Object Oriented Programming CS F213

J. Jennifer Ranjani
email: jennifer.ranjani@pilani.bits-pilani.ac.in
Chamber: 6121 B, NAB
Consultation: Appointment by e-mail

BITS Pilani
Pilani Campus

Java – an Introduction

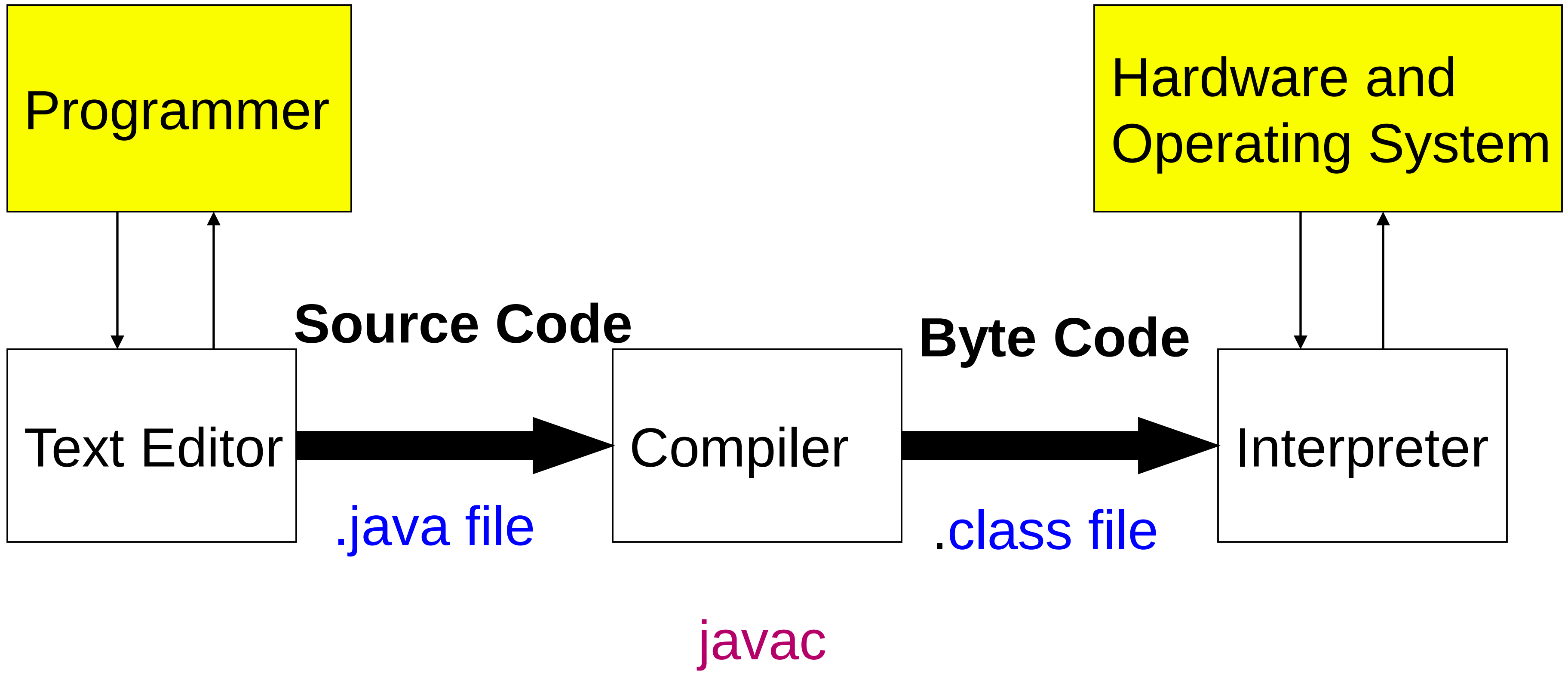
BITS Pilani
Pilani Campus

What is Java?

- Programming language and a platform

- **Platform:** Any hardware or software environment in which a program runs
  - Java has its own runtime environment (JRE) and API

Java is compiled and interpreted



Where is Java used?

Acc. To Sun, 3 billion devices run Java

- Desktop applications
  - Acrobat reader, media player, antivirus etc
- Web applications
- Enterprise applications
- Mobile
- Embedded System
- Smart card
- Robotics
- Games etc.

Java Platforms / Editions

- Java SE (Standard Edition)
  - Programming platform
- Java EE (Enterprise Edition)
  - Web and enterprise applications
- Java ME (Micro Edition)
  - Mobile applications
- JavaFx
  - Rich internet applications. Uses light weight user interface APIs.

History of Java

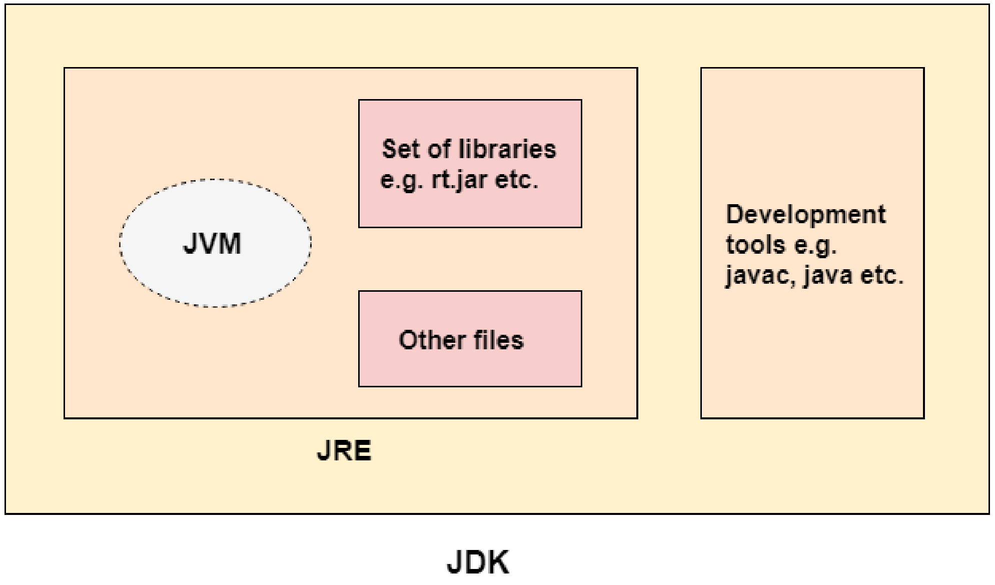
- James Gosling, Mike Sheridan, Patrick Naughton initiated the project in June 1991 (Green Team).
- Originally designed for small embedded systems
- “Greentalk” with file extension .gt
- Oak – symbol of strength and national tree of countries like U.S., France, Germany, Romania etc.
- Suggested names: Dynamic, Revolutionary, Silk, Jolt, DNA etc
  - Java is named after an island in Indonesia where first coffee was produced
  - Java is a name not an acronym
- Released in 1995.
- JDK 1.0 was released in Jan 23, 1996.

Java Version History

- JDK Alpha and Beta (1995)
- JDK 1.0 (23rd Jan, 1996)
- JDK 1.1 (19th Feb, 1997)
- J2SE 1.2 (8th Dec, 1998)
- J2SE 1.3 (8th May, 2000)
- J2SE 1.4 (6th Feb, 2002)
- J2SE 5.0 (30th Sep, 2004)
- Java SE 6 (11th Dec, 2006)
- Java SE 7 (28th July, 2011)
- Java SE 8 (18th March, 2014)
- Java SE 9 (21st Sep, 2017)
- Java SE 10 (20th March, 2018)

JDK

- JVM – provides runtime environment for bytecode execution; loads, verifies, executes code
- JRE – contains libraries and files used by JVM
- JDK – JVM, java, javac, jar, Javadoc etc. for complete java application development.

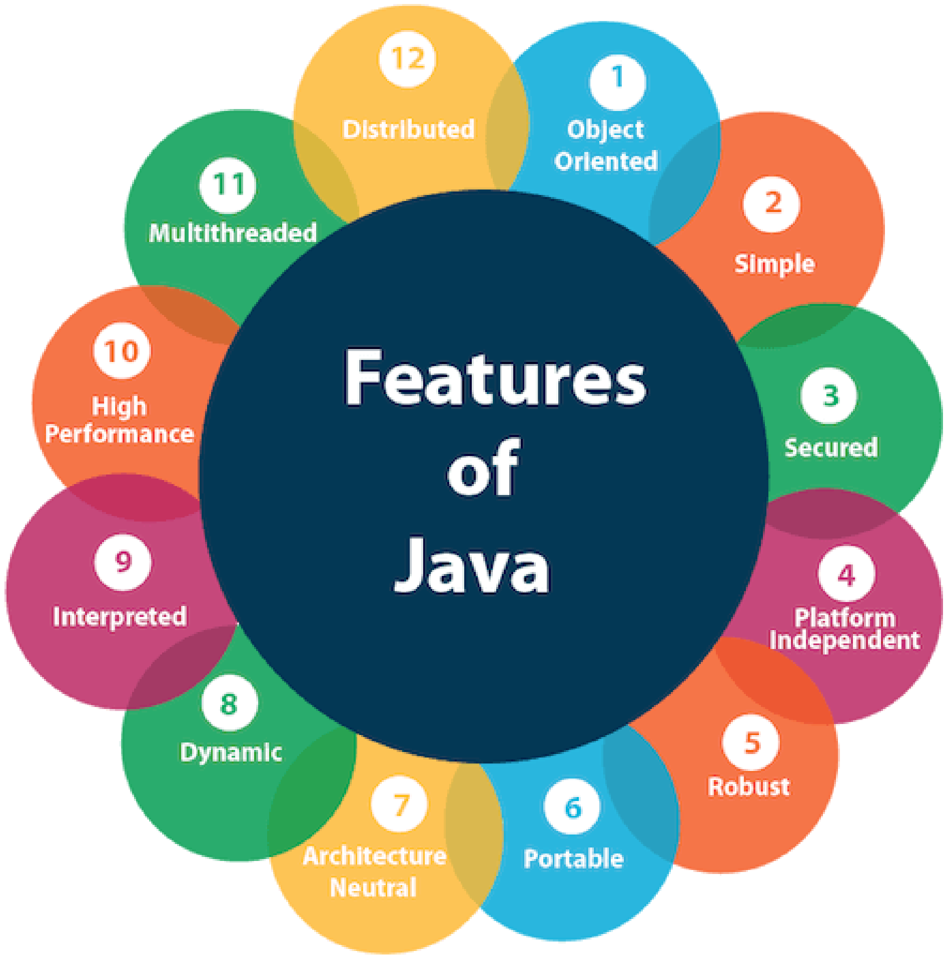

JVM
Set of libraries e.g. rt.jar etc.
Other files
Development tools e.g. javac, java etc.
JRE
JDK

innovate achieve lead

BITS Pilani
Pilani Campus

Features of Java

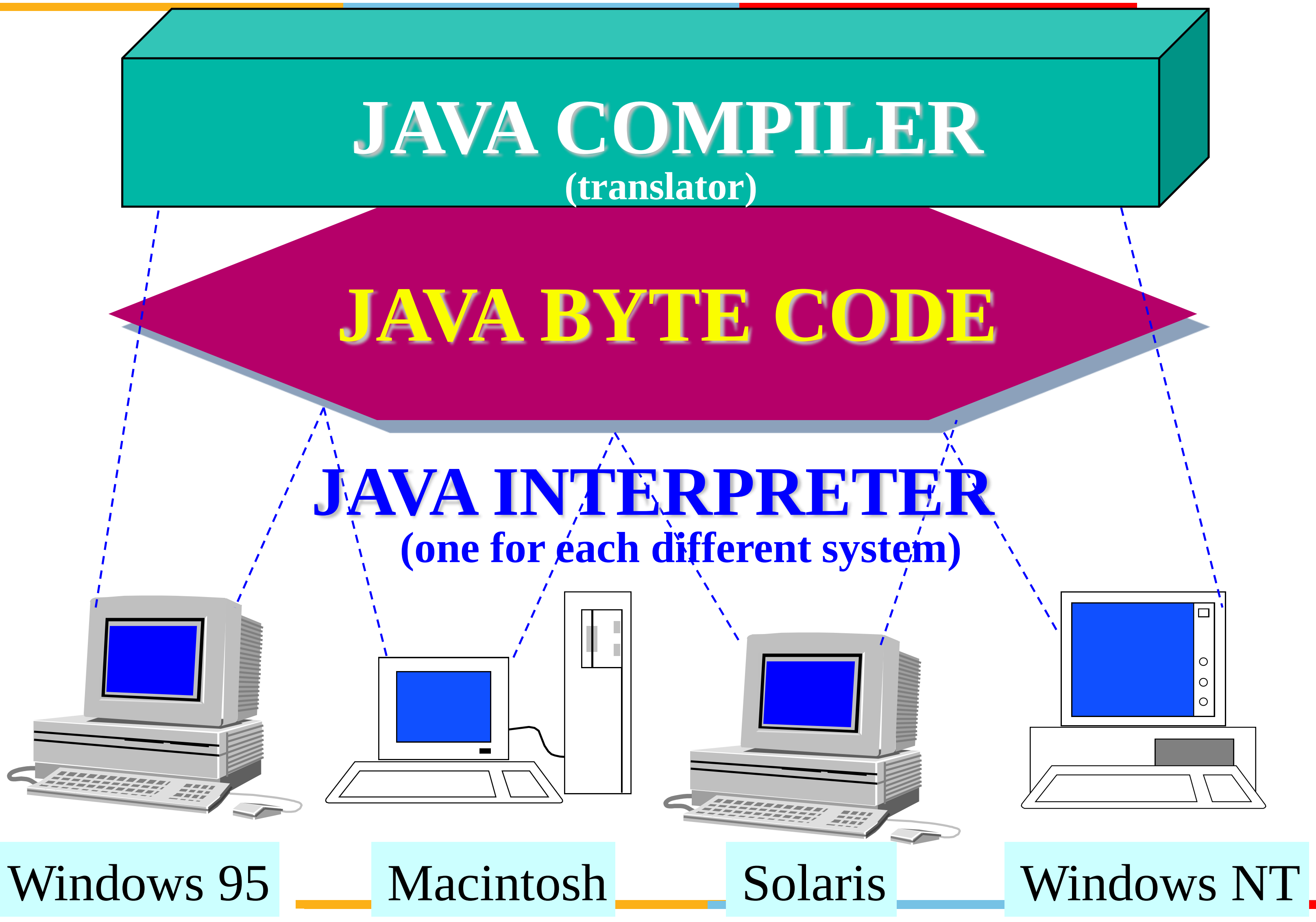
Java Buzzwords



Features of Java

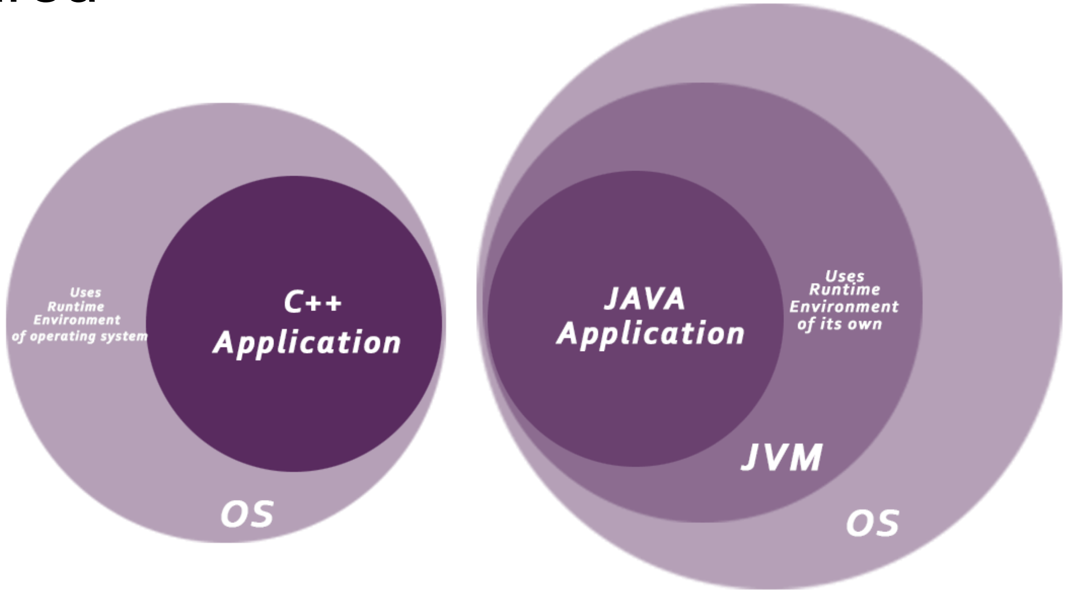
- Simple
  - Syntax based on C++
  - Removed confusing and rarely used features like pointers, operator overloading etc.,
  - Automatic garbage collection
- Object Oriented
  - Object, Class, Inheritance, Polymorphism, Abstraction, Encapsulation
- Platform Independent
  - Compiler converts Java code to bytecode
  - Bytecode is platform independent
  - Write Once and Run Anywhere

Platform Independence



Features of Java

- Secured


Uses Runtime Environment of operating system
C++ Application
OS
JAVA Application
Uses Runtime Environment of its own
JVM
OS

- Robust
  - Strong memory management, secure due to lack of pointers, automatic garbage collection, exception handling and type checking

Features of Java

- Architecture-neutral and Portable
  - Size of primitive types is fixed i.e., 4 bytes for both 32 and 64 bit architectures
  - Porting the java system to any new platform involves writing an interpreter.
  - The interpreter will figure out what the equivalent machine dependent code to run

Features of Java

- High Performance
  - Bytecode is close to native code
  - It is an interpreted language hence slower than C, C++
- Distributed
  - Enables access to files by calling methods from any machine on the internet
  - RMI, EJB
- Multi-threaded
  - Thread is like a separate program executing concurrently
  - Doesn’t occupy memory for each thread
  - Multimedia, Web applications etc
- Dynamic
  - Supports dynamic loading of classes i.e. classes are loaded on demand
  - Also supports functions from native languages i.e. C and C++

innovate achieve lead

BITS Pilani
Pilani Campus

Comparison with C++

Comparison Index	C++	Java
Platform-independent	Platform-dependent.	Platform-independent.
Mainly used for	System programming.	Application programming.
Goto	Yes	No
Multiple inheritance	C++ supports multiple inheritance.	Java doesn't support multiple inheritance through class. It can be achieved by interfaces in java.
Operator Overloading	Yes	No
Pointers	C++ supports pointers. You can write pointer program in C++.	Java supports pointer internally. But you can't write the pointer program in java.

Comparison Index	C++	Java
Compiler and Interpreter	C++ uses compiler only.	Java uses compiler and interpreter both. Java source code is converted into byte code at compilation time. The interpreter executes this byte code at run time and produces output.
Call by Value and Call by reference	C++ supports both call by value and call by reference.	Java supports call by value only.
Structure and Union	C++ supports structures and unions.	Java doesn't support structures and unions.

Comparison Index	C++	Java
Thread Support	C++ doesn't have built-in support for threads. It relies on third-party libraries for thread support.	Java has built-in thread support.
Virtual Keyword	C++ supports virtual keyword so that we can decide whether or not override a function.	Java has no virtual keyword. Non-static methods are virtual by default.
unsigned right shift >>>	C++ doesn't support >>> operator.	Supports unsigned right shift >>> operator that fills zero at the top for the negative numbers. For positive numbers, it works same like >> operator.

Simple program

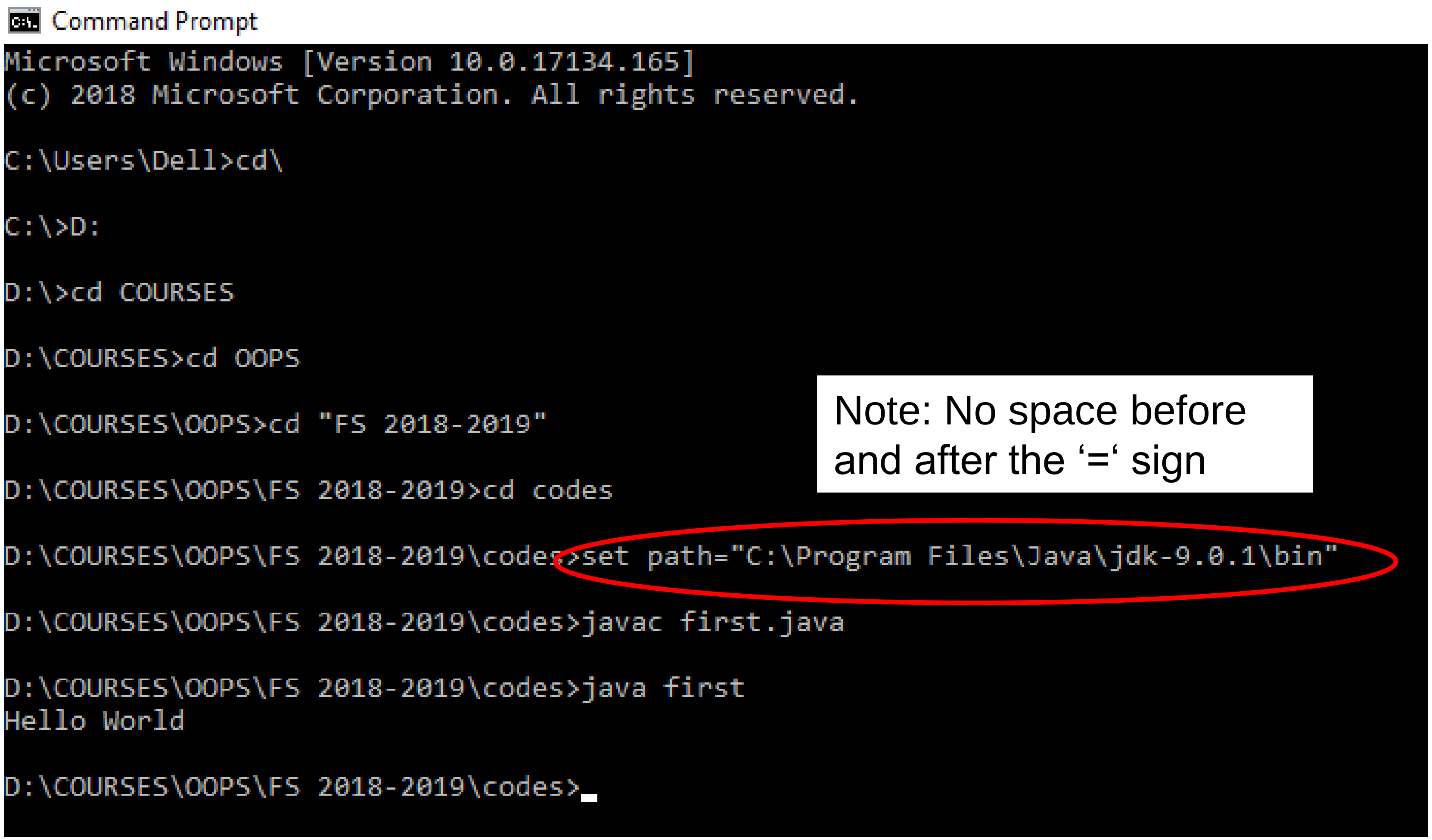
BITS Pilani
Pilani Campus

My first program

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

Save the file as first.java

Set path, Compile and Execute


Command Prompt

```
Microsoft Windows [Version 10.0.17134.165]
(c) 2018 Microsoft Corporation. All rights reserved.

C:\Users\Dell>cd\

C:\>D:

D:\>cd COURSES

D:\COURSES>cd OOPS

D:\COURSES\OOPS>cd "FS 2018-2019"

D:\COURSES\OOPS\FS 2018-2019>cd codes

D:\COURSES\OOPS\FS 2018-2019\codes>set path="C:\Program Files\Java\jdk-9.0.1\bin"

D:\COURSES\OOPS\FS 2018-2019\codes>javac first.java

D:\COURSES\OOPS\FS 2018-2019\codes>java first
Hello World

D:\COURSES\OOPS\FS 2018-2019\codes>_
```

Note: No space before and after the '=' sign

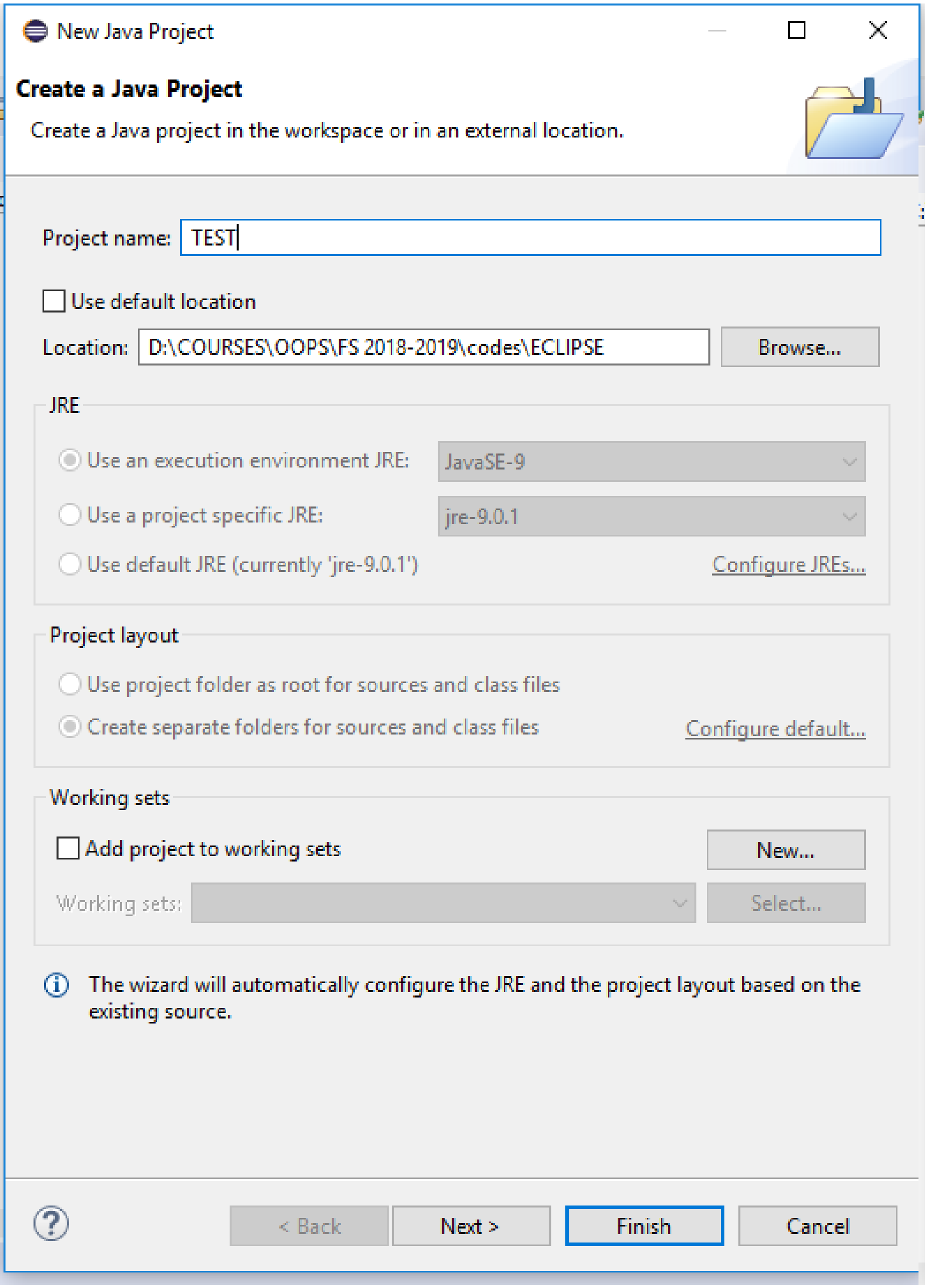
Setting PATH variable

- Right click My Computer → Properties → Advanced system setting → Environment Variables → System Variables → Path (Edit)
    - Add C:\Program Files\Java\jdk-9.0.1\bin

- It is sufficient to do this once.

BITS Pilani
Pilani Campus

Creating a New Project, New Class using Eclipse


New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: TEST

☐ Use default location

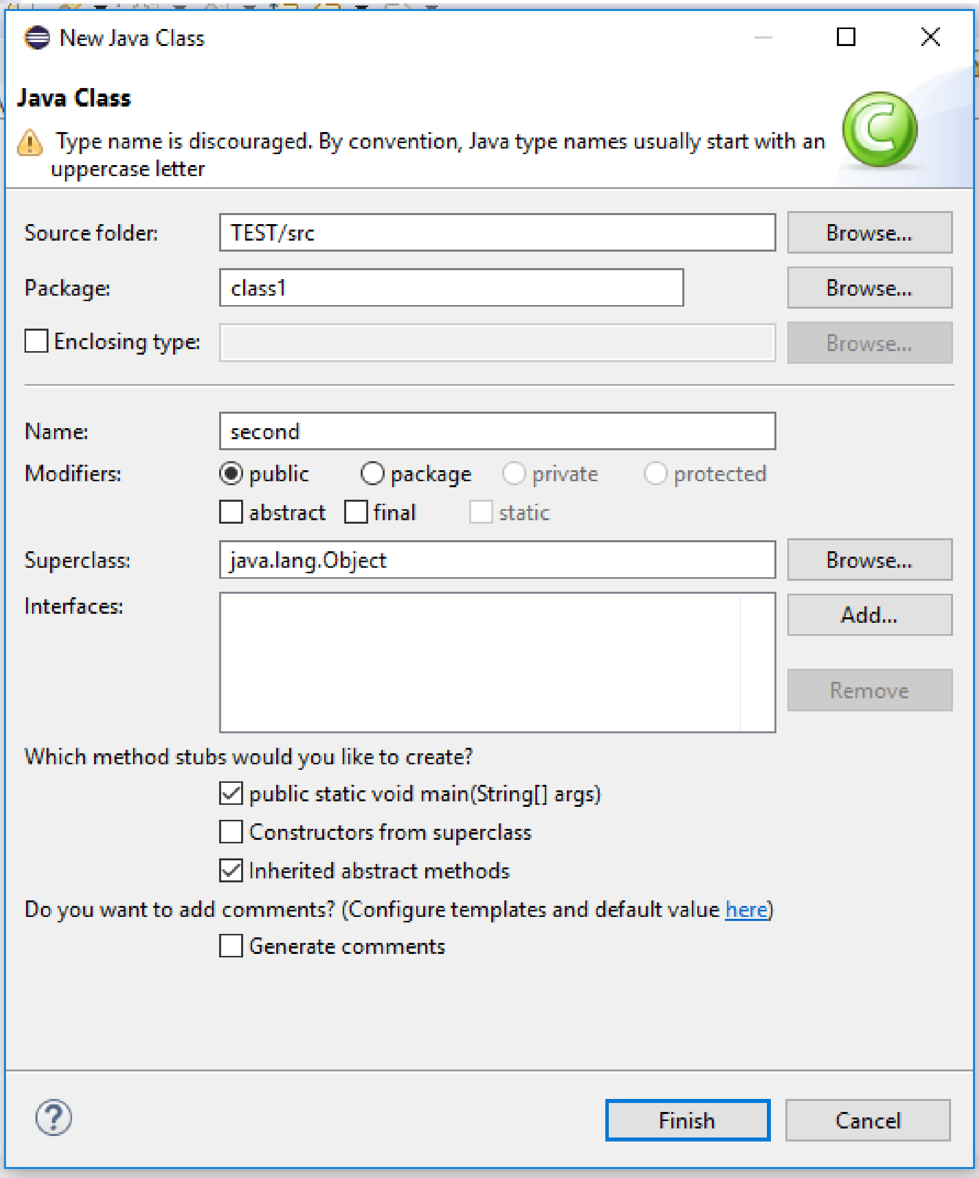
Location: D:\COURSES\OOPS\FS 2018-2019\codes\ECLIPSE

JRE
- ● Use an execution environment JRE: JavaSE-9
- ○ Use a project specific JRE: jre-9.0.1
- ○ Use default JRE (currently 'jre-9.0.1')

Project layout
- ○ Use project folder as root for sources and class files
- ● Create separate folders for sources and class files

Working sets
- ☐ Add project to working sets
- Working sets:

The wizard will automatically configure the JRE and the project layout based on the existing source.


New Java Class

Java Class

Type name is discouraged. By convention, Java type names usually start with an uppercase letter

Source folder: TEST/src

Package: class1

☐ Enclosing type:

Name: second

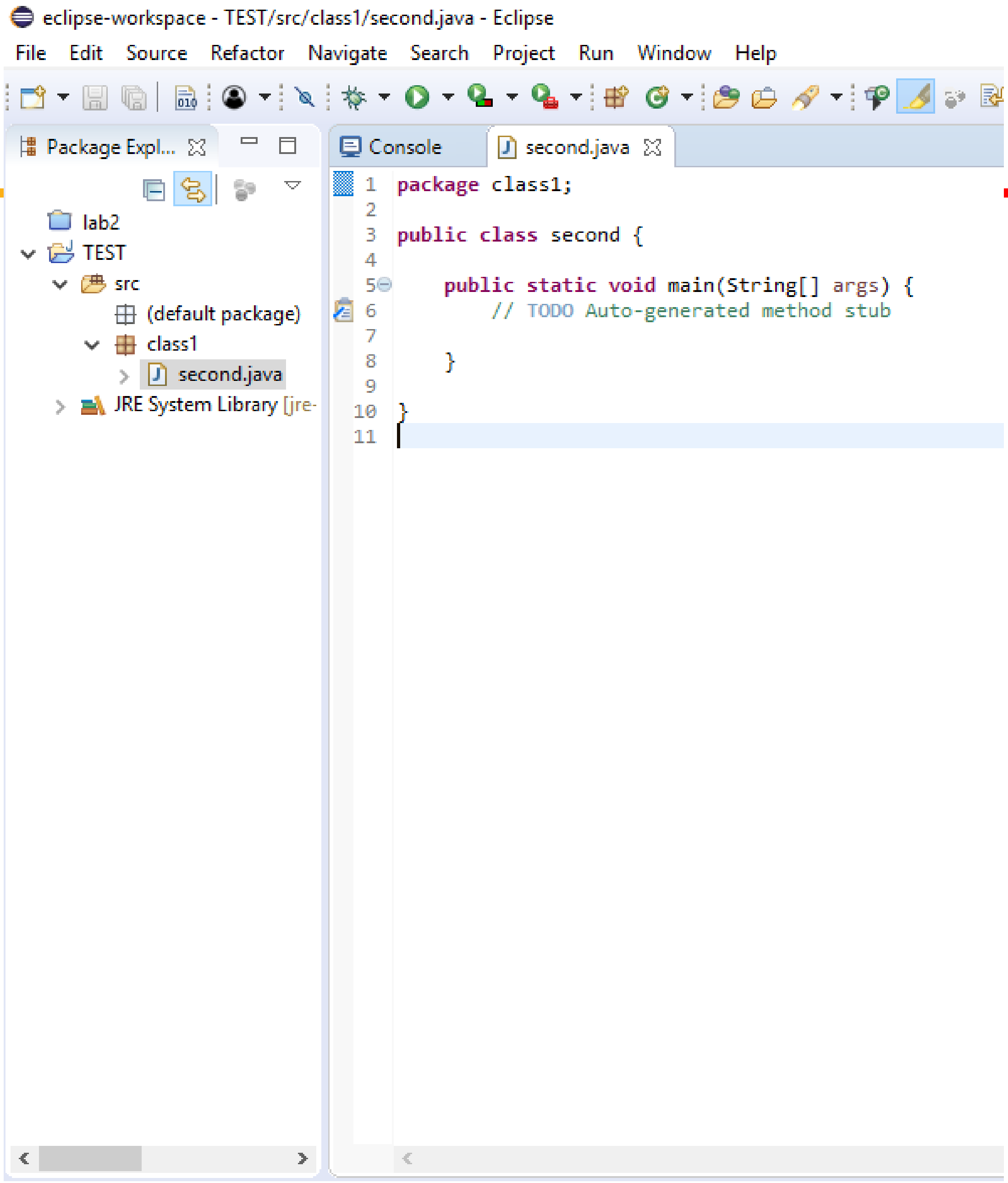
Modifiers: ● public ○ package ○ private ○ protected
☐ abstract ☐ final ☐ static

Superclass: java.lang.Object

Interfaces:

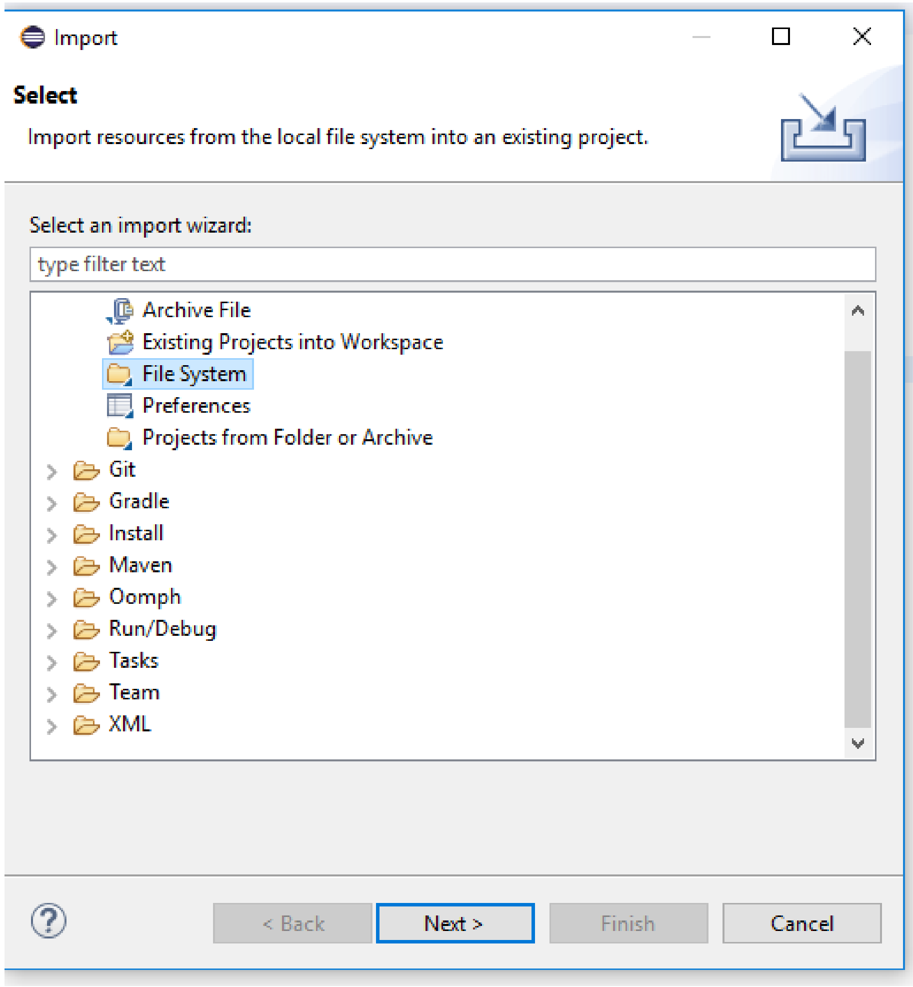
Which method stubs would you like to create?
- ☑ public static void main(String[] args)
- ☐ Constructors from superclass
- ☑ Inherited abstract methods

Do you want to add comments? (Configure templates and default value here)
- ☐ Generate comments


eclipse-workspace - TEST/src/class1/second.java - Eclipse
File Edit Source Refactor Navigate Search Project Run Window Help
Package Expl...
lab2
TEST
src
(default package)
class1
second.java
JRE System Library [jre-
Console
second.java

```
package class1;

public class second {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

    }

}
```

Importing a File



Import

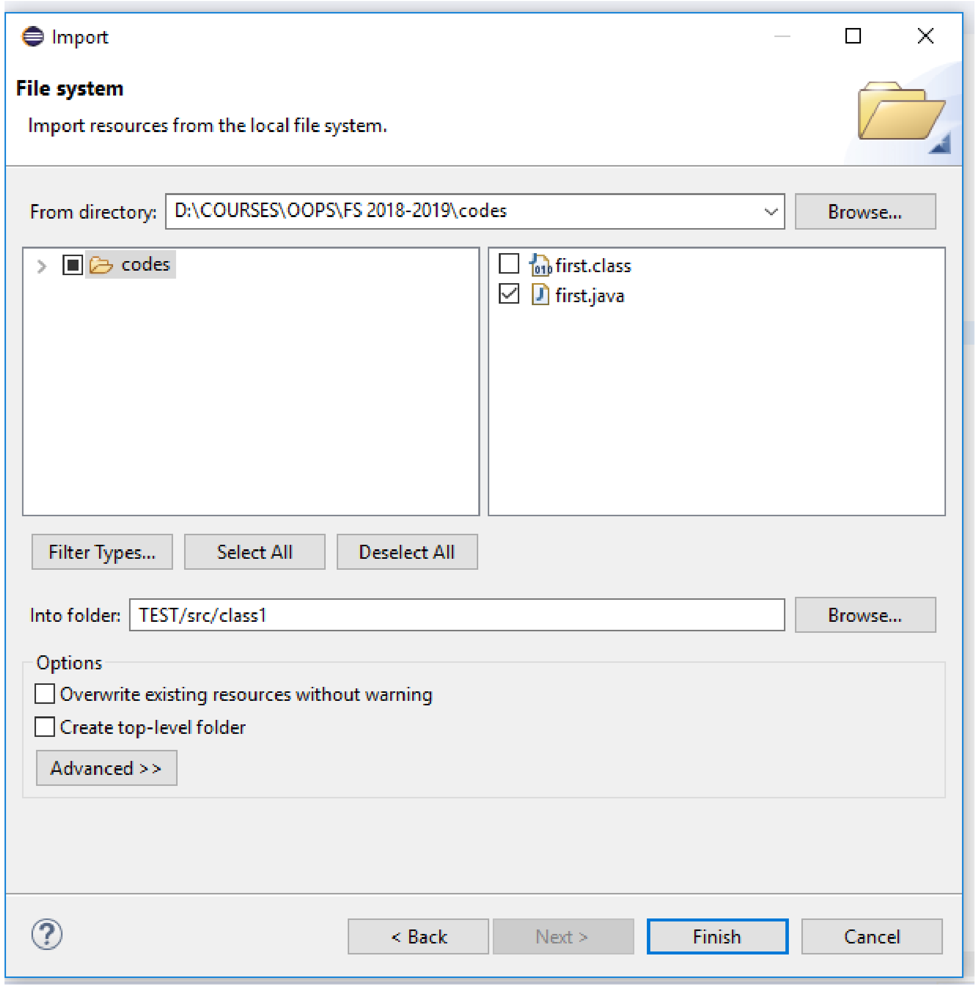
Select

Import resources from the local file system into an existing project.

Select an import wizard:

type filter text

- Archive File
- Existing Projects into Workspace
- File System
- Preferences
- Projects from Folder or Archive
- Git
- Gradle
- Install
- Maven
- Oomph
- Run/Debug
- Tasks
- Team
- XML



Import

File system

Import resources from the local file system.

From directory: D:\COURSES\OOPS\FS 2018-2019\codes

codes

- [ ] first.class
- [✓] first.java

Into folder: TEST/src/class1

Options

- [ ] Overwrite existing resources without warning
- [ ] Create top-level folder

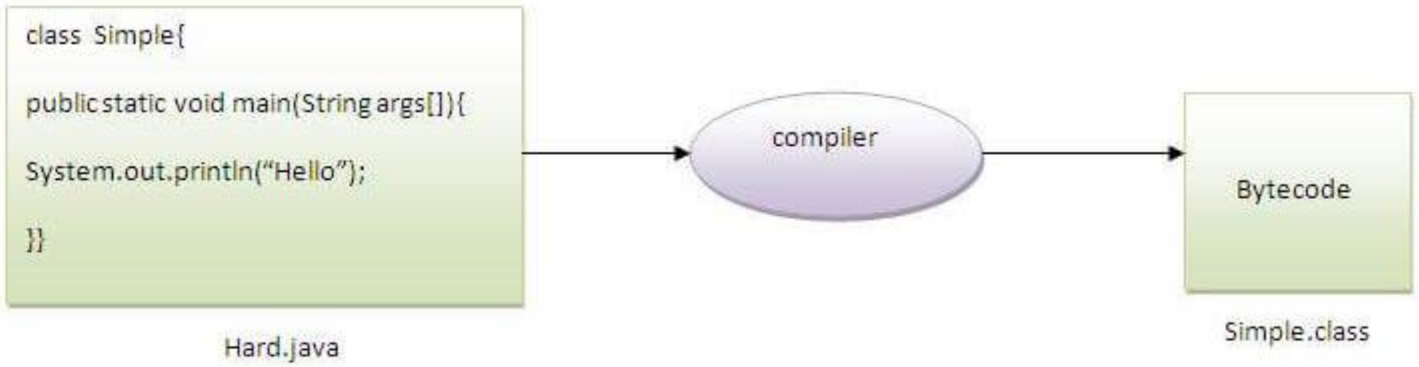
Parameters used

- **class** keyword is used to declare a class in java.
- **public** keyword is an access modifier which represents visibility, it means it is visible to all.
- **static** is a keyword. The core advantage of static method is that there is no need to create object to invoke the static method. The main method is executed by the JVM.
- **void** is the return type of the method, it means it doesn't return any value.
- **main** represents the starting point of the program.
- **String[] args** is used for command line argument.
- **System.out.println()** is used print statement.

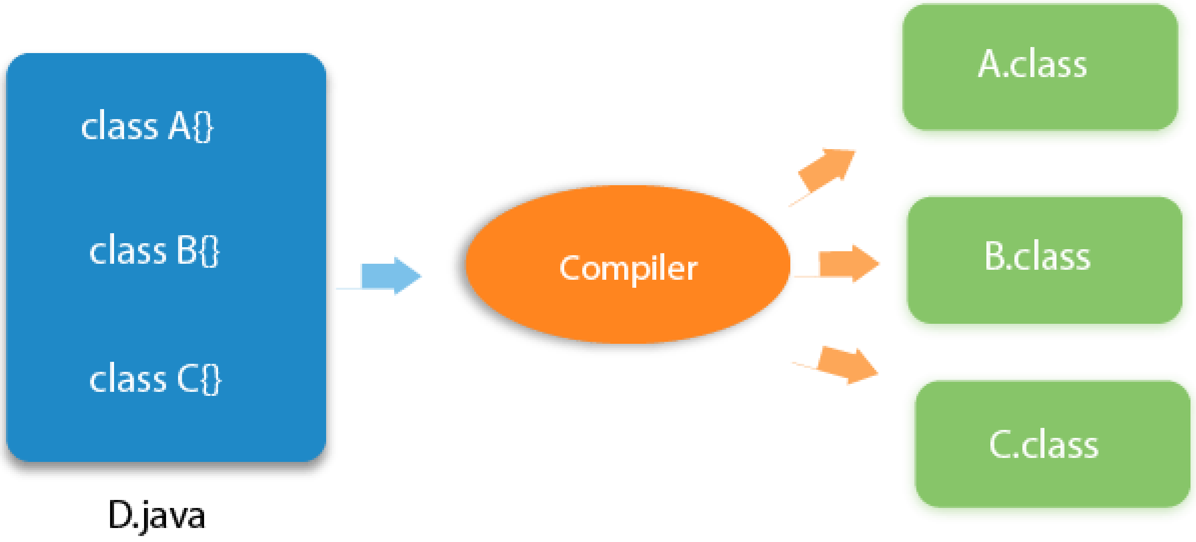
Valid and Invalid ‘Main’ Signatures

- VALID
  - **public static void** main(String[] args)
  - **public static void** main(String []args)
  - **public static void** main(String args[])
  - **public static void** main(String... args)
  - **static public void** main(String[] args)
  - **public static final void** main(String[] args)
  - **final public static void** main(String[] args)
  - **final strictfp public static void** main(String[] args)
- INVALID
  - **public void** main(String[] args)
  - **static void** main(String[] args)
  - **public void static** main(String[] args)
  - **abstract public static void** main(String[] args)

Additional details



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


class A{}
class B{}
class C{}
D.java
Compiler
A.class
B.class
C.class