* Packages:

Packages are Java's way of grouping a variety of classes and/or interfaces together. The grouping is usually done according to functionality. In fact, packages act as 'containers' for classes. By organizing our classes into packages we achieve the following benefits:

1. The classes contained in the packages of other pgms. can be easily ~~accessed~~ reused.
2. In packages, classes can be unique compared with classes in other packages, ie, 2 classes in 2 different packages can have the same name. They may be referred by their fully qualified name, comprising their package name & the class name.
3. Packages provide a way to 'hide' classes thus preventing other pgms. ~~&~~ / packages from accessing classes that are meant for internal use only.
4. Packages also provide a way for separating 'design' from 'coding'. First we can design classes & decide their relationships, & then we can implement the Java code needed for the methods. It's possible to change the implementation of any method without affecting the rest of the design.

Java's packages are therefore classified into 2 types:

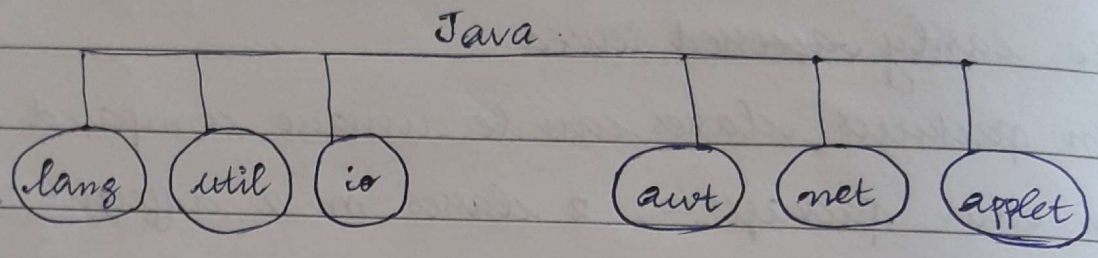
Scanned by CamScanner

(i) JAVA API packages

(ii) ~~ab~~ user defined packages.

- JAVA API packages :

" " provides a large no. of classes grouped into different packages according to functionality.



Functional breakdow of API packages that are frequently used in the pgms.

- User defined packages :

(i) Creating packages -

We can create our own packages. For this we first declare the package by using the keyword package.

SYNTAX : package package_name ;

NOTE : The package declaration must be the first line in the java source file. After package declara^n we define a class.

Scanned by CamScanner

PTO : Pg - 43

A package can have more than one class definition. Then only one class can be declared as public & the file is saved using this class name. When source file contains more than 1 class definition, java creates independent .class files for those classes.

Java supports the concept of package hierarchy. Then in the package declaration stmt. we specify multiple names.

package firstpackagename.second packagename;

* Accessing a package:
Java s/m package can be accessed ~~for~~ either by specifying the fully qualified classname or by using import stmt. The same approach can be used to access a user-defined package.

The import stmt. can be used to search a list of packages for a particular class.

SYNTAX:
eg:- import package1 [.package2][.package3] ... classname;

package1 is the name of the top level package & package2 is the name of the package inside package1 &

so on. After specifying all package name, finally class name is specified. This stmt. ends with ;

The import stmt. must be the first stmt. in pgm. It must appear before the class definition. In a pgm. we can use more than 1 / mul import stmt.

eg: import Sample.Exam;

After defining this stmt., can access any method directly by using the classname or its obje name.

import packagename.*;

Then compiler searches for the entire package name when a class name is encountered.

```
package package1;
public class classA
{   protected int a = 10;
    public void display()
    {  s.o.pln ("a=" + a); }
}
```

```
import package1.classA;
class Test1 { psvm ( String args[ ]) throws IOExcep
{ classA A = new classA();
  A.display(); } }
```

Scanned by CamScanner

Consider there are 2 packages & these 2 packages contain same class name.

```
package pack1;
public class Teacher { = }
public class Student { = }
```

```
package pack2;
public class Course { = }
public class Student { = }
```

If we import these 2 packages in a pgm.

```
import pack1.*;
import pack2.*;
```

```
Student s = new Student();
```

This stmt. will produce error, ∵ both packages contain class Student & compiler can't understand which one to use.

Then an explicit name is necessary

```
pack1.Student s1;
pack2.Student s2;
```

* Adding a class to a package:

We can add one more class to an existing package.

For this;

Scanned by CamScanner

1) Place the package stmt
   package packagename;
2) Define the class to make it as public.
3) Save the file by using class name to store in the directory packagename.
4) Compile the file.

eg: Consider P1 is an existing package to we want add a class Student to this package.

```
package P1;
public class Student {=}
```

Save the file Student.java in the directory P1. Compile Student.java file. This will create Student.class file in the directory P1.

Java source file can have only one class decl as public. The result is that the filename sho be same as that of public class name.

If we want to create a package with multi public classes in it:

- Decide the name of the package.
- Create a subdirectory with the name of the package.

Scanned by CamScanner

- Create classes that are to be placed in the package in separate source files & declare package stmt. at the top of each source file.
- Compile each source file. Then package should contain .class files of all source files.

* Hiding classes:

When we import a package the public classes are imported.

import package1.*; → All ~~packages~~ public classes of packages are imported.

We can hide some classes from accessing outside of the package. Such classes are not declared public.

eg: package P1;
public class classA { = }
class classB { = }

Here classB is not declared as public & is hidden from outside of package P1.

When creating package, the java source file contains only 1 public class & any no. of non-public classes.

import P1.*;
=

class obA; ✓
class obB; × classB is not available.

Scanned by CamScanner

* Static Import :

By using import stmt. we can import classes from a package to use them without qualifying the package. Like wise static import stmt. is used to import static members from classes to use them without qualifying the classname.

import static package name . subpackage name . classname . static member

import static packagename . subpackagename . * ;

eg: public interface salary
{ public static final int code = 101;
}

import static employee . salary ;
class salary
{ psvm { String args[]) { s.o.pln(" Code: " +
}
}

Scanned by CamScanner

```
package package_name;
public class classname
  {  =
  }
```

eg:
```
package Sample;
public class Exam
  { =
  }
```

Here Sample is the package name & the class Exam is considered as a part of the package Sample.

After typing the pgm, save the file by using the class name (Exam.java) & this file must be located in the directory named packagename (Sample).
Then the source file is compiled which creates & stores a .class file ~~&~~ in the same directory.
This directory will be a subdirectory of the directory where classes that will import the package are located.
Then the steps are;

1. Declare the package at the beginning of the file.
2. Define the class that we want to add the part of the package & declare it as public.
3. Create the subdirectory under the directory where the main source files are stored.
4. Save the file by using the class name.
5. Compile the file. This create .class file

Sub-directory name must match the package name.

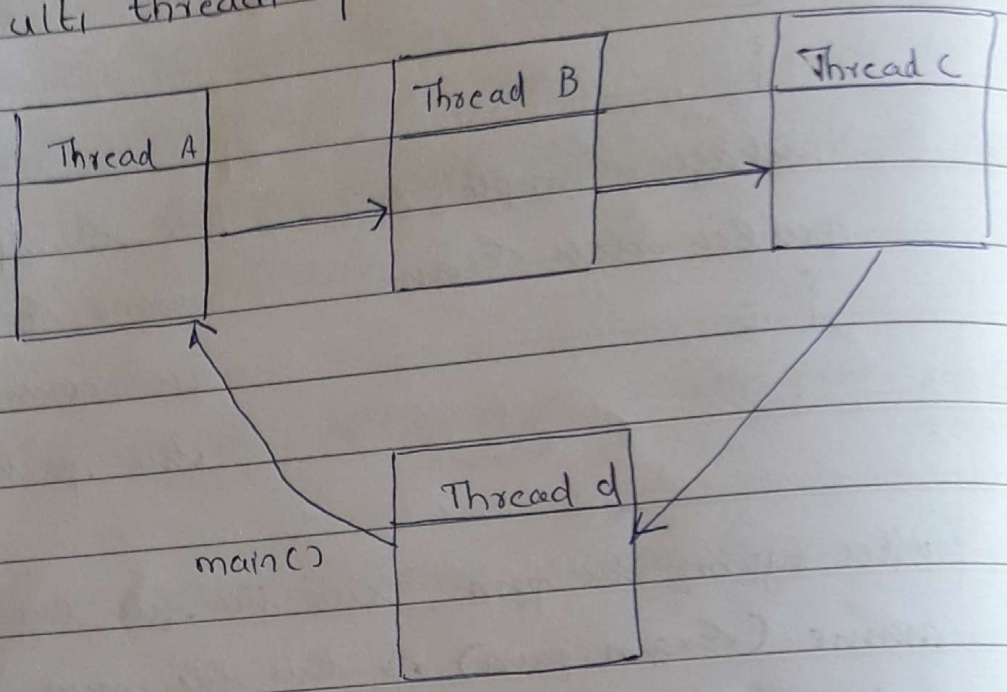
Scanned by CamScanner

MODULE : IV

Threads

06. 04. 2019

multi tasking
multi threading



time sharing
Light weight thread (∵ same mem. space / s/w)

Q. Difference b/w multithreading & multitasking

Multithreading	Multitasking
(i) pgmming concept in which a pgm. / process is divided into 2/ more subpgms. / threads that are ~~clas~~ executed at the same time in ||	It's an os concept in which multiple tasks are performed simultaneously.
(ii) It supports execn of multiple parts of a single pgm. simultaneously.	It supports execution of multiple pgms. simultaneously.

Scanned by CamScanner

(iii)	The processor has to switch b/w diff. parts / threads of a pgm.	The processor has to switch b/w diff. pgms. / processes.
(iv)	highly efficient.	less efficient in comparison to multithreading.
(v)	A thread is the smallest unit in multithreading.	A pgm. / process is the smallest unit in a multitasking env.
(vi)	It helps in developing efficient pgms.	It helps in developing efficient OS.
(vii)	cost-effective in case of context switching.	expensive in case of context switching.

Creating a thread

1) Derive a class from Thread class
2) " " " Runnable Interface

```
class classname extends Thread
{
    ≡
}
```

```
run()        public void run()
             {
                ≡
             }
```

Scanned by CamScanner

```
class A extends Thread
{ public void run()
  { int i;
    for(i=1; i<=5; i++)
    { s.o.println("class A");
    }
  }
}
class B extends Thread
{ public void run()
  {
    for(int i=1; i<=5; i++)
    { s.o.println("class B");
    }
  }
}
class C extends Thread
{
  public void run()
  {
    for(int i=1; i<=5; i++)
    {
      s.o.p("class C");
    }
  }
}
```

Scanned by CamScanner

```
class D
{   public static void main(String args[])
    {
        A obl = new A();
        B ob2 = new B();
        C ob3 = new C();
        obl.start();
        ob2.start();
        ob3.start();
    }                          new A().start();
}
```

class A	class A
class A	" C
class B	" C
class B	" C
class C	" B
class C	" B
class A	" A
class A	" A
class B	" A
class B	" B
class C	" B
class C	" C
class A	" C
class B	" A
	" B

Scanned by CamScanner

create a thread class by using runnable interface

```
class classname implements Runnable
{
        public void run()
        {
            ==
        }
}
```

```
Thread X = new Thread (obje name).
```

object name

```
class A implements Runnable
{
    public void run()
    { int i;
        for(i=1; i<=5; i++)
        {
            S.o.p("class A");
        }
    }
}
```

Scanned by CamScanner

=

```
class ThreadExample
{ public static void main(String args[])
  {
      A ob = new A();
      Thread ob2 = new Thread(ob)
      ob2.start();
  }
}
```

life cycle of Thread

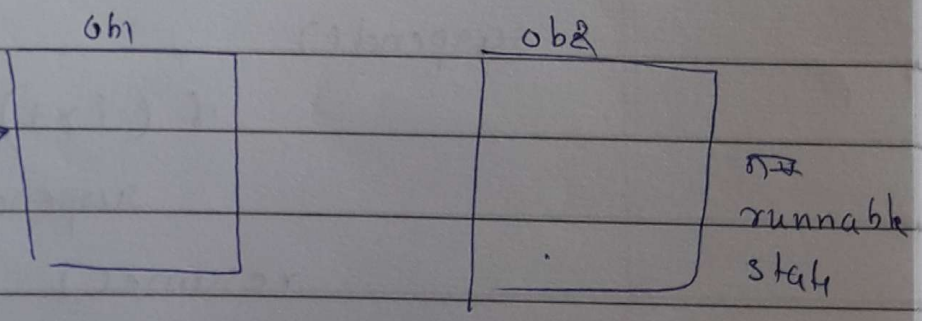
new born state()

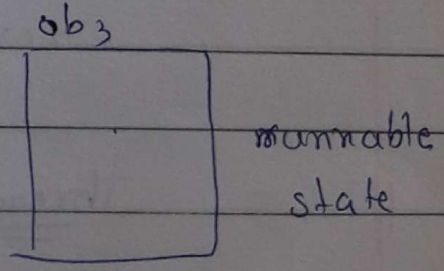
Runnable state

Running state

Dead state

Blocked state





Scanned by CamScanner

sleep()

```
public void run()
{ int i;
  for(i=1; i<=100; i++)
  {  s.o.p("class A");

     sleep(1000)
     =
  }
```

yield()

```
if (i==3)
    yield()        // can't resume
```

suspend()

```
if (i>3)
    suspend();
resume()
if (i==5)
    resume()
```

Thread priority

setPriority()

Scanned by CamScanner

Threadname . setPriority (Priority Number)

1 – 10

MIN_PRIORITY – 1

MAX_ " – 10

NORM_ " – 5

```
class A extends Thread
{
=
}
A obj = new A()
obj.setPriority(1); or obj.setPriority(MIN_PRORITY
```

```
class A extends Thread
{
    public void run Display()
    {
        {
        -
    }
}

class B extends Thread
{
    public B(A ob)
    {
        ob.start();
    }
}
B x = new B(new A())
```

Scanned by CamScanner

```
class A extends Thread
{ public void run()
  {   int i;
      for(i=1; i<=5; i++)
      {  s.o.p(i);
      }
  }
}

class B
{
    public B(A ob)
    {
        ob.start();
    }
}
class C
{     main()
      {
          B obl=new B(new A())
      }
}
```

Scanned by CamScanner

MODULE : IV

Java Stream Class

1 byte unidirectional — Byte Stream classes

Character Stream classes — 2 byte unicode bidirectional

Input / output Streams

Stream - connecⁿ / path / interface / object b/w source and destination (eg: memory & IO devices).

standard streams

standard input — System.in

standard output — System.out

standard error — System.err

BufferedReader

```
InputStreamReader isr = new InputStreamReader(System.in)
BufferedReader br = new BufferedReader(isr)
String s
System.out.print("Enter data");
s = br.readLine();
```

read()

This method is used to read a char.

variable = object name . read()

input stream object

read and return an int value

WAP to read a set of characters from keyboard until we input a char 'q'

Scanned by CamScanner

```
class    ExampleforStream
{
  pub
  {    char c ; string s=" "
       InputStreamReader isr = null
       isr = new InputStreamReader (System.in)
       System.out.print ("Enter chars
         one by one and enter q for
         exit ");

       do
       {
           c = (char) isr.read();
           s = s+c;
       }
  while ~~until~~ (c != 'q');
    System.out.println(s);
```

A
A
B
B
q
q

a
b
c
d
q

abcdq

Scanned by CamScanner

FILE

To open a file

Appropriate class is selected according to the purpose of the file

(charst) FileReader
(Bytest) FileInputStream } for input operation

(charst) FileWriter
(Bytest) FileOutputStream } " o/p "

using constructor we open the file.

File name is passed as argument.

using parameterized constructor

~~Output Stream~~ ~~FileWriter~~ f1 = new ~~Output~~ ~~Input Stream~~ FileWriter ("Text.dat")

~~File~~ OutputStream file1 = new FileOutputStream ("Sample")

write()

f1.write(ch)

f1.close()

WAP to read a character from keyboard and store that char in a file

{

InputStreamReader isr = new InputStreamReader (System.in)

System.out.println("Enter a char");

char c;

c = (char) isr.read()

Scanned by CamScanner

```
OutputStream fout = new FileOutputStream
                    ("Text.dat")

    fout.write((byte) c);
    fout.close();

  }

}
```

```
byte b[] = [10, 95, 20, 85]
OutputStream fout = new FileOutputStream
                    ("num.dat")

{ for (i=0; i<b.length; i++)
  { fout.write (b[i]);
  }
fout.close ()
```

```
~~fout~~ fout.write (b)
         fout.write (b, 0, 2)
```

Assume Sample.dat is an existing file. Display the content of the file

```
{
    InputStream fin = new FileInputStream
                      ("Sample.dat")
```

Scanned by CamScanner

```
byte x;
for( i=0 ; i< fin.available() ; i++)
{
        x= fin.read();
        System.out.print(x);
}
fin.close();

}
```

WAP to copy the content of one file to another file. Display the content of the file

```
{
    InputStream f1 = new FileInputStream("Sample");
    byte x; int i; char y;
    OutputStream f2 = new FileOutputStream("Text");
    for( i=0 ; i< f1.available() ; i++)
    {
        x= (byte) f1.read();
        f2.write(x);
    }
    f1.close();
    f2.close();
    InputStream f1 = new FileInputStream("Text");
    for(i=0 ; i < f1.available() ; i++)
    {
        y = (char) f1.read();
        S.o.p(y);
    }
    f1.close();
}
```

while (x = f1.read() != -1)
{ f2.write(x);
}

Scanned by CamScanner

read()

write()

finalize() – cleans up the connection b/w stream and file

```
String dirname= "/java/temp/user"
File d = new File(dirname)
d.mkdir()
```

File InputStream f = new ____

```
File f = new File("text.dat")
InputStream s = new FileInputStream(f)
```

Scanned by CamScanner

```
class Sample
{ private int a, b;
  public Sample ( int x, int y)
    {
      a = x;
      b = y;
    }
  =
}
class Test extends Sample
{
  private int c, d;
  public Test ( int x, int y, int p, int q)
    {
      super ( x, y);
      c = p;
      d = q;
    }
  =
}

Test T = new Test (10, 20, 40, 60);
```

Scanned by CamScanner