Chapter - 6

Packages

Packages

Introduction

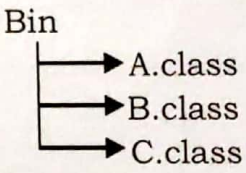
When a java program is compiled we get the .class files for each class present in the program. For example consider the following program

Bin>edit A.java

```
class A
{
        main(..)
        {
                ...
        }
}
class B { ... }
class C { ... }
```

Bin> javac A.java

The java compiler compiles the A.java program and generates three .class files. They are A.class, B.class and C.class. These .class files are stored in the working directory **Bin.**



Now if we execute the program using the A.class file then the program executes successfully.

Bin> java A

program executes successfully.

We write another program in the same Bin directory.

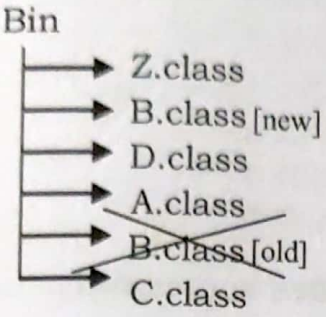
Bin>edit Z.java

```
class Z
{
        main(..)
        {
                ...
        }
}
```

```
class B { ... }
class D { ... }
```

Bin> javac Z.java

The java compiler compiles the **Z.java** program and generates three **.class** files. They are **Z.class**, **B.class** and **D.class**. These **.class** files are stored in the working directory **Bin.**



Bin directory already consists of **B.class** file therefore old file is deleted and the latest **B.class** file generated from **Z.java** is stored. Therefore the **Z.java** program executes successfully where as **A.java** program does not execute because the **B.class** file belong to **A.java** was deleted.

Bin> java Z

No. error, executes successfully

Bin>java A

Error: **B.class** of **A.java** was deleted.

As explained above when different java programs are compiled their may be a chance of two or more class files have same names and this is known as ***name space collision.*** In this process the older .class files are deleted and new files are stored, as a result the old java programs when re-executed they does not execute. This is the problem that arises in java.

To overcome the above problem packages concept can be used in java. For example consider the following programs.

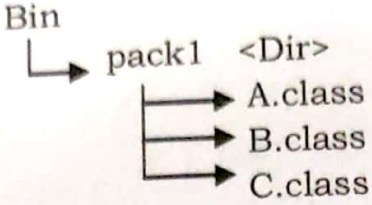
Bin>edit A.java

```
package pack1;
class A
{
        main(..)
        {
                ...
        }
}
```

```
class B { ... }
class C { ... }
```

Bin> javac –d . A.java

The java compiler compiles the **A.java** program and generates three .class files (**A.class**, **B.class** and **C.class**). These .class files are placed in **pack1** directory and this directory is stored in the working directory **Bin** as shown below.



Now, if we execute the program it executes successfully.

Bin> java pack1.A

No error, Executes successfully.

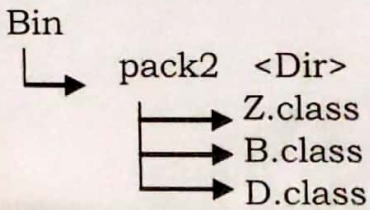
We write another program in the same Bin directory.

Bin>edit Z.java

```
package pack2;
class Z
{
        main(..)
        {
                ...
        }
}
class B { ... }
class D { ... }
```

Bin> javac –d . Z.java

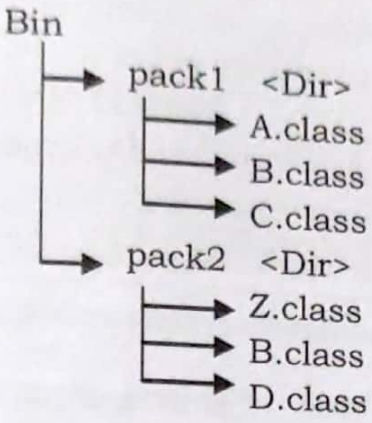
The java compiler compiles the **Z.java** program and generates three .class files (**Z.class**, **B.class** and **D.class**). These .class files are placed in **pack2** directory and this directory is stored in the working directory **Bin** as shown below.



Now, if we execute the program it executes successfully.

Bin> java pack2.Z

No. error. Executes successfully.



As B.class files are stored in different directories (packages) as shown above, therefore their will be no name space collision.

Definition: A package is a container of classes.

Advantages:

1. Packages eliminate name space collision.
2. Two or more .class files with same name can be stored in different packages.
3. The classes can be classified and can be organized in hierarchical order.
4. The classes in packages can be reused or shared in other java programs.

Syntax:

```
package packagename1[.packagename2[.pacakgename3.[...]]];
```

Package programs should be complied with the following syntax.

javac –d <path> pack_prog.java

-d parameter creates a directory on the name of package present in the pack_prog.java.

<path> specifies where the package directory should be placed.

Package programs should be executed using the following syntax.

java pack_name.classname

Note: Package declaration statement should be the first statement in the java program.

Example:

Bin>edit p1.java

```
package pack1;
public class p1
{
    public static void main(String argv[])
    {
    System.out.print("\n This is sample package program.");
    }
}
```

Bin>javac -d . p1.java

pack1 <Dir>
p1.class

Bin>java pack1.p1

This is sample package program.

Explanation:

When the above program is compiled in **Bin** directory as

Bin>javac –d . p1.java

The java creates a new directory on the name of packagename **pack1** given in the program and the .class files(**p1.class**) are stored in the **pack1** directory. This **pack1** directory will be placed in **Bin** directory that is specified using **.** (current directory).

The above package program can be executed in the **Bin** directory as

Bin> java pack1.p1

The jvm executes the **p1.class** file present in **pack1** directory and the output we get as

This is sample package program.

Note:

1. The classes in packages should be public otherwise the classes cannot be accessed outside the package.
2. Instead of using **.(dot)** in **javac** statement above, we can also specify the path of working directory or any other directory where we want to store package directory. Therefore the above javac statement can also be written as:

Bin> javac –d c:\program files\java\jdk1.5.0_05\Bin p1.java

3. The package statement should be the first statement in the java program.
4. If the package directory already exist then the same directory is used for storing the .class files without creating the new directory.

Importing package classes

The classes present in packages can be used in other programs by importing the classes of packages in other java programs.

Syntax:

```
import packagename.classsname;
        or
import packagename.*;
```

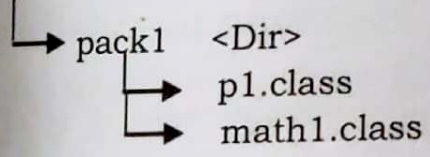
The first syntax imports only one class where as second syntax imports all classes present in the package.

Example:

Bin> edit math1.java

```
package pack1;
public class math1
{
      public void sum(int x,int y)
      {
            System.out.print("\n sum="+(x+y));
      }
      public void square(int x)
      {
            System.out.print("\nsquare="+(x*x));
      }
}
```

Bin>javac -d . math1.java



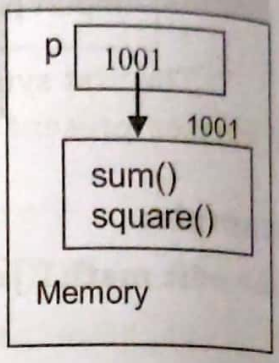
Explanation:

When the above program is compiled the jvm generates math1.class which is placed in pack1 directory of Bin

Don't execute the math1.class file because it does not contain main(). This class can be used for importing into other programs.

Bin>edit m1.java

```
import pack1.math1;
class m1
{
        public static void main(String argv[])
        {
                pack1.math1 p=new pack1.math1();
                p.sum(5,6);
                p.square(5);
        }
}
```



Bin> javac m1.java
Bin> java m1

```
sum=11
square=25
```

Explanation:

In the above program, the statement **import pack1.math1;** imports the **math1** class into the program. In the main(), an object **p** is instantiated to **math1** class present in **pack1** and using this object **sum()** and **square()** methods are executed and we get the output as shown above.

Example:

Bin> edit math2.java

```
package pack1;
public class math2
{
        public void sub(int x,int y)
        {
                System.out.print("\n sub="+(x-y));
        }
}
```

Bin>javac -d . math2.java

```
pack1    <Dir>
    p1.class
    math1.class
    math2.class
```

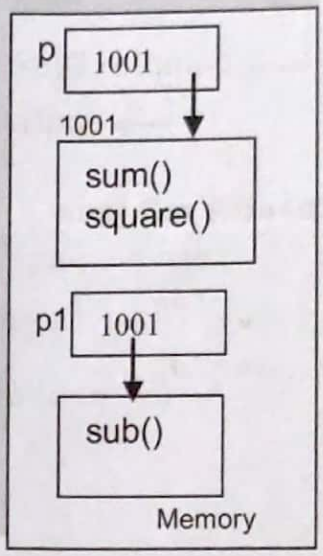
Explanation:

When the above program is compiled the jvm generates math2.class which is placed in pack1 directory of Bin

Don't execute the math2.class file because it does not contain main(). This class can be used for importing into other programs.

Bin>edit m2.java

```
import pack1.*;
class m2
{
    public static void main(String argv[])
    {
        pack1.math1 p=new pack1.math1();
        p.sum(5,6);
        p.square(5);
        pack1.math2 p1=new pack1.math2();
        p1.sub(12,7);
    }
}
```



Bin> javac m2.java
Bin> java m2

```
sum=11
square=25
sub=5
```

Explanation:

In the above program, the statement **import pack1.*;** imports all classes present in **pack1** package into the program. In the main(), an object **p** is instantiated to **math1** class present in **pack1** and using this object **sum()** and **square()** methods are executed and **p1** object is instantiated to **math2** class and **sub()** is called and we get the output as shown above.

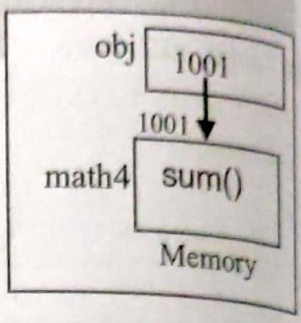
Inheritance in packages

We can also extend a class of a package from another package. For example consider the following program.

Bin>edit math3.java

```
package pack2;
import pack1.math1;
```

```
public class math3 extends pack1.math1
{
        public void mult(int x,int y)
        {
                System.out.print("\n mult = " +(x*y) );
        }
}
```

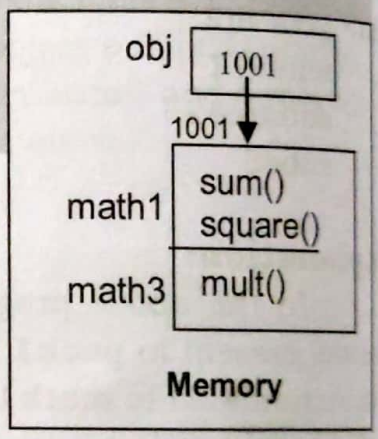


Bin>javac -d . math3.java

pack2 <Dir>
math3.class

Bin>edit m3.java

```
import pack2.math3;
class m3
{
        public static void main(String argv[])
        {
                pack2.math3 obj=new pack2.math3();
                obj.sum(5,6);
                obj.square(5);
                obj.mult(12,7);
        }
}
```



Bin> javac m3.java
Bin> java m3

sum=11
square=25
mult=84

Explanation:

In the above program the import statement imports the **math3.class** present in **pack2** package. In the main(), **obj** is an object created to **math3** class of **pack2** and **math3** is inherited from **math1** class of **pack1** therefore using the **obj,** methods of **math1** and **math3** classes are called and we get the output as shown above.

Nested Packages

Packages can be nested. A package created inside another package is known as nested packages. This nesting can go in any number.

Syntax :

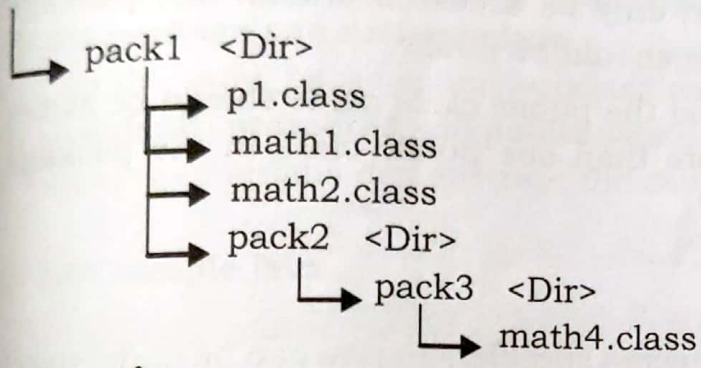
```
package packagename1.packagename2.pacakgename3...;
```

Example:

Bin>edit math4.java

```
package pack1.pack2.pack3;
public class math4
{
      public void sum(int x,int y)
      {
            System.out.print("\n sum=" + (x+y) );
      }
}
```

Bin>javac -d . math4.java



Explanation:

In the above program the statement **package pack1.pack2.pack3;** creates a nested packages in **pack1** as shown above and **math4.class** file will be stored in nested package **pack3**.

Bin>edit m4.java

```
import pack1.pack2.pack3.math4;
class m4
{
      public static void main(String argv[])
      {
            pack1.pack2.pack3.math4 obj=new
            pack1.pack2.pack3.math4();
            obj.sum(5,6);
      }
}
```

Bin> javac m4.java

Bin> java m4

sum=11

Explanation:

In the above program the import statement imports the **math4.class** present in **pack1.pack2.pack3** package. In the main(), **obj** is an object created to **math4** class of nested package **pack3** and using the **obj** the method sum() is called and we get the output as shown above.

Points to Remember:

1. If no package declaration is given then the working directory is considered as default package.
2. A package program can contain any number of classes without public modifier but these classes can not be accessed outside the package.
3. To access the class outside the package, it should be declared as public.
4. A package program should contain only one public class.
5. A public class of package can only be accessed outside the package therefore all classes in package should be public.
6. The package program name and the public class name should be same, therefore we cannot write more than one public class in the package program.

Access Specifiers in Java

The complete meanings of access specifiers in java can be understood in packages concepts. The access specifiers are **private**, **public**, **protected** and **default**(no modifier). The visibility of these access specifiers in different classes in packages are shown in the below table.

	Private	**Default** (No modifier)	**Protected**	**Public**
Same class	Yes	Yes	Yes	Yes
Same package non-subclass	No	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Different package non-subclass	No	No	No	Yes
Different package subclass	No	No	Yes	Yes

According to above table the **private**, **default**, **protected** and **public** members of class can be accessed inside the same class.

The **private** cannot be accessed in non-subclass of same package where as **default**, **protected** and **public** can be accessed.

The **private** cannot be accessed in subclass of same package where as **default**, **protected** and **public** can be accessed.

The **private**, **default** and **protected** cannot be accessed in non-subclass of different package where as **public** can be accessed

The **private** and default cannot be accessed in subclass of different package where as **protected** and **public** can be accessed.

Example programs on access specifiers

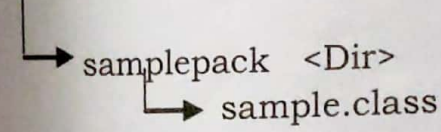
1. Same package and non-subclass

In the same package non-subclass can access all types of members such as **default**, **protected** and **public** where as **private** cannot be accessed. The following programs demonstrate the same.

Bin> edit sample.java

```
package samplepack;
public class sample
{
      private int x=10;
              int y=20; //default access
      protected int z=30;
      public int p=40;
}
```

Bin>javac –d . sample.java

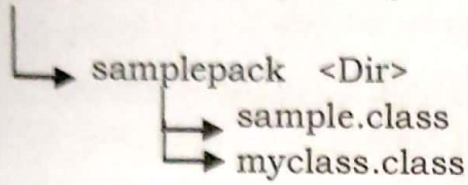


Bin>edit myclass.java

```
package samplepack;
public class myclass
{
      public void show()
      {
```

```
        sample obj=new sample();
        //System.out.print("\n obj.x="+obj.x); //Error
        System.out.print("\n obj.y="+obj.y);
        System.out.print("\n obj.z="+obj.z);
        System.out.print("\n obj.p="+obj.p);
    }
}
```

Bin>javac –d . myclass.java



In the above program, **sample** and **myclass** are present in the same package and **myclass** is a non-subclass therefore except **private** all other members can be accessed.

Bin> edit A.java

```
import samplepack.myclass;
class A
{
    public static void main(String argv[])
    {
        samplepack.myclass obj=new samplepack.myclass();
        obj.show();
    }
}
```

Bin>javac A.java
Bin>java A

obj.y=20
obj.z=30
obj.p=40

2. Same package and subclass

If the super and sub classes are present in the same package and subclass can access all types of members such as default, protected and public where as private cannot be accessed.

Note: Re-execute the above program with following modifications

1. Extend the **myclass** from sample class and do modifications in the **show()** method as shown below and the output will be as above.

```
package samplepack;
public class myclass extends sample
{
   public void show()
   {
    //subclass can access member of superclass directly
    //System.out.print("\n x="+x); //Error
    System.out.print("\n y="+y);
    System.out.print("\n z="+z);
    System.out.print("\n p="+p);
   }
}
```

3. Different package and non-subclass

If classes are present in different packages and one class in not the sub class of other and in this situation inside the non-subclass default and protected cannot be accessed where as public can be accessed. The private however cannot be accessed outside the class. The following programs demonstrate the same.

Bin>edit myclass1.java

```
package samplepack1;
import samplepack.sample;
public class myclass1
{
   public void show()
   {
        samplepack.sample obj=new samplepack.sample();
        //System.out.print("\n obj.x="+obj.x); //Error
        //System.out.print("\n obj.y="+obj.y); //Error
        //System.out.print("\n obj.z="+obj.z); //Error
        System.out.print("\n obj.p="+obj.p);
   }
}
```

Bin>javac -d . myclass1.java

└→ samplepack1 <Dir>

└→ myclass1.class

In the above program, **sample** and **myclass1** are present in different packages and **myclass1** is a non-subclass therefore except **public** all other members cannot be accessed.

Bin> edit B.java

```
import samplepack1.myclass1;
class B
{
    public static void main(String argv[])
    {
        samplepack1.myclass1 obj=new samplepack1.myclass1();
        obj.show();
    }
}
```

Bin>javac B.java
Bin>java B

obj.p=40

4. Different package subclass

If super and subclasses are present in different packages and in this situation inside the subclass default cannot be accessed where as protected and public can be accessed. The private of super class cannot be accessed outside of it.

Note: Re-execute the above program with following modifications

1. Extend the **myclass1** from **sample** class and do modifications in the **show()** method.

```
package samplepack1;
import samplepack.sample;
public class myclass1 extends samplepack.sample
{
    public void show()
    {
        samplepack.sample obj=new samplepack.sample();
        //System.out.print("\n obj.x="+obj.x); //Error
        //System.out.print("\n obj.y="+obj.y); //Error
        System.out.print("\n obj.z="+obj.z);
        System.out.print("\n obj.p="+obj.p);
    }
}
```

Path and Classpath Environment Variables

Path is one of the environmental variables of operating system. It is used to set the path for **.exe**, **.bat** and **.com** files. When a program file with extension **.exe** or **.bat** or **.com** is given at the prompt for execution, then the computer searches for the program file in the working directory first. If the program file is found in the working directory then the program executes. In case program does not found in the working directory then the computer searches for the program file in the path specified with path environmental variable of operatiing system. If the file is found in any diretories specified in **PATH** then the program file is executed otherwise the computer gives error message.

To see the existing path, use the **PATH** command as:

```
C:\>PATH
PATH=C:\WINDOWS\system32;C:\WINDOWS;C:\WINDOWS\System32\Wbem;C:\Program Files\Mi
crosoft SQL Server\90\Tools\binn\;C:\Program Files\Microsoft SQL Server\80\Tools
\Binn\;C:\Program Files\Microsoft SQL Server\90\DTS\Binn\;C:\Program Files\Micro
soft SQL Server\90\Tools\Binn\VSShell\Common7\IDE\;C:\Program Files\Microsoft Vi
sual Studio 8\Common7\IDE\PrivateAssemblies\;C:\Program Files\Common Files\Autod
esk Shared\;

C:\>
```

The **javac.exe** and java.exe files are present in **bin** directory and the path is :

C:\Program Files\Java\jdk1.6\bin

We write **A.java** program in bin directory as:

C:\Program Files\Java\jdk1.6\bin>edit A.java

```
class A
{
  public static void main(String argv[])
  {
    System.out.print("\n sample program");
  }
}
```

C:\Program Files\Java\jdk1.6\bin>javac A.java

The **javac.exe** file is present in the working directory therfore it executes as a result the java compiler(javac.exe) compiles **A.java**(english) into byte code (**A.class**) and stores in working directory(**bin**).

C:\Program Files\Java\jdk1.6\bin>java A

The **java.exe** program file is present in bin directory therefore it executes which intern executes **A.class** bytecode.

Suppose, if we want to compile and run java programs from different directories then it does not work. For example, we create a directory(**mydir**) in c drive and the path is **C:\mydir**

C:\>md mydir

In the **mydir** dirctory, we create a java program as shown below.

C:\mydir>edit B.java

```
class B
{
  public static void main(String argv[])
  {
    System.out.print("\n This is B.java");
  }
}
```

If we compile the above program using javac.exe command as

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

We get error, because **javac.exe** program file is not present in the working directory(**mydir**) or in the directories specified in the **PATH.**

How can we use *javac.exe* command any where in the computer?

It can be achieved by setting path to **mydir** directory and it can be done using **SET PATH** command as:

C:\mydir> SET PATH= give existing path ; c:\program files\java\jdk1.6.0\bin;

Now, we compile the program as:

C:\mydir>javac B.java

The program compiles successfully, and **B.class** file stores in the working directory(**mydir**). It works because **javac.exe** file is searched in working directory, as it is not found in the working directory, the computer searches for the **javac.exe** file in **PATH**. The javac.exe is found in **c:\program files\java\jdk1.6.0\bin** directory therefore the **javac.exe** executes and compiles **B.java** to **B.class**(bytecode).

Now, we execute the program as:

C:\mydir>java B

This is B.java

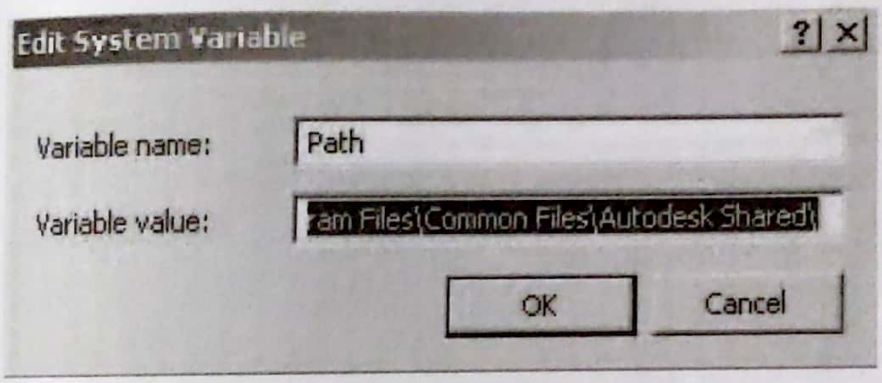
In this case, the computer searches for the **java.exe** in the **PATH** and it finds in **c:\program files\java\jdk1.6.0\bin** directory. Therefore **java.exe** program executes which intern executes **B.class** file.

In this way, if we want to compile and run java programs, we should set the path to **bin** directory.

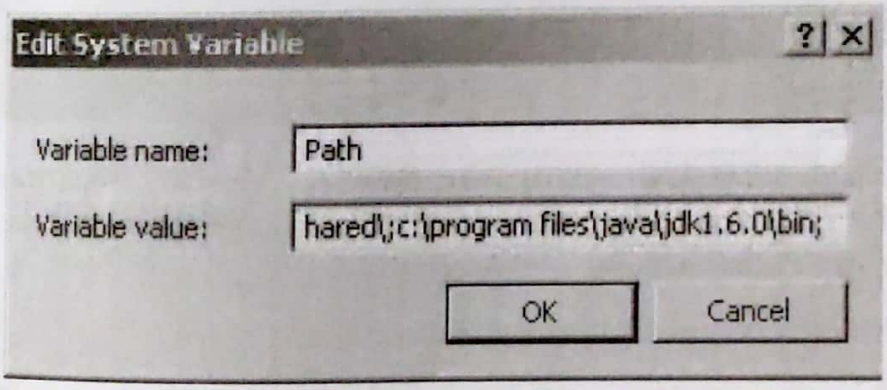
Note : Setting the path using **SET PATH** command is temporary i.e. the new path is deleted and old path is restored when the dos window is closed.

To create permanent path setting, follow these steps:

1. Right click on **mycomputer** icon on the desktop and select **properties** command.
2. Select **Advanced** tab.
3. Click on **Environmental Variables** button.
4. Select **path** variable from the **System Variables** list.
5. Click on **Edit** button, then we get the window as shown below.



6. Append the text **c:\program files\java\jdk1.6.0\bin;** at the end of **Variable value** text box as:.



7. Click on **Ok**, **Ok** and then **Ok** buttons.

Now, the path is set permanently. So, we can compile and execute java program from any directory of the computer.

Classpath

Path setting is used to set the path for **.exe**, **.com** and **.bat** files. Similarly, if we want to set the path to **.class** files of java programs from any directory of the computer then we have to use **CLASSPATH** environmental variable.

To execute **.class** files of java from any directory of computer then set the **CLASSPATH** to the directory of the **.class** file. For example, **B.class** file is present in **C:\mydir** and we can set the classpath to **B.class** as:

```
C:\mydir>SET CLASSPATH=.;c:\mydir;

C:\mydir>
```

Now, we can execute the **B.class** from any directory. For example:

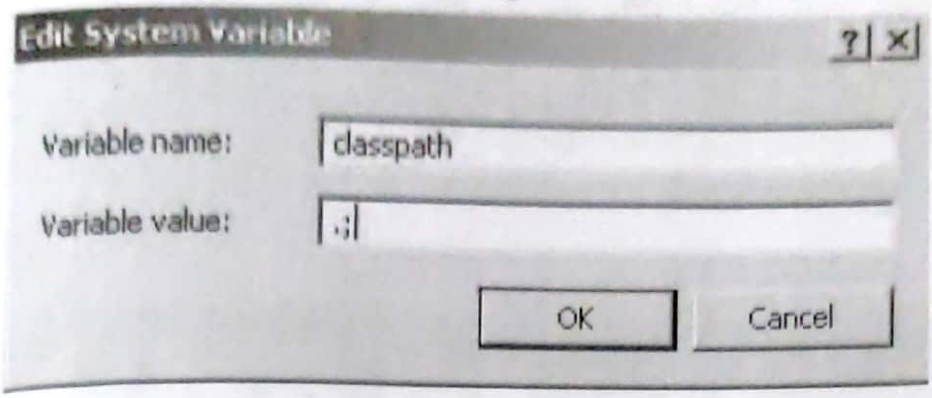
```
C:\mydir>java B

 This is B.java
C:\mydir>cd\

C:\>java B

 This is B.java
C:\>e:

E:\>java B

 This is B.java
E:\>c:

C:\>cd C:\Program Files\Java\jdk.1.6.0\bin

C:\Program Files\Java\jdk.1.6.0\bin>java B

 This is B.java
C:\Program Files\Java\jdk.1.6.0\bin>_
```

Note : Setting the classpath using the **SET CLASSPATH** command is temporary i.e. when the dos window is closed the newly set classpath is deleted and restores the old classpath.

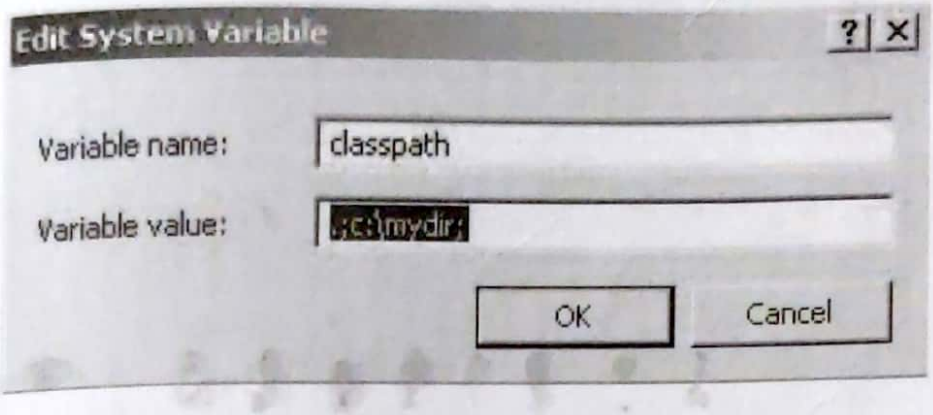
To create permanent classpath setting, follow these steps.

1. Right click on **mycomputer** icon on the desktop and select **properties** command.
2. Select **Advanced** tab.

3. Click on **Environmental Variables** button.
4. Select **classpath** variable from the **System Variables** list.
5. Click on **Edit** button, and then we get the window as shown below.



6. Append the text **c:\mydir;** at the end of **Variable value** text box as:.



7. Click on **Ok**, **Ok** and then **Ok** buttons.

Now, the classpath is set permanently. So, we can execute **.class** files present in **c:\mydir** directory from any directory of the computer.

Note: If classpath variable does not exist in the **System Variables list** then create the classpath variable using the **New** button as: