Development

javac :-

We can use this Command to Compile a Single or group of .java files.

Syn:- javac [options] A.java/

A.java B.java

*.java

[options]
- -d
- -Source
- -cp
- -classpath
- -version

java :-

We can use java Command to run a .class file

Syn:- java [options] A ↵

[options]
- -ea | -esa | -da | -dsa
- -version
- -cp / -classpath
- -D
- :

Note:- We can Compile a group of .java files at a time where as we can run only on .class file at a time.

Classpath :-

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

→ JVM will always use Classpath to locate the required .class file.

→ The following are various possible ways to Set the Classpath.

① Permanently by using Environment variable classpath.

→ This classpath will be preserved after system restart also

② At Command prompt level by using Set Command.

Set classpath = %classpath%;D:\path>

→ This classpath will be applicable only for that particular Comand prompt window only. Once we close that Command prompt automatically classpath will be lost

③ At Command Level by using -cp option

java -cp D:\path> Test ↵

→ This classpath is applicable only for this particular Command. Once Command execution Completes automatically classpath will be lost.

* Among the above 3 ways the most Commonly used approach is Setting classpath at Command Level.

Ex!- class Test
{
p.s.v.m(——)
{
S.o.pln("Classpath Demo");
}
}

D:\Durgaclasses\> javac Test.java ↵
>java Test ↵

o/p!- Classpath Demo

✗ D:\java Test ↵ R.E!- NoClassDeffoundError

✓ D:\> java -cp D:\Durgaclasses Test ↵ ✓

o/p!- ClasspathDemo

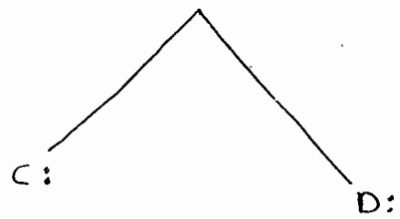
✓ D:\> java -cp D:\Durgaclasses Test ↵

o/p!- ClasspathDemo

Note:-

If we set classpath explicitly then we can run Java program from any location but if we are not setting the classpath then we have to run java program only from current working directory.

Ex:-




```
public class fresher
{
  public void m1()
  {
    S.o.pln ("I want JOB");
  }
}
```

```
class Company
{
  p.s.v.m (---->)
  {
    fresher f = new fresher();
    f.m1()
    S.o.pln("Getting JOB is very
             easy.. Not required to
             worry");
  }
}
```

C:\> javac fresher.java ✓

D:\> javac Company.java ✗

C.E:- Cannot find symbol
Symbol: class fresher
location: class Company

D:\> javac -cp C: Company.java ✓

✗ D:\javac Company ←

R.E:- NoClassDefFoundError: fresher

✗ D:\java -cp C: Company ←

R.E:- NoClassDefFoundError: Company

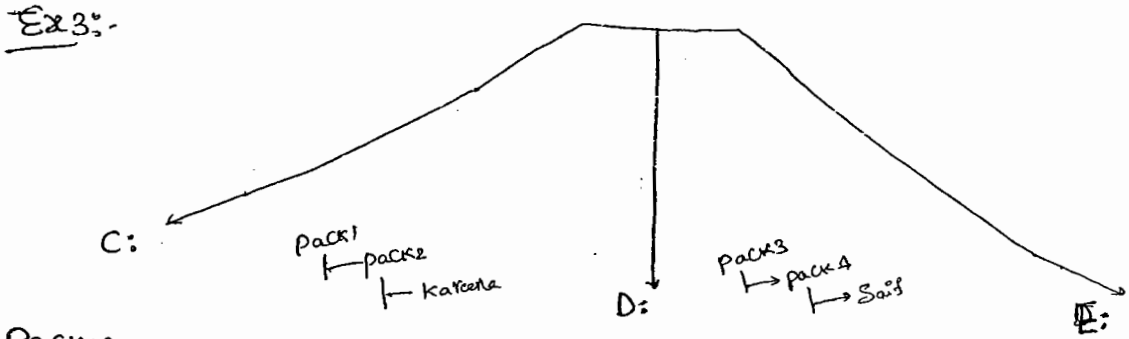
✓ D:\ java -cp D:;C: Company (or) D:\java -cp .;C: Company

o/p :- I can JOB

Getting JOB is very easy... not required to worry.

✓ E:\ java -cp D:;C: Company

Ex3:-




```
Package pack1.pack2;
Public class Kareena
{
 public void m1()
 {
S.o.pln(" Hello Saif Can I help U");
 }
}
```

```
package pack3.pack4;
Import pack1.pack2.Kareena
Public class Saif
{
Public void m1()
{
Kareena k = new Kareena();
K.m1();
S.o.pln(" Not possible.. As I am in SCJP class");
}
}
```

```
Import pack3.pack4.Saif;
class Durga
{
 p.s.v.m(—)
 {
 Saif s = new Saif();
 S.m1();
 S.o.pln(" Can please tell hello —func");
 }
}
```

✓ C:\> java -d. Kareena

✓ D:\> java -d. Saif.java

C.E:- Cannot find Symbol

Symbol : class Kareena

Location : class Saif

✓ D:\ java -cp c: -d . Saif.java

✗ E:\ javac Durga.java

C.E:- Cannot find Symbol

Symbol : class Saif

location : class Durga

✓ E:\> javac -cp D: Durga.java

✗ E:\> java Durga ↵

R.E:- NoClassDefFoundError : Saif

✗ E:\> java -cp D: Durga ↵

R.E:- NoClassDefFoundError : Durga

✗ E:\> java -cp .;D: Durga

R.E:- NoClassDefFoundError : Kasreena

✓ E:\> java -cp E:;D:;C: Durga

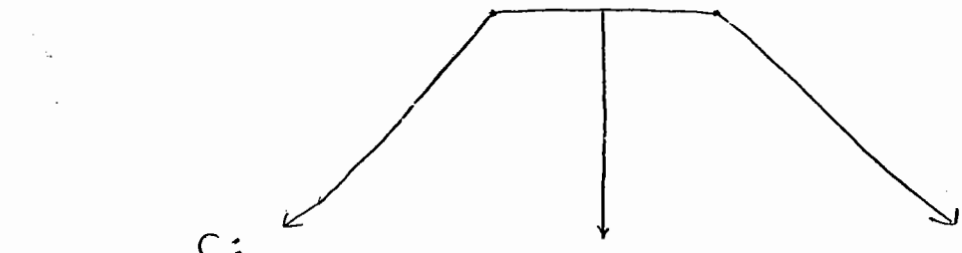
Note:-

① Compiler will check only one level dependency where as JVM will checks all levels of dependency

② If any folder structure created because of package statement it should be resolved through import statement only. & Base package location we have to update in classpath.

③ within the classpath the order of locations is very important for the required .class file, JVM will always search the locations from

Left → Right in classpath. Once JVM finds the required file then the rest of the classpath won't be searched.




```
Class Nagavali
{
 p.s.v.m(--)
 {
 S.o.pln("C: Nagavali");
 }
}
```

```
Class Nagavali
{
 p.s.v.m(--)
 {
 S.o.pln("D: Nagavali");
 }
}
```

```
Class Nagavali
{
 p.s.v.m()
 {
 S.o.pln("E: Nagavali
 }
}
```

C:\> javac Nagavali.java ✓

D:\> javac Nagavali.java ✓

E:\> javac Nagavali.java ✓

C:\> java Nagavali ✓

o/p C: Nagavali

D:\> java -cp C:;D:;E: Nagavali ↵

o/p C: Nagavali

D:\> java -cp E:;D:;C: Nagavali ↵

o/p: E: Nagavali

D:\> java -cp D:;E:;C: Nagavali ↵

o/p: D: Nagavali

JAR file :-

→ If Several dependent files are available then it is never recommended to set each class file individually in the classpath we have to group all those .Class file into a Single Zip file. & we have to make that Zip file available in the class path. This zip file is nothing but JAR file.

Ex(1) :-

To develope Servlet all required .class files are available in Servlet-api.jar. we have to make this Jar file available in the classpath then only Servlet will be Compiled.

jar Vs war Vs ear :-

① jar : (java archieve file)

→ It Contains a group of .class files

② war :- (web archieve file)

→ It represents a web application which may Contain Servlets, Jsps, HTMLs, css file, JavaScripts, e.t.c..

③ ear :- (Enterprise archeive file)

→ It represents an enterprise application which may Contains Servlets, Jsps, EJBs, JNS Components e.t.c.

Various Commands :-

① To Create a jar file :-

```
jar -cvf durga.jar A.class B.class C.class
                   *.class
```

② TO extract a jar file :-

```
jar -xvf durga.jar
```

③ TO Display table of contents of a jar file :-

```
jar -tvf durga.jar
```

Ex :-

```
public class DurgaColorfullCalc
{
 public static int add(int x, int y)
 {
  return x*y;
 }
public static int add(int x, int y)
 {
  return 2*x*y;
 }
}
```

```
C:\> javac DurgaColorfullCalc.java
C:\> jar -cvf durgaCalc.jar DurgaColorfullCalc.class
```

```
class Bakasa
{
 p.s.v.m(——)
 {
 s.o.pln(DurgaColorfulCalc.add(10,20));
 s.o.pln(DurgaColorfulCalc.multiply(10,20));
 }
}
```

✗ D:\> javac Bakasa.java

✗ D:\> javac -cp c: Bakasa.java

✓ D:\> javac -cp c:\durgaCalc.jar Bakasa.java

✓ D:\> javac -cp .;c:\durgaCalc.jar Bakasa.

o/p:- 200
400

Note:-

→ when ever we are placing a jar file in the classpath Compulsary name of the jar file we should include, just Location is not enough.

Shortcut way to place jar file :-

→ If we are placing the jar file in the following Location then it is not required to set classpath explecitly bydefault it is available to Jvm & Java Compiler.

JDK
↳ JRE
↳ Lib
↳ ext
↳ *.jar

System properties :-

→ For every System persistance information will be maintain in the form of System properties. These may include O.S name, Varchual machine version, User Country .e.t.c....

→ We can get System properties by using getProperties() method of System class

Ex!- Demo program to print all System properties.

```
import java.util.*;
class Test
{
  public static void main(String[] args)
  {
    properties p = System.getProperties();
    p.list(System.out);
  }
}
```

→ We can set System property from the Command prompt by using -D option

Ex!- Java -Ddurga=SCJP Test

Space is not allowed

durga → name of the property

SCJP → Value of the property

9) JDK vs JRE vs JVM :-

JDK:(Java developement kit) :-

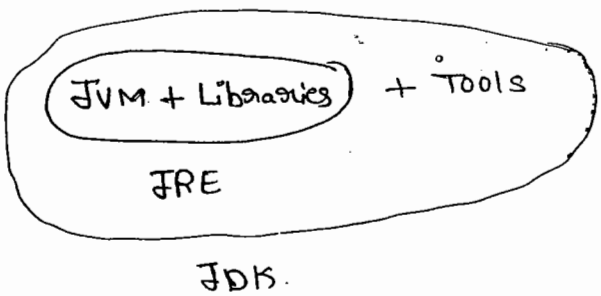
→ To develop & run Java application the required environment provided by JDK.

JRE:- (Java Runtime Environment) :-

→ To run Java application the required environment provided by JRE

JVM :-

→ this machine is responsible to execute Java program.



JDK = JRE + Tools

JRE = JVM + Libraries.

Note:-

→ On client machine we have to install JRE, where as on the developers machine we have to install JDK.

diff: b/w path & classpath :-

→ We can use classpath to describe the location where required .class files are available.

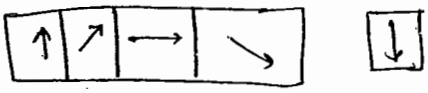
→ If we are not setting the classpath then our program won't be run.

Path :-

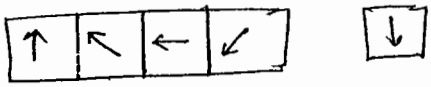
→ We can use path variable to describe the Location where required Binary executables are available.

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

3) Clockwise

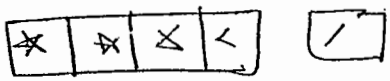


9) Anticlock

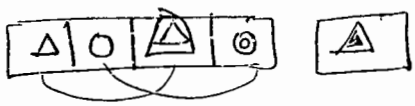


(3) Increasing

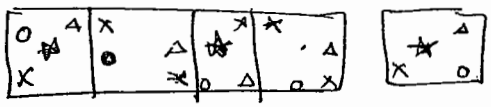
(4) Decreasing



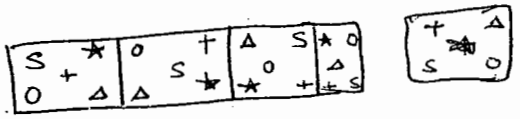
(5) Alternate



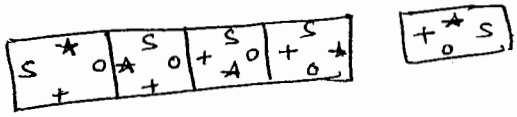
(6) multiple movement=



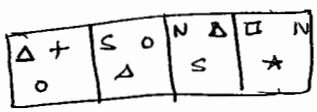
(7) Rotation



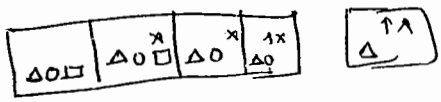
(8) Interchange

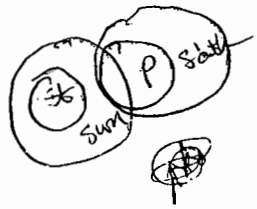


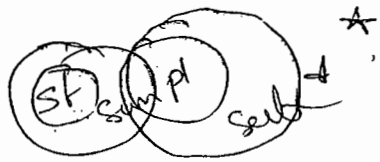
(9) missing of fig

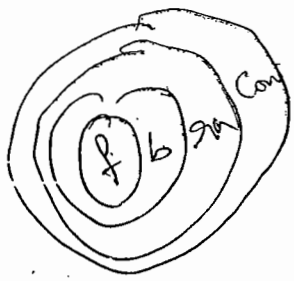


(10) Substitution









miracle

1.5V
- Autoboxing & Unboxing ✓
- generics ✓
- Vara-arg ✓
- For-each
- enum
- Annotations ✓
- Queue ✓
- Static imports / not recommended ✓
- Co-Varic of return types.

Walk
↓
Jogging
↓
Running
↓
Sprinting

Siddhartha (VNR VJIT)
9951884313
Siddharthahprs@yahoo.co.in

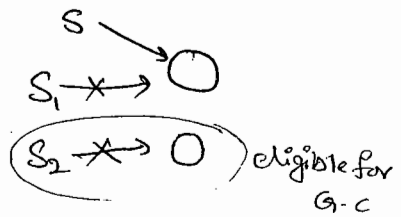
Vishnuteja.Y.S
9703340473, 9493410648
Vishnuteja87@gmail.com

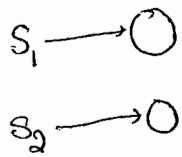
Vasu - 9179969444 (CEVMG)
Sivudsarma@gmail.com

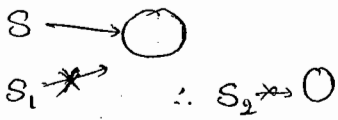
Ex 2:-

```
Class Test
{
 p.s.v.m(String[] args)
 {
  Student s = m1();
 }
 p.s.Student m1()
 {
  Student s1 = new Student();
  Student s2 = new Student();
  return s1;
 }
}
```

one object eligible for G.C →



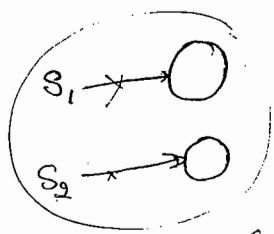




Ex 3 :-

```
Class Test
{
 p.s.v.main(String[] args)
 {
  m1();
 }
 p.s.Student m1()
 {
  Student s1 = new Student();
  Student s2 = new Student();
  return s1;
 }
}
```

Two objects eligible for G.C →

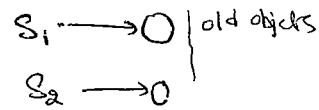


2) Reassigning the Reference variable :-

→ If an object is no longer required then reassign its reference variables to some other objects then that old object automatically eligible for G.C.

Ex(1) :-

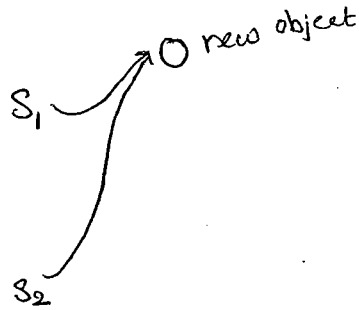
```
Student S1 = new Student();
Student S2 = new Student();
```



no object eligible for G.C →

```
S1 = new Student();
```

One object eligible for G.C →



Two objects eligible →

```
S2 = S1;
```

(3) Objects Created Inside a method :-

→ The Objects which are Created inside a method are by default eligible for G.C after Completing that method.

Ex :-

```
class Test
{
  p.s.v.main(String [] args)
  {
    m1();
  }
  p.s.v.m1()
  {
    Student S1 = new Student();
    Student S2 = new Student();
  }
}
```

2 objects eligible for G.C →