javac

→ we can use javac command to compile a single / group of java source files.

- javac [options] Test.java
- javac [options] A.java B.java C.java
- javac [options] *.java

→ options can be
- -version
- -d
- -source
- -verbose
- -cp / -classpath

java

→ we can use java command to run a single class file

- java [options] Test A B C
  - A B C: command line Arguments

→ Options can be
- -version
- -D
- -cp / -classpath
- -ea / -esa / -dsa / -da

> Note: we can compile any number of source files at a time But we can run only one class file at a time.

classpath

→ classpath describes the location where required .class files are available.

→ Java compiler & JVM will use classpath to locate required .class file.

→ By default, JVM will always searches in CWD for the required .class file.

→ If we set classpath explicitly then JVM will search in our specified classpath location &

JVM won't search in CWD.

⇝ we can set classpath in the following three ways :

① By using Environment Variable classpath

- This way of setting classpath is permanent & will be preserved across system Restarts.
- whenever we are installing a permanent slow in our system then this approach is recommanded

② At command prompt level by using set command

- [set classpath = c:\durga-classes]
- This way of setting classpath will be preserved only for particular cmd, once cmd closes automatically classpath will be gone.

③ At command level by using -cp option

- [java -cp c:\durga-classes Test]
- This way of setting classpath will be preserved only for particular command, once command executes the classpath is lost.

Note: Among 3 ways of setting classpath, setting classpath at command-level is recommanded, bcz dependent classes are varied from command to command

⇝ Once we set the classpath, we can run our program from any location.

⇝ Once we set classpath, JVM won't search in CWD & it will always search in specified classpath location only.

c:\durga_classes

Ex1:

```
class Test
{
    p s v main(Sm)
    {
        s.o.p("classpath Demo");
    }
}
```

- c:\durga_classes > javac Test.java
- c:\durga_classes > java Test

[o/p : classpath Demo]

- c:\ > java Test

[RE: NoClassDefFoundError: Test]

- c:\ > java -cp c:\durga_classes Test

[o/p : classpath Demo]

- D:\ > java -cp c:\durga_classes Test

[o/p : classpath Demo]

- E:\ > java -cp c:\durga_classes Test

[o/p : classpath Demo]

- c:\durga_classes >
  java -cp E: Test

[R.E. NoClassDefFoundError - Test]

- c:\durga_classes > java -cp ;E: Test

[o/p : classpath Demo]

~ we can specify multiple classpath location by separating the locations by semi-colon (;).

Ex2:

C:
  - student.class

```
public class student
{
    public void m()
    {
        s.o.p("I want Job");
    }
}
```

D:
  - IT.class

```
public class IT
{
    p s v main(Sm)
    {
        student st = new student();
        st.m();
        s.o.p("you will get soon");
    }
}
```

- C:\> java student.java

- D:\> java IT.java

[C.E: cannot find symbol
symbol : class Student
location: class IT]

- D:\> java -cp c: IT.java

- D:\> java IT

[R.E. NoClassDefFoundError : Student]

- D:\> java -cp c: IT

[R.E. NoClassDefFoundError : IT]

- D:\> java -cp .;c: IT ☺

- E:\> java -cp D:;c: IT ☺

[O/P:- I want Job
You will get soon]

EX3.

C:
- pack1
  - pack2
    - kareena.class


```
package pack1.pack2;

public class kareena
{
    public void m1()
    {
        S.O.P("Hello saif, can u
              plz hellotune");
    }
}
```

D:
- pack3
  - pack4
    - saif.class

```
package pack3.pack4;

import pack1.pack2.kareena;

public class saif
{
    public void m2()
    {
        kareena k = new kareena();
        k.m1();
        S.O.P("Not possible bcz I
              am in SCJP class");
    }
}
```

E:
- durga.class

```
import pack3.pack4.Sait;

public class Durga
{
    p s v main (Str )
    {
        Sait s = new Sait();
        s.m2();
        s.o.p ("Hello Kareena
                can I help you");
    }
}
```

- C:\> javac -d . Kareena.java
- D:\> javac -d . Sait.java

C.E: Cannot find Symbol
Symbol : class Kareena
Location : class pack3.pack4.Sait

- D:\> javac -d . cp C: Sait.java
- E:\> javac Durga.java

C.E: cannot find symbol
symbol : class Sait
location : class Durga

- E:\> javac -cp D: Durga.java
- E:\> java Durga

R.E: NoClassDefFoundError :
pack3.pack4.Sait

- E:\> java -cp .;D: Durga

R.E: NoClassDefFoundError :
pack1.pack2.Kareena

- E:\> java -cp .;D:;C: Durga
- F:\> java -cp E:;D:;C: Durga

Conclusions :-

(1) If any location created because of package statement that location should be resolved by using "import" statement and base package location we have to update in "classpath"

(2) Compiler will check only one level dependency (Bcz class file of previous level class will only be

generated it that class compiles successfully, so that ~~the~~ compiler don't need to go beyond one level dependency) whereas JVM will check all levels of dependency (Bez after successful generation of .class file it might be deleted)

(3) In classpath the order of locations is important & JVM will always considers from left to Right until required match available

Ex. C:

```
public class Nagavalli
{
    p s v main (SA)
    {
        S.O.P("C:Nagavalli");
    }
}
```

D:

```
public class Nagavalli
{
    p s v main (SA)
    {
        S.O.P ("D:Nagavalli");
    }
}
```


E:

```
public class Nagavalli
{
    p s v main (SA)
    {
        S.O.P("E:Nagavalli");
    }
}
```

→ java -cp D:; C:; E: Nagavalli

O/P :- D:Nagavalli

→ java -cp E:; D:; C: Nagavalli

O/P :- E:Nagavalli

JAR Files

⇝ If several dependent classes are there then its not recommanded to set the classpath individually.

⇝ [java -cp C:; D:; E:; ...]

Not Recommanded

→ To handle this situation, we have to group all this .class files into a single zip file & we have to place that zip file in the classpath.

→ This zip file is nothing but JAR files.

→ All third party s/w plugins are by default available in the form of JAR files only.

→ Ex1 :- To develop a servlet all dependent classes are available in "Servlet-api.jar"

- we have to place this jar file in classpath to compile a servlet program.

→ Ex2 :- To run JDBC program all dependent classes are available in "ojdbc14.jar"

- To run JDBC program, we have to place this JAR file in classpath.

→ Ex.3 :- To use Log4J in our application, dependent classes are available in "log4j.jar".

- we have to place this JAR into classpath then only Log4J based application can run.

[Log4J = Log for JAVA]

→ Various Commands :-

(1) To Create a JAR File (Zip file)

- Jar -cvf durgacalc.jar Test.class
- Jar -cvf durgacalc.jar A.class B.class
- Jar -cvf durgacalc.jar *.class
- Jar -cvf durgacalc.jar *.*

[cvf = .create
.verbose mode
.named file]

(2) To Extract a Jar file (Unzip file)

Jar -xvf durgacalc.jar

[-xvf = extract..]

(3) To display table of contents :

Jar -tvf durgacalc.jar

⇝ Ex. Service Provider's Role

```
public class DurgaColorfulCalc
{
    public static void add
            (int x, int y)
    {
        s.o.p(x*y);
    }
    public static void multiply
            (int x, int y)
    {
        s.o.p(2*x*y);
    }
}
```

```
javac DurgaColorfulCalc.java ↵
jar -cvf durgacalc.jar DurgaColorful
Calc.java Calc.class
```

durgacalc.jar

⇝ Note : To place ".class" file in class path just location is enough but To make "JAR" file available in classpath location is not enough compulsory we have to include name of the JAR file also.

Client's Role

⇝ He download the JAR file and he placed in D: of client's machine.

⇝

```
class Bakara
{
    p s v main(sm )
    {
```

```
DurgaColorfulCalc.add(10,20);
DurgaColorfulCalc.multiply(10,20);
    }
}
```

C:\durgaclasses> javac Bakara.java
(x) C.E.

C:\durgaclasses> javac -cp D: Bakara.java
(x) C.E

C:\durgaclasses> javac -cp D:\durgacalc.jar Bakara.java
(✓)

C:\durgaclasses> java Bakara
R.E: NoClassDefFoundError: DurgaColorfulCalc

C:\durgaclasses> java -cp D: Bakara
R.E: NoClassDefFoundError: Bakara

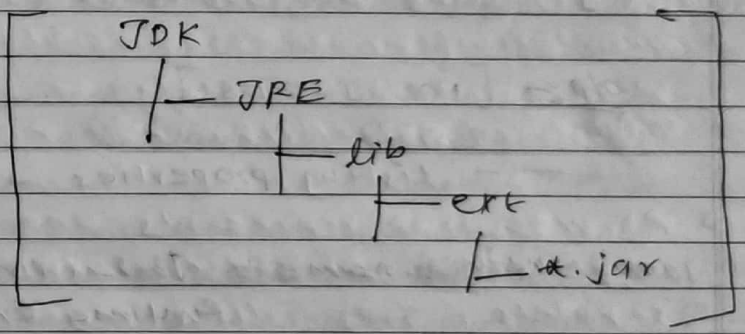
C:\durgaclasses> java -cp .;D: Bakara
R.E. NoClassDefFoundError: DurgaColorfulCalc

C:\durgaclasses> java -cp .;D:\durgacalc.jar Bakara
(✓)


```
[ 0/p:
  200
  400 ]
```

→ Short-cut way to place JAR file in classpath:-

- If we place JAR file in the following location then all classes & interfaces present in JAR file by default available to Java-compiler & JVM, we are not required set classpath explicitly.



(*) System Properties

→ For every system some persistent information will be maintained in the form of System Properties.

→ This include name of the OS, JAVA version, JVM vendor, User country etc.

→ Demo program to print system Properties.

Ex. import java.util.*;

```
class Test
{
    p s v main (Str )
    {
        Properties p =
            System.getProperties();
        p.list (System.out );
    }
}
```

O/p:- [For JDK 1.6]

```
-- listing properties --

java.runtime.name = Java(TM) SE
                    Runtime Environment

sun.boot.library.path = c:\program Files
                \Java \Jdk 1.6.0.\bin --

java.vm.version = 11.0-b16
java.vm.vendor = http://java.sun.com/
java.vm.vendor = Sun Microsystems Inc.
```

→ We can set system property explicitly from the CMD by using -D option

```
java -D durga = OCJP Test ↵
```

(-D: No space allowed; durga: prop name; OCJP: prop value)

→ The main advantage of setting system property is we can customize the behavior of Java program.

Ex.

```
class Test
{
    p s v main (Str )
    {
        String course =
            System.getProperty ("course");

        if (course.equals ("scjp"))
        {
            s.o.p ("scjp information");
        }
        else
        {
            s.o.p ("other course
                    information");
        }
    }
}
```

o/p :-

java -Dcourse=scjp Test ↵
scjp information

java -Dcourse=scwcd Test ↵
other scwcd information.

JAR (vs) WAR (vs) EAR

⇒ JAR :- (Java Archieve)

↝ It contains a group of .class files

⇒ WAR :- (Web Archieve)

↝ A WAR file represents one web application which contains servlets, JSPs, HTML pages, JS files, CSS etc.

↝ The main advantage of maintaining web application in the form of WAR file is Project Deployment, Project Delivery & Project Transportation will become easy.

⇒ EAR (Enterprise Archieve)

↝ An EAR file represents one Enterprise application which contains servlets, JSPs, EJBs, JMS components etc.

Note :- In general EAR file represents a group of WAR files & JAR files

Web Application (vs) Enterprise Application

↝ A web application can be developed by only web related technologies like servlets, JSPs, HTML, CSS, JS, xml etc.

Ex- Online Library Management System
Online shopping cart

↝ An Enterprise application can be developed by any technology from JAVA J2EE like servlets, JSPs, EJBs, JMS components etc.

Ex. Banking Application
Telecom Based Project-

Note :- J2EE / JEE compatible application is Enterprise application.

Web server vs Application Server

→ Web server provides environment to run web application.

→ Web server provides support for web related technologies like servlets, JSPs, HTML etc.

→ Ex. Tomcat

→ Application server provides environment to run Enterprise application.

→ Application server can provide support for any technologies from JAVA J2EE like servlets, JSPs, EJBs, JMS components etc.

→ Ex. weblogic, Websphere, JBoss etc.

Note :- Every Application server contains In-built Web server to provide support for web related technologies

→ J2EE compatible server is Application server.

How to Create Executable JAR file ?

→
```
import java.awt.*;
import java.awt.event.*;

public class JarDemo
{
    p s v main(str)
    {
        Frame f = new Frame();
        f.addWindowListener
            (new WindowAdapter(){
            public void window(losing(WEe)
            {   for(int i=1; i<10; i++)
                {
                    s.o.p("close:"+i);
                }
                System.exit(0);
            }
        });
```

```
f.add(new Label("I can create
        Executable Jar File!!"));
f.setSize(500,500);
f.setVisible(true);
}
}
```

→ manifest.MF

```
Main-Class: JarDemo
```

(Enter after writing main class name is mandatory, means cursor should come at new line)

→ javac JarDemo.java

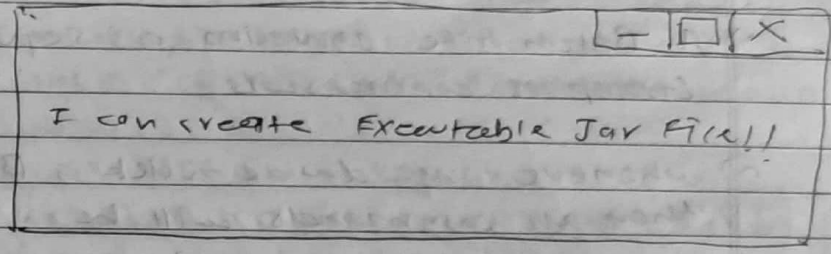
JarDemo.class JarDemo$1.class (Inner class)

→ Jar -cvfm demo8.jar manifest.MF JarDemo.class JarDemo$1.class

demo8.jar (JAR File created)

→ Java -jar demo8.jar

O/P:-



Even we can run the JAR file by just double-clicking it.

(*) How many ways to run Java Program

→ we can run a Java program in the following ways.

(i) From CMD we can run ".class file" with "java" command.

java JarDemo

(ii) From CMD we can run ".jar file" with "java" command

java -jar demo.jar

(iii) By double clicking a JAR file

(iv) By double clicking a Batch file

→ A Batch file contains a sequence / Group of commands

→ whenever we double-click a Batchfile then all commands will be executed one by one in the sequence.

→ abc.bat

```
java -cp c:\durga_classes JavDemo
```

(*) Classpath (vs) Path

→ classpath describes the location where required .class files are available

→ Java-compiler & JVM will use classpath to locate required .class files.

→ If we are not setting classpath then our program may not compile or may not run.

→ Path describes the location where required Binary-executables are available.

→ If we are not setting path then "javac" & "java" command won't work.

→ [set path = c:\Program Files\Java\Jdk 1.6.0-11\bin]

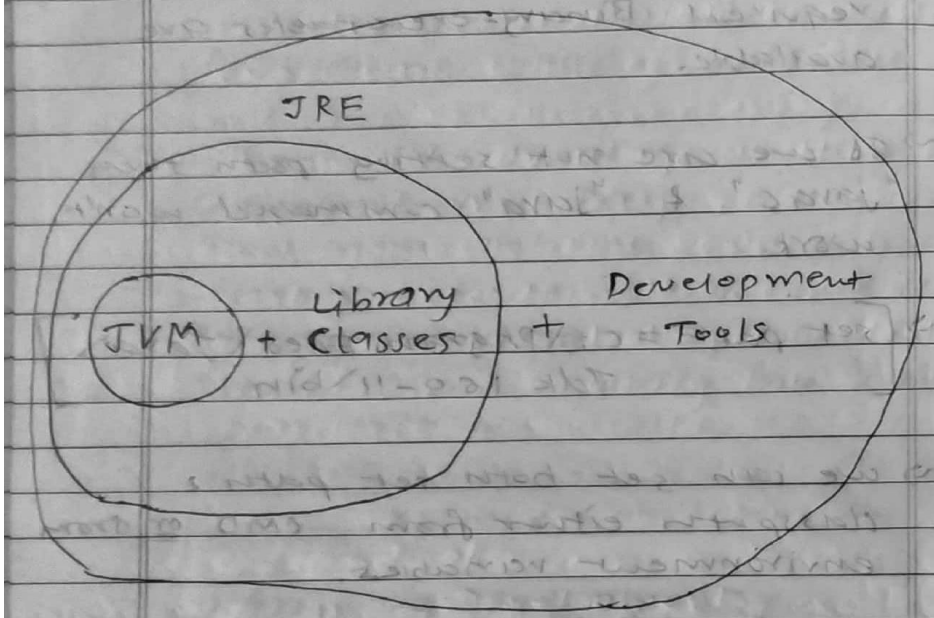
→ we can set both set path & classpath either from CMD or from environment variables.

(*) JDK (vs) JRE (vs) JVM

→ JDK (Java Development Kit) provides environment to develop & run Java Application.

→ JRE (Java Runtime Environment) provides environment to run Java Application.

→ JVM (Java Virtual Machine) is responsible to run Java program line by line, hence it is an Interpreter.



JDK = JRE + Development Tools

JRE = JVM + Library classes

- JVM is a part of JRE whereas JRE is a part of JDK.
- At Developer's machine we have to install JDK whereas At the client's machine we have to install JRE

java (vs) javaw (vs) javaws

- we can use "java" command to run a java .class file wherer s.o.p will be executed and corresponding o/p will be displayed to the console.
- We can use "javaw" (Java without console o/p) command to run a java .class file where s.o.p will be executed but corresponding o/p won't be displayed to the console.
- In general we can use "javaw" command to run GUI based applications.
- we can use "javaws" (Java web start utility) command to download a java application from the web & to start its execution.
- we can "javaws" command as follows

```
javaws <URL>
```

→ It downloads the application from the specified URL & starts execution.

→ The main advantage in this approach is every end user will get updated version & enhancement will become easy bcz of centralized control.

End of Development

Start of Garbage Collection