9/8/25

Module : 3

3.1 Packages
3.2 Interfaces
3.3 Exception handling

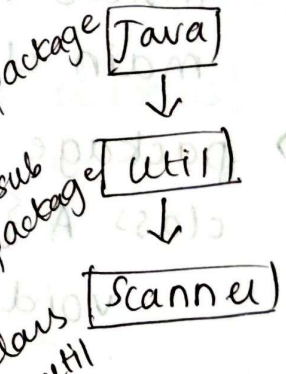
Packages

- contains related classes
- for maintainence, reduce naming conflicts
- like a folder

eg:
```
package myPack;
class Calculate{
        p.s.v.m(s A){
                s.o.p("Calculating");
        }
}
```

- Two types of packages
i. User defined
ii. Built in

Example for packages
- import java.util Scanner;
  package, sub package, class



- A package contains subpackages, classes, interfaces.
- used for reusing code.

Q. How to import a package from library?
→ Using 'import' keyword
ie, import package.class name

Three ways to access a package from outside the package.

1. import package.*;
   - all particular classes and interfaces in the package will be imported.
   - sub packages will not be imported (by default)

   ie, import java.*; / import classesand interfaces
   import java.util.*; / also include subpackage util.

2. import package.subpackage.classname;
3. Fully qualified name;

Q.2) Create a package 'pack'. Inside of pack create a class 'A' which prints "Hello".
Create another package 'myPack', Here import pack which is an already created package and inside myPack, create a class B which is a main class that access ~~A~~ class A in pack

→
```
package pack;
class A {
    void msg() {
        s.o.p("Hello");
    }
}
```

A.java
A class
pack

```
Package myPack;
import packA;
class B {
    p.s.v.m(string[] args){
    A a=new A();
```

B.java
B class
myPack

```
    a.msg();
}}
```

Q. Java prgm to check if a string is palindrom

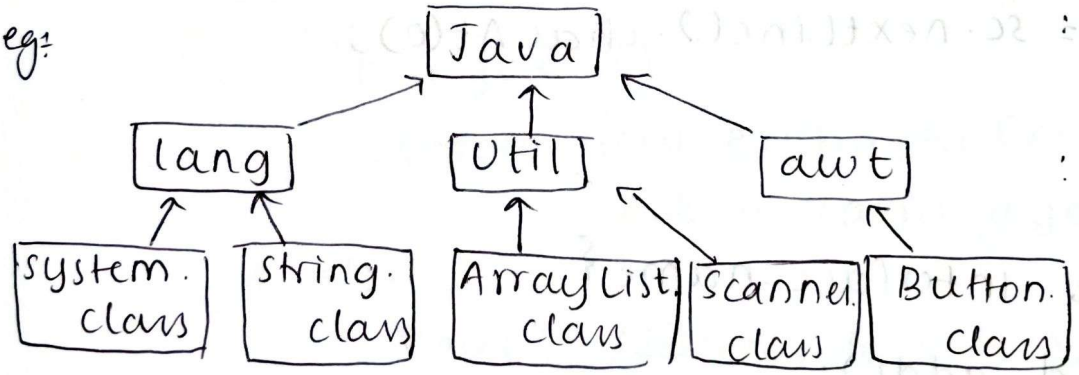
→

14/8

Uses of Packages

- Easy access
- Reduce naming conflicts
- Group related classes and interfaces.
- for encapsulation
- Reuse code

eg:



: Package

: Subpackages

: classes

3) Fully qualified name

eg:

```
package pack;
public class A {
    P
```

```
package myPack;
class B {
    p.s.v.m(...) {
```

→

```
class MyClass{
    p.s.v.m (s A){
        s.o.p ("Enter a name");
```

```
java.util.Scanner sc = new java.util.Scanner(
                                    System.in);

String s = sc.Next
String = sc.nextLine();
S.O.P ("You entered" + name);

String name = sc.nextLine();
S.O.P ("You entered" + name);
}}
```

Taking inputs

- `String s = sc.nextLine();`
- `int i = sc.nextInt();`
- `char c = sc.next().charAt(0);`
  (or)
  `char c = sc.nextLine().charAt(0);`

Interface

eg:
```
interface interface-name {
        void add();
        void sub();
}
```

Here, - methods add and sub is abstract
- no need to write abstract add();
- attributes and methods are abstract, static, public.

Multiple Inheritance

- More than one parent having one child class.

class MyClass {
p.s.v.m(String

18/8/9

Exception Handling

```
import java.io.*;
```

⇒ Throw keyword

- exception handling keyword
- it can handle both checked and unchecked exception.
- Declare:

```
throw new Exception_name("Msg to display");
```

- throw is used to 'explicitly' throw an exception.

Eg:

```
public class ThrowExample{
    static void validAge(int age){
        if (age<18)
            throw new ArithmeticException("
                not a valid age");
        else
            sout("Welcome to vote");
    }
    p.s.v.m(String args[]){
        validAge(13);
    }
}
```

⇒ In this example, the throw is handled by JVM.

[OR]

```
p s vm (String args[]){
    try{
        validAge(13)
}
```

```
        catch (Exception e){
            SOP("Exception caught "+e)
        }
    }
```

19/9

⇒ Throws keyword

- handle only ~~un~~checked exception (compile)
- to declare an exception

Syntax

```
return_java method_name() throws
                exception_class_name {
    // method code
}
```

Checked exceptions

1. throws } }
2. try } }
3. catch } }

eg:

```
class throwsExample{
    public void read() throws IOException{
        throw new IOException("Exception is IO");
    }
    p.s.v.m (S A){
        try{
            throwsExample t=new throwsExample();
            t.read();
        }
        catch (Exception e){
            s.o.p ("IO Exception")
        }
}}
```

Java Input & Output and Files

- Stream: seaquence of bytes/data.
- API: application program interface
- Java I/O used to process input and produce o/p.
- File handling done using Java I/O API.
- Using stream concept makes I/O faster.

Three streams

1. System.out : standard output stream
2. System.in : " input "
3. System.err : " error "

```
eg: public class MyClass {
        p.s.v.m (S A){ throws Exception{
            int a = System.in.read();
            S.O.P (a);
        }
}
```

o/p ⇒ A
65

eg ⇒ If "throws IOException" is to be used,
import "java.io.*".

```
eg: import java.io.*
    public class MyClass {
        p.s.v.m (S A){
            try{
              int a = System.in.read();
            } S.O.P (a);
            }
            catch(Exception e){
    }}}
```

⇒ to convert int → char

```
int a= system.in.read();
char c = (char) a;
s.o.p(c);
```

⇒ To convert a string : use bubber reader class or scanner class.

Output stream class

- is an abstract class

Useful methods:

1. public void write(int) throws IOException.
2. public void writebyte ([]) throws IOException
   - used to write an array of bytes to the current output stream.
3. public void flush() throws Exception
   - flushes the current o/p stream.
4. public void close() throws Exception
   - close the current o/p stream.

Swing Fundamentals

- Java swing is a GUI framework
- import javax.swing
- more powerful and flexible than AWT
- platform independent and lightweight
- container : where components (label, i/p box, buttons etc) are placed.
- used to create window-based applications
  eg: MS office, calculator

⇒ Containers : AWT

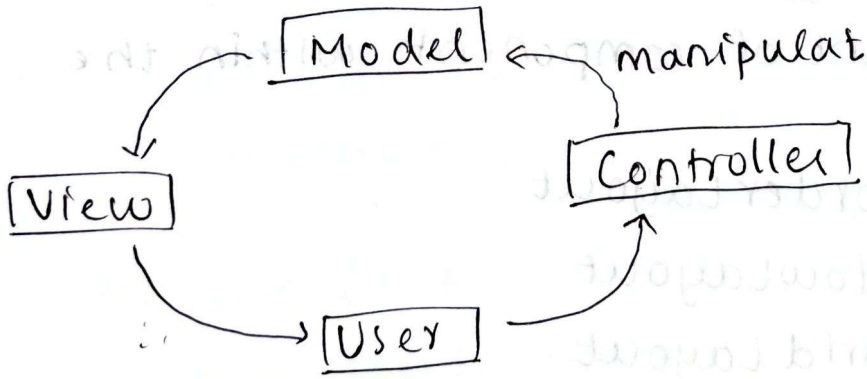
Components : swing

AWT	Swing
• AWT components are platform dependent	• Java swing components are platform independent
• heavyweight	• lightweight
• doesn't support pluggable look and feel.	• supports
• provide less components	• more components
• doesn't follows MVC (model view controller)	• follows MVC

Model View Controller

Three components:

1. Model : manages data, logic and rules of the applicatn
2. View : used to present data to user.
3. Controller : accept i/p from user and converts it to



- Components : independent visual control
- Container : eg : JFrame, JApplet, JWindow, JDialog.

Eg :
```
import java.swing.JFrame;
import java.swing.SwingUtilities;
public class Example extends JFrame {
    public Example(){
```

```
        setTitle("Simple Example");
        setSize(300, 200);
        SetLocationRelativeTo(null);
        SetDefaultCloseOperation(JFrame.EXIT_ON_CLOSE
    }
    public static void main(s A){
        Example ex=new Example(); } or new Example();
        ex.setVisible(true);
}}
```

Event handling in swings [change in state of an object]

- the object created when user interacts ~~with the~~ is known as an event
- the object is passed to a listener, the listener is responsible for handling the event
- Event handlers for input, button, textbox etc. are different.

Swing Layout Manager

- arrangement of components within the container.

1. java.awt.BorderLayout
2. java.awt.FlowLayout
3. java.awt.GridLayout
4. java.awt.CardLayout
5. java.awt.GridBagLayout
6. java.swing.BoxLayout
7. java.swing.GroupLayout
8. java.swing.SpringLayout

Q.1) write a Java prgm to create two buttons labelled as YES & NO on a JFrame in flow layout arrangement.

→
```
import javax.swing.*
import java.awt.event.*
import java.awt.*

class TestSwing extends JFrame {
    TestSwing(){
        SetTitle("Test_Swing_Example");
        Set Size(300, 200);
        set Location Relative To(null);

        JButton jbt1=new JButton("Yes");
        JButton jbt2=new JButton("No");

        SetLayout(new FlowLayout());
        set DefaultCloseOperation(JFrame.EXIT_ON_
                                  CLOSE);

        add(jbt1);
        add(jbt2);
        set Visible(true);
    }
    p.s.v.m(s A){
        new TestSwing();
    }
}
```

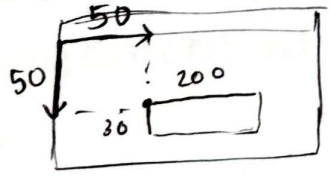
2) write a java prgm to place a textbox on a JFrame of size (400, 400)

→
```
import javax.swing.*
//import java.awt.* & no need
```

```
class TextBox extends JFrame {
    TextBox{
        setTitle(" Test_Field_ Example ");
        setSize(400,400);
        setLocationRelativeTo(null);
    JTextfield jf1=new JTextfield(25);
    setLayout(new FlowLayout());
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE)
    add(jf1);
    setVisible(true);
    }
    p.s.v.m(s A){
        new TextBox();
}}
```

Adding Labels

label.SetBounds(50,50,200,30)



Jbt1.SetBounds(20,20,50,30);

26/9

3)
```
import javax.swing.*;
class SLabelDemo1{
    p.s.v.m(s A){
        JFrame jf=new JFrame("studytonight");
        JLabel label1, label2;

        label1 = new JLabel("Welcome to
                            studytonight.com");
```

```
label2 = new Jlabel("How are you");
label1.setBounds(50, 50, 200, 30);
label2.setBounds(50, 100, 200, 30);

jf.add(label1);
jf.add(label2);
jf.setSize(300, 300);
jf.setLayout(null);
jf.setVisible(true);
}
```

Two types of Event

1. Foreground event :
   - due to direct interaction of the user.

   eg: clicking on a button, moving mouse, selecting something from list, scrolling down.
2. Background event
   - doesnot require interaction of the user.

   eg:

Delegation Event Model : key participants are

a) source : object on which event occurs (will be "class")

b) Listener : event handler (will be interfaces)

How events are handled?

1. source generates an ~~ob~~ event by creating an object
2. Event passed to one or more listener
3. Listener processes and returns

Event classes and interfaces

Event classes	Description	Listener Interfa
1. Action event	generated when button pressed, menu item selected, list item double clicked	Action Listener
2. Mouse event	mouse dragged, moved, clicked, pressed or released / enter or exit a component	mouse listener
3. key event	i/p received from key board	key listener
4. item event	checkbox / list item clicked.	item listener
5. text event	value of text area or textfield is changed	text listener
6. mouse wheel event	mouse wheel is moved.	mousewheel listener
7. window event	window activated or deactivated, opened or closed.	window listener
8. component event	component is hidden, moved, ~~released~~ resized or set visible.	component listener
9. container event	component added or removed from container	container listener
10. Adjustment	scroll bar is manipulated	
11. Focus event	component gains or loses key focus.	

→ Event Sources : button, check box, choice, list, menu item, scroll bar, text components, window

→ Event listeners : Action l., adjustment l., component l., container l., focus l., item l., key l., mouse l., mousemotion l., Mouse-wheel l., Text l., window Focus l., window l.

Q4) Write a java prgm to place a text box on a container (Applet). The text entered in the textbox should come as a label on the container. (no main method for applet prgm)

→
```
import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
class MyApplet extends JApplet implements
                        KeyListener{

    JTextField jtf;
    JLabel label;

    public void init(){
        setSize(600,300);
        setLayout(new FlowLayout());
        jtf = new JTextField(20);
        add(jtf);
        jtf.addKeyListener(this);
        label = new JLabel();
        add(label);
    }
    public void keyPressed(KeyEvent ke){}
    public void keyReleased(KeyEvent ke){}
    public void keyTyped(KeyEvent ke){
```

```
label.setText(String.valueOf(ke.
                    getKeyChar()));
```

method in converts i/p into string

```
}  }
}
```

Q.5) write a java prgm to add a button on a j frame labeled 'Click here'. when it is clicked, a name should be displayed.

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

class button extends JFrame implements
                    ActionListener{
JButton b=new JButton("Click here");
SetSize(400,400);
public button(){
    JButton jbt1=new JButton("Click here")
    JLabel label=new JLabel("Name print");

    setsize(600,300);
    add(jbt1);
    void actionPerformed(ActionEvent e){
        if(e.getSource=buttonobj){
            label.setVisible(true);
    }  }
}
```

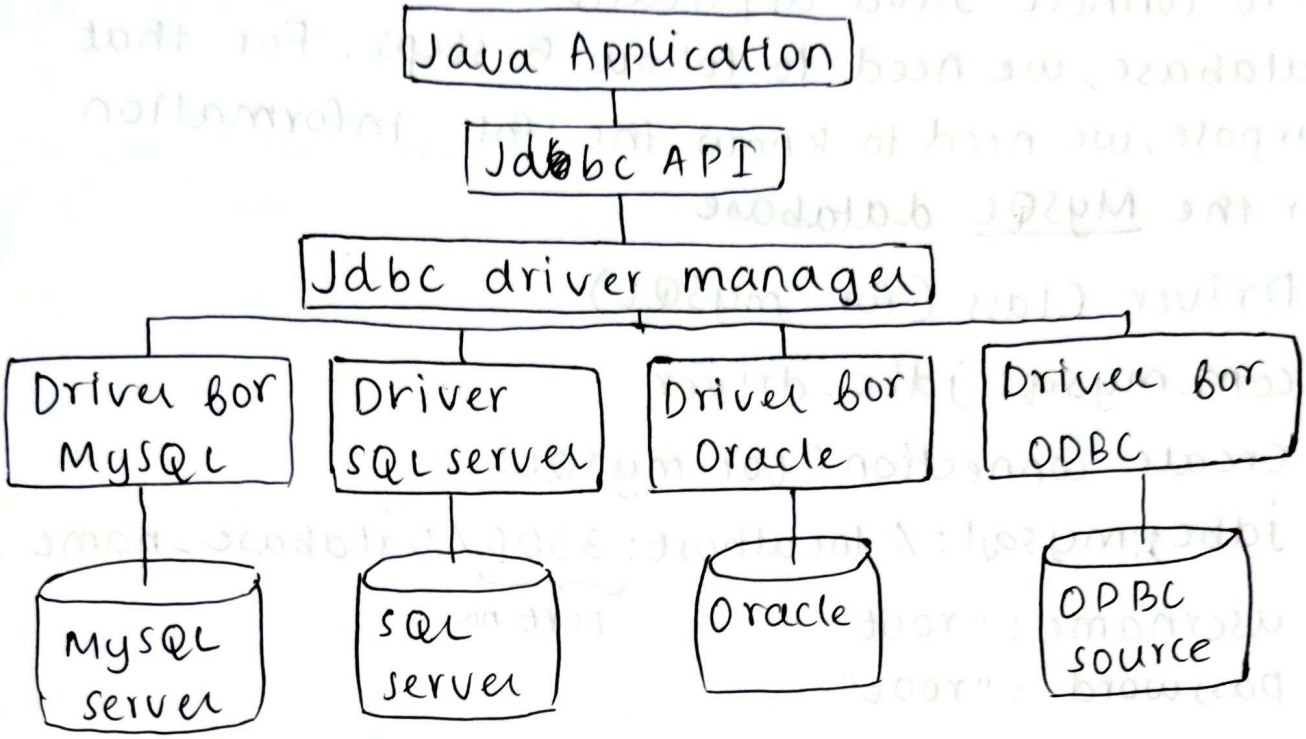
10/10 **JDBC**

- Universal language to connect a java prgm to database.
- It is an API

Architecture have 2 layers

↳ jdbc API

↳ jdbc driver API



1. Jdbc API

- It ~~prodt~~ provides the ~~app~~ connection b/w jdbc application and driver manager.

2. Jdbc driver API

This supports the connection ~~b/w~~ jdbc manager to driver.

5 Steps to connect a Java application to DB

1. Register the driver manager : forName()
2. Create connection : getConnection()
3. " statement : createStatement()
4. Execute Query : executeQuery()
5. Close connection : close()

→ getConnection() is of Driver manager class.

→ createStatement() is of connection interface.

→ executeQuery() is of statement interface

Java DB connectivity using MySQL

-To connect Java application with MySQL database, we need to follow 5 steps. For that purpose, we need to know the foll. information for the MySQL database

1. Driver Class (for mySQL)
   com.mysql.jdbc.driver
2. Create connection (for mySQL)
   jdbc:mysql://localhost:3306/Database_name
   (3306 — port no.)
   username: "root"
   password: "root"

- Before establishing a connection blw java application and DB, we need to create a DB first and then tables in it.

Eg: Create database studentDB;
use studentDB;
create table std1 (id int, name varchar(20), mark int);

→
```
import java.sql.*
class MySqlCon{
    p.s.v.m(S[], A){
        try{
            class.forName("com.mysql.jdbc.Driver");
Connection con= DriverManager.getConnection("jdbc:
mysql://localhost:3306/Database_name");
Statement stmt = con.createStatement();
Result set rs = stmt.executeQuery("Select * from
```

```
            std 1");
while (rs.next()) { sop(rs.getInt()+" "+
            rs.getString()+" " + rs.getInt()); }
con.close();
 }
catch(Exception e){ sop("   "); }
}
```