Introduction to Java

Shakir Hussain

Contents

- What is Java?
- Brief history of Java
- The Java Programming Language
  - Buzzwords
- The Java Platform
- Development Environment Setup
- First Java Program
- Terminology

What is Java?

- High level programming language
- Originally developed by Sun Microsystems (now, Oracle), Which was initiated by James Gosling
- Designed with a concept of write once and run anywhere
- First version of java released in 1995
- Initial name was Greentalk, later renamed to Oak and finally Java
- Java is a Platform

Brief history of Java

James Gosling

https://dzone.com/articles/a-short-history-of-java

https://www.javatpoint.com/history-of-java

The Java Programming Language

Java Language - Buzzwords

- Platform Independent (architecture neutral)
  - *Write once run anywhere*
- Simple
  - *Small language, large libraries*
- Object Oriented
  - *Supports Abstraction, Encapsulation, Polymorphism, Inheritance etc.*

Java Language - Buzzwords..

- Auto Garbage Collection
  - *Memory management handled by Java Virtual Machine*
- Secure
  - *No memory pointers, program run inside virtual machine*
  - *Java Bytecode verification*
  - *Array Index limit checking*

Java Language – Buzzwords…

- Portable
  - *Primitive data type size and their arithmetic behavior are specified by the language.*
  - *Libraries define portable interfaces*
- Distributed
  - *Libraries for network programming*
  - *Remote method invocation*

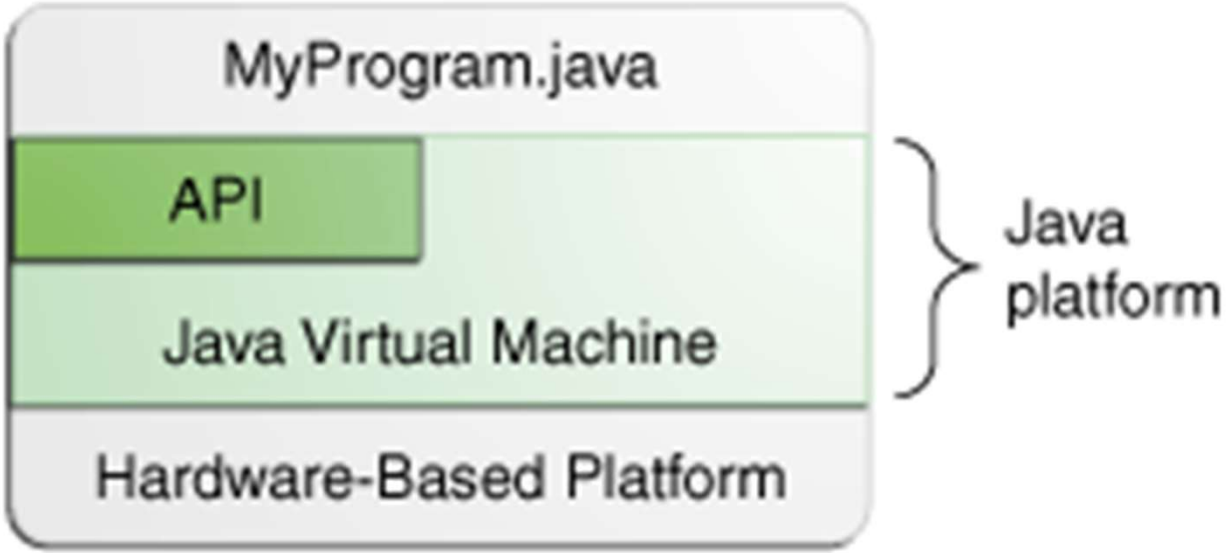
Java Language – Buzzwords….

- Multithreaded
  - *Easy to create and use*
- Robust
  - *Strong memory mgmt.*
- Dynamic
  - *Finding runtime type information is easy.*
  - *The linking of data and methods to where they are located, is done at run-time.*
  - *New classes can be loaded while a program is running. Linking is done on the fly.*

The Java Platform

- Software-only platform that runs on top of other hardware-based platforms.

- Environment in which java program runs.



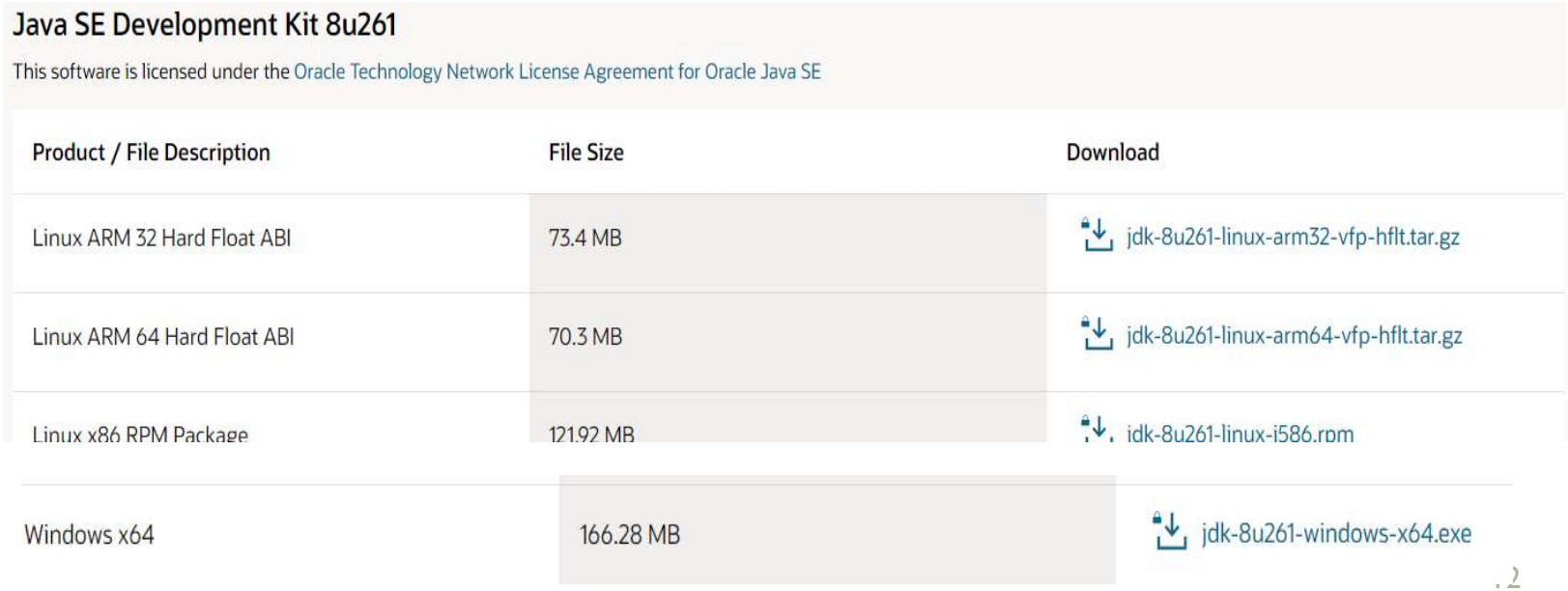
Setup Dev Environment

- What do you need to write and run java program?
  - Java Development Kit (JDK)
  - ASCII Text Editor

Download JDK

- https://www.oracle.com/in/java/technologies/javase/javase-jdk8-downloads.html

- Go to the below section, download platform specific installer



Java SE Development Kit 8u261

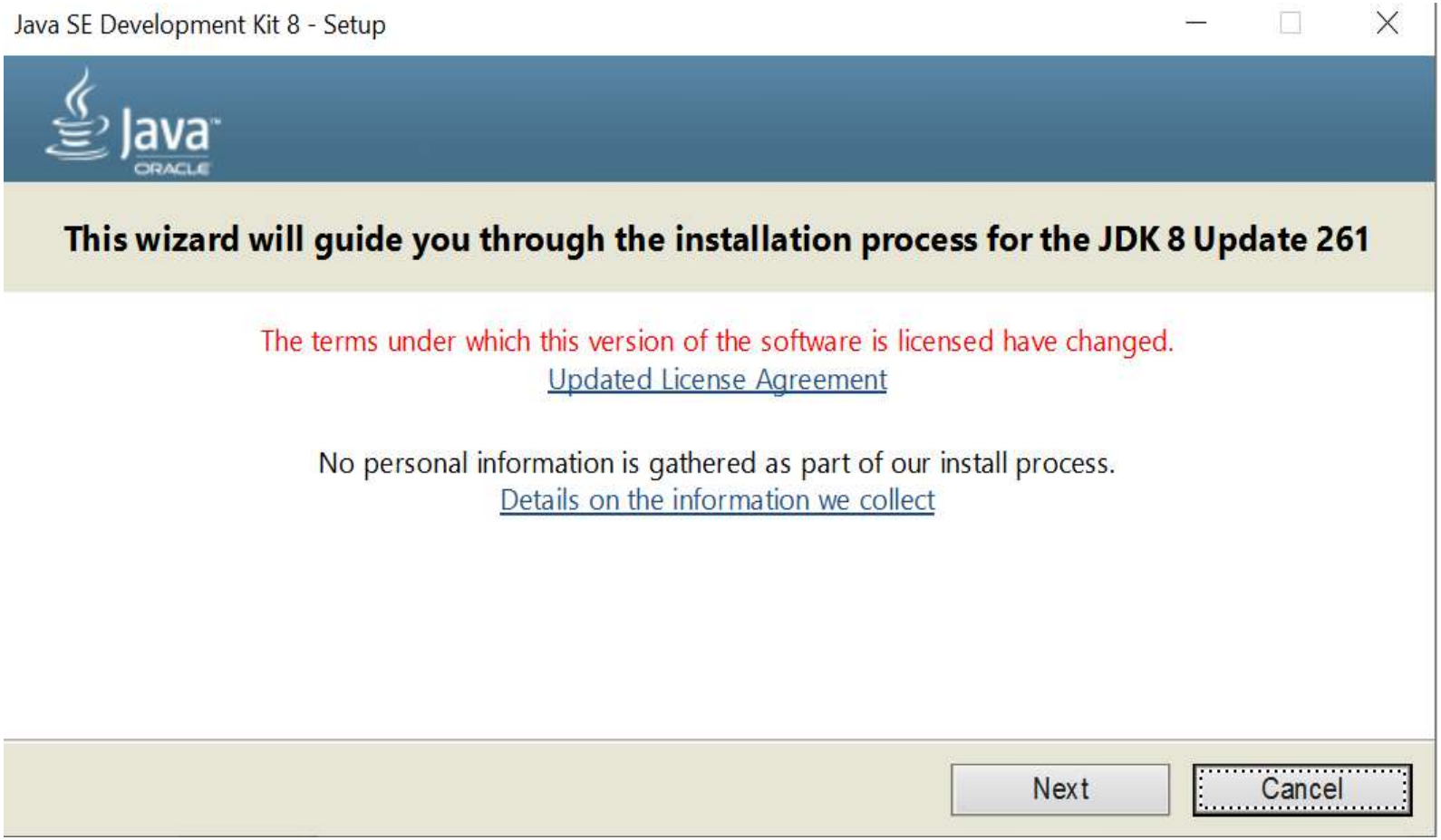
This software is licensed under the Oracle Technology Network License Agreement for Oracle Java SE

Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	73.4 MB	jdk-8u261-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	70.3 MB	jdk-8u261-linux-arm64-vfp-hflt.tar.gz
Linux x86 RPM Package	121.92 MB	jdk-8u261-linux-i586.rpm
Windows x64	166.28 MB	jdk-8u261-windows-x64.exe

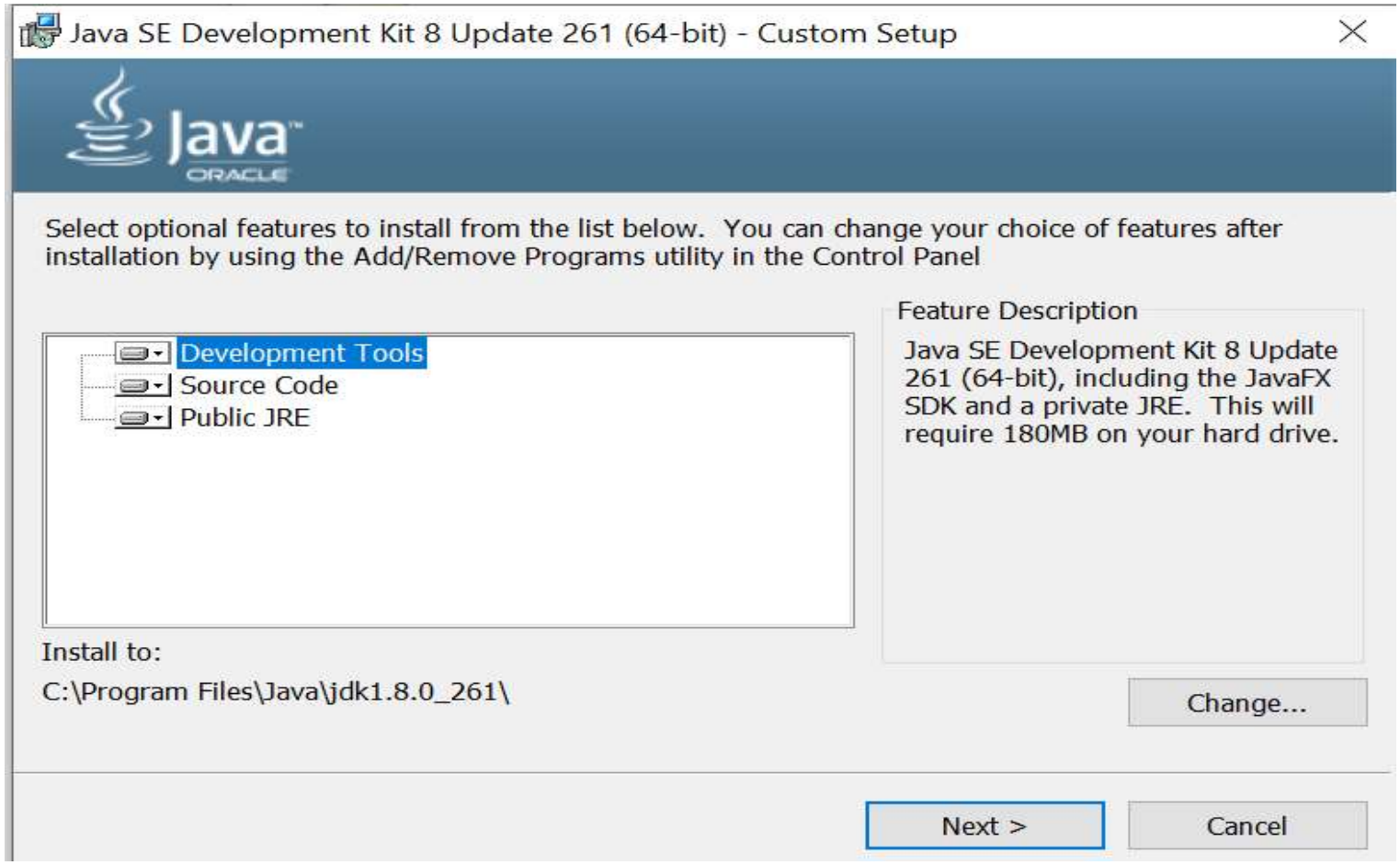
Install Steps

- Once downloaded (e.g. Jdk-8u261-windows-x64.exe) , click to install.
- Steps for windows JDK will be shown in next slides

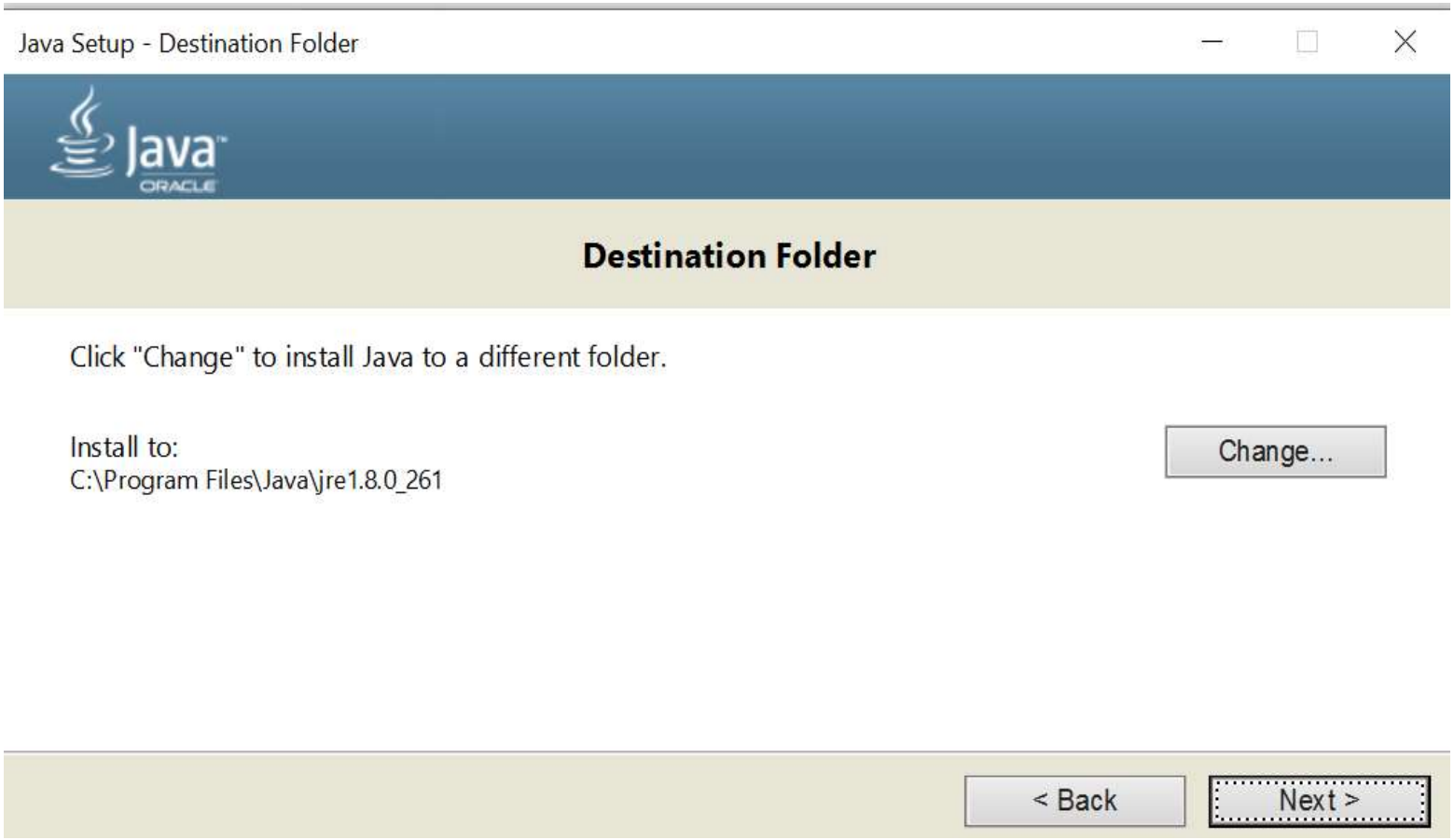
Install Steps – 1st Window



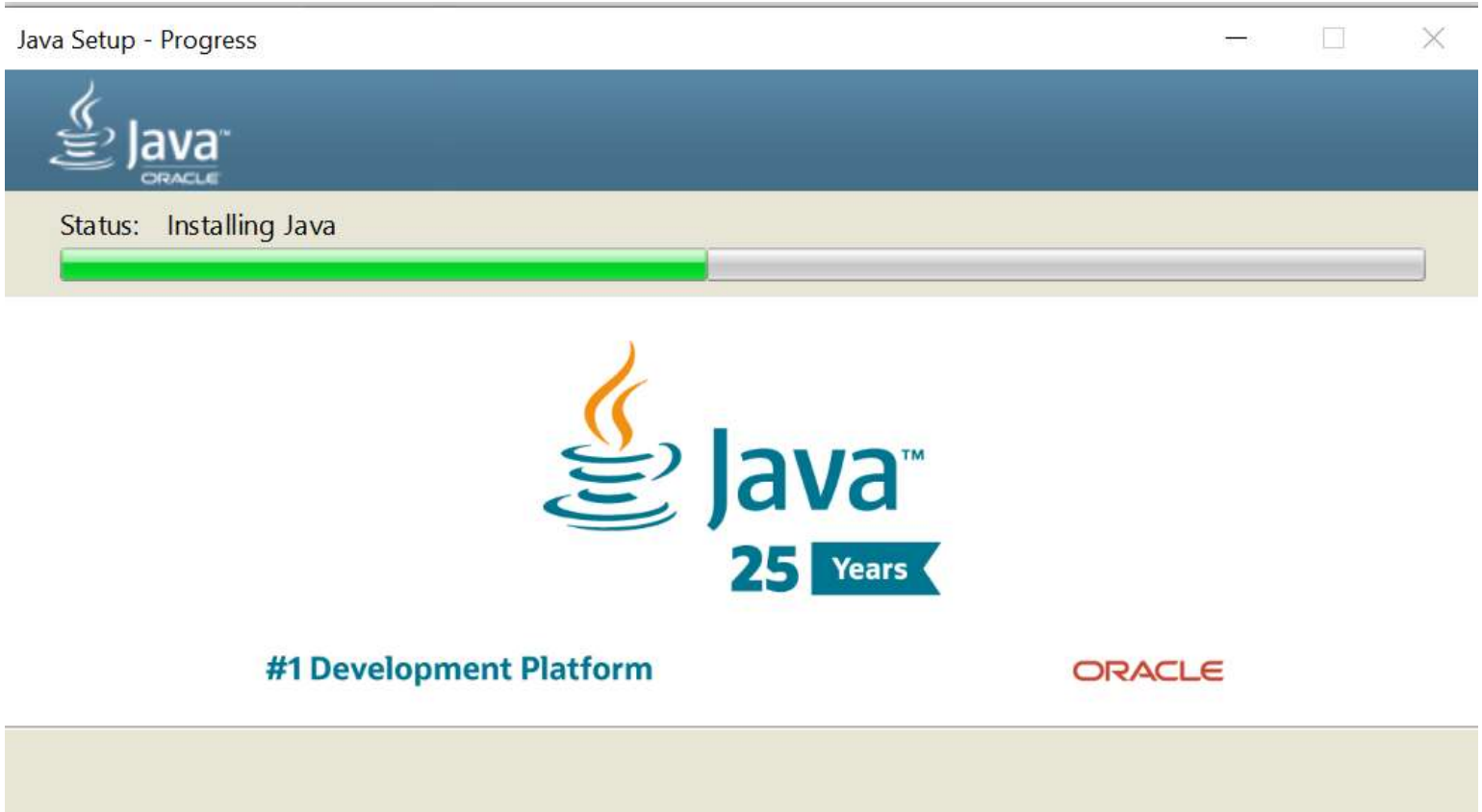
Install Steps – 2nd Window



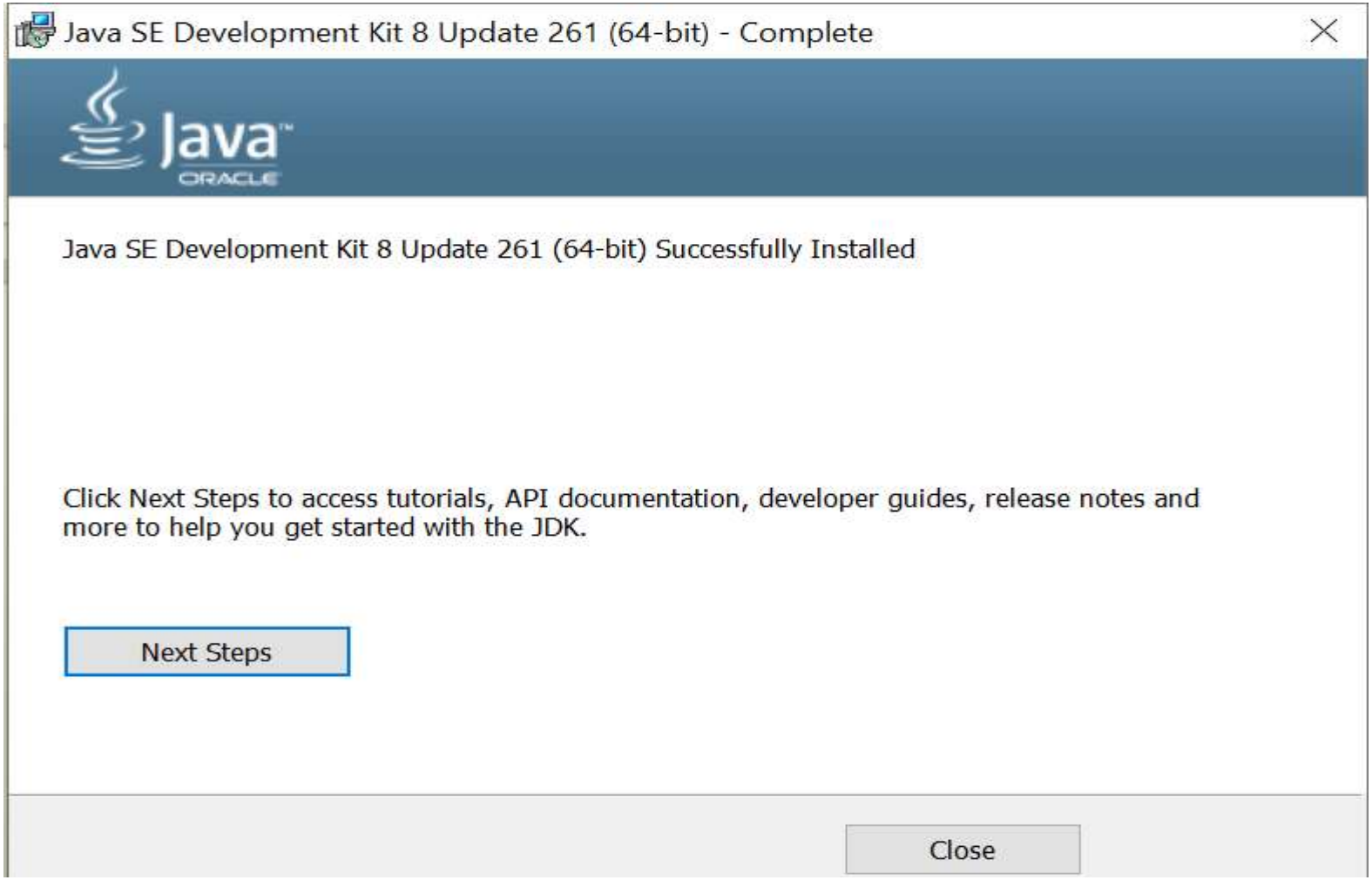
Install Steps – 3rd Window



Install Steps - 4th Window

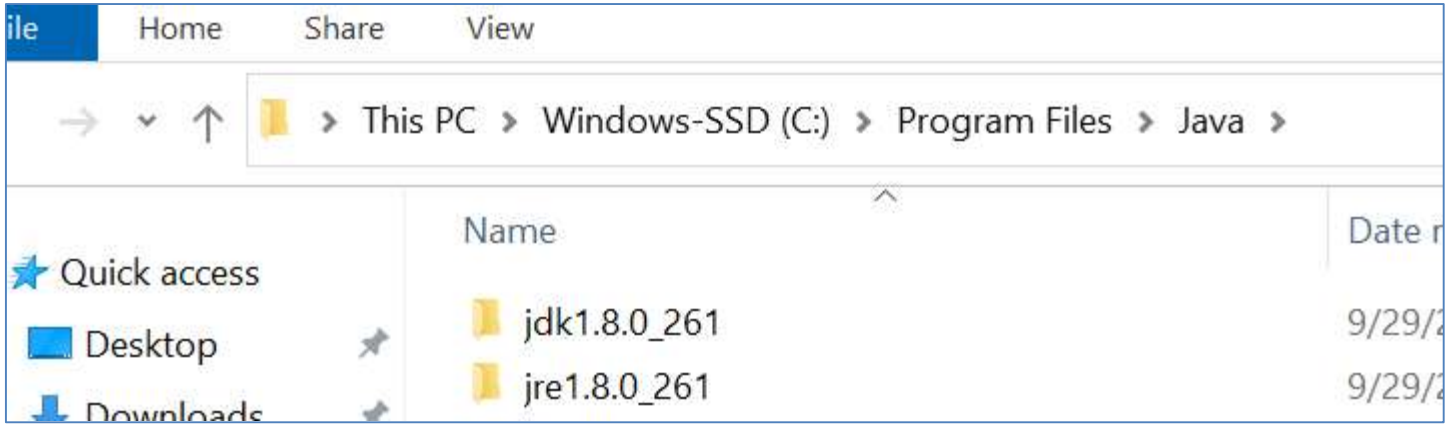


Install Steps – 5th Window



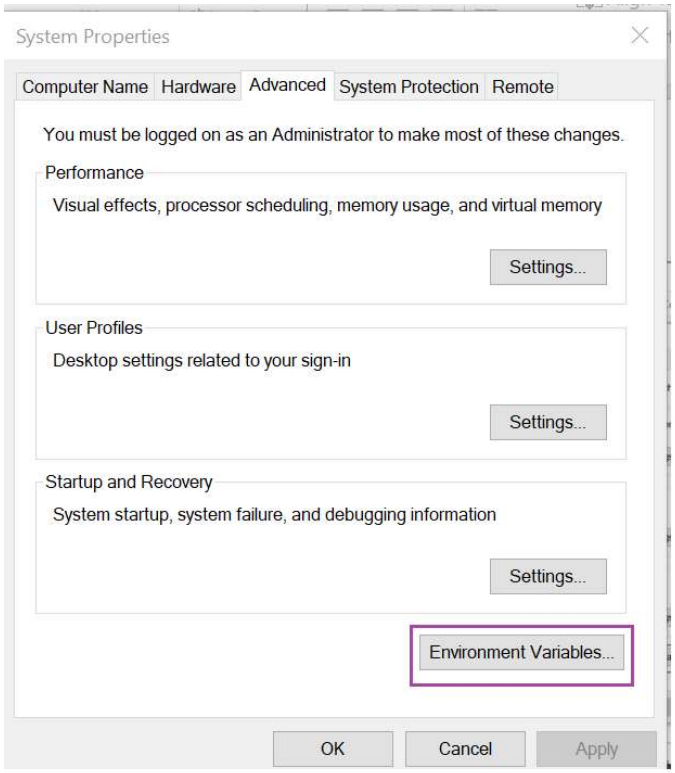
Setup

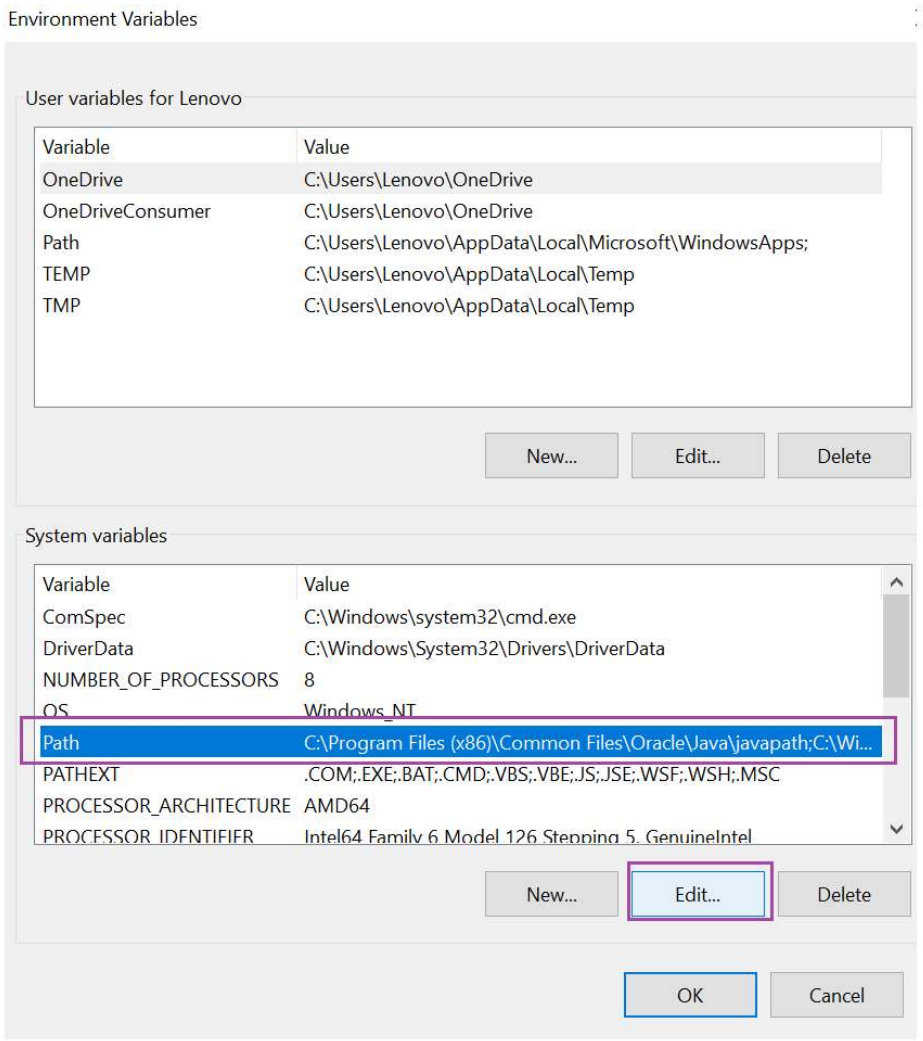
- Verify JDK and JRE on your hard disc.



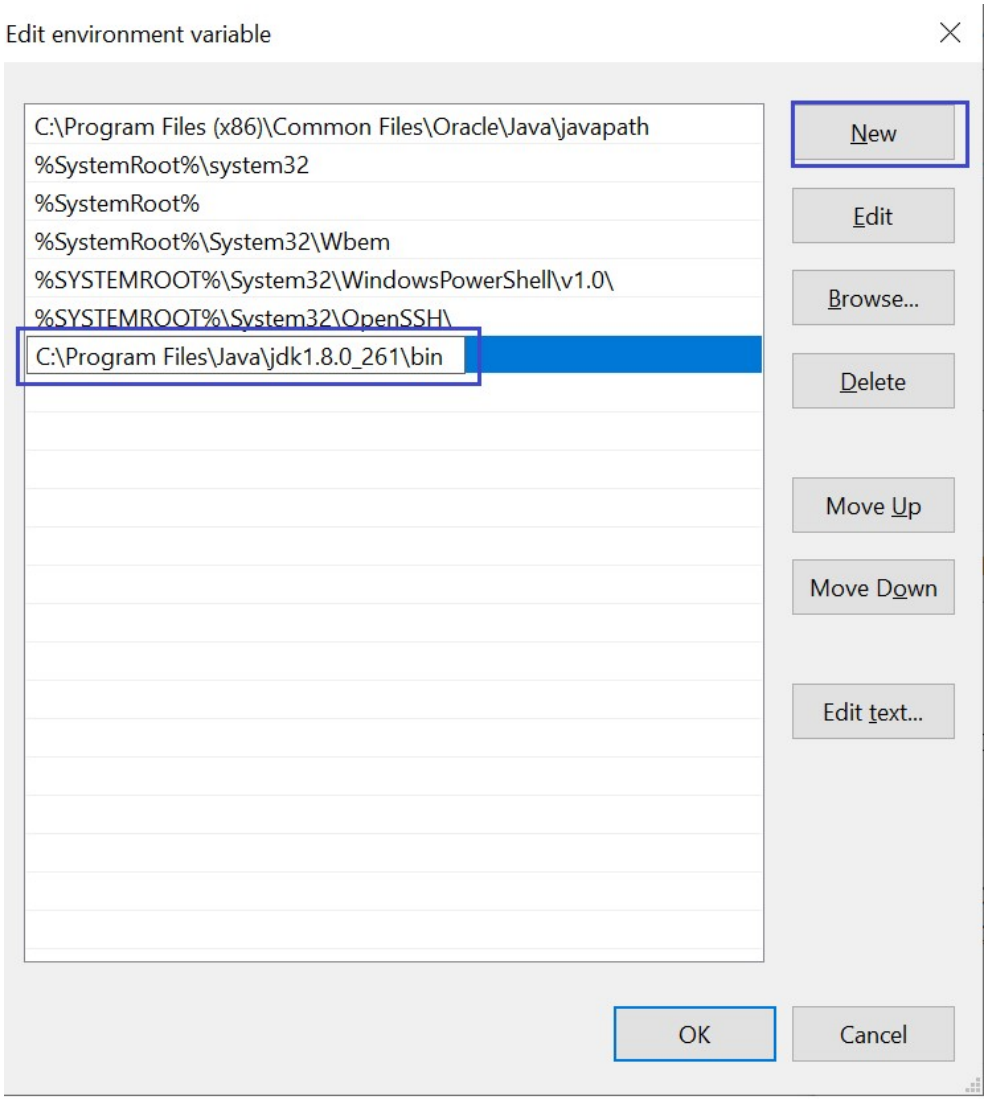
- Now, Set the path environment variable as explained in next slide.

Setup..


System Properties
Computer Name
Hardware
Advanced
System Protection
Remote
You must be logged on as an Administrator to make most of these changes.
Performance
Visual effects, processor scheduling, memory usage, and virtual memory
Settings...
User Profiles
Desktop settings related to your sign-in
Settings...
Startup and Recovery
System startup, system failure, and debugging information
Settings...
Environment Variables...
OK
Cancel
Apply


Environment Variables
User variables for Lenovo
Variable
Value
OneDrive
C:\Users\Lenovo\OneDrive
OneDriveConsumer
C:\Users\Lenovo\OneDrive
Path
C:\Users\Lenovo\AppData\Local\Microsoft\WindowsApps;
TEMP
C:\Users\Lenovo\AppData\Local\Temp
TMP
C:\Users\Lenovo\AppData\Local\Temp
New...
Edit...
Delete
System variables
Variable
Value
ComSpec
C:\Windows\system32\cmd.exe
DriverData
C:\Windows\System32\Drivers\DriverData
NUMBER_OF_PROCESSORS
8
OS
Windows_NT
Path
C:\Program Files (x86)\Common Files\Oracle\Java\javapath;C:\Wi...
PATHEXT
.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE
AMD64
PROCESSOR_IDENTIFIER
Intel64 Family 6 Model 126 Stepping 5, GenuineIntel
New...
Edit...
Delete
OK
Cancel

Setup…


Edit environment variable
C:\Program Files (x86)\Common Files\Oracle\Java\javapath
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
%SYSTEMROOT%\System32\WindowsPowerShell\v1.0\
%SYSTEMROOT%\System32\OpenSSH\
C:\Program Files\Java\jdk1.8.0_261\bin
New
Edit
Browse...
Delete
Move Up
Move Down
Edit text...
OK
Cancel

First Java Program

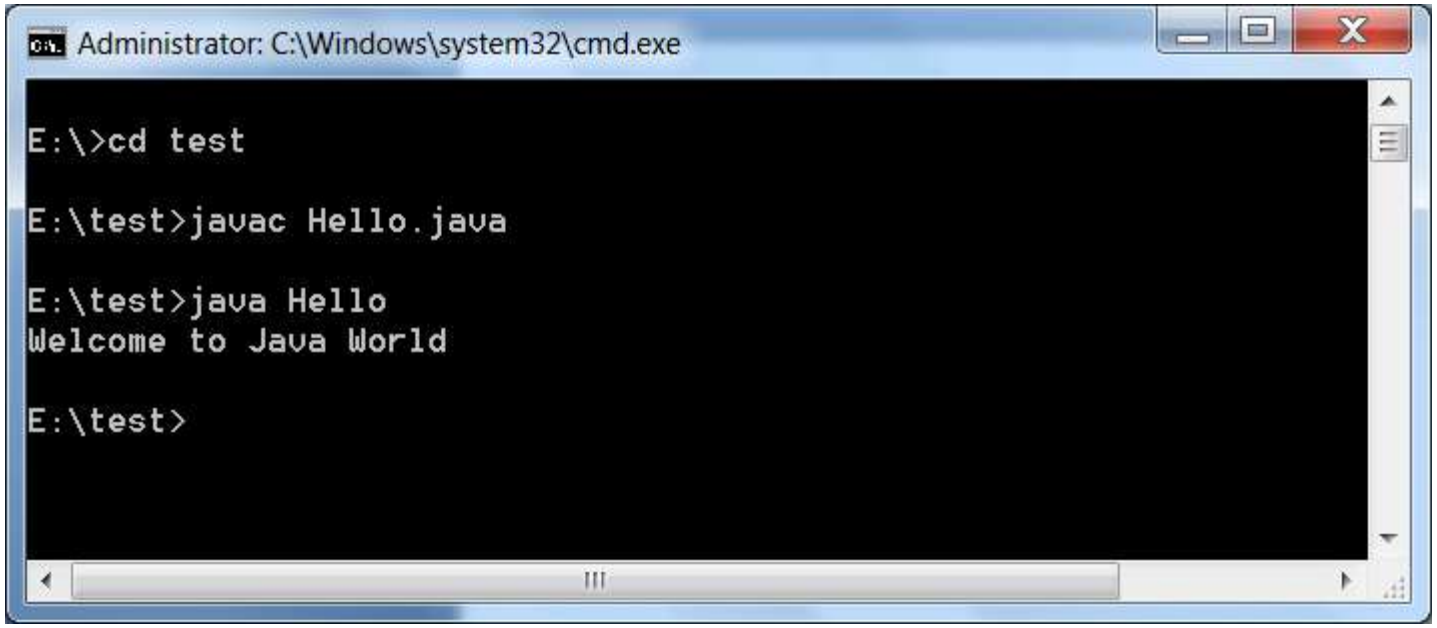
- What do you need?
  - Java Development Kit (JDK)
  - A Text Editor

- Open the notepad, write the following code and save the file as Hello.java (Let's say file is saved as E:/test/Hello.java)

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

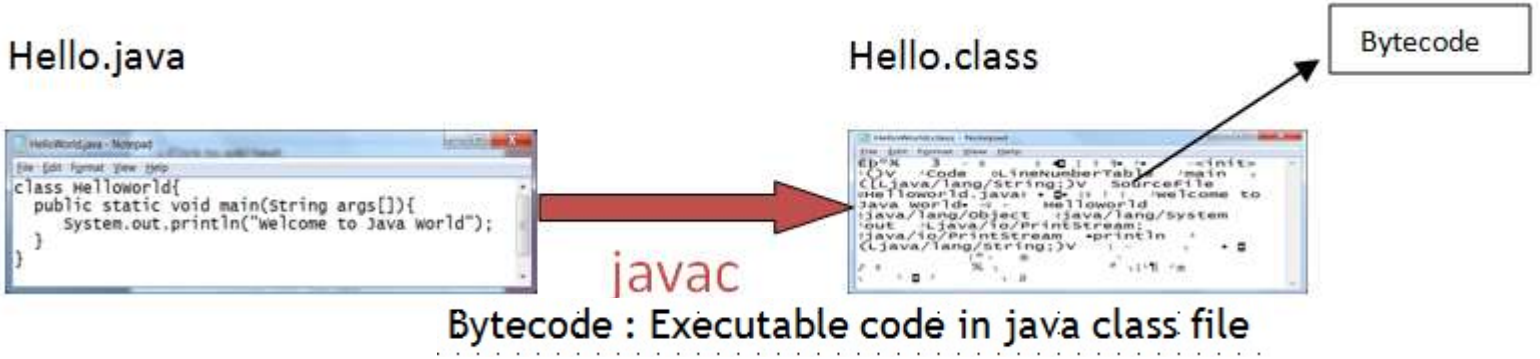
First Java Program

- Now, open command prompt, change directory to the folder where you have saved your file.
- Compile (javac Hello.java)
- Run (java Hello)



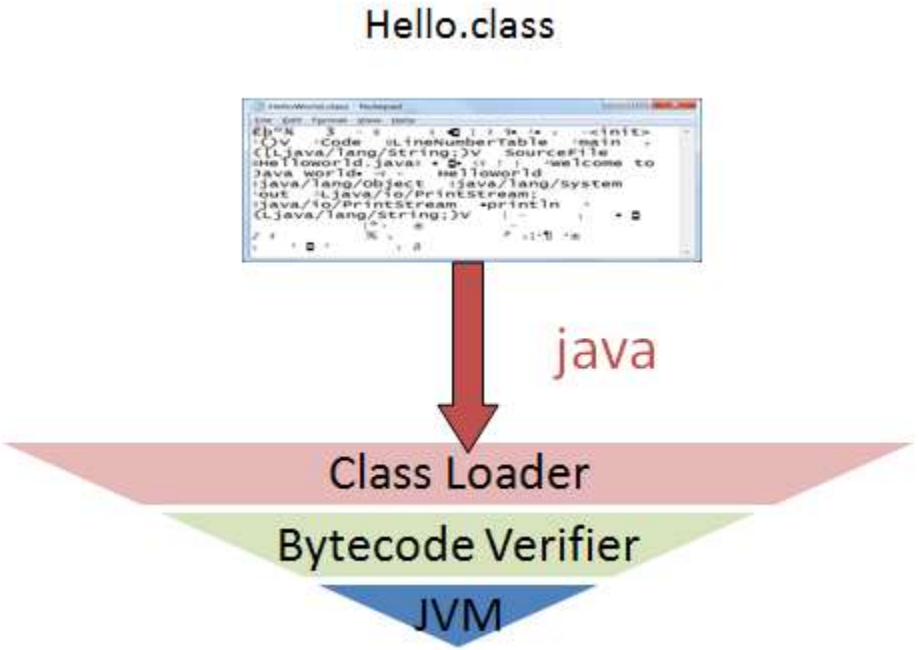
Java Environment

- Compilation
  - Java Compiler translates the java source file into the class file
  - Class file contains bytecodes, the "machine language" of the java Virtual machine (JVM).
  - This java class file can run on any hardware platform and Operating System, which hosts JVM.



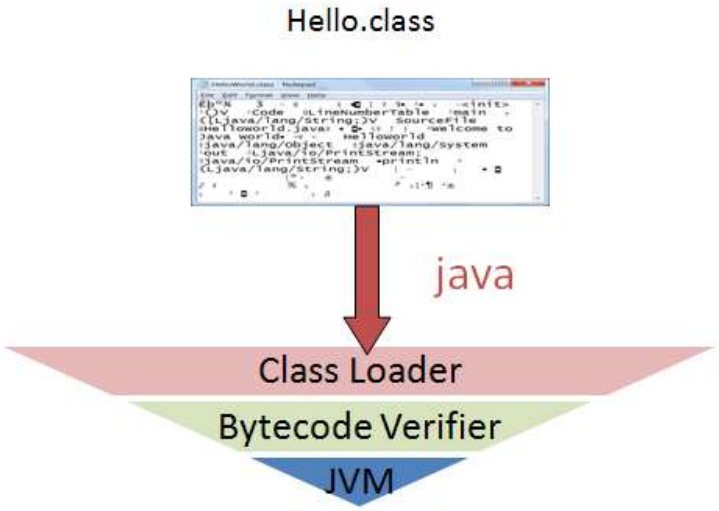
Java Environment

- Class Loader
  - Class Loader loads class files into memory.
  - Class file can be loaded from local disc, network or over the internet



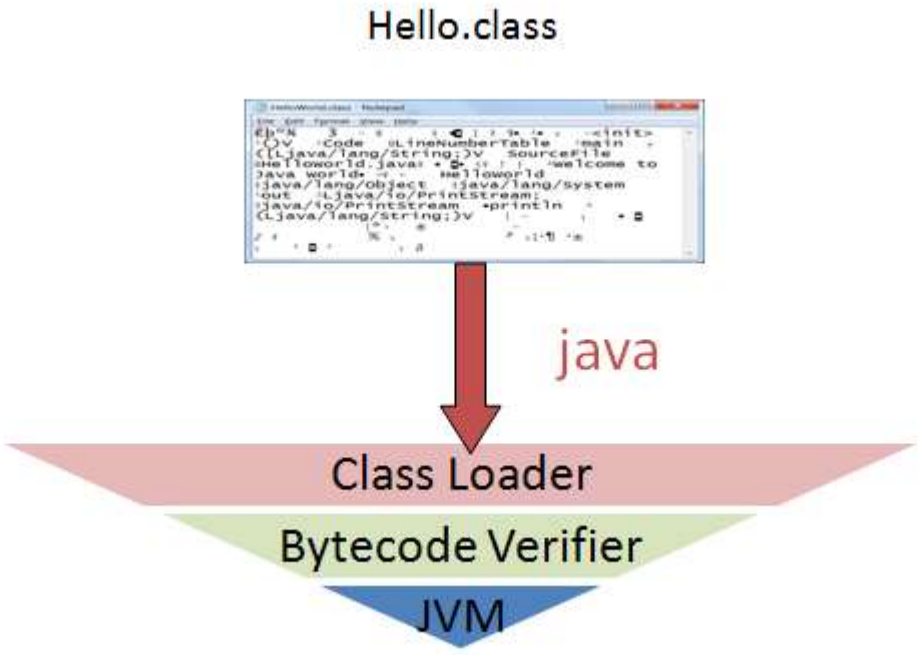
Java Environment

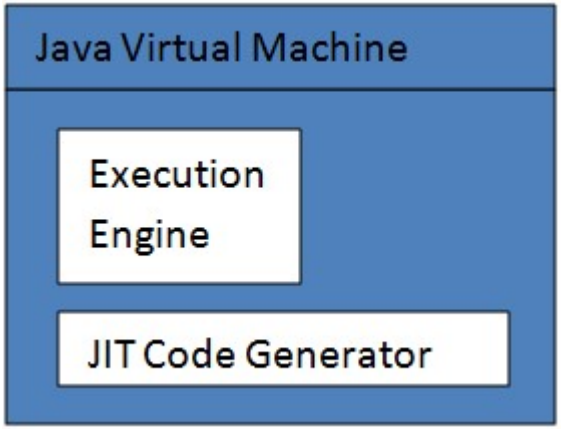
- Bytecode Verifier
  - It verifies that the bytecodes are valid and safe.
  - Does not violate java security restrictions
  - Checks the internal consistency of the class and validity of the code



Java Environment

- Java Virtual Machine
  - Translates the byte code into machine code depending upon the underling operating system and hardware combination. Which later executed by processor.

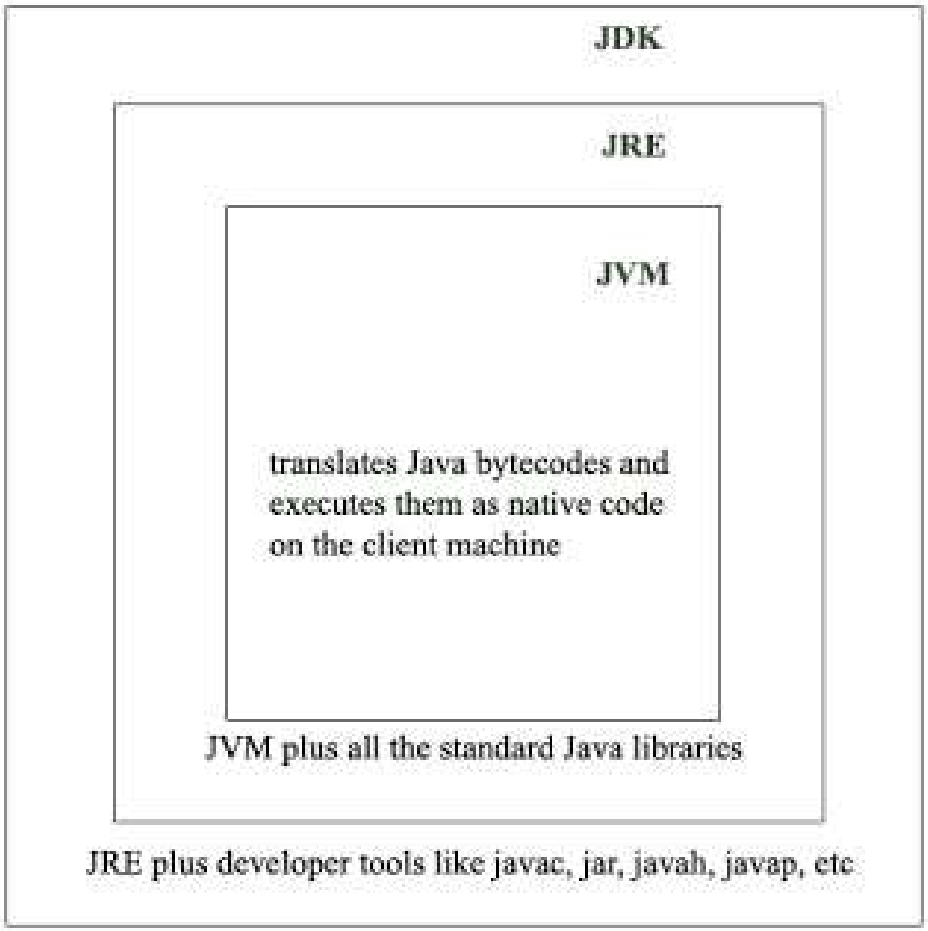




Java Terminology

- JDK : Java Development Kit
  - All you need to develop, compile, debug and run your java program.
- JRE : Java Runtime Environment
  - Subset of JDK
  - Includes Minimum Elements required to run java class file
  - Does not contain development tools like compiler, debugger etc.
- JVM : Java Virtual Machine
  - Actually runs the java program and uses the library and other supporting files provided by JRE

Java Terminology

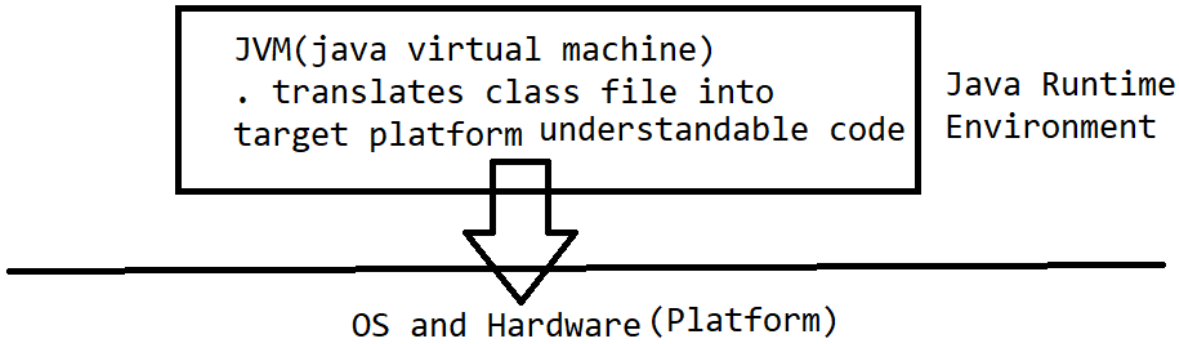


Points to remember

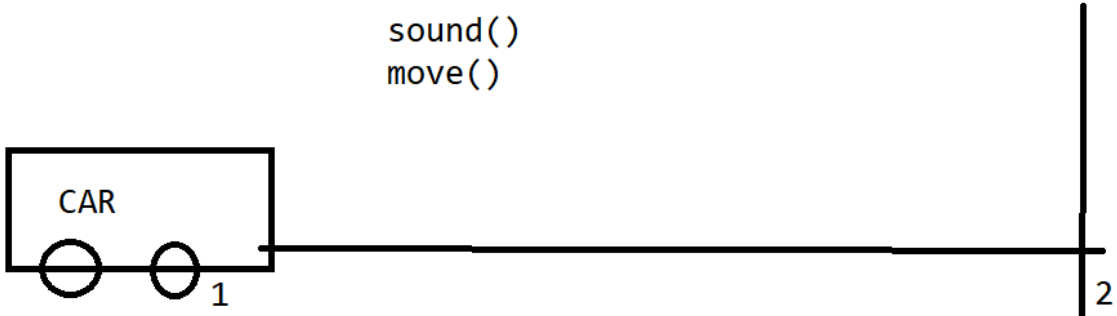
- Java Compiler, JVM ..All are platform dependent.
- Only java class file (Bytecode) is platform independent.


Q.A

java Hello


JVM(java virtual machine)
. translates class file into
target platform understandable code
Java Runtime
Environment
OS and Hardware (Platform)

Problem : Write a program to move a car from point 1 to point 2

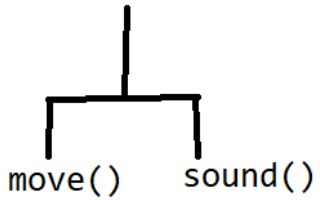



```
//Procedure (Function/Method) top-down
Car - how to draw car
void move(Object v, Point p1, Point 2){
   if(v is Car) move like car;
   if(v is Bus) move like Bus;
   ...Train
   ...Aeroplane
   ... bike
}
```

```
// single threaded(1 exec path)
void main(){
    move();
    sound();
}

void main(){
   run move() as well sound();
}
```

Object Oriented Approach (bottom up approach)

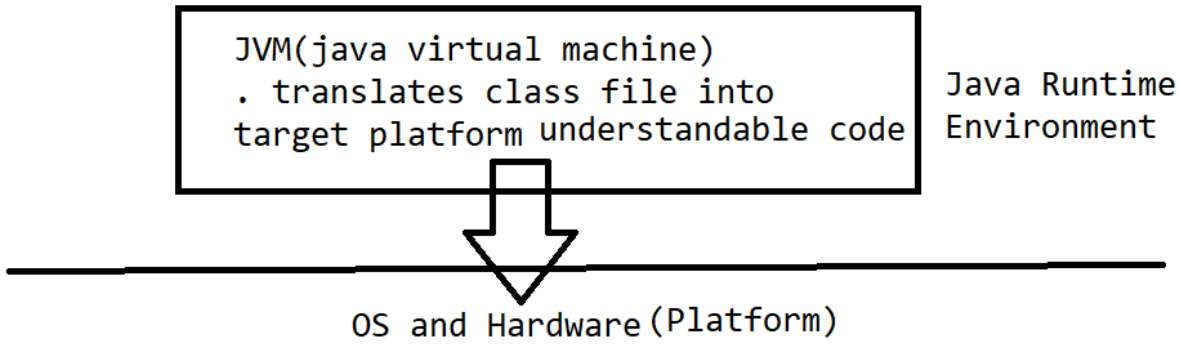
```
class Vehicle{....common functionaltiy...}
class Car extends Vehicle{

  move() {...}
}

class Bus extends Vehicle{
  move() {....}

}
```



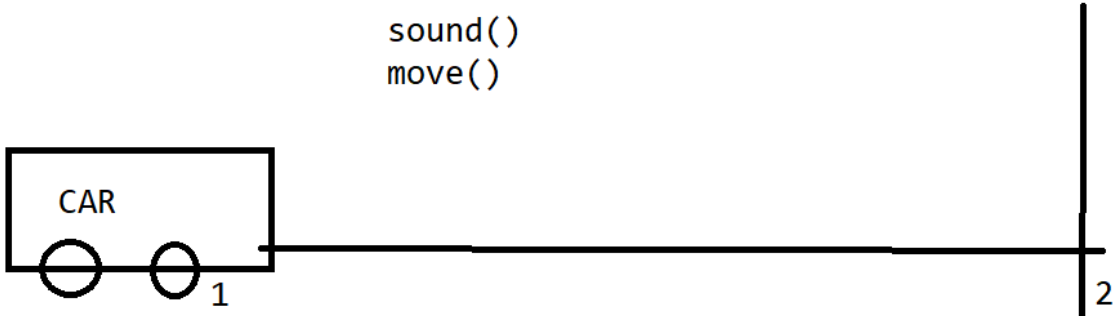
Maintanble
Exentable
Simpler

OO Concepts:
Abstraction, Encapsulation, Inheritance, Polymorphism

java Hello


JVM(java virtual machine)
. translates class file into
target platform understandable code
Java Runtime
Environment
OS and Hardware (Platform)

Problem : Write a program to move a car from point 1 to point 2

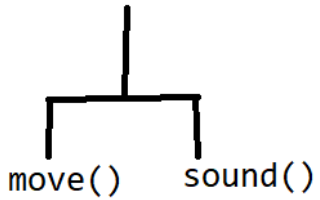



```
//Procedure (Function/Method) top-down
Car - how to draw car
void move(Object v, Point p1, Point 2){
   if(v is Car) move like car;
   if(v is Bus) move like Bus;
   ...Train
   ...Aeroplane
   ... bike
}
```

```
// single threaded(1 exec path)
void main(){
    move();
    sound();
}

void main(){
   run move() as well sound();
}
```

Object Oriented Approach (bottom up approach)

```
class Vehicle{....common functionaltiy...}
class Car extends Vehicle{

  move() {...}
}

class Bus extends Vehicle{
  move() {....}

}
```



Maintanble
Exentable
Simpler

OO Concepts:
Abstraction, Encapsulation, Inheritance, Polymorphism

Basic Language Elements

Shakir Hussain

Identifiers

- A name in a program is called an Identifier.
- Used to denote classes, methods, variables and labels.

- Each character can be either `letter` or a `digits`
- `First` character in an Identifier must be a `letter` (or underscore or currency symbols like $, but never recommended)

Keywords

- Reserved words that are predefined in the language and can not be used to denote other entities.
- All keywords are in `lower case`.
- A reserved word can not be used as an Identifier.

JAVA KEYWORDS				
abstract	continue	for	new	switch
assert	default	goto*	package	synchronized
boolean	do	if	private	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum****	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp**	volatile
const*	float	native	super	while

RESERVED KEYWORDS (not currently in use)				
const	goto			

RESERVED LITERALS				
null	true	false		

Literals

- Denotes a constant value, the value that a literal represents remains unchanged in the program.

- Example :

```
Boolean literals : true, false
Integer literals :  100, 200 etc.
```

Primitive Types and Variables

- boolean, char, byte, short, int, long, float, double etc.
- These basic (or primitive) types are the only types that are not objects (due to performance issues).
- This means that you don't use the new operator to create a primitive variable.
- Declaring primitive variables:

```
float initVal;
int retVal, index = 2;
double gamma = 1.2, brightness ;
boolean valueOk = false;
```

Initialisation

- If no value is assigned to local variable prior to use, then the compiler will give an error
- Java sets primitive variables to zero or false in the case of a boolean variable[non-local variable]
- All object references are initially set to null
- An array of anything is an object
  - Set to null on declaration

Declarations

```
int index = 1.2;            // compiler error
boolean retOk = 1;          // compiler error
double fiveFourths = 5 / 4;   // no error!
float ratio = 5.8f;         // correct
double fiveFourths = 5.0 / 4.0; // correct
```

- 1.2f is a float value accurate to 7 decimal places.
- 1.2 is a double value accurate to 15 decimal places.

Assignment

- All Java assignments are right associative

```
int a = 1, b = 2, c = 5;
a = b = c;
System.out.print(
"a= " + a + "b= " + b + "c= " + c)
```

- What is the value of a, b & c
- Done right to left: `a = (b = c);`

Basic Mathematical Operators

- `*` `/` `%` `+` `-` are the mathematical operators
- `*` `/` `%` have a higher precedence than `+` or `-`

```
double myVal = a + b % d - c * d / b;
```

- Is the same as:

```
double myVal = (a + (b % d)) -
                    ((c * d) / b);
```

Statements & Blocks

- A simple statement is a command terminated by a semi-colon:

```
name = "Fred";
```

- A block is a compound statement enclosed in curly brackets:

```
{
    name1 = "Fred"; name2 = "Bill";
}
```

- Blocks may contain other blocks

Flow of Control

- Java executes one statement after the other in the order they are written
- Many Java statements are flow control statements:

```
Alternation:   if, if else, switch
Looping:       for, while, do while
Escapes:       break, continue, return
```

If – The Conditional Statement

- The if statement evaluates an expression and if that evaluation is true then the specified action is taken

```
if ( x < 10 ) x = 10;
```

- If the value of x is less than 10, make x equal to 10
- It could have been written:

```
if ( x < 10 )
x = 10;
```

- Or, alternatively:

```
if ( x < 10 ) { x = 10; }
```

Relational Operators

==	Equal (careful)
!=	Not equal
>=	Greater than or equal
<=	Less than or equal
>	Greater than
<	Less than

If… else

- The if … else statement evaluates an expression and performs one action if that evaluation is true or a different action if it is false.

```
if (x != oldx) {
  System.out.print("x was changed");
}
else {
  System.out.print("x is unchanged");
}
```

Nested if … else

```
if ( myVal > 100 ) {
  if ( remainderOn == true) {
      myVal = mVal % 100;
  }
  else {
    myVal = myVal / 100.0;
  }
}
else
{
  System.out.print("myVal is in range");
}
```

else if

- Useful for choosing between alternatives:

```
if ( n == 1 ) {
  // execute code block #1
}
else if ( j == 2 ) {
  // execute code block #2
}
else {
  // if all previous tests have failed, execute
  code block #3
}
```

A Warning…

WRONG!

```
if( i == j )
    if ( j == k )

      System.out.print
      (
          "i equals
      k");
     else
      System.out.print(
       "i is not equal
       to j");
```

CORRECT!

```
if( i == j ) {
  if ( j == k )
  System.out.print(
       "i equals k");
}
else
  System.out.print("
  i is not equal to
  j");    //
  Correct!
```

The switch Statement

```
switch ( n ) {
  case 1:
    // execute code block #1
    break;
  case 2:
    // execute code block #2
    break;
    default:
    // if all previous tests fail then
    //execute code block #4
    break;
}
```

The for loop

- Loop n times

```
for ( i = 0; i < n; n++ ) {
  // this code body will execute n times
  // ifrom  0 to n-1
}
```

- Nested for:

```
for ( j = 0; j < 10; j++ ) {
  for ( i = 0; i < 20; i++ ){
    // this code body will execute 200 times
  }
}
```

while loops

```
while(response == 1) {
  System.out.print( "Hello World ");
  n++;
  response = readInt( "Enter response?");
}
```

What is the minimum number of times the loop is executed?

What is the maximum number of times?

do {… } while loops

```
do {
  System.out.print( "Hello World");
  n++;
  response = readInt( "Enter response?" );
}while (response == 1);
```

What is the minimum number of times the loop is executed?

What is the maximum number of times?

Break

- A break statement causes an exit from the innermost containing while, do, for or switch statement.

```
for ( int i = 0; i < maxID, i++ ) {
  if ( userID[i] == targetID ) {
    index = i;
    break;
  }
} // program jumps here after break
```

Continue

- Can only be used with while, do or for.
- The continue statement causes the innermost loop to start the next iteration immediately

```
for ( int i = 0; i < maxID; i++ ) {
  if ( userID[i] != -1 ) continue;
  System.out.print( "UserID " + i + " :" +
    userID);
}
```

Break with label

```
first:
for( int i = 0; i < 10; i++) {
  second:
  for(int j = 0; j < 5; j ++ ) {
    break myLabel;
  }
}
```

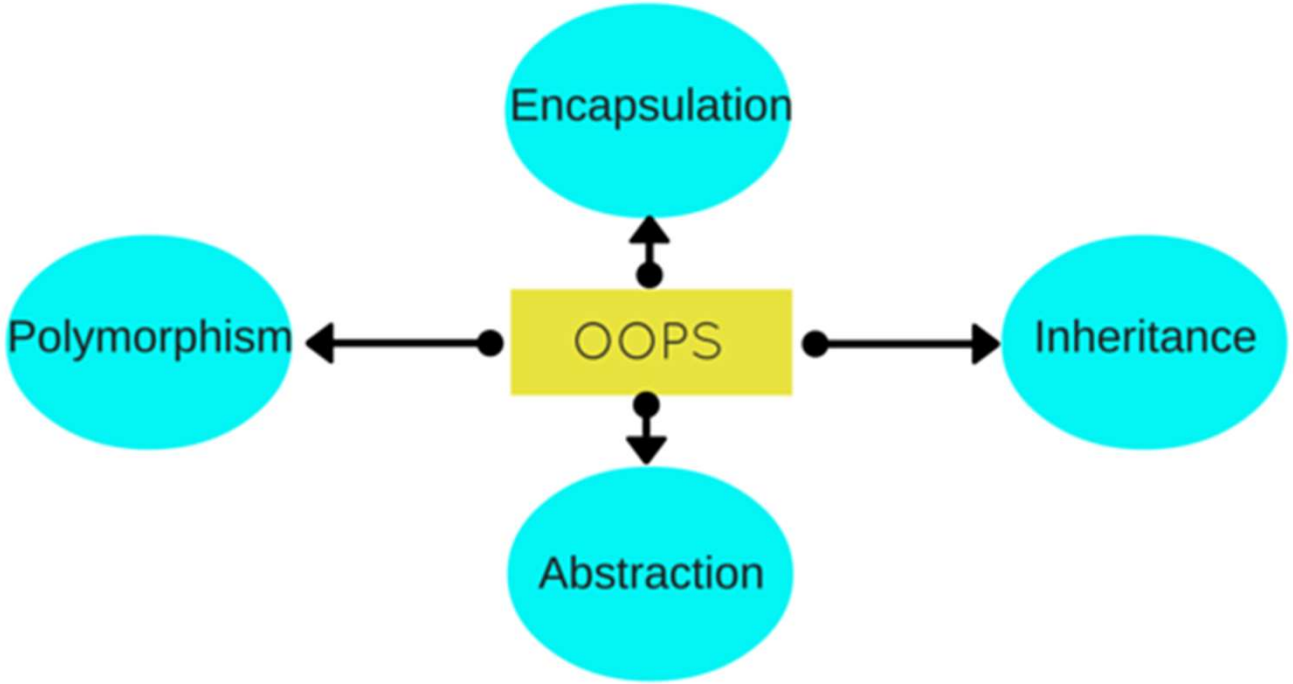
You can replace **myLabel** with first or second.

Similarly , you can have continue with label in java.


Q&A

Object Oriented Concepts

Shakir Hussain



Need Of Object Oriented Approach

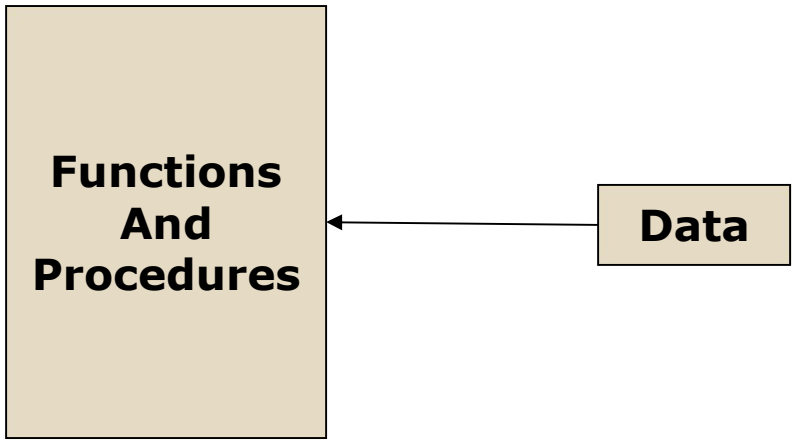
- Software is Inherently Complex
  - Impedance mismatch between user of a system and it's developer.
  - Changing Requirements during development.
  - Difficulty of managing software development process. It's a team effort.
  - Easy User Interface.
  - Clients want systems to be adaptable and Extensible

Object Oriented Approach

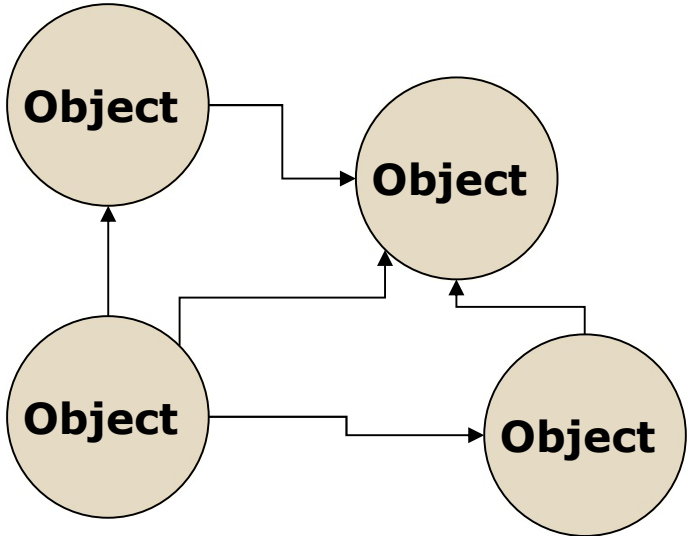
- The Claim
  - Object oriented approach helps to handle the complexity of software development and aids in generation of adaptable and Extensible Systems

Procedural Vs Object Oriented Approach

Procedural Method



Object Oriented Method



Object ?

- An object is an entity that has well defined structure and behavior
- Characteristics of an object
  - State
  - Behavior
  - Identity

State of an Object

Car Attributes

Color
Average
Make
Power
Fuel type

Static

Speed
Fuel level
Tyre pressure
Gear

Dynamic

Values of all attributes at any moment defines the state of the car

Behavior of an Object

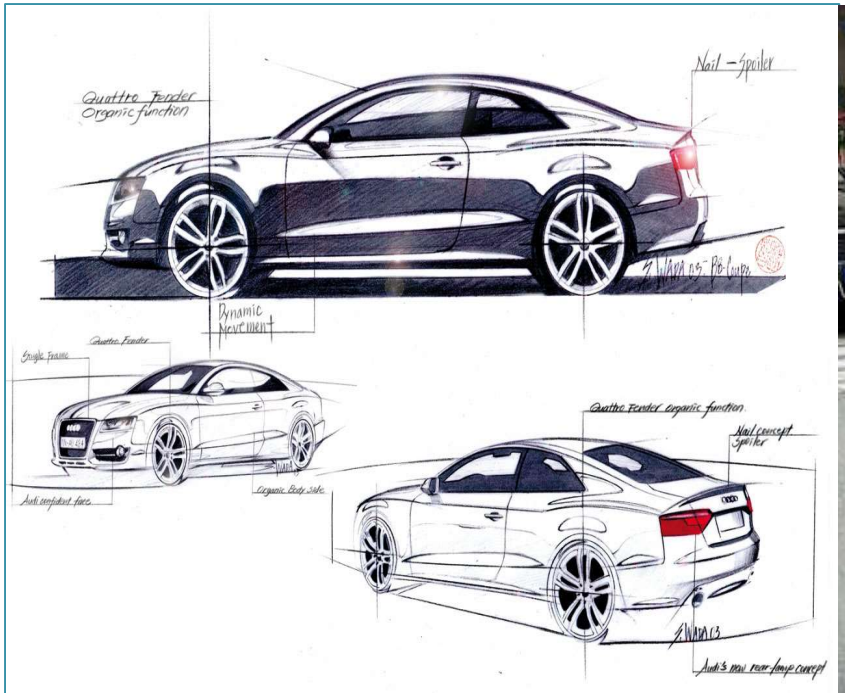
- Behavior is how an object acts or reacts, in terms of its state changes and operations performed upon it

Identity of an Object

- Identity is that property of an object which distinguishes it from all other objects

Class ?

- Class denotes a category of objects and act as a blueprint for creating such objects.
- An object is an instance of a Class.
- Class : Model/Template/Prototype/Blueprint



Major concepts of OOPS

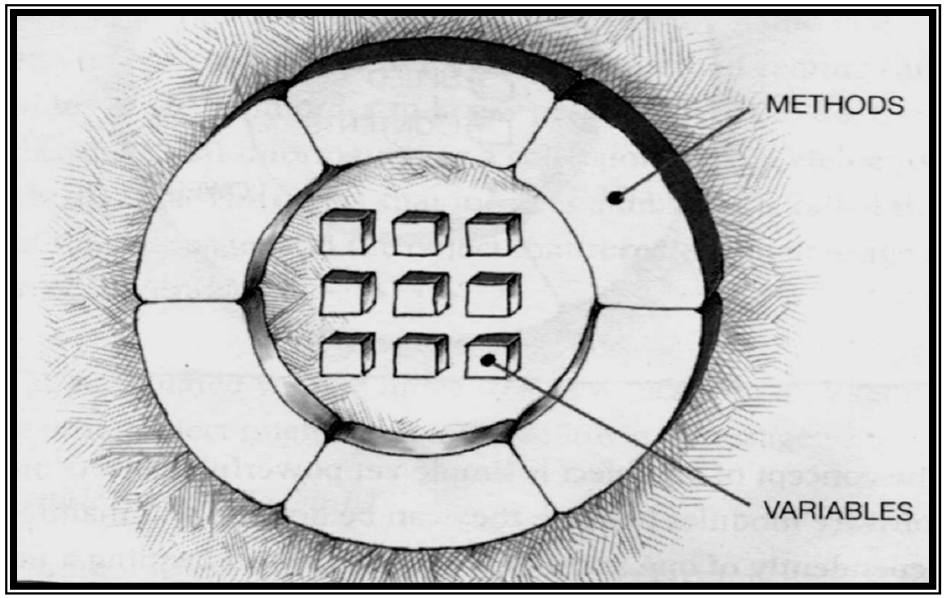
- Abstraction
- Encapsulation
- Inheritance (IS –A)
- Polymorphism

Abstraction

- Abstraction is the process of identifying the key aspects of an entity and ignoring the rest
- We select only those aspects which are important to us
- Only Domain Expertise can do right abstraction

Encapsulation

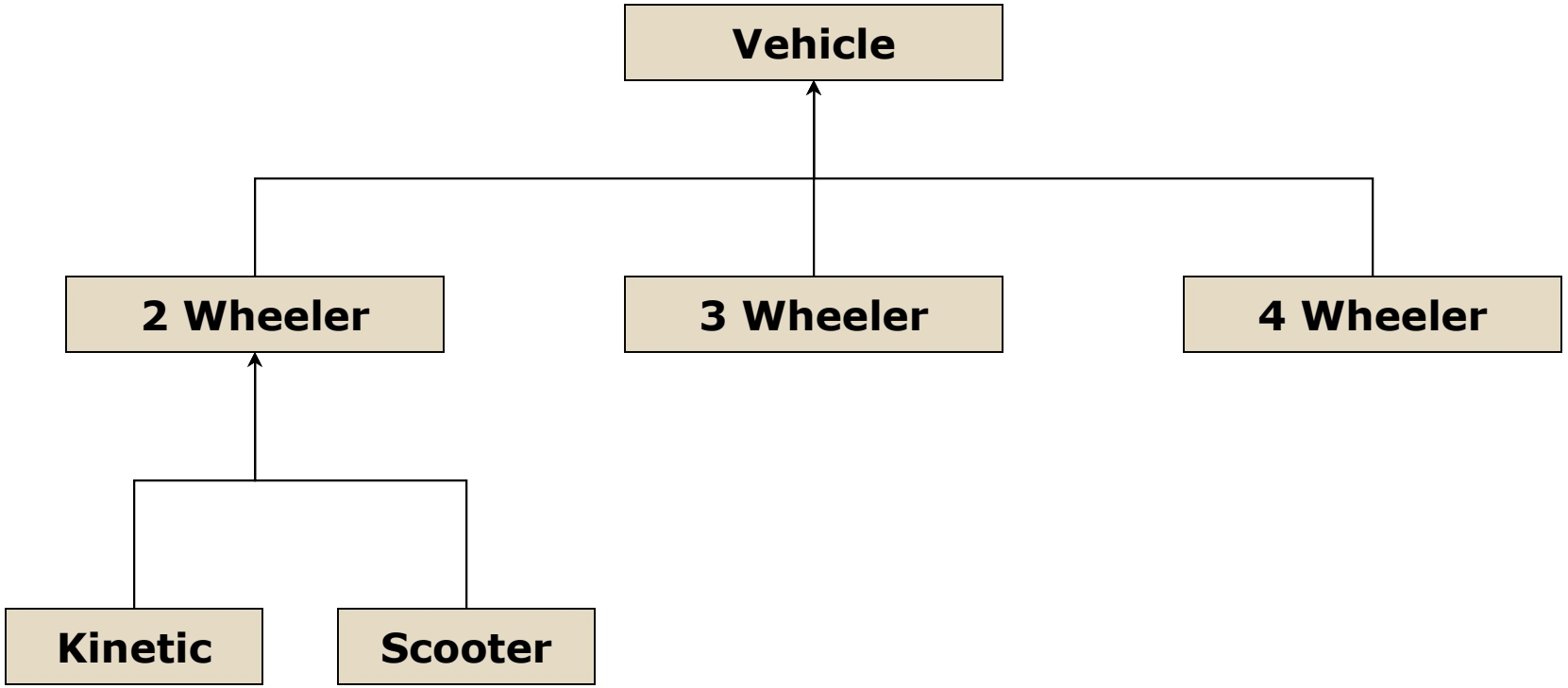
- Binding state of the object with its behavior
- Encapsulation ensures that data within an object is protected; it can be accessed only by its methods



Inheritance

Creating new things from existing one.

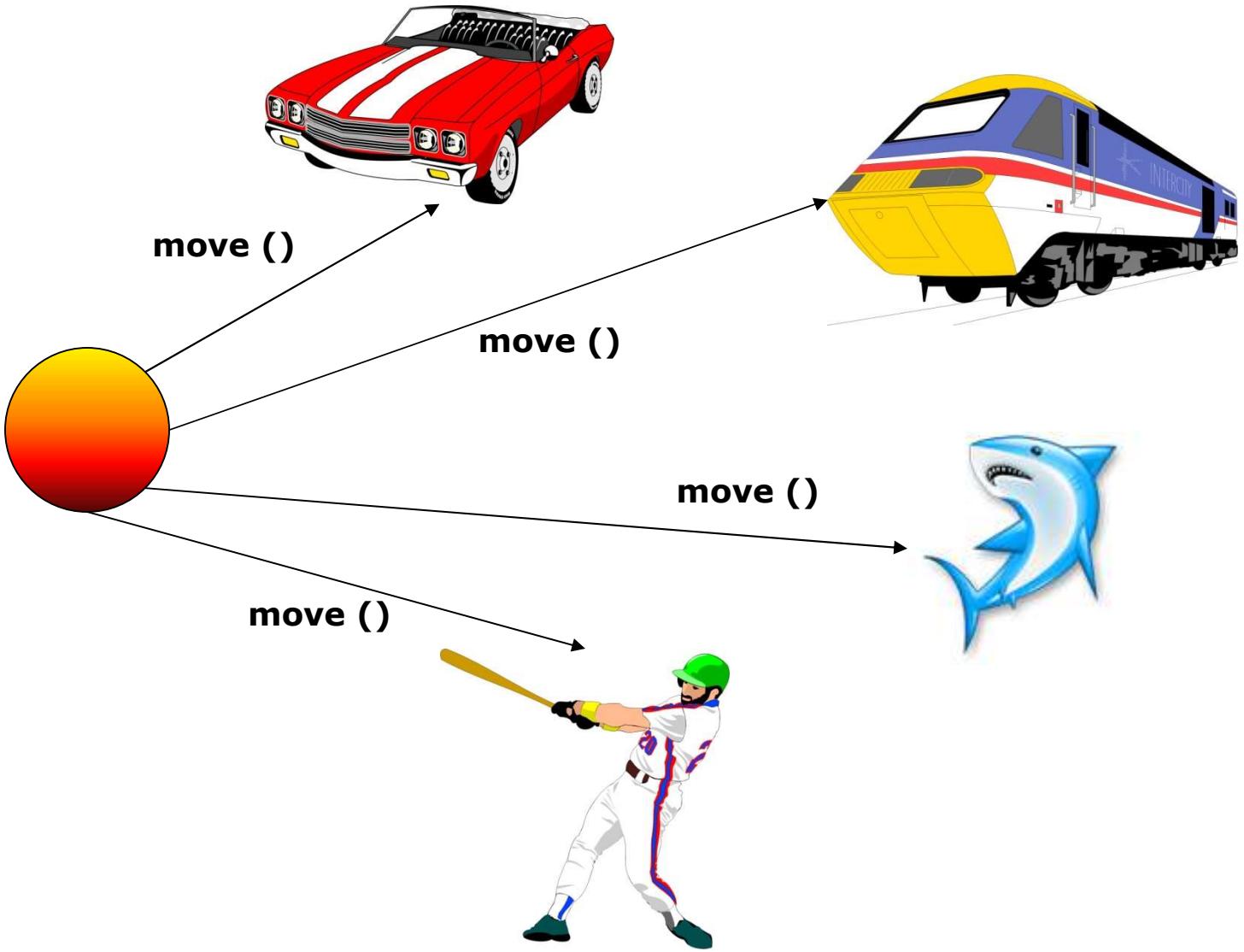
'is-a' kind of hierarchy



Polymorphism

- The ability of different objects to respond to the same message in different ways is called polymorphism
- Polymorphism helps us to :
  - Design extensible software as we can add new objects to the design without rewriting existing procedures

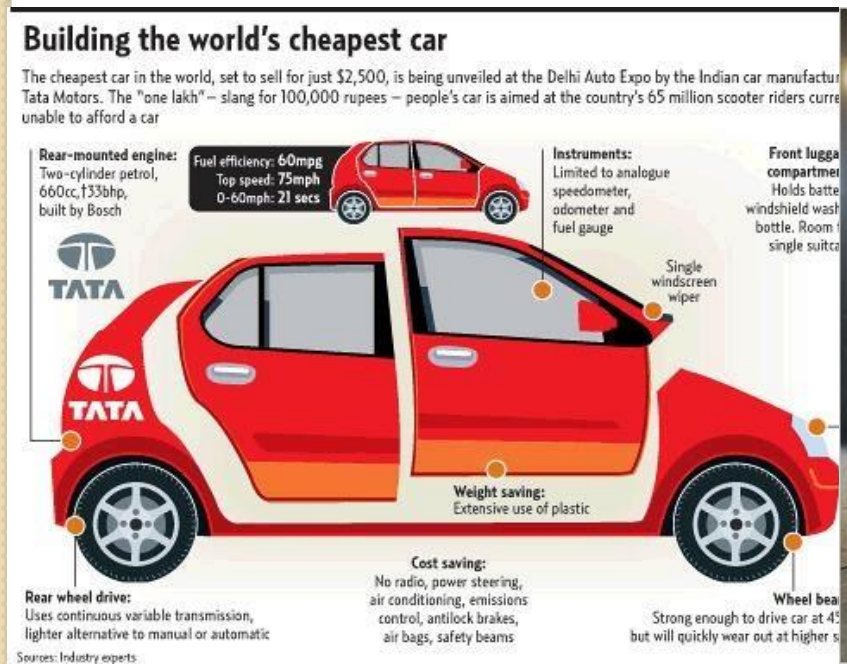
Polymorphism




Q.A

Java Classes and Objects

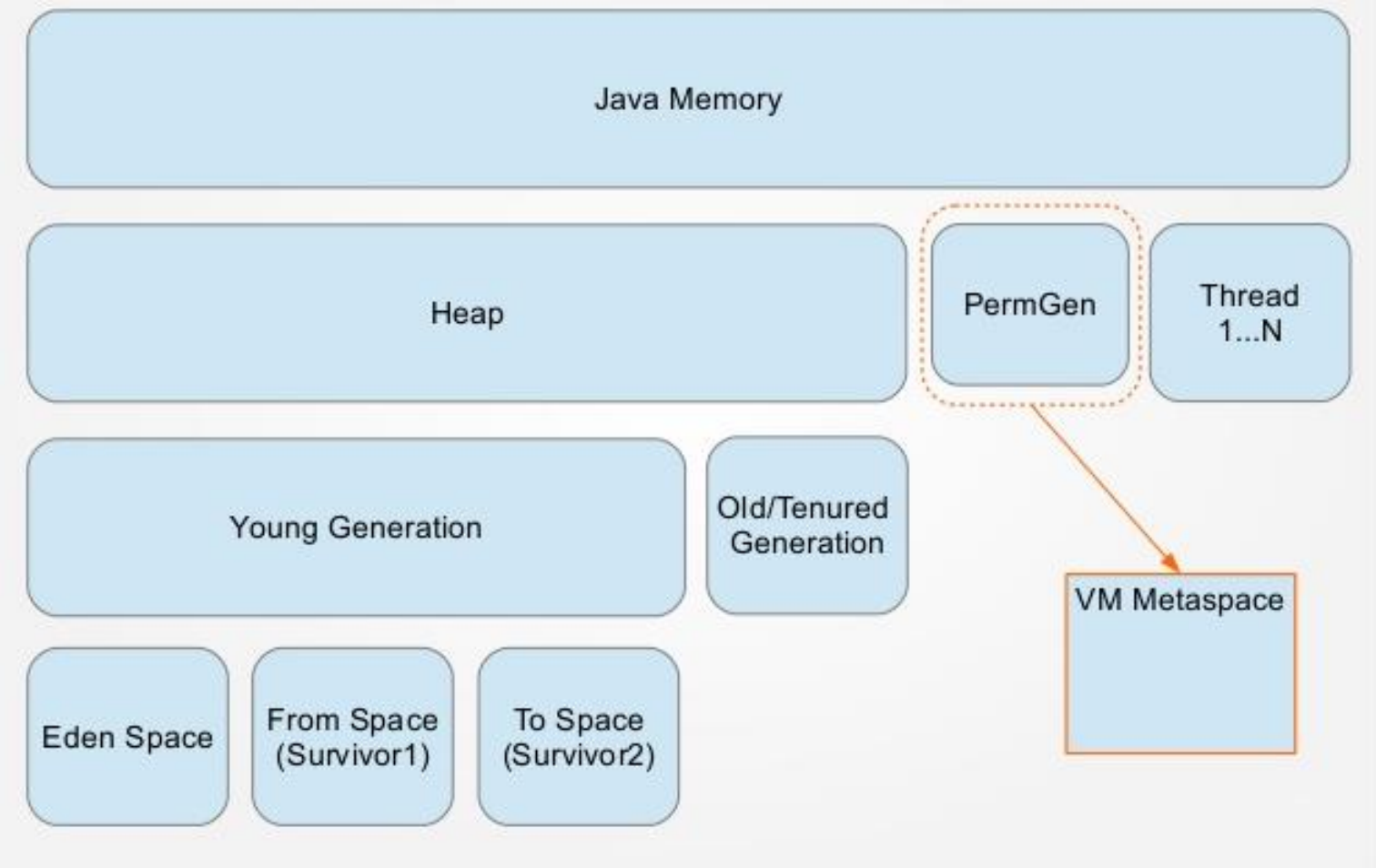
- Shakir Hussain



Will cover

- Memory Model in Brief
- Writing java class with
  - instance variables & members
  - Reference variables
  - Method overloading
  - “this” reference
- Writing java class with
  - Constructors
- Static variables & methods
- Arrays
- Packages
- Access Specifier
- Static Import
- **Introduction to Eclipse IDE for Java development**

Java Memory model



Reference data types

- Primitive data types are built in.
- array & user defined datatypes are reference
- Key difference between primitive and reference datatypes is how they are represented.
- Primitive type variable hold the actual value of the variable
- Reference type variable hold the value of a reference to the object.

Method overloading

- Reusing the same name for a method.
- Arguments should be different.
  - Type of arguments
  - Number of arguments
  - Order of arguments
- The method calls are resolved at compile time using the method signature.
- Compile time error occurs if compiler can't match the arguments or if more than one match is possible.

“this” reference

- Every class member gets a hidden parameter : the this reference.
- this is a keyword in Java.
- this points to the current object.
- this always holds address of an object which is invoking the member method.
- It is often a logic error when a method contains a parameter or local variable that has the same name as a field of the class. In this case use *this* reference if you wish to access the field of the class- otherwise a method parameter or local variable will be referenced

Constructor

```
class Emp {
    private int empId;
    private String empName;

    Emp() {}  //default constructor
    Emp(int id, String name) //parameterized constuctor
    {empId=id; this.empName=name;}

    public  void displayEmpDetails(){
       SOP("empId="+empId); SOP("empName="+empName);
    }

}
```

Constructor …

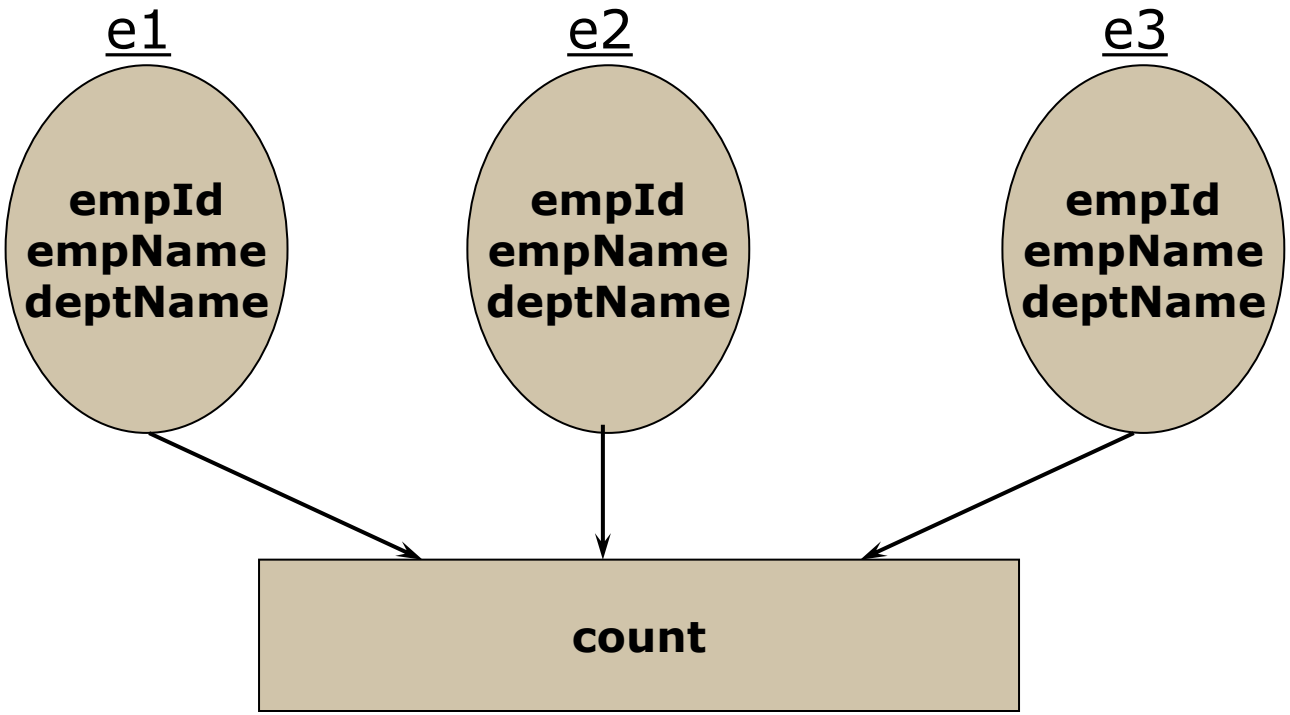
- Used to initialize the data members of the class
- Special method with same name as it's class name
- No return type for constructor. Not even void
- Constructors are implicitly called when objects are created
- By default, the compiler provides a default constructor with no parameters in any class that does not explicitly include a constructor
- Constructors can be overloaded

- No destructor in java.

Static variable

- Only single copy exists per class
- It's a class variable
- Can be accessed outside of its class through the use of the class name.
- Can be accessed from inside any member class method with or without the class name.

Static variables in Memory



Static Member methods

- Static member methods can access static data members only.
- Static member method is invoked using class name

  class name . method name()

- Reference *this* is never passed to a static member function

Static block

- Static block can be used to initialized static data members
- Static block executes as soon as the class loaded in the memory.

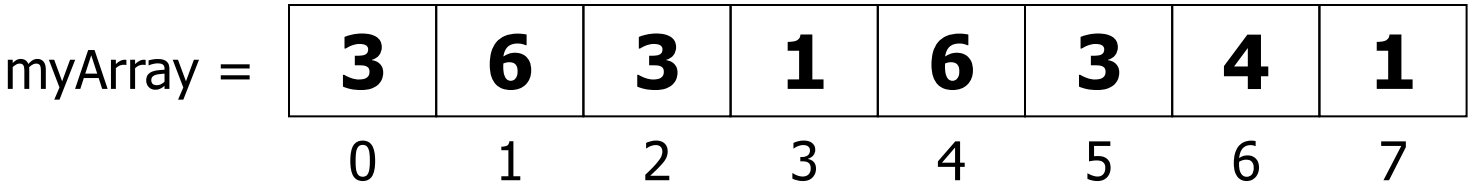
Variables in java

- Class variables: Static variables. Copy created per class.
- Instance variables : Copy created per instance of the class.
- Local variables: occur within methods or blocks: Copy created per method call.

If used, must be initialized or the compiler complains.

Arrays

- An array is a list of similar things
- An array has a fixed:
  - name
  - type
  - length
- These must be declared when the array is created.
- Arrays sizes cannot be changed during the execution of the code



myArray has room for 8 elements

- the elements are accessed by their index
- in Java, array indices start at 0

Declaring Arrays

```
int myArray[];
```

declares *myArray* to be an array of integers

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

sets up 8 integer-sized spaces in memory, labelled *myArray[0]* to *myArray[7]*

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

combines the two statements in one line

Assigning Values

- refer to the array elements by index to store values in them.

```
myArray[0] = 3;
myArray[1] = 6;
myArray[2] = 3;   ...
```

- can create and initialise in one step:

```
int myArray[] = {3, 6, 3, 1, 6, 3, 4, 1};

int myArray[]=new int[]{10,20,30};
```

Iterating Through Arrays

for loops are useful when dealing with arrays:

```
for (int i=0;i<myArray.length; i++){
  System.out.println(myArray[i]);
}
```

Arrays of Objects

- So far we have looked at an array of primitive types.
  - integers
  - could also use doubles, floats, characters…
- Often want to have an array of objects
  - Students, Books, Loans ……
- Need to follow 3 steps.

Array of Objects ...

1. Declare the array

```
private Student studentList[];
```

   - this declares studentList

2 .Create the array

```
studentList = new Student[10];
```

   - this sets up 10 spaces in memory that can hold references to Student objects

3. Create Student objects and add them to the array:

```
studentList[0] = new Student("Cathy",
"Computing");
```

Multi-Dimensional Array

- Array of arrays :

```
int twoDim[][]=new int[2][];
twoDim[0]=new int[2];
twoDim[1]=new int[5];

int twoDim[][]=new int[][2];     // illegal
int twoDim[][]=new int[2][2];    // legal
```

- Can have non-rectangular array

Array Resizing

- Can not resize an array.
- Can use the same reference variable to refer to an entirely new array

```
int  elements[]=new int[10];
        elements=new int[20];
```

Arrays: First class object

- Arrays are first class objects in Java.
  - int arr[ ] ;

    arr = new int[10];
  - int arr[ ] = {10, 20, 30, 40, 50}; //first class object

- Array references are stored on stack.
- The actual array is created on heap.

Recursion

- Recursion is a process by which a method calls itself again and again.
- Method which calls itself again and again is called Recursive Method.
- It is used to solve complicated problem by breaking down into simple problem.

Recursion – Two basic parts

- Method call which can call itself i.e. recursive
- A precondition that will stop the recursion

Recursion – Pros & Cons

- When a recursive call is made, new storage locations for variables are allocated on the stack. As, each recursive call returns, the old variables and parameters are removed from the stack. Hence, recursion generally uses more memory and is generally slow.

- On the other hand, a recursive solution is much simpler and takes less time to write, debug and maintain.

Packages

- Used to group related classes and interfaces.
- Convenient for organizing your work & separating your work from code libraries provided by others
- Reduce problems with naming conflicts
- Classes with same name can be put into different packages

Creating package

- Write a **package** statement at the top of the source file in which the type(class or interface) is defined

```
package pkg;
public class Emp {
}
```

- Create a subdirectory of that package name. Compile the file and keep the .class files in the directory.
- Alternatively, you can directly compile like:

```
javac –d . Emp.java
```

Here, the compiler will create directory structure for you.

Using package

```
import pkg.*;
class EmpTest{
   public static void main(String args[]){
        Emp e = new Emp();
          pkg.Emp e2=new pkg.Emp();
   }
}
```

- Import statement imports all the public classes within the package; it does not import sub packages.

Finding Packages and CLASSPATH

- How does the Java run-time system know where to look for packages that you create?

- Packages are like directories.
- You can specify a directory path or paths by setting the CLASSPATH environmental variable

CLASSPATH

- For java to be able to use a class, it has to be able to find that class on the file system.

- Java uses 2 elements to find classes
  - The package name
  - The directories listed in CLASSPATH variable

How Compiler/JVM locates a File

- Check current directory.
- Compiler looks through all directories specified in the CLASSPATH for

  the actual class file

  OR

  the subdirectory that has the name of the imported package.
- Then, looks for the file in one of the imported packages.
- Finally looks for file in java.lang package.
- If compiler still does not locate the file, it gives an error.

Access Specifiers

- private
- Package-level (default, if not specified)
- protected
- Public

- Either four can be applied to class members(variable or function)

- Only, Package-Level & public allowed for class.

Access Modifier example

- Create a package having a class , which will hold

public, private,default,protected members (field & methods)

- Create another package having Test class which will access above package and class members
- Which members you will be able to call ?

Static Import

- Starting with JDK 5.0, the **import** statement has been enhanced to permit the importing of static methods and fields, not just classes.

  For example if you add the directive

  import static java.lang.System.;*

  to the top of the source file, then you can use static methods and fields of the System class without the class name prefix – for e.g.

  out.println("Hello World");

- You can also import a specific method or a field

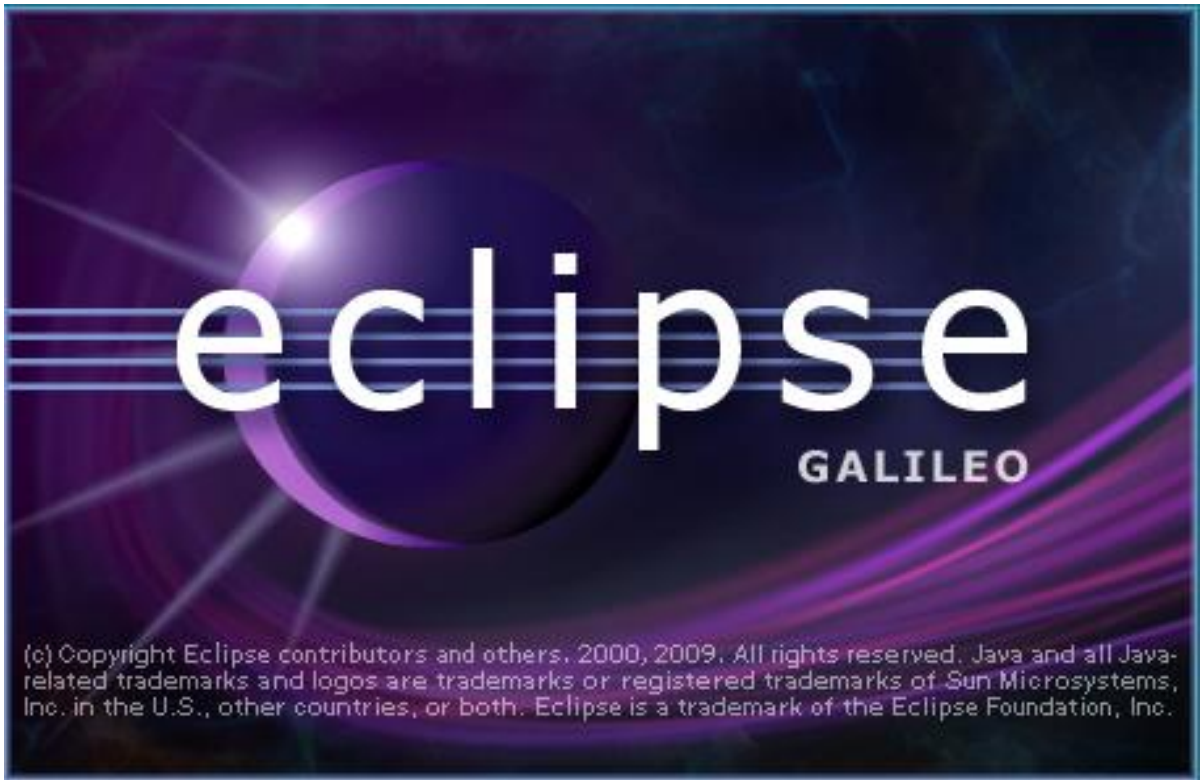
  import static java.lang.System.out;

Static Import : Precaution

- Makes the code confusing to read
- It can create naming conflicts.

  For example, if you have two different classes with an "add()" method, how will you and the compiler know which one to use?

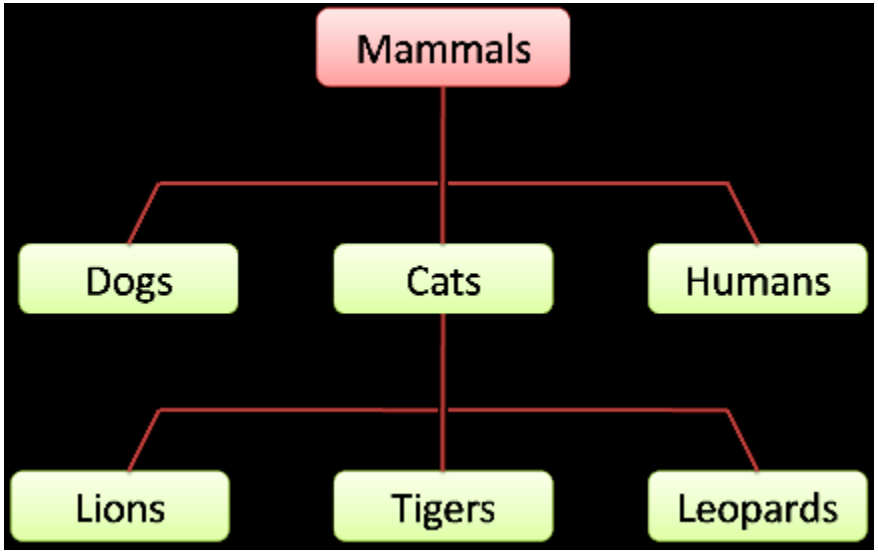
Introduction to eclipse IDE for Java Development




QA

Writing java classes II

Shakir Hussain



Will cover :

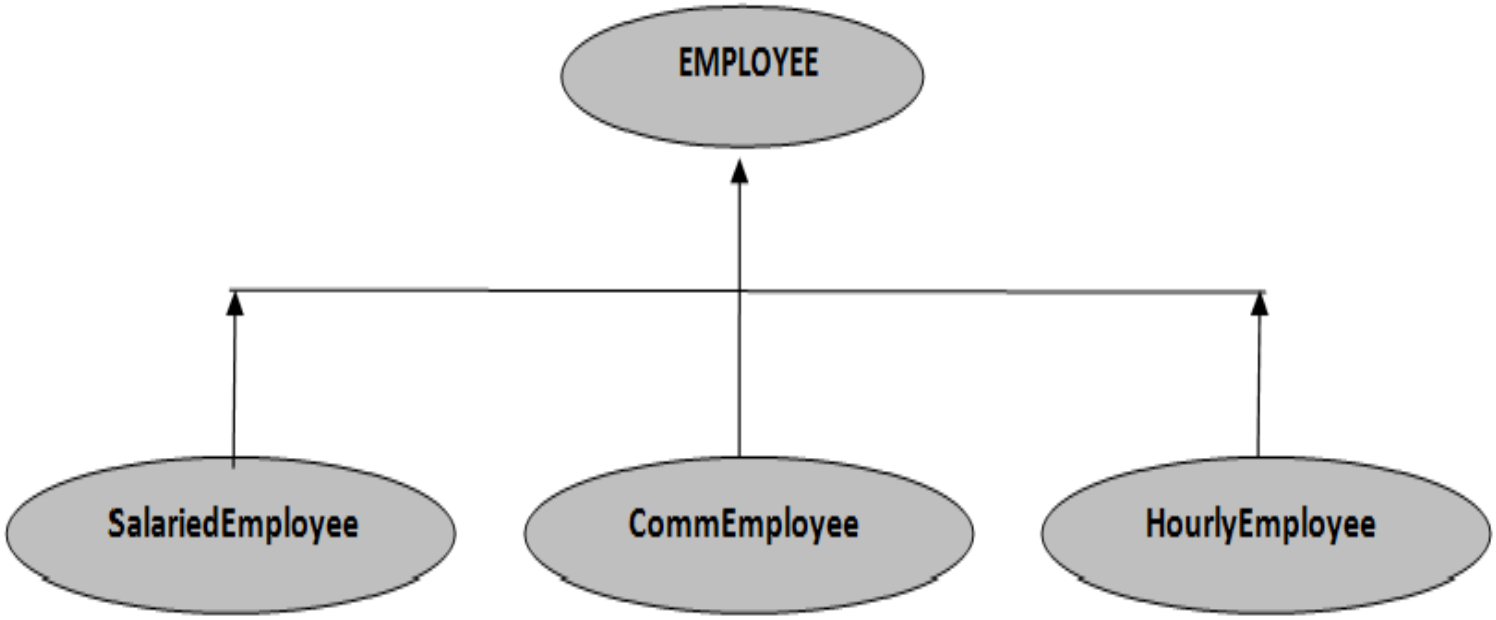
- Inheritance
- Polymorphism : Method Overriding
- Keyword : super
- Abstract Class
- Interface
- final : class, method, variable

Inheritance

- Inheritance is the capability of a class to use the properties and methods of another class while adding its own functionality.

Example : You could create a generic employee class with states and actions that are common to all employees. Then more specific classes could be defined for salaried, commissioned and hourly employees.

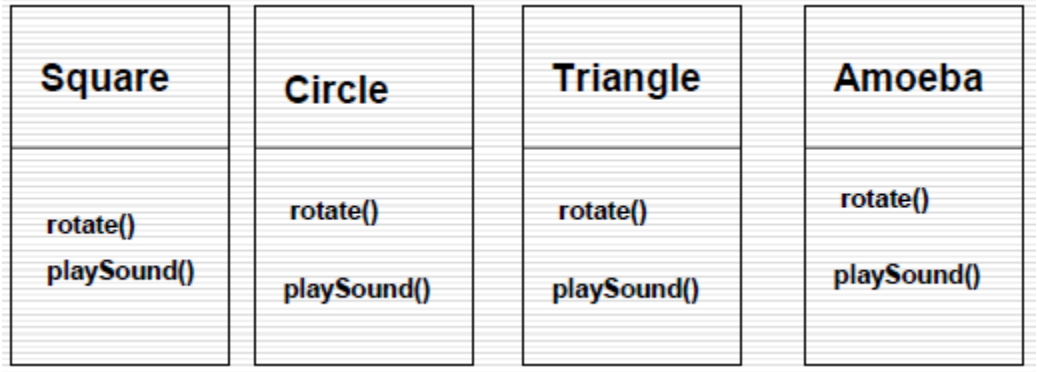
Inheritance..



Generic class is known as Parent/Super/Base class
Specific class is known as Child/Sub/Derived class

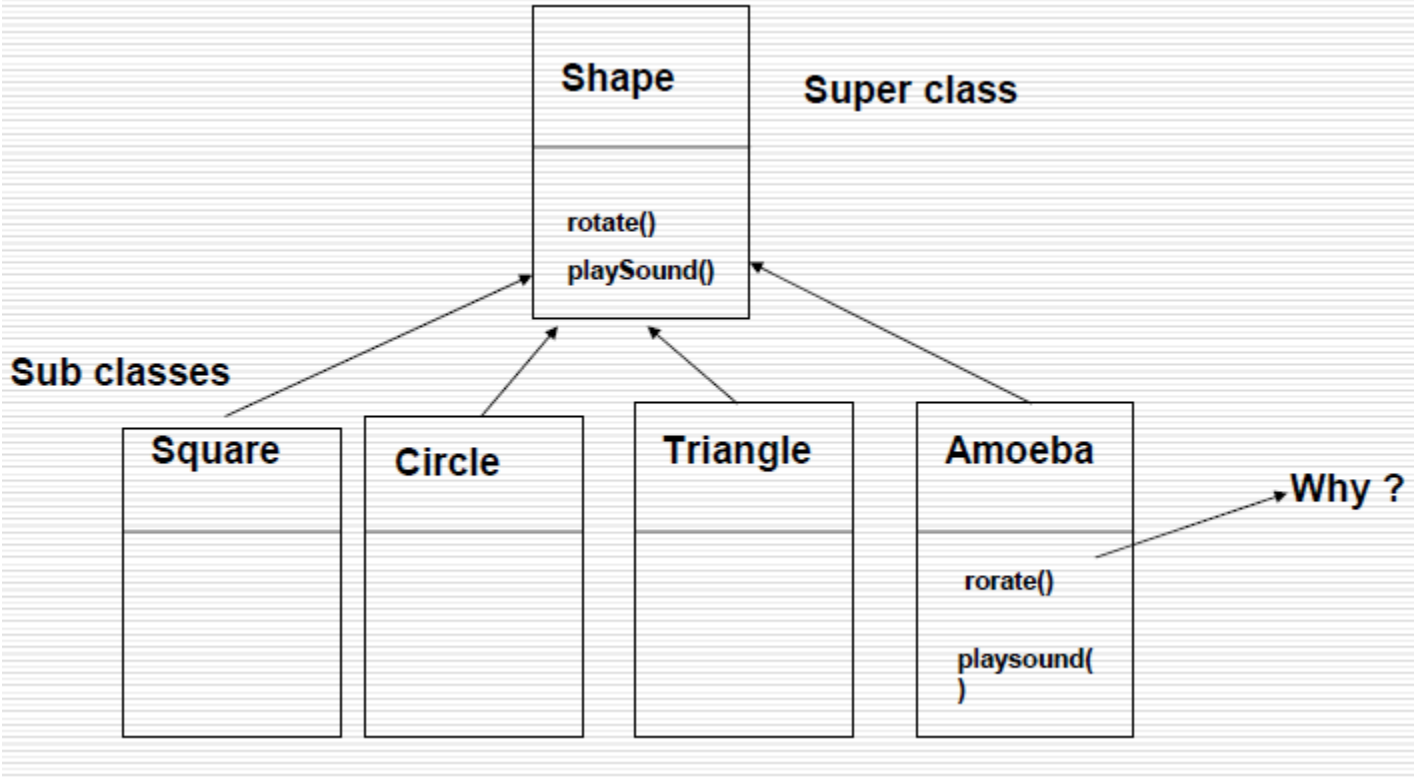
Inheritance...

Another Example



- See , common features
- Create a generic class which holds all common features
- All other classes will inherit it and use it.

Inheritance….



Inheritance…..

- Inheritance is Generalization to Specialization
- Inheritance is nothing but creating new class based on existing class.
- Is-A relationship
- "*extends*" keyword is used for inheritance in java.
- private members are not inherited.
- constructors are not inherited.

Keyword : **super**

- If your method overrides one of its super class's methods, you can invoke the overridden method through the use of the keyword ***super***

- Can Invoke super class constructor as :

  super()

  Invocation of a super class constructor must be the first line in the subclass constructor.

Method Overriding

- Method Overriding is achieved when a subclass overrides non-static methods defined in the super class, following which the new method implementation in the subclass that is executed.
- The new method definition must have the same method signature (i.e., method name and parameters) and return type.
- The new method definition cannot narrow the accessibility of the method, but it can widen it

Abstract Class

- Facility to collect generic / common features into a single super class.
- One or more methods are **declared** but **not defined**.
- Advantages :
  - Useful to achieve polymorphism.
  - Helps to create base classes that are generic and implementation is not available.

Abstract Class…

- Java allows creation of a super class that declares a method but does not supply any implementation. Such a method is called **abstract method**
- Any class with zero or more abstract method(s) is called an **abstract class**
- You cannot create instances of abstract classes
- Abstract classes may have data attributes, concrete methods, and constructors
- It is a good practice to make constructors protected

Abstract Class…

- If a sub-class does not provide implementation for abstract methods, it must be declared abstract
- A failure to do so will result in compiler errors

Interface

- An interface is a contract between client code and the class that implements that interface
- In Java it is identified by the "**interface**" keyword
- A class can implement many, unrelated interfaces
- Many unrelated classes can implement the same interface

Interface…

- Concrete classes that implement an interface must implement every abstract method in the interface
- The interface name can be used as a type of reference

variable. The usual dynamic binding takes place

- You can use the **instanceof** operator to determine if an object's class implements an interface

Interface…

The valid combinations are

- class extends class
- class implements interface
- interface extends interface

All other combinations are invalid.

Abstract Classes & Interfaces

- Abstract classes are used only when there is a "is-a" type of relationship between the classes.
- You cannot extend more than one abstract class.
- Abstract class can contain abstract as well as implemented methods

- Interfaces can be implemented by classes that are not related to one another.
- You can implement more than one interface.
- Interfaces contain abstract methods. Can contain non overridable methods like – default (from java 8), static(from java 8), private(from java 9)

Final Classes

- Java allows you to apply **final** to classes
- Final classes cannot be sub-classed
- An ideal usage scenario is where you don't want to allow your class to be extended for security reasons
- E.g. String class is declared final

Final methods

- Java allows you to apply **final** to methods
- Final methods cannot be overridden
- An ideal usage scenario is where you don't want to allow your method implementations to change to ensure a consistent state of your object
- Methods declared final are sometimes used for optimizations
- When used in situations where class hierarchies exist, using final with methods leads to significant performance enhancements
- Methods marked static or private are automatically marked final because dynamic binding cannot be applied in either case

Final variables

- Variables marked final are treated as constants
- Any attempt to change a final variable results in a compiler error
- If you mark a reference type variable final, you cannot use it to refer to any other object. You can however change the object's contents


QA