JAVA Programing

Author: Stephane Sintes

stephane.sintes@renault.com
stephane.sintes@gmail.com

10002066 Version:1.0

Stephane Sintes

System SW Architect

2 years at Renault - **SW specialist in AD/ADAS Platform Vehicle at RNSL**

4 years at Intel - **System SW architect - Audio**
Mobile – Tablet – PC – (Windows – Android - Linux)

Based in France, Nice

15 years in TI - **Senior Member of Technical Staff (SMTS)**

- OMAP6 Chief System SW Architect 2010-2013
- OMAP4 Chief System SW Architect 2007-2010
- Lead the WW OMAP System SW architecture (10-17 people) (2011-2012)
- Lead the OMAP SW architecture in Nice (13 people) (2006-2010)
- Lead the OMAP3430 Advanced development team in OMAP System SW Architecture (2005) (3 people)
- ARM & DSP Performances Technical Leader in OMAP System SW Architecture 2004-2005 (3 people)
- Audio & Music Technical Leader in OMAP Software Architecture (2002-2003)
- OMAP Software Development 1999 – 2002
- Audio Software Development 1997 – 1998
- Engineer from SUP Telecom Bretagne & EURECOM (Cellular phone Option)

Teaching and Expertize

University Teacher (19 years) / Coaching

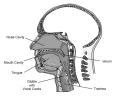
- Java teacher at Polytech Sophia (2016-2020)
- Lead Autonomous Driving Course - Polytech Sophia (2018-2020)
- Member of Intel Universities program (2015-2017)
- Speech processing teacher at university of Nice and Polytech Sophia (engineer school) (2001-2011)
- OMAP Multimedia teacher in ESSI (engineer school option embedded system) (2004-2012) & Polytech Sophia (2012) Campus ID (2011-2012)

Expertise

- System Expert

- Speech Vocoders

Compresses the voice by synthesis the speech



- High Fidelity Audio

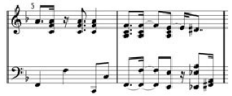
MP3

MPEG4-AAC

Compresses the song by mainly uses ear limitation

- MIDI

Compresses the music by synthesis the notes



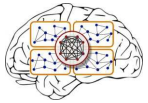
- Video

H264 Enc /Dec

WMV9 Decoder

- Autonomous Driving

- Deep Neural Network



Interactive questions Audio-Video-Speech

- How MP3 compression is working?
  - frequency masking
- How video compression is working?
- Do you know MIDI? How does it work?
- How speech codec work?

JAVA Course Organization

- Java Slides-set in English
- Java Course in French or English
- Java Course is a mix of main course and Lab work
  - We need to adapt with the Covid-19 – we will try to have the session interactive
  - Support for remote lesson is through Microsoft Team application
- Goal is also to understand also the industrial work and how we are doing SW programing
  - Concrete example will be given on my work at Texas Instruments, Intel or RN

Round Table

- Present yourself
- What work would like to do?
  - SW versus HW / marketing
  - Management versus Developer
  - Romain C Dev – AD – Var (recherche 2 mois / alternance)
  - Jeremie Pietri – SW – dev – Nice (2 mois University Islande)
  - Brandon SW – dev – Antibes 2 mois University PayBas)
  - Quentin – Bas Niveau SW (driver) – HW – La Gaude
  - Wei Jiang SW – Chine –- SW – dev
  - Hamza.Benmalek SW – Maroc - Nice SW – dev (recherche 2 mois / alternance)
  - Benjamin C Nice SW – dev / Management
  - Marie – Clermont HW/SW - recherche 2 mois / 6 mois Management
  - Philippe Management SW – dev Marne-la - recherche 2 mois / 6 mois Management
  - Camille – SW/HW - recherche alternance Rouret
  - Auguste SW – dev puis management Nice recherche 2 mois / 6 mois
  - RG – Chine SW – dev Nice recherche 2 mois / 6 mois
  - Ricardo Perou Sophia SW/HW dev puis management recherche 2 mois / 6 mois
  - Benoit SW – dev 2 mois Synergie Cad recherche 6 mois La Gaude
  - Paul T - SW – instrument de musique (2 ou 3 mois Madrid) Gatiere
  - Mathis Thailand - SW dev Dijon (recherche 2 mois / alternance)
  - Sean Mcgannon – HW-SW – Robotic – (recherche 2 mois / alternance) Nice

Agenda (1/5)

- Stephane Presentation
- Round Table
- JAVA Course Organization
- Agenda
- Java Labs & Android Labs
- Java History
- Java Features
- Java Platform Editions / Java Environment/ JDK
- Compiling and Executing C program
- Compiling and Running Java program
- Java Main Method
- First Java programs
- Compiling & running Java
- Java Virtual Machine
- Executing a Java program
- Java bytecode Definition
- Disassembling Java bytecode
- JVM Runtime Data Area
- JVM Runtime Data Area Multithread
- Java Virtual Machine Stack
- Frame & Memory example
- Just-In-Time compilation

Agenda (2/5)

- Object-Oriented Design
  - What is an Object?
  - Class vs Objects
  - Class Attributes Methods Interface
  - UML Unified Modeling Language
- Encapsulation
- Inheritance
- Abstraction
- Polymorphism
- Interface
- Implementing Interface
- Interface versus Abstract
- Eclipse
  - Eclipse – Windows
  - Eclipse File/Class Creation
  - Eclipse - Auto
- Java Basics
- Java Comments

Agenda (3/5)

- Package
  - What is a package?
  - Java built-in packages (Java API)
  - Creating a package
  - Package naming convention
  - Using package members
  - Package Usage
- Primitive data types
  - byte primitive data type
  - int primitive data type
  - float primitive data type
  - boolean primitive data type
  - const
- String
- Print and Println method
- MATH - Basic math functions
- Java Variables
  - Variable Declaration
  - Instance variables
  - Local variables
  - Class variables
  - Getters and Setters
  - Variable naming convention

Agenda (4/5)

- Constructor and Methods
  - Constructor
  - Methods
- Override Method
- Continue Keyword
- Java versus C++ comparison
- Javac options
- JAVAC Xlint
- JAVAC non standard option
- Class Files in Specific folder
- UML Class Diagram
- Arrays
  - Array (1 dimension)
  - Array (2 dimensions)
- JAVA Graphical User Interface
- Package javaawt
- AWT Components Hierarchy
- JME

Agenda (5/5)

- Introduction to Android Applications Development
- What is Android?
- Android Java usage
- Android versions usage share
- Android market share
- Android Architecture
- Android Software Development Kit
- Android Library
- Android Applications The Basics
- Application components Activities
- Application components Services
- Application components Content Providers
- Application components Broadcast Receivers
- Activating application components
- The Android Manifest File
- Application Resources
- The Activity Lifeycle onCreate()

Labs Work expected - Eval

- Labs -
  - Documentation/Report in English
  - Code
  - **Sent the package to: stephane.sintes@renault.com**

- Eval – Subject in English
  - You can answer in English or in French
  - No lesson/support authorized
  - Mixed of QCM, Question and Exercises
  - Need to adapt with the Covid-19

Java Labs

- Lab 1: Java Environment setup
  - **Java Development Environment Setup**
  - **first java application: "Hello New World!"**
  - **javac, java, javap**
  - **Eclipse Usage**
  - **Javadoc Usage**
- Lab 2: My first Java classes
  - Rectangle / Circle / Static member
- Lab 3: Inheritance and Polymorphism
- Lab 3B : Package
- Lab 4: Control flow statements
  - if
  - for
  - while
  - switch
- Lab 5: Be Autonomous - Calendar Program
  - Eclipse with debugger
- Lab 6: Arrays - catch of errors
- Lab 6B: Regular Expression
- Lab 7: Unit Testing

Android Labs

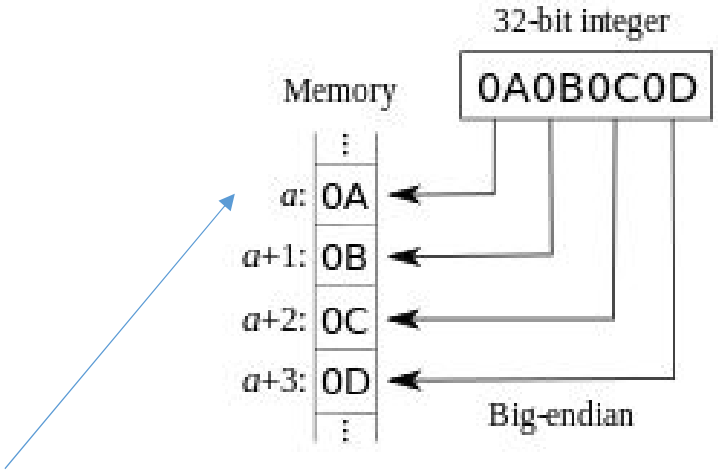
- Lab 7: My First Android Application – Hello World!
- Lab 8: Image Capture and Filtering Application

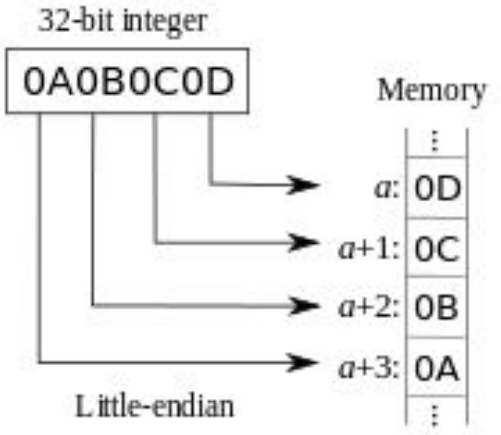
Java History Summary

- Developed in 1991 by Sun Microsystems ("Green Team" led by James Gosling) while looking for green fields in consumer electronics market.
- Named oak and finally Named JAVA after a coffee break…
- Java is not be the most common name for coffee, but it is only name that has inspired a computer programming language.
- Objective: enable a heterogeneous network of consumer electronic devices to communicate with each other (interactive television). Develop a platform agnostic language.
  - Failure
  - Equivalent to IOT
- First public release in 1995, JDK1.0 release. Beginning of internet, Netscape Navigator embeds Java technology.
  - "write once, run anywhere"
- 1999, Java 1.2 or "Java 2"
- 2010 Sun Microsystems acquired by Oracle.
- 2011, Java Standard Edition 7 (JDK 1.7)
- Download Java : http://www.oracle.com/technetwork/java/index.html
- 10 Millions of Java developers

Java features - Portable / Platform agnostic

- Portable / Platform agnostic
  - Java is Platform independent so it has to be Robust and Secure
  - Agnostic, The size of the primitive type int is a 32-bit signed integer from -2^31 to 2^31-1 whether the Java Virtual Machine (JVM) is running on a 16bit/32bit/64bit Operating.
  - JVM is big-endian => high bytes first (you store the most significant byte in the smallest address)
    - Host OS and CPU can be little endian but the JVM is big-endian





high bytes come first while the @ order increase

Java features - Object-Oriented

- Java is not a pure Object oriented language, more an "Hybrid" language.
- Object-Oriented
  - Fully Object-Oriented OO but not pure
    - Encapsulation (yes) hiding/isolating data
    - Inheritance (yes)
    - Polymorphism (yes)
    - Abstraction (yes)
    - User defined types must be Objects (yes)
    - All predefined types must be Objects (no)
      - Decision on purpose, did not take an extreme approach :Everything is an object
      - Java has wrapper classes
    - All operations performed on objects must be **only** through methods exposed at the objects (no)
      - String redFlag = “Red" + “Flag" ;
        - We are using + operator but not a method
  - C# / Visual Basic and C++ are also Fully Object-Oriented Languages
  - Pure Object language : Python, Ruby, SmallTalk

Interactive questions – Interpreted Language

- What is an interpreted language?

Interpreted Language

- Interpreted programs is not compiled
  - Compiled: The code is transformed in the instructions of the target machine (x86, ARM …)
  - Compiled code has faster performance
- An interpreted code can modify itself by adding or changing functions at runtime.
- Perl and Python are interpreted language
  - Rapid prototyping
  - Don't have to recompile your code each time you want to test
- MATLAB is also an interpreted language for Equations
  - In RN we are using Matlab/Simulink for Model generation

Interactive questions – Multi Thread

- Multi-Threaded?
- Is it difficult to make Multi-Threaded code?

Java features - Multi-Threaded – Robust - Interpreted

- Multi-Threaded
  - Each of the threads can run concurrently
  - Strong support for multi process synchronization
- Robust
  - Powerful Java compiler (early checking for possible errors)
  - Interpreter for runtime error (Compiler checks the program whether there any static error and interpreter checks any run time error)
  - Strong memory allocation (Strongly typed – no pointer access)
  - Garbage Collection
  - Crash Recovery – Trapping error - exception handling (Lab-6)
- Interpreted
  - bytecode is interpreted
  - Just In Time (JIT) compiler
    - Translates bytecode into native code very efficiently
    - At run time with the condition (example we know we are making a 90 degree rot) the code is optimized, all not used code is removed, it is like we have written a dedicated code that makes a 90 degree rotation

Interactive - Programming Language

- How to choose a specific programming Language?
  - C++/Java/Perl/Python/C/C#/ADA
- Make the slide All together

Java features - Automatic Memory Management

- Automatic Memory Management
  - Opposite to Explicit memory management – less prone to error
    - memory management is the programmers responsibility
  - Reduced Debugging time => key feature
  - No Dangling memory:
    - deallocate the space used by an object but the other objects still have a reference on it => unpredictable result
      - A new object can have been allocated In the same place
    - Dangling pointer
  - No space leak
    - Linked list example - remove first element of the list – rest of the list is not referenced and cannot ne accessed
  - Garbage collector (time / fragmentation)
    - Garbage collector is a program
    - Object that is still referenced (live object) will never be garbage collected
      - Garbage collection avoids the dangling reference problem, because an object that is still referenced somewhere will never be garbage collected and so will not be considered free.
    - No space leak
      - GC automatically frees all memory no longer referenced (dead object)
    - Garbage collection = finding and freeing reclaiming dead object
    - Garbage collection is an intensive & complex task taking time and resources
      - Tradeoffs have to be made between time of execution and Fragmentation
    - Memory Allocation/DesAllocation is done by the Garbage collector

Java features – Secure - Architecture neutral

- Secure
  - Strongly typed – not allow implicit conversion
  - No pointer access, no arithmetic on pointer
    - pointers directly access the memory of your device so any erroneous/malicious program can infect your data of your computer
  - No pointer visible for end user but Java use them Internally, a reference to an object is implemented as a pointer
  - Bytecode verifier
- Architecture neutral
  - No impact on CPU changing
  - No impact on OS upgrade
  - =>longevity along with portability

Java features – Distributed - Dynamic

- Distributed
  - Distributed- Java is designed for distributed environment of the Internet because it handles TCP/IP protocols.
  - Java supports RMI (Remote Method Invocation) enabling programs to invoke methods across a network.
    - The Java RMI system allows an object running in one JVM to invoke methods on an object running in another JVM.
    - Very useful when you have to split the execution in several computer
- Dynamic
  - Loads the class files at run time. Anything that happens at run time is considered as Dynamic.
  - run-time Information used in verifying and resolving access to objects at runtime
  - Object type are known at run time

Java features – Simple - Widely used

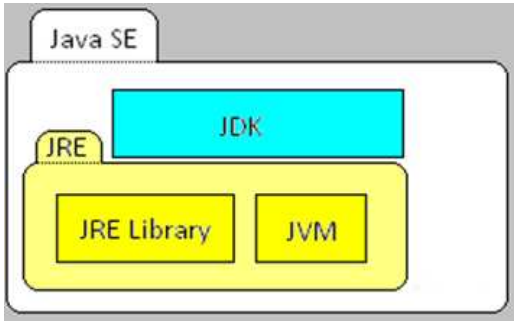
- Simple: Easy to learn / Easy to re-use
  - Was in the Java requirement Design
  - Java inherits C/C++ syntax and object oriented concepts of C++ => more easy for classical programmer
  - Up to you to confirm
- Widely used (Web, Desktop, Mobile devices, …)
- Java is not the same has JavaScript (HTML document)

Java Platform Editions

There are several Java Platform Editions:

- Java Standard Edition (JSE) – our Focus
  - Aimed at desktop application development
  - Client side : UI or console based
- Java Enterprise Edition (JEE)
  - Aimed at enterprise software development
  - E.g. Network applications, Web services, …
- Java Micro Edition (JME)
  - Targeting embedded systems.
  - MIDP (Mobile Information Device Profile): JME profile for mobile devices, cell phones.
  - Not used on latest smartphones: Android, iOS, etc…

Java Environment

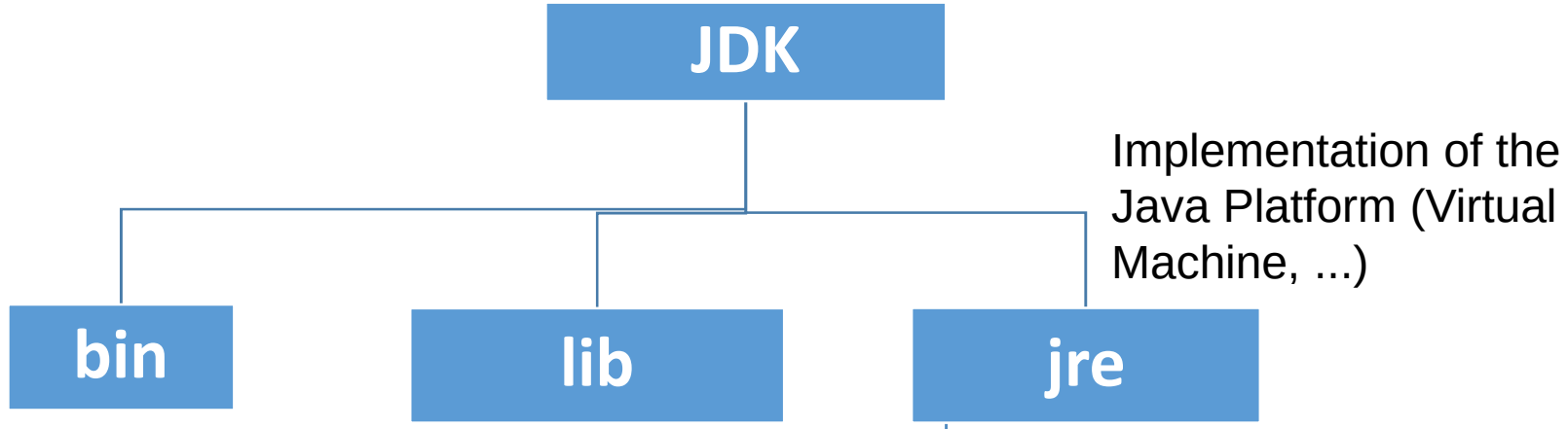


- JRE : Java Runtime Environment
  - Aimed at Java users: needed to support Java programs execution
  - Java API + Java Virtual Machine (JVM)

- JDK : Java Development Kit
  - Aimed at Java developers: needed to develop Java programs.
  - Includes Tools + Libraries + JRE

The JVM is the core component of the Java Runtime Environment

JDK directory tree (simplified)



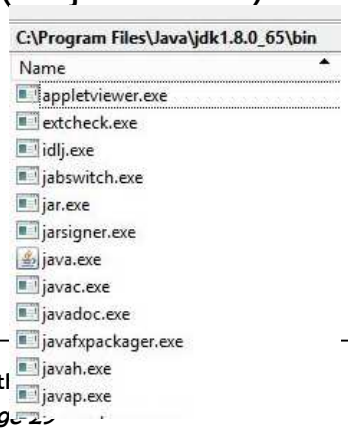
Implementation of the Java Platform (Virtual Machine, ...)

Executable files for the development tools contained in the Java Development Kit. (i.e javac.exe)

Libraries used by the development tools (took.jar)
In tools.jar are all tools that are needed by a JDK

Executable files and DLLs for tools and libraries used by the Java platform.

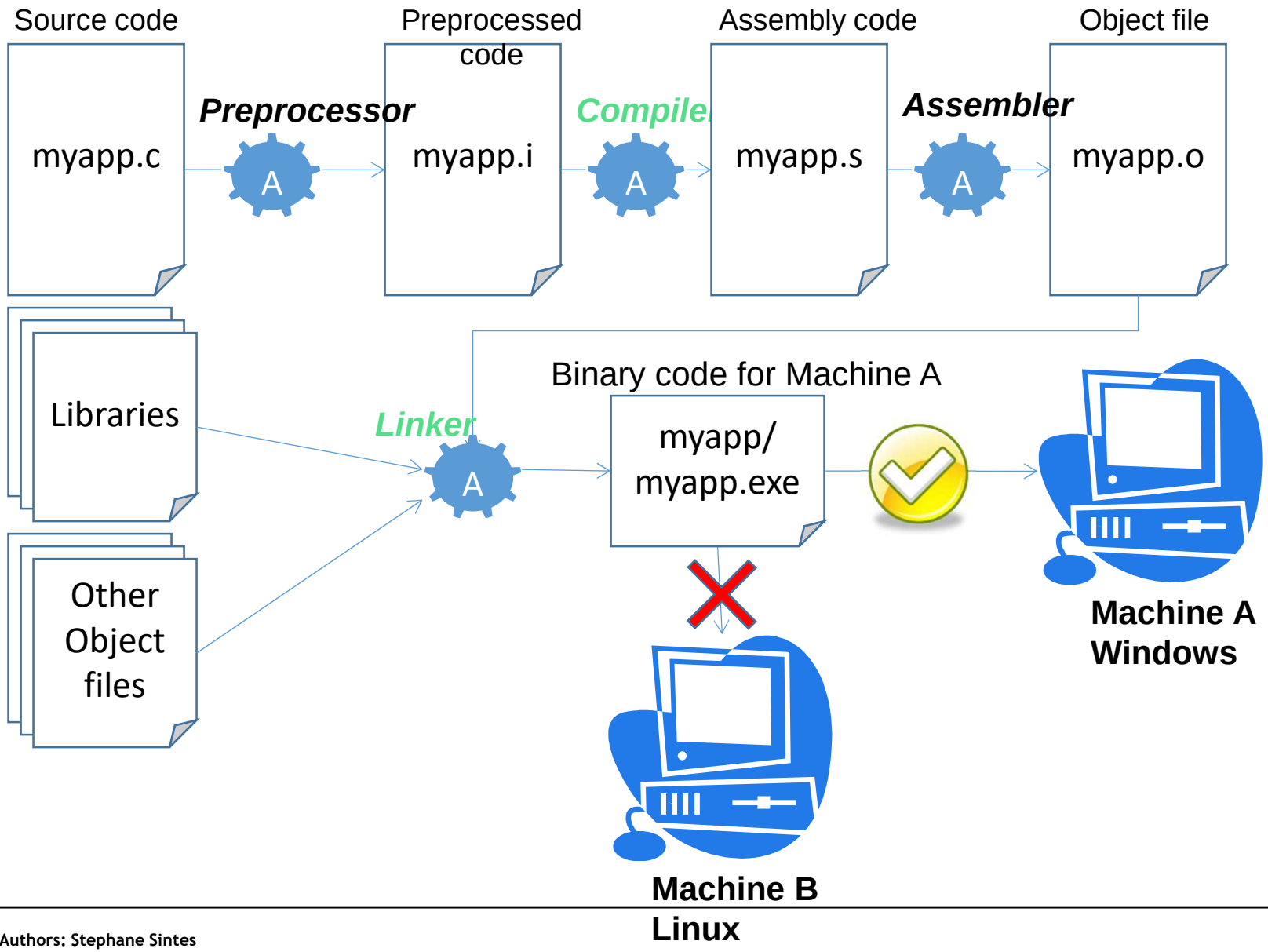
Code libraries, property settings, and resource files used by the Java runtime environment



Basic JDK tools

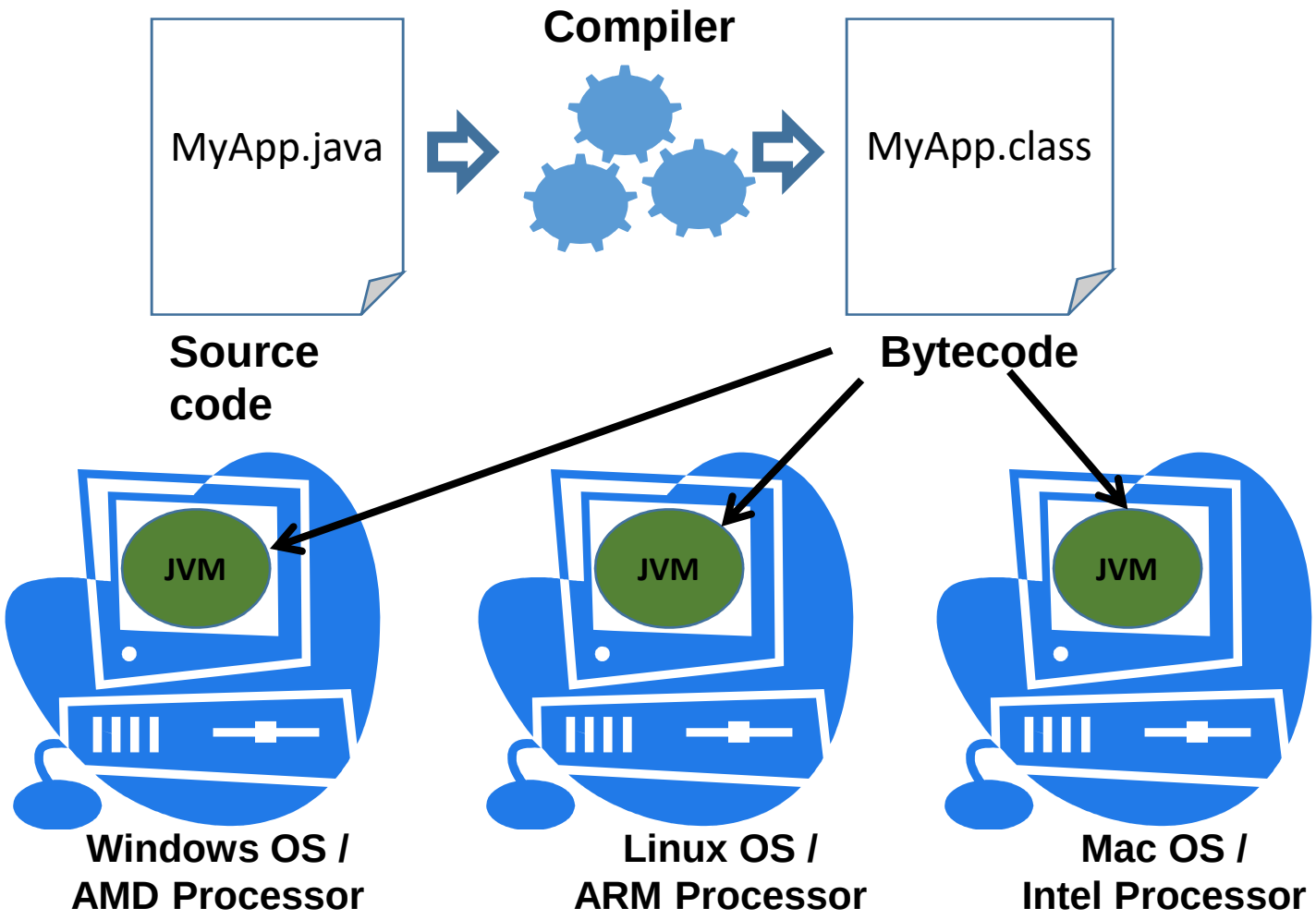
- ***javac*: Java programming language compiler**
  - This tool compiles Java source code files (.java) into Java bytecode files (.class)
- ***java*: Java application launcher**
  - This tool launches a Java application.
  - Start Java runtime environment which contains the JVM, load specified class and invoke its main method.
- ***javap*: Java bytecode disassembler**
  - This tool disassembles a class file.
- ***javadoc*: Java API documentation generator**
  - This tool generates HTML pages containing API documentation from Java source files.
- ***jar*: Java archive tool**
  - Combine multiple files into a single JAR archive file (Package)

Compiling and Executing C program



Compiling and Running Java program

WORA:
Write **O**nce,
Run
Anywhere



Lab 1

- Lab 1: Java Environment setup
  - **Java Development Environment Setup**
  - **first java application: “Hello New World!”**
  - **javac, java, javap**
  - **Eclipse Usage**
  - **Javadoc Usage**

Java Main Method

The main method provides the control of program flow. The Java interpreter executes the application by invoking the main method.

The main method looks like this in your file:

```
public static void main(String[] args) {
  // Statements;
}
```

when you compile your java file, you generate .class files

You can have several .class files, but only one needs to contains the Main() method.

First Java programs

```java
import java.io.*;

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

HelloWorld.java

```java
import java.io.*;

public class Adder{

    public int add(int a, int b){
        int sum = a + b;
        return sum;
    }

    public static void main (String args[]){
        Adder mAdder = new Adder();
        int sum = mAdder.add(6,7);
        System.out.println("Sum = "+sum);
    }
}
```

Adder.java

Compiling & running Java

Compiling and running Java program

- Compiling *HelloWorld* program:

  >C:\ javac HelloWorld.java

  => bytecode generated in HelloWorld.class

- Running *HelloWorld* program:

  >C:\ java HelloWorld

Interactive questions – Virtual Machine

- What is a virtual machine?

Virtual Machine (VM)

- Software program emulating a physical computing environment (e.g. a CPU)
- Running as an application on a “Host” operating system
- Abstract details of the underlying hardware platform
- Enable applications portability over multiple hardware platforms/operating systems

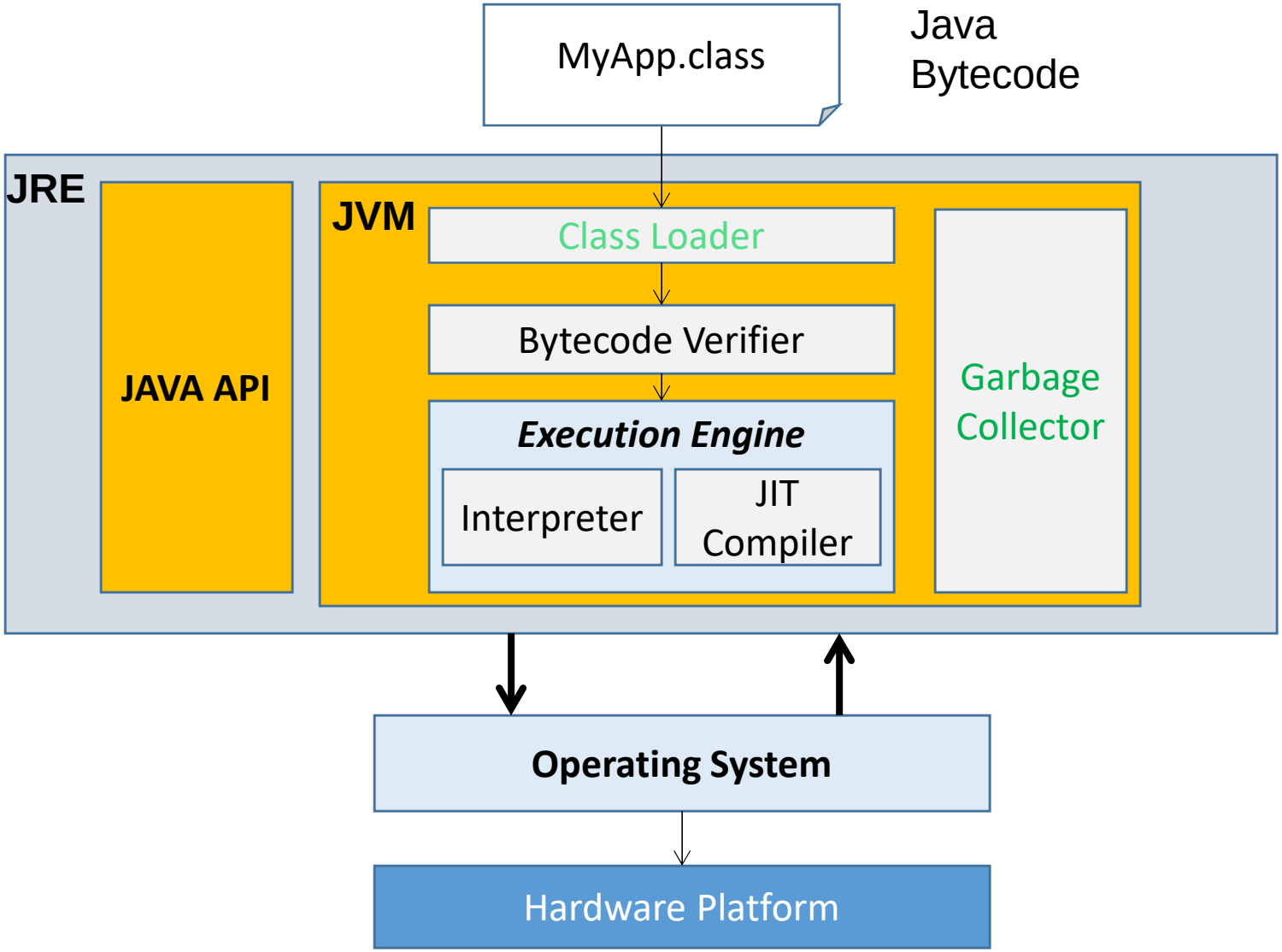
Java Virtual Machine (JVM)

- The JVM is the core component of the Java Runtime Environment
- VM interpreting/executing Java bytecode
- JVM specification from Sun/Oracle (here)
- Different JVM implementations available.
  - Examples:
    - Sun/Oracle's Hotspot JVM (see here)
    - IBM's JVM (see here)
- Support of various operating systems and hardware platforms (Linux/Win/Mac – Intel/ARM/Power PC/AMD…)
- Features
  - **Stack-based virtual machine** (vs. register-based in Android)
  - Garbage collection (automatic memory management)
  - Clear specification of primitive data types
  - Symbolic reference for classes and interfaces (Constant Pool)
  - Network byte order (platform independence) (Network byte order =big-endian)
  - Byte code interpretation
  - Just-In-Time (JIT) compilation => language machine => cache
  - …

JAVA – JVM - Applications

- In General one application – one JVM even if it possible to run several application in the same JVM (Java Application Servers)

Executing a Java program

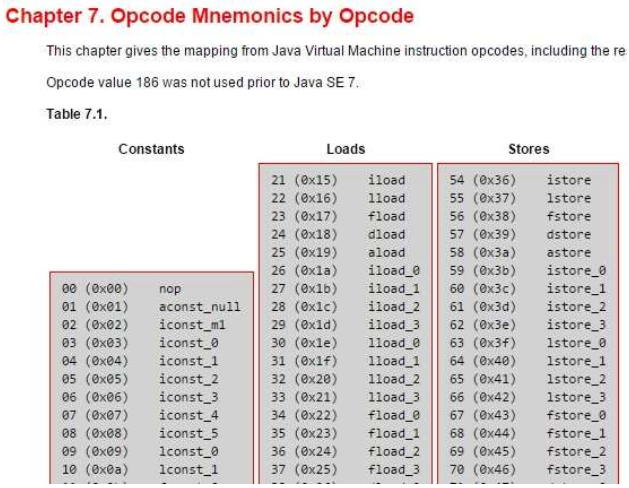


Bytecode Verifier

- The JVM verifies all bytecode before it is executed. This verification consists primarily of three types of checks:
  - Branches are always to valid locations (inside the current Method)
    - Checked During the Load
  - Data is always initialized and references are always type-safe
    - Checked During the Load
  - Access to private or package private data and methods is rigidly controlled
    - Checked data items or methods are first accessed by another class.

Java bytecode Definition

- The Instruction Set:
  - 1 byte per instruction => « bytecode » or opcode: operational code
    - 256 bytecodes : 204 bytecodes currently in use
  - zero or more bytes for passing operand parameters
  - All Mnemonics and associated Opcodes: http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-7.html



Chapter 7. Opcode Mnemonics by Opcode

This chapter gives the mapping from Java Virtual Machine instruction opcodes, including the re

Opcode value 186 was not used prior to Java SE 7.

Table 7.1.

Constants		Loads		Stores	
		21 (0x15)	iload	54 (0x36)	istore
		22 (0x16)	lload	55 (0x37)	lstore
		23 (0x17)	fload	56 (0x38)	fstore
		24 (0x18)	dload	57 (0x39)	dstore
		25 (0x19)	aload	58 (0x3a)	astore
		26 (0x1a)	iload_0	59 (0x3b)	istore_0
00 (0x00)	nop	27 (0x1b)	iload_1	60 (0x3c)	istore_1
01 (0x01)	aconst_null	28 (0x1c)	iload_2	61 (0x3d)	istore_2
02 (0x02)	iconst_m1	29 (0x1d)	iload_3	62 (0x3e)	istore_3
03 (0x03)	iconst_0	30 (0x1e)	lload_0	63 (0x3f)	lstore_0
04 (0x04)	iconst_1	31 (0x1f)	lload_1	64 (0x40)	lstore_1
05 (0x05)	iconst_2	32 (0x20)	lload_2	65 (0x41)	lstore_2
06 (0x06)	iconst_3	33 (0x21)	lload_3	66 (0x42)	lstore_3
07 (0x07)	iconst_4	34 (0x22)	fload_0	67 (0x43)	fstore_0
08 (0x08)	iconst_5	35 (0x23)	fload_1	68 (0x44)	fstore_1
09 (0x09)	lconst_0	36 (0x24)	fload_2	69 (0x45)	fstore_2
10 (0x0a)	lconst_1	37 (0x25)	fload_3	70 (0x46)	fstore_3

- *Format:*
  - *Mnemonic (OPERATIONS/Actions)*
    operand1
    operand2
- Examples:

Mnemonics	Opcode	Description
aload_0	0x2A	Load a reference onto the stack from local variable 0
iadd	0x60	Add two « int » (i for int)
lcmp	0x94	Compare two « long » (l for long)

Java bytecode Math Example

```
 96 (0x60)    iadd
 97 (0x61)    ladd
 98 (0x62)    fadd
 99 (0x63)    dadd
100 (0x64)    isub
101 (0x65)    lsub
102 (0x66)    fsub
103 (0x67)    dsub
104 (0x68)    imul
105 (0x69)    lmul
106 (0x6a)    fmul
107 (0x6b)    dmul
108 (0x6c)    idiv
109 (0x6d)    ldiv
110 (0x6e)    fdiv
111 (0x6f)    ddiv
112 (0x70)    irem
113 (0x71)    lrem
```

```
114 (0x72)    frem
115 (0x73)    drem
116 (0x74)    ineg
117 (0x75)    lneg
118 (0x76)    fneg
119 (0x77)    dneg
120 (0x78)    ishl
121 (0x79)    lshl
122 (0x7a)    ishr
123 (0x7b)    lshr
124 (0x7c)    iushr
125 (0x7d)    lushr
126 (0x7e)    iand
127 (0x7f)    land
128 (0x80)    ior
129 (0x81)    lor
130 (0x82)    ixor
131 (0x83)    lxor
132 (0x84)    iinc
```

Disassembling Java bytecode (1/4)

- Get back your HelloWordDirect.java from Lab1
- And execute: ***$ javap -v*** HelloWordDirect.java

HelloWorldDirect.java - SciTE

File Edit Search View Tools Options Language Buffers Help

1 HelloWorldDirect.java

```
1  /*
2          Author: Stephane Sintes
3          Comments: Hello World for Direct Simple Compilation
4
5   */
6
7   // MAIN is the primary method to be called
8   // if you replace main by main2 : eror msg : Error: Main method not found in class apples
9  public class HelloWorldDirect {
10         public static void main(String args[]){  // mandatory main  =>static keyword is mandatory
11
12                 //print a line  PRINTLN
13                 System.out.println("Hello Great World!");
14
15         }//end main
16
17
18  }
19
```

Disassembling Java Bytecode (2/4)

$ javap -v HelloWordDirect.java

```
E:\E\Java\NonEclipseProject>javap -v  HelloWorldDirect.class
Classfile /E:/E/Java/NonEclipseProject/HelloWorldDirect.class
  Last modified Mar 11, 2016; size 558 bytes
  MD5 checksum 71a16ad5a0f154a77ac0280ca6932fd8
  Compiled from "HelloWorldDirect.java"
public class HelloWorldDirect
  minor version: 0
  major version: 52
  flags: ACC_PUBLIC, ACC_SUPER
Constant pool:
   #1 = Methodref          #6.#20         // java/lang/Object."<init>":()V
   #2 = Fieldref           #21.#22        // java/lang/System.out:Ljava/io/PrintStream;
   #3 = String             #23            // Hello Great World!
   #4 = Methodref          #24.#25        // java/io/PrintStream.println:(Ljava/lang/String;)V
   #5 = Class              #26            // HelloWorldDirect
   #6 = Class              #27            // java/lang/Object
   #7 = Utf8               <init>
   #8 = Utf8               ()V
   #9 = Utf8               Code
  #10 = Utf8               LineNumberTable
  #11 = Utf8               LocalVariableTable
  #12 = Utf8               this
  #13 = Utf8               LHelloWorldDirect;
  #14 = Utf8               main
  #15 = Utf8               ([Ljava/lang/String;)V
  #16 = Utf8               args
  #17 = Utf8               [Ljava/lang/String;
  #18 = Utf8               SourceFile
  #19 = Utf8               HelloWorldDirect.java
```

Disassembling Java Bytecode (3/4)

- Write Adder.class
- And execute: ***$ javap -v*** Adder.java

1 Adder.java

```
1   /*
2           Author: Stephane Sintes
3           Comments: class Adder for disassember
4    */
5   public class Adder {
6
7           public Adder(){
8
9           }
10
11          public int sum(int a, int b){
12                  int result;
13                  result  =  a +b ;
14                  return result;
15          }
16
17   }
```

Disassembling Java Bytecode (4/4)

$ javap -v Adder.class

1 Adder.java

```
1  /*
2          Author: Stephane Sintes
3          Comments: class Adder for disassember
4   */
5  public class Adder {
6
7          public Adder(){
8
9          }
10
11         public int sum(int a, int b){
12                 int result;
13                 result = a +b ;
14                 return result;
15         }
16
17  }
```

```
  public Adder();
    descriptor: ()V
    flags: ACC_PUBLIC
    Code:
      stack=1, locals=1, args_size=1
         0: aload_0
         1: invokespecial #1                  // Method java/l
         4: return
      LineNumberTable:
        line 3: 0
        line 5: 4

  public int sum(int, int);
    descriptor: (II)I
    flags: ACC_PUBLIC
    Code:
      stack=2, locals=4, args_size=3
         0: iload_1
         1: iload_2
         2: iadd
         3: istore_3
         4: iload_3
         5: ireturn
      LineNumberTable:
        line 10: 0
        line 12: 4
}
SourceFile: "Adder.java"
```

Stacked based
Load
Load
Add Integer
Store
Load
Return Integer

Interactive questions – Stack - Heap

- What is a stack?
- What do Push and Pop mean?
- FIFO or LIFO? Last In First Out
- What is a Heap?

Java Memory Model

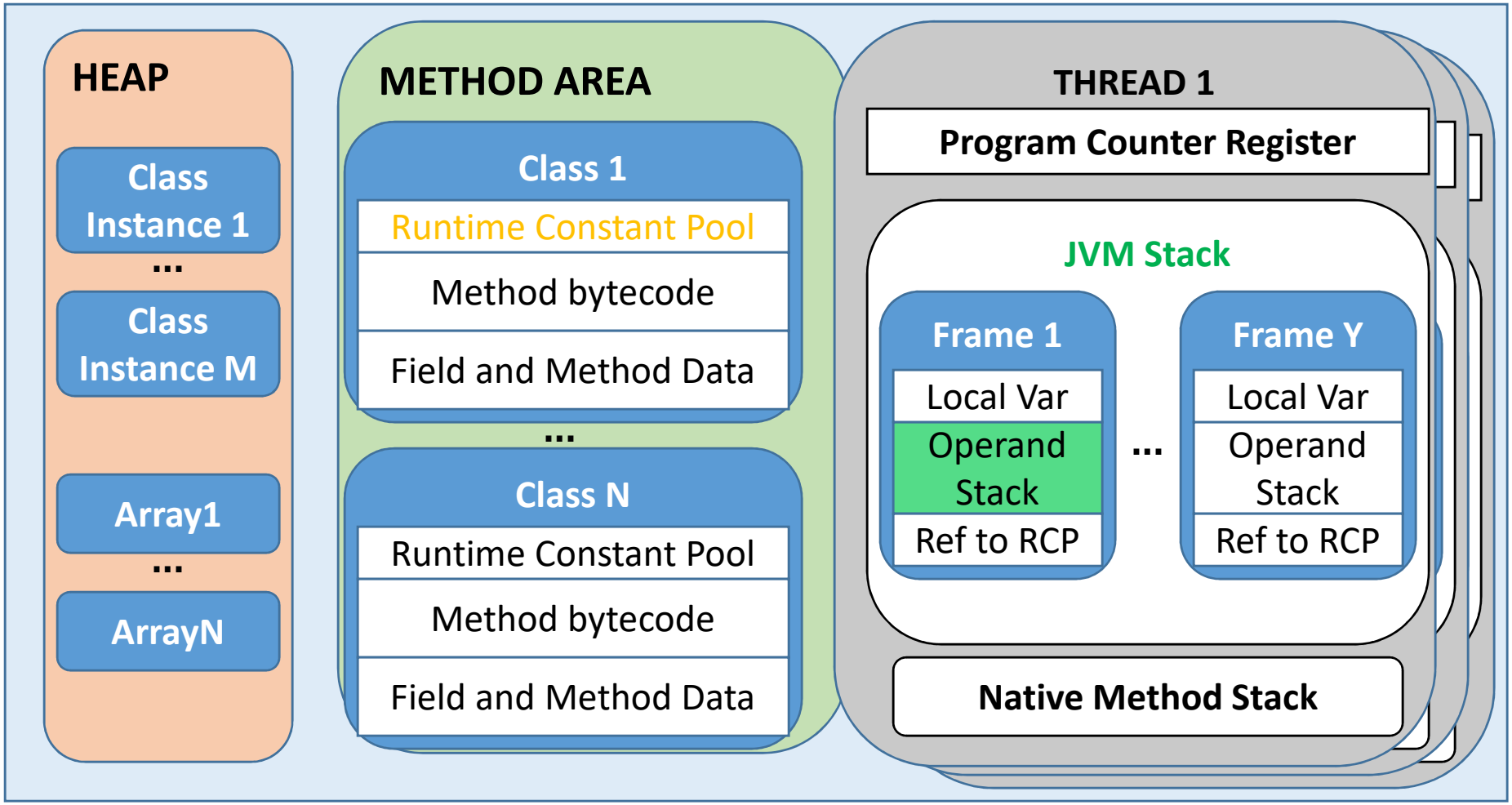
- Java has 3 mains type of memory area
  - Heap
    - Created on VM start-up
    - Store Object Instance (static fields, instance field …)
    - Arrays
    - Garbage collector acting on instances
    - Error msg : “OutOfMemory Error”
  - Method Area
    - Method byte code
    - Shared between all Classes => all class method code reside in the same area location
  - JVM Stack
    - One JVM Stack per thread
    - Store Frame
    - Operand stack … for basics well defined operations
    - Error msg :“StackOverflowError”

Object

Code

Compute

JVM Runtime Data Area

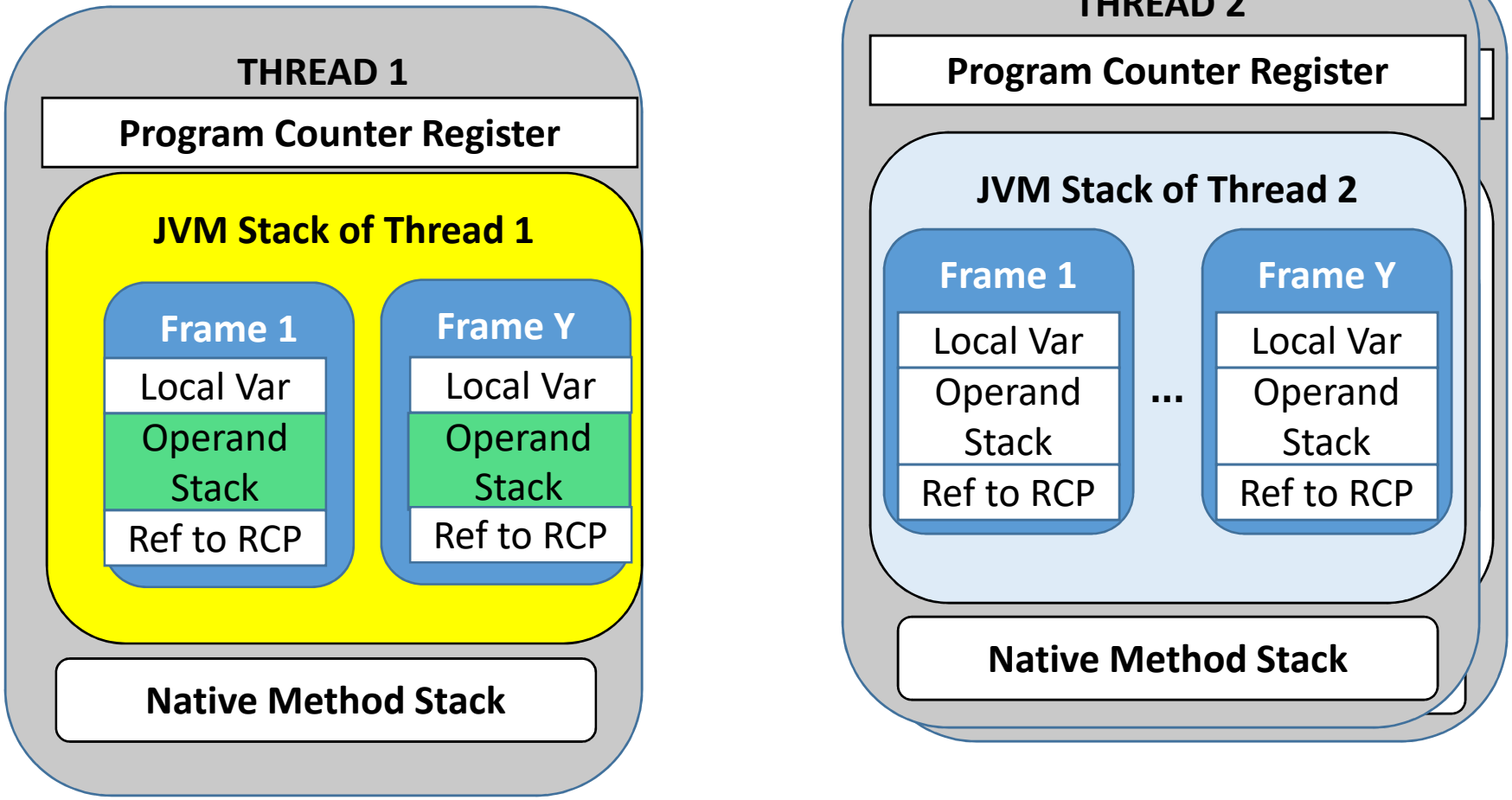


Detailed information available at:
http://docs.oracle.com/javase/specs/jvms/se7/html/jvms-2.html

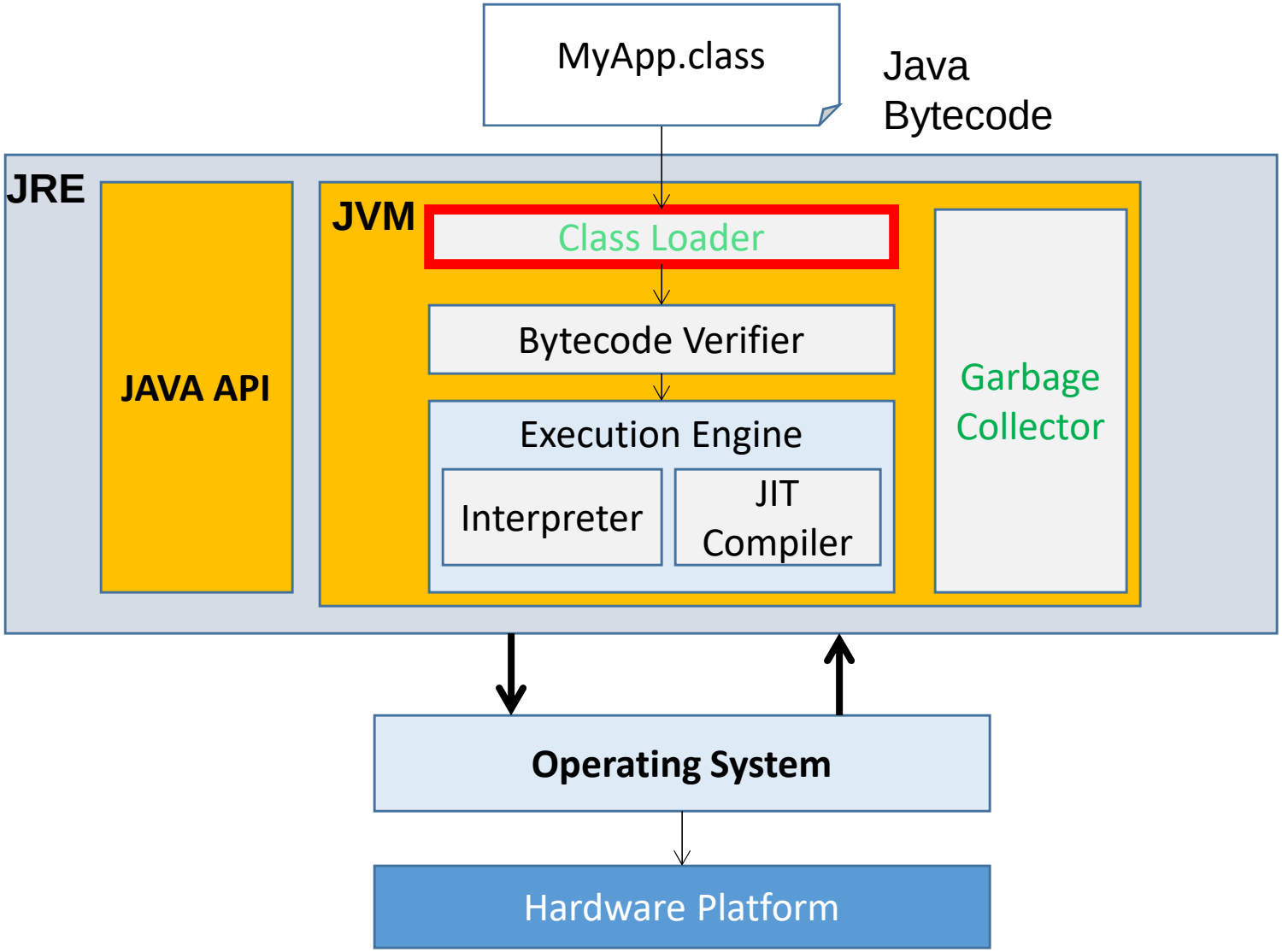
RCP: Runtime Constant Pool

JVM Stack

- One JVM Stack per thread

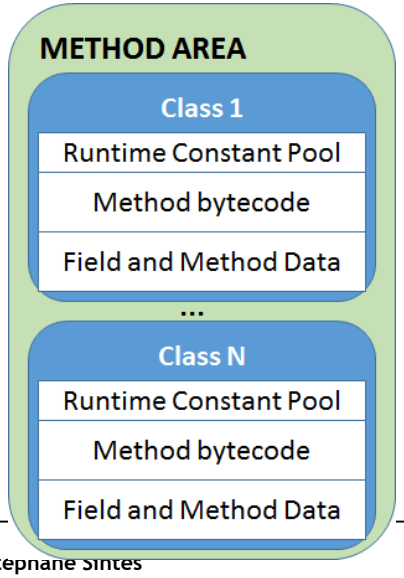


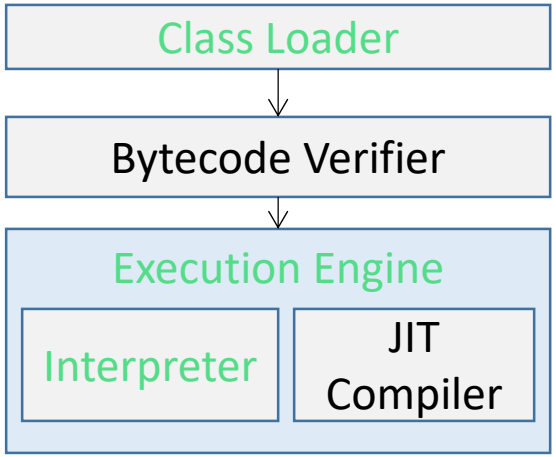
Executing a Java program (cont)



Class Loader

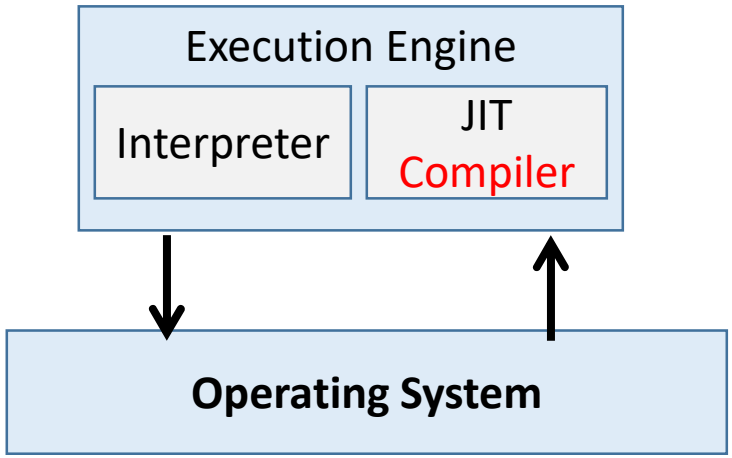
- In order to avoid disk I/O, not making Hard drive access, the bytecode is loaded into the JVM by class loaders in one of the runtime data areas in RAM (Method area)
  - This Bytecode is the method bytecode
- This bytecode remains in RAM memory (Method Area) until the JVM is stopped.
- The loaded bytecode code is then interpreted and executed by an execution engine
- A Class object is created by the ClassLoader the first time the *.class file is loaded





Execution engine

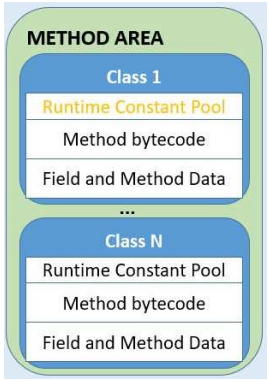
- The JAVA execution engine store several data
  - PC: Program Counter: pointer to the line of code being executed.
    - **Program Counter Register for each Thread**
  - Data from the method/class executed
- The execution engine handles the Operating System Interaction
- The JIT Compiler will increase the JAVA performances
  - Bytecode is compiled and store in the code cache (no heap mem)



Operand Stack

- JVM executes code by executing basics operations from the Java method bytecode.
- Java instruction operates on data parameters: Operands
- The data parameters are processed from a Stack: the Operand Stack (not register based)
- So when we are doing a+b, the addition operation is done from the Stack
- The operand stack is also used to pass and get parameters between Frame/method invocation

Runtime constant pool

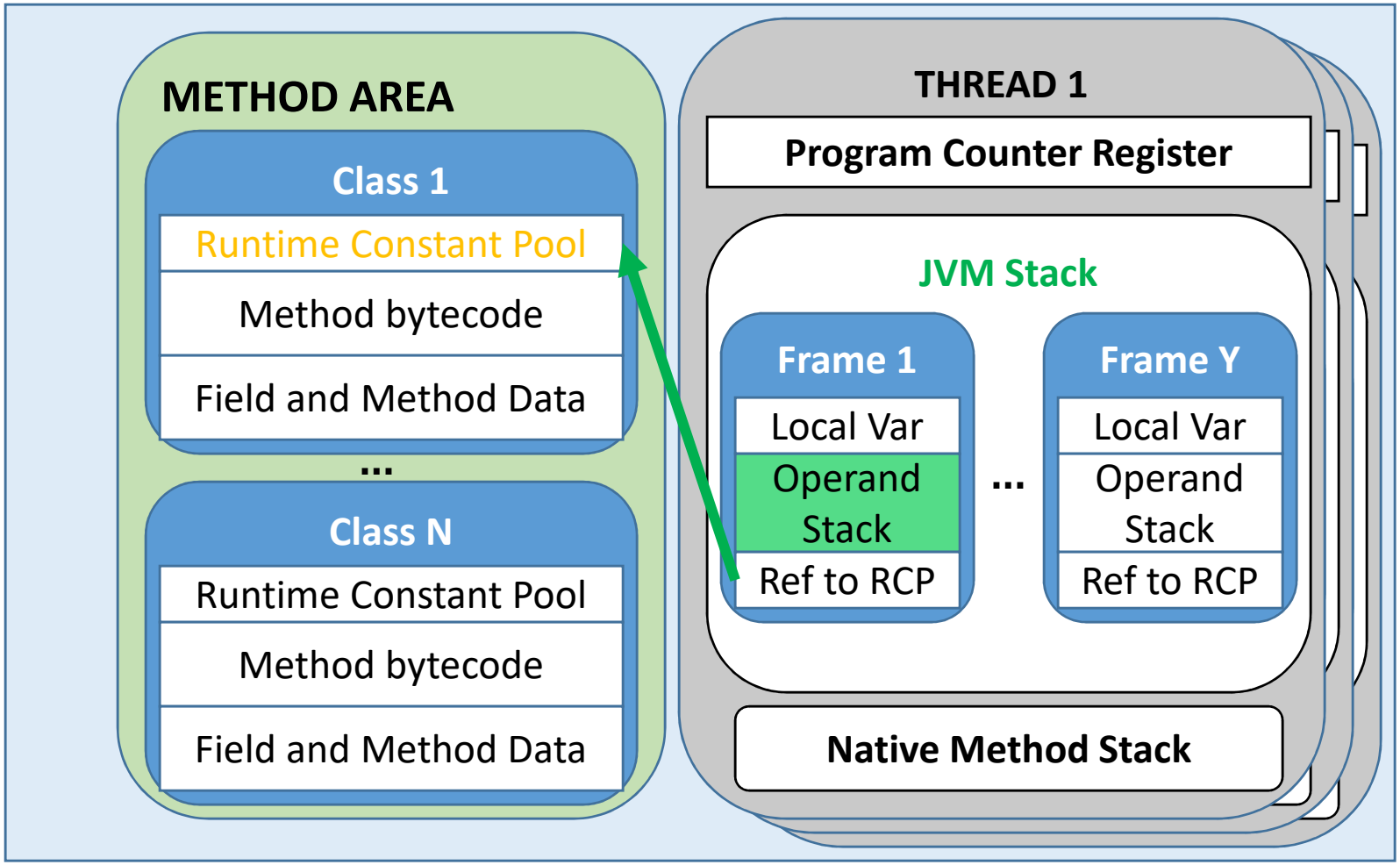


- This RCP is a subpart of the Method Area
- Each Java frame contains a reference to the runtime constant pool
  - **Run-time constant pool reference**: reference to the constant pool of the **current class** of for the **current method** being executed. Used by the JVM to translate symbolic method/variable reference (ex: myInstance.method()) to the real memory reference.
- RCP is like a symbol table for a conventional programming language
- Constant pool is used to keep track of the class and its members
  - when a class, method or field is referred to, the JVM searches the actual address in the memory by using the runtime constant pool.
  - Method is at a specific address in memory
- Contains constant values like string or constant.

0	<Class> "My Class"
1	<String> "Hello World"
2	<Integer> 1234567
...	...

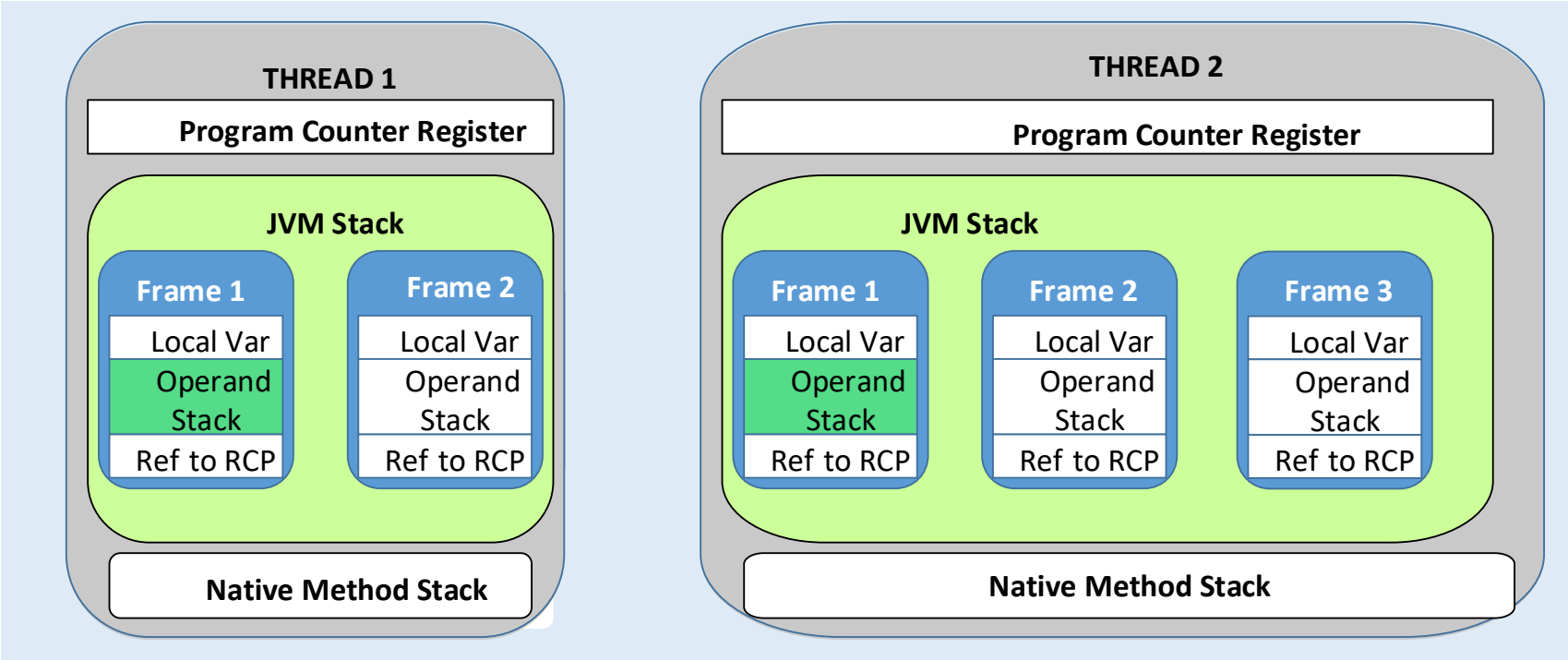
String myString1 = “Hello the Team”;
static final int MY_CONSTANT=2;

Runtime Constant Pool



RCP: Runtime Constant Pool

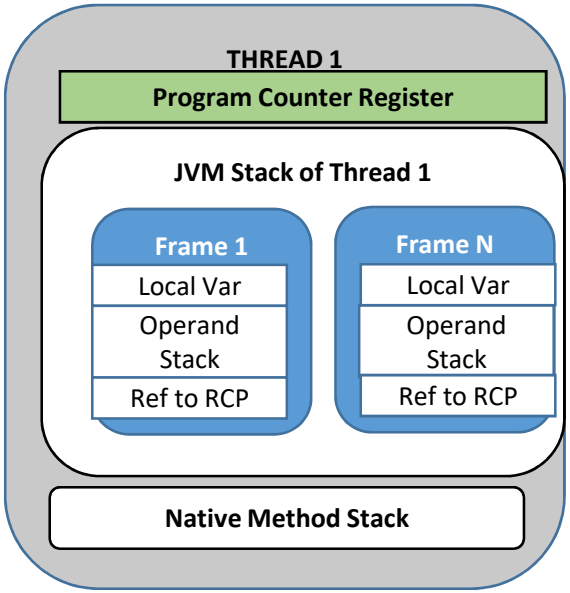
JVM Runtime Data Area – Multithread



RCP: Runtime Constant Pool

Program Counter Register

- Each thread has its own PC (program counter) register, created when the thread is created
- The JVM Threads are executing concurrently
- At any point, each Java Virtual Machine thread is executing the code of a single method, namely the current method for that thread.
- The PC register contains the address of the Java Virtual Machine instruction (in the method area) currently being executed.
- If the method currently being executed by the thread is native, the value of the Java Virtual Machine's PC register is undefined.
  - Not anymore pointing to the ByteCode, Native method does not run in the JVM



Interactive questions – Native Code

- What is a Native Method?
  - Method not written in Java
  - Not in virtual Machine
  - Machine dependent
  - Compiled on the target – native => machine (instruction machine)
    - Ex code Intel 64 bits
  - Not anymore portable => not anymore WORA (Write Once Run Anywhere)
  - Performance increase => very used in Android
  - Smartphone – ARM -> Neon acceleration HWA => native =>HWA

Native Method / Native Stack

- Native method does not run in the JVM. Native method is typically written in C or C++.
- For Android, this is usual to lack some functionality in Java and you have to implement some C code (ARM Neon acceleration can be used)
- Native methods also allow to accelerate the execution code.
- Native method are called through JNI (Java Native Interface)
- Native method runs from Native stack, the behavior of this stack is entirely dependent of the underlying OS.
- WORA concept is killed by using Native Method

Java Virtual Machine Stack

Java source code

```
public int add(int a, int b){
    int sum = a + b;
    return sum;
}
```

« javac » compilation

Java bytecode

```
0: iload_1
1: iload_2
2: iadd
3: istore_3
4: iload_3
5: ireturn
```

Bytecode address inside the method

Stack based architecture

Hidden for the developer

Strong Impact on Code generation

Operand: value on which an instruction operates.

Java Virtual Machine Stack

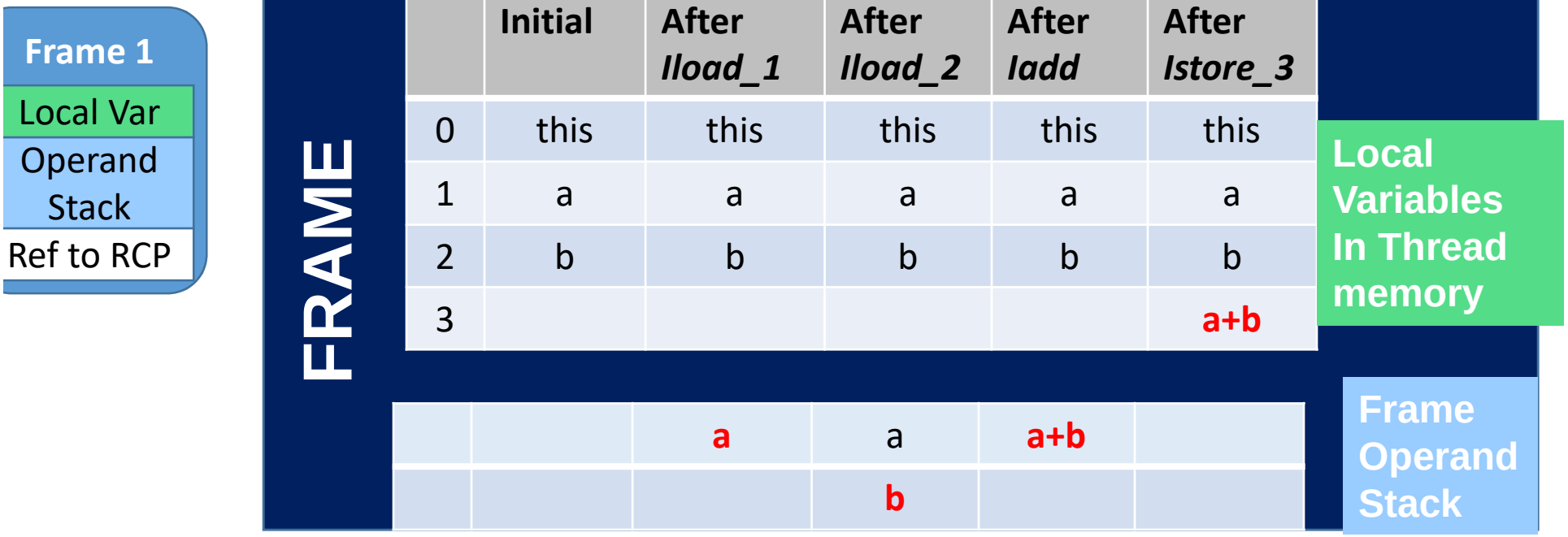
Java source code

```
public int add(int a, int b){
    int sum = a + b;
    return sum;
}
```

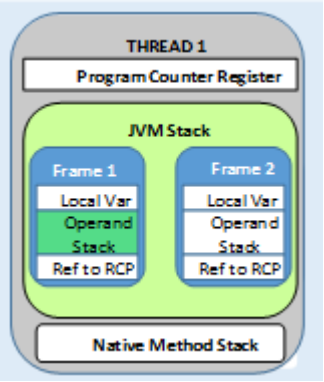
« javac » compilation →

Java bytecode

```
0: iload_1
1: iload_2
2: iadd
3: istore_3
4: iload_3
5: ireturn
```



Frame



- A Java Virtual Machine stack (in green) stores frames.
- A new frame is created and put in the JVM stack each time a method is invoked.
- A frame is destroyed when its method invocation completes, whether that completion is normal or abrupt (it throws an uncaught exception).
- Only one frame, the frame for the executing method, is active at any point in a given thread.
- This frame is referred to as the current frame, and its method is known as the current method.
- The class in which the current method is defined is the current class.
- Operations on local variables and the operand stack are with the current frame.
- Each Java frame contains a reference to the runtime constant pool

Interactive questions - Frame

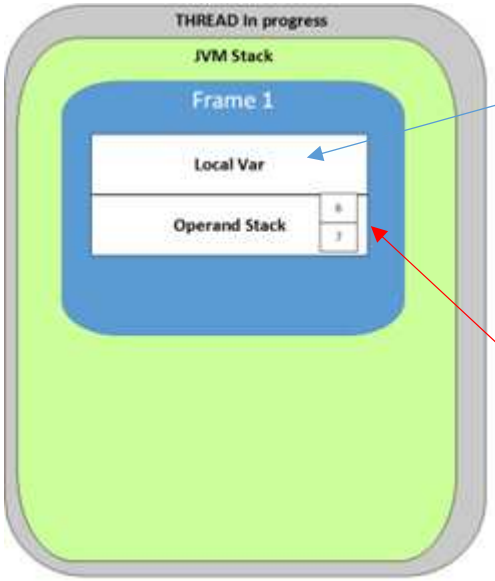
- Why do we have several frames per Thread? Because nested method
  - Method_a => Frame1 created
  - Method_b => Frame1 destroyed Frame 1 created
  - Method_c => Frame 1 created

- Method_a (=> Method_b) method_a =>
  - Frame1 created
  - Frame2 created
- Method_c => Frame 1 created

Frame & Memory example (1/3)

```
Multiplication of two data parameter: 6*7

//ss here we have a method calling another method
//ss so we have 2 methods => 2 frames
//ss if we have 10 nested calls we have 10 frames

public void method1_wrapper(){
  int result = method2_multiply(6,7); //ss calling method2
}


public int method2_multiply(int m, int n){
  return m*n;
}

//ss m and n are Local variables in the method2_multiply method
```

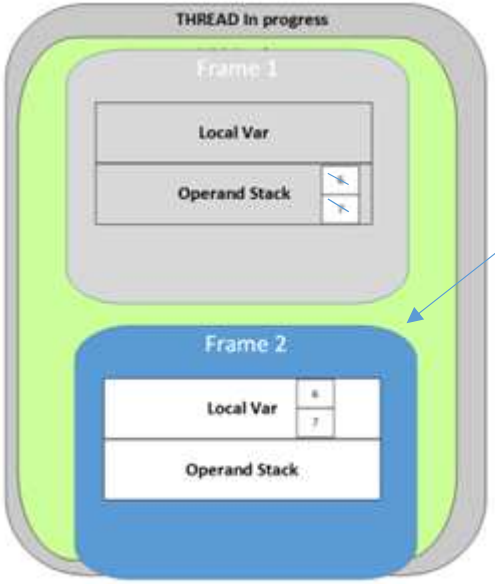
Frame & Memory example (2/3)



The Java Virtual Machine uses local variables to pass parameters on method invocation

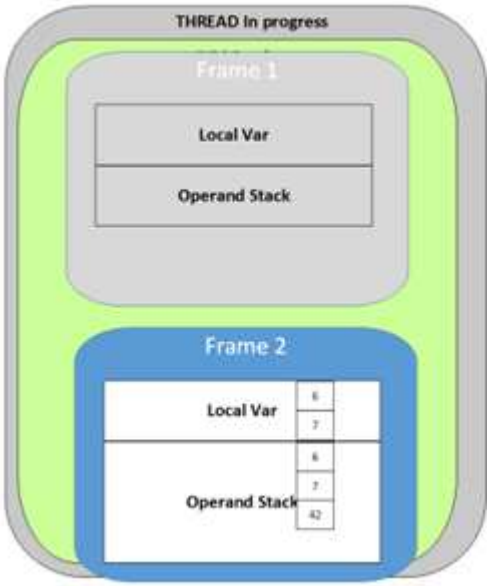
The array of the called method (method2) is created from the operand stack of the calling method (method1)

Frame 1 active:
we pushes 6 and 7 local variable
in the Frame1 operand stack
Frame one Inactive

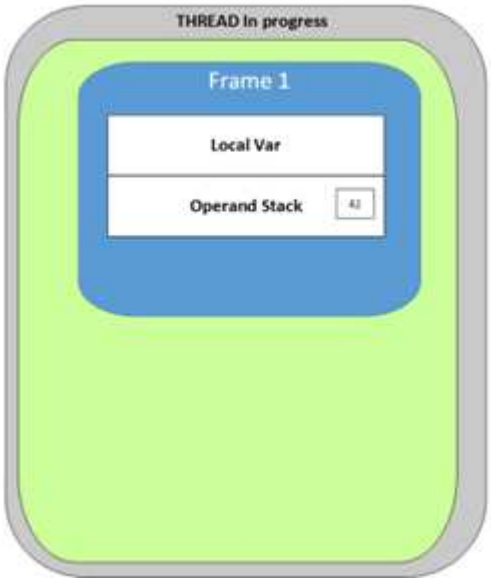


Frame 2 active:
The Operand Stack of Frame 1 are popped in Local var space for Frame 2
So the values 6 and 7 are not anymore in the Operand Stack of Frame 1

Frame & Memory example (3/3)



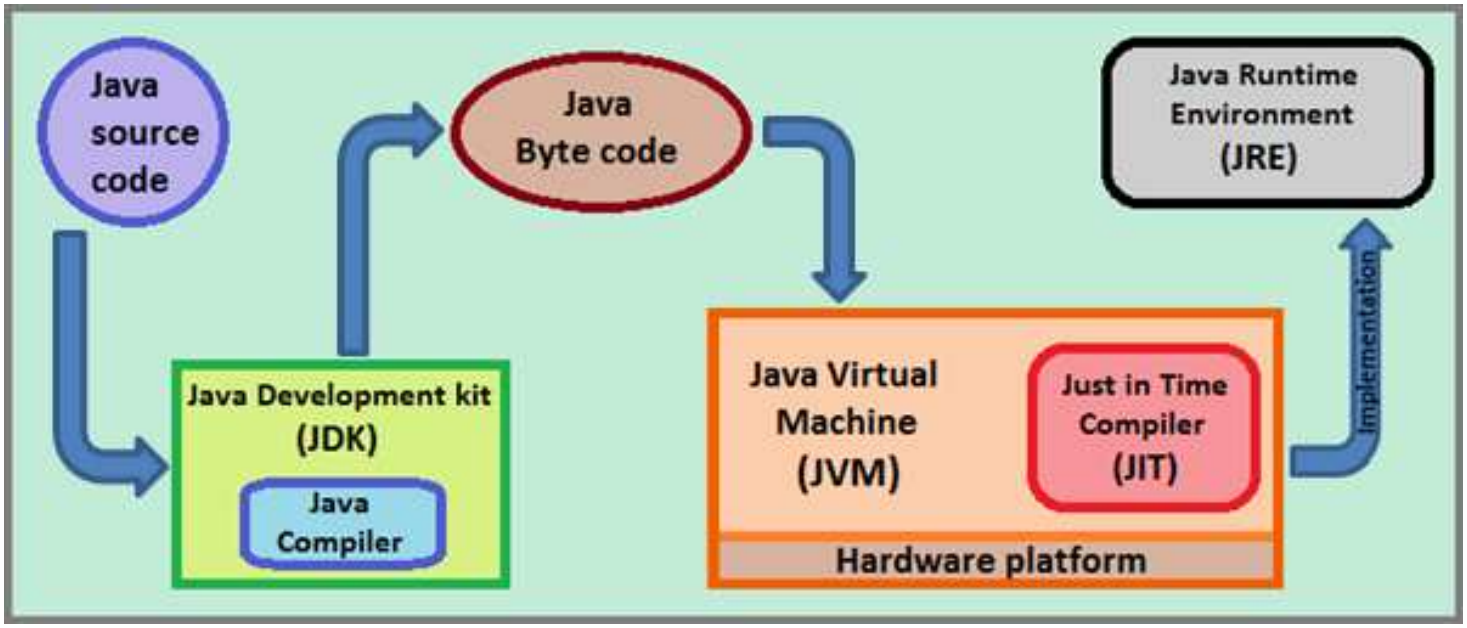
Frame 2 active:
we push 6 and 7 (local variable) in the operand stack of Frame2
Then calls the imult (integer multiplication) instruction that generate
42 on the Frame 2 Operand Stack



Frame1 active:
Result “42” is pushed on Frame 1 Operand Stack
Frame 2 is destroyed (popped out of the stack)
Frame 2 does not exist anymore
Final result it obtained by popping Frame1 operand stack

Just-In-Time compilation

- Compiling bytecode to machine language in order to accelerate program execution.
- Oracle Hotspot JVM has 2 compilers (C1 & C2)
  - C1 or Client: fast, low memory usage, less optimization
  - C2 or Server: Aggressive optimization... but slower than C1
    - Server are running during long period – data profiling collection



Object-Oriented Design

"Object-Oriented" Design

- **Goal:** Designing a system to solve a problem!
- **Principle:** System is designed as a collection of interacting objects.
- **Alternative :** "Functional" approach. System is designed as a hierarchy functions

➔Object-Oriented Programming: Java, C++, C#, Python …

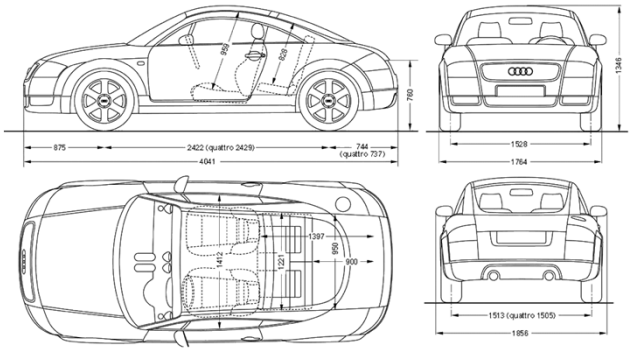
➔Functional Programming : C, …

What is an "Object"?

- An object is an entity characterized by its "**state**" and its "**behavior**"
- For instance, a "**Car**" object consists of:
  - ***State:*** gear position, speed, tires, color, number of doors…
  - ***Behavior:*** shifting gears, accelerating, braking, changing tires size,…
- For instance, a "**TV**" object consists of:
  - ***State:*** channel, volume, brightness…
  - ***Behavior:*** changing channel, changing volume, modifying brightness…
- Java applications are made of objects …

"Class" versus "Objects"

- Several objects are actually of the same type, built from the same blueprint. Thousands of cars are based on the same blueprint.
- In Object-Oriented Design, the blueprint from which objects are created is called a "Class"



Class

- An "Object" is an "Instance" of a "Class"
- To Create the object the new Java keyword is used.
  - new creates a Java object and allocates memory for this object on the heap.
  - new is also used for array creation, as arrays are also objects.

Objects

Class, Attributes, Methods, Interface

- A Class provides "**fields**" or "**attributes**" to store object's state.
- Object behavior is represented by Class' "**methods**" or "**functions**".
- Object's methods are exposed to enable interaction with other objects. Methods form the object's "**interface"** with the rest of the system.

Interactive questions – Open Source

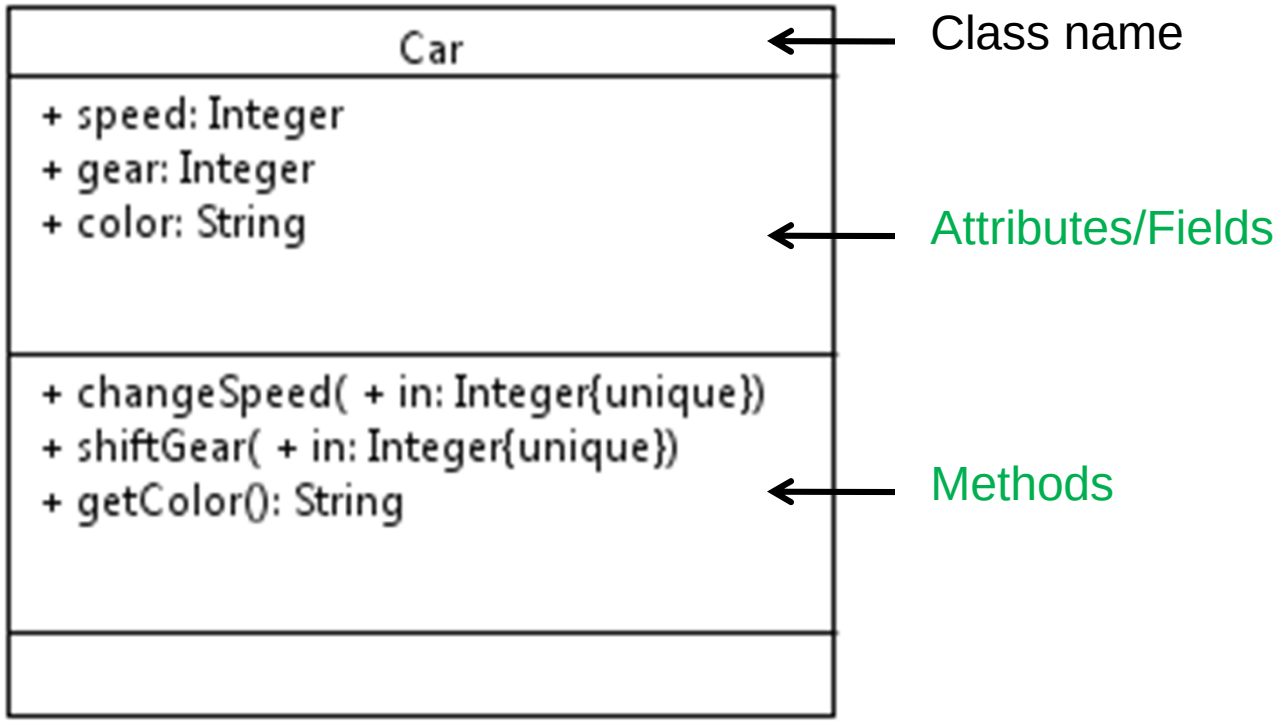
- How open source word world is working? Linux example
- What is a patch?
- What is a maintainer?
- How patch submission is working?
- Importance of Naming Convention

JAVA Class Naming Convention

- Class names should be nouns, in mixed case
  - with the first letter of each internal word Capitalized.
- Try to keep your class names simple and descriptive.
- Use whole words—avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).

UML: Unified Modeling Language

Modeling a “Class”



Access level modifiers

- Access level modifiers determine whether other classes can use a particular field or invoke a particular method

Class Members Access Control

We position from the current class

Derived Class outside the package

Modifier	Class	Package	Subclass	World
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no modifier	Y	Y	N	N
private	Y	N	N	N

Strongest restrict

http://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

Y : Accessible
N: Not Accessible

Default Access Modifier

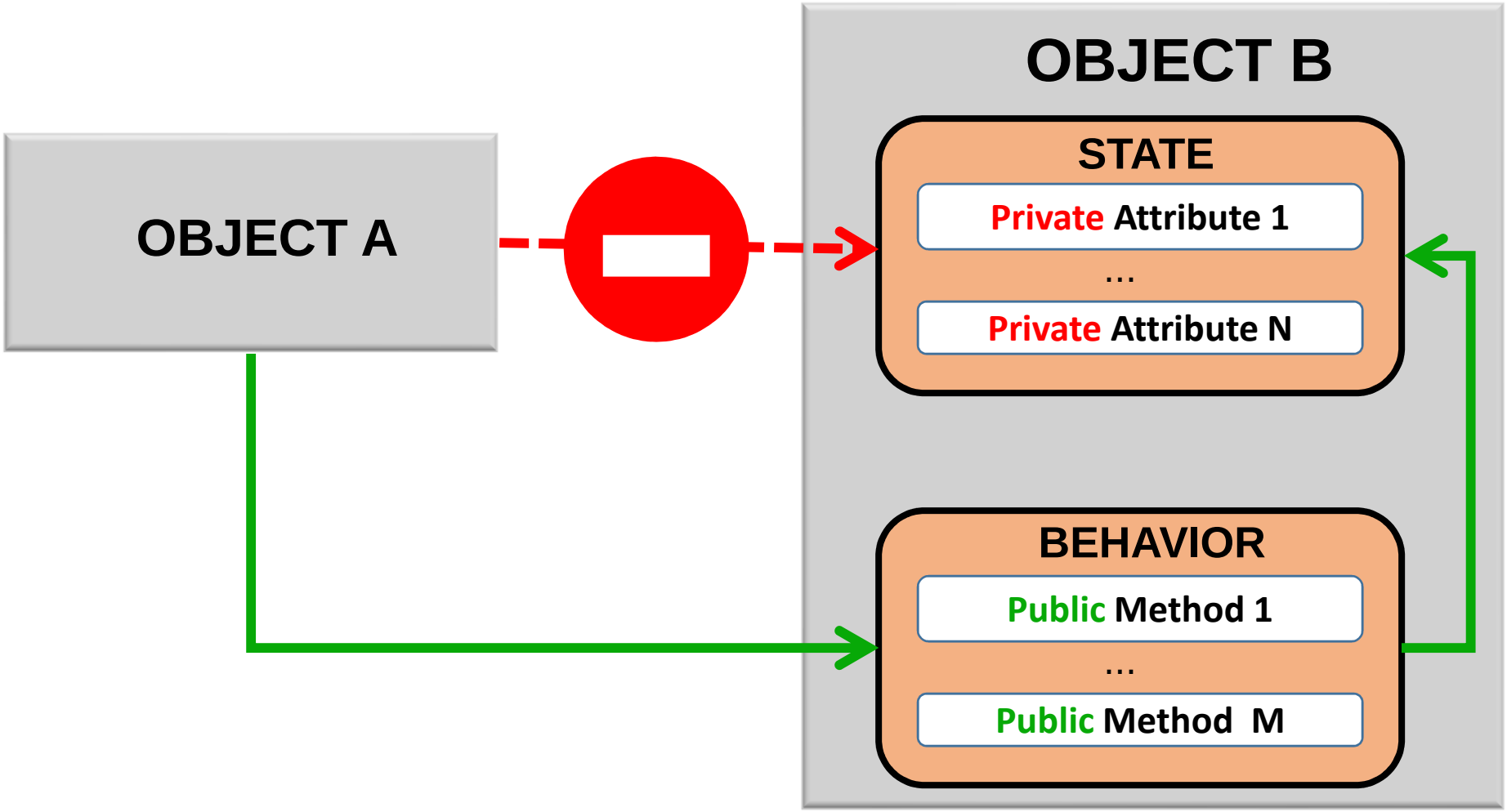
Default Access Modifier - **No keyword**:
Default access modifier means we do not explicitly declare an access modifier for a class, field, method, etc.
A variable or method declared without any access control modifier is available to any other class in the same package => "Package private"
So a constructor with no modifier is Package private
So this constructor can be called from within the same class or from any other class in the same package, but not from subclasses in a different package.

The fields in an interface are implicitly public static final and the methods in an interface are by default public.

Encapsulation (1/3)

- Encapsulation = hiding/isolating data
- Combining data (state/ attributes/field) and ways to manipulate it (behavior/methods) in a single place the class.
- Protecting object's data integrity by preventing direct access from outside world. Object's state can only be modified through object's method (access modifier)
- Hiding the complexity of an object behind a stable interface. No need to understand implementation details of an object to use it, knowing its interface is enough. Encapsulating state/methods behind an interface allows modifying object's implementation without impacting other components of the system.
  - This is the public modifier

Encapsulation(2/3)



Encapsulation (3/3)

- Encapsulation is about distinguishing specification from implementations
  - The specification expresses what all objects of some type are expected to do
  - An implementation expresses how some objects of that type are doing it, this is part can be seen as a black box from external world
    - Implementation should be hidden
- Encapsulation is key Object-Oriented

LAB2

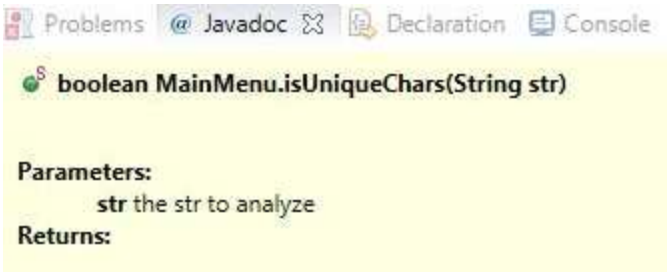
- Start LAB2

Objectives:

- Write my first simple Java classes
  - Rectangle and Circle class
- Controlling access to members of a class (*private* / *public* keywords and no keywords)
- Understanding the difference between Instance and Class members (*static* keyword*)*
- Understand the use of the *final* keyword with variable.
- Use Math library

JavaDoc

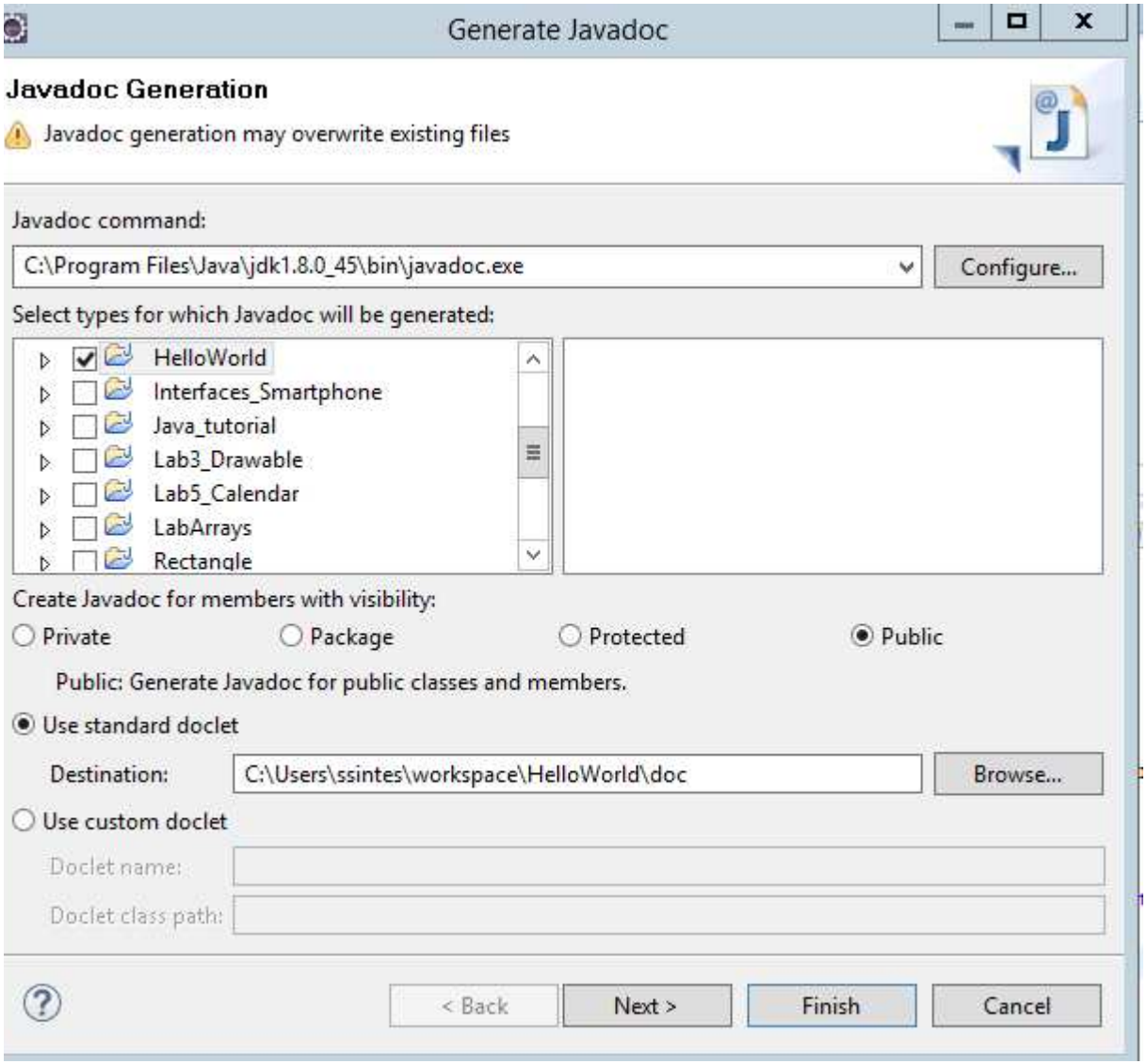
```
Type: /** followed by enter
Automatically generated
/**
 *
 * @param str str is used for
 * @return
 */
```




```
public  static boolean isUniqueChars(String str) {
```

Automatic doc generation for Class, Method, Constructor

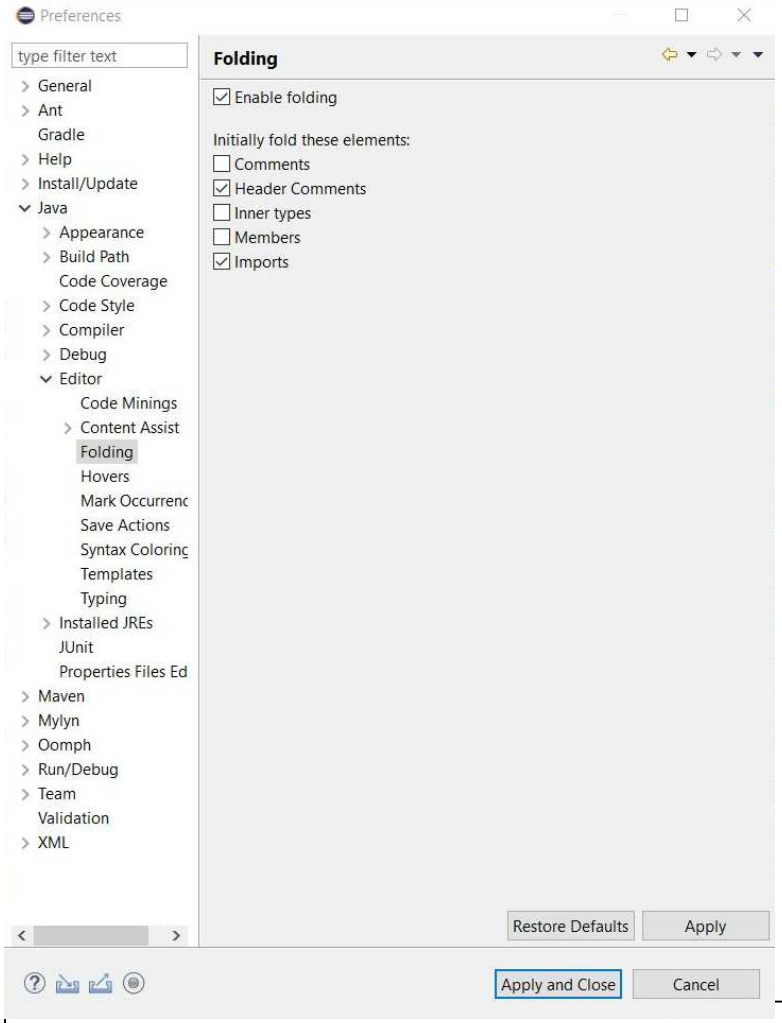
Eclipse – javadoc

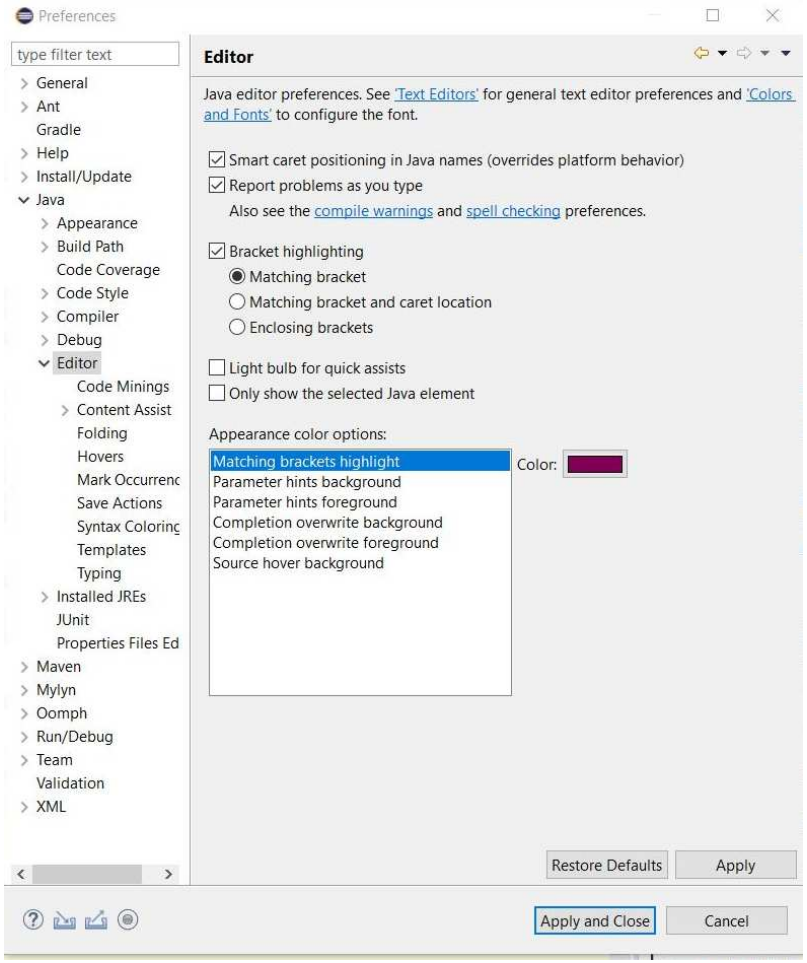


Project / generate javadoc

Java Editor Preferences

- Menu Windows /





Semicolon auto

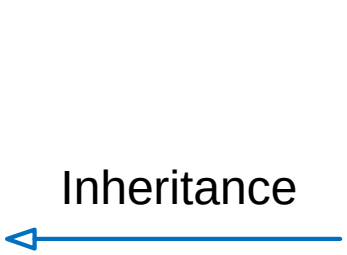
Inheritance

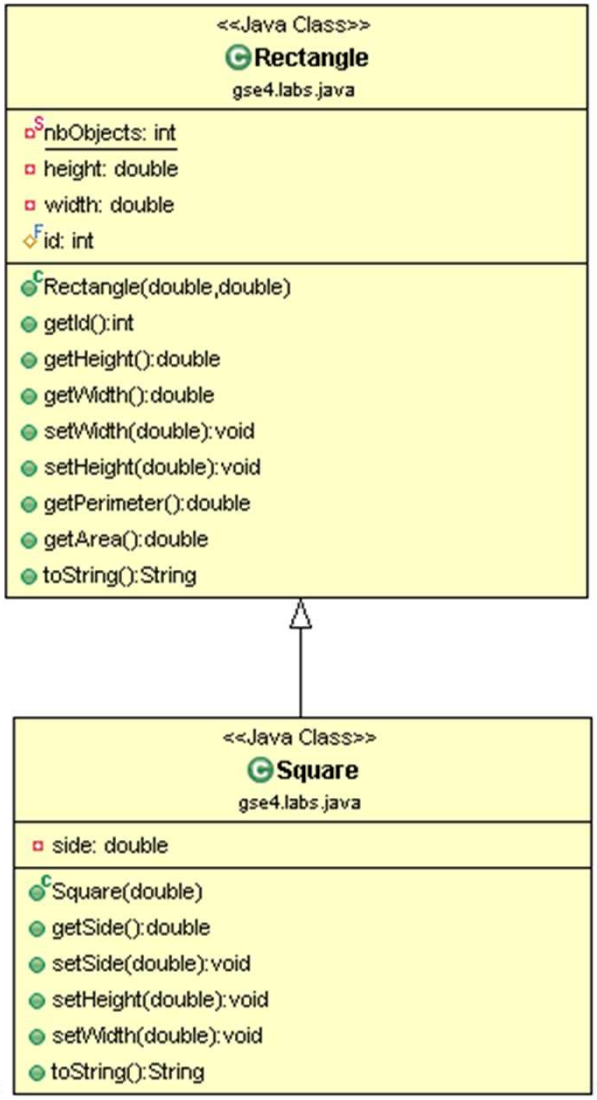
- Inheritance is a mechanism allowing to re-use state/behavior from existing classes (reducing development effort, simplifying maintenance…).
- *Class B* can inherit attributes and methods from an existing *Class A*.
- *Class A* is called "**super class**" or "**parent class**".
  Class B is called "**derived class**" or "**child class**" or "**sub class**".
- *Class B* can redefine methods inherited from *Class A*
- *Class B* can define additional attributes/methods that differentiate it from *Class A*.

UML Class Diagram inheritance

Modeling the “inheritance” relationship

Inheritance





Square inherit from Rectangle

Careful about the arrow direction

Abstraction (1/2)

- An abstract class is a class that cannot be instantiated. It provides a common template for sub-classes.
- abstract class can contain fields and methods. It can include methods with no implementation (abstract methods)
- Methods with no implementation are called method stub.

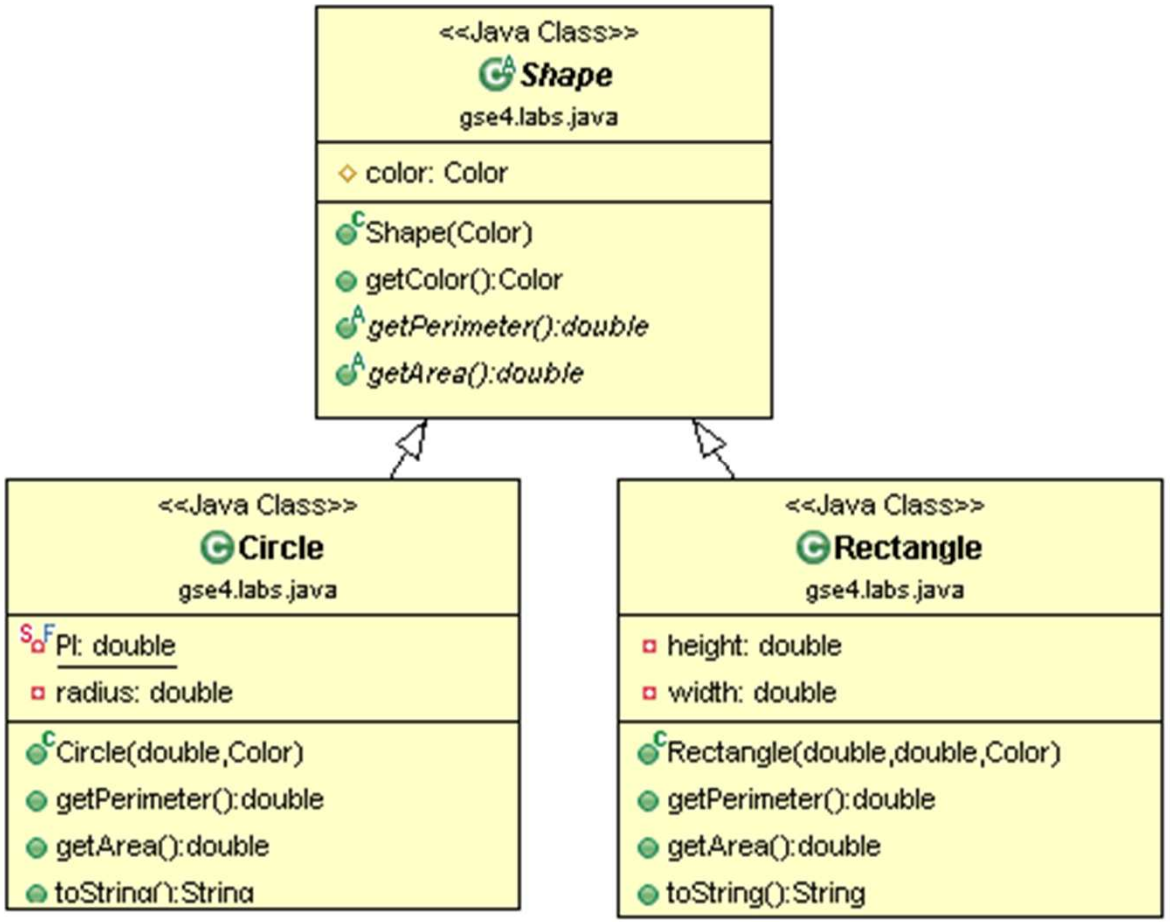
Abstraction (2/2)

- Rectangles, squares and circles are three particular geometric shapes. Therefore, the **Rectangle**, **Square** and **Circle** classes defined before could actually inherit from a generic **Shape** class.
- Geometric shapes supports common operations, like computing "area" or "perimeter", but performed in different ways depending on the kind of shape. For instance, circles and squares are geometric shapes with different area formulas [$A_{square} = R \times R$ / $A_{circle} = \pi \times R \times R$]
- Therefore, a **Shape** class could declare operations (methods) common to all shapes (e.g. computing area) but would not be able to provide an implementation of it (area formula depending on the actual shape type)

UML Class Diagram – Abstract

Inheriting from an abstract class

Inheritance



Designed with Eclipse UML Plugin: http://www.objectaid.net/

Abstract class code

```
public abstract  class Shape  {

protected Color color;

Shape(Color c){  //ss Shape constructor – same name as class
color = c;

}
```

Polymorphism

- Inheritance brings Polymorphism. Polymorphism is the ability for an object to take on many forms
- Polymorphism happens when the reference to a "super class" is used to refer to a "derived class"
- A **Circle** IS-A **Circle** / A **Circle** IS-A **Shape**

A **Circle** IS-A **Circle:**

Circle circle = new Circle();

A **Circle** IS-A **Shape**:

Shape shape = circle

From right to left:
A Circle is a shape

Reference to a super class

Java Interface (1/2)

- Notion Interface:Objects expose their behavior to other objects through their public methods. The set of methods of an object form its *interface* to the external world.

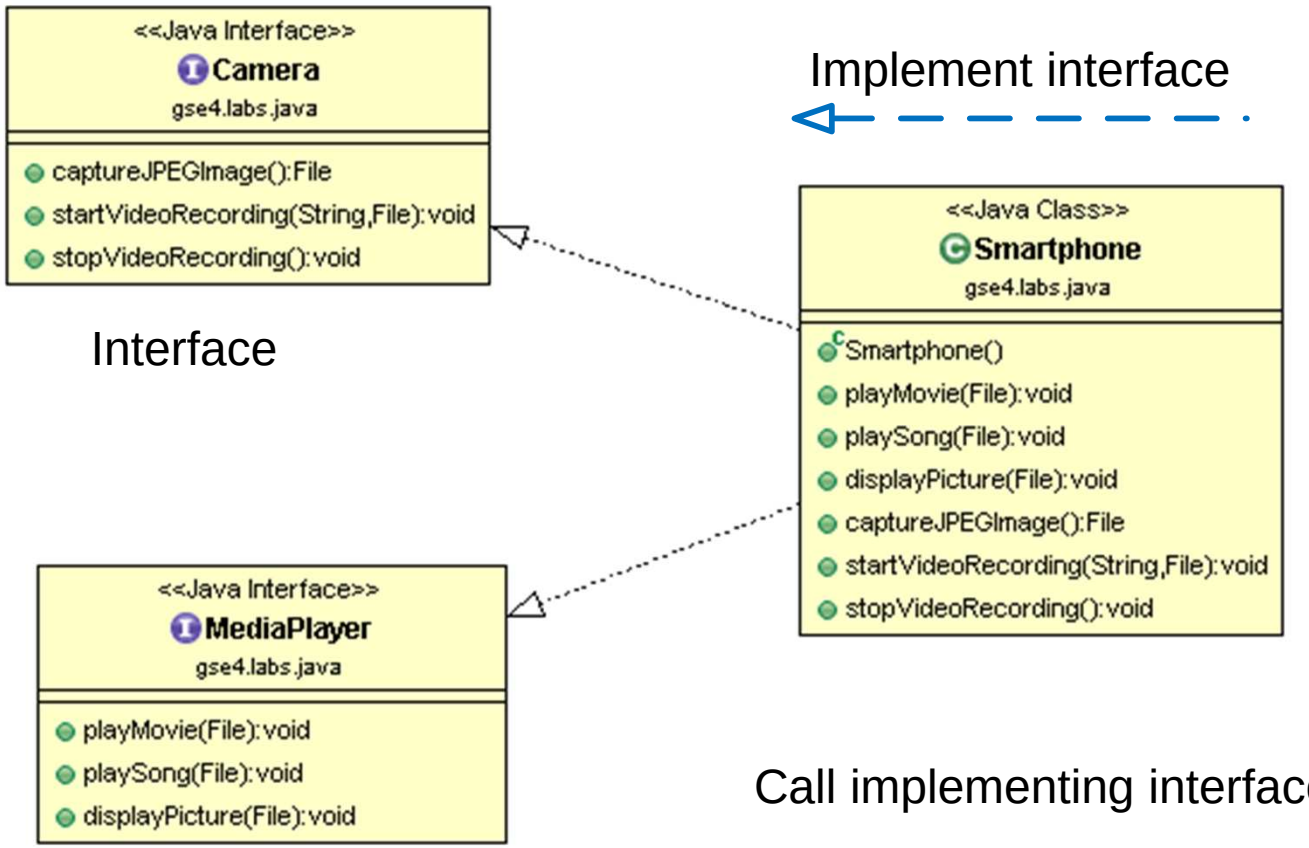
- Most of the time, an JAVA Interface consists in a list of related methods grouped together. An Interface declares a group of methods but does not provide an implementation of these methods (this will be the responsibility of the class).

- An Interface can be seen as a “contract” between the class that implements it and the user of this class. A class claiming to implement a given Interface promises to provide a specific behavior (the one specified by the Interface) to its users. The class must then implement all the methods of the interface.

Java Interface (2/2)

- Java Interfaces are critical because they separate what a class does from how the class does it
- The contract is defining what a client can expect and leaves the developer free to implement it any way they choose, as long as they the contract is respected.
- Many examples can be found in the JDK.
  - in java.sql package contains many interfaces.
  - Why? To allow to use different relational databases.
  - Java data base developers are free to implement interface methods for their particular product. Data base Clients need only deal with the interface reference types. Moving to new relational java databases is as easy as swapping one JDBC (Java Database Connectivity) driver JAR for another.
  - Clients do not need to change their code.

Implementing Interface



Interface

Interface

Call implementing interface

Smartphone is a class that implements 2 interfaces

Interface conflict name

- What happen if implement 2 interfaces with same method name?

Interface versus Abstract

- A class that implements the interface have to implement ALL the methods defined in the interface
- Abstract classes can have constants, members, method stubs and defined methods, whereas interfaces can only have constants and methods stubs.

```
interface SpaceShip {
        //ss next line is an error, you cannot declare a non-constant variable
        //int member;
        String common = "My space Ship"; //ss this is a constant so it work
    public void move();
    public void fire();
    public void explose();
}
```

Interface Implements

public class Rectangle **implements** Drawable {

Drawable is an interface

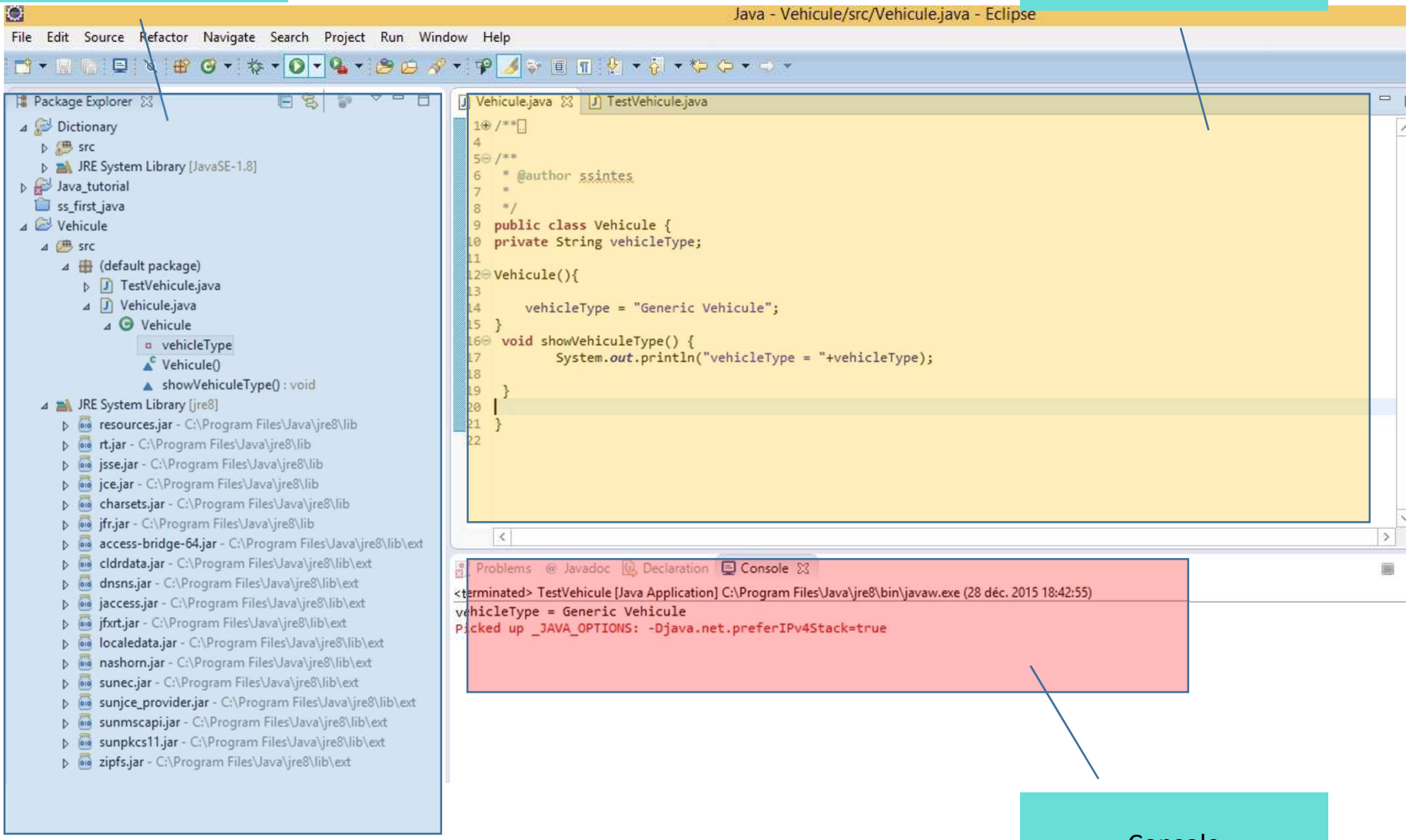
Drawable myD = **new Rectangle();** **//ss we can assign an interface (polymorphisme)**

Drawable myD = new Drawable (); **//ss we cannot instantiate an interface as abstract**

Package Explorer

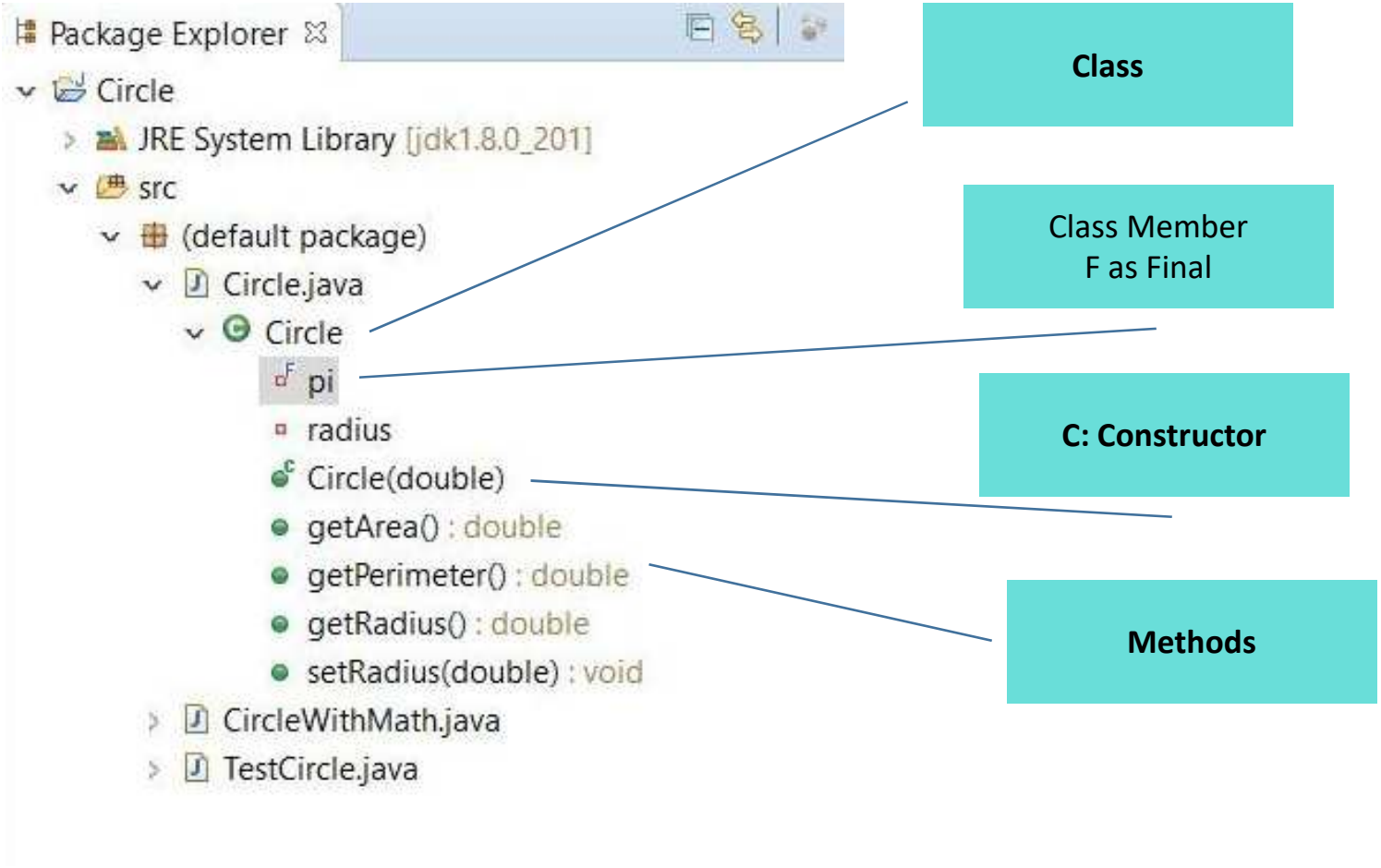
Eclipse - Windows

Java source code in progress

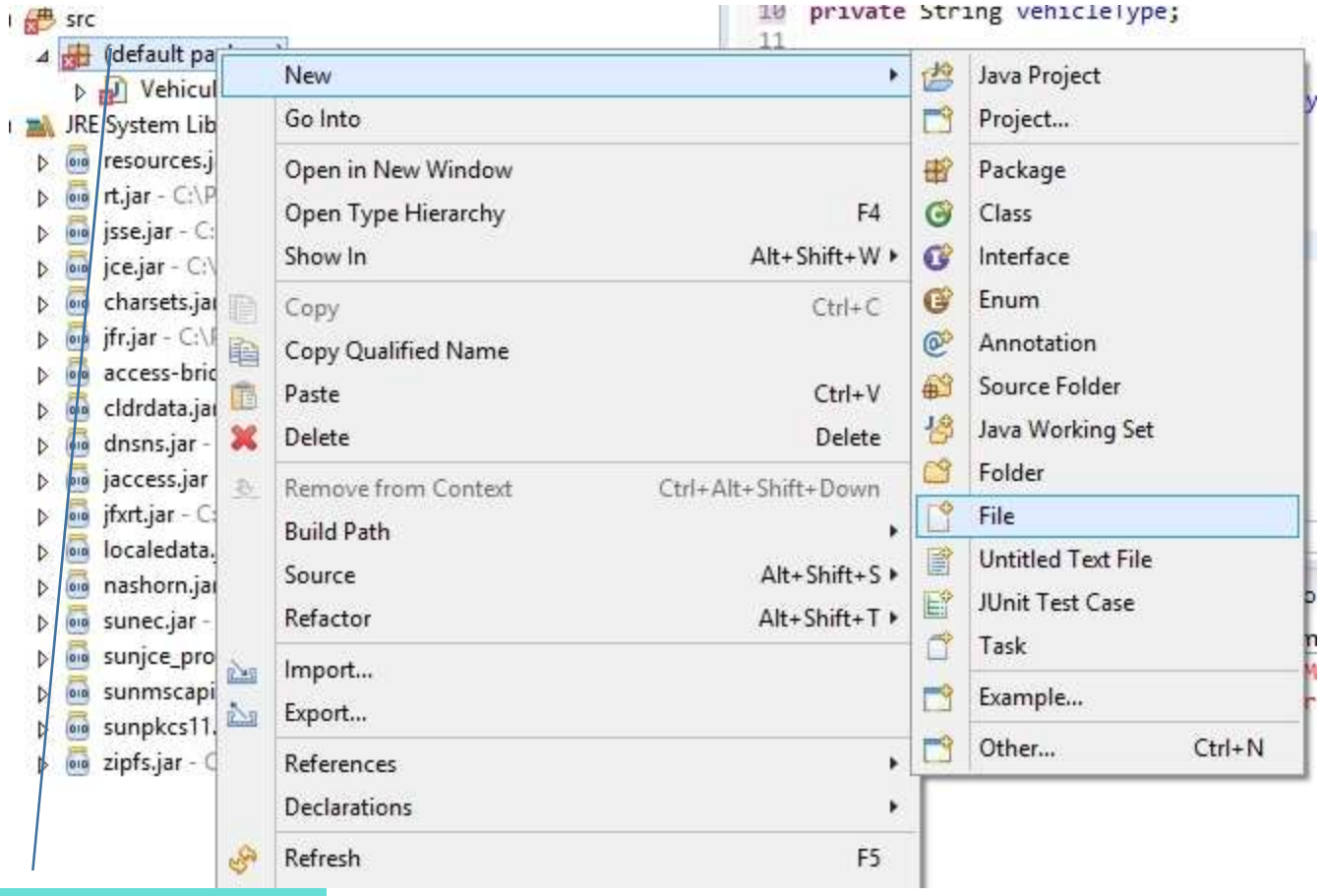


Console

Package Explorer

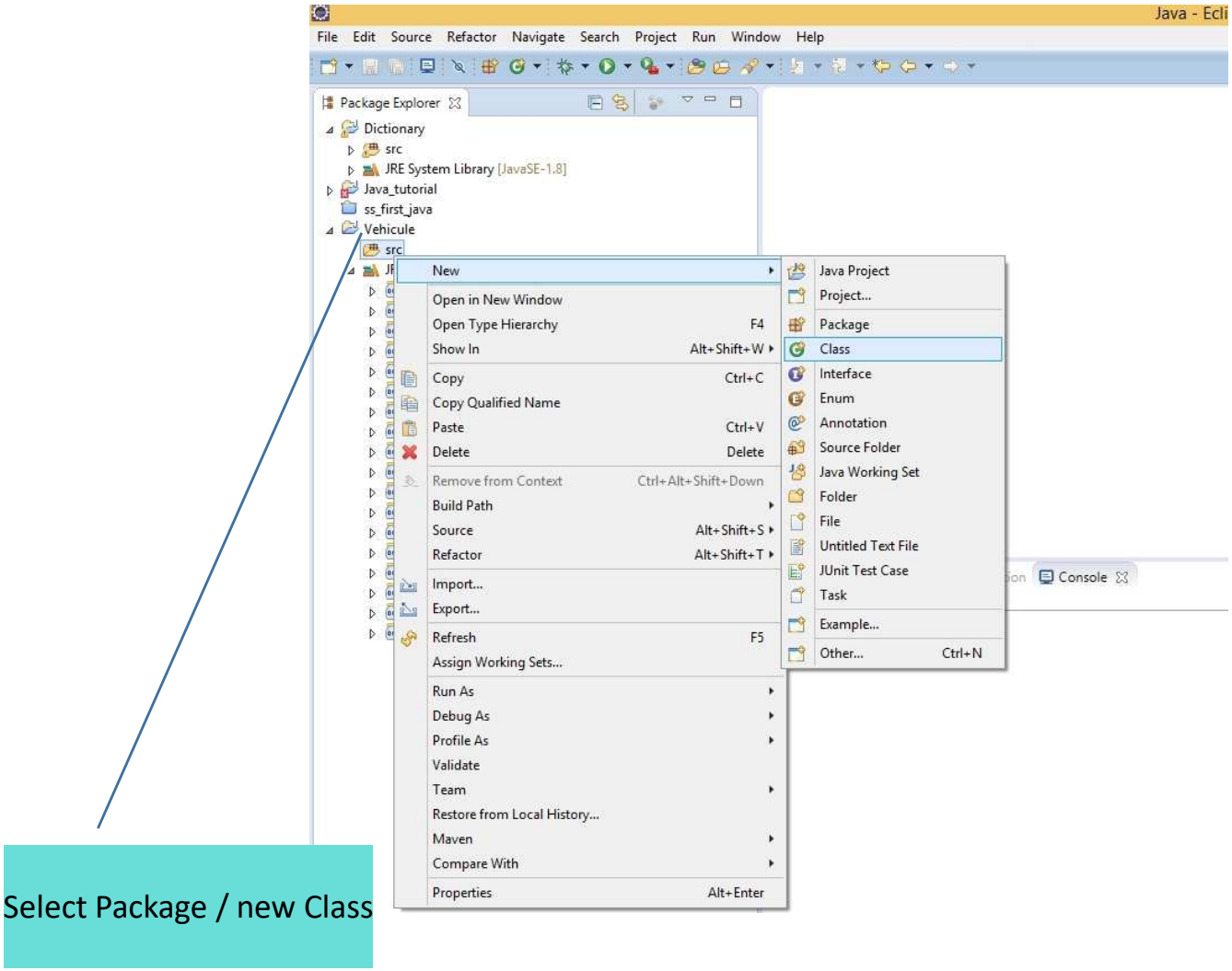


Eclipse – File Creation

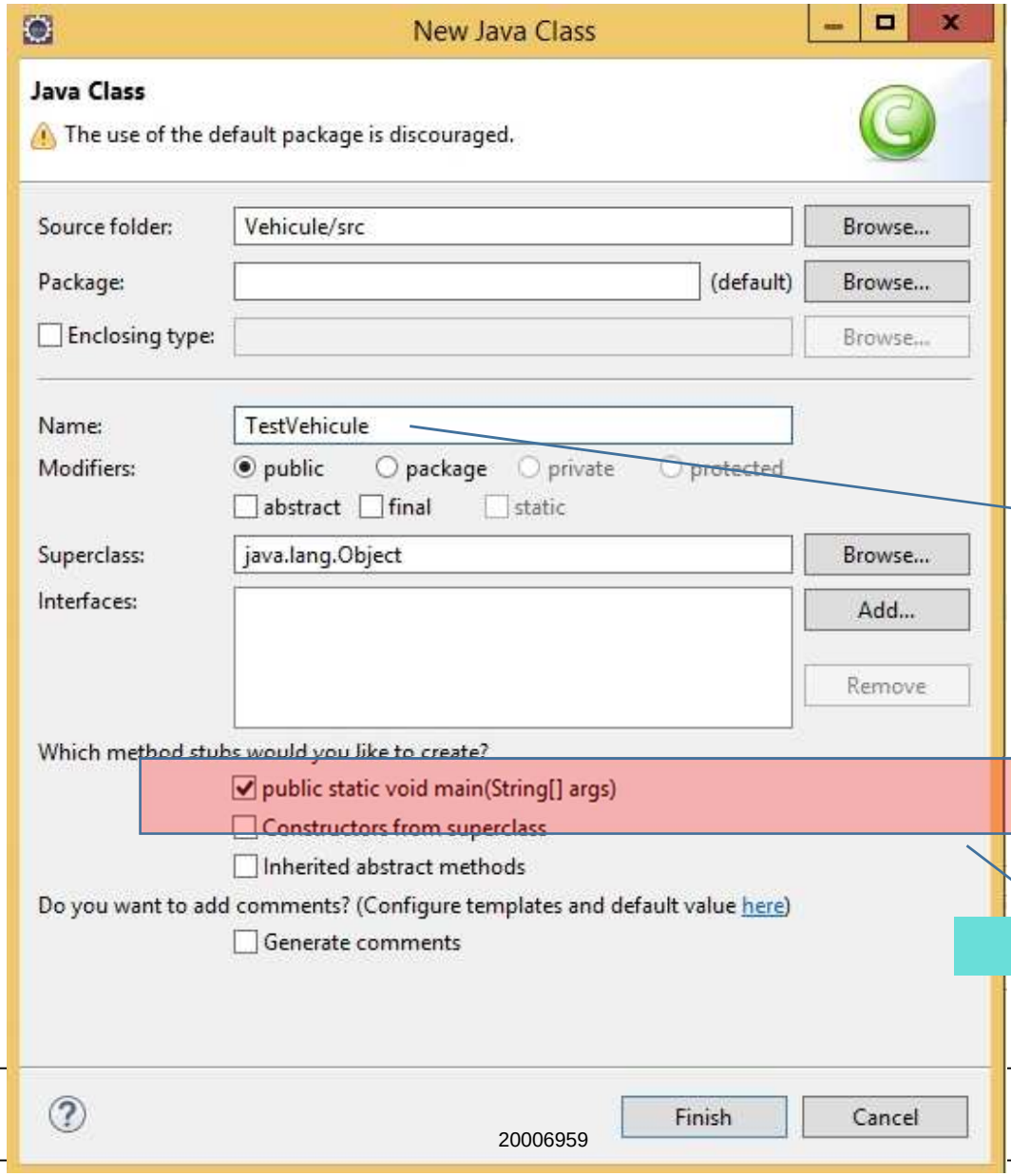


Select Package / new file

Eclipse – File/Class Creation (1/2)



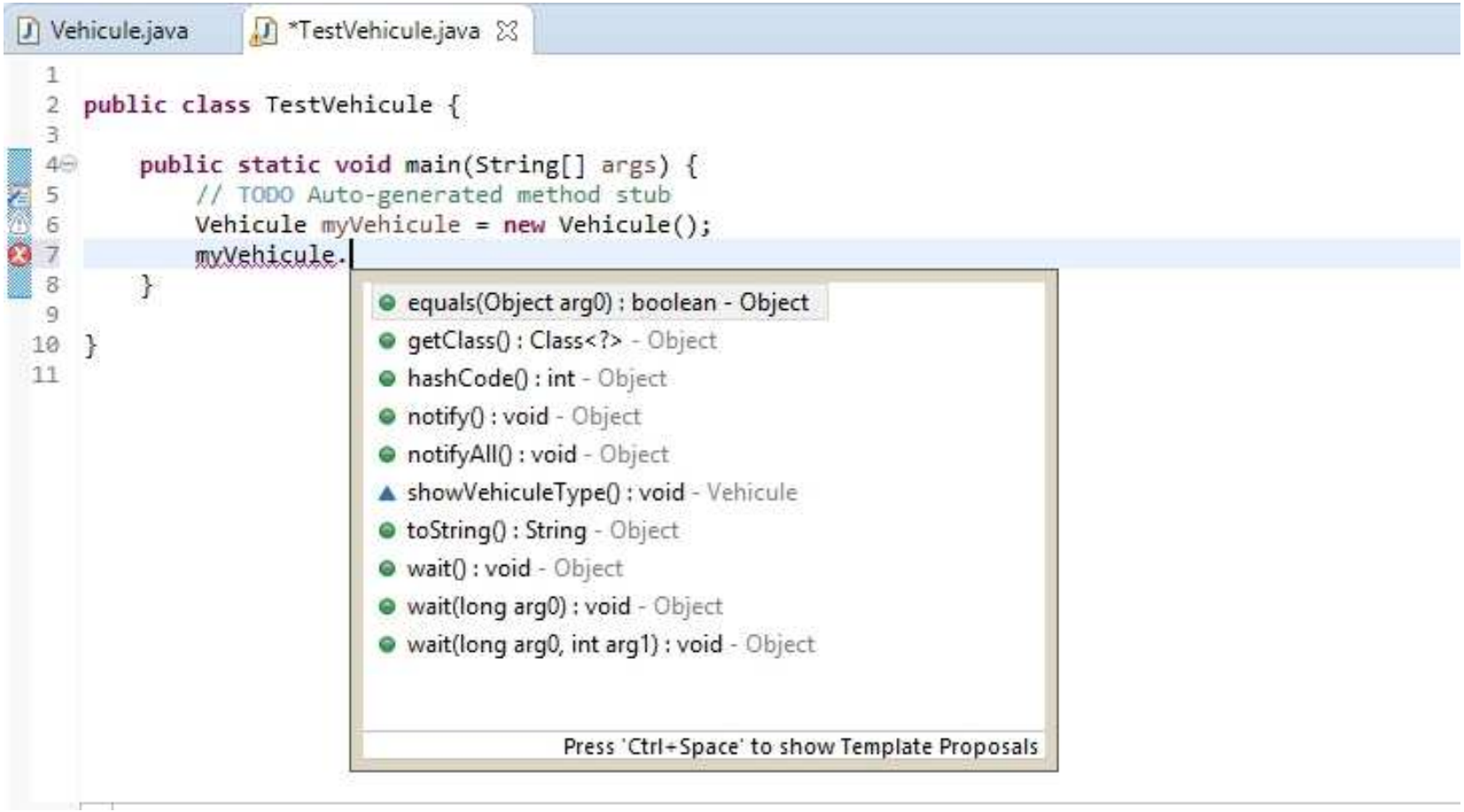
Eclipse – File/Class Creation (2/2)



Class Name

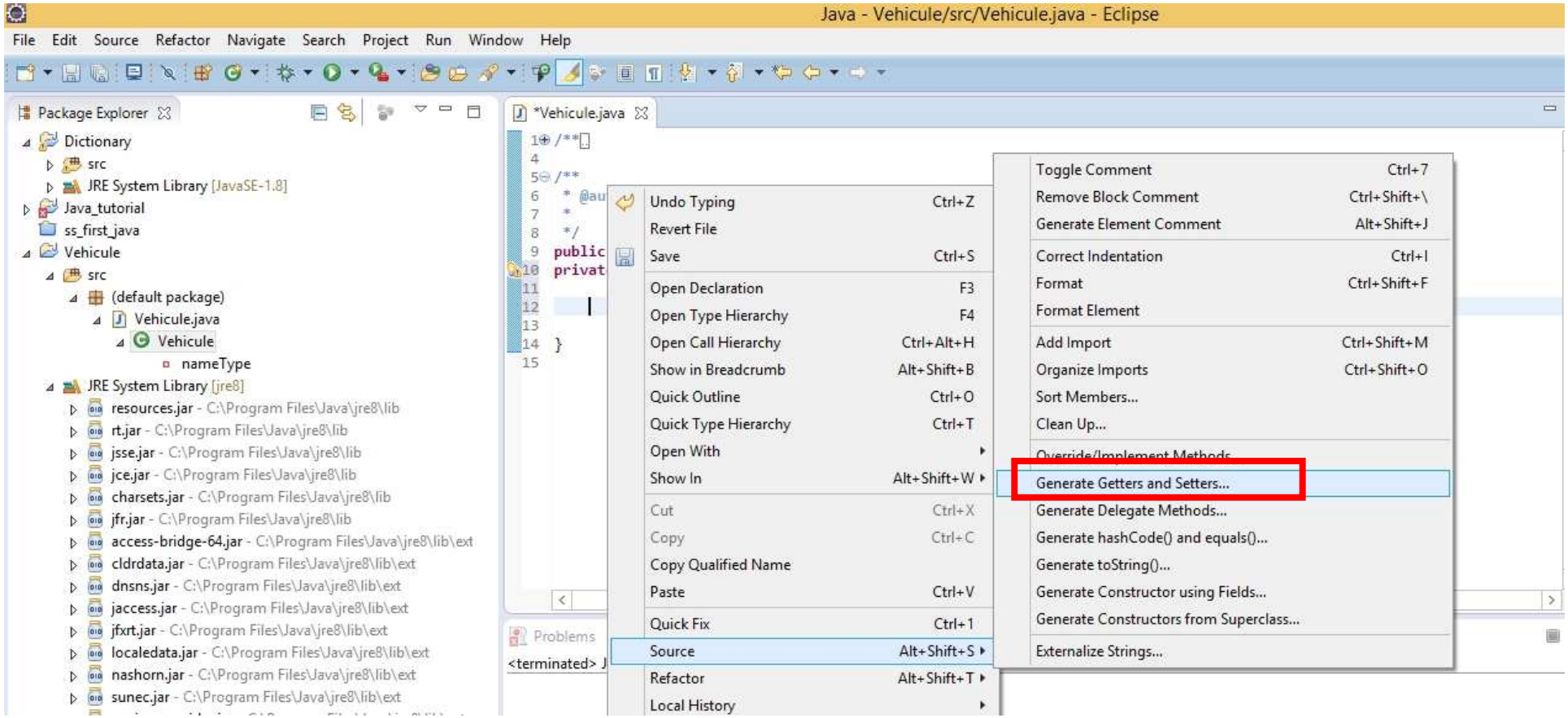
Add the Main

Eclipse – Auto Completion



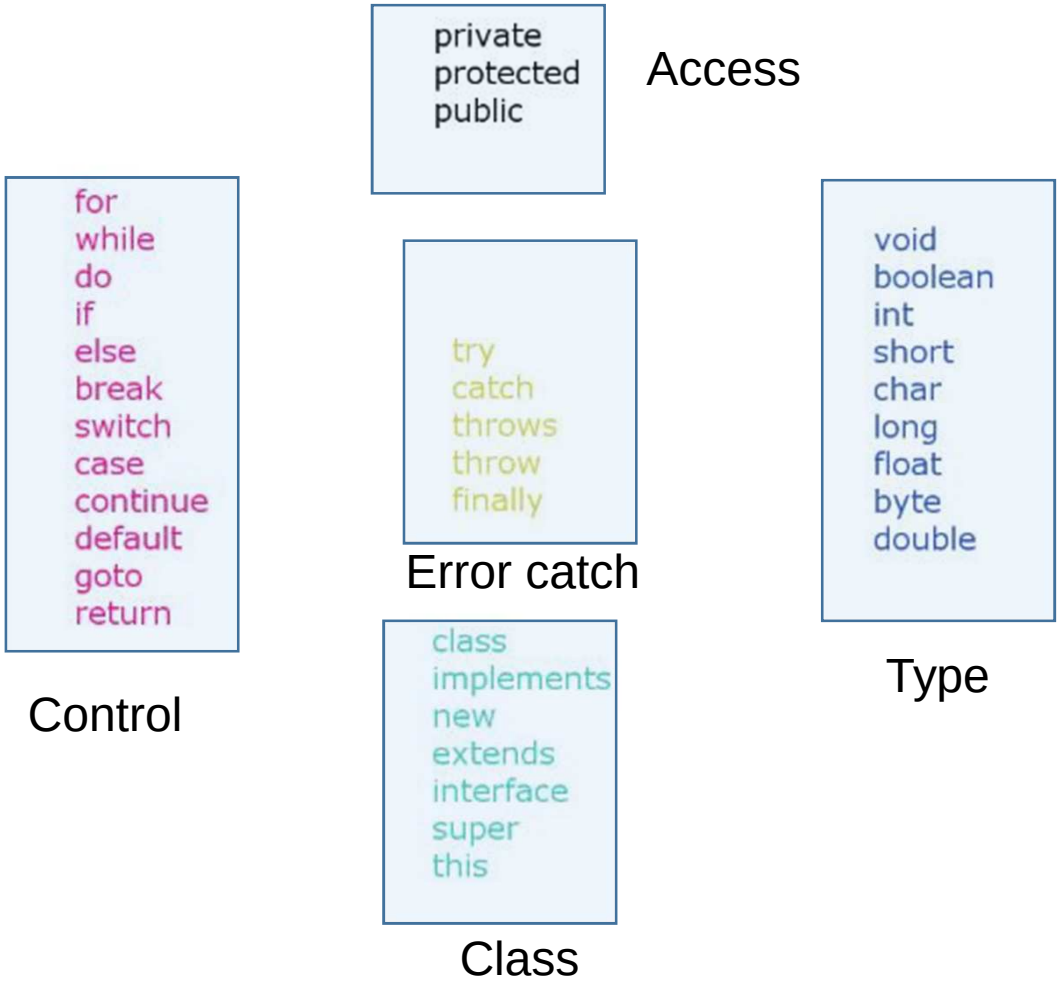
Getters and Setters

- Eclipse has option to generate Getters and Setters
  - From menu / source



Java Basics

Java Keyword List



Java Code

- Indentation is not mandatory as in Python
- Semi colon for each line of code
  - Except when curly brackets

Java Comments

In Java as C++, comments are :

- preceded by two slashes (//) in a line
- enclosed between /* and */ in one or multiple lines.

When the compiler detects //, it ignores all text after // in the same line.

When it detects /*, it scans for the next */ and ignores any text in between

/** used for documentation can be copy past in c++

Vehicle.java | TestVehicle.java

```
1  /**
2   * TestVehicle code to test the Vehicle class
3   * @author ssintes
4   * @version 1.0
5   *
6   */
7
8  /*
9   *
10  * TestVehicle comments
11  */
12 public class TestVehicle {
13
14     public static void main(String[] args) {
15         // TODO Auto-generated method stub
16         Vehicle myVehicule = new Vehicle();
17         myVehicule.showVehicleInfo();
18         Vehicle myCar = new Car();
19         myCar.setSpeed(100);
20         myCar.showVehicleInfo();
21     }
22
23 }
```

Java Package

What is a package?

- A package is a grouping of related classes and interfaces.
- The package name reflects the directory structure where the classes of this package are stored
- The package organize the source code, facilitate re-use, prevent naming conflicts, control access protection
  - Ensure the Class name is unique
    - Vehicle Class in different Package: no Conflict
- Java provides an enormous class library composed a set of packages called Application Programming Interface (API).
- Programmer can re-use classes from existing packages and/or create their own packages

Java built-in packages (Java API)

- Included in the JRE (jar file format)
- Some important Java API packages:
  - *"**java.lang**" : the fundamental classes (Object, Type, String, Class, System, Thread, ...)*
    - *Import is implicit for the java.lang*
  - *"**java.io**": input / output classes (File, PrintStream, ...)*
  - *"**java.net**": internet protocol classes (URL, ServerSocket, SocketAddress,...)*
  - *"**java.awt**" : graphical user interface, image painting (Frame, Label, Panel, ...)*
- Full Java API is available here:

http://docs.oracle.com/javase/8/docs/api/

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
java.awt.image	Provides classes for creating and modifying images.

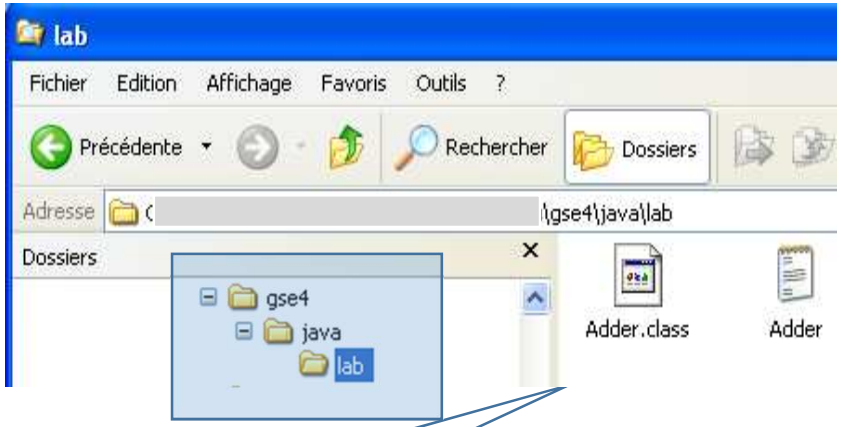
```
//ss import java packages
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.EnumSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Random; //ss to import the Random Class from the java.util package
import java.util.Scanner; //ss to import the scanner class for input from the java.util package
import javax.swing.JFrame;  //ss to import the JFrame class for GUI from the java.util package
```

Creating a package

- To create a package or add a class to your package, use « ***package*** *mypackagename* » as the first statement of your class source file.
- Adding a class Adder to your « gse4.java.lab » package.

Add the *package* statement

```
package gse4.java.lab;

public class Adder{

    public int add(int a, int b){
        int sum = a + b;
        return sum;
    }
}
```



Store the file in a directory structure according to the package name (at least the .class file).

Package naming convention

- Use the reversed internet domain name as the beginning of the package name (avoid package naming conflicts)
  - Generally the domain name is unique
  - Sun Recommendation

- Examples:
  - package com.intel.bluetooth.obex;
  - package com.facebook.api;
  - package com.apple.eawt;
  - ....

Using package members

- Code outside a package can reference any public class or interface from this package using its *fully-qualified name*.
- *Fully-qualified name = packagename.ClassName*

  java.io.File *file = new File("myfile.txt");*

- Code inside a package can reference classes/interfaces from this package using their basic name
- To avoid using fully-qualified name(s), a class outside a package can import package's member(s)

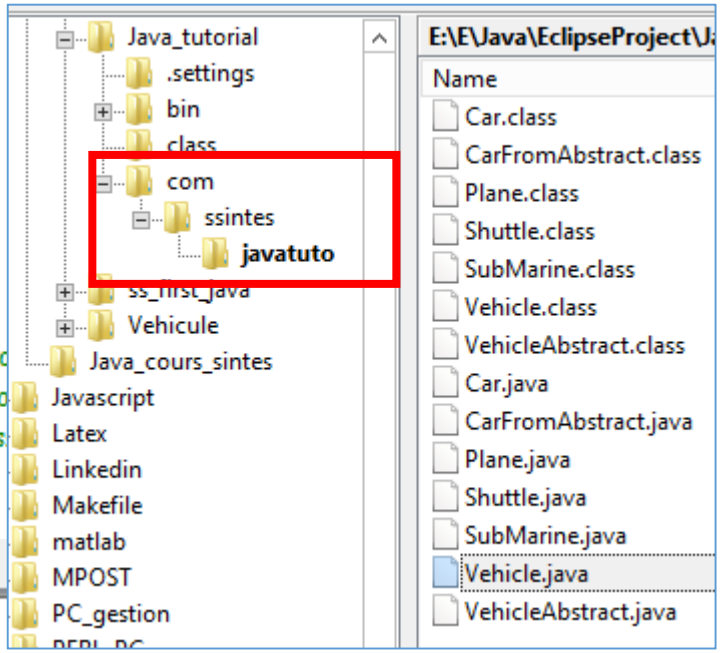
  import java.io.File;

  ....

  File *file = new File(« myfile.txt »);*

Package Usage

```java
//ss import java packages
import com.ssintes.javatuto.*;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.io.BufferedReader;
import java.io.FileReader;
import java.util.EnumSet;
import java.util.LinkedList;
import java.util.List;
import java.util.Random; //ss to import the Random Class fro
import java.util.Scanner; //ss to import the scanner class fo
import javax.swing.JFrame;  //ss to import the JFrame clas
```



Vehicle.java

```java
package com.ssintes.javatuto;
public class Vehicle {
protected String vehicleType;

        void showType() { //ss method showType
                System.out.println("vehicleType = "+vehicleType);
        }

}
```

Package Lab

- LAB3B - Labs\LAB3B_packages.doc

Java Primitive data types

The **byte** primitive data type

- 8-bit signed two's complement integer
- Min = -128 / default=0 / Max = 127
- Example: byte myvar = 23;

The **short** primitive data type

- 16-bit signed two's complement integer
- Min = -32768 / default=0 / Max = +32767
- Example: short myvar = -25000;

The **int** primitive data type

- 32-bit signed two's complement integer
- Min = -2147483648 / default=0 / Max = 2147483647
- Example: int myvar = 1300000;

The **long** primitive data type

- 64-bit signed two's complement integer
- Min = -2^63 / default=0L / Max = 2^63-1
- Example: long myvar = -4000000000;

The **float** primitive data type

- single-precision 32-bit IEEE 754 floating point
- default=0.0f
- Example:

Float pi = 3.141592653f;

The **double** primitive data type

- single-precision 64-bit IEEE 754 floating point
- default=0.0d
- Example: double vehiculeSpeed = 109.9;

The « **boolean** » primitive data type

- 1-bit of information (one byte inside the JVM)
- Only 2 possible values: « true » or « false »
- Default value: « false »
- Example: boolean isADmode = true;

The « **char** » primitive data type

- single 16-bit Unicode character
- Min = 0 / Max = 65535
- => careful single quote
- Example: char gear = 'D';

String

- String is not a type, but a Class
  - It starts with a capital letter String
    - This is a class so the capital letter
  - Each Chain with double quote is an instance of the String class
    - Therefore the new for allocation can be omitted:

```
String common = "My space Ship";
```

```
String myStringExampleOne = "string with double quote";
String myStringExampleTwo = new String("new => string with double quote");
String myStringExampleThree = new String('new => string with simple quote');  //sse not accepted
String myStringExampleFour = 'string with double quote'; //sse not accepted
```

    - Single quote is a character =>error generated
  - String Concatenation with the + operator

```
String firstName = "Sintes";
String surName = "Stephane";
String concatenation = firstName + " " + surName;
System.out.println(concatenation);
```

Constants

```
public static final String INPUT_EXAMLE = "This course is
demonstrating the usage of java regular expression";
public static final String NAME_WITH_EMAIL = "Stephane Sintes
stephane.sintes@gmail.com";
public static final double PI = 3.141592653589793;
```

Print and Println Methods

- System is a Class
- out – is a static member field of System class and it is of type PrintStream
- PrintStream being a class, a static nested class
- Print is the method of PrintStream Class

```
public final class System {
    static PrintStream out;
    static PrintStream err;
    static InputStream in;
    ...
}

public class PrintStream extends FilterOutputStream {
    //out object is inherited from FilterOutputStream class
    public void println() {
    ...
}
```

PrintIn = Print line

```
System.out.print("hello stephane");  //ss print a text
System.out.println("Sintes!");  //ss print a text with carrier return
System.out.print("Hi Stephane\n");
System.out.print("Hi Dude\n");   //ss print a text with carrier return in the text
```

MATH

Basic math functions

- + Addiction
- - Substation
- * Multiplication
- / Division
- % Modulo

Shortcut for increment/decrement operation

- idx = idx + 1;
- idx += 1;
- idx++; //ss post-increment
- ++idx; //ss pre-increment

Same with decrement (--)

Shortcut for math

- result = result * 4;
- result *= 4;
- result = result / 4;
- result /= 4;

Logic operator

- Boolean logical AND : &
- Boolean logical OR : |
- >> : arithmetic right shift
  - 4 >> 1 => 4/2 = 2
- << : arithmetic left shift
  - 4 << 1 = 8 0100 = 2^2 1000 = 2^3 = 8

Classical Exo

- Create a method detecting if the word contains only unique character
- isUniqueChars()

JAVA Variables

Java Variables

- 3 types of java variables
  - Instance Variables
  - Local Variables
  - Class Variables

Usage of Instance & Class variable have been made in Lab2 (UID and counter)

Variable Declaration

- Declaration format:

type identifier [= value, [identifier = value]...]

- Examples of variable declaration:

```
int a, b, c;
double d = 9.2, e = 23.45;
char f, g='a';
```

Instance variables (1/2)

- Also called « non-static fields »
- Declared in a class (outside *constructors* and *methods*)
- Space reserved in the *Heap* at object creation time
- *Access identifier* is associated to an instance variable (*public*, *private* or *protected*)
- Visible in *constructors* and *methods* of the class
- Outside the class, the variable is called with *objectName.variableName*
- Instance variables have default values (numbers=0, boolean=false).
- Values are unique for each instance of the class (each object).
  - Same name but different value

Instance Variables (2/2)

```java
package gse4.java.lab;

public class Adder{

    /*Instance variables visible outside Adder class*/
    public int operandA;
    public int operandB;

    /*Instance variables visible in Adder class only*/
    private int sum;

    /*Default constructor */
    public Adder() {
    }

    /*Constructor 1*/
    public Adder(int a, int b) {
        operandA = a;
        operandB = b;
    }

    /* Method sum(). Returns the sum of operandA and operandB*/
    public int sum(){
        sum = operandA + operandB;
        return sum;
    }
}
```

```java
public class Calculator{

    public static void main (String args[]){

        //Create 2 instances of Adder class
        Adder adder1 = new Adder(1,2);
        Adder adder2 = new Adder();

        //Public instance variables are accessed using objectName.variableName
        adder2.operandA = adder1.sum();
        adder2.operandB = adder1.operandB;
        int sum2 = adder2.sum();
    }
}
```

Local variables

- Declared inside *methods* or *constructors*.
- Created when entering *method* or *constructor.* Destroyed when exiting *method* or *constructor*
- No access modifier. Visible only inside the method or constructor where it is declared.
- Have to be initialized before usage (no default value)
- Space allocated in the "JVM Stack"

```
public int sum(){
    //Local variables
    int a = 5;
    int b = 6;
    int sum = a + b;

    return sum;
}
```

Class variables

- Fields declared with the *static* access modifier
- There is only one copy of the variable for all instances of the class (objects)

```
public class Shape{
    //class variable
    public static int shapeId;
    //Instance variable
    public int shapeNumber;

    //Constructor
    public Shape(){
        shapeId = shapeId + 1;
        shapeNumber = shapeNumber + 1;
    }
}
```

```
public class TestShape {
    public static void main (String args[]){
        Shape shapeA = new Shape();
        Shape shapeB = new Shape();
        Shape shapeC = new Shape();
        System.out.println(shapeA.shapeId);
        System.out.println(shapeB.shapeNumber);
        System.out.println(shapeC.shapeId);
    }
}
```

What is the result of this program?

Class Variable – static access

```
**
 *
 * @author Stephane sintes
 * @version 1.0
 */
public class RectangleFull {
public  double height;
public  double width;
private static int rectCnt;
private int uid;
/**
 * @param height
 * @param width
 */
public RectangleFull(double height, double width) {

     this.height = height;
     this.width = width;
     // this.rectCnt++;  //ssw the static field should be accessed in static way
     //this.uid = this.rectCnt;
     rectCnt++;
     this.uid = rectCnt;
}
```

`the static field should be accessed in static way`

rectCnt is member of the class, not the object, does not make sense to Access it through the instance: `this.rectCnt`

rectCnt is data member associated with the class.

Variable naming convention

- Case sensitive
  int alpha, aLpha, alPHa ; //3 different variables

- Variable cannot start with number: e.g 3rdChoice
- Don't use space character
- If variable name consists in 1 word, use lowercase letters (e.g. int area;)
- If variable name consists in more than one word, all subsequent words should start with a capital letter (e.g. int shapeArea; int currentCarSpeed;)
- Variables storing constant value use capital letters and words are separated with underscore

  e.g. static final int HOURS_PER_DAY = 24;

Constructor and Methods

Constructor (1/2)

- The « Constructor» is used to create instances of a Class (objects).

```
/*Constructor*/
public ClassName (type identifier1, ...,type identifierN){
        //initialize object state...
}
```

- The name of the constructor is the same name of the class
- It is common practice to initialize object's attributes inside the constructor
- A class can specify multiple constructors
- In Java, the constructor is not a method. Constructor has the name of the class and a specific visibility.
- By Default a constructor is package private

Constructor (2/2)

- A class which has no constructor has a default constructor, with no argument and it is equivalent to the following code:

```
public ClassA() {
   super();
}
```

Methods

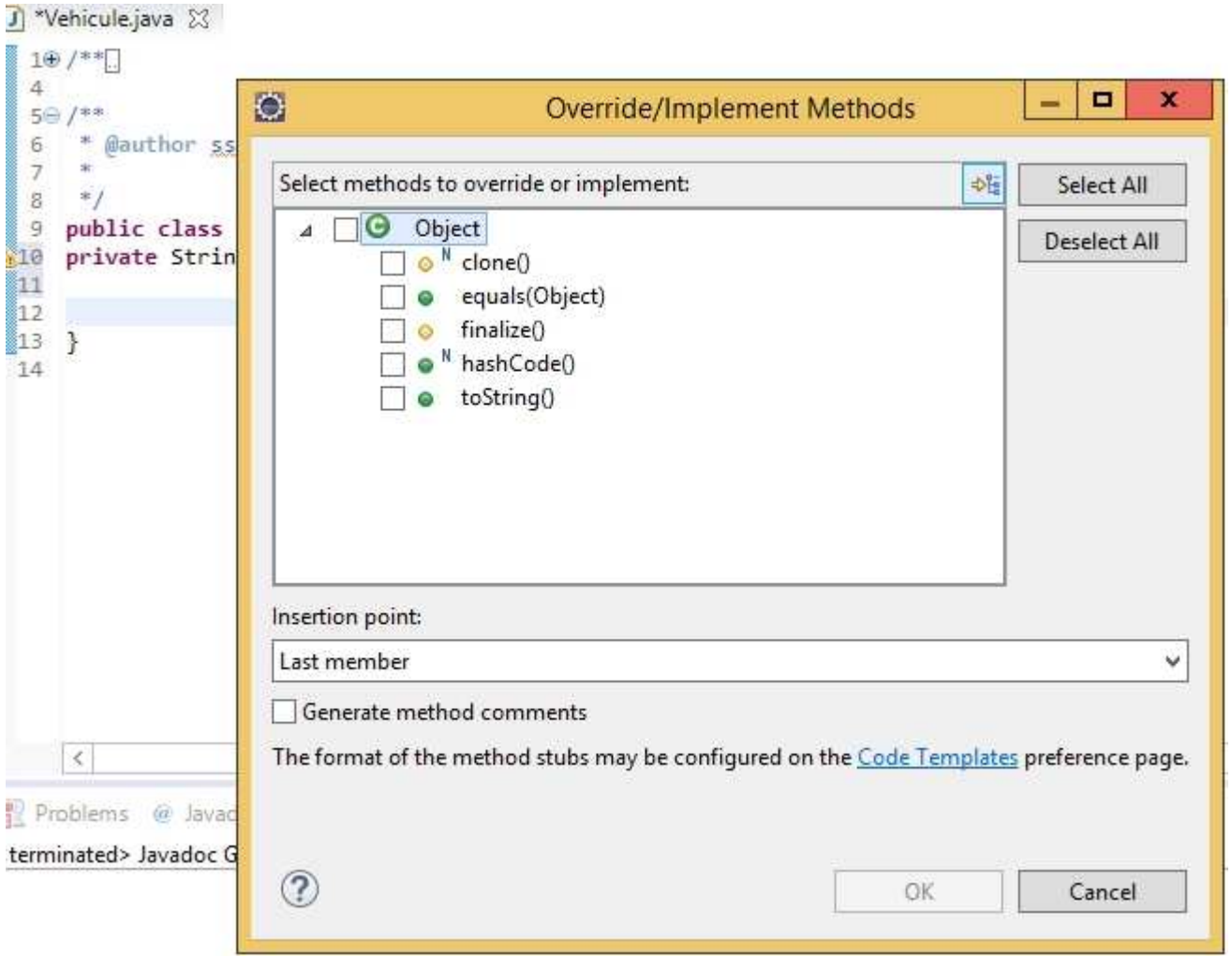
```
<modifier> <return type> <name> (<args>) <exceptions>{
    <method body>
}
```

- Modifier: public, private, protected,...
- Return type
  - data type of the value returned by the method,
  - void if the method does not return data
- Method name: a verb in lowercase. In case of multi-word, first word is a verb in lowercase and following words are nouns or adjectives starting with a capital letter
  - **public double getArea()**
  - **public void setRadius(double radius)**
- Arguments: a list of input parameters in parenthesis, separated by coma
- Exceptions: will be addressed later
- Method body: the behavior of the method, into brackets.

Method Overriding

- Declaring a method in **subclass** which is already present in **parent class** is known as method overriding.
  - methods can be overridden only if they are inherited by the subclass.
  - A method declared final cannot be overridden
    - Final = Full Stop
  - The argument list have to be the same as that of the overridden method.
  - The return type has to be the same as the return type declared in the original overridden method in the super class.
  - The access level cannot be more restrictive than the overridden method's access level. For example: if the super class method is declared public then the overriding method in the sub class cannot be either private or protected.
  - A method declared static cannot be overridden
  - Constructors cannot be overridden

Override Method in Eclipse



LAB3 - INHERITANCE & POLYMORPHISM

- Understanding and using the Inheritance mechanism of Java (extends keyword)
  - Inheritance
  - Method Overriding
  - super method
- base class: java.lang.Object
- Viewing the main method of base class:
  - getClass()
  - toString()
  - equals()
- Exploring polymorphic behaviors in Java applications
- Defining and using abstract classes and interfaces
- Review instanceof java keyword

Instanceof Operator

- instanceof operator is used to check the type of an object at runtime.
- instanceof operator is important when you cast object at runtime as seen in Lab3
- instanceof operator return a boolean value

Drawable drawable1 = **new Rectangle(6, 5);**

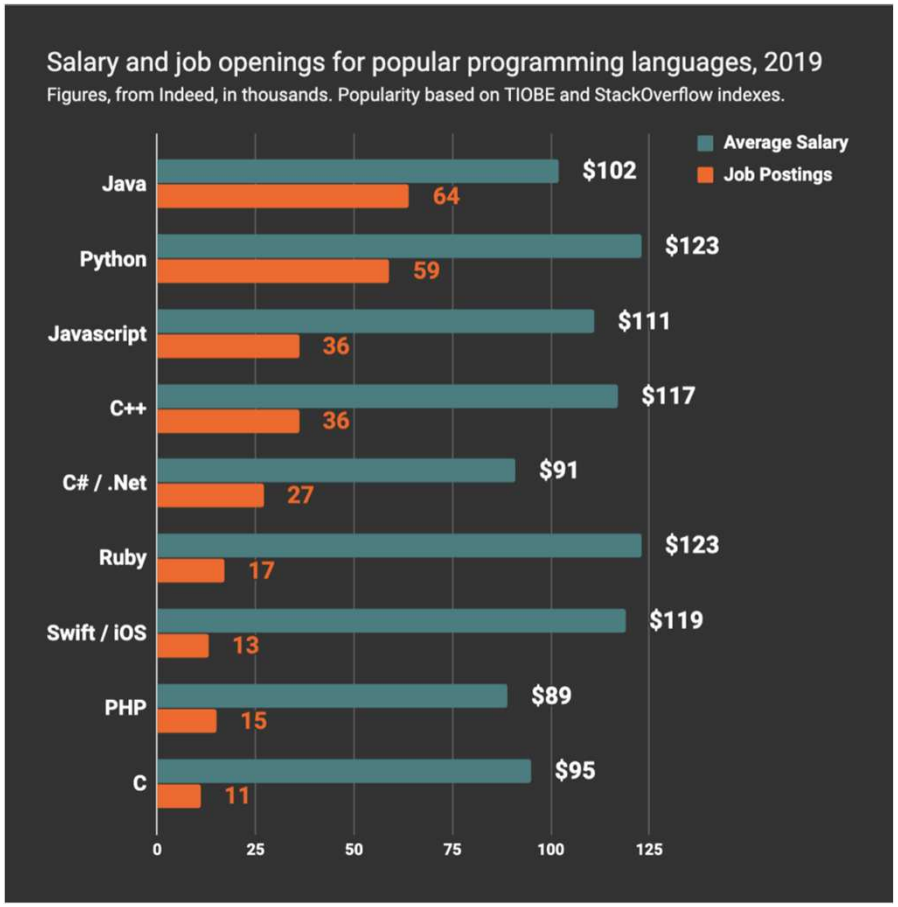
if (drawable1 instanceof Rectangle)

System.***out.println(((Rectangle)drawable1).getArea());***

Java GUI example

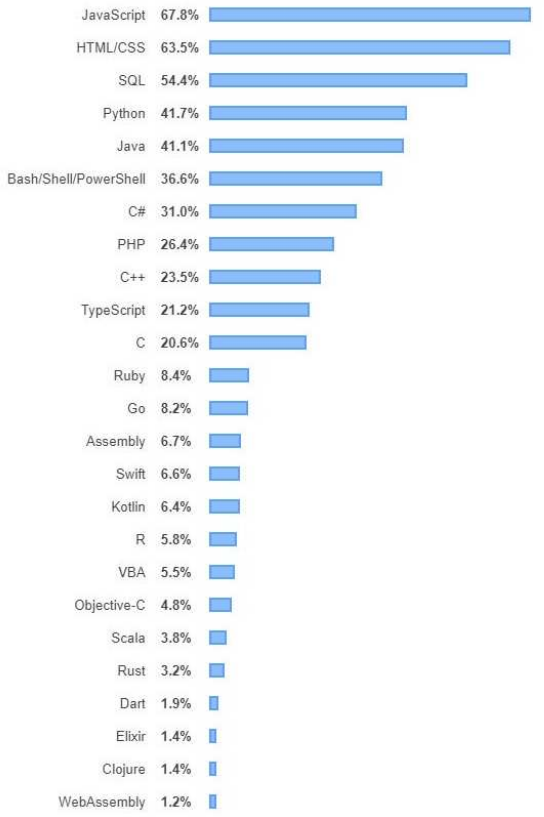
```
import javax.swing.JOptionPane;
public class ssGUIjava {
        public static void main(String[] args){

          String fn = JOptionPane.showInputDialog("Enter first number");
          String sn = JOptionPane.showInputDialog("Enter second number");
          int num1 = Integer.parseInt(fn);
          int num2 = Integer.parseInt(sn);
          int sum = num1 + num2;
          JOptionPane.showMessageDialog(null, "The answer is " + sum, "the title",
JOptionPane.PLAIN_MESSAGE);

        }
      }
```

Programing Languages Ranking



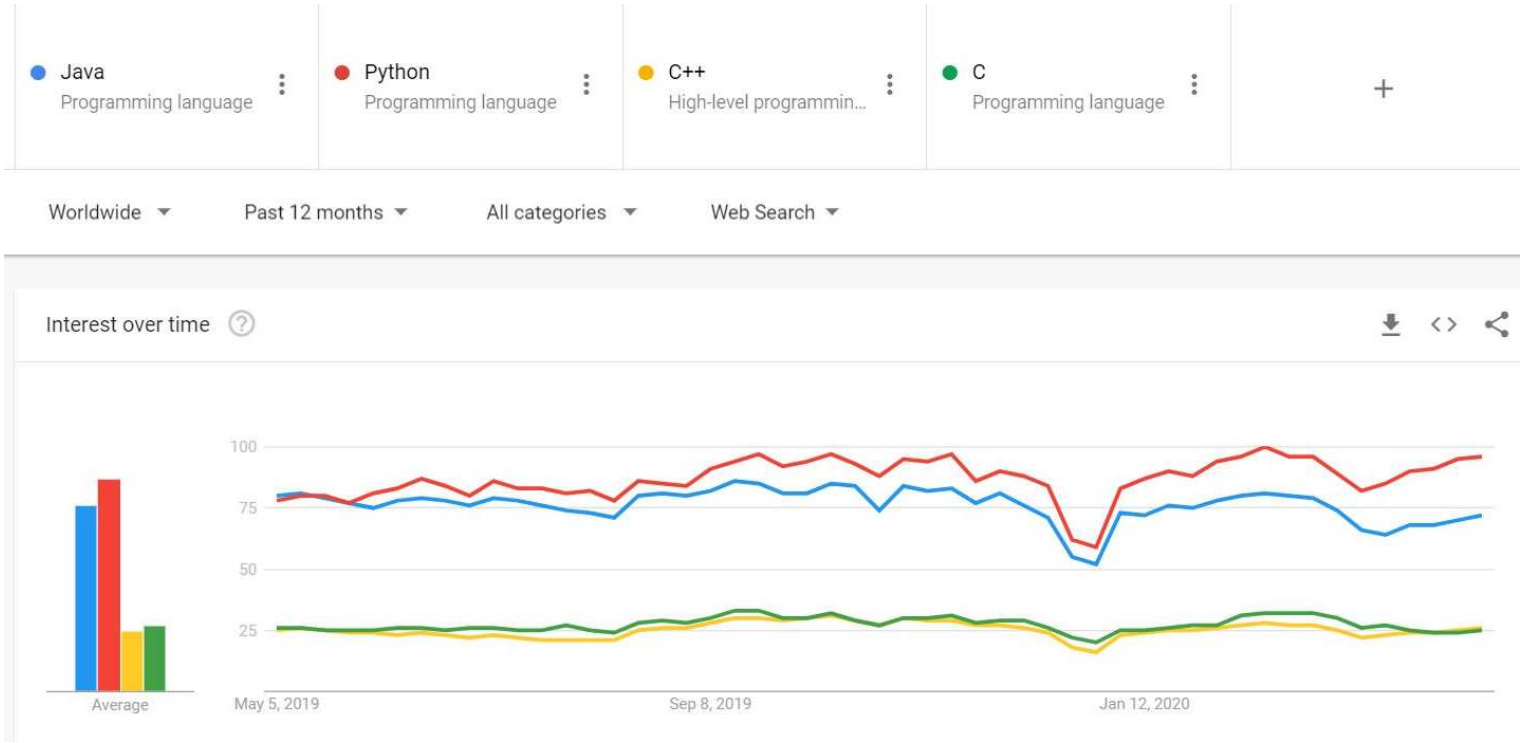
Java still on the top
Python very attractive
C is a niche

Reference:
https://www.codeplatoon.org/the-best-paying-and-most-in-demand-programming-languages-in-2019'



Java/Python/C/C++ trends

Reference:
Google trends



For Keyword (1/3)

```
for( char ch = 'a' ; ch <= 'z' ; ch++ )
System.out.print(ch+"  ");
System.out.print("\n");
 //improved loop

int javaArrayLoop[] = {2,4,6,8,10};
int i=0;

for(int total:javaArrayLoop){//ss you can loop in the array
(same as foreach in perl
System.out.println("javaArrayLoop["+i+"] = " + javaArrayLoop[i]
);
i++;
}
```

: versus ;

For Keyword (2/3)

```
String arrayString[] = {"apple", "lemon", "orange",
"watermelon"};
for(String s : arrayString)
        System.out.println(s);
```

For Keyword (3/3)

```
For (;;;)      infinite loop => careful on usage
```

Do / While Continue Keywords

```
// continue_statement.cpp
do {
      idx++;
      System.out.println( "before the continue\n“ );
      continue;
      System.out.println( "after the continue never print\n“ );
   } while (idx < 3);

   System.out.println("after the do loop\n“);
```

Switch / case / break Keywords

```
switch (input){

case 0:System.exit(0);  //ss   exit for the program
break;
//ss print and println
case 1:
  System.out.print("hello stephane");  //ss print a text
System.out.println("Sintes!");  //ss print a text with carrier return
System.out.print("Hi Stephane\n"); //ss print a text with carrier return
in the text
break;
default:
System.out.println("does not exist");
break;
}
```

break Keyword

```
public void testBreak(){

int i;
boolean isFound = false;
int valToFind = 120;
int[] arrayOfInts = { 52, 97, 3, 800, 589,
120, 1076, 32767,
8, 622, 250,-10 };

for (i = 0; i < arrayOfInts.length; i++) {
if (arrayOfInts[i] == valToFind) {
        isFound = true;
        break;
        }
}

if (isFound) {
System.out.println("Found " + valToFind + " at index " + i);
} else {
System.out.println(valToFind + " not in the array");
}
}
}
```

Label

```
int[][] arrayOfInts = {
            { 32, 87, 3, 589 },
            { -12, 1076, 2000, 8 },
            { 622, 127, 77, 955 }
        };
        int valToFind = -12;

        int i;
        int j = 0;
        boolean isFound = false;

    search:  //ss label
        for (i = 0; i < arrayOfInts.length; i++) {
            for (j = 0; j < arrayOfInts[i].length;j++) {
                if (arrayOfInts[i][j] == valToFind) {
                    isFound = true;
                    break search;
                }
            }
        }

        if (isFound) {
            System.out.println("Found " + valToFind + " at  line" + i + " column " +
j);
        } else {
            System.out.println(valToFind + " not in the array");
        }
```

LAB4 - JAVA CONTROL FLOWSTATEMENTS

Objectives:

- Design simple classes to illustrate Java control flow statements
  - The "*if-then*", "*if-then-else*" conditional statements
  - The "*for*", "*while*", "*do-while*" loop statements
  - The "*switch*" conditional statement

You will create :

- Car
- CarDriver
- RadarSpeedMonitor

To display UI :

import javax.swing.JOptionPane;

JOptionPane.*showMessageDialog*(***null, "Slow Down!!");***

Return Keyword

- The return keyword is used for branching
- The return statement exits from the current method, and control flow returns to where the method was invoked.
- The return statement has two forms:
  - one that returns a value, and one that doesn't.
- The data type of the returned value must match the type of the method's declared return value. When a method is declared void, use the form of return that doesn't return a value.
- Return can be used to exit from nested loop

- How do we exit from nested loop?

Java Static Class / Nested Class

- **Can a class be static in Java ?**
  Yes, there is static class in java.
- In java, we have static instance variables as well as static methods. Classes can also be made static in Java.
  Java allows to define a class within another class. This is a nested class. The class which enclosed the nested class is called Outer class.
- The top level Class cannot be static. ***Only nested classes can be static***.
- Vocabulary:Non-static nested class is also called Inner Class.

Java Static Class / Nested Class

- **static and non-static nested classes differences**
  - Nested static class doesn't need reference of Outer class, but Non-static nested class or Inner class requires Outer class reference.
  - Inner class can access both static and non-static members of Outer class. A static class cannot access non-static members of the Outer class. It can access only static members of Outer class.
  - An instance of Inner class cannot be created without an instance of outer class
  - An Inner class can reference data and methods defined in Outer class , so we don't need to pass reference of an object to the constructor of the Inner class. For this reason Inner classes can make program simple and concise.

Java Static Class / Nested Class:code

```
class TopOuterClass{
   private static String msg = "I am a static variable of Outer Class"; //ss static variable

   // Static nested class
   public static class NestedStaticClass{

      // Only static members of Outer class is directly accessible in nested
      // static class
      public void printMessage() {
        // Try making 'message' a non-static variable, there will be compiler error
        //ss inner class access to static member of Outer class
        System.out.println("I am a nested static class: " + msg);
      }
   }
   // non-static nested class - also called Inner class
   public class InnerClass{

      // Both static and non-static members of Outer class are accessible in this Inner class
      public void display(){
        System.out.println("I am a inner class: "+ msg);
      }
   }
}
class TestNestedStaticClass{
   public static void main(String args[]){

      // create instance of nested Static class
      OuterClass.NestedStaticClass printer = new OuterClass.NestedStaticClass();
      // call non static method of nested static class
      printer.printMessage();
      // In order to create instance of Inner class we need an Outer class
      // instance. Let us create Outer class instance for creating
      // non-static nested class
      TopOuterClass outer = new TopOuterClass();
      TopOuterClass.InnerClass inner  = outer.new InnerClass();
      // calling non-static method of Inner class
      inner.display();
      // we can also combine above steps in one step to create instance of
      // Inner class
      TopOuterClass.InnerClass innerObject = new OuterClass().new InnerClass();

      innerObject.display();
   }
}
```

Class, Object, Instance, Reference

- a **class** is the blueprint for a car. Using this blueprint, you can build as many cars as you like.
- each car you build, or instantiate, is an **object**, also called an **instance**.
- each car also has an address in the memory. To use the car object this address is used, this is the object's **reference**.
- To access to a **member** of the class you use the address, This is called **dereferencing**.

Class, Object, Instance, reference: code

```
Car myRenaultCar = new Car();      // 1 Create instance
callNewBuild(myRenaultCar);          // 2 passing by value
myRenaultCar.toPaint("red")        // 4 Deferencing myRenaultCar

void callNewBuild(Car rentCar) {
   rentCar = new Car();       // 3
      //ss no more access on myRenaultCar

}
```

in line 1. new Car() we request the JVM to build a new car using Car blueprint.
The JVM does so, and returns a reference to the Car built.
You then copy this reference to myRenaultCar. ex myRenaultCar value = a57993 a57993 is the address where the car obj is located
myRenaultCar :**Car@a57993**
In line 2, you give this address a57993 to another method, callNewBuild.

in Line 3, we have a reference Car rentCar. Java is pass-by-value, so the rentCar in callNewBuild method is a copy of the myRenaultCar. So we do not have anymore access to myRenaultCar.

Back in the first method, we dereference myRenaultCar to call toPaint method .
The address of myRenaultCar is unchanged because we passed by value the @ of myRenaultCar

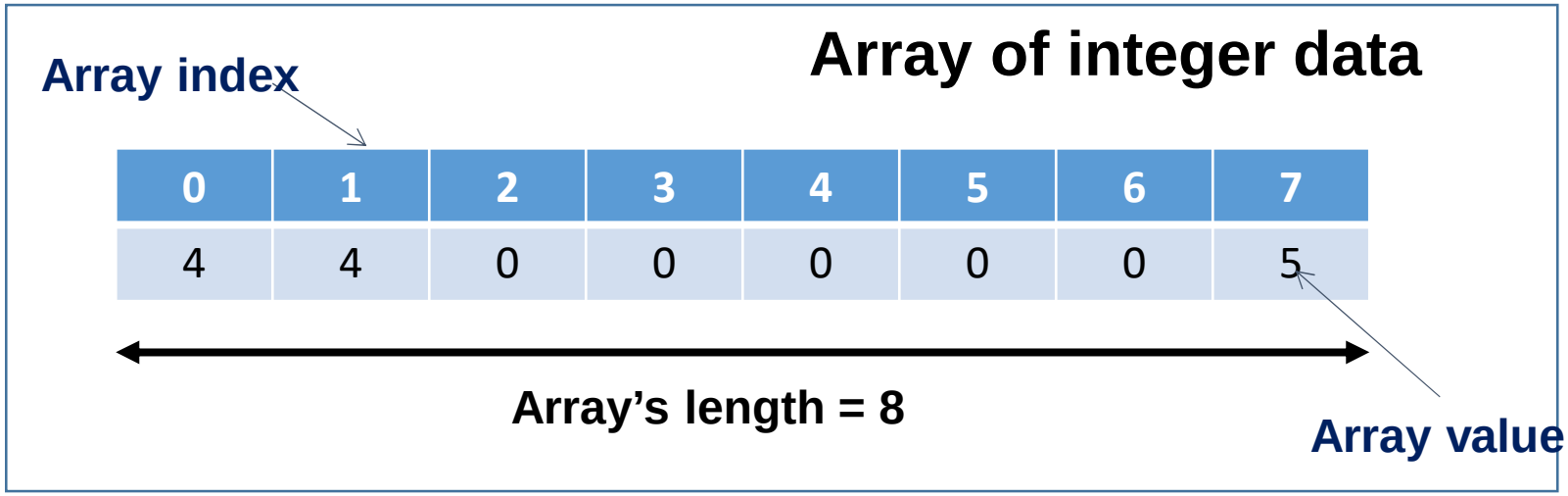
Package Lab

- LAB3B - Labs\LAB3B_packages.doc

Java Arrays

Array (1 dimension)

- An array is an object that contains a fixed number of values of a same given type.
- The length of the array must be defined



Java Array Declaration

```
Declaration example:
int []  javaArray ; //ss preferred array declaration
int javaArray[]; //ss this array declaration works but not
the preferred one   (C++ int cppArray [5] = { 16, 2, 77,
40, 12071 };)
```

```
Declaration example:
int []  javaArray = new int [8];
int []  javaArray = {1,2,17};

.length gives the number of elements:
javaArray.length
```

Java Arrays versus C++ Arrays

In C++, when you declare an array, storage memory for the array is allocated.

In Java, when you declare an array, only the pointer to an array is allocated; storage for the array will be allocated when you use "new"

C++

```
int cppArray[10]; // cppArray is an array of length 10
cppArray[0] = 5;  // set the 1st element of array cppArray
```

JAVA

```
int [] javaArray;     // javaArray is a pointer to an array
A = new javaArray[10]; // now javaArray points to an array of
length 10
javaArray[0] = 8;     // set the 1st element of the array
pointed to by javaArray
```

Array Filling and Exception

```
int []  javaArray = new int [8];

javaArray[0] = 4;
javaArray[1] = 6;
javaArray[7] = 3;
```

Exception :

```
javaArray[8] = 5;
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 8
```

```
int []  javaArray = new int [10]; //ss instantiate array of integers
int i;
for ( i = 0; i < 10; i++ ) //ss fill the array
javaArray[i] = 17+i;


 // Print all the array elements
      for (int data: arrayOfInt) {
         System.out.println(data);
      }
```

Multi Dim Array (2 dimensions)

Declaration example:

```
String [][] arrayOfAnimals;  //ss
arrayOfAnimals = new String[2][4]; //ss create with new operator
arrayOfAnimals[0][0] = “cat”
```

	0	1
0	“cat”	“dog”
1	“horse”	“chicken”
2	“snake”	“bird”
3	“cow”	“frog”

2D arrays used in slide 20007077

Multi Dim Array

```
public static void  generateMultiDimArray() {


int javaMultiDimArray[][]={{1,2,3,4},{-1,-2,-3,-4}}; //ss 2 dimensions array declaration  - use brace for the array and braces for the rows: first row followed by second row

//ss array => .length gives the number of elements, but not the total, if you use 2 dimensional array, it will give the number of rows

System.out.println("javaMultiDimArray length : number of rows =>  "+javaMultiDimArray.length);


System.out.println("javaMultiDimArray[1][2]= "+javaMultiDimArray[1][2]);


for(int row =0;row<javaMultiDimArray.length; row++){

//ss array : you can also  have the length of a row, so this is the number of columns

for(int column =0;column < javaMultiDimArray[row].length; column++){

System.out.println("javaMultiDimArray["+row+"]["+column+"]= "+javaMultiDimArray[row][column]);

}

}

}
```

Catch Error/Exception

int arrayOfInt[] = **new int**[10];

```
try {
System.out.println(arrayOfInt[10]);
        } catch(ArrayIndexOutOfBoundsException error) {
        System.out.println("ArrayIndexOutOfBoundsException cached");
        error.printStackTrace();
 }
```

Lab 6

Objectives:

- Manipulating 1-Dimensional Arrays
  - **Try and Catch**
  - **Sort**
- Manipulating 2-Dimensional Arrays
- Introduction to image processing using Arrays
- Image Rotation

LAB6A_Arrays.doc

Static Keyword

- Static Keyword is used to declare a field, method, or inner class as a class field.
- Classes maintain one copy of class fields regardless of how many instances exist of that class.
- static is also used to define a method as a Class method.
- Class methods are bound to the class instead of to a specific instance, and can only operate on class fields therefore cannot operate on instance field.

Static Method / Instance Methods

- static methods allow you to call class methods without having to create an instance of the class
- The static class methods apply to the class and not to an object
- For "instance methods" the object need to be created first.

Final keyword

- Final keyword is used to define an entity which may only be assigned once.
- A **final method** cannot be overridden or hidden by subclasses. This is used to prevent unexpected behavior from a subclass altering a method that may be crucial to the function or consistency of the class.
- A **final class** cannot be subclassed. Doing this can confer security and efficiency benefits, so many of the Java standard library classes are final, such as java.lang.System and java.lang.String.
- Example:
  - public final class MyFinalClass {...}
  - public class ThisIsWrong extends MyFinalClass {...} // forbidden
- A **final variable** can only be initialized **once**, either via an initializer or an assignment statement.
  - It does not need to be initialized at the point of declaration: this is called a "**blank final**" variable.

```
final int wheel_size;
wheel_size = 30;
```

  - **A blank final** instance variable of a class must be definitely assigned in every constructor of the class in which it is declared; similarly, a **blank final static** variable must be definitely assigned in a static initializer of the class in which it is declared; otherwise, a compile-time error occurs in both cases
  - Unlike the value of a constant, the value of a final variable is not necessarily known at compile time. It is considered good practice to represent final constants in all uppercase, using underscore to separate words.

Java Testing

- What are software tests?
  - Printf
  - SW
- What are the test level?
  - Test Unitaire
  - Test Integration
- What type of test
  - Robustness
  - Performance
- Alpha – Code functionnal 50% test
- Beta – 80% test passing
- What is CUT? Code Under Test
- What is a DUT? Device under Test

Lab7 Java Testing

Objectives:

- Understand Unit Testing
- Unit test a simple Java classes
- Use Junit lib (JUnit framework)
- Use Assert

Generic java class<?>

```
Box<Integer> integerBox = new Box<>();
Box<String> stringBox = new Box<>();
```

- java class<?>
- *unbounded wildcard*.
- generics enable *types* (classes and interfaces) to be parameters when defining classes, interfaces and methods

```
class TupleDict<K, V>
{
  private final K k;
  private final V v;

  public Tuple(K key, V value) {
    k = key;
    v = value;
  }

  public String toString() {
    return String.format("KEY: '%s', VALUE: '%s'", k, v);
  }

}
```

```
public class TestGenericTuple {

  public static void main(String[] args) {
 TupleDict <String, Integer> t = new TupleDict<String, Integer>("email", 1);
    System.out.println(t);
  }

}
```

Input from Keyboard (1/2)

- Need to instantiate a Scanner object from standard input: **System.in**

```
import java.util.Scanner;
System.out.print("Enter your value: ");
in = new Scanner(System.in);//ssa instantiate class Scanner
System.out.println(in.nextLine()); //ss enter a data followed by carrier return
```

Input from Keyboard (2/2)

```
System.out.println("Enter a string");
String s = in.nextLine();
System.out.println("You entered string "+s);

System.out.println("Enter an integer");
int a = in.nextInt();
System.out.println("You entered integer "+a);

System.out.println("Enter a float");
float b = in.nextFloat();
System.out.println("You entered float "+b);
System.out.println("Enter a Double");
double db = in.nextDouble();
System.out.println("You entered a Double "+db);
```

Input from Keyboard - Char

```
System.out.println("Enter one single
character/letter :");
String s2 = in.nextLine();
char c = s2.charAt(0);
System.out.println("Here the character : " + c);
```

Javaaa
Javaa
Java
Not Javaaaa
The course in in Java. => not
Javaaa is super.

Java* => bad
Java+
Java{1,3}
^Java{1,3}

JAVA Regular Expression (1/3)

Regular Expression	Description
.	Matches any character
^regex	Finds regex that must match at the beginning of the line.
regex$	Finds regex that must match at the end of the line.
[abc]	Set definition, can match the letter a or b or c.
[abc][vz]	Set definition, can match a or b or c followed by either v or z.
[^abc]	When a caret appears as the first character inside square brackets, it negates the pattern. This pattern matches any character except a or b or c.
[a-d1-7]	Ranges: matches a letter between a and d and figures from 1 to 7, but not d1.
X|Z	Finds X or Z.
XZ	Finds X directly followed by Z.

JAVA Regular Expression (2/3)

Regular Expression	Description
\d	Any digit, short for [0-9]
\D	A non-digit, short for [^0-9]
\s	A whitespace character, short for [\t\n\x0b\r\f]
\S	A non-whitespace character, short for [^\s]
\w	A word character, short for [a-zA-Z_0-9]
\W	A non-word character [^\w]
\S+	Several non-whitespace characters
\b	Matches a word boundary where a word character is [a-zA-Z0-9_].

JAVA Regular Expression (3/3)

Ce prof est super

Regular Expression	Description	Examples
*	Occurs zero or more times, is short for {0,}	X* finds no or several letter X, .* finds any character sequence
+	Occurs one or more times, is short for {1,}	X+ - Finds one or several letter X
?	Occurs no or one times, ? is short for {0,1}.	X? finds no or exactly one letter X
{X}	Occurs X number of times,{} describes the order of the preceding liberal	\d{3} searches for three digits, .{10} for any character sequence of length 10.
{X,Y}	Occurs between X and Y times,	\d{1,4} means \d must occur at least once and at a maximum of four.
*?	? after a quantifier makes it a *reluctant quantifier*. It tries to find the smallest match. This makes the regular expression stop at the first match.	([a-zA-Z0-9\s])

JAVA Regular Expression

```
package regex;
import java.util.regex.Matcher;
import java.util.regex.Pattern;


public class Regex {
public static final String INPUT_EXAMLE = "This course is demonstrating the usage of java regular expression";
public static final String NAME_WITH_EMAIL = "Stephane Sintes stephane.sintes@gmail.com";

public static void main(String[] args) {
//ss Compiles the given regular expression into a pattern with the given flags.
Pattern pattern = Pattern.compile("\\w+", Pattern.CASE_INSENSITIVE);
System.out.println("Pattern.CASE_INSENSITIVE="+Pattern.CASE_INSENSITIVE+" pattern.flags()="+pattern.flags());
//ss Creates a matcher that will match the given input against this pattern:
Matcher matcher = pattern.matcher(INPUT_EXAMLE);
System.out.println(matcher);
System.out.println(INPUT_EXAMLE);
while (matcher.find()) {  //ss Attempts to find the next subsequence of the input sequence that matches the
pattern.

System.out.print(matcher.group()); //ss Returns the input subsequence matched by the previous match.
System.out.println("=>"+"Between  " + matcher.start() + " and "+  matcher.end() + " ");
//ss  matcher.start() Returns the start index of the previous match.
//ss  matcher.end() Returns the offset after the last character matched.


}
}
}
```

JAVA Regular Expression - Lab

- Add a method that count the number of words of the sentence
- Add a method remove email address from the input text with using the replaceAll method

Java versus C++ (1/2)

- Already familiar with C++ object-oriented programming language? What's different with Java?

C++	JAVA
Compiled	Interpreted / Compiled
Speed : ++	Speed : -- (JIT to accelerate)
pointer	No pointer
Memory Handling	No Memory Handling (Garbage Collection)
Compatible C code	Strongly influenced by C++/C syntax
Write once, compile anywhere (WOCA)	Write Once Run Anywhere (WORA)
Runs as native executable machine	Runs in a virtual machine
Single and Multiple inheritance of classes	Single inheritance of classes. Supports multiple inheritance through the Interfaces construct

Java versus C++ (2/2)

C++	JAVA
Operator overloading for most operators.	Operators are not overridable The language overrides + and += for the String class.
const keyword for defining immutable variables and member functions that do not change the object	“final” provides a version of const Not necessarily known at compile time
Supports the goto statement	Supports labels with loops and statement blocks

Class Files in Specific folder

- E:\E\Java\EclipseProject\Java_tutorial>javac -g -d classfolder ss_first.java

All generated class files are put in the classfolder folder

$ cd class

$ java MainMenu

Back-up

Java Specific Class - Class Class

- java.lang.Class

- getName() is a method of the Class class
- date.getClass().getName().

Javac options (1/2)

E:\E\Java\EclipseProject\Java_tutorial>javac -help

Usage: javac <options> <source files>

where possible options include:

-g Generate all debugging info

-g:none Generate no debugging info

-g:{lines,vars,source} Generate only some debugging info

-nowarn Generate no warnings

-verbose Output messages about what the compiler is doing

-deprecation Output source locations where deprecated APIs are used

-classpath <path> Specify where to find user class files and annotation processors

-cp <path> Specify where to find user class files and annotation processors

-sourcepath <path> Specify where to find input source files

-bootclasspath <path> Override location of bootstrap class files

-extdirs <dirs> Override location of installed extensions

-endorseddirs <dirs> Override location of endorsed standards path

-proc:{none,only} Control whether annotation processing and/or compilation is done.

-processor <class1>[,<class2>,<class3>...] Names of the annotation processors to run; bypasses default discovery process

Javac options (2/2)

-processorpath <path> Specify where to find annotation processors

-parameters Generate metadata for reflection on method parameters

-d <directory> Specify where to place generated class files

-s <directory> Specify where to place generated source files

-h <directory> Specify where to place generated native header files

-implicit:{none,class} Specify whether or not to generate class files for implicitly referenced files

-encoding <encoding> Specify character encoding used by source files

-source <release> Provide source compatibility with specified release

-target <release> Generate class files for specific VM version

-profile <profile> Check that API used is available in the specified profile

-version Version information => **javac 1.8.0_45**

-help Print a synopsis of standard options

-Akey[=value] Options to pass to annotation processors

-X Print a synopsis of nonstandard options

-J<flag> Pass <flag> directly to the runtime system

-Werror Terminate compilation if warnings occur

@<filename> Read options and filenames from file

JAVAC Xlint

- The JDK 5.0 add the option-Xlint to the compiler to check issue/warning in the code :
- -Xlint or -Xlint check everything

- E:\E\Java\EclipseProject\Java_tutorial>javac -Xlint ss_first.java
  - ss_first.java:290: warning: auxiliary class JFrameClass in ssGUIjava.java should not be accessed from outside its own source file
  - JFrameClass objGUI = new JFrameClass(); //ssa
    instantiate class JFrameClass ^

JAVAC non standard option

- javac –X lists the non standard option

Java Specific Class - Object Class

- java.lang.Object
- getClass() is a method that returns a Class instance / Class object
  - A Class object is created by the ClassLoader the first time it is the *.class file is loaded
- ***date***.getClass() returns an instance of Class<Date>
- Date date = **new Date (5, 2, 2016);**
- ***date.getClass());***

JME

- Java 2 'Micro Edition' (J2ME) is a Java-based application platform for mobile devices developed by Sun Microsystems, as it then was.
- Unlike Android it is not a complete platform, just a virtual machine
- J2ME is very limited for powerful handsets so explaining the success of Android:
  - Android Java has a complete proprietary user interface mode
  - Android has floating point support
  - Android resolved part of security concerns (VM)

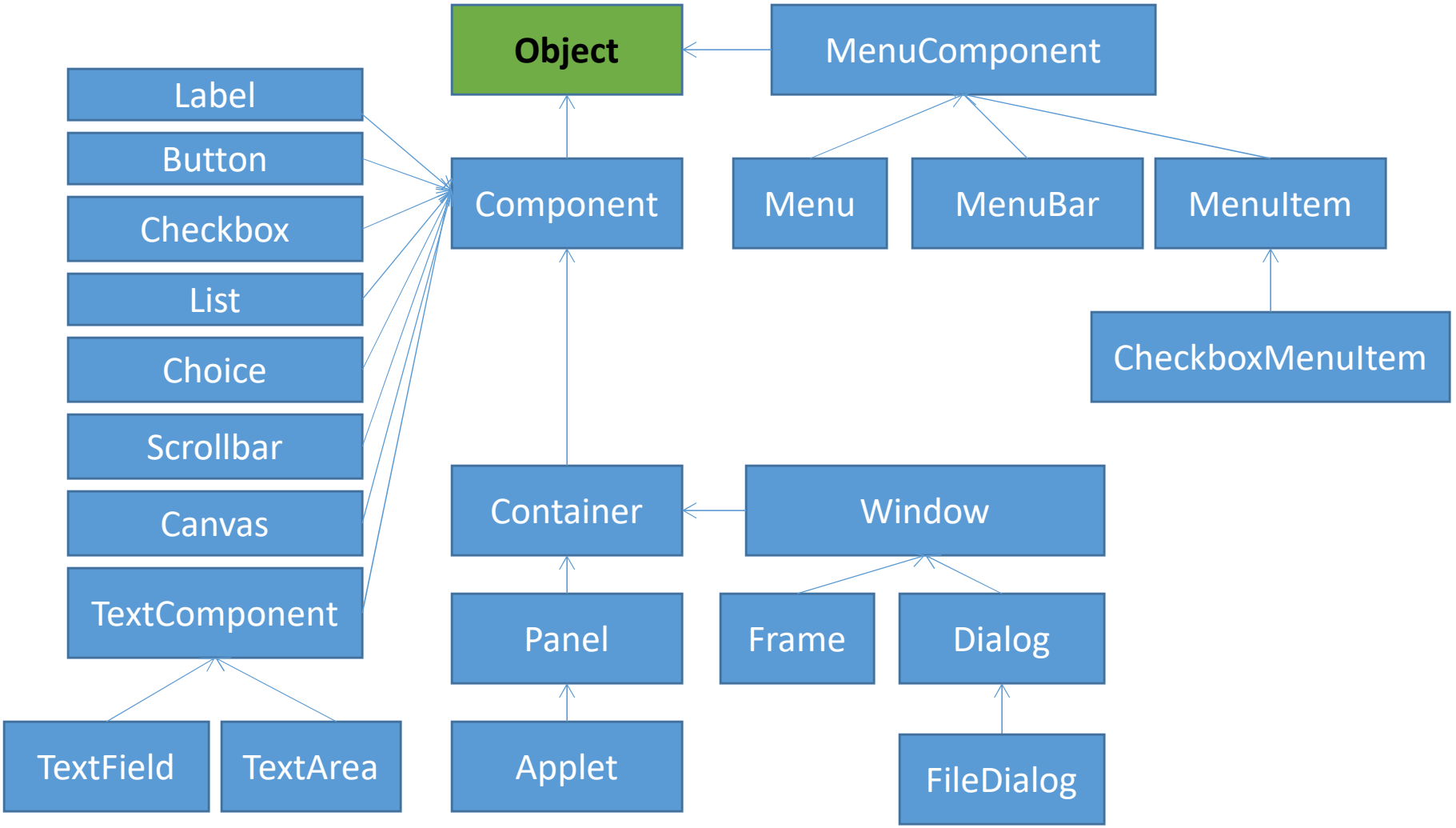
JAVA Graphical User Interface (GUI)

Package « java.awt »

- Abstract Windowing Toolkit

- Provide classes for designing user interfaces , for painting graphics and images

AWT Components Hierarchy



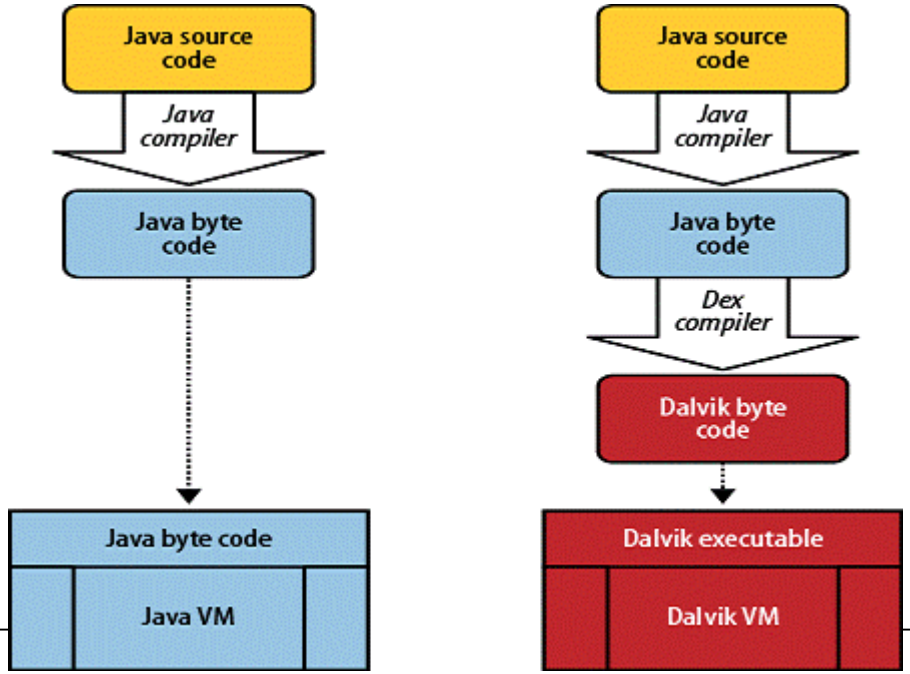
Introduction to
Android Applications Development

What is Android?

- Android is an open source software stack, designed for mobile devices (Phone, Tablets...)
- Based on Linux kernel
- Code released under the Apache License.
- Originally developed by Android Inc. company, bought by Google in 2005 and unveiled in 2007 along with the Open Handset Alliance
- Intended to compete with Symbian OS and Apple iOS
- Oct. 2008, 1st Android phone: HTC Dream (G1)

Android Java usage

- Android uses the dalvik VM and now (ART : Android Runtime)
- Dalvik VM is a "register based" VM, as opposed to java VM, which is "stack based "
- The Java bytecode does not run directly in the VM

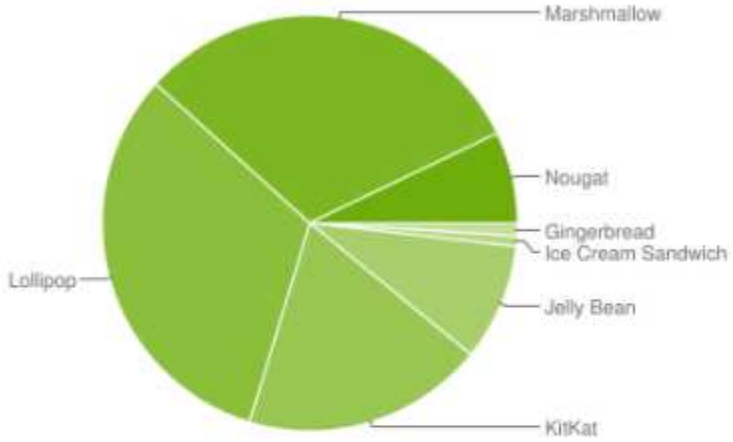


Android Java usage

- Dalvik is "register based" for performance reasons:
  - avoid unnecessary memory access
  - 30% fewer instructions

Android versions usage share

Version	Codename	API	Distribution
2.3.3 - 2.3.7	Gingerbread	10	1.0%
4.0.3 - 4.0.4	Ice Cream Sandwich	15	0.8%
4.1.x	Jelly Bean	16	3.2%
4.2.x		17	4.6%
4.3		18	1.3%
4.4	KitKat	19	18.8%
5.0	Lollipop	21	8.7%
5.1		22	23.3%
6.0	Marshmallow	23	31.2%
7.0	Nougat	24	6.6%
7.1		25	0.5%



Source:
http://developer.android.com/about/dashboards/index.html

Android market share

Phones

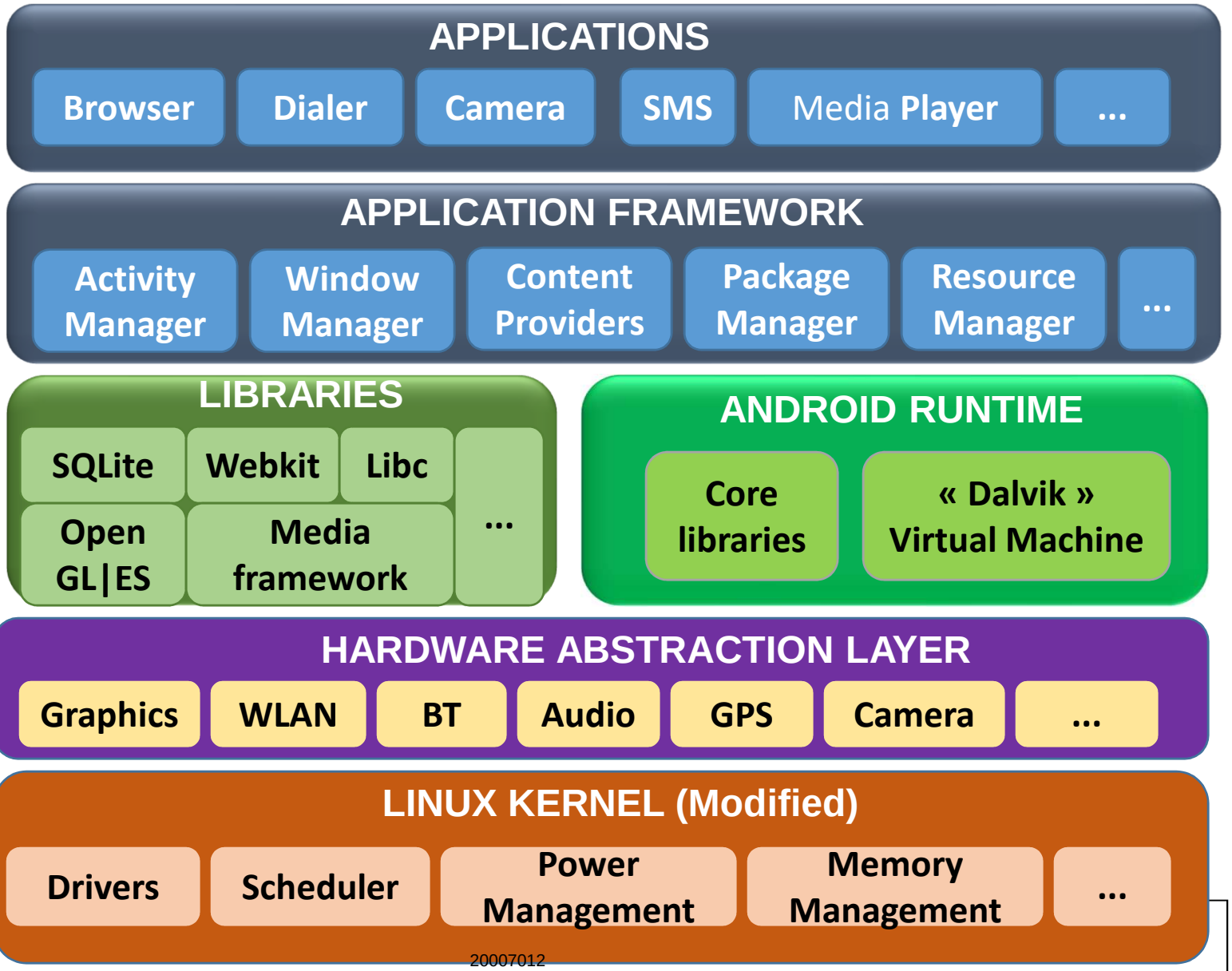
Vendor	4Q12 Unit Shipments	4Q12 Market Share	4Q11 Unit Shipments	4Q11 Market Share	Year over Year Change
Android	159.8	70.1%	85.0	52.9%	88.0%
iOS	47.8	21.0%	37.0	23.0%	29.2%
BlackBerry	7.4	3.2%	13.0	8.1%	-43.1%
Windows	6.0	2.6%	2.4	1.5%	150.0%
Linux	3.8	1.7%	3.9	2.4%	-2.6%
Others	3.0	1.3%	19.5	12.1%	-84.6%
Total	227.8	100.0%	160.8	100.0%	41.7%

Tablets

Vendor	1Q13 Unit Shipments	1Q13 Market Share	1Q12 Unit Shipments	1Q12 Market Share	Year-over-Year Growth
Android	27.8	56.5%	8.0	39.4%	247.5%
iOS	19.5	39.6%	11.8	58.1%	65.3%
Windows	1.6	3.3%	0.2	1.0%	700.0%
Windows RT	0.2	0.4%	0.0	N/A	N/A
Others	0.1	0.2%	0.2	1.0%	-50.0%
Total	49.2	100.0%	20.3	100.0%	142.4%

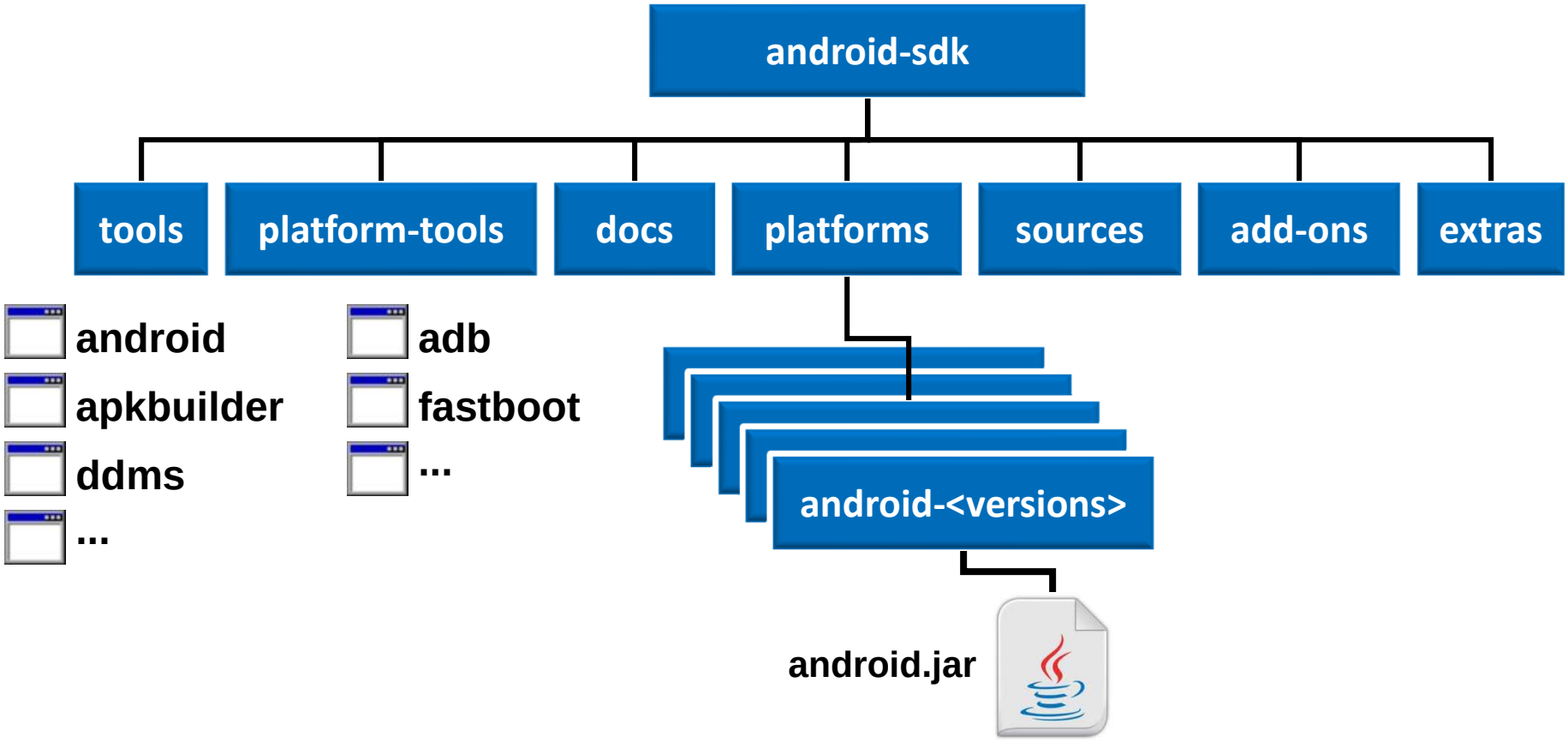
source: http://www.idc.com/

Android Architecture



Android Software Development Kit

- Android SDK provides API libraries, development tools to build, test and debug applications



Android Library

<sdk>/platforms/android-17/android.jar

```
META-INF/
META-INF/MANIFEST.MF
res/
res/raw-xlarge/
res/raw-xlarge/incognito_mode_start_page.html
res/raw-ko/
res/raw-ko/incognito_mode_start_page.html
res/raw-ko/loaderror.html
res/raw-ko/nodomain.html
res/layout-sw600dp-port/
res/layout-sw600dp-port/keyguard_host_view.xml
res/layout-sw600dp-port/keyguard_status_area.xml
res/raw-sl-xlarge/
res/raw-sl-xlarge/incognito_mode_start_page.html
```

Android Applications: The Basics

- Written in Java programming language
- Code compiled with SDK tools and packaged along with data / resource (images, etc...) files in an archive file used for installation on devices: the *Android Package (".apk" extension).*
- Android OS is a multi-user Linux system in which each application is seen as a different user (unique user ID)
- Each application runs in its own Linux process
- Each process has its own Virtual Machine (applications are isolated from each other)
- This way, applications have minimal privilege on the system and can only access parts of the system required to do their work (secure environment).

Application components: Activities

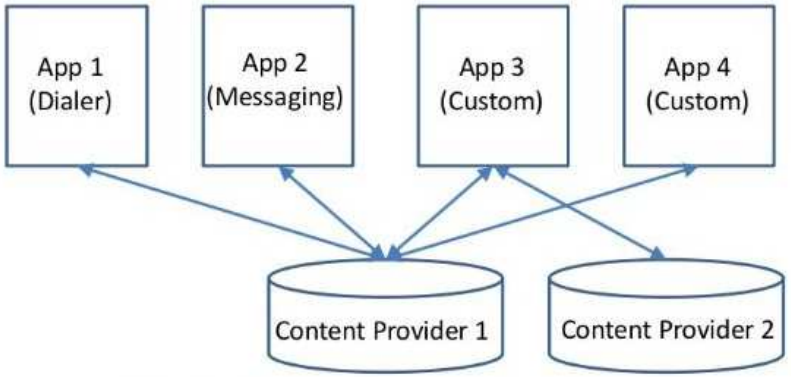
- An ***activity*** is defined as “a single, focused thing the user can do”.
- Activities are usually designed to interact with end users; therefore an activity often represents **a single screen** including graphical user interface components.
- http://developer.android.com/reference/android/app/Activity.html

Application components: Services

- A ***service*** is a process running in background to perform operations without blocking user interaction with the device (same as windows)
- A service does not provide a graphical interface to interact with user.
- Example: a service enables Android users to listen to music (background), while browsing the web (activity in foreground)
- http://developer.android.com/reference/android/app/Service.html

Application components: Content Providers

- *A **content provider*** is used to manage application data (images, videos, contacts, ...) in a structured way.
- Application data can be stored either on filesystem, SQLite database, on a web server, etc...
- Data managed through content providers can be shared between applications (read and modify)
- http://developer.android.com/guide/topics/providers/content-providers.html



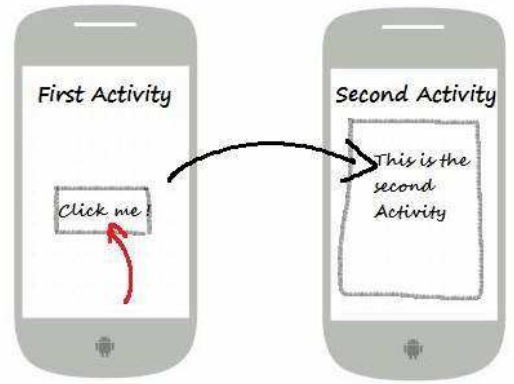
Application components: Broadcast Receivers

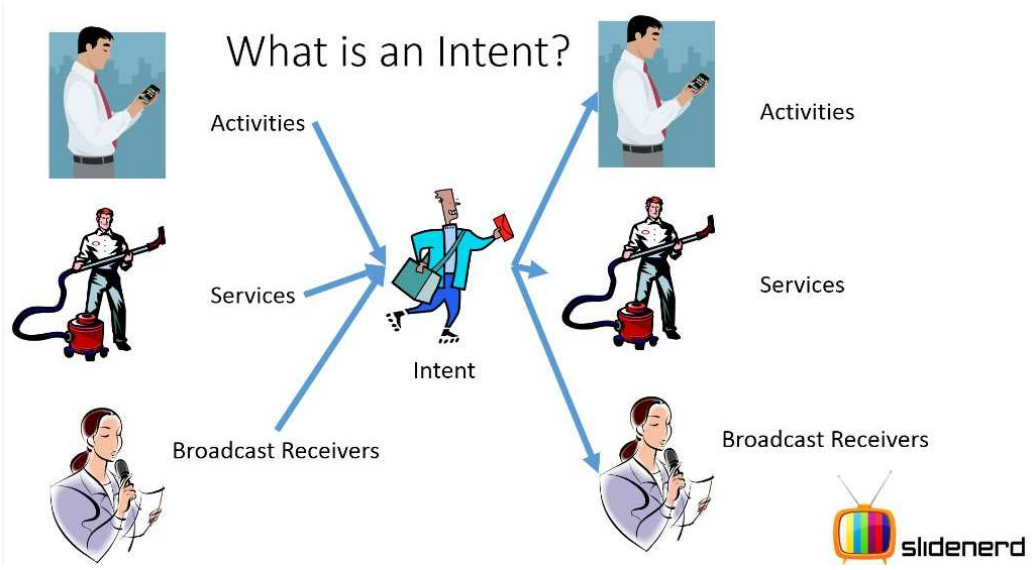
- A ***broadcast receiver*** component is used to catch and respond to "messages" broadcasted by the system: http://developer.android.com/reference/android/content/BroadcastReceiver.html
- An application can also broadcast its own "custom messages" to other applications.
- Broadcasted messages are delivered as **Intent** objects: http://developer.android.com/reference/android/content/Intent.html

```
android.app.action.ACTION_PASSWORD_CHANGED
android.app.action.ACTION_PASSWORD_EXPIRING
android.app.action.ACTION_PASSWORD_FAILED
android.app.action.ACTION_PASSWORD_SUCCEEDED
android.app.action.DEVICE_ADMIN_DISABLED
android.app.action.DEVICE_ADMIN_DISABLE_REQUESTED
android.app.action.DEVICE_ADMIN_ENABLED
android.bluetooth.a2dp.profile.action.CONNECTION_STATE_CHANGED
```

Activating application components

- Activities, Services and Broadcast Receivers components are activated by an asynchronous message called an ***intent*.**
- The intent defines the action/processing requested to the targeted component.





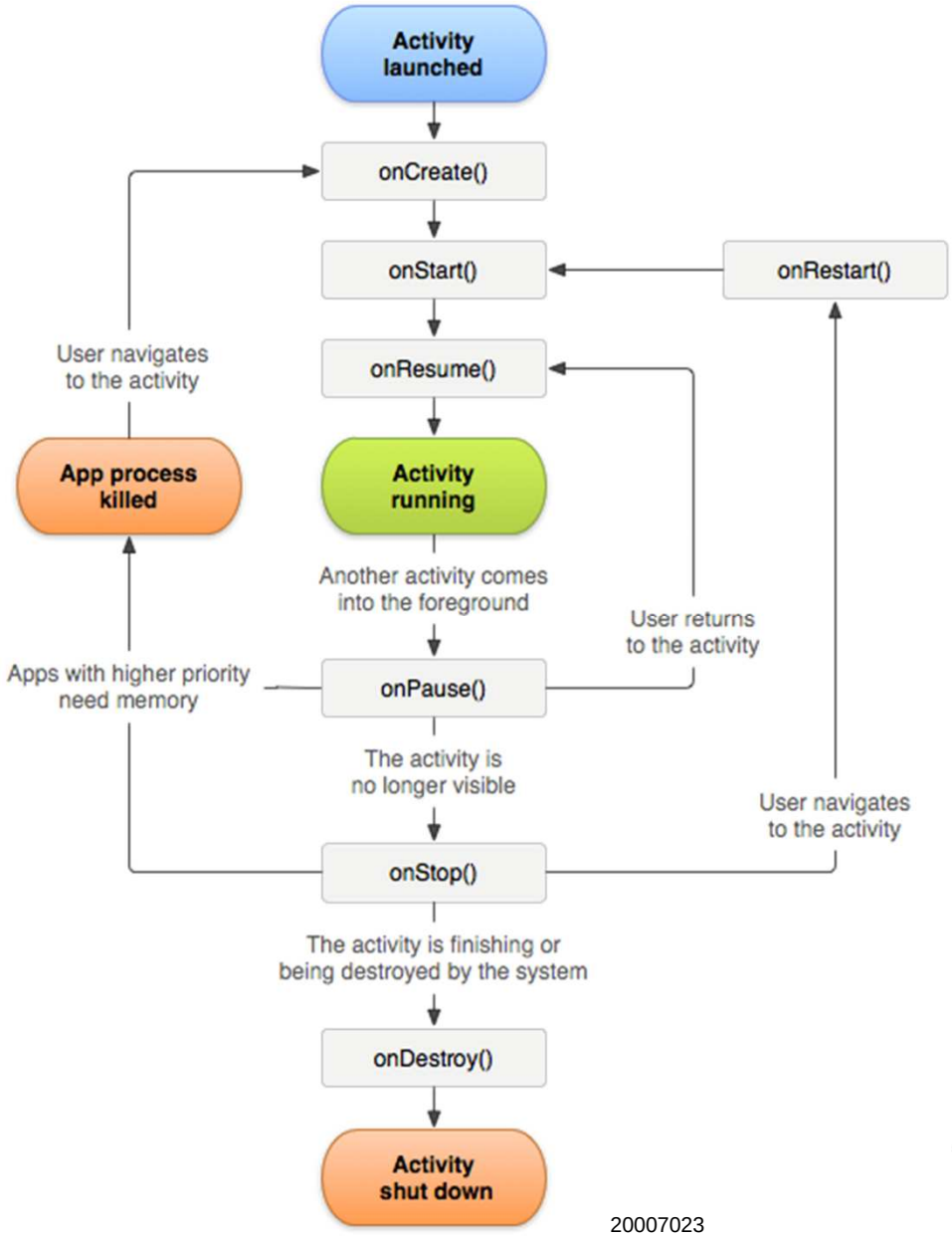
The Android Manifest File

- An application must define an *AndroidManifest.xml* file in its root directory. This file contains necessary information for Android OS to execute the application
- The AndroidManifest.xml file specifies:
  - The package name of the application
  - The components composing the application (activities, services, broadcast receivers, content providers)
  - The permissions
  - The minimum level of Android API required
  - The API libraries the application needs to be linked against (others libraries than Android API)
  - HW and SW features required by the application (e.g. Bluetooth, camera, etc...)

Application Resources

- An application is composed of code and resources.
- Resources are audio, video, image files and anything related to application visual aspect defined in XML files (layout, styles, menu, colors, string values...), separated from source code.
- Keeping resources separated from code provides the required flexibility to support multiple devices configurations (screen size, orientation, language) without modifying the code.
- http://developer.android.com/guide/topics/resources/index.html

The Activity Lifeycle



source: http://developer.android.com/guide/components/activities.html

onCreate()

- onCreate method is called when the activity is first created.
  - This is where we create the view
  - This method also provides you with a Bundle containing the activity's previously frozen state, if there was one for save and restore
- onCreate is Always followed by onStart() method

Private Constructor

The source code for compile() method in java.util.regex.Pattern is:

public static Pattern compile(String regex) {

return new Pattern(regex, 0);

}

```
/**
 * This private constructor is used to create all Patterns. The pattern
 * string and match flags are all that is needed to completely describe
 * a Pattern. An empty pattern string results in an object tree with
 * only a Start node and a LastNode node.
 */
private Pattern(String p, int f) {
    pattern = p;
    flags = f;

    // to use UNICODE_CASE if UNICODE_CHARACTER_CLASS present
    if ((flags & UNICODE_CHARACTER_CLASS) != 0)
        flags |= UNICODE_CASE;

    // Reset group index count
    capturingGroupCount = 1;
    localCount = 0;

    if (pattern.length() > 0) {
        compile();
    } else {
        root = new Start(lastAccept);
        matchRoot = lastAccept;
    }
}
```