UNIT-II-PART-IV

Class path

How to set path in Java

The path is required to be set for using tools such as javac, java, etc.

If you are saving the Java source file inside the JDK/bin directory, the path is not required to be set because all the tools will be available in the current directory.

However, if you have your Java file outside the JDK/bin folder, it is necessary to set the path of JDK.

There are two ways to set the path in Java:

1. Temporary
2. Permanent

1) How to set the Temporary Path of JDK in Windows

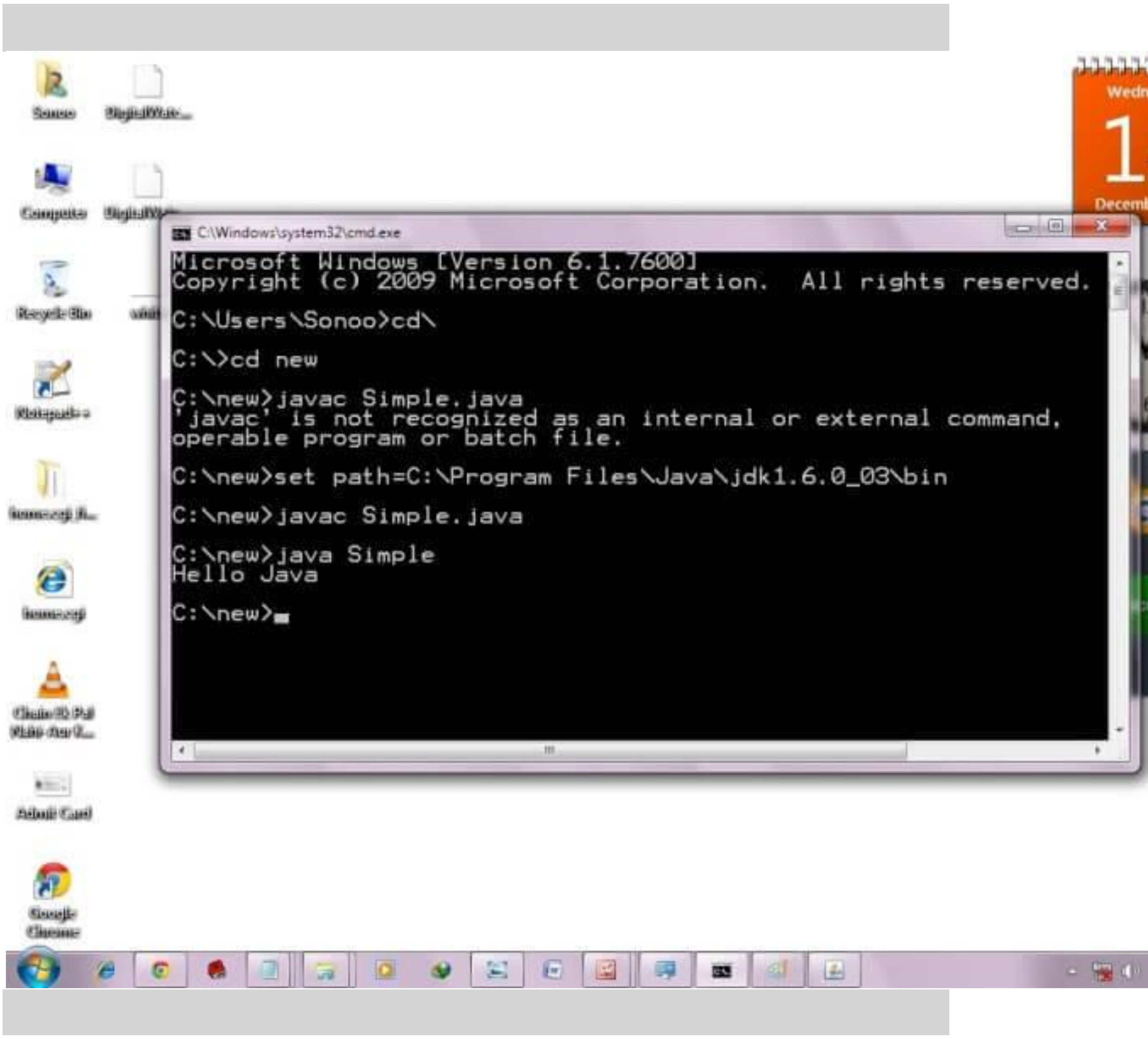
To set the temporary path of JDK, you need to follow the following steps:

- Open the command prompt
- Copy the path of the JDK/bin directory
- Write in command prompt: set path=copied_path

For Example:

set path=C:\Program Files\Java\jdk1.6.0_23\bin

Let's see it in the figure given below:



```
Microsoft Windows [Version 6.1.7600]
Copyright (c) 2009 Microsoft Corporation.  All rights reserved.

C:\Users\Sonoo>cd\

C:\>cd new

C:\new>javac Simple.java
'javac' is not recognized as an internal or external command,
operable program or batch file.

C:\new>set path=C:\Program Files\Java\jdk1.6.0_03\bin

C:\new>javac Simple.java

C:\new>java Simple
Hello Java

C:\new>
```

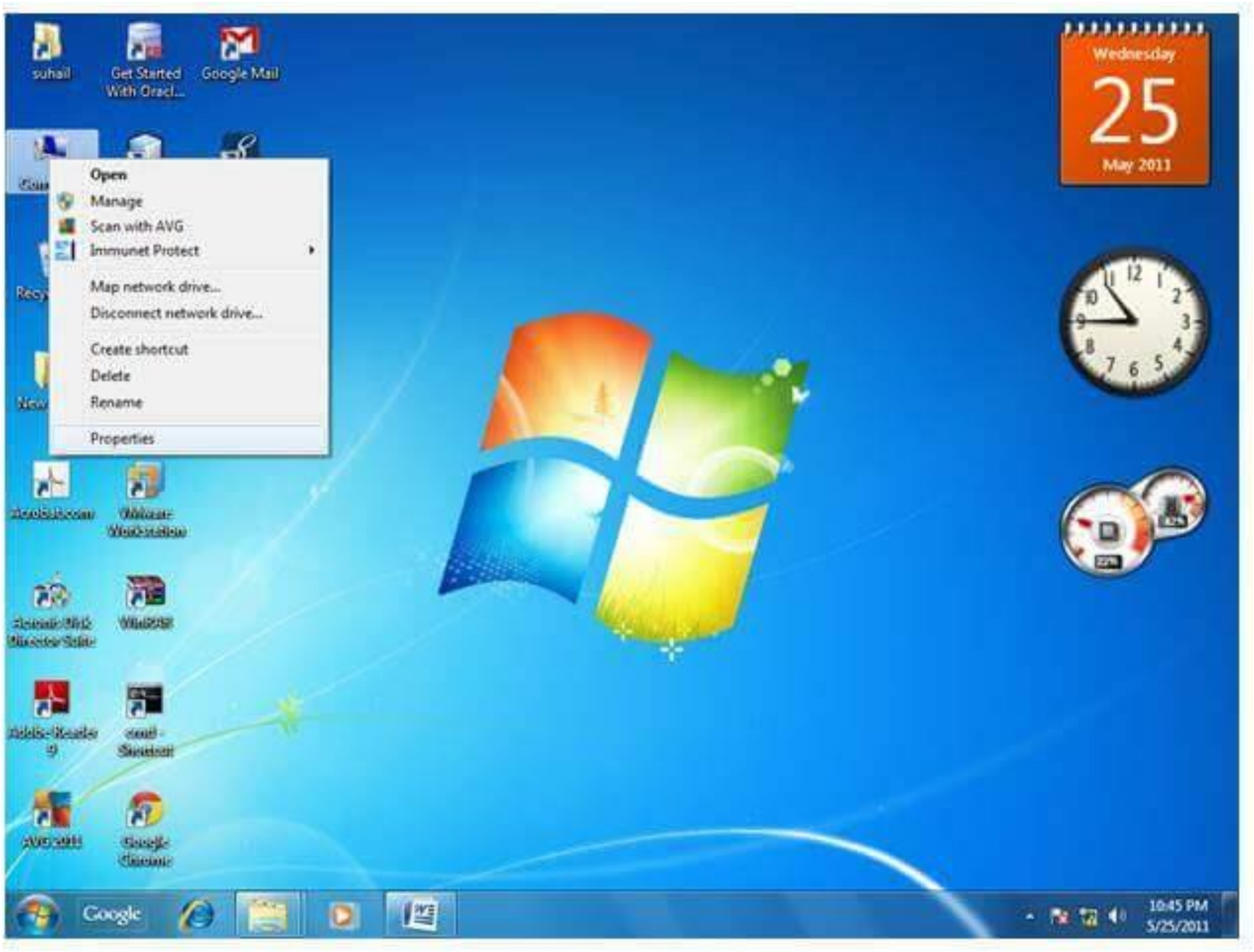
2) How to set Permanent Path of JDK in Windows

For setting the permanent path of JDK, you need to follow these steps:

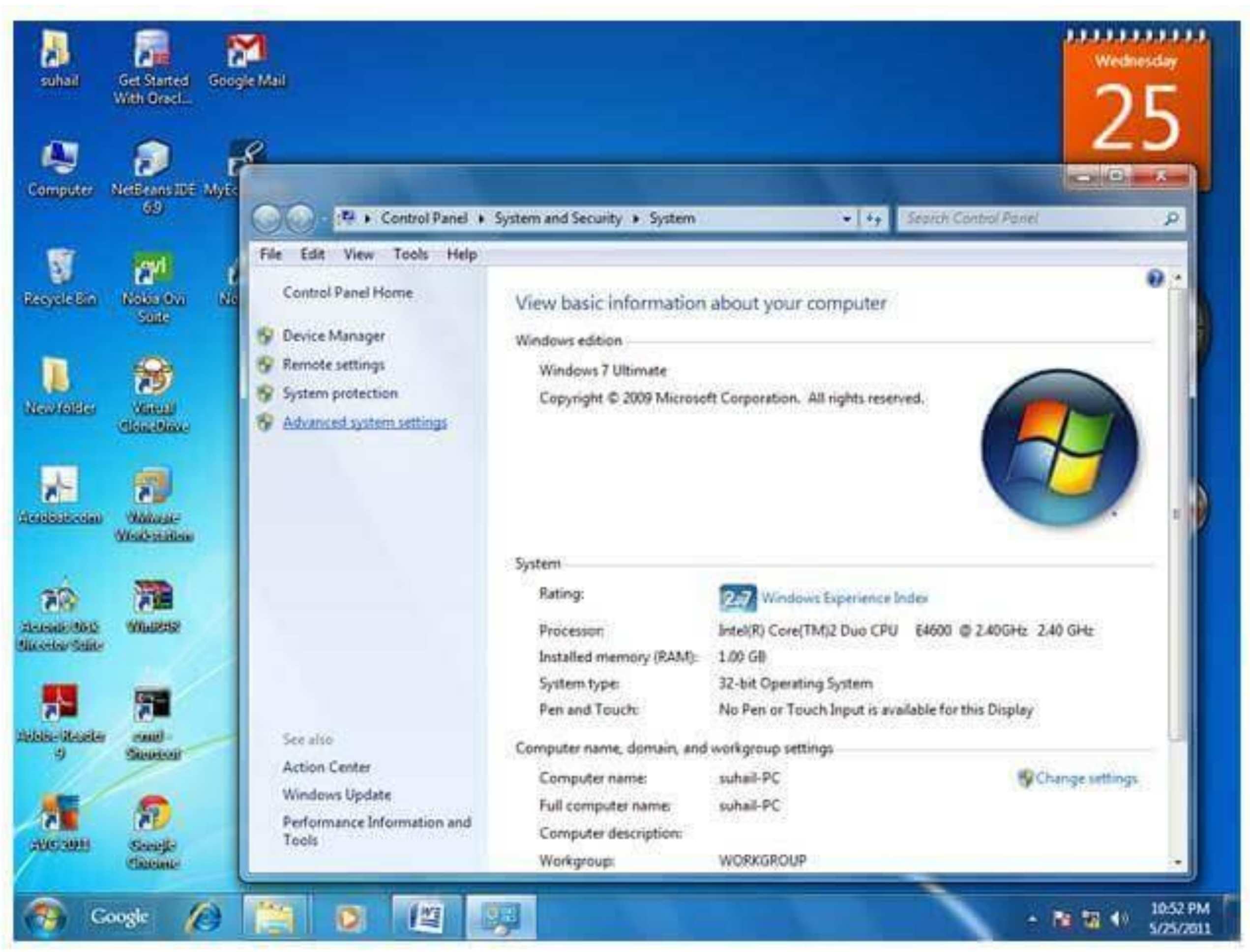
- Go to MyComputer properties -> advanced tab -> environment variables -> new tab of user variable -> write path in variable name -> write path of bin folder in variable value -> ok -> ok -> ok

For Example:

1) Go to MyComputer properties



2) Click on the advanced tab



View basic information about your computer

Windows edition

Windows 7 Ultimate

Copyright © 2009 Microsoft Corporation. All rights reserved.

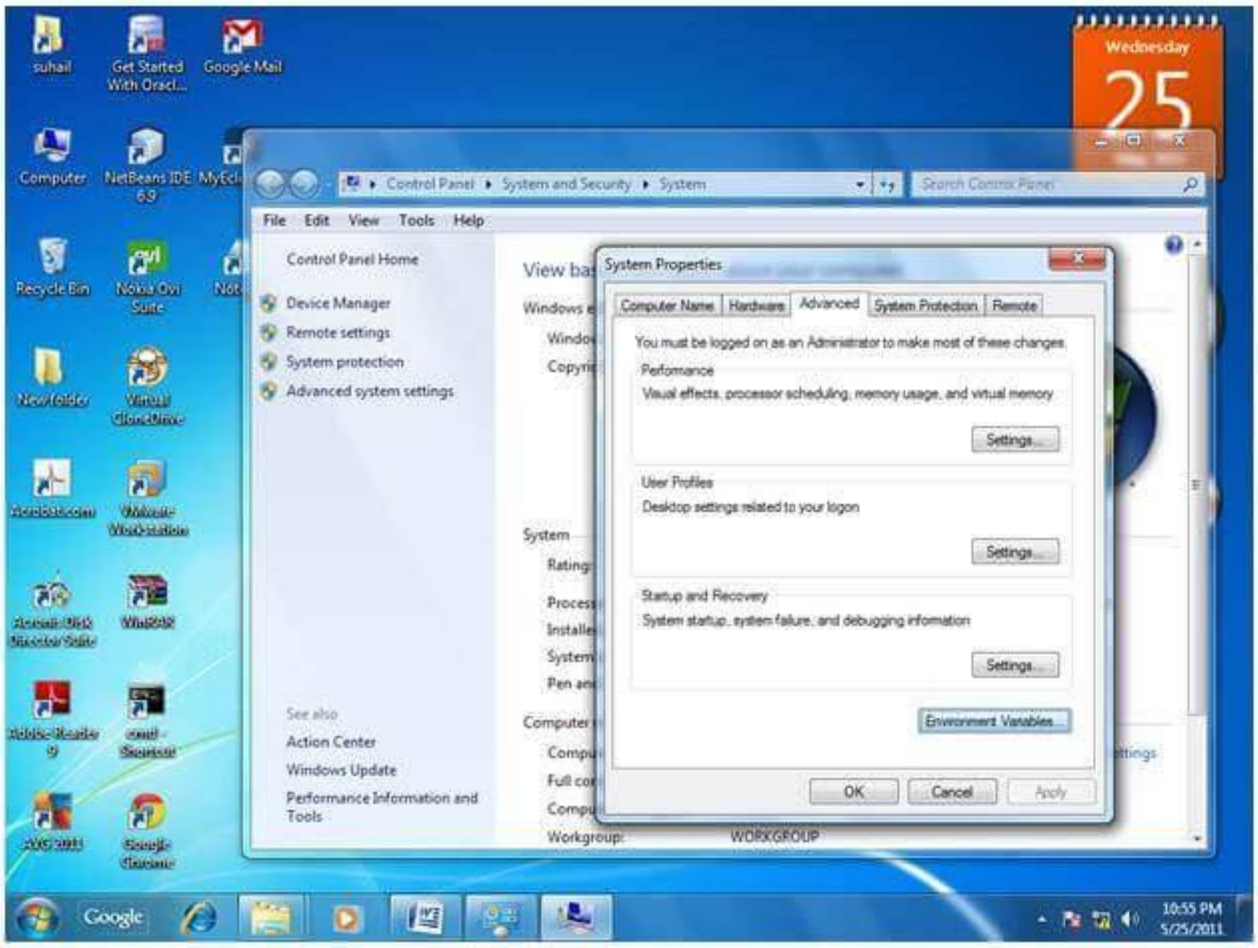
System

Rating:	2.7 Windows Experience Index
Processor:	Intel(R) Core(TM)2 Duo CPU E4600 @ 2.40GHz 2.40 GHz
Installed memory (RAM):	1.00 GB
System type:	32-bit Operating System
Pen and Touch:	No Pen or Touch Input is available for this Display

Computer name, domain, and workgroup settings

Computer name:	suhail-PC
Full computer name:	suhail-PC
Computer description:	
Workgroup:	WORKGROUP

3) Click on environment variables



You must be logged on as an Administrator to make most of these changes.

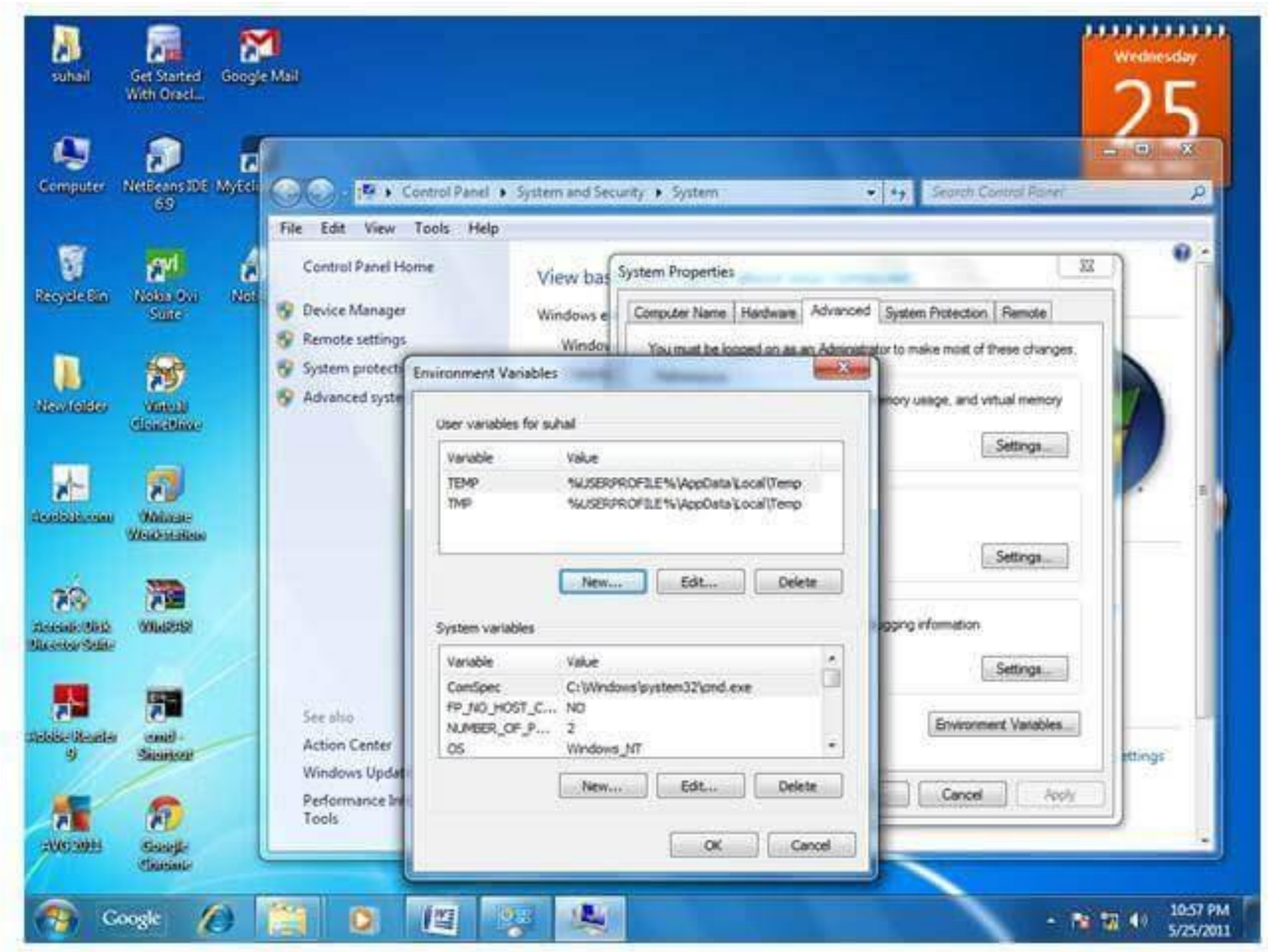
Performance
Visual effects, processor scheduling, memory usage, and virtual memory

User Profiles
Desktop settings related to your logon

Startup and Recovery
System startup, system failure, and debugging information

Workgroup: WORKGROUP

4) Click on the new tab of user variables



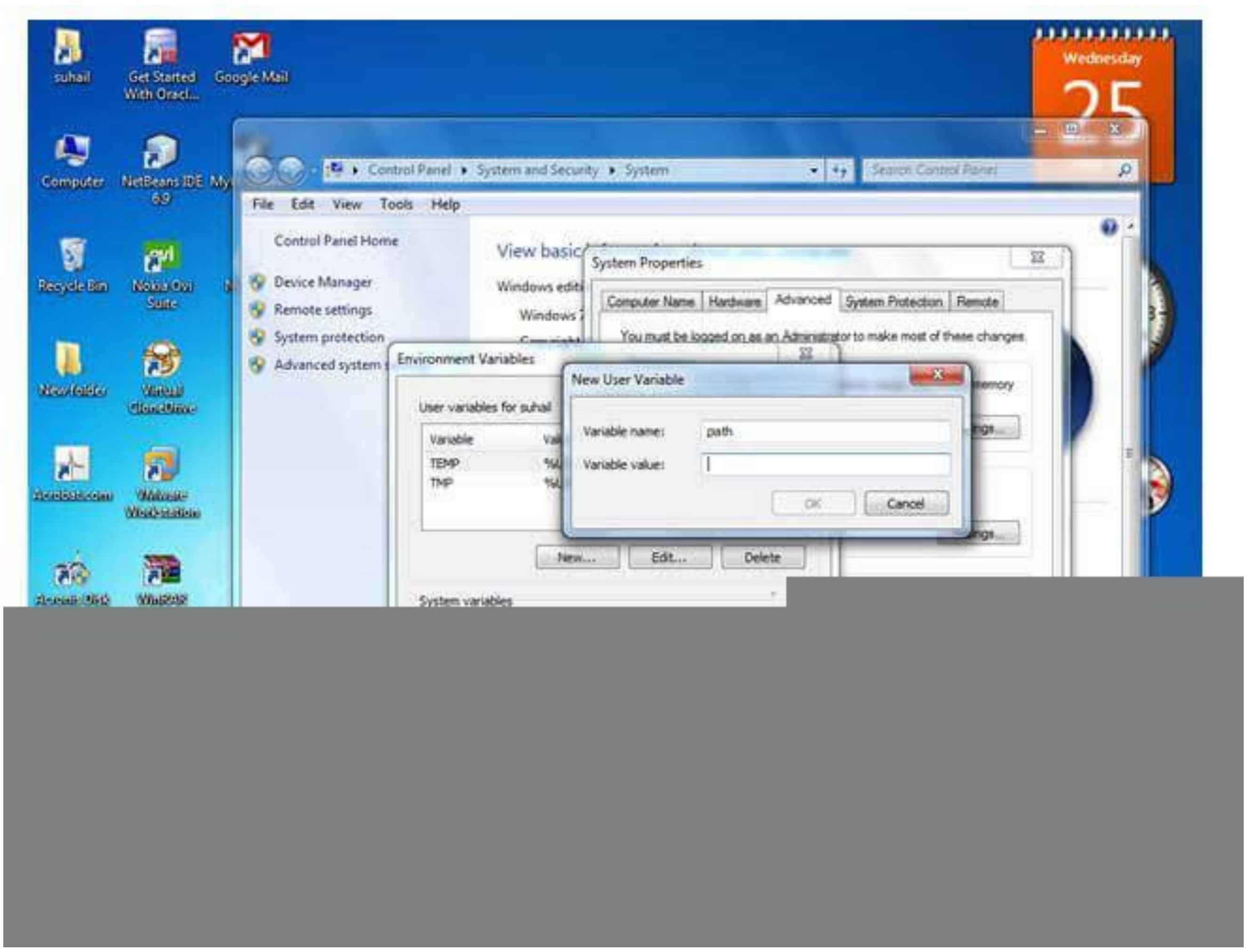
User variables for suhail

Variable	Value
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2
OS	Windows_NT

5) Write the path in the variable name



New User Variable

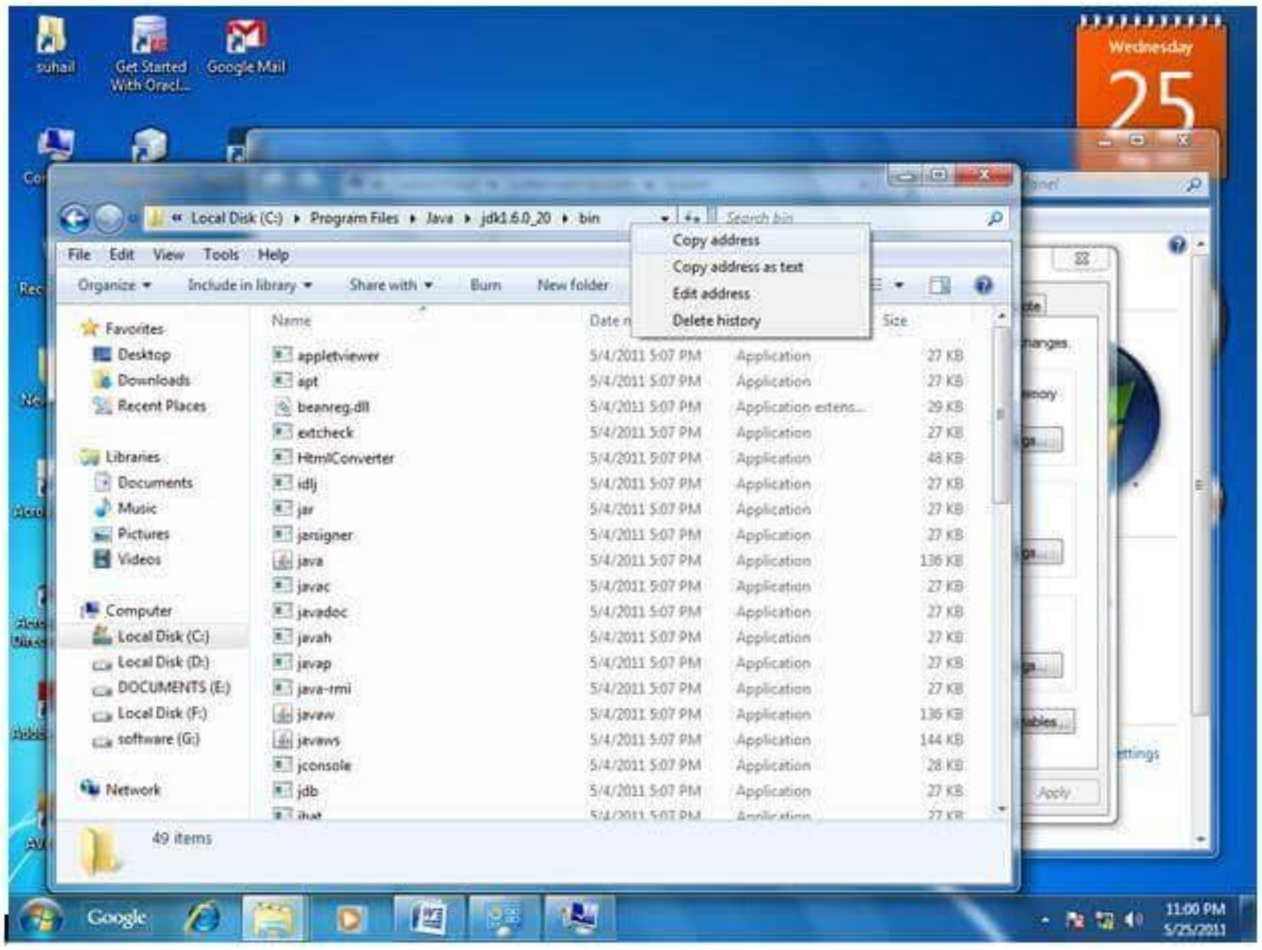
Variable name: path

Variable value:

You must be logged on as an Administrator to make most of these changes.

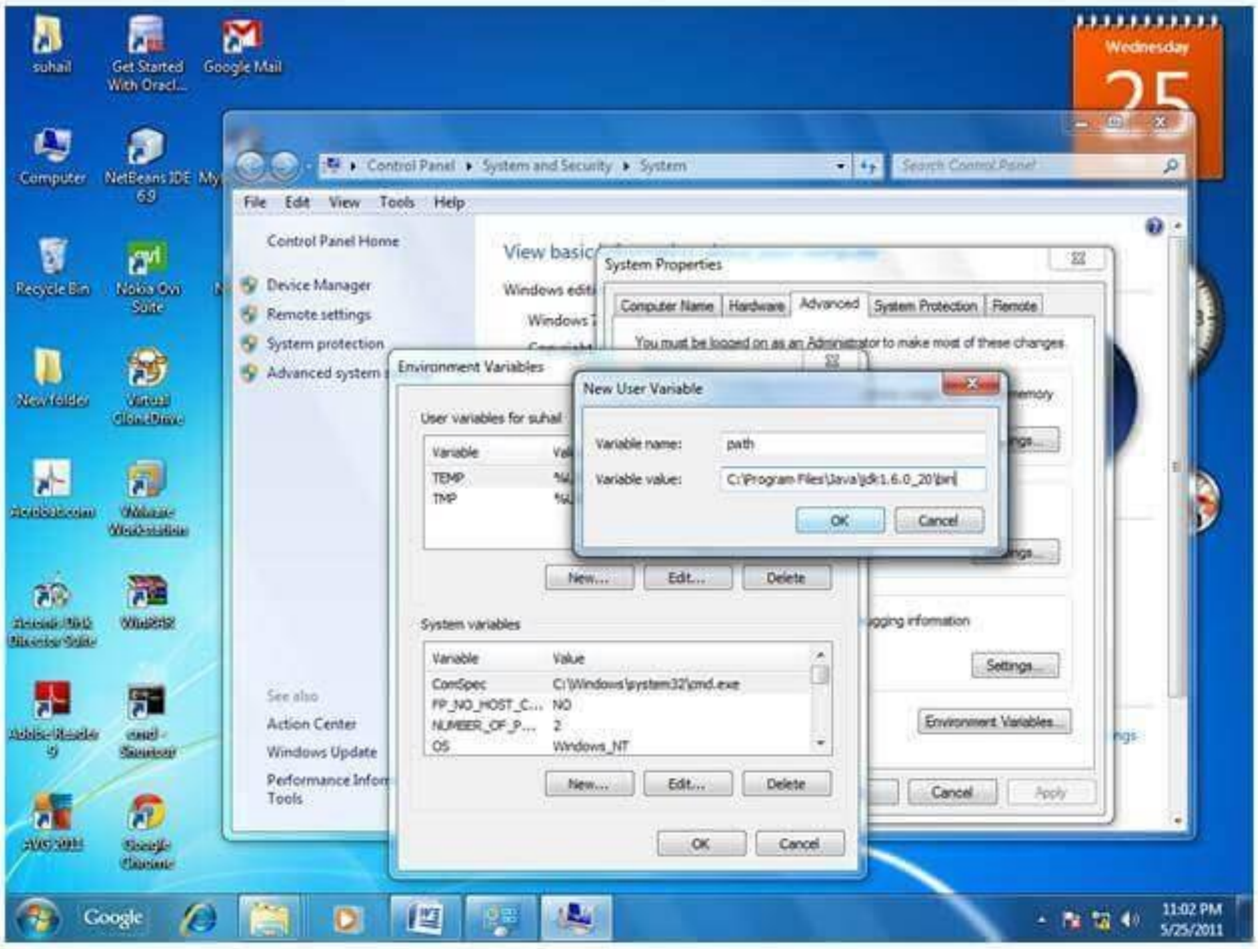
Variable	Value
TEMP	%U
TMP	%U

6) Copy the path of bin folder


Local Disk (C:) ▸ Program Files ▸ Java ▸ jdk1.6.0_20 ▸ bin
Copy address
Copy address as text
Edit address
Delete history
File Edit View Tools Help
Organize Include in library Share with Burn New folder
49 items

Name	Date modified	Type	Size
appletviewer	5/4/2011 5:07 PM	Application	27 KB
apt	5/4/2011 5:07 PM	Application	27 KB
beanreg.dll	5/4/2011 5:07 PM	Application extens...	29 KB
extcheck	5/4/2011 5:07 PM	Application	27 KB
HtmlConverter	5/4/2011 5:07 PM	Application	48 KB
idlj	5/4/2011 5:07 PM	Application	27 KB
jar	5/4/2011 5:07 PM	Application	27 KB
jarsigner	5/4/2011 5:07 PM	Application	27 KB
java	5/4/2011 5:07 PM	Application	136 KB
javac	5/4/2011 5:07 PM	Application	27 KB
javadoc	5/4/2011 5:07 PM	Application	27 KB
javah	5/4/2011 5:07 PM	Application	27 KB
javap	5/4/2011 5:07 PM	Application	27 KB
java-rmi	5/4/2011 5:07 PM	Application	27 KB
javaw	5/4/2011 5:07 PM	Application	136 KB
javaws	5/4/2011 5:07 PM	Application	144 KB
jconsole	5/4/2011 5:07 PM	Application	28 KB
jdb	5/4/2011 5:07 PM	Application	27 KB

7) Paste path of bin folder in the variable value



New User Variable

Variable name: path

Variable value: C:\Program Files\Java\jdk1.6.0_20\bin

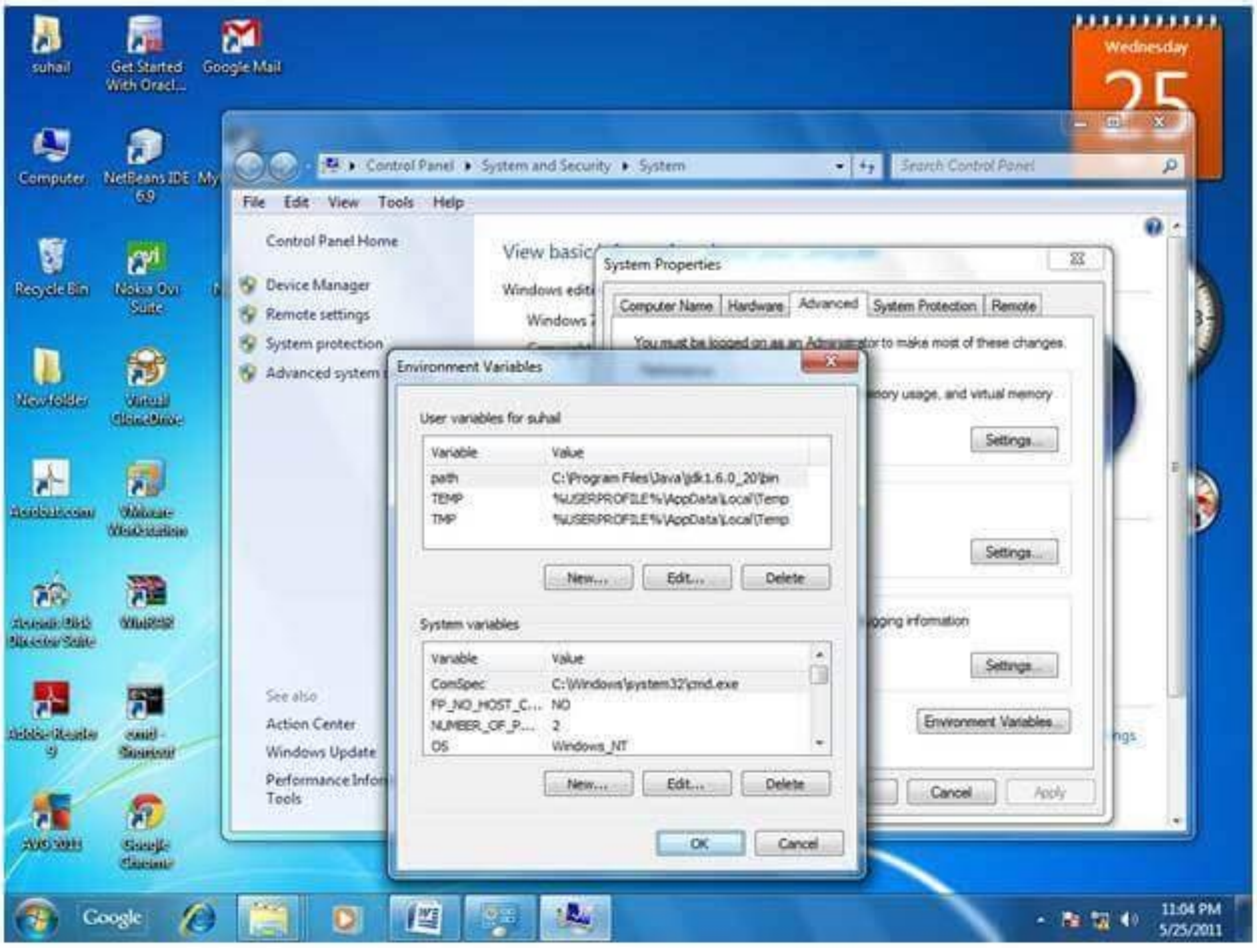
User variables for suhail

Variable	Value
TEMP	%U
TMP	%U

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2
OS	Windows_NT

8) Click on ok button



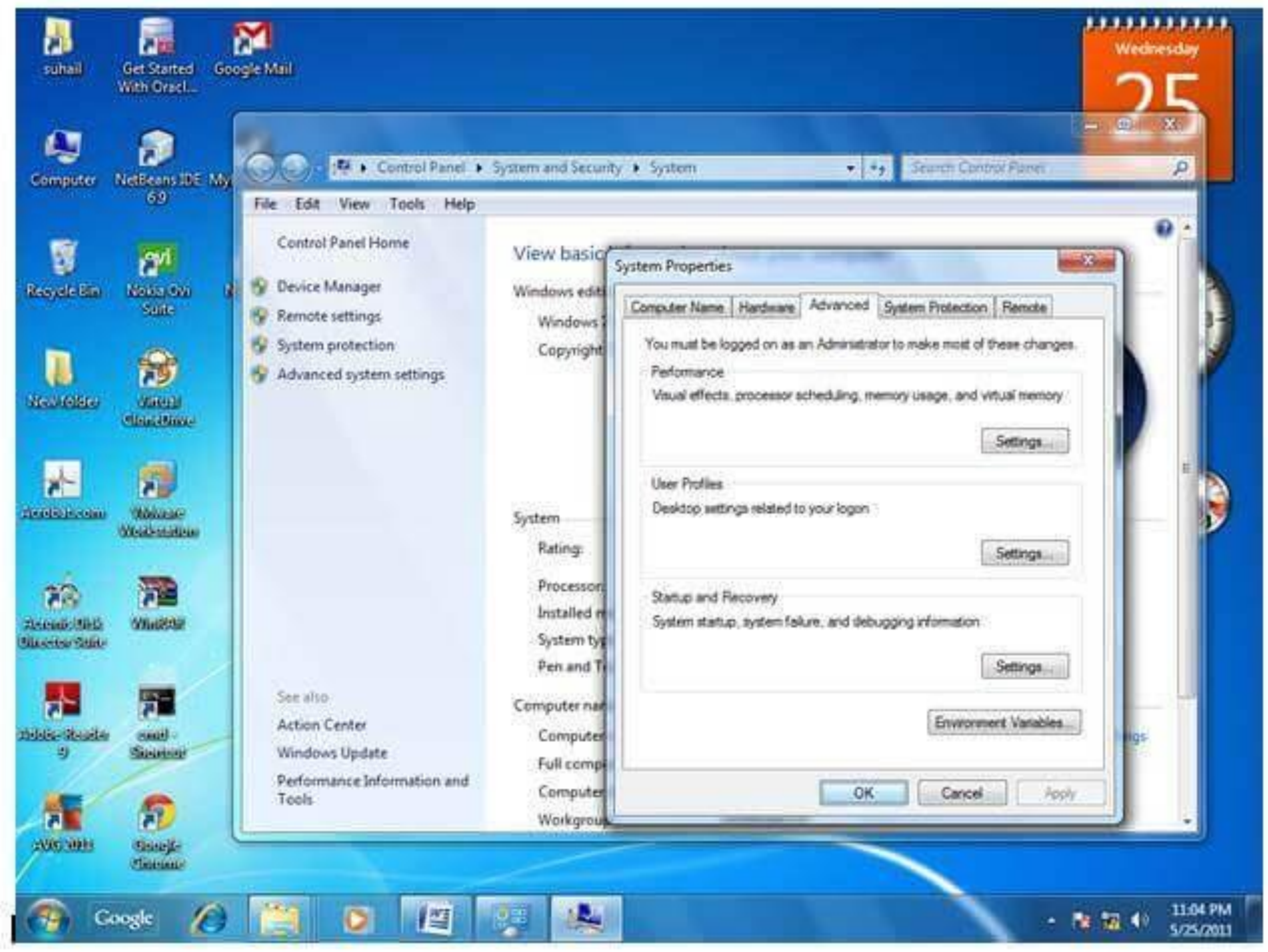
User variables for suhail

Variable	Value
path	C:\Program Files\Java\jdk1.6.0_20\bin
TEMP	%USERPROFILE%\AppData\Local\Temp
TMP	%USERPROFILE%\AppData\Local\Temp

System variables

Variable	Value
ComSpec	C:\Windows\system32\cmd.exe
FP_NO_HOST_C...	NO
NUMBER_OF_P...	2
OS	Windows_NT

9) Click on ok button



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 logon
Settings...

Startup and Recovery
System startup, system failure, and debugging information
Settings...

Environment Variables...

OK Cancel Apply

Now your permanent path is set. You can now execute any program of java from any drive.

Access specifiers in java

Access Modifiers in Java

There are two types of modifiers in Java: **access modifiers** and **non-access modifiers**.

The access modifiers in Java specifies the accessibility or scope of a field, method, constructor, or class.

We can change the access level of fields, constructors, methods, and class by applying the access modifier on it.

here are four types of Java access modifiers:

1. **Private**: The access level of a private modifier is only within the class. It cannot be accessed from outside the class.
2. **Default**: The access level of a default modifier is only within the package. It cannot be accessed from outside the package. If you do not specify any access level, it will be the default.
3. **Protected**: The access level of a protected modifier is within the package and outside the package through child class. If you do not make the child class, it cannot be accessed from outside the package.
4. **Public**: The access level of a public modifier is everywhere. It can be accessed from within the class, outside the class, within the package and outside the package.

Understanding Java Access Modifiers

Let's understand the access modifiers in Java by a simple table.

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	Y	N	N	N
Default	Y	Y	N	N
Protected	Y	Y	Y	N
Public	Y	Y	Y	Y

1) Private

The private access modifier is accessible only within the class.

Simple example of private access modifier

In this example, we have created two classes A and Simple. A class contains private data member and private method. We are accessing these private members from outside the class, so there is a compile-time error.

```
class A{
private int data=40;
private void msg(){System.out.println("Hello java");}
}

public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

2) Default

If you don't use any modifier, it is treated as **default** by default.

The default modifier is accessible only within package. It cannot be accessed from outside the package.

It provides more accessibility than private. But, it is more restrictive than protected, and public.

Example of default access modifier

In this example, we have created two packages pack and mypack. We are accessing the A class from outside its package, since A class is not public, so it cannot be accessed from outside the package.

```

//save by A.java
package pack;
class A
{
  void msg()
{
System.out.println("Hello");}
}
```

```
//save by B.java
package mypack;
```

```
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error
  }
}
```

In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

3) Protected

The **protected access modifier** is accessible within package and outside the package but through inheritance only.

The protected access modifier can be applied on the data member, method and constructor.

It can't be applied on the class.

It provides more accessibility than the default modifer.

Example of protected access modifier

In this example, we have created the two packages pack and mypack.

The A class of pack package is public, so can be accessed from outside the package.

But msg method of this package is declared as protected, so it can be accessed from outside the class only through inheritance.

```
//save by A.java
package pack;
public class A{
protected void msg(){System.out.println("Hello");}

}
```

```
//save by B.java
package mypack;
import pack.*;

class B extends A{
  public static void main(String args[]){
   B obj = new B();
   obj.msg();
  }
}
```

```
Output:Hello
```

4) Public

The **public access modifier** is accessible everywhere. It has the widest scope among all other modifiers.

Example of public access modifier

```
//save by A.java

package pack;
public class A
{
public void msg(){System.out.println("Hello");}
}
```

```
//save by B.java

package mypack;
import pack.*;

class B
{
  public static void main(String args[])
{
   A obj = new A();
   obj.msg();
  }
13.         }
```

Output:Hello

Difference between JDK, JRE, and JVM

JVM

JVM (Java Virtual Machine) is an abstract machine.

It is called a virtual machine because it doesn't physically exist.

It is a specification that provides a runtime environment in which Java bytecode can be executed.

It can also run those programs which are written in other languages and compiled to Java bytecode.

JVMs are available for many hardware and software platforms. JVM, JRE, and JDK are platform dependent because the configuration of each OS is different from each other.

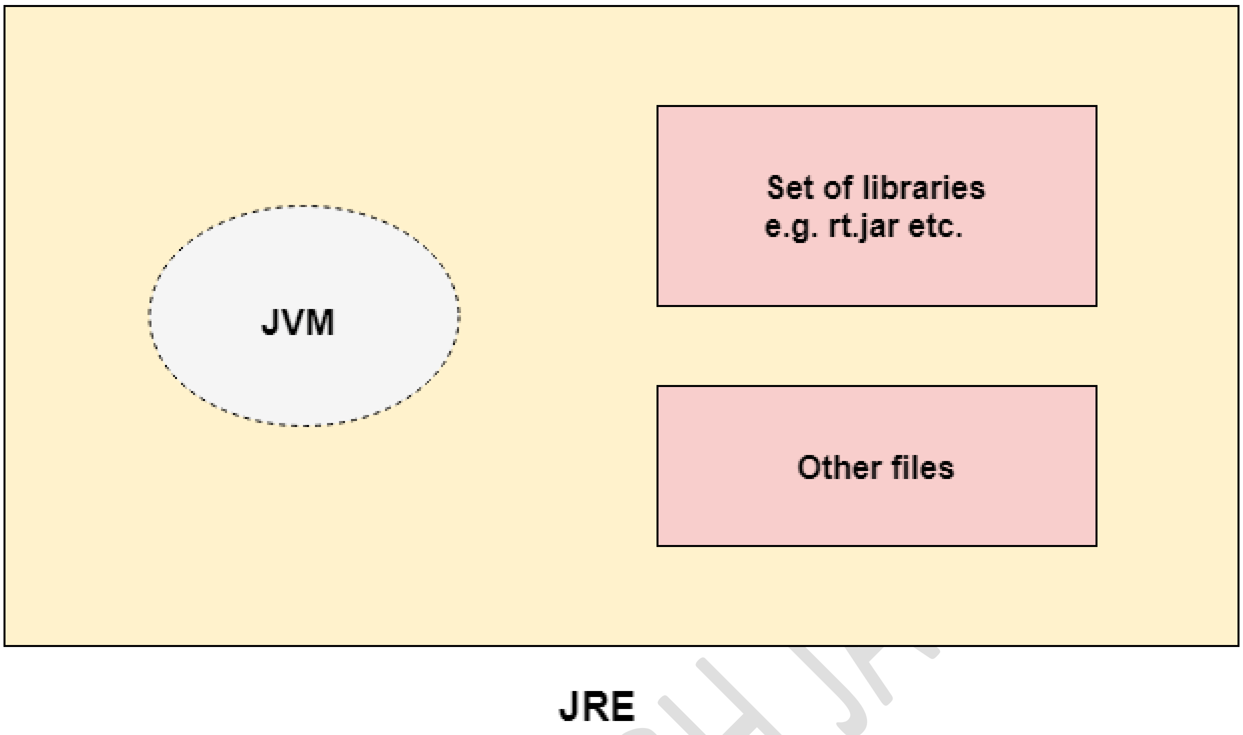
The JVM performs the following main tasks:

- Loads code
- Verifies code
- Executes code
- Provides runtime environment

JRE

JRE is an acronym for Java Runtime Environment. It is also written as Java RTE. The Java Runtime Environment is a set of software tools which are used for developing Java applications. It is used to provide the runtime environment. It is the implementation of JVM. It physically exists. It contains a set of libraries + other files that JVM uses at runtime.

The implementation of JVM is also actively released by other companies besides Sun Micro Systems.



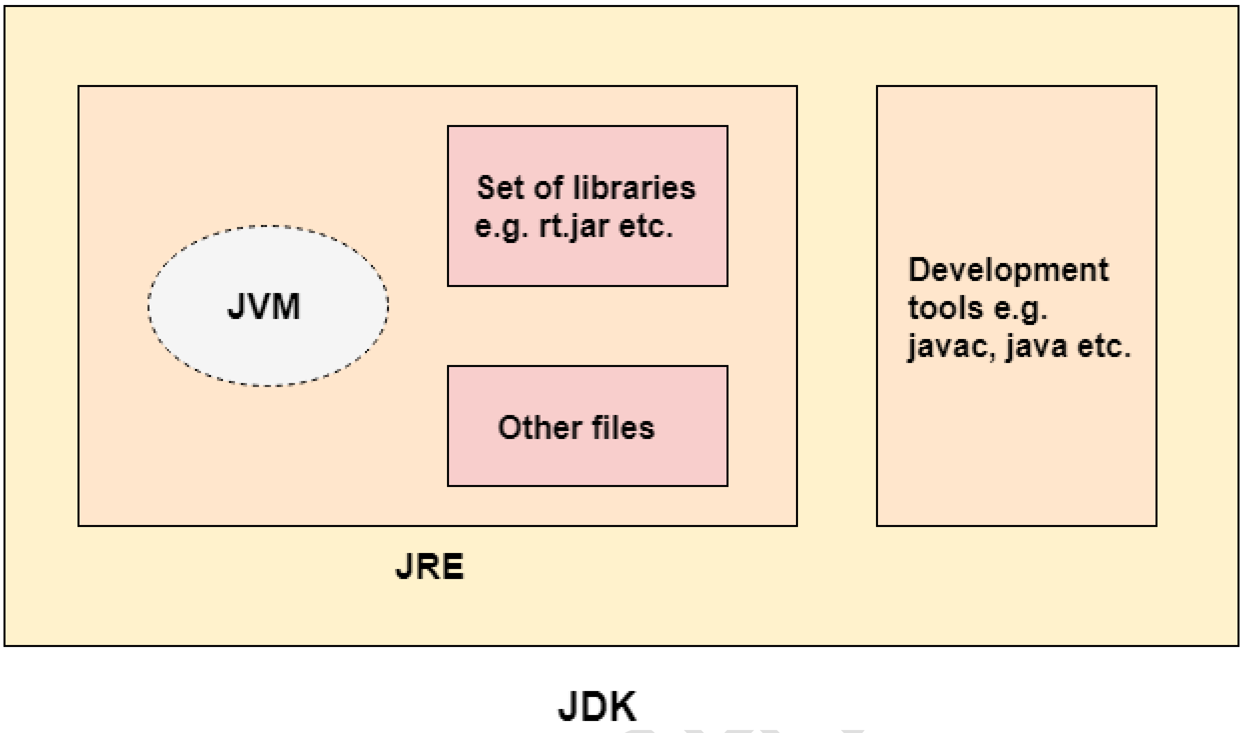
JDK

JDK is an acronym for Java Development Kit. The Java Development Kit (JDK) is a software development environment which is used to develop Java applications and applets. It physically exists. It contains JRE + development tools.

JDK is an implementation of any one of the below given Java Platforms released by Oracle Corporation:

- Standard Edition Java Platform
- Enterprise Edition Java Platform
- Micro Edition Java Platform

The JDK contains a private Java Virtual Machine (JVM) and a few other resources such as an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc), etc. to complete the development of a Java Application.


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