MASSEY UNIVERSITY
TE KUNENGA KI PŪREHUROA
UNIVERSITY OF NEW ZEALAND

Programming Paradigms
159.272
Introducing Object-Oriented Programming and Java

Amjed Tahir
a.tahir@massey.ac.nz

Readings

1. Java Tutorial Trail "Getting Started":
   http://docs.oracle.com/javase/tutorial/getStarted/index.html
1. Java Tutorial Trail "Learning the Java Language", first section "Object-Oriented Programming Concepts"
   http://docs.oracle.com/javase/tutorial/java/index.html

Object-Oriented Programming

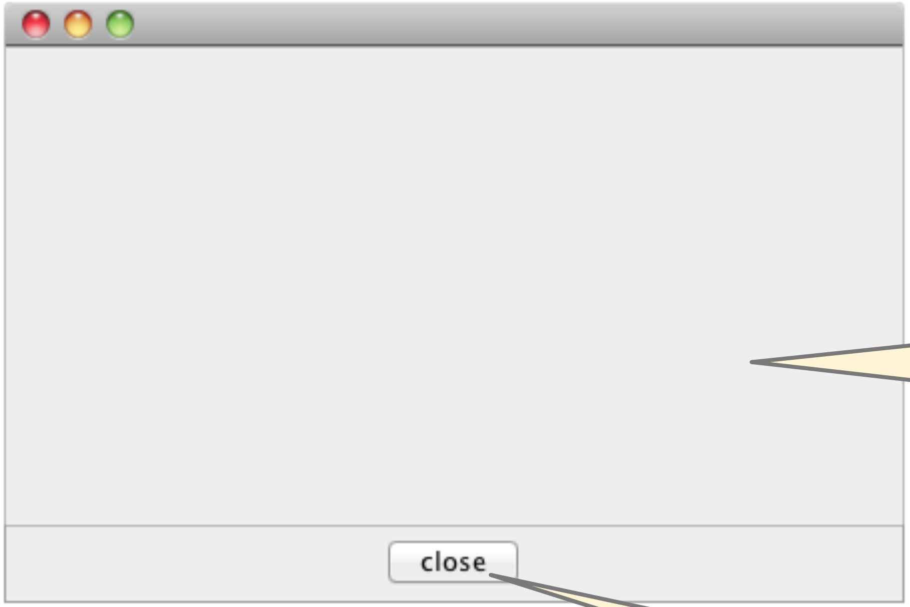
- OOP emerged in the 70s (Smalltalk,C++) and become a mainstream language in the early 90s

- based on the idea of organising programs around entities ("objects") which talk to each other.

- OO aim to improve modularity (separate modules from each other so they can be reusable).

- closely aligned with (English) grammar - therefore a very natural way to model the real world in programs

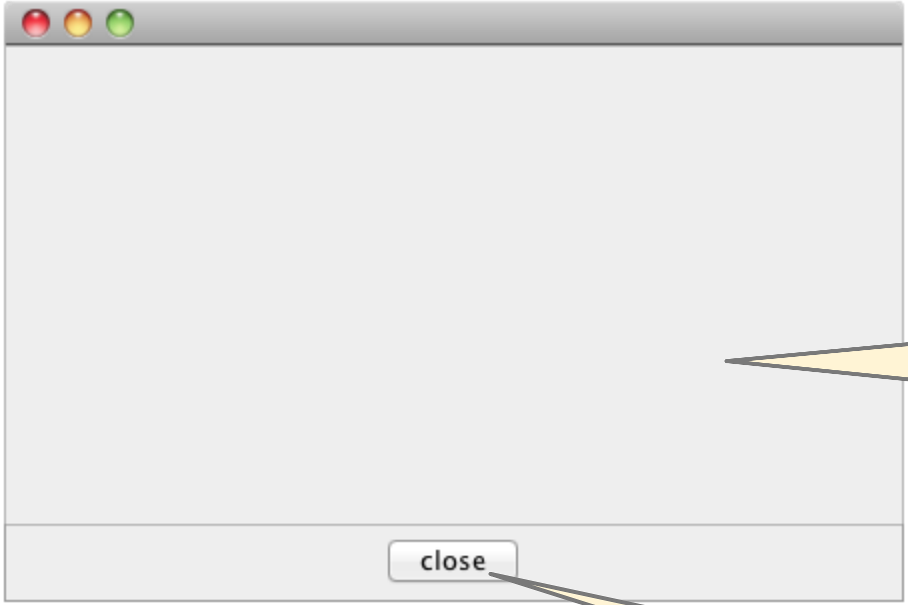
User Interface Components as Objects



object 1:
a window

object 2:
a close button

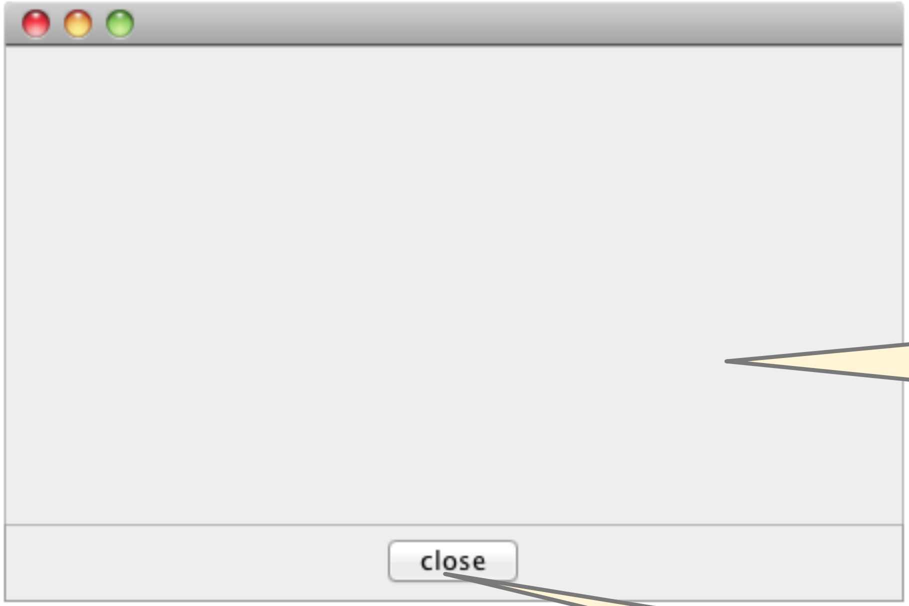
Objects Have State (=Properties)



object 1:
background: light gray
size: 400x300

object 2:
label: "close"
size: 60x20

Objects Have Behaviour (=Functions)

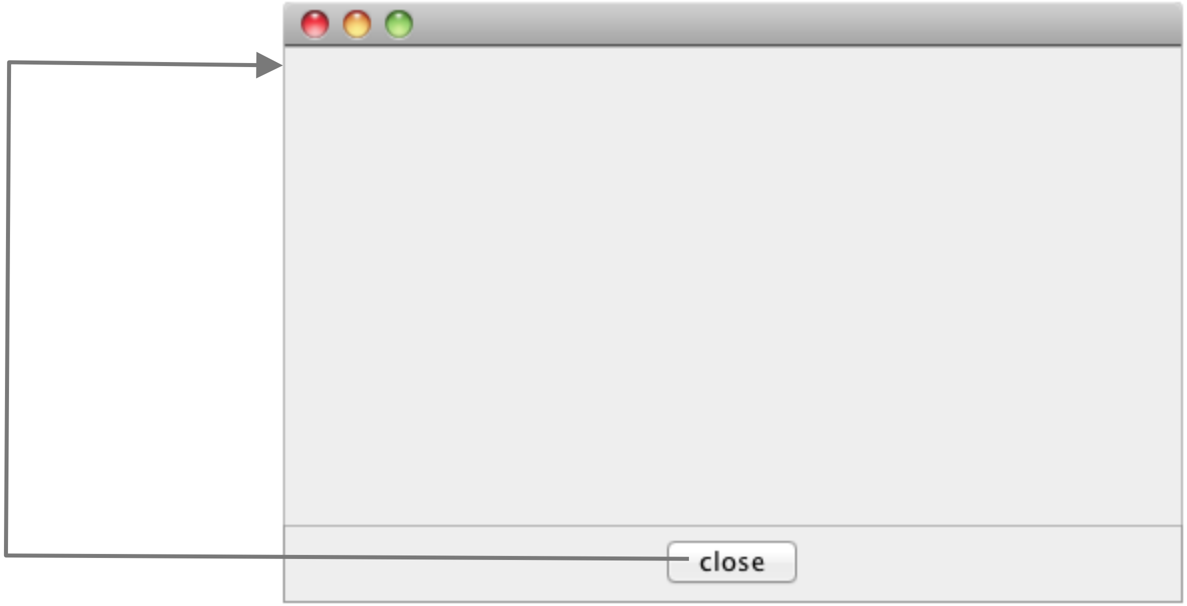


object 1:
iconify, resize, change background, **close**

object 2:
change label, disable, enable, make invisible

these functions attached to objects are called **methods**

Objects Interact (Talk To Each Other)



- when clicked, the close button sends a message "close" to the window
- close is one of the **methods** (behaviours) of the window object
- C/Python/Java syntax: **`window.close()`**

Alignment With English Grammar

Code (syntax: C/Python/Java style):

```
window.close()
```

English:

```
close the window
```

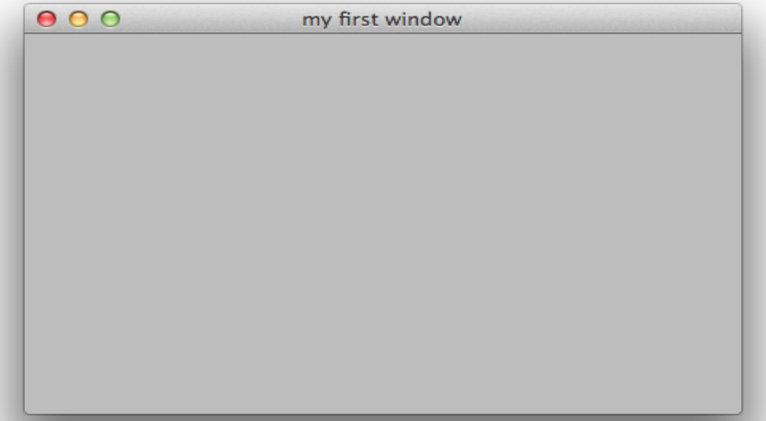
nouns = objects
verbs = methods

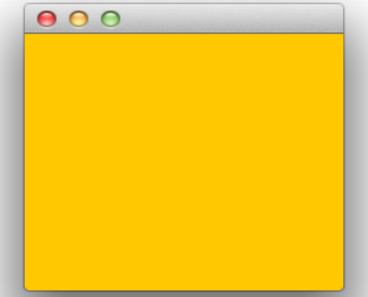
this correspondence is sometimes used to identify classes of objects when designing OO programs

Methods can have Parameters

- `window.resize(1000,500)` - resize window to 1000x500
- `button.setLabel("exit")` - change button label to exit

Classes





- same **kind** of object
- but different properties (size, title, background colour)
- methods are the same
- the object kind or type is called a **class**
- a class defines methods and properties (both windows have size and background colour, but the values differ!)

OO History: The PARC Story

- PARC (Palo Alto Research Centre) is the Xerox research centre near San Francisco
- Many important technologies were developed at PARC, including **graphical user interfaces**, the mouse, ethernet, and laser printers
- PARC developed the first pure OO language: **Smalltalk**
- Smalltalk heavily influenced the development of modern OO languages, including Java

- Additionally, PARC has also helped to develop Aspect Oriented Programming (AOP) - an improvement to OO programming! More on AOP later in the course!

Smalltalk

- very pure OO language
- syntax very different from modern (mainly C-like) languages
- key people: Adele Goldberg and Alan Key
- first public release: Smalltalk-80 around 1980
- documented in The Blue Book (free eBook available online)
- modern implementations: Squeak, Pharo, VA Smalltalk
- modern very Smalltalk-like language: Ruby

Other `pure` OO languages

- Eiffel - > Bertrand Meyer in 1985

- Java?

- C#

Hybrid OO languages

- Python, Scala, C++, Objective-c, Cobol, D, Perl, Swift, Ruby…..............

Java History

1991 A group of SUN engineers developed software for interactive TV sets (so-called Green project, Patrick Naughton, James Gosling). Interpreter named **OAK** (later renamed Java)

1993 WWW boom started. SUN team developed WEB runner (later re-named HotJava) - browser able to execute small OAK programs, so-called **applets.** Applets were the first Java killer-application - playing a role similar to what is done with Flash and HTML5 animations today.

1995 Netscape license Java and included it in its browser software

Java History (ctd)

1996 Release of the Java Development Kit 1.0, Green team became separate company JavaSoft, industry support for java from IBM, Oracle, Borland, Netscape etc. Release of JDBC java - database connectivity and java beans component architecture.

1997
release of the Java Development Kit 1.1
many IDE’s on the market (IBM, Sun, Symantec, Borland,..)

Java History (ctd)

Late 90ties - present

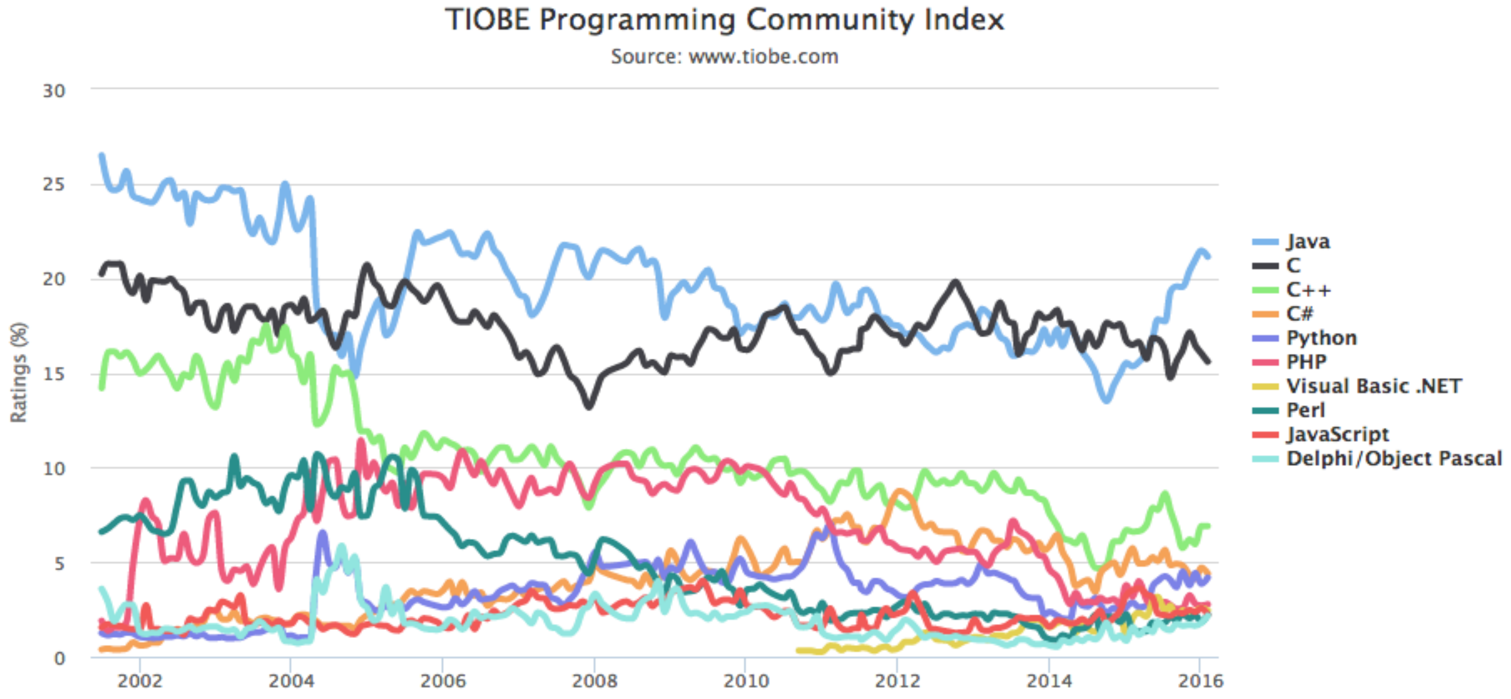
- Java is strong as programming language for servers / enterprise applications (J2EE, incl. servlets and jsp)
- Java on Linux gains importance
- Microsoft invents the very similar C# language in order to offer an alternative to Java
- Java tool market driven by two major open source projects: Netbeans (sponsored by SUN) and Eclipse (sponsored by IBM)
- version 1.5 (2004) added modern language features like generics and annotations

Current Trends

- major new language features :
  - functional features in 1.8 (released in 2013)
  - new modularity features (project Jigsaw) in 1.9 (released in 2014)
- the java runtime environment is widely used to execute other languages such as:
  - Ruby (JRuby)
  - JavaScript
  - Python
  - Scala, Closure, Fantom, Kotlin, Groovy, ...
- large competitive eco-system of free/open-source and commercial tools and libraries

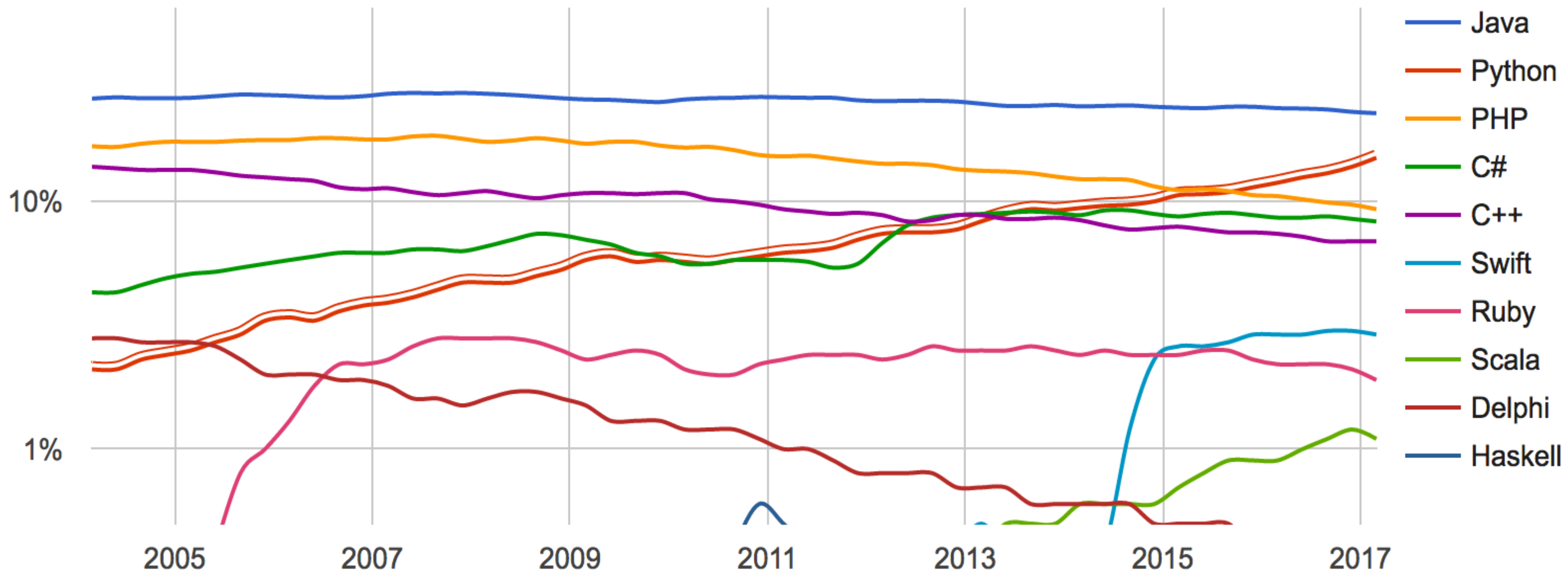
Why Java ?

1.Java is popular



http://www.tiobe.com/content/paperinfo/tpci/index.html

PYPL PopularitY of Programming Language



http://pypl.github.io/PYPL.html

Why Java ?

2. Java is safe (mostly!), robust, scalable and reliable

the Java language and the Java runtime environment have been developed and fine-tuned for almost 20 years

Java is widely used in enterprise computing with its high requirements for safety, scalability and reliability

Java has built-in language features such as multi-threading, sandboxing and exception handling that directly support these qualities

Why Java ?

3.Java is platform-independent

there is an ever-increasing diversity of operating systems (OSX, Windows, Linux, Unix, Android, ..)

Java supports all major PC (desktop and server) OSs - Mac, Linux/Unix and Mac OS X

Java is the main programming language used to write Android applications

Why Java ?

4. Java is not Python (and not very similar to it), but it is similar to C, C++ and C#

- Java is compiled and uses **static typing**, whereas Python (taught in 159.1** papers) is interpreted and uses dynamic typing
- most mainstream programming languages fall into one of these categories, and we therefore think it is good for students to have a good understanding of at least one language from each category!
- once you know Java, you should be able to pick up other C like languages fairly quickly

Why Java ?

5.The Java Eco-System

- there is a large number of (competing) tools and libraries for Java, both commercial and open source

- this has driven prices down and quality up, and has promoted innovation in the Java ecosystem
- this applies at several levels:

  Java language tools (e.g., multiple free or low cost industry-strength development environments)

  Java libraries

  Java platform based programming languages

Java Heritage

when Java was created, it:

- combined aspects of **Smalltalk** (single inheritance, garbage collection, byte code, exception handling)
- with the popular syntax of **C/C++**
- and added some new innovative features (interfaces)

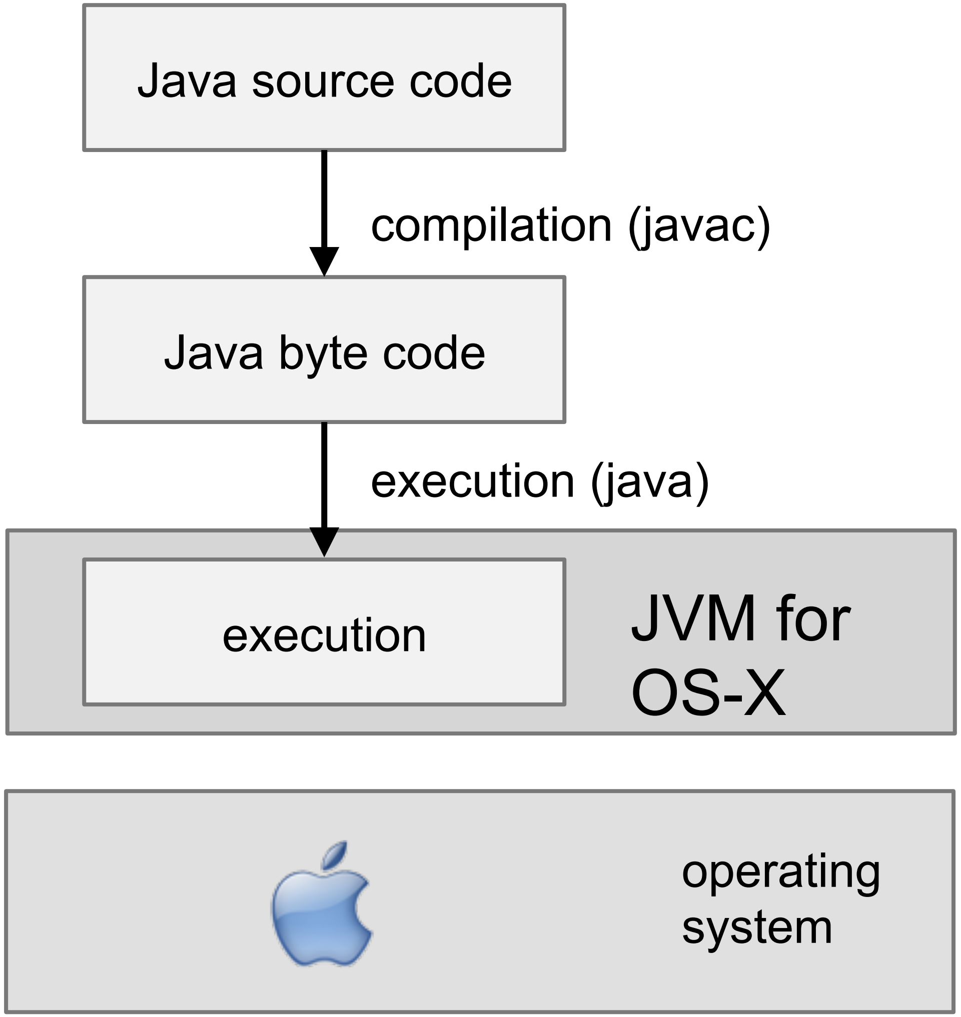
Compilation and Byte Code

- Java is both **compiled and interpreted**
- Java source code is compiled by the **Java compiler (javac)** to Java **byte code**
- the Java compiler performs **correctness checks**, in particular type safety checks
- the Java interpreter (java) performs additional checks and executes the byte code
- the execution environment is called the **Java Virtual Machine (JVM)**

The Java Virtual Machine (JVM)

- different JVM implementations for different hardware and operating systems
- **abstracts** from different hardware and OS
- acts as an adapter between the Java executable and the OS
- ensures that Java programs can run on all platforms (which are supported a JVM)
- "write once, run everywhere"
- the JVM performs further safety checks (bytecode verification) incl type checks when bytecode is loaded

Compilation, Byte Code and the JVM



Java source code

compilation (javac)

Java byte code

execution (java)

execution

JVM for OS-X

operating system

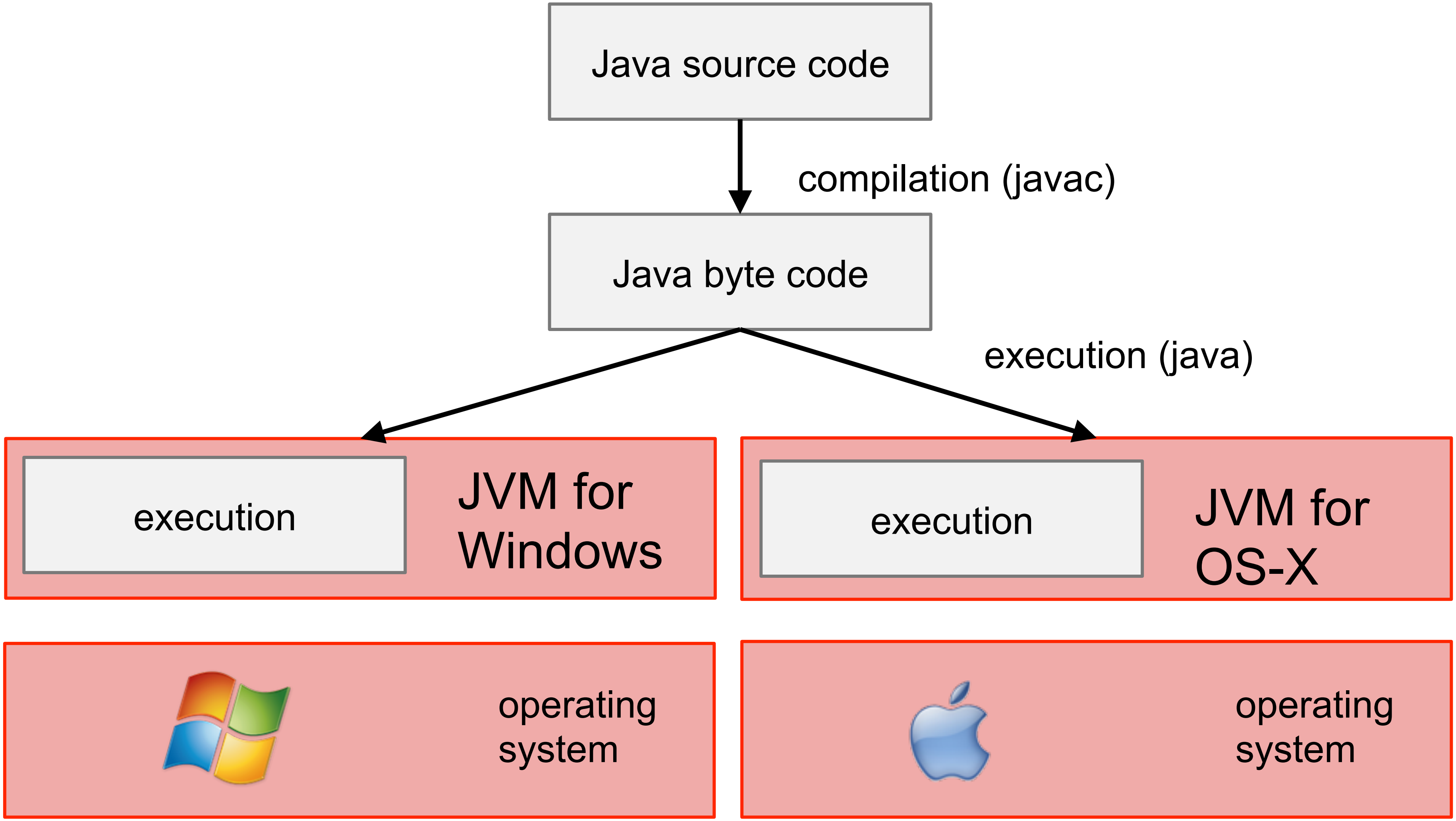
Standardisation and Implementations

- both the Java language and the JVM are standardised
- the standard can be found here: http://docs.oracle.com/javase/specs/
- there are multiple implementations of the Java language, the JVM and the compiler available
- the most popular implementation (language, JVM and compiler) used is the free and open source **OpenJDK**
- the OpenJDK project is lead by Oracle (after Oracle purchased SUN in 2010), and supported by IBM, Apple and SAP
- a major alternative Java implementation is Apache Harmony

Advantages
(of Standardisation and Architecture)

- run Java programs on any OS
- use the JVM with other languages
- use Java with other JVM

Run Java Program on Different OS



Java source code

compilation (javac)

Java byte code

execution (java)

execution

JVM for Windows

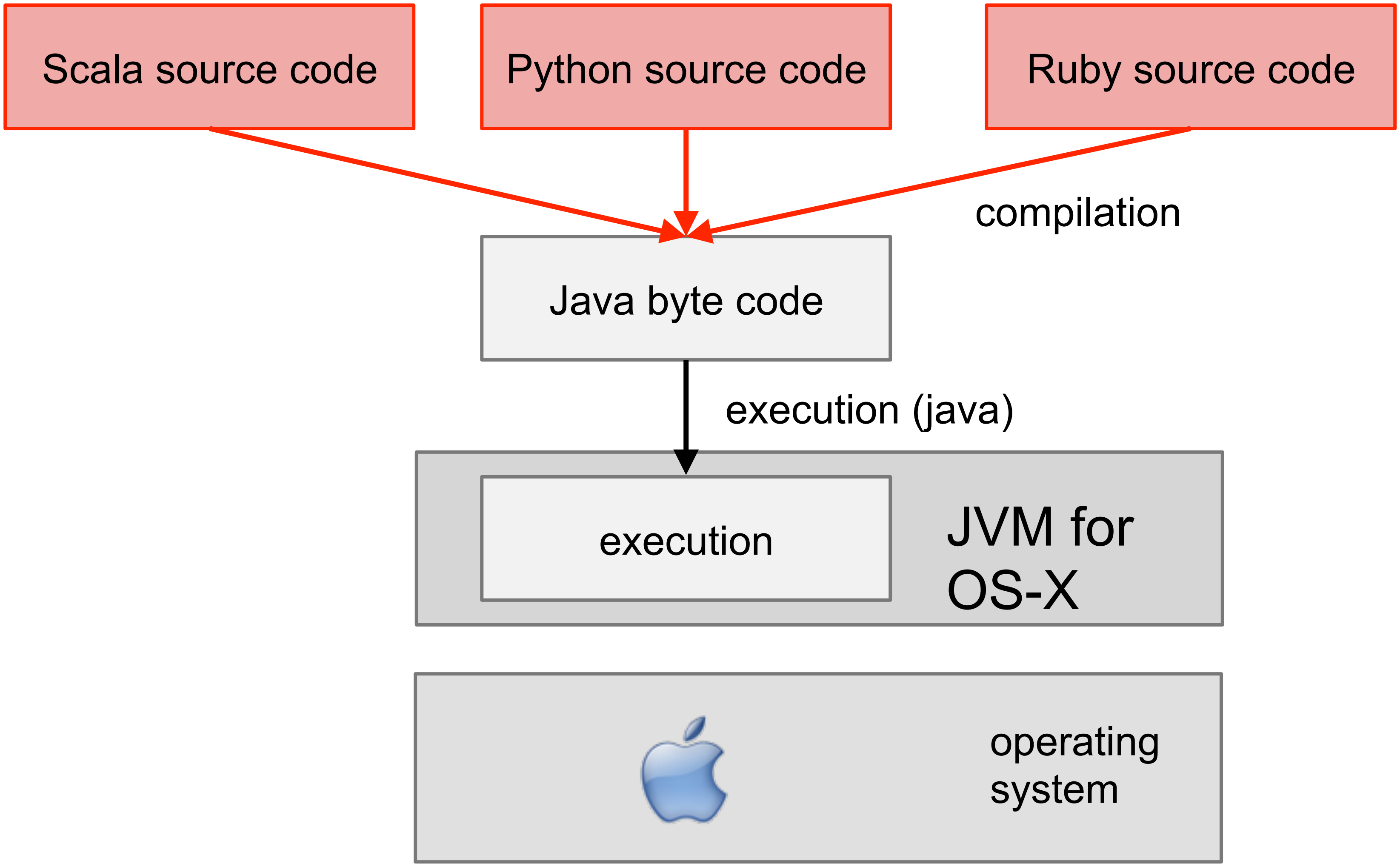
execution

JVM for OS-X

operating system

operating system

Use the JVM with Other Languages



Scala source code

Python source code

Ruby source code

compilation

Java byte code

execution (java)

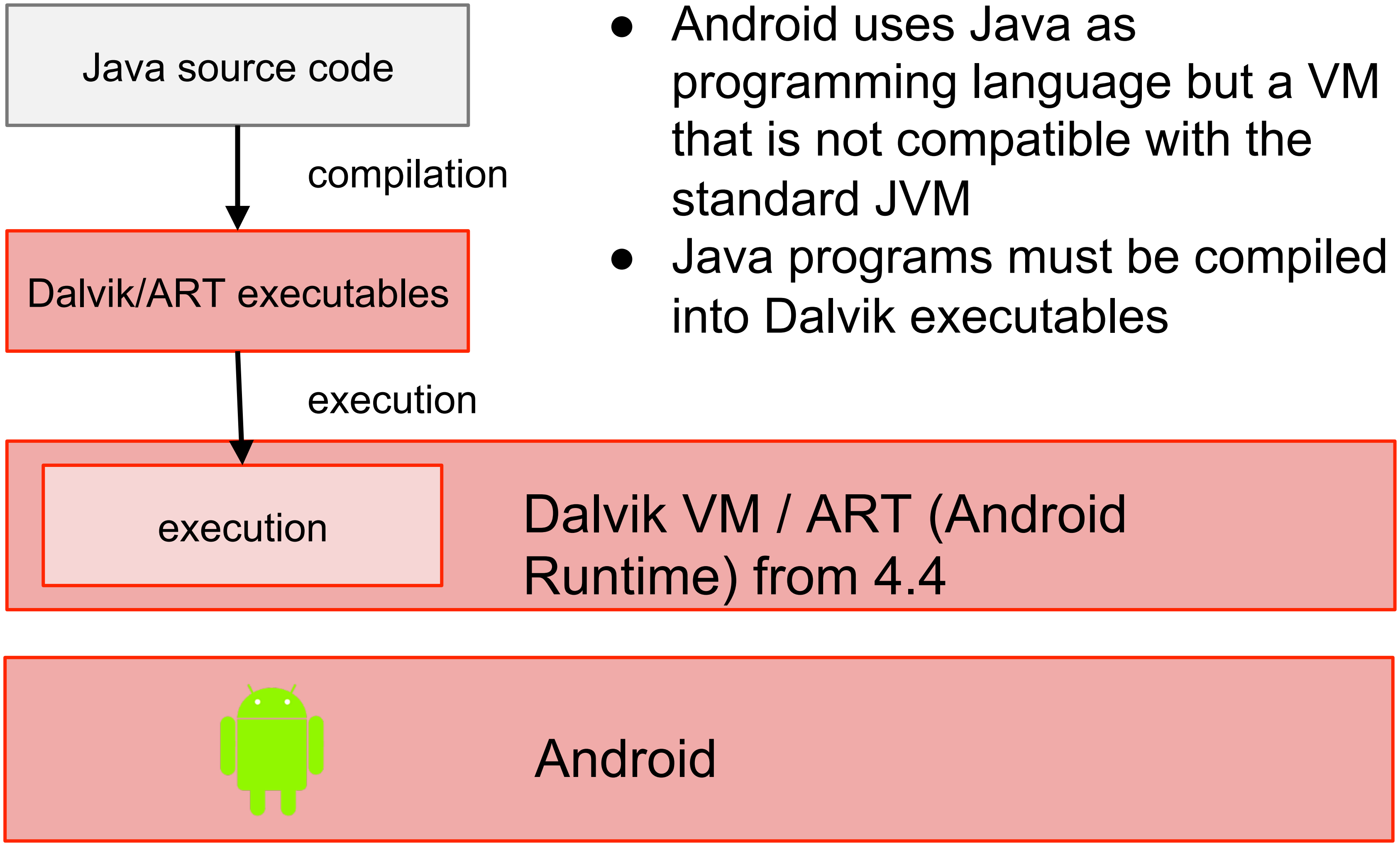
execution

JVM for OS-X

operating system

Use Java with Other VMs

- Android uses Java as programming language but a VM that is not compatible with the standard JVM
- Java programs must be compiled into Dalvik executables



Java source code

compilation

Dalvik/ART executables

execution

execution

Dalvik VM / ART (Android Runtime) from 4.4

Android

The Java Runtime Environment (JRE)

- aka **Java Platform**
- comprises two parts - **JVM** and the **Java API**
- the JVM is the bytecode interpreter
- the Java API is a large collection of ready-made software components that provide many useful capabilities, such as graphical user interface (GUI) components (aka widgets)
- the Java API is grouped into packages of related components.

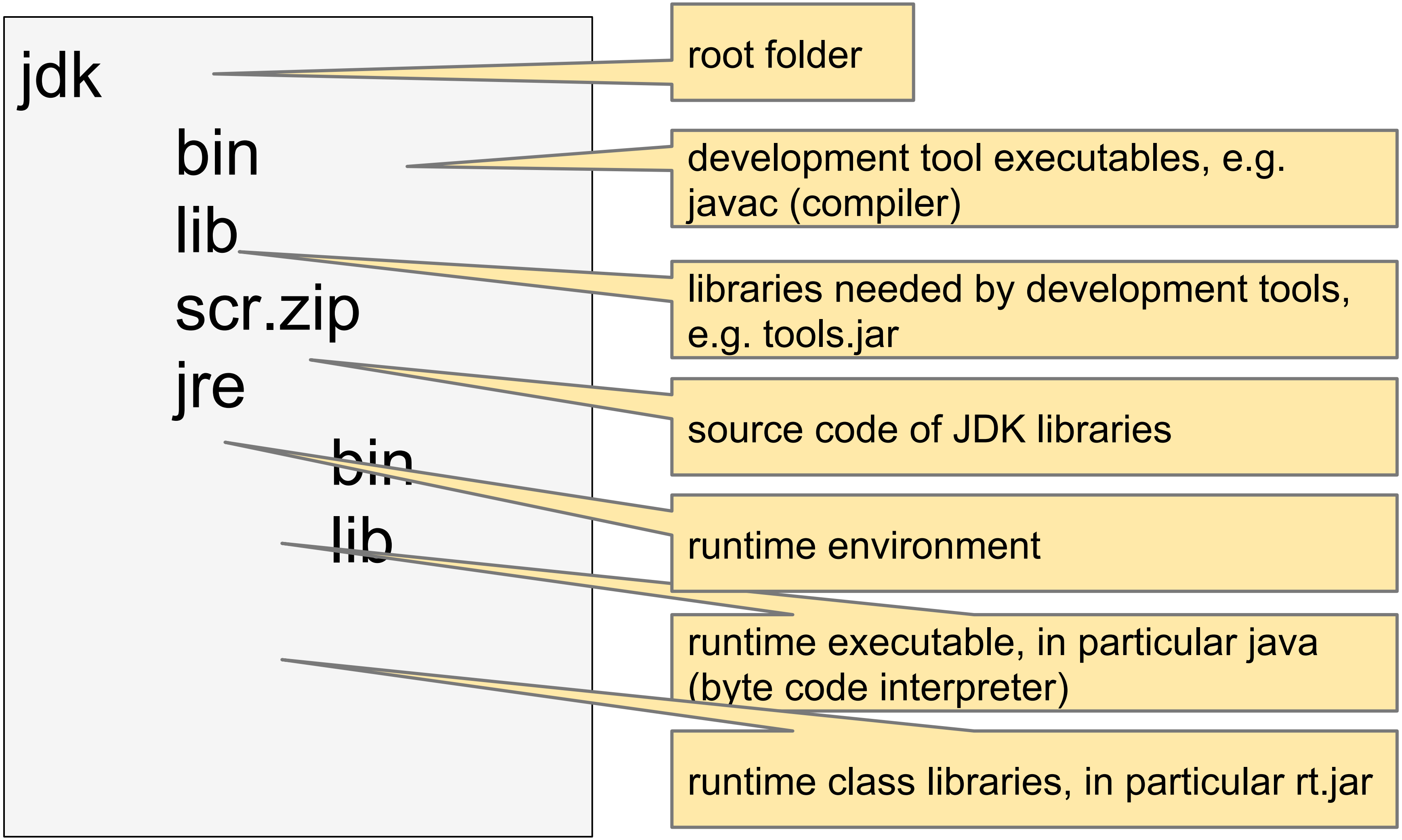
The Java Development Kit (JDK)

- superset of the JRE
- also contains tools for development, including:
    - **javac** - the compiler
    - **javadoc** - a code documentation tool
    - **jar** - a code packaging tool
    - .. and much more

JRE vs JDK - Practical Considerations

- you should download and install the JDK, not just the JRE for this paper
- Eclipse has its own compiler, and therefore requires only the JRE *- but install the JDK anyway - we might use it in some assignments!*
- the executables (javac, java, ..) are in bin folders within the JDK/JRE installation folders
- the JDK contains the JRE

JDK Installation Structure



jdk

bin

lib

scr.zip

jre

bin

lib

root folder

development tool executables, e.g. javac (compiler)

libraries needed by development tools, e.g. tools.jar

source code of JDK libraries

runtime environment

runtime executable, in particular java (byte code interpreter)

runtime class libraries, in particular rt.jar

Hello World !

```
/**
* Hello World example.
* @author Jens Dietrich
* @version 1.0
*/
public class HelloWorld {

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

HelloWorld.java

this is a multiline comment

special tags @author, @version - useful later when generating web pages from comments

a program consists of classes, the name must be the same as the file name

classes with such a main method are "executable"

System.out is a reference to the console window

the file name must be the same as the class name, the file extension is java

Command Line Compilation

```
javac HelloWorld.java
```

- execute this program **terminal** (OS X, Linux) / cmd window (windows)
- assumptions:
    - javac is in the current folder or in **PATH**
    - HelloWorld.java is in current folder
    - for options, execute javac without parameters
- this will generate **HelloWorld.class**
- this file contains the (executable) byte code

Command Line Execution

```
java HelloWorld
```

- execute this program **terminal** (OS X, Linux) / cmd window (windows)
- assumptions:
  - java is in the current folder or in **PATH**
  - HelloWorld.class is in current folder
- if an error occurs that the class cannot be found, try this: java -cp . HelloWorld
- the option “-cp .” instructs java to explicitly look for classes in the current directory

Inspecting Bytecode

```
javap -c HelloWorld.class
```

- uses the disassembler javap
- prints bytecode instructions
- these are the instructions the JVM can interpret
- the structure of the bytecode is outside the scope of this paper !

HelloWorld Bytecode

```
Compiled from "HelloWorld.java"
public class HelloWorld {
  public HelloWorld();
    Code:
    0: aload_0
    1: invokespecial #1
      // Method java/lang/Object."<init>":()V
    4: return
   public static void
      main(java.lang.String[]);
    ...
```

Packages

- Java programs are written by defining classes
- usually, one class corresponds to one source file
- with inner classes, one file can sometimes contain multiple classes (to be discussed later)
- typical applications consist of thousands of classes:
  - tomcat 7.0.2, a web server: 2390 classes
  - azureus-4.5.0.4 (aka vuze), a torrent client: 7713 classes
  - note that this does not include libraries these applications use: the actual number is much higher!
- this creates a need to organise classes

Namespacing

- one particular problem are name clashes
- there can only be one class with a certain name
- if applications are created by combining classes from different sources, there is a chance that the same name is used twice
- to avoid this, classes are organised in packages
- packages serve as name spaces: the full class name used to identify a class is the package name plus the local name

Packages ctd

- the package names guarantee that classes have unique names
- this facilitates building programs by using classes from different sources ("assemble applications like lego")
- Java names can be (almost) arbitrary strings
- further conventions are needed to control uniqueness of package names
- the convention is to take advantage of unique internet domain names, and use them as package name
- usually, domain names are reverted, and additional parts identifying projects, products or organisational substructures are appended

Package Name Example

domain name:

`massey.ac.nz`

package name =
reverted domain name + additional tokens:

`nz.ac.massey.cs.oobasics`

Hello World 2 !

package declaration

```
package nz.ac.massey.cs.oobasics;
/**
* Hello World example.
* @author Jens Dietrich
* @version 1.0
*/
public class HelloWorld2 {

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

HelloWorld2.java

Compiling and Running HelloWorld2

- `java HelloWorld2` fails (NoClassDefFoundError): there is no such class !
- the class name is `nz.ac.massey.cs.oobasics.HelloWorld2`
- this is also called the **fully qualified class name**
- **but** `java nz.ac.massey.cs.oobasics.HelloWorld2` also fails: the class cannot be found
- `java expects HelloWorld2.class` in the subfolder `nz/ac/massey/cs/oobasics` (see note)
- the hierarchical package structure corresponds to the hierarchical structure of the file system
- after creating these folders, and copying `HelloWorld.class` into this folder, the application can be executed

Using Classes in Other Packages

- example: use **java.util.Date** - represents timestamps
- once this class is available, an instance (object) can be created by invoking the constructor new Date()
- this object represents the current date and time

options to work with Date:

1. use the fully qualified name directly:
2. import class
3. import package
4. exception: core classes like String and Object are in the core package **java.lang** - this package does not have to be imported

Option 1: Use Fully Qualified Name

```
public class PrintCurrentTimeAndDate {
       public static void main(String[] args) {
      java.util.Date now = new java.util.Date();
              System.out.println(now);
    }
}
```

this is the **constructor** used to instantiate the Date class

Option 2: Import Class

```
import java.util.Date;
public class PrintCurrentTimeAndDate {
       public static void main(String[] args) {
       Date now = new Date();
              System.out.println(now);
   }
}
```

now only the local name is required

Option 2: Import Entire Package

the * serves as a **wildcard**

```
import java.util.*;
public class PrintCurrentTimeAndDate {
       public static void main(String[] args) {
       Date now = new Date();
             System.out.println(now);
   }
}
```

now only the local name is required

Jar Files

- as discussed, the number of classes in real-world applications can be large
- each class is typically very small (HelloWorld.class = 425 bytes)
- this make the storage of classes and the network transfer very inefficient
- solution: zip many classes stored in a folder together using the standard zip algorithm!
- this is called a jar file!
- jar files also contain metadata (manifests) with additional info about the classes contained, such as version info, authors etc

The Classpath

- typical Java applications consist of multiple jar files and class files stored in folders
- the list of these directories and jars is called the **classpath**
- to start a java application that uses additional classes from jar files, use the -cp option:

```
java -cp <list of jars and folders> main-class
```

Integrated Development Environments

- Integrated Development Environments (IDE) make working with source code convenient and productive
- Java has extremely good IDE support
- there are three major IDEs (Eclipse, NetBeans, IntelliJ)
- Eclipse and NetBeans are free
- all IDEs are plugin-based and therefore highly customisable
- there are plugin ecosystems - this keeps prices low and quality up

Core IDE Functions

- project management
- work with source code (auto-completion, syntax highlighting, formatting, ..)
- keep track and annotate errors
- refactoring (systematically change code)
- debug
- interact with remote code repositories
- provide visual user interface builders
- reformat (pretty-print) code
- ...