Java: An introduction

Brief history of Java

- Creation of Java Language → By team named “Green” with members lead by James Arthur Gosling
- Originally called Oak (1991)
- First version of Java was released: 1995
- FOSS (GPL):
    - FOSS is a free and open source software and GNU General Public License
    - The GNU General Public License is a free, copy-left license for software and other kinds of works
- Netscape Navigator Internet browser was the first Java enabled browser

Versions

- JDK 1.0 (1995)
- JDK 1.1 (1997)
- J2SE 1.2 (1998)) →Playground
- J2SE 1.3 (2000) → Kestrel
- J2SE 1.4 (2002) → Merlin
- J2SE 5.0 (2004) → Tiger
- Java SE 6 (2006) → Mustang
- Java SE 7 (2011) →Dolphin

→ Called Java2

→ Version for the session

Tell me why

- Why do we have so many versions?
- Java started as a language originally targeting the digital cable television industry but it was ahead of its time for this industry. The Internet was beginning to boom at that time and Java turned out to be just right for it.
- Eventually more APIs were added and Java (just like any other language) progressively evolved in terms of providing more features for security, reflection mechanism, newer utility classes etc. A quick look at API page ..\docs\technotes\guides\lang\index.html, will give a idea what was added to the API in which version.
- Therefore with each version, the scope of the language has increased in terms of features .

Features: Simple

- Syntax similar to C/C++
  - Same types of loops
  - Same data types
- Error-free code
  - no pointers
  - no array index out of bounds problems
  - no explicit memory allocation & de-allocation

Java is able to overcome memory related problems by the virtue of Garbage collection which is a tool that attempts to free unreferenced memory (memory occupied by objects that are no longer in use by the program) also called garbage, in program

Feature: Object oriented programming

Object-oriented programming is a method of implementation in which programs are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships.

- Grady Booch

Programming approaches

- Structured Approach
  - Based on functions
  - goto branching
  - C, C++, COBOL, Pascal
  - Some disadvantages: No constructs for encapsulation, chances of code repetition, No strong data hiding concept, difficult to debug

- Object-oriented Approach
  - Smalltalk, Java, C#,

Object-Oriented Programming Concepts

- Object
- Class
- Abstraction
- Encapsulation
- Inheritance

Object

- A thing in a real world that can be either physical or conceptual. An object in object oriented programming can be physical or conceptual.
- Conceptual objects are entities that are not tangible in the way real-world physical objects are.
- Bulb is a physical object. While college is a conceptual object.
- Conceptual objects may not have a real world equivalent. For instance, a Stack object in a program.
- Object has state and behavior.

What is the state and behavior of this bulb?

Attributes and Operations

- The object’s state is determined by the value of its properties or attributes.
- Properties or attributes → member variables or data members
- The object’s behaviour is determined by the operations that it provides.
- Operations → member functions or methods

Putting it together

A bulb:

object

1. It’s a real-world thing.

methods

2. Can be switched on to generate light and switched off.
3. It has real features like the glass covering, filament and holder.
4. It also has conceptual features like power.

member variables

5. A bulb manufacturing factory produces many bulbs based on a basic description / pattern of what a bulb is.

class

Class

- A class is a construct created in object-oriented programming languages that enables creation of objects.
- Also sometimes called blueprint or template or prototype from which objects are created.
- It defines members (variables and methods).
- A class is an **abstraction.**

Abstraction

Abstraction denotes essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer.

-Grady Booch

- Abstraction is the process of taking only a set of essential characteristics from something.
- Example
  - For a Doctor→ you are a Patient
    - Name, Age, Old medical records
  - For a Teacher→ you are a Student
    - Name, Roll Number/RegNo, Education background
  - For HR Staff→ you are _______________
    - ____________,______________,____________

Java Class

```
public  class Student{
public int regno;                    → public data member
public String name;
public void display()                → public member function
{
//display statements
}
}
```

Creating Student Object

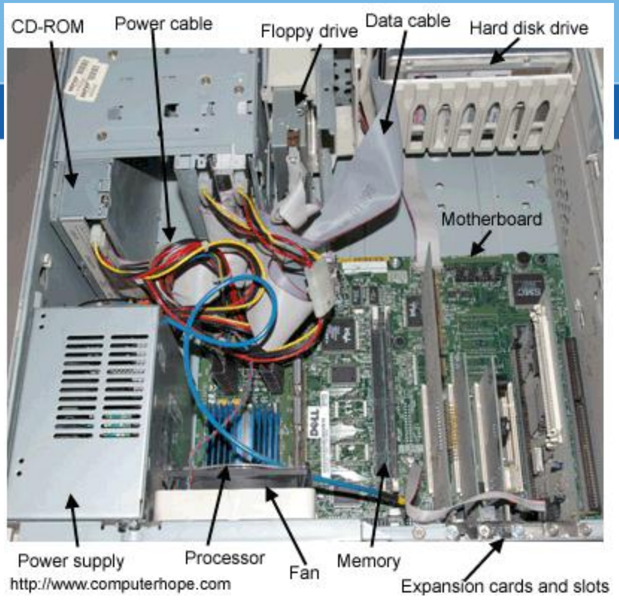
```
Student s= new Student();
s.display(); → access members using '.'
```

More on classes in the later sessions

Encapsulation

Would you like it if your CPU is given to you like this?

What are the problems if it were given to you like this?



Encapsulation is the process of compartmentalizing the elements of abstraction that constitute its structure and behavior; encapsulation serves to separate the contractual interface of an abstraction and its implementation.
- Grady Booch

- Encapsulation is binding data and operations that work on data together in a construct.
- Encapsulation involves Data and Implementation Hiding.

Data and Implementation hiding in Java class

`public` vs **`private`** access specifiers

```
public  class Student{
private
~~public~~ int regno;

private
~~public~~ String name;
```

visible

```
public void display(){
//display statements
}

}
```

Not visible to any other class other than Student class

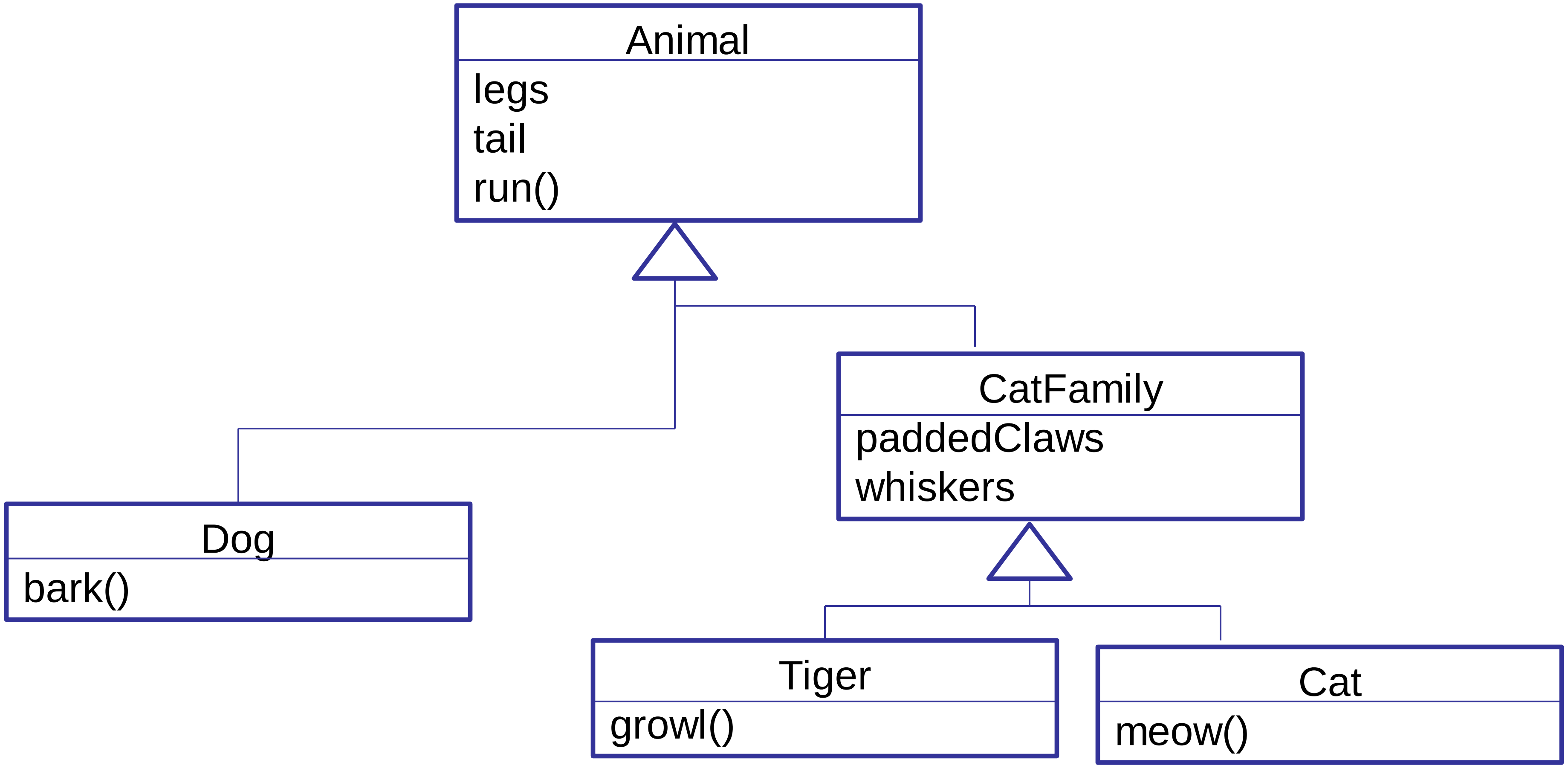
```
public  class Test{
public void print(){
Student s= new Student();
~~s.regno; s.name;~~
s.display();
}}
```

Change in implementation of display() does not affect Test class.

More on this in the later sessions

Inheritance and Polymorphism

Inheritance defines relationship among classes, wherein one class share structure or behavior defined in one or more classes.
- Grady Booch



More on this in the inheritance section

Portable and platform independent…

Before we understand portability and platform independence, we need to understand a few concepts.

- Java Code can be compiled anywhere
- Bytecode can be executed anywhere

$\Big\}$ Write-once-run-anywhere

Let's begin by writing a simple java program to understand this.

Simple Hello World in Java

Understanding
Portable and platform independent

Hello.java

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

Special statement used to display data on console. 'println' causes the next print statement to be printed in the next line.

main() is a method from where program execution begins.

Save the file as Hello.java. A public class must be saved in the same name as class name. More on this in package section.

Feature: Portable

- Java source code can be compiled in any java-aware system. Also when java code executes, it behavior is exactly same in any java-aware system.
- There are no platform-specific code in java programs that causes compilation problems in any other OS. (For example in C if you include conio.h library, this will work only in Window OS and will not work in linux).
- Also some features that were considered not portable in C/C++ (like sizes of int, right shift operator behavior etc.) were eliminated for Java.
- So, Java programs are portable, which implies that they behave the same way when executed in any system and produce the same result.

But what is Platform Independence then? Let us compile our code first which will take us there.

Environment to compile and execute

- Compile java programs
  - From command prompt
  - Through an IDE (Integrated development environment)
    - Eclipse →Apache
    - NetBeans →Oracle SDN
    - JBuilder → Borland
    - Integrated Development Environment → IBM

- *But what is an IDE?*

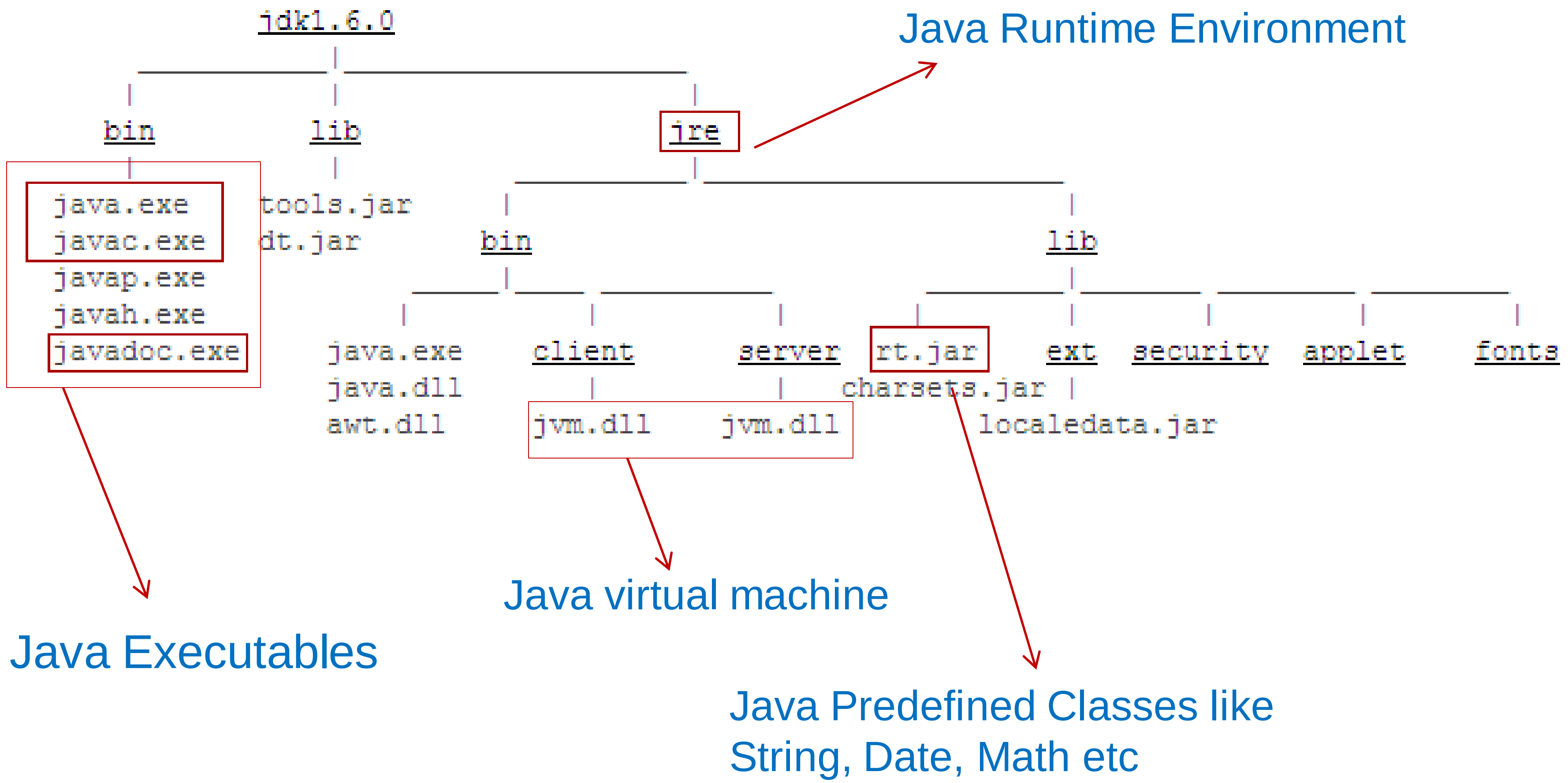
IDE

- Integrated development environment (sometimes also called integrated debugging environment or interactive development environment or integrated design environment) is an GUI interface that allows programmers to build and test (and sometimes design) their application.
- Typically IDE consists of
  - An editor where code can be written.
  - A compiler: Some IDEs (like all of them listed in previous slide) compile as and when the code is written. In eclipse, redline underline is used to indicate compilation errors and yellow underline is used to indicate warnings.
  - Run and Debug features
  - Tools to create language specific components
  - Context sensitive help (In eclipse, ctrl spacebar)
  - Tools for auto-build (packaging a JEE application in eclipse)

Setup JDK

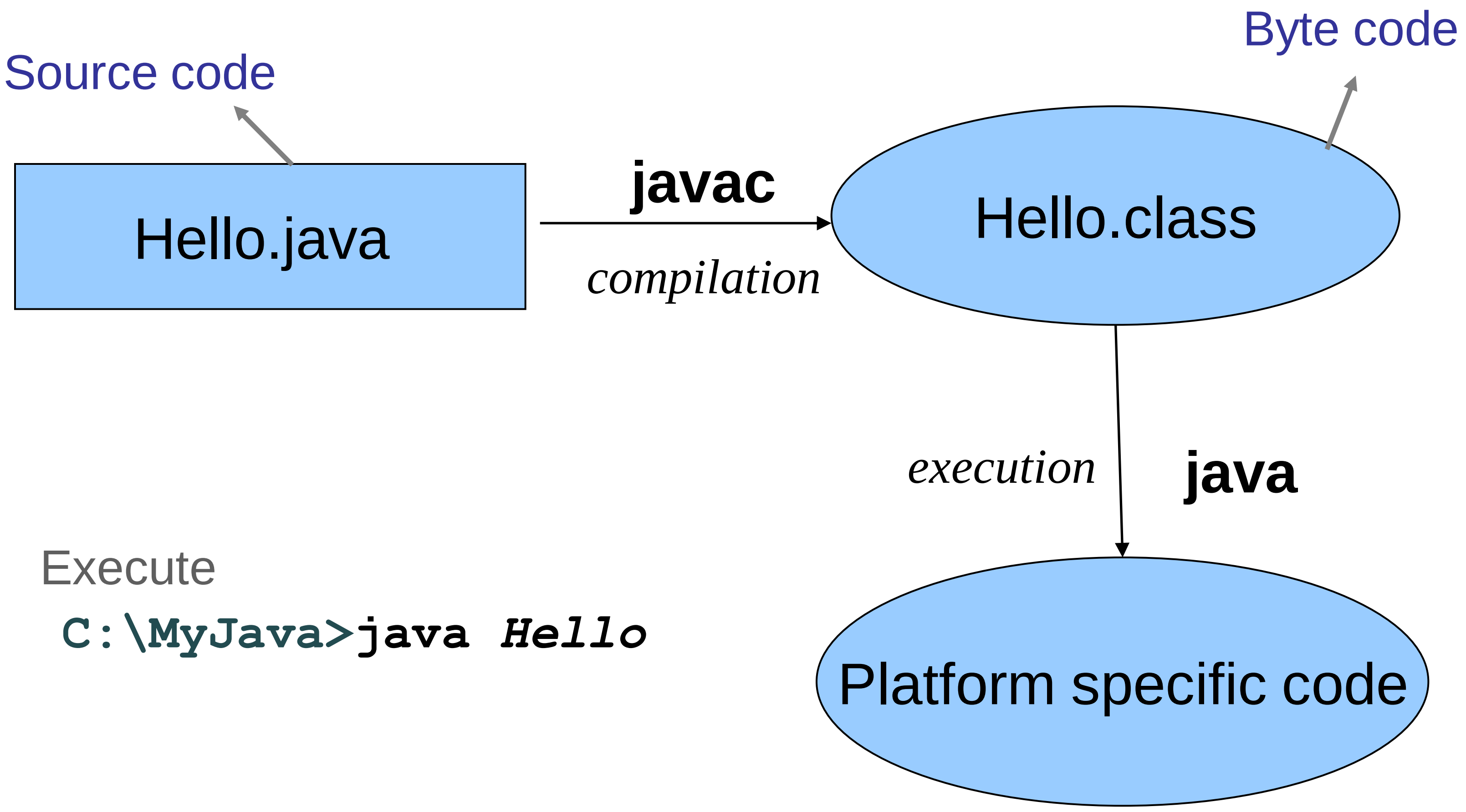
- http://java.com/en/download/index.jsp or find appropriate link in http://www.oracle.com/technetwork/indexes/downloads
- Download Java based on the type of OS
  - Windows
  - Linux
  - Mac OS
  - Solaris
- Install JDK

JDK installation directory



Compilation and execution model (from command prompt)

- Compile
  - `C:\MyJava>javac Hello.java`



- Execute

```
C:\MyJava>java Hello
```

Other useful javac options

- javac [options] [sourcefiles]
- sourcefiles can also include relative or absolute path components
- –d directory
  - Set the destination directory for class files.
  - **`D:\ javac –d bin Test.java`** places **`Test.class`** in **`D:\bin`** folder. Please note that the bin folder exist before giving this command
- -nowarn
  - Disable warning messages.
- -version
  - Displays the version used for compilation
- -target
  - Generate class files for specific VM version

Other useful java options

- java [options] class [argument ...]

  later

- -verbose:class
  - Display information about each class loaded.
- -verbose:gc
  - Report on each garbage collection event
- -version
  - Display version information and exit.
- -showversion
  - Display version information and continue.
- -?
- -help
  - Display usage information and exit.

About Eclipse 3.6 IDE

- IDE where Java programs can be written, compiled and executed.
- Open source
- JRE needs to be installed before eclipse is installed.
- 3.6 release of the Eclipse Project is developed on a mix of Java 1.4, Java 5 and Java 6 VMs
- Available for different OS(Windows, Mac OS, Linux, Solaris, IBM AIX, HP-UX)

Activity: Writing code in Eclipse IDE

- To start Eclipse double-click on the file "eclipse.exe“.
- Enter the workspace (path) where you want store your java files.
- Close the Welcome screen.
- There are many perspective that Eclipse provides. By default Java Perspective is opened.
- Select from the menu File -> New-> Java project. Enter the project name as Java1. Select "Create separate source and output folders".
- Right click on src folder on the left under Java1 and select New -> Class
- Enter the name of the class as “Hello”. Make sure “public static void main” is ticked. Click finish.
- Enter the print statement in the source code file that opens.

Compile and execute

- Compilation happens automatically. Note that red lines appear in case there are errors. Yellow lines are for warnings.
- Right click on your Java class and select Run-as-> Java application

Bytecode, JRE

- Java Bytecode is produced when Java programs are compiled.
- To execute Java program, JRE must be installed in the system
- JRE or Java Runtime environment contains
  - Java Virtual Machine
  - Standard class libraries (APIs)
  - Java Plug-in
  - Java Webstart
- JRE gets installed automatically when JDK is installed. JRE can be installed independent of JDK as well. This will mean that Java programs can be executed in that system.

Feature: Platform Independent

- A Java program requires JVM (part of JRE) to execute Java code. When java application starts to executes, that Java Virtual Machine also starts.
- Bytecode has instructions that Java Virtual Machine can understand and execute.
- JVM converts the Bytecode to machine specific code.
- Java Bytecode can be copied on to any machine that has JVM and executed. This is what makes Java Platform Independent.
- "Write Once, Run any where"

Is JVM platform independent?

No, since JVM needs to convert the byte code to machine specific code, it is different for each machine or OS, since each OS has its own native language.
That is the reason why JDK/JRE is available for different platforms.
Sun Microsystems claims that there are over 4.5 billion JVM-enabled devices!

JVM: perspectives

- JVM can be looked as
  - a runtime instance: JVM life cycle begins when applications starts to run (that is when main method is called) and ends when the application ends.
  - the abstract specification: Specification that Java team at Sun (Oracle) provides which tells JVM makers how they must design JVM for their OS.
  - a concrete implementation: JVM that is build specific for a specific OS based on abstract specification.
- *When we talk about JVM in these sessions we mean a runtime instance of JVM*

Feature: Robust

- Reliable →that the code works as desired in normal circumstances for specified periods of time.
- Robustness → the code responding appropriately in abnormal circumstances.

- Features that make Java robust
  - Strong Error handling mechanism: Lot of checks are done at the compilation itself. More checks are performed at runtime system to make sure that code does not malfunction.
  - Automatic garbage collection

Error handling in java

- Compilation error : generated by the compiler. Examples of situation when it arises:
  - incorrect syntax, bracket mismatch, if keywords are used as variables
  - Using uninitialized variables
  - Unreachable code: **`while(false){…}`**
  - Strong type checking
- Run-time error : generated by the JVM.
  - Examples of situation when it arises:
    - Attempt to divide an integral value by 0 or access array index beyond the defined range.
    - Trying to access a class that does not exist at runtime. (What happens if you delete Hello.class and then run Hello.
  - Java has strong exception handing mechanism that allows programmers to handle such errors and come out of the situation gracefully.

Automatic garbage collection

- The garbage collector is a tool that attempts to free unreferenced memory (memory occupied by objects that are no longer in use by the program) also called garbage, in program.
- *Automatic garbage collection* is an integral part of Java and its run-time system.
- Java technology has no pointers. So there is no question of allocation and freeing by programmers.
- Does that mean Java does not support dynamic memory allocation?
- No. It means that Java takes care of memory and relieves the programmers from memory-related hazels.
- java –verbose:gc can be used to get more information about garbage collection process.

More on this in classes session

Recall of features that we have seen so far

- Simple
- Object oriented language
- Portable
- Platform independent
- Robust

Feature: Multithreaded

- At the OS level, smallest unit of work that can be scheduled, is a thread.
- Thread is a sequence of execution of code.
- A process consists of one or more threads.
- Multiple threads in the same program share same resources.
- Unlike a multiple process, multiple threads in a process share same memory location. That is why threads are sometimes called Light weight process.
- Java Standard API has rich set of classes that allows us to work with multiple threads simultaneously

More in Threads section

Feature: Dynamic Linking

- The Java class loader(a represented by a class called **`ClassLoader`**) is a part of the Java Runtime Environment that dynamically loads Java classes into the Java Virtual Machine.
- It links classes with the executing code.
- Usually classes are only loaded on demand- only the classes specified in the code are loaded.
- JVM class loaders looks for the classes specified in the the following order-
  - Bootstrap classes → loads java libraries (rt.jar etc.)
  - Extensions classes → loads external libraries in lib/ext paths. For example, **`<JAVA_HOME>/lib/ext`** etc.
  - Classes in classpath→loads classes in CLASSPATH environment variable or system property **`java.class.path`**. (By default, the **`java.class.path`** property's value is '**`.`**')

Activity: Understanding Classpath

```
public class Student{
public String name;
public void display(){
System.out.print(“Welcome “);
System.out.println(name);}
}
```

C:\MyJava\Student.java

```
public class StudentTest{
public static void main(String args[]){
Student object= new Student();
object.name=“Gayathri”;
object.display();}
}
```

C:\MyJava\StudentTest.java

Set Classpath

- To successfully compile **`StudentTest`**, **`Student.class`** must be in the **`classpath`**

To set the classpath temporarily in command line

1. In the command prompt, enter the command

```
SET CLASSPATH=C:/MyJava;%CLASSPATH%;.
```

3. Go to **`D:\MyJava`** and compile **`StudentTest.java`**.
4. Execute **`StudentTest`** class (because that is where the main method is !)

Classpath

- Classpath is an environment variable that java compiler and JVM (system class loader) uses to locate the classes in the file system.
- Command to specify classpath
  - `javac –classpath directory1;directory2`
  - `javac –classpath C:/MyJava`
- Command to specify uncompiled source file to be compiled and included in the classpath
  - `javac – sourcepath directory1;directory2`
  - `javac –sourcepath C:/MyJava`

Feature: Security

- Security is in-built into
  - Java's language rules
    - Has no syntax for pointers. So it is impossible to access illegal memory
    - Has extensive syntax (like private, protected) to secure data from being accessed by illegal objects
    - Comprehensive API with support for a wide range of cryptographic services and for Authentication and Access Control
    - Secure communication
  - Java Compiler
    - Flags error on illegal conversions
    - Also makes sure that all java's language rules are adhered to

- Java Runtime System
  - The Java class loader and the bytecode verifier makes no assumptions about the primary source of the bytecode. (It could be that bytecode is produced by java compiler or some malicious compiler or may be hand written)
  - JRE does 3 levels of checks
    - class loader – loads the class and confines objects to the space demarcated for it
    - bytecode verifier verifies classes after loading to make sure no illegal operation is performed like –illegal access to private variables, type-safe casting, array bound checking etc.
    - security manager - determines what resources a class downloaded from a network can access such as reading and writing to the local files, creating a new process. Sandbox security model for applets, policy files needed for any java network application to access disk.

Feature: Performance

- JIT (Just-in-Time Compilation)
- GC (Garbage Collector) runs as low priority thread
- JNI (Java Native Interface)

More on this coming up later sections

Interpreter vs JIT

- Java Bytecodes were originally designed to be interpreted by JVM meaning bytecode are translated to machine code without it being stored anywhere.
- Since bytecode verifier (which is part of JVM) performs runtime checks, line by line execution was important.
- Since speed became an issue, Just-in-Time Compilation (JIT) came into being. JIT converts chunks of code, stores it temporarily in memory and then executes the converted code.
- JIT compilers are typically bundled with or are a part of a virtual machine and do the conversion to native code at runtime, on demand.
- The compiler also does automatic register allocation and some optimization when it produces the bytecodes.
- Therefore, JIT is hybrid compiler.

JNI

- Java Native Interface is way in which Java code can call native methods or functions like C++ or C.
- This feature of Java allows to
  - unleash platform-specific features or program library
  - make use of code is already written native language then Java code can make use of existing code
  - improve performance

Summary of the features of Java Language

- Simple
- Object oriented language
- Portable and platform independent
- Robust
- Multithreaded
- Dynamic Linking
- Secure
- Performance

Flavors Of Java

- JSE
  - Java Standard Edition formerly known as J2SE.
  - This forms the core part of Java language.
- JEE
  - Java Enterprise Edition formerly known as J2EE.
  - These are the set of packages that are used to develop distributed enterprise-scale applications.
  - These applications are deployed on JEE application servers.

Our focus

- JME
  - Java Micro Edition formerly known as J2ME.
  - These are the set of packages is used to develop application for mobile devices and embedded systems.

Commenting source code

Why should you comment your code?

How much should you comment?

- 3 types of comment
  - Single line comment : //
  - Multi-line comment: /* */
  - Documentation Comment (Doc comment): /** */
- For standard Java class library, Sun supplies HTML documentation that you can download from the oracle's site.

Example for single line and multiline comments

```
public class College{
public String name;
//public int id;
public void display(String user){
/*System.out.println("User:");
System.out.println(user);*/
System.out.println("Name:");
System.out.print(name);
return "welcome";
}
}
```

Some Guidelines for Doc comments

- Who owns and edits the Doc Comments: usually the programmer
- Doc comments for classes, methods, fields:
  - Brief description of what it does. In case longer description is required, link to an external document (word, pdf) can be included.
  - For methods, whether they are thread-safe and what runtime exception they throw must be specified. $\longrightarrow$ *Revisit this after threads and exception*
- Proper indentation of documentation for better readability.
- Entities for the less-than (<) and greater-than (>) symbols should be written &lt; and &gt;. Likewise, the ampersand (&) should be written &amp;.

Visit this after finishing Java $\swarrow$

For More Information
http://download.oracle.com/javase/1.4.2/docs/tooldocs/windows/javadoc.html#documentationcomments

Example

```
/** This class is used to represent a college.
 * @author Murali
 * @version 1.0, 08/16/2010
*/

public class College{
/** Name of the college */
public String name;

/**  This method is used to display and return the
welcome message.
 * @param  user is the name of the user
 * @return returns the welcome message
 */
public String display(String user){
System.out.println(user);
System.out.print(name);
return "welcome";
}
```

a special keyword that the Javadoc tool can process called Annotation

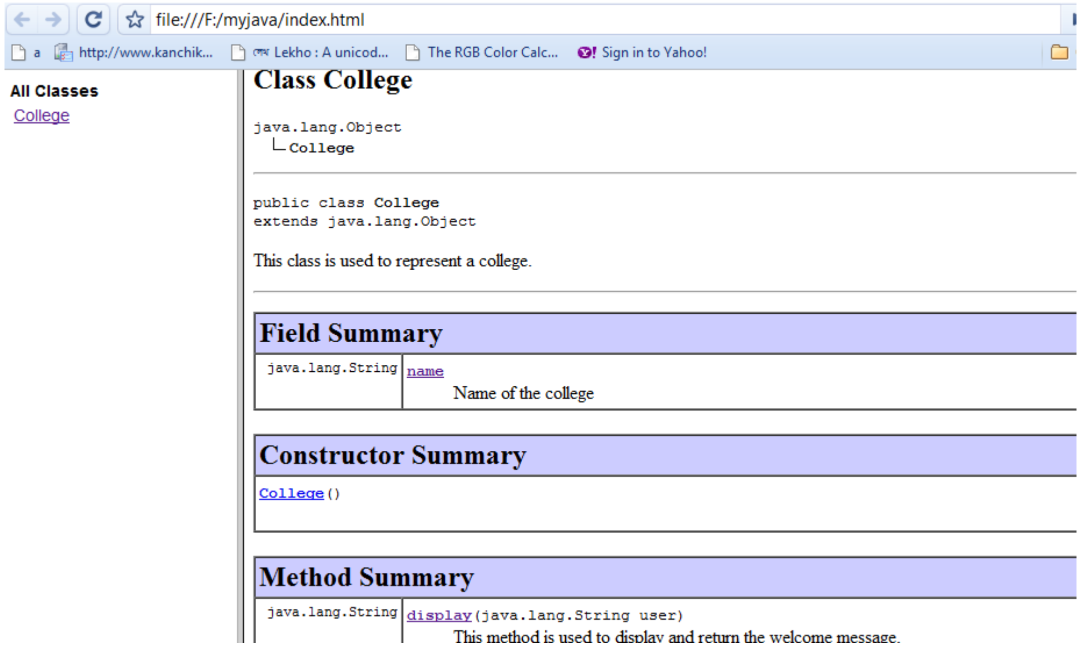
```
/** This is where the program execution begins.*/
public static void main(String args[]){
College collegeObject= new College();
collegeObject.name="XYZ College";
collegeObject.display("Deepak");
}
}
```

javadoc

- Tool that is used to produce HTML pages by parsing through the documentation comment in java source code.
- Produces documentation for public and protected classes/members.
- Works on entire package or a single source file
- Usage on source file
  - **`javadoc  sourcefilename.java`**

Use the following command to generate java documentation.

```
javadoc College.java
```



All Classes
College

Class College

```
java.lang.Object
  └College
```

```
public class College
extends java.lang.Object
```

This class is used to represent a college.

Field Summary

java.lang.String	name Name of the college

Constructor Summary

College()

Method Summary

java.lang.String	display(java.lang.String user) This method is used to display and return the welcome message.

Check out the files that are generated. Did you notice where the commented lines are inserted ?

Nesting comments

- Valid Nesting

```
/* // */            /** // */
///* */              // /** */
```

- Invalid Nesting

  `/* */` and `/** */` nested with itself and nested with each other gives error.

```
/* /* */ */
/** /* */ */
```

Annotations

- *Annotations* are extra text starting with @ symbol that are added in the Java program to provide information about a program.
- Annotations do not directly affect program semantics.
- They provide information about the program to tools and libraries, which can in turn affect the semantics of the running program.
- Annotations can be read from source files, class files, or reflectively at run time.

Uses

- Annotations for the compiler
  - **`Example:`**
    - **`@SuppressWarnings`** : can be used by the compiler to detect errors or suppress warnings
- Annotations for the application servers and tools
  - Application server tools can process annotation information to generate code etc for services it provides like security etc.
  - **`javadoc`** tool uses annotations like **`@author`** etc.
- Runtime processing — Some annotations are available to be examined at runtime.