Unit-3

* Derived class declaration
* Types of inheritance
  - Java support single inheritance.
* Advantage of Inheritance.
* Implementation of Inheritance
* Super keyword, construction in derived class, eg. overriding, member accesibility.
* Diff b/w overloading & overriding
* Multilevel inheritance

pg: 225

→ Diff b/w static binding and dynamic binding

* Polymorphism / run time overriding
  - It means many forms or many type.
  - One operation can perform in different way in diff. classes.
  - A class that provide a signature of a method to a vanity of its derived classes to provide diff. implementation is known as polymorphic class.
  - So it is most useful when there are many hierarchically related classes with the method having same signature.
  - It specify what shell occure and not how it occure.

M. Imp Packages

PAGE NO.:
DATE:

- Polymorphism is achieve with the help of dynamic binding.

g.227 Ex

* ~~Extract~~ class

* Abstract class

→ Abstract method

Abstract void abc();

{

}

Note: There is no ~~nothing~~ plus parenthesis.

Ex

abstract class One

{

abstract void display(); // definition

}

class two extends one

{

void display()

{

System.out.print("Hello");

}

}

Class Main

{

```
public static void main (String s[])
{ Two t=new Two();
  t.display();
```

Rule 1

Note: The class must be abstract class if it contain abstract method.

- There is no body of abstract method

```
One o=t;
o.display();
}
}
```

Rule 2

Note: We can not create object of abstract class.

Rule 3 Implementate abstract method in different class.

→ It is a process of hiding implementation detail & showing only functionality to the users.

- Abstraction let you focus on what the object does instead of how it does.

→ 2 ways to achieve abstraction in java

1) Using abstract class
2) Interface

1) Abstract class

- A class that is declared as an abstract is known as abstract class.
- It can have abstract and non-abstract method.
- It need to be extended and its method need to implement.
- It can not be instantiate.

* Interface :

- It is a blueprint of a class.
- It has static constant and abstract method.
- It is a mechanism to acheive abstraction.
- There can be only abstract method in interface not a method body.
- It also represent is-a relationship
- It can not be instantiate just like a abstract class.

→ Use of java interface :-

- To acheive abstraction
- Support the functionalities of multiple inheritance.
-

→ class 1 ↑ extends class 2 — interface ↑ extends. interface — interface ↑ implements class

Note: The name of the interface should be starts with I.
- We need to initalize variable in interface.
- The methods and variables in interface should be public.

Eg:

```
interface Demo
{
    public static final int a=10;
    public abstract void printMethod();
}
interface Demo
{
    int a=10;
    void printMethod(); // abstract method
}
```

Eg:

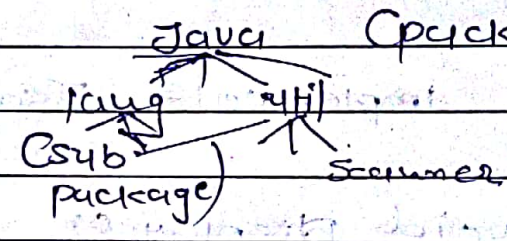
```
interface Isample
{
    int x=10;

    void Display();
}
class temp implements Isample
{
    public void display()
    {
        public static void main(String
        System.out.print("Hello"+x);
    }
}
class Main
{
    temp o=new temp();
    o.display();

}
```

* Package :

- It is a group of similar type of classes, interface and sub-package :
- Java package can be classified into 2 parts,
  1) building package
  2) User-define : package.

1) building package

→ java.util.Scanner; {fully qualified}
(java → package, util → package, Scanner → class)



→ Advantage of Java package

- Used to categories the classes and interface, so that they can be easily maintained
- It will provide access protection
- It will remove naming collision

javac -d . abc.java

abc.java

Documentation.

directory

package (xyz);

current dir.

import java.util.*;

eg: class Person

{

string name;

int age;

Person()

{

name = null;

age = 0;

}

void Getdata()

{

Scanner sc = new Scanner(System.in);

name = sc.nextLine();

age = sc.nextInt();

}

void put()

{

}

}

class Package

{ public static void main (String args)

{

}

) Compile code

javac -d . abc.java

) Run :

java xyz.classname

packagename

- To compile a program:

javac -d . abc.java
e: (in e drive)

where, -d created directory
. represent current dir
abc.java filename

→ Run a program

java ayz.package

where,
ayz is a package name
which package.class which lo

→ class file in another directory

compile ∴ javac -d e: abc.java

(It will create ayz package in e drive)

- run

- we need set cl class path from current dir to e drive

set classpath = e:\ ;

- Now we can run a program with java ayz.Package Classname)

How to access package from another package :

→ 1) import packagename.classname /** ;

ex import mypackage1.Person ;

→ 2) import packagename.* ;

ex import mypackage1.* ;
// It will importing all the classes from pkg

→ 3) Fully qualified name (without import statem

pkgname.classname objname=new pkgname.classname()

ex mypackage1.Person obj=new mypackage1.Person()

Note: If you use import packagname.* ; then all the class and interface of that package will be accesible but not sub-packages.

* Three types of error
  1) Syntax error
  2) Semantic "
  3) Run-time "

* What is an exception ?
  - Abnormal condition will represent as exception.
  - OR
    It is an event that distyrb the normal flow of the program.
  - Two types :
    1) Checked exception
    2) Unchecked "

* What is exception handling ?
  - It is one of the powerful mechanism to handle run time error so that normal flow of the application can be maintain

ex

class not found
IO exception

* Advantage:
- Maintain the normal flow of program.

→ During the program execution and unexpected situation such as, inability to find a file or problem in a network connectivity or division by zero or Array index etc may arise such situation are known as exceptional event which we can handle using exception handling

→ Hirarchy of

Throw

* Hierarchy of exception

Throwable class

exception	error
1) IOException	1) StackOverflowError
2) SQLException	2) VirtualMachineError
3) ClassNotFound exception	3) OutOfMemoryError
4) Run-time exception	{ code cannot handle these errors }
- Arithmetic "	{ It is irrecoverable }
- Null-pointer "	
- NumberFormat "	
- IndexOutOfBound "	
- Array index out of bound	
- String index out of bound	

* Types of Exception

1) checked exception
2) Unchecked "
   (Array is Unchecked exception).

* Diff b/w checked and Unchecked

Checked	Unchecked
- The class that extend throwable class except run-time exception and error are known as checked exception.	The classes that extend run-time exception are known as unchecked exception
eg:- IoException, SQLException, classnotfound.	eg: Arithmatic, null-point, Index Out of bound
- Checked Exception are checked at compile time.	Unchecked Exception are not checked at compile time it will checked at run-time.

* How to handle exception (Exception handling).

→ Java exception handling is managed via 5 keyword:

1) Try
2) Catch
3) Finally
4) Throw
5) Throws

→ 1) Try :

- A program statement that you want to monitore for exception are contain within Try block.
- If an exception occures within the Try block is thrown.

Ex Try
{
code
}

→ 2) Catch :

- Your code can catch thrown exception using catch block and handle it in some rational manner.
- It must be used after the try block only
- You can handle multiple catch blocks with a single try block
- It is an optional

→ 3) Finally :

- Any code that must be executed before termination of a program is put in finally block.
- We can use break and continue but not return statement.

try - catch
try - finally
try - catch - finally.

PAGE NO.:
DATE: / /

- It is an optional block.

Note: At a time only one exception is occured and at a time only one catch block will execute.

- All catch block must be in order that is from most specific to general

Ex: catch for arithmatic exception must come befor catch for exception.

*	throw	throws
-	Followed by object	- Followed by classname
-	within method	- with method
-	Ex: throw e;	- Ex: void Display() throws Arithmatic.e
-	used explicitly	- used to declare an exception
-	Can not throw multiple exception	- Can use multiple exception.

e: Only class object is used with throw.

ex

```
DemoExceptHow()
{
    try
    {
        int a[] = {1, 2, 3};
        System.out.print("value" + a[6]);
    }
    catch(Exception e)
    {
        sop("Error");
        throw e;
    }
    finally
    {
        System.out.print("Error Displayed");
    }
}
```

DATE: / /

* User define Exception

Ex

```
Import java.lang.*;
class Balauce extends Exception
{
    int a;
    Balauce (int amt)
    {
        a= amt;
    }
    public string tostring()
    {
        String str = "Enough Amount Not in
                      Account";
        return str;
    }
}
public class UserDefineException
{
    public static void main(String[] a)
    {
        Scanner s=new Scanner(System.in);
        int i,j,k;
        int amt= 10000;
        try
        {
            sop("Enter Amount to withdrawn");
            i= s.nextInt();
```

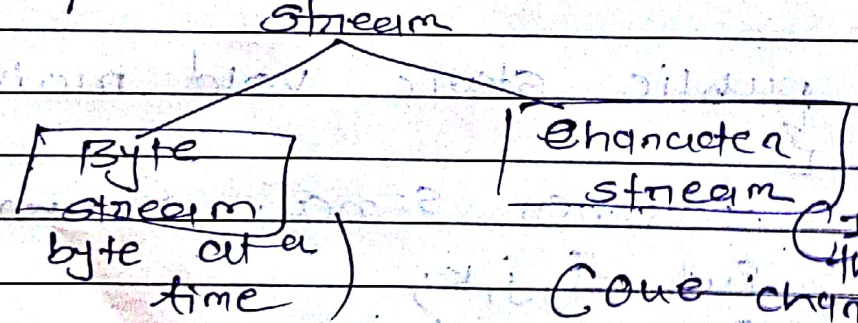
-> tostring() return string value

```
if (i > a.length)
{
    // throw new Balance();
    Balance b = new Balance(i);
    throw b;
}
else
{
    print(a[i-1]);
}
}
catch (Balance e)
{
    print("Error raised");
    print(e);
}
}
}
```

same we can use only one of them

will call tostring() method

/* Stream / Java stream :



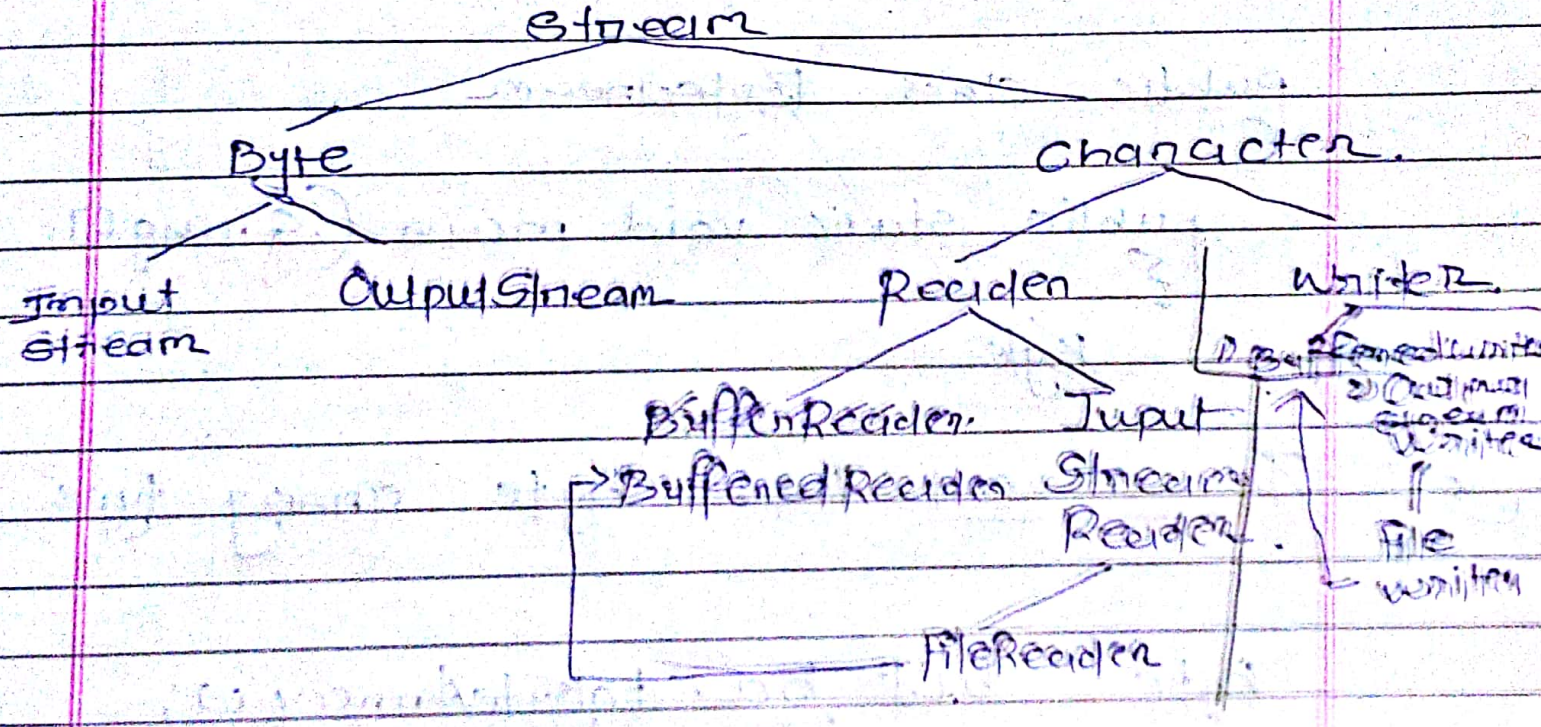
-> Stream is used to take input from any sequence.

→ Byte value of A is 65

PAGE NO.:
DATE: / /

→ It represent ordered sequence of data
- It represent common characteristics shared by all I/O devices.
- It is easy to used use object oriented interface b/w program and I/O devices.
- Java.IO package contain a fairly large number of classes.
- Java categories these stream into two large series based on that ability to deal with diff data.

2) Byte
2) Character.



DATE:

* Byte

0 1 2 3 4 5 6

→ Byte b[] = { 65, 66, 67, 68, 69, 70, 71 }

Read (b[], 2, 3) → length
start with
offset index total

write

> abstract void write (int b)
void write (byte[])
void write (byte [], int offset, int leng

ex import java.IO.*;

public class Bytestream
{
public static void main (String [] a
{
Byte

→ To convert into byte array type

byte b[] = ba.toByteArray();
object of class function

* Convert to character type

print ((char) b[i] + " ");

* Reading

ByteArrayInputStream bi = new ... (b);

int c;

while ((c = br.read()) != -1)

{

print ((char) c);

}

* ba.write (65);

ba.write ("abc".getBytes());

convert into byte.

Eg:

```
public class ByteStreamTest
{
    public static void main (String [] args)
                                throws IOException
    {
        ByteArrayOutputStream ba = new
                                    ();

        ba.write (65);
        ba.write (45);
        ba.write("abc".getBytes());

        byte b[] = ba.toByteArray();

        for (int i=0; i<b.length; i++)
        {
            print((char) b[x] / b[x]+' ');
        }

ByteArrayInputStream bi = new
                                    ();
    int c;

    while ((c=bi.read()) != -1)
    {
        print ((char) c);
    }
}
```

DATE: / /

* Character Stream

→ Take Input: BufferReader class.

↓

ReadLine()

{always take input in String}.

→ Execution of Buffer class is quiet faster than Scanner class.

→ Diff b/w BufferReader and Scanner.

Scanner	BufferReader
- It is not synchronous	- It is synchronous.
	- larger buffer

- 1) BufferReader is synchronous while Scanner is not. BufferReader should be used if we are working with multiple threads.

2) BufferReader has significantly larger buffer memory than Scanner.

3) The Scanner has a little buffer (1 KB char buffer) as opposed to the BufferReader (8KB byte buffer), but it's more than enough.

Note: if we use java.io package then it will raise IOException and we need to handle it manually in code. using throws

4) BufferReader is bit faster as than Scanner because Scanner does parsing of input data.

5) Issue with scanner when nextLine is used after nextXxx():

```
import java.io.*;
class Friend
{
    Friend()
    { try {
        InputStreamReader obj=new
                    InputStreamReader(System.in);
        BufferedReader br=new BufferedReader(obj);
        print("Name");
        name=br.readLine();

        print("Number");
        control=Integer.parseInt(br.readLine)
        } catch(IOException e)
        { print("Raised"+e); }
    }
    void Display()
    {
        ...
    }
main class
{
}
```

try catch block.

PAGE NO.:
DATE: / /

* FileReader & and FileWriter

→ To write - FileWriter.

→ FileWriter.

Ex- class FilewriterDemo
{
public s v m (s[]) throws Exception
{
try
{
FileWriter f = new FileWriter("File.txt", true);
f.write(" Hello ")
f.write(" BMIIT ");
for (int i=1; i<=10; i++)
{
f.write(i);
}
fw.close();
}
catch (Exception e)
{

(true - to existing file append the content)

Problem

File processing

```
try
{
    FileWriter f = new FileWriter("abc.txt");
    f.write("UTU");
    f.write("BHIIT");
    for (i=1; i<10; i++)
    {
        f.write(i);
    }
    f.close();
}
catch (IOException e)
{
}
```

f.write(" "+i); — converts string

or

f.write(Integer.toString(i));

ex

```
import java.io.*;
class FileWriter
{
    psvm
    {
        FileReader f1 = new FileReader("file");
        int c; count++;
        while((c=f1.read()) != -1)
        {
            System.out.print((char)c);
```

PAGE NO.:
DATE: / /

print Integer value with writer:

1) f.write("" + 123);
   (convert to String)

2) f.write(Integer.toString(123));
   ↳ use wrapper class.

3) f.write(String.valueOf(10.1));

→ FileOutputStream (Byte)
FileWriter (Character)

or Copy the data from one file to another

```
import java.io.*;
class FileWriter
{
  public static void main(...) throws Exception
  {
    FileReader f1 = new FileReader("BMIIT.txt");
    int c;
    FileWriter f2 = new FileWriter("CVR.txt");
    while ((c = f1.read()) != -1)
    {
      f2.write((char) c);
    }
    f1.close();
    f2.close();
}
```

Terminate the program :- 1) return;
2) System.exit(0); value.

File Management

- With the help of File class one can manage a file like, find file in a certain director, rename some files, delete a file, create a temporary file, create a directory, create a new sub-directory etc..

→ for create a file:

ex File f = new File("Demo.txt");
f.createNewFile();

→ File f = new File("Demo.txt"); // this Constructor will not create a file

→ Methods

[1] f.createNewFile();

- It will create a new file it does not exists

[2] f.exists();

- It will return boolean value: true if file exists and false if file is not exists.

Eg:
```
if (f.exists())
{
    print("File is exists");
}
else
{
    f.createNewFile();
}
```

[3] f.delete()

- It will return a boolean value true after delete is completed otherwise returns false.

[4] f.getAbsolutePath()

- It will give full path from root to current directory.

[5] f.mkdir()

- It will return a boolean value. true
- It will create Demo directory in current directory.

Eg:
```
boolean b = new File("Demo").mkdir();
if (b)
{
    print("Created");
}
else
{ print("Not Created"); }
```

[6] f.isDirectory()

- Ex File f1 = new File("Demo")
  " f2 " " C"\"."txt")
  if (f1.isDirectory());
  { }

- return a boolean value, if respected one is directory then return true else false.

[7] new File("parent"+File.separator+"child").mkdirs();

- Return a boolean value.
- It will create a Directory as a "parent" and create a sub-directory as 'child'.

1) f.f File Separator

- used to Seprate a directory
  for windows ("\")
  for other os ("/")

[8] f.list();

PAGE NO.:
DATE: / /

Eg File f1 = new File("Unit1");

```
if (f1.isDirectory())
{
    print("Dir");
String str[] = f1.list();
    for (int i=0; i<str.length; i++)
    {
        File f = new File(str[i]);
        print(str[i]);
print(f.getAbsolutePath);
    }
```

Unit-6

* JDBC :- Java Database connectivity.
- JDBC & API which consist of set of java interfaces classes which need to connect, store & query the data.

→ Advantage:
It is very flexible and portable solution that don't depend on any specific data base system.

- JDBC API relies on JDBC drivers.

* What is drivers?
- A drivers are just plugable items, they can replace existing driver with another drivers without having to derive the application ~~part~~ code.

- So to make the database system work with java specific implementation of JDBC driver for that database system must be provided to application developer.

-> JDBC Interface that programmer generally deal with following :-

1) Driver Manager :
- Driver manager object is the basic service to manage.
1) JDBC driver & Initialization
2) Connection object.

PAGE NO.:
DATE: / /

2) Connection class :

- A Connection object controlling the connection with data source or driver manager.
- All the statement are created through Connection object.

Use :

1) Establish the connection
2) Execute the statement

3) Statement

- Statement object including statement prepared statement and callablestatement object can be use to execute query & store procedure.

4) Resultset

- A resultset object through with a query result is return by statement object
- It contains Row of the table in the database and provide method to access that.

PAGE NO.:
DATE:

* JDBC Driver :

- JDBC driver is a SW component enable Java application to insert with the database.

1) JDBC . ODBC Bridge Driver

- Open database Connectivity) ODBC
- This driver class uses ODBC drive to connect with database. Those are open source.
- That converts JDBC method call to ODBC function call.

→ Advantage :

- Easy to use.
- can be easily connect to database

→ Disadvantage :

- Performance will decrease, because JDBC method will convert Into ODBC function call.
- ODBC driver need to be install to the client machine.

2) Native API driver

- The driver uses client side library of the database such as database sql server or oracle.
- The driver convert method call into native code in API. Not entire in Java.

→ Advantage.

- performance is upgraded than JDBC & ODBC

→ Disadvantage

- Native Driver need to be install each client machine.
- The client library need to install client machine.

3) Net Net-protocol driver

- It is uses middle server then translate the request to the database specific request to the database server.
- It is fully costy in Java.

→ Advantage

- No client side library is required because application server can perform many coeill of it like load balancing, loging etc.

→ Disadvantage

- Network support is required on client machine.
- Required database
- Maintainance of network protocol driver is costly.

4) Thin driver:

- It convert JDBC call to the under specific database protocol. That way it is called thin driver.

→ Advantage

- Better performance than all other driver.
- No s/w required client side or server side.

→ Disadvantage

- Driver depend on database

* 5 steps

(1) Register the driver class
- The forName() method of class is use to the register the driver class.
- This method is use to dynamically load the driver class.

→ Syntax :

public static void forName(String className)
throws classNotFoundException

Ex

ClassName.forName("com.mysql.JDBC.Driver");

[The driver class for the mysql database is com.mysql.jdbc.Driver]

(2) Creating Connection :

- getConnection() method of driver manager class is used to establish connection with database

Syntax :

public static void Connection getConnection
(String url) throws sqlException.

Ex Connection con = DriverManager.getConnection
("jdbc:mysql://localhost:3306/test",
"root", "admin");

- To identify hostname

SHOW VARIABLES WHERE Variable_name = "hostName";

→ Here, Jdbc an API, mysql is database
localhost is a server name on which

mysql is running test is database name is username admin is a password

(3) Creating statement :

- Createstatement() method of connection interface is used to create statement The object of statement.

Syntax :

public Statement createstatement() throws SQLException

Ex: Statement st = con.CreateStatement();

(4) Executing queries :

1) To execute select query.

→ public resultset executeQuery(String sql);
{It will return object of result set}

2) To execute specific query create, delete, update etc.

public int executeUpdate(String sql);

3) To execute a multiple query.

public boolean execute (String SQL)
{ It will return multiple data }

(5) Closing connection

- By closing connection object statement and Resultset will be closed automatically. The close() method of connection interface is used to close connection.

- public void close() throws SQLException