" : "man") +
          "␣of␣" + age + "␣years␣and␣my␣name␣is␣" + name;
    }
}
```

```java
System.out.println(new Person("Tina", 22, false));
// I am a woman of 22 years and my name is Tina
```

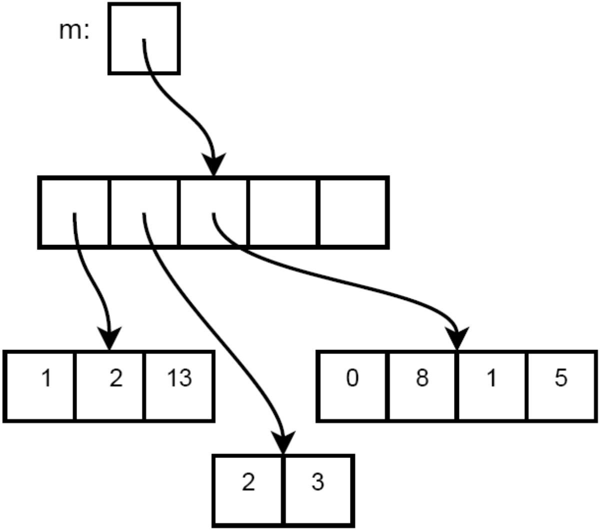
Variable Declaration and Assignment

```
int i;
int j = 23;
boolean jPositive = j > 0;
i = j = j - 1;                  // = is right associative
float f1 = j/10;
float f2 = j/10f;
Object obj = null;
PrintStream o = System.out;
int[] a = new int[3];           // int[] is an object type
int[] b = {1, 0, 0};
int[] c = a;                    // Same object as a
a[0] = 1;                       // Index starts with 0
obj = 7;                        // Autoboxing int ⇒ Integer

o.println("i=" +i+ ";␣j=" +j);  // Prints i=22; j=22
o.println(f1);                  // Prints 2.0
o.println(f2);                  // Prints 2.3
o.println(a.length);            // Prints 3; length is a field
o.println(Arrays.toString(c));  // Prints [1, 0, 0]
o.println(a == c);              // Prints true
o.println(a == b);              // Prints false
o.println(obj);                 // Prints 7
```

Multidimensional Arrays / Matrices

```
int[][] m;
```



Allocating Multidimensional Arrays

- A 5×5 matrix / array:

```
int[][] m = new int[5][];
for (int i = 0; i < m.length; i++)
    m[i] = new int[5];
```

- Or, as a shortcut:

```
int[][] m = new int[5][5];
```

- Iterating over 2-dimensional array: (Interpretation as array of row vectors.)

```
for (int[] rowVec : m)
    System.out.println(Arrays.toString(rowVec));
```

Exercise

- Let $f \colon \mathbb{R}^n \to \mathbb{R}^m$ be a linear map.
- Compute the transformation matrix M for arbitrary f such that $f(x) = Mx$ for all $x \in \mathbb{R}^n$.

See the guidance for this exercise on the Moodle page.