19. Development

javac :-

→ We can use javac command to compile a single or group of Java source files.

Ex: javac [options] A.java
A.java B.java C.java
*.java

- -source
- -d
- -version
- -verbose
- -cp|-classpath
- ⋮

java:-

→ We can use java command to run a single class.

Ex: java [options] Test A B C
(A B C: command line arguments)

- -version
- -cp|-classpath
- -verbose
- -D
- -ea|-da|-esa|-dsa
- ⋮

Note:- We can compile any no. of source files at a time but we can run only one class at a time.

classpath:-

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

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

→ We can set classpath in the following 3 ways.

1) By using Environment variable classpath.
2) By using set command at command prompt.
3) By using -cp option at command level.

1) By using Environment variable classpath :-

→ This way of setting classpath is permanent and preserved accross system restarts.

→ This way of setting classpath is recommended whenever we are installing a permanent s/w in our system.

2) By using set command at command prompt :-

```
set classpath = C:\durga_classes
```

→ This way of setting classpath will be preserved only for that command prompt.

→ Once we close command prompt automatically classpath will be lost.

3) By using -cp option at command level :-

→ This way of setting classpath will be preserved only for that command.

→ Once command execution completes automatically classpath will be lost.

Ex: java -cp C:\durga_classes Test ↵

Note :- The most commonly used approach in Realtime is Third approach i.e., to set classpath at command level.

Conclusions :-

1) By default JVM will always search in current working Directory for the required .class file.

2) If we set classpath explicitly then JVM won't search in current working Directory (CWD) & it will search only in our

specified location.

3) If we set classpath explicitly then we can run our program from anywhere.

Ex: ① class Test

```
{
  p s v main(String[] args)
  {
    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↵ ✓

D:\java -cp c:\durga_classes Test↵ ✓

E:\java -cp c:\durga_classes Test↵ ✓

c:\durga_classes>java -cp E: Test↵ ✗

RE: NoClassDefFoundError: Test

c:\durga_classes>java -cp E:;. Test↵ ✓

Ex②:

c:
└ AStudent.class

```
public class AStudent
{ public void m1()
  {
    S.o.p("I want JOB
          immediately");
  }
}
```

D:

```
class ItIndustry
{
  p s v m(-)
  {
    AStudent a=new AStudent();
    a.m1();
    S.o.p("U will get soon!!!");
  }
}
```

```
C:\> javac AStudent.java ↵ ✓
D:\> javac ItIndustry.java ↵ → CE : cannot find symbol
                               symbol: class AStudent
                               location: class ItIndustry
D:\> javac -cp C: ItIndustry.java ↵ ✓
D:\> java ItIndustry ↵ ✗
```

RE : NoClassDefFoundError : AStudent

```
D:\> java -cp C: ItIndustry ↵ ✗
```

RE : NoClassDefFoundError : ItIndustry

```
D:\> java -cp .;C: ItIndustry ↵ ✓
F:\> java -cp D:;C: ItIndustry ↵ ✓
```

Ex③:

```
C:
 ├ pack1
    ├ pack2
       ├ Kareena.java
```

```
package pack1.pack2;
public class Kareena
{
 public void m1()
 {
  S.o.p("Hello Saif----
   can u plz set hello
   tune");
 }
}
```

```
D:
 ├ pack3
    ├ pack4
       ├ Saif.java
```

```
package pack3.pack4;
import pack1.pack2.
        Kareena;
public class Saif
{
 public void m2()
 {
  Kareena k=new
        Kareen();
  k.m1();
  S.o.p("Not possible
   as I m in SCJP
   class");
 }
}
```

```
E:
 ├ Durga.class
```

```
import pack3.pack4.Saif;
class Durga
{
 P s v m(-)
 {
  Saif s=new Saif();
  s.m2();
  S.o.p("Hello Kareena
   can I help u");
 }
}
```

C:|>javac -d . Kareena.java ↵ ✓

D:|>javac -d . Saif.java ↵ X → CE: cannot find symbol
symbol: class Kareena
location: class Saif

D:|>javac -cp C: -d . Saif.java ↵ ✓

E:|>javac Durga.java ↵ X → CE: cannot find symbol
symbol: class Saif
location: class Durga

E:|>javac -cp D: Durga.java ↵ ✓

E:|>java Durga ↵ X

RE: NoClassDefFoundError: Saif

E:|>java -cp D: Durga ↵ X

RE: NoClassDefFoundError: Durga

E:|>java -cp .;D: Durga ↵ X

RE: NoClassDefFoundError: Kareena

E:|>java -cp .;D:;C: Durga ↵ ✓

F:|>java -cp E:;D:;C: Durga ↵ ✓

Conclusions:-

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

2) Compiler will check only one level of dependancy whereas JVM will check all levels of dependancy.

3) In classpath, the order of locations is important and JVM will always search from left to right until required class file available.

Ex:

C:
```
class Nagavalli
{
  p s v m(-)
  {
    S.o.p("C: Nagavalli");
  }
}
```

D:
```
class Nagavalli
{
  p s v m(-)
  {
    S.o.p("D: Nagavalli");
  }
}
```

E:
```
class Nagavalli
{
  p s v m(-)
  {
    S.o.p("E: Nagavalli");
  }
}
```

```
java -cp C:;D:;E; Nagavalli ↵
```

o/p : C: Nagavalli

```
java -cp E:;D:;C: Nagavalli↵
```

o/p : E: Nagavalli

Jar file:—

→ If several dependent classes are available then its never recommended to set the classpath individually. we have to group all those class files into a single zip file which is nothing but Jar file, we have to place that jar file in classpath.

→ All third party s/w plug-in's are available in the form of jar file only.

Ex①: To write a Servlet all dependent classes are available in servlet-api.jar. we have to place this jar file in classpath. So that we can compile Servlet class.

Ex②: To use log4J in our application all required class files available in log4j.jar. we have to place this jar file in classpath. So that we can use log4J.

Various Commands:—

1) To create a jar file (zip file):—

```
jar -cvf durgacale.jar A.class ↵
                       A.class B.class C.class
                       *.class
                       *.*
```

2) To extract jar file (unzip):—

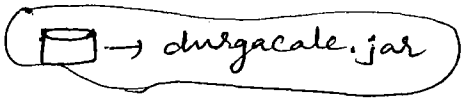
```
jar -xvf durgacale.jar ↵
```

3) To display table of contents of a jar file:—

```
jar -tvf durgacale.jar ↵
```

Ex: Service Provider Role:—

```
public class DurgaColorfulCale
{
  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 DurgaColorfulCale.java ↵
jar -cvf durgacale.jar DurgaColorfulCale.class ↵
```



Client's Role:—

Downloaded jar file and placed in D: of local machine.

```
class Bakara
{
    p s v m()
    {
      DurgaColorfulCalc.add(10, 20);
      DurgaColorfulCalc.multiply(10, 20);
    }
}
```

```
C:\durga_classes> javac Bakara.java ✗
C:\durga_classes> javac -cp D: Bakara.java ↵ ✗
C:\durga_classes> javac -cp D:\durgacalc.jar Bakara.java ↵ ✓
C:\durga_classes> java Bakara ↵ ✗
C:\durga_classes> java -cp .;D:\durgacalc.jar Bakara ↵ ✓
```

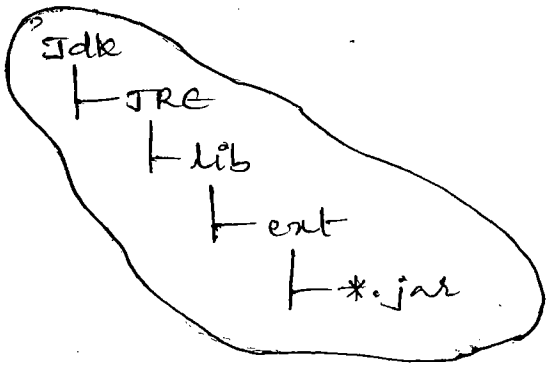
O/P : 200
400

Note:- To place jar file in the classpath just location is not enough compulsory we have to include name of the jar file also.

Short cut way to place jar file in the classpath:-

→ If we place jar file in the following location automatically it is available to Java compiler and JVM.

→ We are not required to set classpath explicitly.



System properties:—

→ For every system some persistant information will be maintained in the form of system properties.

→ These include

1) Java version
2) JVM vendor
3) JVM vendor url
4) Username
5) User country
6) OS name etc.

Demo program to display all system properties:—

```
import java.util.*;
class Test
{
  p s v m(--)
  {
    Properties p=System.getProperties();
    p.list(System.out);
  }
}
```

→ We can set a system property from the command prompt by using -D option.

Ex: java -Ddurga=scjp Test ↵

Space is not allowed — property name — property value.

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

Ex:
```
class Test
{
  p s v m(-)
  {
    String course = System.getProperty("course");
    if(course.equals("scjp"))
        S.o.p("scjp information");
    else
        S.o.p("Other course information");
  }
}
```

```
java -Dcourse=scjp Test ↵
```

o/p : scjp information

```
java -Dcourse=scwcd Test ↵
```

o/p : Other course information

1) jar Vs war Vs ear :-

i) jar (Java archieve) :-

→ A jar file contains a group of .class files.

ii) war (web archieve) :-

→ A war file represents one web application which contains Servlets, Jsp's, HTML pages, Css files, Javascript files etc.

→ If we maintain web application in the form of war then project delivery transportation and deployment will become easy.

iii) ear (enterprise archieve) :-

→ An ear file represents an enterprise application which contains Servlets, Jsp's, EJB's, JMS components etc.

Note :- Whether the file is jar or war or ear, we can create only by using jar command based on extension the corresponding

file will be created.

Ex: jar -cvf durgacale.jar *.*↵
jar -cvf webapp1.war *.*↵
jar -cvf appl.ear *.*↵

② Web application Vs Enterprise application:—

→ Web application can be developed by only web related technologies like Servlets, JSP's, HTML, CSS etc.

→ Whereas Enterprise application can be developed by any technology from Java J2EE like Servlets, JSP's, EJB's etc.

Note:- J2EE compatible application is enterprise application.

③ Web Server Vs Application Server:—

→ Web server provides environment to run web applications.

Ex: Tomcat.

→ web server can provide support only for web related technologies like Servlets, JSP's, HTML etc.

→ Application server provides environment to run enterprise applications.

Ex: Weblogic, Websphere, JBOSS etc.

→ Application server provides support for any technology from Java J2EE like Servlets, JSP's, EJB's etc.

Note:-① J2EE compatible server is Application server.

② Every Application server contains in-built web server.

④ path Vs classpath:—

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

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

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

→ path describes the location where binary executables are available.

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

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

⑤ JDK Vs JRE Vs JVM :—

i) JDK (Java Development Kit) :—

→ JDK provides environment to develop and run Java appl's.

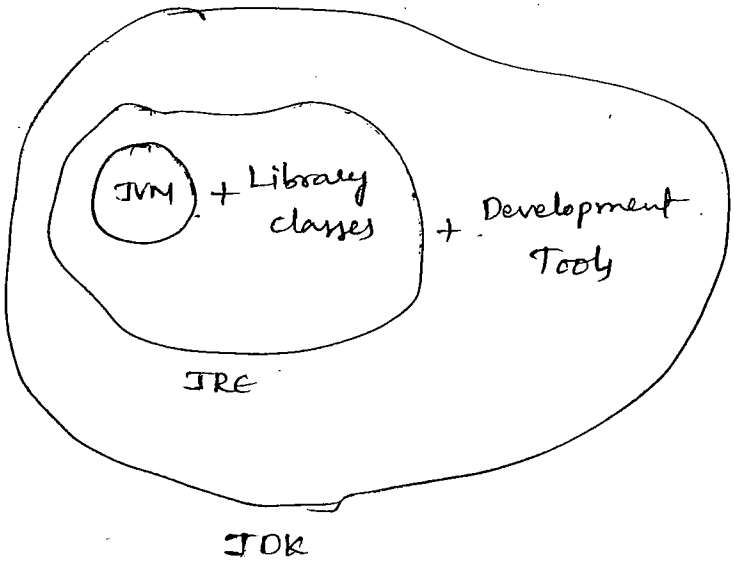
ii) JRE (Java Runtime Environment) :—

→ It provides environment to run Java appl's.

iii) JVM (Java Virtual Machine) :—

→ JVM is an interpreter which is responsible to run Java program line by line.

→ JVM is the part of JRE whereas JRE is the part of JDK.



JDK = JRE + Development Tools
JRE = JVM + Library classes

Note:- On the developer's machine we have to install JDK whereas On the client's machine we have to install JRE.

⑥ java Vs javaw Vs javaws :—

i) java :—

→ We can use java command to run Java class.

→ In this case, console o/p will be considered.

ii) javaw :—

→ We can use javaw command to run Java application without considering console output.

→ In this case, S.o.p statements will be executed but won't display the corresponding o/p to the console.

→ Even in the case of exception also exception information won't be displayed to the console.

→ This utility is best suitable to run GUI based applications.

Ex: ①
```
import java.io.*;
class Test
{
  p s v m(_) throws IOException
  {
    PrintWriter out = new PrintWriter("abc.txt");
    out.println("File Output");
    out.flush();
    S.o.p("Console Output");
  }
}
```

java Test ↵	javaw Test ↵
In this case, console output will be displayed to the console & File output will be return to the file	In this case, File Output will be return to the file but Console output won't be displayed at the console.

Ex②:
```
class Test
{
  p s v m(-)
  {
    s.o.p(10/0);
  }
}
```

java Test↵	javaw Test↵
In this case, Exception information will be displayed to the console.	In this case, Exception information won't be displayed to the console.

→ Java is to run Java programs with console o/p and javaw is to run Java programs without console o/p.

iii) javaws (java webstart utility):-

→ javaws is used to launch a Java application that is distributed through web.

Syntax: `javaws jnlp_url`

It downloads application from the url & launches it.

→ It is useful to distribute application to users & use central control to provide updates and ensures all users are using latest software.

→ When the application is invoked it is cached in the local computer.

→ Everytime i is launched it checks if there is any update available from the distributor.

⑦ How to create executable jar file?

JarDemo.java:-
```
import java.awt.*;
import java.awt.event.*;
```

```
public class JarDemo
{
    p s v m(-)
    {
      Frame f=new Frame();
      f.addWindowListener(new WindowAdapter()
      {
        public void windowClosing(WindowEvent e)
        {
          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↵

↵ → Cursor will be in the next line.

javac JarDemo.java↵

JarDemo.class JarDemo$1.class

jar -cvfm demo3.jar manifest.MF JarDemo.class JarDemo$1.class↵

java -jar demo3.jar↵

⑧ In how many ways we can execute a Java program?

1) With java command to run .class file

```
java JarDemo↵
```

2) with java command to run .jar file

```
java -jar demo3.jar↵
```

3) By double clicking a jar file.

4) By double clicking a batch file.

abc.bat

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