unit 1

1.1 Object oriented programming

object oriented programming (oop) is based on the concept of "objects" which may contains data, in the form of fields, attributes and code (data members in the form of procedures / methods (member functions).

Feature / concepts / prinuples of oops.

objects
classes
Data abstraction
data encapsulation
Inheritance
polymorphism
Dynamic binding
Message passing

Abstraction

Abstraction is to handle complexity by hiding unnecessary details from the user. That enables the user to implement more complex proces without understanding or even thinking about all the hidden complexity.

Real Time Example

Human Beings can talk, hear, eat. but the details are hidden from the outside world.

Objects and classes.

Class:

A class is the collection of variable declaration and method definition.

Real Time Example

Considering humanBeing a class, which has properties like hands, legs, eyes etc..

object:

object is the instance of the class. IE's the keyto enter and process the class members.

Real Time Example:

Name of the human being is called object.

Encapsulation:

It can also be said data binding. Encapsulation is all about binding the data variables and functions together in class.

Real time Example:

our legs are binded to help us walk.

our hands, helps us hold things

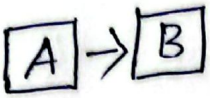
Inheritance:

Inheritance is the process by which objects of one class acquire the properties of objects of another class.

Single inheritance

In single inheritance, one class is derived from an already existing base class.

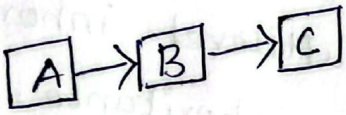
Here, A is base class and B is the derived class



Multiple inheritance (Level)

In multi level inheritance, a new class is derived from a class already derived from the base class

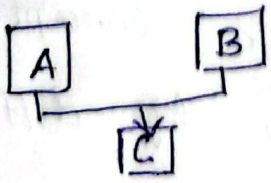
Here B is derived from class A and class C is further derived from the derived class B



Multiple inheritance

In multiple inheritance, a sengle class is derived from more than one base class.

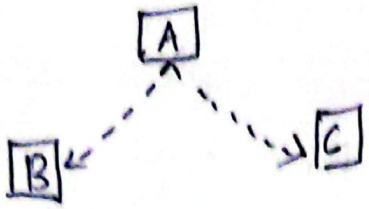
Here class C is derived from base classes A and B



Hierarchical inheritance

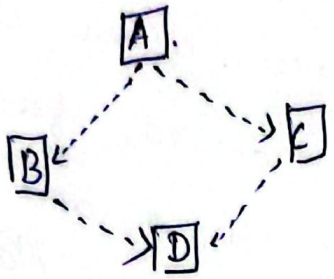
In hierarchical inheritance more than one class is derived from a sengle base class.

Here class B and C derived class A



Hybrid inheritance:

Hybrid inheritance is defined as a combination of more than one inheritance.



here classes A, B, C represents hierarchical inheritance. classes A, B, & D and classes A. C and D represent multilevel inheritance. Classes B, C, D represent multiple inheritance.

Real Time Example.

child will have the basic genetic characters, diseases like parents.

Polymorphism:

polymorphism in java is a concept by which can be perform a single action by different ways.

polymorphism is derived from 2 (two) greek words poly → many, morphs → forms. } more than one form.

Real Time Example

In a class there will be more students with a same name, we can differentiate the students with initial register no etc...

oop in java

oop concepts in java are the main ideas behind java's object oriented programming

1. these are an abstraction, encapsulation, inheritance and polymoriphism.
2. Grasping them is key to understanding how java works.

Basically,

java oop concepts let us create working methods and variables, then reuse all part of them without promising security

List of oop concepts in java

1. Abstraction

Abstraction means using simple things to represent complexity.

we all know how to turn the TV on, but we dont need to know how it works inorder to enjoy it.

Java Buzzwords

Overview of Java

1. Java programming Language was originally developed by sun Microsystem by James Gosling patrick Naughton
2. Java programming Language was initially called as OAK in the year 1992
3. OAK was renamed as java in the year 1995
4. Java was originally meant to be a platform neutral language for embedded software in devices.
5. portable and platform Independent.
6. Language could be used to produce platform neutral code.
7. Java is guaranteed to be write once, Run any where.

Java is

* Object oriented.
    1. In java everything is an object
    2. Java can be easily extended since it is based on the object model.
* platform Independent.

unlike many other programming language including C and C++; when Java is compile

Features of Java

Simple

1. Java is easy to write and more readable and eye catching
2. Java has a concise, cohensive set of features that make it easy to learn and use.
3. Most of the concepts are draw from C++ thus making Java learning compiler.

Secure

1. Java program cannot harm other system thus making it secure
2. Java provides a secure means of creating Internet applications.
3. Java provides secure way to access web applications

portable

1. Java programs can excute in any environment for which there is a Java runtime system
2. Java programs can run on any platform.
3. Java programs can be transferred over world wide web. (eg. applets)

object oriented :

Java programming is a object oriented programming language.

Like C++, Java provides most of the object oriented features

Java is pure oop Language.

Robust

Java encourages error programming being strictly typed and performing run time checks.

Multithreaded

Java provides integrated support for multithreaded programming.

Architecture neutral:

1. Java is not tied to a specific machine or operating system architecture

2. machine Independent

Interpreted:

1. Java supports cross platform code through the use of Java bytecode

2. Bytecode can be interpreted on any platform by Jvm.

Highly performance:

1. Bytecodes are highly optimized

2. Jvm can excuted them much faster

Distributed

Java was designed with the distributed environment

Java can be transmit, run thr over internet

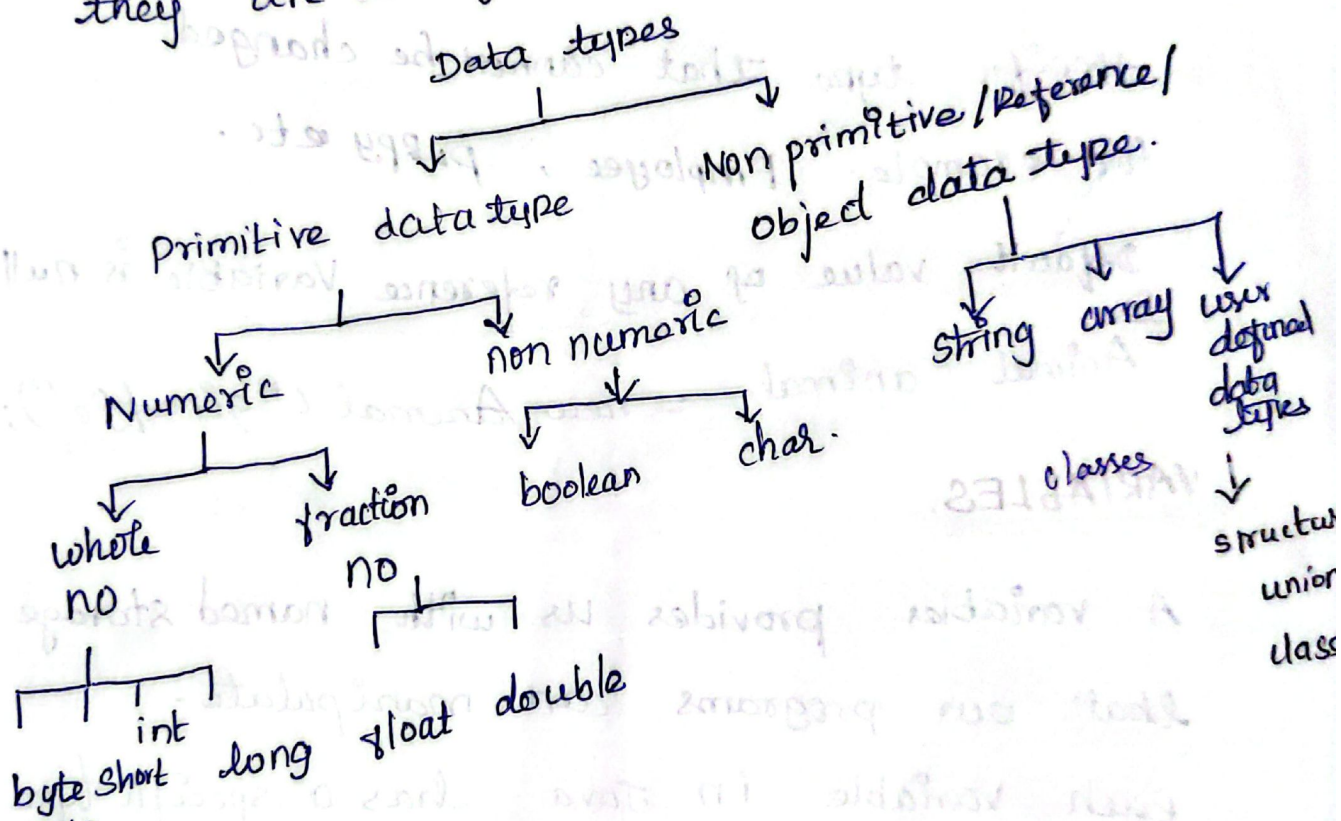
Dynamic

1. Java programs carry with them substantial amounts of runtime type information that is used to verify and resolve accesses to objects at run time.

DATA TYPES.

Data types of a variable represents the operating system to allocate memory and decided what can be stored in reserved memory.

they are two types.



Data type — integer — byte — one byte

Data type	Size	minimum Value	maximum Value	Default Value.
Byte	8bit	-128	128	0
short	16bit	-32,768	32767	0
Int	32bit	-2,147,483	2,147,483	0
long	64bit	-9,223	9,223	0

Reference / object data type

objects (reference type variables) are created using

Reference variables are created using defined constructors of the class. they are used to access objects. these variables are declared to be of a specific type that cannot be changed

for example. Employee, puppy etc.

Default value of any reference variable is null

Ex

Animal animal = new Animal ("giraffe");

VARIABLES.

A variables provides us with named storage that our programs can manipulate.

each variable in Java has a specific type

```
int a,b,c                    // Declares three int a,b,c
int a=10, b=10;              // Example of initialization
byte B = 22;                 // initializes a byte type
double pi = 3.14             // declares and assigns a value of PI
char a = 'a';                // the char variable a is initialized with value
```

memory large ← byte B = 22; array use ← static

Vari B

Instance variable

A variable which is that are declared inside the class but outside the scope of any method are called instance variable in java

instance variable are declared in a class but outside a method, constructor or any block

Instance variable can be declared in class level before or after use

Access modifier can be given for instance variable

instance variable are visible for all methods

it hathe an default values 0.

Syntax

```
Class A
{
int a;
}
```

```
class A
{
int a;
p.sum ( )  s.y.out.P
}
p.sum();
}
S.y.o.Print(n ("In a");
}}
```

Type of Variables:

1. Local Variables
2. Instance variables
3. class /static variables.

Local variables.

Local variable are declared in methods, constructors, or blocks. (which is declared inside the body of the method)

1) Access modifiers can not be used for local variables
2) Local variables are visible only within the declare method, constructor or block
3. Local variables are implemented at stack
4. there is no default value for Local variable.

Syntax:

```
Void fun (int a)
```

parameter → Local variable

Example. program

```
Public class Test
{ public void pupAge()
{
  int age=0;
  age =age+7;
  System.out.println("puppy age is: " + age);
} public static void main (String args[]) {
    Test test = new Test();
    test.pupAge();
```

Instance variables.

1) Instance variables are declared in a class, but outside a method, constructor or any block.

2. when a space is allocated for an object in the heap a slot for each instance variable value is created.

3. Instance variables are created when an object is created with the use of the keyword 'new' and destroyed when the object is destroyed

4. Instance variables can be declared in class level before or after use.

5. acess modifiers can be given for Instance variables.

6. The instance variables are visible for all methods, constructors and block in the class.

7. Instance variables have default values.

8. Instance variables can be accessed directly by calling the variable name inside the class

Object Reference. Variable Name;

Example:

```
import java.io.*;
public class Employee                    // this instance variable is
{                                          visible for any child class.
public string name;
```

```
/salary variable is visible in Employee
Private double salary;
/*The name variable is assigned in the constructor.
Public Employee (String empName)
{
name = empName;
}
// Salary variable is assigned a value.
Public void SetSalary (double empsal)
{
Salary = empsal;
}
// This method prints the employee details
Public void printEmp()
{
System.out.println ("name : " + name);
System.out.println (" Salary: " + Salary);
}
    public static void main (String args[]) {
    Employee empone = new Employee ("Ramsika");
    emp one. SetSalary (1000);
    emp one. printEmp();
    } }
```

O/P name: Ramsika
Salary : 1000.0

Arrays

1) Array is a collection of similar data type values that shares the common name
2) each element in the array can be accessed by using the index number
3. Java array is an object

Declarations :

1) array can be declared by specifying the data type followed by [] with assay name
2. Array declaration only declares the name of the array.
3. No memory space is allocated

syntax : data_type array_name[];

Eg : int a[]

Initialization / instantiation

An array can be instantiation after declaration
Instantiation is nothing but allocating space to an array.
This can be done by using new operator.
once array is instantiated size cannot be changed.

syntax : array_name = new data_type [size];

eg: Mark = new int [5];

creation

new operator with the following syntax.

dataType [] arrayRefVar = new dataType [arraySize];

and also by giving the value directly without allocate the memory,

syntax:

dataType [] arrayRefVar = {v0, v1, ... vk};

Example

double [] myList = new double [10];

Example:

```
class ArrayAddition
{
public static void main (String s[])
{
 int a[]= {1,2,3,4,5}; b[] = {6,7,8,9,10}. c[];
  c = new int [5];
  for (int i=0; i<5 ;i++)
  {
  c[i] = a[i] +b[i];
  System . out. println ("c [" + i + "] = " + c [i];
  }}}
```

output

c[0] = 7

c[1] = 9

c[2] = 11

c[3] = 13

c[4] = 15.

Basic functions of Array class

the following methods are within util package so we need to import java.util.Arrays" package for processing basic array methods

Array length :

size of an array can be obtained by accessing the attribute length.

Array-name.length

(Eg.) size = mark.length ;

Array copying.

Array can be copied into another array in two ways.

designation-array = source-array.

copyOf() method.

Array name = Arrays.copyOf (Existing-array.array-size);

eg. Num = Arrays.copyOf (Num, 10);

Array Sort() Method

Array will be sorted automatically in ascending oder

Syntax Array.sort (array object) (obj

Eg. Array.sort (arr)

Arrays.toString() Method

Array can be converted to string and the

values will be displayed with close and open square bracket [].

syntax : string object = Arrays. toString (array object);

Eg: String str = Arrays. tostring (arr);

Arrays. equals (Array) Method:

Array value will be checked with the another array value and tell whether these two arrays are equal or not

syntax

public static boolean equals (Object [] a, object[] a_2)

Eg int a[] = new int [] {1, 2, 3, 4, 5};

int b[] = new int [] {1, 2, 3, 4, 5};

boolean b = Arrays. eqaals (a, b);

Types of arrays

1. One dimensional Array

Represented by using single Index

syntax

data type[] object = new datatype [size];

Eg

int [] arr = new int [5]

Example

import. java. util. Arrays;

```
import java.util.Scanner;
Public class Arrayprocess
{
Public static void main (String[] args)
{
int arr[] = new int [5];
Scanner s = new Scanner (System.in);
System.out.println ("Enter 5 integer values");
for (int i=0; i<5; i++)
  arr [i] = s.nextInt();
 int arr1[]= new int[5];
   arr1 = arr;
  Arrays.sort (arr);
  Arrays.equals (arr, arr1);
  System.out.println (Arrays.toString(arr));
  System.out.println();
  }}
```

output

Enter the 5 integer values

5
6
2
4
1
[1, 2, 4, 5, 6]

Multi dimensional Array

1) Represented by using Multiple Index
2) It is the collection of one dimensional Array

Array Instantiation - Two dimensional

int a[][] = new int [2][2]

Array Initialization - Two dimensional

int a[][] = {10,10}, {30,40}};

Ragged Arrays:

Ragged arrays are arrays in which different rows have different length.

In Java, Multidimensional arrays are represented as array of arrays.

there is no no seperate multidimensional arrays in java as in c++

PAckages :-

1) Provides a mechanism for grouping a variety of classes and/or interfaces together

2) Grouping is based functionality

Benefits

1. The classes contained in the package of other programs can be reused

2. In packages, classes can be unique compared with classes in other packages

Operators.

Java provides a rich set of operators to manipulate variables

Arithmetic operators

1. An arithmetic operator is a mathmatical function that takes two operands and preform calculation on them.
2. They are used in common arithmetic and most computer languages contain a set of such operators that can be used with in equations to perform a number of types of sequencial calculation

operator	Description	Example
+	Adds two operands	A+B will give 30
-	subtracts second operand from the first	A-B will give -10
*	Multiplies both operands	A*B will give 200
/	divides numerator by denumerator	B/A will give 2
%	modulus operator and remainder of after an integer division	B%A will give 0

Relational operator

A relational operator is a programming language construct or operator that tests or defines some kind of relation between two entities.

If the relation between two entities it return true else it will return false

operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A==B) is not true
!=	Checks if the value of two operands are equal or not, if the values are not equal then condition becomes true	A!=B is true.
>	Checks if the value of left operand is greater than the value of right operand if yes then condition becomes true.	A>B is not true
<	Checks if the value of left operand is less than the value of right operand if yes then condition becomes true.	A<B is true.

Logical operator

Logical operator are the function of logic gates in digital circuits., logical operations include AND, OR, Not and combinations of those operations.

operator	description	Example
&&	called logical AND operator. If both the operands are nonzero then condition becomes true	(A && B) is false
||	called logical OR operator. If any of the two operands is non zero, then condition becomes true	(A || B) is true
!	called logical Not operator use to reverses the logical State of it operand. if a condition is true. then Logical NOT operator will make false	!(A && B) is true.

Bitwise operator :-

A bitwise operator is an operator used to perform bitwise operations on bit patterns or binary numerals that involves the manipulation of Individual bits.

Operator	Description	Example
&	Binary AND Operator copies a bit to the result if it exists in both operands	A&B will give 12 which is 0000 1100
|	Binary OR operator copies a bit if it exists in their operand.	(A | B) will give 61 which is 0011 1101
^	Binary XOR operator copies the ~~bit~~ if it is set in the one operand but not both	(A^B) will give 49 which is 0011 00
~	Binary ones complement operator is unary and has the effect of flipping bits	~A will give -61 which is 1100 0011 in 2's Complement ~~from~~ due to a signal binary no
<<	Binary left shift operator	A<<2 will give 240 which is 1111 0000
>>	Binary Right shift Operator	A>>2 will give 15 which is 0000 1111

Simplified operations

$V >> n \Rightarrow V/2^n$

Assignment operator.

An assignment operator is the operator used to assign a new value to a variable

operator	Description	Example
=	Simple assignment operator	C=A+B will assign value of A+B into C
+=	Add AND assignment operator,	C+=A is equivalent to C=C+A
-=	Subtract AND assignment operator	C=A is equivalent C=C-A
*=	Multiply AND assignment operator.	C*=A is equivalent to C=C*A
/=	Divide AND assignment operator	C/=A is equivalent to C=C/A
%=	modulus AND assignment operator	C%=A is equivalent to C=C%A
<<=	left shift AND assignment operator	C<<=2 is same as C=C<<2
>>=	Right shift	

Increment & Decrement operator

It works on the single operand which increase decrease the value by 1

Operator	Description	Example
++	Increment, increases integer value by one	A=10 A++ will give 11
--	Decrement operator decrease integer value by one	B=20 B-- will give 19

Example

```
import java.io.*;
class operator operation
{
public static void main (String args[])
{ int a=10;
  int b=20;
// Arithmetic operator
System.out.println("A+B = " +(a+b));
System .Out.println ("A-B =" +(a-b));
// Relational operator
System .out .println (" A ==B =" + (a==b));
System .out. println ("A<B = "+(a<b));
```

// Bitwise operators

System.out.println("A<<3 = "+(a<<3));

// Print as [A<<3 = (10*2^3) -> 10*8) = 80]

System.out.println("B>>2 = "+(b>>2));

//Assignment operators

int c = 30;

c += a;

System.out.println("c = "+c);

// prints as 40 {c = c + a -> c = 30 + 10 = 40}

// Misc operators

System.out.println("Greater value = "+(c>b? c:b));

// returns 40 as condition c>b is true

Output

A + B = 30

A - B = -10

A == B = false

A < B = true

A && B = true

A || B = true

A || B = true

A > B && A < B = false

A << 3 = 80

B >> 2 = 5

C = 40

Greater value = 40.

Control Flow

Conditional statements

Decesion making structures require that the programmer specify one or more condition to be evaluated or tested by the program along with a statement or statements to be executed. i

if the condition is determined to be true and optionally,

other statements to be executed if the condition is determined to be false.

Statements	Description
if statement	A if statement consists of a Boolean expression followed by one or more statements.
if ... else statement.	A if statement can be followed by optional else statement, which executes when the boolean expression is false
Switch statement	A switch statement allow a variable to be tested for equality against a list of values.
nested if statements	you can use one if or else if statement inside another if or

Example

```
class IfConditionEg
{
Public static void main (String args[])
{
int a=10; b=20;
if (a>b)
{
System.out.println ("Greatest Value =" +a);
} else
System.out.println ("Greatest Value =" + b);
}}
```

o/p

Greatest Value =20

Looping Statements

A loop is a way of repeating a statement a number of times until some way of ending the loop occurs.

It might be run for a present number of times typically in for loop repeated as long as an expression is true (a while loop) or repeated until an expression becomes false in do while loop.

Loop Type	Description
while loop	Repeats a statement or group of statements while a given condition is true. if the test the coditiun before executing the loop body.
for Loop	Execute the sequence of statements ~~while a given condition is true. if tests t~~ multiple times and abbreviates the code that manages the loop variable
do ... while loop	Like a while statement, except that it tests the conditiun at the end of Loop body
nested Loops	you can use one or more loop inside any another while, for or do.. while loop.

control statement	Description
break statement	terminates the loop or switch statement and transfers excation to the statement immediately following the loop.
continue statement	causes the loop to skip the remainder of its body and immediately retest its

condition prior to
reiterating

Example program

```
Public class LoopExample
{
Public static void main (String [] args)
{
//value to print from 1-10 without printing values
System.out.println("Number from 1-10 without value
    using the loop");
for (int i=1; i<=10; i++)
{ if (i==5)
    continue.
else
System.out.print(i + "\t");
}
System.out.println("\n Number from 1-10 without
values while loop");
 int i=0;
 while (i<=10)
 {i++;
 if(i==5)
    continue;
 if(i>10)
    break;
 else
   System.out.print(i + "\t");
        }}}
```

Programming Structure in Java

JDK	JRE
Java source file	Class Loader ← inbuilt Java Library Byte code verification Java interpreter Java Just in compiler
Java Bytecode	Run time system Operating system and hardware.

Structure of Java program has been classified into five stages.

1. user can editor to type Java program (welcome.java)

2. compile.

(a) use a compiler to translate Java program into an intermediate language called bytecodes

understood by Java interpreter

(Javac Welcome.java)

b. use a compiler to create .class file, containing byte codes (welcome.class

3. Loading use a class loader to read bytecodes from .class file into memory

4 verify use a bytecode verifier to make sure bytecodes are valid and do not violate Security restrictions.

5. Execute

a) Java virtual machine use a combination of interpretation and just time compilation to translate bytecode into machine language.

b) Applications are run on user's machine.

(ie) excuted by interpreter with java command (Java welcome).

Fundamental programming structure in java

Documentation Section — Suggested

Package statement — optional

Import Statement

Interface Statement — option

class definition

main method — essential

Definition

1. Documentation section

It includes the comments to tell the program's purpose.

it improves the readability of the program

these are the optional but we suggest you use them because they are useful to understand the operation of the program.

/** and end with */

Java doc tool reads these comments and uses them to prepare your program documentation HTML format

Package Statement

It includes Statement that provides a package declaration.

Defining classes in java

A class is a blueprint or prototype that defines the variable and methods common to all objects of a certain kind.

Declaration

A class is declared using class keyword followed by the name of the class

syntax

```
class classname
 {
    //variables declaration
    // methods declaration
    // Methods definition
     }
```

Example:

```
class Goodbye world
{
public static void main (String args[])
}}
```

Class variables

In object oriented programming with classes, a class variable is a variable defined in a class of which a single copy exists regardless of how many instances of the class exist

The class contain the following items with it.

member variable
member function
constructor

Public class
class should be declared as public to have its access from anywhere.

object:
object refers to a particular instance of a class

creation:
1) Object is a block of memory that contains space to store all the instance variable.
2) one class can have more than one objects
3. Every object has its own storage block which contains the space for fields.

Character of objects:
1) identity -- the name to identify the object uniquely among multiple objects
2) Behaviour - the processes that can be done on fields by the objects. (methods)
3) State - the change in the data after the processing
Initialization.

creating an objects:
java object created with a statement

Classname Objname = new Classname();

This statement creates a new class object
The single statement declares in stantiates and initializes the object

if there is a constructor it will be called otherwise default constructor without any parameters will be called.

Constructors:

Special type of method used to initialize object.

when the object is initialized this method is invoked automatically before the new operator completes

This method must be in the same class and with the same class name

No returntype

can be declared with/ without arguments

can be overloaded.

Invoked by using new operator.

Declaration of constructor:

[access specifiers] class_name [arguments])

{

statements;

}

Types of constructors

constructors are categorized into two ways.

Default constructor:

the constructor without arguments are called default constructor

Parameterized constructor:

the constructor with arguments are called parameterized constructor.

Example program

```
class Constructorclass
{
public Constructor (class C)
{
System.out.println (" Default constructor is called");
}
public Constructor Class (int a)
{
System.out.println(" constructor with Integer
                    is called ");
}
public ConstructorClass (float f)
{
System.out.println(" Constructor with Float parameter
                    is called ");
} public Constructor Class (String f)
{ System.out.println (" constructor with string
                      Parameter is called");
}}
```

```
Public class ConstructorExample {
public static void main (String [] args)
{
  // Default Constructor is called below
Constructor class cd = new Constructor class ();
// Constructor with Integer parameter is called below
  Constructor class ci = new Constructor class (6);
// constructor with Float parameter is called below
  Constructor class cf = new Constructor class (float) 10.25);
// Constructor with String parameter is called below
Constructor Class cs = new Constructor Class ("Hai welcome");
  }}
```

output

Default Constructor is called
Constructor with Integer parameter is called
Constructor with Float parameter is called
Constructor with String parameter is called

Finalize Method :

1. finalize method is similar to a destructor in C++
2. manual memory reclamation is not needed because java does automatic garbage collection
3. finalize method will be called before the garbage collector removes the object
4. only one finalize can be defined a class.

Methods (a way of doing something)

Declaration

1. functions defined inside the class definition are called as methods
2. Method are the member functions used to manipulate the fields.
3. Methods define the behavior of an object so every class should have atleast one method to respond
4. A method can receive argument and return value
5. A method that does not return value is declared as void.

Character of Method

Defines the behavior of the objects

1. can access all the fields in its own class
2. must be defined inside the appropriate class
3. It can be accessed by normal method of it own class and the derived class
4. can manupulate the value of both normal and static fields.

It can be preceded with an special specifiers

Method definition

[access specifiers] return_type method name(formal_arguments_List)

{

Statements;

[return value] // if needed.

Types of methods:

methods are divided into two categories

✓ Mutators → Methods that change field value of an objects.

```
public void chang (int n1, int n2)
{
int Num1 =n1;
int Num2 =n2;
}
```

Accessors

Methods that only access fields of an object without modifying them

```
public int get Value()
{
  return Num1;
  }
```

Method parameters.

A method can be defined to accept zero or more parameters.

Each parameter in the parameter list is defined by its type and name.

The parameters in the method defenition are called formal parameters.

The name of the method together with the list of formal parameters is called the signature of the method

Example:

Public Void setTime (int h, int m, int s)

Method calling

for a method, if should be called, they are: ways in which a method is called (ie) method returns a value or returning nothing

This process of method calling is simple.

1. return type is executed
2. reaches the method ending close curly brace
3. methods returning void considered as call Statement.

System . out . println ("This is method example program!");

This method returning value can be understood by the following example

Int result = Sum(6,9)

Parameter passing methodologies

Parameter can be passed

1. by value
2. by reference

Pass by value:

The method get a copy of all parameter value and the method cannot modify the content of any parameter variable that are passes do it

Example:

```
Public class ExampleMinNumber
{
Public static void main (String [] args) {
  Int a= 11;
  Int b=6;
 Int c = minFunction (a,b)
System . out . println ("minimum Value = "+c);
}
```

```
/** returns the minimum of two numbers */
public static int minFunction (int n1, int n2) {
int min;
if (n1 > n2)
min = n2;
else
   min = n1;
   return min;
   }}
```

Pass by Reference

The method gets all parameter values, and the method modify the contents of any parameter variables that are passed to it

Example

```
Public class SwappingExample
{
  public static void main (String [] args) {
int a = 30;
int b = 45;
System.out.println(" Before Swapping a = " + a +
// invoke the swap method and b = " + b);
SwapFunction (a, b);
System.out.println ("\n** Now. Before and Aft
```

```
    Swapping values will be same here**.");
System.out.println("After swapping a=" +a+ "and b"
                                        +b);
}
Public static void SwapFunction (int a, int b)
{
System.out.println("Before swapping (Inside),
                a=" +a "b=" +b);
// swap n1 with n2
Int c=a;
  a=b;
  b=c;
System.out.println("After Swapping (Inside), a
                =" +a+ "b=" +b);
}}
```

output

Before Swapping a=30 and b=45

Before swapping (Inside), a=30 b=45

After swapping (Inside), a=45 b=30

** Now, Before and After swapping values will be same here **;

After swapping, a=30 and b is 45.

Access Specifiers

Access modifiers are keywords in object oriented programming languages that set the accessibility of classes, methods and members.

Access modifiers are a specific part of Programming language syntax used to facilitate the encapsulation of components

A class cannot be declared as private.

the following are the access specifiers of java and access rights.

Access Modifiers	same class	same package	subclass	other package
public	Y	Y	Y	Y
protected	Y	Y	Y	N
no access specifier or modifier	Y	Y	N	N
private	Y	Y	N	N

Static Members.

Static members (variables) or methods that belong a static or non static class itself

rather than to objects of the class. static members always remain the same regardless of where and how they are used

Static members are of two types

- static variable
- static method

static variable.

1. Shared among all objects of the class
2. only one copy exists for the entire class to use
3. Stored within the class code. separately from Instance variable that describe an Individual object
4. public static final variables are global constants

Example program.

```
class countobject
{
Static int count ;
Countobject()
{
count++;
}}
Public class StaticVariableExample
{
public Static void main (String[] args)
  }
```

```
CountObject c1 = new CountObject();
System.out.println("count after c1 object is created =
                                    "+c1.count);
CountObject c2 = new CountObject();
System.out.println("Count after c2 object is created =
                                    "+c2.count);
Countobject c3 = newCountObject();
System.out.println("Count after c3 object is created =
                                    "+c3.count);
CountObject c4 = newCount object();
System.out.println("count after c4 object is created=
                                    "+c4.count);
CountObject c5= new CountObject();
System.out.println("count after c5 object is created =
                                    "+c5.count);
}
}
```

Output

Count after c1 object is created = 1
Count after c2 Object is created = 2
Count after c3 object is created = 3
Count after c4 object is Created = 4
Count after c5 object is created = 5

Note :- If the count variable is declared as

instance variable the output will be

Count after c1 object is created = 1
Count after c2 object is created = 1
Count after c3 object is created = 1
Count after c4 object is created = 1
Count after c5 object is created = 1

As individual memory will be allocated for each and every object created for all the class.

Static Methods.

1) Static methods can only access directly the static members and manipulate a class's static variables.
2) Static methods cannot access non static members. (instance variable or instance methods of the class
3. Static method can't access this and super references.

13 JAVA DOC COMMENTS

The java SDK contains a very useful tool called javadoc that generates HTML documentation from your sourcefiles

, if you add comments that start with the special delimiter /** to your source code, you too can produce professional looking documentation easily.

If you put your documentation into a seperate file, then you probably know that the code and comments tend to diverge over time

But since the documentation comments are in the same file as the source code, it is an easy matter to update both and run javadoc again

1.1. How to insert comments

The javadoc utility extracts information for the following items

1. packages
2. public classes and interfaces
3. public and protected methods
4. public and protected fields

1. A comment starts with a /** with a */. Each /** documentation comments contains free from text followed by tags.

2. A tag starts with an @ such as @author or @ param

3. The first sentence of the free form text should be summary statement.

4. The javadoc utility automatically generates summary pages that extract these sentences

5. In the free form text, you can use HTML modifiers such as ...

6. However, stay away from heading <h1> or rules <hr> they can interfere with the formatting of the document.

Class comments

The class comment must be placed after any import statements, directly before the class definition

```
Public class card
{
    ....
}
```

Method comments

Each method comment must immediately precede the method that it describes; In addition to the general purpose tags

@ param variable description

1) This tag adds an entry to the parameters || Selection of the current method

2. The tag description can span multiple lines and can use HTML tags

@ return description

3. This tag add a return selection to the current method.

Field Comments

Need to document public fields - generally that means static constants.

ex

```
/**
 The "Hearts" card suit
*/
public static final int HEARTS = 1;
```

Gathering comments

@ author name

This tag makes an "author" entry. You can have multiple @ author tags, one for each another

@ version text.

How to Extract comments

doc Directory is the name of the directo where you want the HTML files to go.

1. change to the directory that contains the so files you want the ~~HTML~~ documents, if you have nested packages to document such as horstmann. corejava, you must be in the directory that contains the subdirectory.

Run the commend,

javadoc -d docDirectory nameOfPackage

single package.

or

runjavadoc -d docDirectory nameOfPack

nameOfpackage2 ... to doc

multipe packages.

If your files are in the default package then run javadoc -d docDirectory *.java instead

If you omit the -d docDirectory option, then the HTML files are extracted to the current directory.

Unit II

Overloading Methods.

overloading methods in java means, a class contains multiple methods with same, same but different in parameter count and parameter data types

It is also called as static polymorphism.

Advantages.

1. It increases the readability of the program.
2. It provides the flexibility to use similar method with different types. parameters.

Three ways to overload a method

1. Number of parameters

for. example.

this is a valid case of overloading

add (int, int)

add (int, int, int)

2. Data type of parameters.

ex

add (int, int)

add (int, float).

3. sequence of Data type of parameters

ex add (int, float)

add (float, int)

Rules of method overloading:

1. first and important rule to overload a method in java is to change method signature.
2. Return type of method is never part of method signature. only changing the return type method does not amount of method overloading.

example.

To find the the minimum of given numbers.

```
Public class overloading Calculation1
{
  Public static void main (String [] args)
  {
    int a=11;        double x=4.3;
    int b=6;         double y=9.4;
    int c=3;

    int result1 = minFunction (a,b,c);
    double result1 = minFunction (a,b,c);
    double result2 = minFunction (x,y);
    double result3 = minFunction (a,x);
    S.O.println (" Minimum ("+a+ ", "+b", "+c" ) = "+result1);
```

```
System.out.println("Minimum("+x+","+y+")="+result2);
System.out.println("Minimum("+d+","+x+")="+result3);
}
public static int minFunction (int n1, int n2,
{                                  int n3)
int min;
int temp = n1<n2 ? n1 : n2;
 min = n3< temp? n3: temp ;
 return min ;
  }
 Public static double minFunction (double n1, double
 {                                  n2)
  double min;
 if (n1> n2)
    min =n2 ;
   else
    min = n1;
    return min;
   }
Public static double minFunction (int n1, double n2)
{  double min;
     if (n1>n2)
        min =n2;
       else
      min = n1;
   }  return min;
```

o/p

minimum (11, 6, 3) = 3

minimum (7, 3, 9, 4) = 7.3

minimum (11, 7, 3) = 7.3

Objects as parameters

Java is strictly pass by value, But the scenario may change when the parameter passed is of primitive type or reference type

1. if we pass a primitive type of a method, then is called pass by value.
2. if we pass an object to a method, then it is called pass by reference.

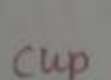
Pass by value

only values are passes to the function parameter

Caller and callee method will have two indepedent variable with same value

Requires more memory

Pass by value

cup = 

fill cup()

pass by reference.

Reference to the object is passed,

caller and callee method use the same reference

less memory

pass by reference

cup = 

fill cup()

Returning objects

In Java, a method can return any type of dat Return type may any primitive data type or class type (ie object)

ex

```
class Add
 {
 int num1, num2, sum;
Static Add CalculateSum (Add a1, Add a2)
{
Add a3 = new Add()
a3.num1 = a1.num1 + a1.num2;
a3.num2 = a2.num1 + a2.num2;
a3.sum = a3.num1 + a3.num2;
      return a3;
 }
public static void main (String args[])
{
Add obj1 = new Add()
obj1.num1=10;
obj1.num2=15;
Add obj2 = new Add()
obj2.num1=100;
obj2.num2=150;
Add obj = calculateSum (obj1, obj2)
System.out.println("object1 -> Sum = "+obj1.
                                    sum);
System.out.println("object2 ->sum= "+obj2.sum);
```

```
System.out.println(" object 3 -> sum = " + obj3.sum);
    }}
```

output

object 1 -> sum=0
object 2 -> sum=0
object 3 -> sum=275

Static Nested and inner class

Definition

inner class is a class that defined the inside the another class.

Benefits:

1. Name control
2. Access control
3. Code blw more readable and maintainable

Syntax

```
[modifier] class outerclassName
{
   ... code ...
   [modifier] class innerclass Name
   {
      ... code ...
   }
}
```

Instantiating an other class

Two methods

Java member inner class.

A non static class that is created inside a class but outside the method is called member inner class

syntax

```
Class outer
{
// code
 class Inner
  {
    // code
  }}
```

Example:

```
class TestMember outer1
{
 private int data=30;
  class Inner
  { void msg()
    {
    System.out.println("data is" + data);
     }}
   public static void main (String args[]).
{
  TestMember Outer1 obj = new TestMember outer1();
  Tst Membo Outer1.inner in= obj.newInner();
```

```
in msg();
}}
```

o/p

data is 30.

2. Java Anonymous inner class.

A class that have no name is known as anonymous inner in Java

It can be created in two ways

1. class (may be abstract or concrete)
2. interface

Ex

```
abstract class person
{ abstract void eat();
{ class TestAnonymousInner
  public { static void main (String args[])
  $
  Person p = new person()
      {
  void eat()
  { System.out.println("nice fruits");
    }};
  P.eat(); }}
```

o/p nice fruits

```
{
private int data = 30;
void display()
 {
 int value=50;
 class Local
 {
 void msg()
  {
 System.out.println(data);
 System.out.println(value);
  }}
  Local l=new Local()
  l.msg();
  }
  public static void main (String args[])
 {
 Local Inner1 obj =new LocalInner1();
 obj.display();
 }}
```

output

30
50

Java anonymous inner class example using interface

```
interface Eatable
{
Void eat();
}
class Test ~~INNER~~ Annonymous Inner1
{
  public static Void main (String args[])
{
Eatable e = new Eatable()
{
~~public~~
public void eat()
System.out.println("nice fruits");
};
e.eat();
  }}
```

o/p nice fruits

Java local inner class

A class (ie) created inside a method is called local inner class in java.

ex

```
public class LocalInner1
```

Rules of Local inner class

1) Local inner class can be invoked from outside the method.
2) Local variable can't be private, public, protected
3. Local inner class cannot access non final Local variable till JDK 1.7 since JDK 1.8 it.

Properties.

1. completely hidden from outside world
2. cannot access the local variable of the method but the local variable has to be declared final to access.

Java static class

A static class (ie) created inside a class is called Static nested class in java. it cannot access non static data members and methods.

1. It can access static data members of outer class including private
2. Static nested class cannot access non static. data member or method

Java static nested class example with nested instance method

class Test outer 1.

```
{
static int data=30;
static class Inner
    {
void msg()
{
System.out.println("data is "+data);
}}
public static void main (String args[]).
{ TestOuter1.Inner obj = new TestOuter1.Inner();
obj.msg();
}}
```

output:

data is 30.

Java static nested class Example with static method

```
Class TestOuter2{
static int data=30
static class Inner
  {static void msg()
  }}
public static void main (String args[])
{ Test outer2Inner.msg();
}}
```

Output

data is 30

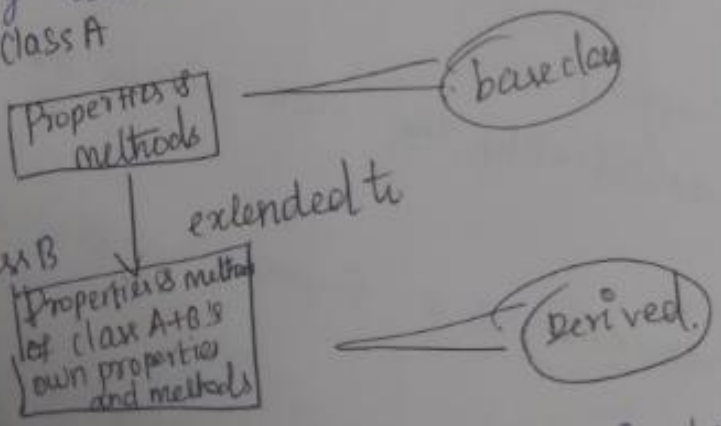
Inheritance

Inheritance is a process of deriving a new class from existing class. also called extending a class.

When an existing class is extended, the new (inherited) class has all the properties and methods of the existing class and also possesses its own characteristics.

Parent class

the class whose property is being inherited by another class "base class" or "parent or super" class



The class that inherits a particular property or set of properties from the base class is called derived class or child class or sub class.

Subclass of a class can define own unique behaviours and yet share the some of the same functionality of the parent class.

Advantages.

1. Reusability of code
2. Effort and time saving
3. Increased Reliability

"Extends" Keyword.

inheriting a class means creating a new class as an extension of another class

syntax

[access specifier] class subclass-name extends superclass.name

{
//body of class
}.

Characteristics of (class inheritance)

1. A class cannot be inherited from more than one base class. Java does not support
2. Sub class can access only the non private members of the super class,
3. private data members of a super class are local only to the class
4. protected features in java are visible to all subclasses as well as all other classes in the same package

Example

Class vehicle

{
String brand ;
String color ;

}

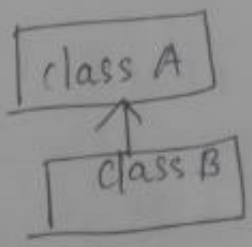
Class car extends Vehicle

{

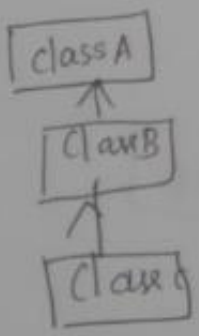
}

Types of inheritance.

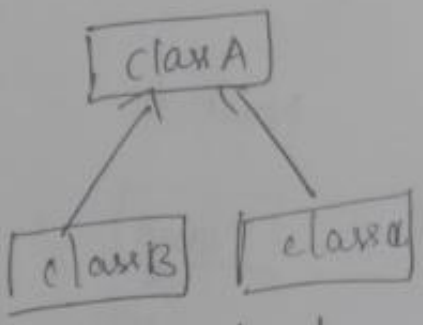
1. Single
2. multilevel
3. multiple. → does not directly support in Jave
4. hierarchical
5. Hybrid.



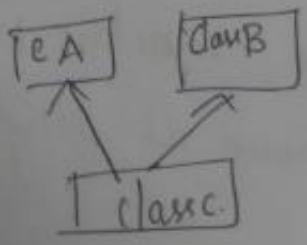
(a) Single

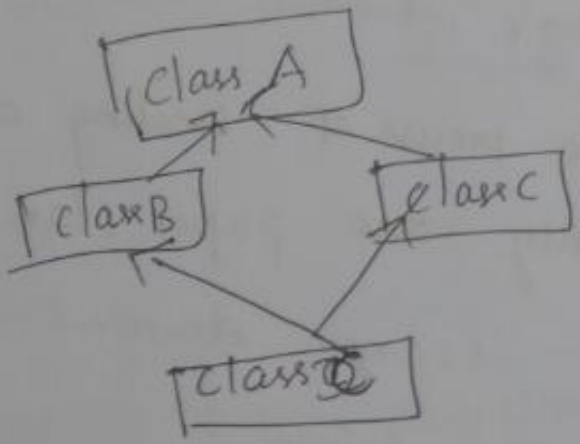


(b) multilevel

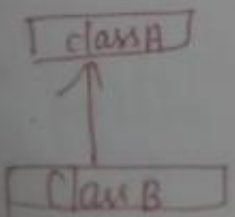


Hrerarchical





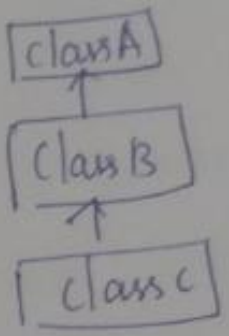
Single inheritance



syntax

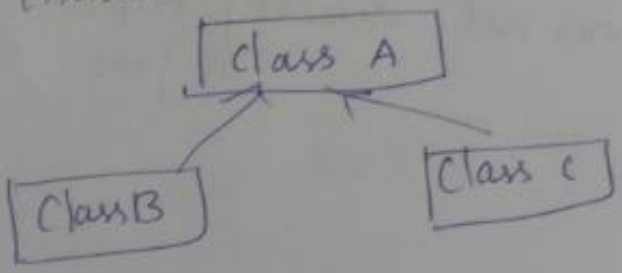
```
public class A
{
    public class B extends A
    {
    }
```

Multilevel



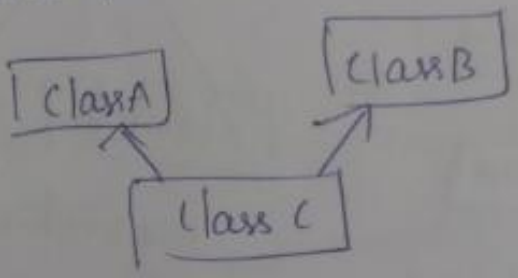

```
public class A { ... }
Public class B extends A { ... }
public class c extends B { ... }
```

Hierarchical inheritance.




```
Public class A { ... }
Public class B extends A { }
Public class c extends A { }
```

Multiple.




```
public class A { ... }
public class B { ... }
Public class c extends AB
{ ... }
}
```

Single inheritance:

The process of creating only one subclass from only one super class is known as single inheritance

Example Animal → Dog

```
Class Animal
{ void eat()
{
System.out.println("eating!..");
}}
```

```
class Dog extends Animal
{ void bark()
  {
   S.o.println ("barking...");
  }}
class Testinheritance
{
public Static void main (String args[])
{
  Dog d = new Dog();
    d. bark();
     d. eat():
      }}
```

o/p

```
$ java Test inheritance
  barking...
  eating...
```

Multilevel.

1) the process of creating a new sub class from an already inherited sub class is known as multilevel inheritance.

2) multiple classes are involved in inheritance. but one class extends only one.

```
class Dog extends Animal
{ void bark()
  {
   S.o.println ("barking...");
  }}
class Testinheritance
{
public Static void main (String args[])
{
  Dog d = new Dog();
    d. bark();
     d. eat();
      }}
```

O/P

```
$ java Test inheritance
  barking...
  eating...
```

Multilevel.

1) the process of creating a new sub class from an already inherited sub class is known as multilevel inheritance.

2) multiple classes are involved in inheritance. but one class extends only one.

eg Animal → Dog → BabyDog

```
class Animal
  {
void eat()
{
S.O.P ("eating"..);
}}
Class Dog extends Animal
{
Void bark()
{
s.o.p ("barking");
}}
class BabyDog  extends Dog
{
void weep()
{
System.out.println ("weeping..");
}}
Class Test inheritance 2
{
public static void main (String args[]){
}
  BabyDog d = new BabyDog();
      d.weep();
      d.bark();
      d.eat();  }}
```

output

weeping ...
barking ...
eating ...

Hierarchical Inheritance.

A process of creating more than one sub classes from one super class is called Hierarchical inheritance


```
Class Animal
{
  void eat()
{
System.out.println("eating...")
}
Class Dog extends Animal
{
 void park()
 {
 System.out.println("barking");
   }
 Class cat extends Animal
  void meow()
      {
 System.out.println("meowing");
                }
```

```
}
class TestInheritance3
{
public static void main (String args[])
{
Cat c = new cat()
  {. meow ();
     c. eat ()}
     }}
```

o/p
meowing ..
eating ...

Super keyword.

Super is a special keyword that directs the compiler to invoke the superclass members, it is used to parent class of the class in which the keyword is used.

three purposes

1. To invoke superclass constructor.
2. To invoke superclass member variables
3. To invoke superclass methods

Invoking superclass constructor

super as a standalone statements (ie super()) represents a call to a constructor of the superclass.

syntax

Super()

or

Super (parameter_list);

✓ Here, parameter list specifies any parameters needed by the constructor in the superclass.

✓ super() must always be the first statement executed inside the subclass constructor

Involving a superclass members

Syntax for Accessing the instance member variable of the superclass

Super.member variable;

Accessing the method of the superclass

Super.methodName();

if a parent class contains a finalize() method

Super.finalize();

Example

```
class A
  {int i;
    A (String str)
  { System.out.println ("welcome to" +str);
    } void show()
```

```
System.out.println("Thankyou");
}}
Class B extends A
  {
  int j;
 B(int a, int b);
{
Super("Java programming");
Super.i=a;
   i=b
   }.
 method overriding
Void show()
 {
 s.o.println("i in superclass: "+super.i);
 s.o.println("i in subclass: "+i);
  Super.show();
  }}
  public class useSuper{
  {
  public static Void main (String [] args) {
     B objB = new B (1,2);
      objB.show();
          }}
```

o/p Welcome to java programming
i in superclass: 1
i in subclass: 2
Thankyou

Method overriding.

The process of a subclass redefining a method contained in the super class (with same method signature) is called method overriding.

Rules

They must have the same argument list
they must have the same return type
they must not have more restrictive access modifier
they must have a less restrictive access modifier.
must not throw new or broader checked exceptions.
final methods cannot be overridden
constructors cannot be overridden

example program:

```
Class Base
{
  public void display() {
  s o p (" display method in class Base");
    }}
Class Child extends Base
{
```

```
public void display() {
    Super.display();
    S.o.p ("Display method in class child");
    }}
Publi class method overriding
    {
Public static void main (String [] args)
        Child c = new child();
            c.display(); }}
```

Output

Display method in class Base
Display method in class child

Advantages:-

It is used to provide specific implementation of a method that is already provided by its Super class.
It is used for runtime polymorphism

Dynamic Method dispatch.

It is a mechanism by which a call to an overridden method is resolved at runtime

Example

```
Class Base {
Public Void display()
{
S.O.P.("Display method in Class Base");
}}
Class child extends Base
 {
  public Void display()
  {
  S.O.P ("Display method in class child");
  }}
public class Method overriding
{
Public static Voidmain (String [] args) {
  Base b = new base();
    b.display();
  b = new child();
    b.display()
  }
```

o/p

Display method in Class Base

Display method in class child

~~Abstract~~

```
class Base
{
public void display () {
S. o.p ["Display method in classBase ");
}}
class child extends Base {
Public void display()
{
S.o.p ["display method in class child");
}}
public class method overriding
{
    public static voidmain (String[] args) {
        Base b = new Base ();
    b = new child();
        b.display(); }}
```

Output

Display method in class Base

Display method in class child

Abstract class.

Abstraction is a process of hiding the implementation details and showing only the essential features to the user.

Ways to achieve Abstraction

they are two ways.

1. Abstract class (0 to 100%)
2. Interface (100%)

Abstract classes:

Abstract classes cannot be instantiated but they can be subclassed.

```
abstract class <class_name>
{
 member variables,
 Concrete methods{}
 Abstrut methods();
}
```

properties

1. abstract keyword is used to make a class abstract
2. abstract class can't be instantiated
3. Abstract classes can have both concrete methods and abstract methods
4. A constructor of an abstract class must implement all the abstract methods unless the subclass is also an abstract classes

```
abstract class GraphicObject
{
int x, y;
...
void moveTo (int newX, int newY)
{
...
}
abstract void draw()
abstract void resize();
}
```

Abstract methods

A method that is declared as abstract and does not have implementation is known as abstract method that are implemented in the sub classes.

```
abstract class classname
{
abstract return type < method name > (parameter list);
// no implementation required
...
}
```

Properties.

1. abstract keyword is also used to declared a method as abstract
2. An abstract ~~class~~ method consists of a method segnature but no method body.

Example

```
abstract class A     // Abstract class
{
abstract void callme();
// concrete methods are still allowed in abstract
                                    classes
void callmetoo()
 {
System.out.println("This is a concrete method.");
  }}
class B extends A
  {
  void callme()
{
System.out.println("B is implementation of
                                    callme");
}}
class AbstractDemo
{
public static void main(String args[])
{
  // Aa = new A();
    Bb = new B();
    b.callme();
    b.callmetoo();
    }
```

O/P

```
B's implementation of
                    callme
This is a concrete
                    method
```

Final With Inheritance

Final is a keyword (or) reserved word in java used for restricting some functionality. It can be applied to member variable, methods, class and local variable in java.

Final is used in

- data
- methods
- class

for declaring variable → to create named constant

for declaring methods → to prevent method overriding

for declaring the class → to prevent class from inheritance.

Final variable.

final variable is a variable that can be assigned only once.

```
final data-type variable_name = value;
```

the value of the final variable will not be changed the execution of the program

Final methods

final keyword in java can also be applied to methods.

A java method with final keyword is called final method and it cannot be overridden in sub class

```
final return-type function_name (parameter list
{
  //method body
}
```

Final classes

java class with final modifier is called final classes in java and they cannot be sub classed or inherited.

Syntax:

```
final class class_name
{
//body of class
}
```

final vari

```
class Bike
{
final int speedlimd=90;
        // final var
void run()
{
Speed limd =400;
}
P.S. void main(String
        args[])
{
Bike obj=new Bike();
obj.run();
}}
```

Example: final method

```
class Bike
{
final void run()
{
S.y.o.p("running");
}}
class Honda extends Bike
{
void run()
{
S.o.p("running safely
        with 100 kmph");
}
P.S. Void main(String[]args
```

final classes

```
final class Bike
{
}
class Hondai extends Bike
{
void run()
{
System.out.p("running
        Safely with 100
        kmph")
}
P.S. Void (String args
[])
{
Hondai honda = new
        Hondai();
```

o/p
compile time
Error

```
Honda honda = new Honda();
Honda.run();
}}
```

o/p

D. Javac Honda.java
java 9. error

```
Void run()
    ^
Overridden method
    is final
```

1 error.

```
honda.run();
}}
```

o/p

D:\>javac Honda.
java

Honda jar:4 error.

Packages & interfaces

Packages.

1) It provides a mechanism for grouping a variety of classes and / or interfaces together

2) Grouping is based on funitonality.

definition.

package can be defined as collection of classes interfaces, enumerations, and annotations, providing access protection and name space management.

Syntax

```
package package name [sub pack name];
public class classname
{. . .
}.
```

Two types of packages

1. Java API packages
2. User defined packages.

Java API packages:

API are in build java packages. which are available in javac compiler

A large number of classes grouped into different packages based on functionality

Examples

- java.lang
- java.util
- java.io
- java.awt
- java.net
- java.applet etc.

Java
awt

package containing awt package

color

Graphics

:

Image

package containing classes

Accessing classes in a package

1. fully qualified class name.

Example : java.awt.Color.

2. import packagename.classname;

Example : import java.awt.color;

or,

import packagename.*;

Example : Import java.awt.*;

User defined packages

these are the packages are created by the used based on their needed

Creating your own package

1. Declare the package at the begining of a file using a form

package packagename;

2. Define the class that is to be put in the package and declare it public

3. Create a subdirectory under the directory where the main source files are stored.

4. store the listing as classname.java in the sub directory created
5. compile the file. This creates .class file in the subdirectory.

Example:

```
package firstpackage;
public class FirstClass
{
  //Body of the class
}
```

Creating packages

Consider the following declaration

Package firstPackage.Second package;

1. A java package can contain more than one class definitions that can be declared as public
2. only one of the classes may be declared public and that class name with .java extension is the source file name

Example program:

```
Package p1;
public class classA
```

```
Public void displayA()
{
System.out.println("classA");
}}

package p2;
public class B
{
protected int m=10;
public void displayB()
{
public static system.out.println("class B");
System.out.println("m = "+m);
}}
import p1.*;
import p2.*;
Class Package Test2
{
public static void main(String Str[])
{
classA obA= new classA();
class B obB = new class B();
obA.displayA();
obB.displayB();
}}
```

Output:

```
class A
Class B
m=10
```

Package and member access.

Access level modifiers determine whether other classes can use a particular field or invoke a particular method

there are two levels.

At the top level – public or package-private

At the member level - public, private protected

Tope level access control.

A class may be declared with the modifier public, in which case that class is ~~variable~~ visible to all classes everywhere.

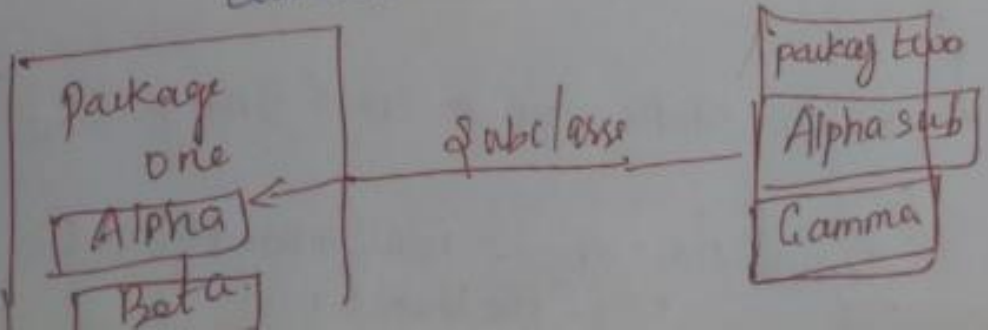
A class has no modifier.

member level access control

public → declared with public, it is visible to all classes anywhere

private → declared with private, It is accessed in that member can access only the class

protected → that the member can only the accessed within its own package.



Package and member access.

Access level modifiers determine whether other classes can use a particular field or invoke a particular method

There are two levels.

At the top level – public or package-private

At the member level - public, private protected

Tope level access control.

A class may be declared with the modifier public, in which case that class is ~~variable~~ visible to all classes everywhere.

A class has no modifier.

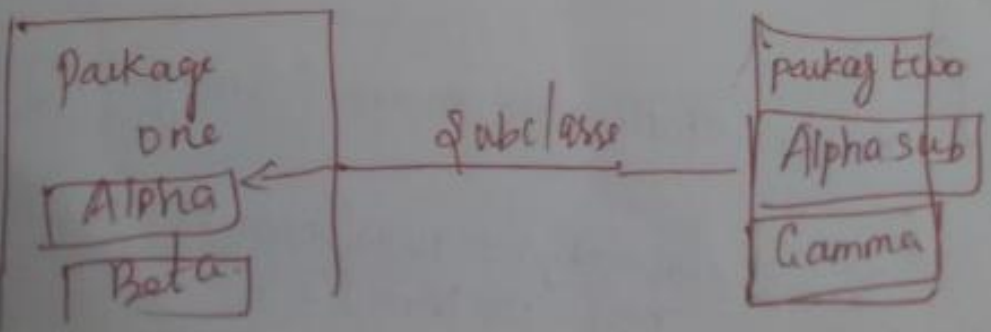
member level access control

Public → declared with public, it is visible to all classes anywhere

private → declared with private, It is accessed in that member can access only the class

protected

→ that the member can only be accessed within its own package.



Example.

z:\Mypack\FirstClass.java.

```
Package Mypack;
Public class FirstClass
    {
Public String i = "I am public variable";
protected String j = "I am protected variable";
private String k = "I am private variable";
String r = "I dont have any modifier";
}
```

z:\Mypack2\Secondclass.java

```
Package Mypack2;
import Mypack.Firstclass
Class Second class extends FirstClass {
 Void method()
 {
 System.out.println(i);
 System.out.println(j);
 System.out.println(k);
 System.out.println(r);
 }
  public static void main (String arg[])
  {
Second class obj = new Second Class();
obj.method();
}}
```

Output

I am public Variable
I am protected variable.

Interfaces

Definition

interface is a keyword which is used to achieve full abstraction, using interface we can specify what the class must do but not how it does.

interface is a collection of method definitions and constant values It is a blueprint of a class. it has static constants and abstract methods

Defining interfaces

Interface is defined much like a class.
The keyword interface is used to define an interface

Syntax

[access_specifier] interface InterfaceName
{
Datatype VariableName1 = Value;
Datatype VariableName2 = Value;
...
Datatype VariableNameN = Value;
return type MethodName1 (parameter list);

ReturnType methodName2 (parameter_List);

⋮

ReturnType methodNameN (parameter_List);

Implementing interfaces.

Syntax.

accessSpecifier Class class_name [extends SuperClassName]
implements interface_name1, interfaceName2 ...
{
// implementation code and code for the method of the interface
}

Accessing implementations through Interface Reference

1) interface reference is required to access the implementation
2) Any instance of the class that implements the interface can be referred to by such a variable

ex.

```
Interface Call
{
  Void callback (int param)
  {
  public void callback (int p)
    {
    S.o.p ("callback called with "+p);
    }}
```

```
public class testInterface
{
public static Void main (String args[])
{
 Call c = new client();
  c. callback (423);
  }}
```

O/p.

callback called with 423.

Extending interface.

An inteface can exlend another Interface, similarly to the way that a class can extend another class

Syntax.

access specifier interface InterfaceName exlends Interface1, interface2...

```
{
code for interface
}.
```

ex

```
interface Author
{ }
interface publisher
```

```
    }..
    }
interface Book extends Author, publisher
  {..
  }
```

Unit III

Exception Handling and Multithreading

Exception Handling basics - multiple catch clauses. Nested try statements - Java's Built in exceptions - user defined Exception. Multithreaded programming Java thread model - Creating a Thread & multiple threads - priorities - synchronization - inter thread communication. - Suspending - Resuming and stopping threads - Multithreading - wrappers - Autoboxing.

3.1 Exception Handling basics.

An Exception is an event that occurs during program execution which disrupts the normal flow of a program. It is a object which is thrown at runtime

In Java an exception is an object,

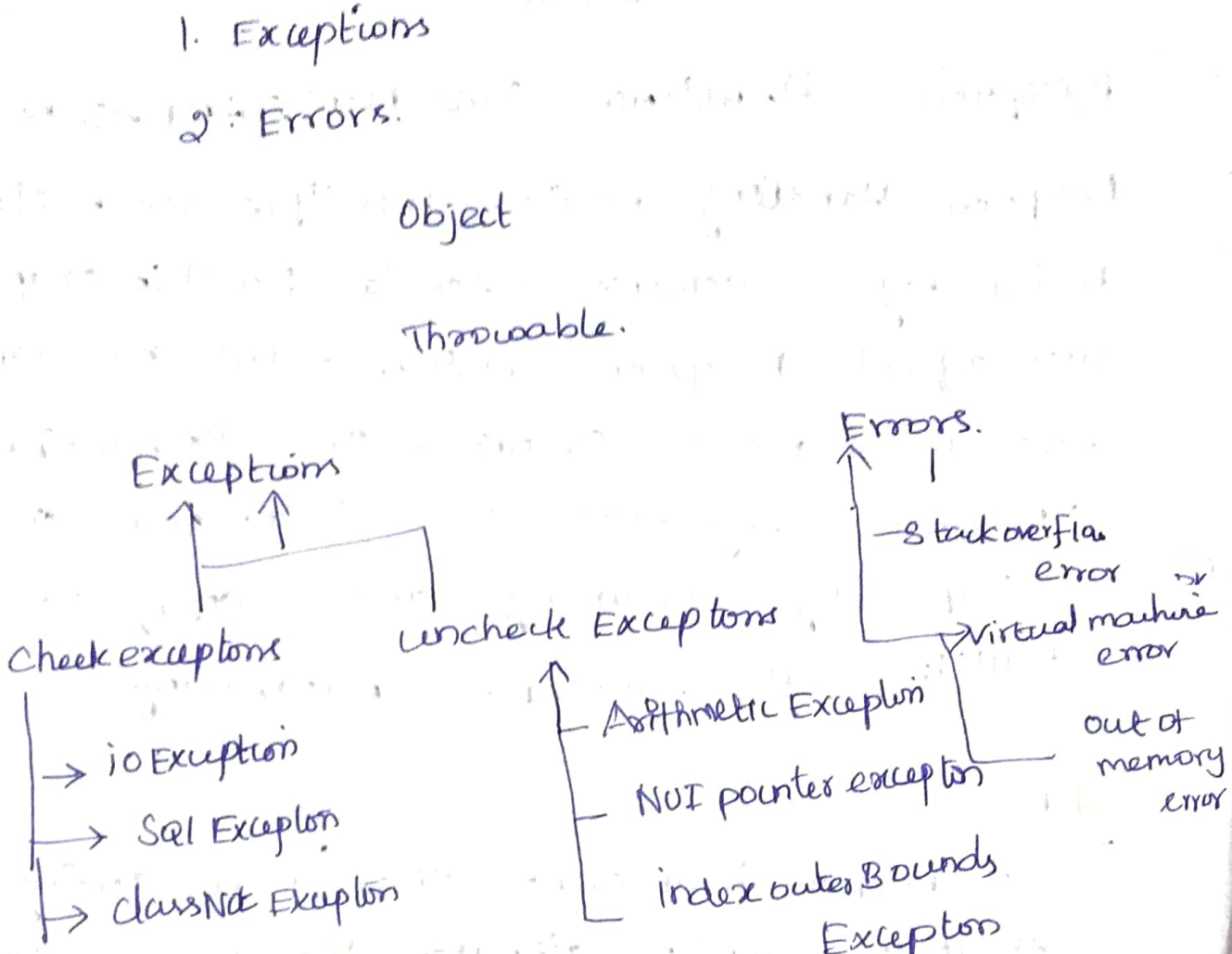
1. information about the error including its type
2. The state of the program when the error occurred.
3. Optionally, Other custom information

Exception Hierarchy

All exceptions and errors extend ~~from~~ from common Java.lang.

1. Exceptions

2. Errors.



Exceptions

Exception represents errors in the java application program, written by the user.

Sometimes it also happens that the exception cannot be caught and the program may get termenated

eg Arithmetic Exception.

Errors

Errors represent internal errors of the java run time. system which could not be handled easily.

eg: outof Memory Error.

Exception	Error.
1. Exception can be recovered	Error are of type java.lang.Error
2. Exceptions are of type. java.lang.Exception.	Errors can be recovered
3. Exceptions can be classified Into two types checked exceptions unchecked exceptions	In a case of Errors, compiler won't have knowledge of errors.
4. Checked Exception SQLException, IO exception, unchecked Exception	Ex java.lang StackOverflow Error; java.lang.outof MemoryError.

3.2 Multiple catch blocks

multiple catch is used to handle many different kind of exceptions that may be generated while running the program.

Syntax

```
try
{
  //code block
}
catch (Exception type e1)
{
 //Handle Exception Type1 exceptions
}
catch //(Exception Type2. exceptions
}
```

Example.

```
Public class Multiple catch Block 2 {
Public static Void main (String[] args)
try
{
Int a[] = {1,5, 10,15,16}
System.out.println ("a[1] = "+a[1]);
System.out.println("a[2]/a[3]="a[2]/a[3]);
System.out .println ("a[5] = "a[5]);
}
catch (Arithmetic Exception e)
{
System.out.println ("Arithmetic Exception occurs");
}
```

```
catch (ArrayIndexOutOfBoundsException e)
{
System.out.println("ArrayIndex out of Bounds Exception
                                                occured");
}
catch (Exception e)
{
 System.out.println ("parent Exception occurs")
 }
 System.out.println ("rest of the code").
```

Output

a[i]=5

a[2] /a[3]=0

ArrayIndexOutBoundsExceptionOccurs.

rest of code.

33 Nested Try Block.

try block within a try block is known as nested try block.

Why use nested try block

Some times a situation may arise where a part of a block may cause one error and the entire block itself

Example

```
Class NestedExcep
{
public static void main (String [] args)
{
try
{
int arr [] = {1, 5, 4, 10};
try
{
 int x = arr [3] / arr [1];
 System.out.println ("Quotient = " + x);
 }
 catch (Arithmatic Exception ae)
 {
 System.out.println ("divide by zero");
 {
 arr [4] = 3;
 Catch (ArrayIndexoutOfBoundsException e)
 {
 System.out.println ("array Index out of bound
                      Exception");
 }
 System.out.println ("End of program");
}}
```

O/P Quotient 2 array index out of bound

Syntax

```
...
try
{
Statement 1;
Statement 2;
 try
 {
 Statement 1;
 Statement 2;
 }
 Catch (Exception)
 {
 }...
```

... End of program ...

3.4 THROWING AND catching Exceptions

Before catching an exception it is mues to throw an Statement besides the implicit exception throw. When throw statement is called.

1. A program can explicitly throw an exception besides the implicit exception thrown.
2. we can throw either checked, unchecked exceptions or custom (user defined) exceptions

the general form of the throw statement

ex

```
Public class Throw Demo
{
 Public Ver· static void validate (int age)·
 {
   if(age<18)
 throw new ArithmeticException ("not valid").
 else
  System.out.println ("welcome to vote");
   }
 Public static Void main (String [] args[])
  {
    validate (13);
System.out.println("rest of code");
      }}
```

O/P

Exception in thread "main" java.lang.ArithmeticException : not valid at ThrowDemo.Validate (ThrowDemo.java:6)

Using throws keyword

1. The throws statement is used by a method to specify the types of exceptions the method throws.
2. If a method capable of raising an exception that it does not handle, the method must specify that the exception have to be handled by the calling method.

Syntax

```
Return_type method_name (arg_list)
                    throws exception_list
{
  //method body
}
```

Example.

```
import java.util.Scanner;
public class ThrowDemo.
{
```

```
Public voide divide (int num, int din) throws
                                    ArithmeticException
{
int result = num/dim
System.out.println ("Result : "+result);
}
Public Static voidmain (String args[])
{
int n,d;
Scanner in = new Scanner (System.in);
System.out.println ("Enter the Numerator:");
n= in.nextInt();
System.out.println ("Enter the denomunator");
 d=in.nextInt() ;
 try
  {
  devide (n,d);
  } }
  System.out. } catch (Exception e)
  {
    System.out.println ("can't Handle :divide
                                by zero error");
  }
  System.out.println ("** continue with
                        rest of the code** ");
  }}
```

o/p Enter the numerator : 4

Enter the denominator:

0

Can't Handle : divide by zero Error

** Continue with rest of the code **

Enter the Numerator:

6

Enter the Denominator

2.

Result : 3

** continue with rest of the code **

Difference between throw and throws.

throw

1. throw is used to explicitly throws an exception
2. Checked exception can not be propagated without throws
3. throw is followed by an instance
4. throw is used with in method
5. you cannot throw multiple exception

throws

1. throw is used to declare an exception
2. checked exception can be propagated with throws.
3. throw is followed by class
4. throw is used with the method signature.
5. you can declare multiple exception

are known as user defined Exceptions

to create user defined exceptions

1. pick a self describing Exception class name
2. Decide if the exception should be checked or unchecked.

Ex

```
public class EvenNoException extends Exception
{
EvenNoException (String str)
{
super(str);
}
public static void main (String [] args)
{ int arr [] = {2,3,4,5};
  int rem
  int i
 for (i=0; i< arr.length; i++)
}
rem = arr [i] % 2
try
{ if (rem==0)
throw exp
}}
S.O.P ("Exup thrown is "+exp);
}}
sys.o.pr ( arr[i] + " is an Even number
} else {
```

Output

exception in thread main java.lang.Error.
: unresolved compilation Error.

unchecked Exception.

The unchecked Exception are just opposite to the checked exceptions

example.

1. ArrayIndexOutOfBoundsException
2. Arithmetic Exception.

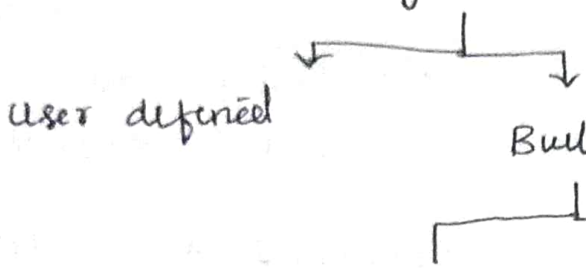
```
class Main
{
public static void main (String args[]) {
    int x = 0;
    int y = 0;
    int z = y/x;
    }}
```

o/p:
exception in thread main // java.lang.
Arithmetic Exception.

3.5. User Defined Exceptions

Exception types created by users to describe the exception related to their applications.

Types of Exceptions



Class Not found Exception
Interrupted Exception
Io Exception
Sql Exception
FileNot found Exception
Arithmetic exception
Class cast Exception
Null pointer Exception
ArrayIndex out-of Bounds Exception
Array store Exception

Built in Exceptions

Built in exceptions are the Exception which are available in java libraries those exceptions are suitable to explain certain error situations.

S.NO	Exception	Description
1.	Arithmetic Exception	Thrown when a problem in arithmetic operation is noticed by the jvm
2.	ArrayIndexOutBounds Exception	Thrown when you access an array with a illegal Index
3.	Class Not Found Exception	Thrown when you try to access a class class which not defined
4.	file Not Found Exception	Thrown when you try to access a non existing file

IndexOutBoundException — some type of Index is out of bounds

NegativeArraysizeException — invalid use of null reference

StringIndex outofBounds — Attempt to index outside the bounds of a string

Checked Exceptions

Example

1) ClassNotFoundException
2) cloneNot supported Exception
3. IOException.

checked exceptions are called compile time exceptions because these exceptions are checked at compiler time by compiler

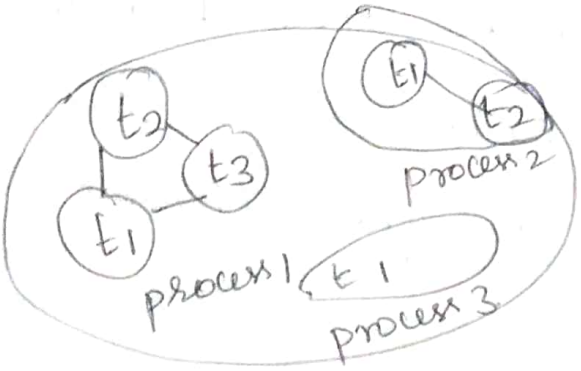
without try catch

```
import java.io.*;
Public class CheckedExceptionExample
{
Public Static Void main (String [Jargs)
FileReader file = new FileReader ("src/someFile.txt");
}}
```

3.6 Multithreaded programming

A thread is a lightweight sub process that defines a separate path of execution. It is the smallest unit of processing that can concurrently with the other parts of the same process

1. Threads are independent
2. if there occurs exception in one ~~method~~ thread, it doesn't affect other threads.
3. It uses a shared data.



In above figure, a thread is executed inside the process.

Difference between Thread and process.

Thread	process.
Thread is a light weight process.	process is a heavy weight program
~~each process has a complete set of its own variables.~~ each threads share a same data	each process has a complete set of its own variable

Cost of communication threads is low	cost of communication thread is high.
thread ~~process~~ are ~~dependent~~ of one another	process are independent of one another.
All thread of particular process shares the common memory and resources	Each process has its own memory and resources.

Multithreading

A program can be divided into number of small processes each small process can be addressed as a single thread

Definition

is a Technique of executing more than one thread performing different tasks simultaneously

Advantages

1. Threads are light weight compared to processes
2. Thread share the same address space. and ~~interface~~ therefore can shake both data and code
3. Cost of thread communicaton is low than inter process communications

Multitasking

multitasking is a process of executing multiple tasks simultaneously.

Multitasking can be achieved two ways.

1. process based multitasking (multiprocessing)

   It is a feature of executing two or more programs concurrently.

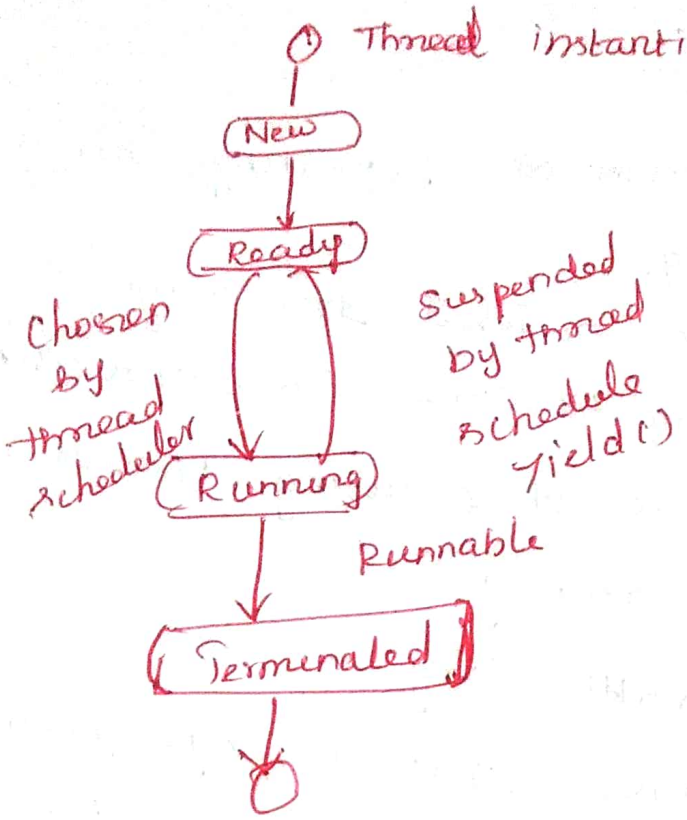
2. Thread base multitasking.

   It is a feature that a single program can Perform two or more tasks simultaneously.

3.1 Java thread model.

Different states, a thread (for applet / servlet) travels from its object creation to object removal is known as life cycle of thread.

1. New state
2. Runnable state
3. Runnable state
4. waiting / Timed waiting
5. Terminated state / dead state



New State

A new thread begens its life cycle in the new state.

1. start() method → which places the thread in the runnable state

when th thread is in this state only start() and stop() methods can be called.

sample Thread MyThread = new Thread();

Runnable state:

After a newly born thread is started a thread becomes runnable or running by calling run() method.

A thread in this state is considered to be executing its task

myThread.start();

3. Running State

Thread scheduler selects thread to go from runnable state. In running state Thread starts executing by entering run() method.

waiting / Timed waiting / Blocked state

Sometimes on thread has to undergo in waiting state because another thread starts executing

A runnable thread can be moved to waiting state by calling the wait()

- A call to notify() and notifyAll() may bring the thread from waiting state by calling the wait

Thread waiting

Sample code

```
try
{
Thread.sleep(3*60*1000);   thread sleeps for 3 minutes
}
catch(InterruptedException exe)
{
}
```

Blocked state

1. This can be achieved by calling suspend() method

2 After the I/O completion the thread is sent back to the runnable state.

Terminated State

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

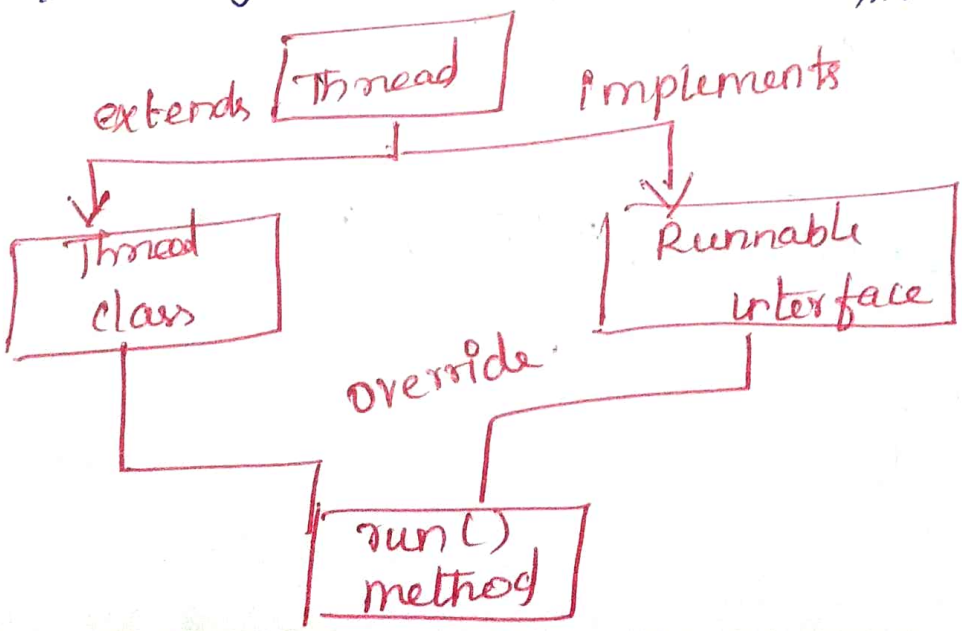
Terminates (when the stop() is invoked)

My.Thread.stop();

3.8 Creating Threads

we can threads by instantiating an object of type thread.

1. By implementing Runnable interface (java.lang.Runnable)
2. By extending the Thread Class (java.lang.Thread.)



creating threads by implementing Runnable interface

```
Public void run()
  {
  // implementation code
  }
```

contains the logic of the thread.

Steps for thread creation

1) Public Class MyThread implements Runnable
   {
   ...
   }
2) override the run() method to define the code executed by the thread.
3. create an object type
   Thread t = new Thread (Runnable Thread obj, String threadName);
4. invoke the start() method
   t.start();

2) Creating thread by extending Thread class

Thread can provide constructors and methods to create and perform operations on a thread.

Thread ()

Thread (String name)

Thread (Runnable r)

Thread (Runnable r,)

Thread (Runnable r, String name)

Comonly used method of thread class

public void run()

public void start()

public void sleep (long milliseconds).

Public void join ()

Steps for thread creation

1. create a class that extends java.lang.thread Class

Public class My Thread extends Thread

{ ...

}

2. override the run() method in th sub class to define the code executed by the thread

3. create object of Subclass

MyThread t = new MyThread (String threadName);

4. invoke the start () method for running.

Start ();

3.9 Thread priority

Thread priority determenies how a thread should be treated with respect to others

priorities are represented by number between 1 and 10

1- minimun priority
5- Normal priority, 10- maximum priority

Thread scheduler is a part of java virtual machine. It decides which thread should excute first among two or more threads that are awaiting for excution

constants defined in thread class

1. public static MIN priority
2. public static int Norm priority
3. public static in max priority

To set a priority, use the set priority() method

To obtain the current priority of a thread. use get priority() method

Example.

```
class Test Multipriority1 Extends thread {

public void run()
    System.out.println("running thread name is :"
```

```
+ thread . current Thread() . get Name() );
System out . println (" running thread priority is "+
        Thread . current Thread (). get priority();
}
Public static void main (String args[]) {
Test MuHiPriority m1 = new Test MuHiPriority1();
Test MultiPriority m2 = new Test Multi priority1();
m1. set priority (Thread MIN. PRIORITY);
m2. Set priority (Thread. MAX. PRIORITY);
    m1. Start();
    m2. start();
```

O/P

running thread name is . Thread 0
running thread priority is 10
running thread name is Thread 1
running thread priority

3:10 Thread, synchronization.

Definition

Thread synchronization is the concurrent execution of two or more threads that share critical resources.

Why use Synchronization

1. To prevent thread interference
2. To prevent consistency problem

Thread synchronization

they are two types of thread synchronization

① mutual exclusive and inter thread communication

1. synchronized method
2. synchronized block
3. static synchronization

② cooperation

Mutual Exclusive.

Mutual Exclusive helps keep threads from interfering with one another while sharing data

1. by synchronized method
2. by synchronized block.

Java synchronized method

1. If you declare any method as synchronized it is known as synchronized method.

Syntax

Access modifier synchronized return type method_name (parameters)

{ ... }

Synchronized block in java

Synchronized block can be used to perform Synchronization on any specific resource of the method

points to remember for synchronized block

→ synchronized block is used to lock an object for any shared resource.

→ scope of synchronized block is similar than the method

Syntax

```
Synchronized (object reference expression)
{
//code block.
}
```

Difference between synchronized method. Synchronized block

Synchronized method	Synchronized block
Lock is acquired on whole method	Lock is acquired on critical block of code only
less preferred	preferred.

Performance will be less as compared to synchronized block	Performance will be better as compared to synchronized method.

3:11. Inter thread Communication

Def:

Inter thread communication is a mechanism in which a thread is paused running in its critical section and other thread is allowed to enter in the same critical section to be executed

1) public final void wait() throws InterruptedException.

2) public final void wait (long timeout) throws InterruptedException)

3. public final void notify()

4. public final void notifyAll()

3:12. Suspending, Resuming and stopping threads

The function of suspend Resume, and stop a thread is performed using Boolean type flags in a multithreading

Program

1. If the suspend flag is set to be true then run() will suspend the execution of the currently running thread.
2. If the resume flag is true then run() will resume the execution of suspended thread
3. If the stop flag is set to true then a thread will get terminated.

3.B. Wrapper class

For all the primitive data types in java, there is a corresponding object representation available which is known as wrapper class.

Need for wrapper class.

- All collection classes in java can store only objects
- primitive datatypes cannot be stored directly in these classes and hence the primitive values needs to be converted to objects.
- we have to wrap the primitive data types in a corresponding object and give them an object representation
- process of converting the primitive data types into objects is called wrapping.

```
int i = 10;
Integer I = new Integer(i)
```

primitive data types	
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float

Boolean class

Boolean class wraps a value of the primitive type boolean in an object.

Field	Description
static Boolean FALSE	This is the Boolean object corresponding to the primitive value false
static Boolean TRUE	This is the Boolean object corresponding to the primitive value true.

Boolean has different default methods

method	Description
boolean boolean value()	this method returns the value of this Boolean object as a boolean primitive
int compareTo(Boolean b)	this method compares this Boolean instance with another
boolean equals(Object obj)	this method returns true if and only if the argument is not null

Character class

Character class wraps a value of the primitive type char in an object

An object of type character stores unicode characters

Methods	Description.
Static int compare. (char x, char y)	compares two char value
int CompareTo (character another character)	compares two character objects numerically.

Integer class

- this class wraps a value of the primitive type int in an object
- An object of Integer class contains a single field of type int value.

Field	Description
Static int MAX_VALUE	A constant holding the maximum value an int can have $2^{31} - 1$.
Static int MIN_VALUE	A constant holding the minimum value an int can have 2^{31}

Double class

Double class generally wraps the primitive type double into an object.

An object of Double class contains a field with the type Double.

Method	Description
byte byteValue()	Returns the value of this Double as a byte.
static int compare (double d1, double d2)	Compares the two specified double values
boolean equals (object obj)	→ Compares this object against the specified object
float floatValue()	→ Returns the float value of this Double object
boolean isInfinite()	

Autoboxing

→ Java 5.0 included automatic conversion between a primitive type and the corresponding wrapper class

→ During assignment the automatic transformation of primitive type to corresponding wrapper type is known as auto boxing.

Primitive types ----> wrapper type (auto boxing)

the automatic transformation of wrapper type into their primitive equivalent is known as unboxing.

Eg int i=0;

i = new Integer(10);

Boxing conversion converts values of primitive type to corresponding value of reference type. But the primitive type cannot be widened/ Narrowed to the corapper classes and vice versa

Eg 1: byte b = 12;

Integer I1 = (int) b; is correct procedure for auto boxing)

Eg 2: byte b = 12;

Integer I1 = ~~Integer~~ b (is wrong as b can be auto boxed only for int data type:

Eg3 3 byte b=12.

Integer I1 = (Integer) b; (as autoboxing will be done only from primitive data type)

unit IV

I/O Generics, STRING HANDLING.

I/O Basics – Reading and writing console I/O. Reading and writing files – Generics. Generic programming – Generic classes Generic methods – Bounded types – Restrictions & limitations of Generics – String Basic string class – methods – String buffer class.

I O Basics :

1. Java I/O (input & output) is used to process the input and produce the output based on the input.
2. Java uses the concept of stream to make I/O operation fast. The java.io package contains all the classes required for input and output operations

Streams :

1) Java program perform I/O through Streams
2) A Stream is an abstraction that either produces or consumes information
3. A Stream is linked to a physical device by the Java I/O system.
4. All streams behave in the same manner. even if the actual physical device to which they are linked differ.

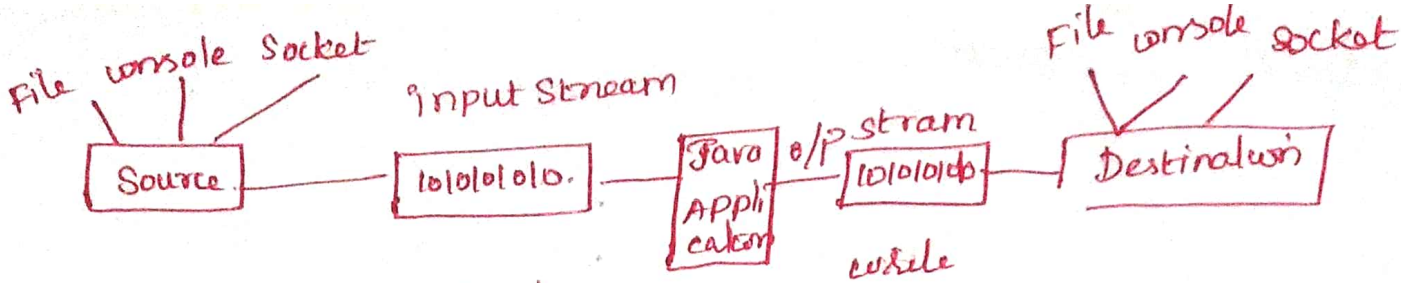
Input Stream can abstract many different kinds of input

from a disk file, a keyboard or a network socket

output

Const a disk file or a n/w connection

Java implements Streams with in class hierarchies defined in the java.io package.



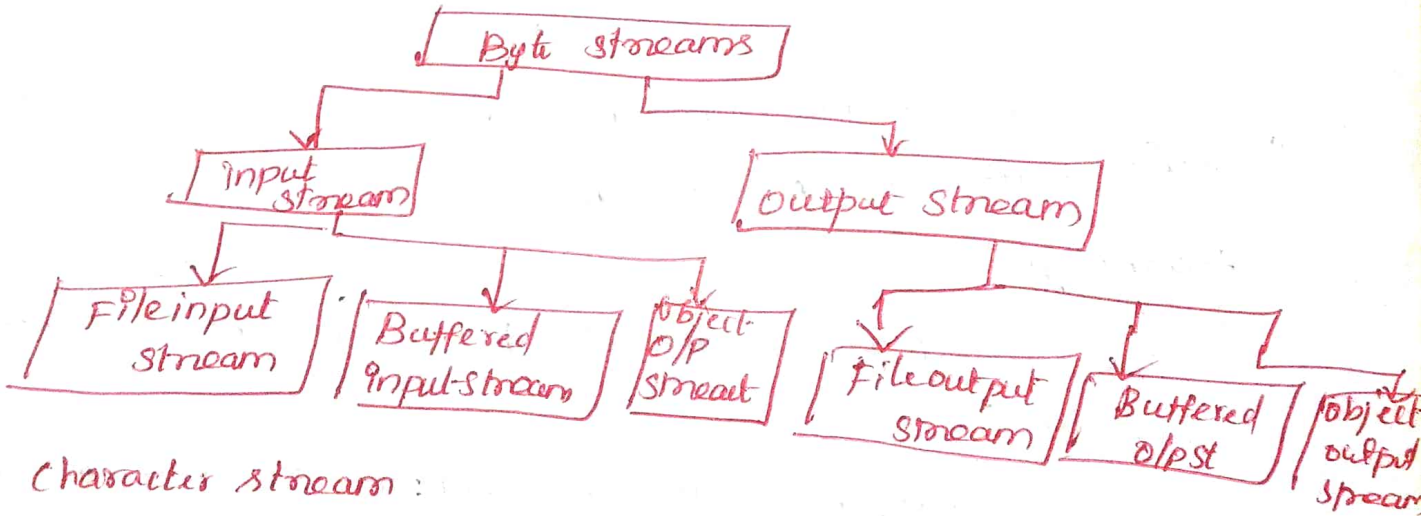
System . out . Standard output stream.

System . in : Standard input stream

system . err Standard error steam

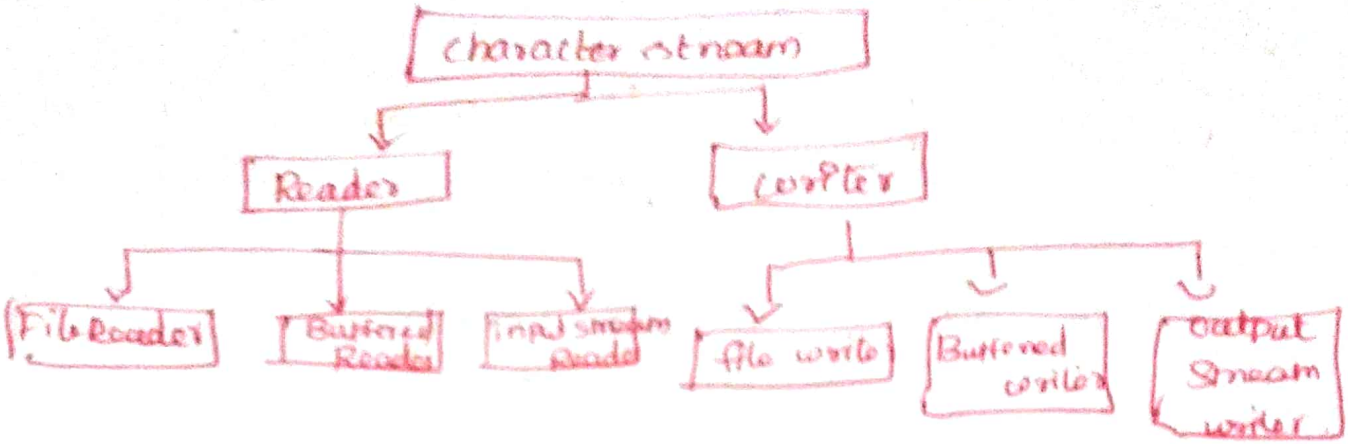
Byte streams.

Byte streams are defined by using two class hierarchies Input streams and output stream. each of these abstract classes has several concrete subclasses that handle the differences among various devices such as disk files. network connections and even memory buffers.



Character stream :

- ✓ Provide a conveient means for handling input and Output of characters.
- ✓ use of unicode and therefore can be internationalized.



3. Reading & writing Console.

1) while reading content from the console all the data will be string type.

2. They are four major methodologies

1. using Data Input Stream class
2. using Buffered Reader class
3. using Scanner class
4. using stream class.

1) using Buffered Reader class.

Advantages

The input is buffered for effcient reading

Draw back

The wrapping code is hard to remember

2) Using Scanner class.

Advantages

1. convenient methods for parsing primitives.

nextInt(), nextFloat(); } from the tokenized input.

Drawback

The reading methods ~~are not~~ are not synchronized

using console class.

Advantages

Reading password without enhoing the entered characters

Reading methods are synchronized

format string syntax can be used.

Draw back

Does not work in non interactive environment

Java console class

The Java console class is used to get input from console. It provides methods to read texts and passwords.

if you read password using console class, it will not be displayed to the user

Java.io.console class is attached with system console internally

Sample:

1. String text = System.console().readLine();
2. System.out.println("text is "+text);

Java console class declaration

Reader reader()

String readLine()

String readLine(String fmt, Object... args)

char[] readPassword()

char[] readPassword(String fmt, Object... args)

Console format(String fmt, Object... args)

Void flush()

How to get object of console

System class provides a static method console() that returns the singleton instance of console class

public static Console console(){}

Java console example

```
import java.io.console;
Class ReadStringTest
{
```

```
public static void main (String args[])
{
Console c = System.console();
System.out.println ("Enter your name :");
String n = c.readLine();
System.out.println ("welcome "+n);
}}
```

O/p

enter your name abcd

Welcome abcd.

4.3 Reading writing files

File handling java implies reading from and writing data to a file.

The file class from the java.io package allows us to work with different formats of files.

in order to use the file class, you need to create an object of the class and specify the file name or directory name.

Example

1) // import the File class
2) import java.io.File.

// Specify the filename

File obj = new File ("filename.txt");

The file class has many usefal methods for creating and getting information about files

example

Method	Type	Description
canRead()	Boolean	Tests whether the file readable or not
canwrite()	"	Tests whether the file is writable or not
CreateNewFile()	"	Creates an empty file
delete()	"	Deletes a file
exists()	"	Test whether the file exists
getName()	string	Return the name of the file
GetAbsolutepath()	"	Return the absolute pathname of the file
Length()	Long	Returns the size of the file in bytes
List()	String[]	Returns an array of the files in the directory
mk dir()	Boolean	Creates a directory.

File operations in Java.

You can perform four operations on a file

1. create a file
2. Get file information
3. Write to a file.
4. Read from a file.

creates a file.

To create a file in java you can use the createNewFile() method. This method return a bookan value.

Examples

```
import java.io.File;
import java.io.IOExcaption
public class CreateFile
{
public static void main (String [] args)
try
{
File Myobj = new File ("filename.txt");
if (myobj.CreateNewFile ()) {
}
System.out.println ("File created : "+ myobj.getname());
}
else
{
System.out.println ("File already excists");
}
} catch (IOExcaptione) {
System.out.println ("An error occured");
e.printStaickTrace();
}}
```

o/p.

File created : filename.txt.

write to a file.

In the following example, file writer class together with its "write()" method to write some text to the file we created in the above example.

Example

```
import java.io.File
Import java.IoExceptions
Public class CreateFile()
{
public Static void main (strings[] args)
 {
 try
 {
 File myobj = new File("filename.txt");
 if (myobj
 FileWriter mywriter = new FileWriter ("filename.txt);
 my writer.write("Files in java might be tricky,
                  but it is fun enough!");

 my writer.close();
 System.Out.println("successfully wrote to
                     the file");
 } catch (IOException e) {
 System out println("An error occured");
    } }
```

Output: Successfully write to the file.

Class reference declaration

Classname <type-arg-list> var. name = new class-Name
<type-arg-List> (cons-arg list);

Type parameter Naming conventions

Type parameters is place holder for a type argument
By convention, type. parameter name are single uppercase letters

E → Element

K → Key

N → Number

T - Type

V - value

u, v et = 2nd, 3rd, 4th types

Generic Methods

A generic method is a method with type parameter we can write single generic method declaration that can be called with arguments of different types, Based on the types of the arguments passed to the generic method. the compiler handles each method can appropriately

Syntax

accessspecifier class Generic classname <Type variable1, Type variable2 ...>

{

Instance variables

Constructors

methods

}

Example

```
public class pair <T, S>
{
private T First;
private S Second;
...
public T getFirst ()
{ return first; }
...
}
```

Instance variable data type

A method with a variable return type

Example:

```
public class pair (T, S)
{ ...
```

Limitation

1. explicit casts must be employed to retrieve the stored data.
2. Type mismatch errors cannot be found runtime

Need for Generic.

1) It allows the code reusability.
2. compact can be created

Advantages of Java generics

1) Code reuse
2) type safety.
3. elimination of casts.

```
List list = new ArrayList();
list.add("hello")
String s = list.get(0);
```

4. Stronger type checks at compiler time

```
List<String> list = new ArrayList<String>();
list.add("hello");
list.add(32)        // compile time error
```

5. Enabling programmers to implement generic algorithms.

Generic classes.

A class that can refer to any type is known as generic class Here we are using T type parameter to create the generic class of specic type.

```
Public  Class NonGenDemo
{
public static voidmain (String [Jargs)
 {
NonGen inlegerObj ;
inleger obj =new NonGen(88);
inleger obj . ShowType();
int v = (Inleger) inleger obj. getob();
System . ow . println ("Value = "+v);
NonGen Str obj = new NonGen ("Non Generic Test");
Str obj . Show Type().
String str = (String) Str obj getob();
System . ow . println ("Value = "+str);
  }}
```

O/p

Type of ob is java . lang. Inleger

Value =88

Type of ob is java. lang. String

Value = Non Generics Test.

Generic programming.

Definition

Generic programming is a style of computer programming in which algorithms are written in terms of "to be specified later" types that are then instantiated when needed for specific type provided as parameters

Non Generics

In Java there is an ability to create generalized classes interfaces and methods by operating through Object class

Or

```
Class NonGen
{
 object ob;
 NonGen(object o)
   {
   }
 object getOb()
    {
    return ob;
    }
 void showType()
    {
    S.O.P ("Type of obj is "+ob.getclass().getName());
    }}
```

Read a File:

In the following example, we use the scanner class to read the contents of the text file, we created in the previous example.

```
import java.io.File.
import java.io.FileNotFoundException : // import this class to handle errors
import java.util.Scanner ; // import the scanner
 public class ReadFile.
{
public static void main (String[] args)
 {
  try
  {
  File MyObj = new File ("filename.txt");
  While (MyReader.hasNextLine() }
  {
String data = myReader.nextInt ();
}
 MyReader.close ();
}
catch (FileFound Exception e) { }
System.out.println ("An error occured")
  e.printStackTrace();
          }}}
```

O/P

Files in Java might be tricky - but fun enough!

Rules

→ All generic method declarations have a type parameter section delimited by angle brackets

→ each type parameter section contains one or more type parameters separated by commas.

→ The type parameter also known as a type variable

→ the type parameter can be used to declare the return type and act as placeholder for the type of the arguments passed to the generic method which are known as actual type arguments

Syntax

Declaring a Generic method

modifiers < Type variable, Type variable ... >

return type

{ methodName (parameters)

body

}

Example

```
public static <E> void print (E[] a)
{
  for (E e : a)
      System.out.print (E[]a).
      {
```

```
System.out.println();
}
```

Example

```
class a <T>
{
<T> void show (T [] e1)
  {
    for (T x : e1)
System.out.println(x);
      }}
Public class Gen method
{
public static void main (String arg[])
{
System.out.println ("Integer assay");
a <Integer> 01 = new a <Integer> ();
Integer [] as = {10, 67, 23};
 01.show (as);
System out println ("string array")
a <String> 02 = new a <string> ();
String [] as1 = {"Hai", "Hello", "welcome",
                 "to", "Java programming"};
  02 Show (as1);
 System.out.println ("Boolean assay");
  a < Boolean> 03 = new a <Bookan>();
```

```
Boolean [] as2 = {true, false};
    03.show(as2);
System.out.println("Double assay");
A<Double> 04 = new A<Double>();
Double [] as3 = {10,234, 67.451,
                 23.90};
    04.show(as3);
}}
```

O/P

Integer assay

10
67
23

String array

Hai
Hello

Welcome
to
Javaprogramming

Boolean Array

true
false

Double assay

10.234
67.451
23.9

Generic with Bounded types

Bounded type parameter is a type parameter with one or more bounds, the bounds restrict the set of types that can be used as type arguments and give access to the methods defined by the bounds.

Syntax

< T extends super class >

```
Public class GenBounds <T extends Number>
{
  T [] nums;
  GenBounds (T [] obj)
    { nums= obj ;
    }
  double average ()
  {
  double sum=0.0;
  for (int i=0; i<nums.length ; i++).
  Sum+= num[i].doubleValue();
  double avg = sum/ nums.length ;
    return avg ;
    }
Public static void main (String [] args)
{
  Integer inum [] = {1,2,3,4,5}
  GenBounds < Integer > iobj = new GenBounds <
                    Integer> (inum);
```

```
system.out.println("Average of Integer Numbers:"
        +obj.average());
Double dnum[] = {1.1, 2.2, 3.3, 4.4, 5.5};
GenBounds<Double> dobj = new GenBounds<Double>
                                (dnum);
System.out.println("\n Average of Double numbers:"
        dobj.average());
String snum[] = {"1", "2", "3", "4", "5"};
GenBounds<String> sobj = new GenBounds<String>
                                (snum);
System.out.println("Average of integer
                Number "+Objaverage);
}}
```

O/p

```
F:\> Java GenBounds.
Average of integer numbers = 3.0
Average of Double numbers = 3.3
```

Restrictions AND Limitations of Generics

In Java, generic types are compile time entities the run time execution is possible only if it is used along with raw type.

Syntax.

Stack<int> is not allowed.

```
public class Test<T> extends Exception
{
  //code
  }
```

4. Instantiation of generic parameter T is not allowed.

for example

```
new T();
new T[10];
```

5. Array of parameterized types are not allowed

for example.

```
New stack<String>[10]; //error.
```

String

String is a sequence of characters, But in java string is an object that represents a sequence of chacters. the Java.lang.string class is used to create string object.

How to create string object?.

By string literal

By string literal is created by using double quotes

for example:

```
String s = "Welcome";
```

By new Keyword

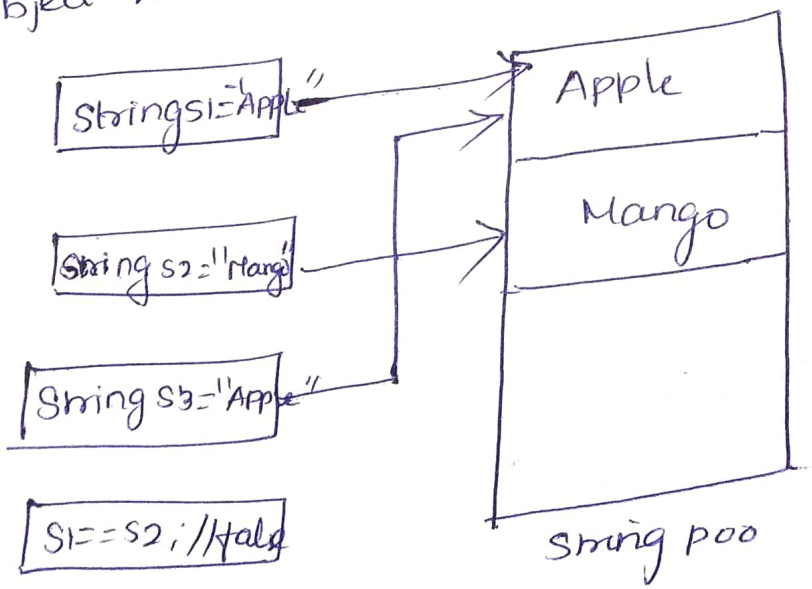
Java string is created by using a keyword "new")

forexample

```
String s = new String ("welcome";
```

Java string pool

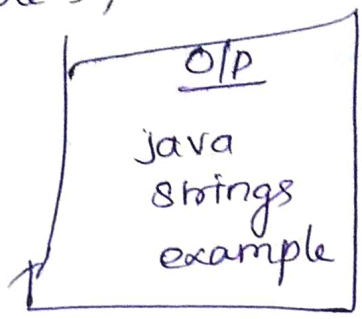
Java string pool refers to collection of strings which are stored in heap memory., in this whenever a new object is created.



example creating strings

```
public class stringName {
```

```
Public static Void main (String args[])
{
String s1 = "java";
char ch[] = {'s','t','r','i','n','g','s'}
String s2 = new string (ch);
String s3 = new string ("example");
System.out.println(s1);
System.out.println(s2);
System.out.println(s3);
}}
```



Immutable string in java

In Java String object are immutable. immutable simply means un modifiable.

1. once string object is created it data or state can't be changed but a new string object is created

```
Class simple
{
Public static void main (String args[])
String S = "Sachin";
S.concat ("Tendulkar");
System.out.println(S);
}}
```

O/p Sachin

Methods

Methods of string class in java.

Java.lang string class provides a lot of methods to work on string. By the help of these methods, we can perform operations on string such as trimming, concatenating converting strings etc..

Important methods

public boolean equals (object an object)

Public boolean equals ignoreCase (String another)

Public String concat (String str)

Public int compareTo (string str)

Public string substring (int beginIndex)

Public Stringtrim()

Public boolean start with (String prefix)

Public boolean ends with (String suffix)

String Compassion

1. we compare the two given strings
2. It is used in authentication sorting, reference matching etc..
3. there are three ways.
   By equals() method

By == operator

By compareTo() method

By equals() method

1. public boolean equals (Object another) {}
2. public boolean equalsIgnoreCase (String another) {}

Ex

```
Class simple
{
public static void main (String [] args)
{
String S1 = "sachin";
String S2 = "Sachin";
String S3 = new String ("sachin");
String S4 = "Saurav";
System.out.println( S1.equals (S2));
System.out.println (S1.equals (S3));
System.out.println (S1.equals (S4));
}}
```

O/P

true
true
true

String Buffer class

String Buffer class is used to created mutable. the string buffer class is same as string except is mutable.

string buffer can be changed dynamically.

String Class.	String Buffer Class
String objects are constants and immutable	String Buffer objects are not constants and mutable
String class supports Constant Strings	String Buffer class supports growable and modifiable strings
the methods in the String Class are not synchronized	the methods of the string Buffer class can be synchronized

ex

```
class String Buffer Example
{
public static void main (String args [])
{
String Buffer sb = new String Buffer ("Hello");
Sb. append ("Java"); //now orginal string is changed
System. Out. println (sb); // prints Hello
}}
```

Java

Unit IV

JAVA FX EVENT HANDLING, CONTROLS AND COMPONENTS

JAVA FX Events and controls - event basics - Handling Key and mouse events. controls ~~checkbox~~ checkbox, Toggle button - RadioButtons - List view - comboBox. ChoiceBox - Text Controls - Scrollpane - Layouts - ~~Flow~~pane - HBox and VBox - Borderpane - Stackpane Gridpane - menus - Basics - Menu - Menubars - Menuitem.

JAVAFX Events and Controls.

Introduction:

"JavaFx is a set of graphics and media packages that enable developers to design, create, test, debug and deploy desktop applications and rich internet applications (RIA) that operate consistently across diverse platforms.

Features of JavaFX:

feature	
Java Library	It consists of many classes and interfaces that are written in java
FXML	FXML is the XML based declarative makeup language.
Scence Builder	Scene Builder generates FXml make up which can be ported to IDE

Web view — Web view uses WebKit HTML markup technology to embed webpages into the Java Applications

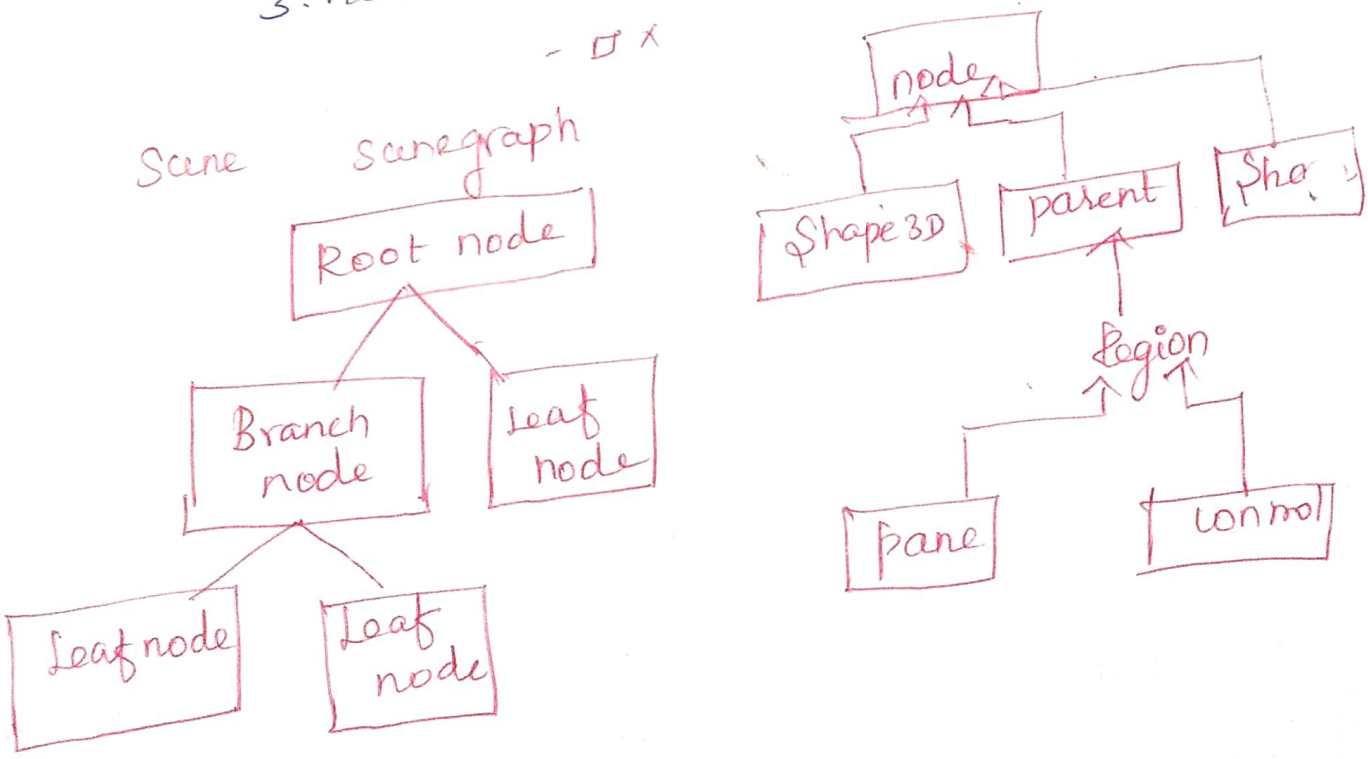
Canvas API — canvas API provides the methods for drawing directly in an area of a JavaFx Scene.

Rich Set of APIs - JavaFx provides a rich set of API to develop GUI application

JavaFx Application Structure

A Javafx application structure will have three major components

1. stage
2. Scene
3. nodes.



Stage

Stage in a JavaFX application is similar to the prime in swing Application It acts like a container for all Javafx objects

Primary stage is created internally by the platform Other stages on further be created by the application

A Stage has too parameters.

1. Decorated
2. undecorated
3. Transparent
4. unified.
5. utility

Scene

A scene represents the physical contents of a JavaFX application . It contains all the contents of a scene graph.

Scene Graph and Nodes.

A scene graph is a tree like data structure representing the contents of the scene graph

The class Scene of the package Javafx.Scene represents the scene Object.

A node may include.

Geometrical → object 2D such as
circle, rectangle, polygen

UI controls such as Button, Checkbox, ChoiceBox, Text area etc.

Containers – such as Border pane, Grid pane

media elements such as Video and image objects

Life cycle of a JavaFX Applications:

1. Launch() – to Launch JavaFX application
2. Init() – An empty method
3. Start()
4. Stop()

Launch()
↓
init()
↓
start
↓
stop()

General rules for writing JavaFX Application.

A JavaFX Application must extend Javafx.application.Application

The main() method should call Application.launch().

Start() called when a stage is connected.

The content of the scene is represented as a hierarchical scene graph.

Scene is the container for all content.

```
public class HelloWorld extends Application
{
public static void main (String [] args)
{
```

```
    Launch(args);
}
@Override
public void start(Stage primaryStage)
{
primaryStage.SetTitle("Hello world");
Stackpane root = new Stackpane();
Button btn = new Button();
btn.SetText("say hello world");
root.getchildren().add(btn);
Primary stage.SetScene(new Scene(root, 300
                                 , 250);
primary stage.Show();
    }
}
```

JAVA FX Events

A GUI based applications are mostly driven by events events are the action that the user performs and the responses the application generates.

Example : Button clicks by user. key press on the application etc.

An event is a notification about a change It encapsulates the State Changs in the event source.

Examples

Action events :

widely used to indicate things like when a button is pressed

Class Action event

Actions - button pressed.

class : Mouse Event

Actions - Mouse clicked, mouse pressed, mouse released mouse moved, mouse entered taget

Drag Event, occurs mouse is dragged.

Class Drag Event

Action - drag entered,

Key Event - indicates that a keystroke has occurred

Class KeyEvent

Actions - key pressed key released key typed.

window event:

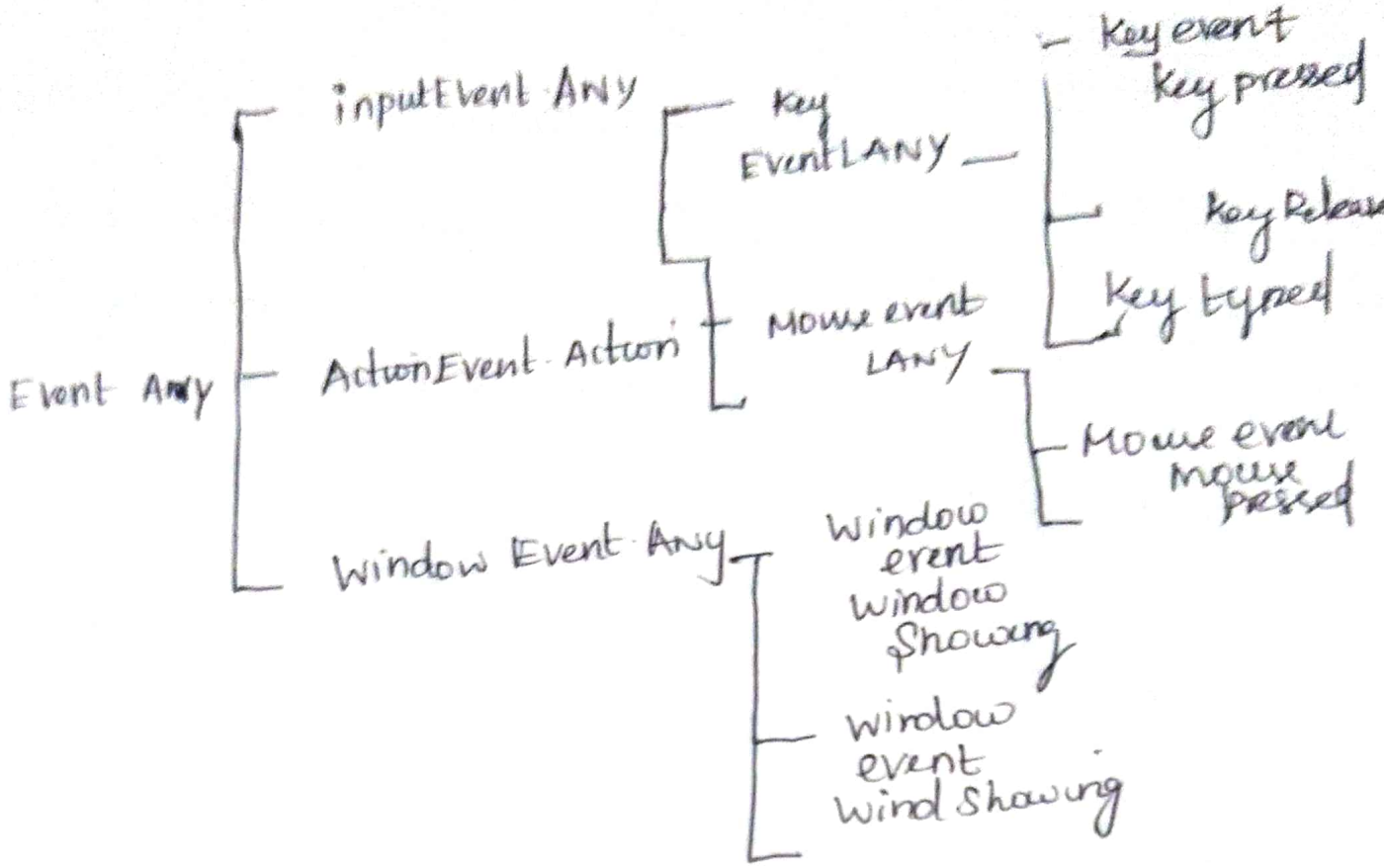
Class window event

Actions - window hiding

scroll Event => indicated scrolling by mouse wheel,

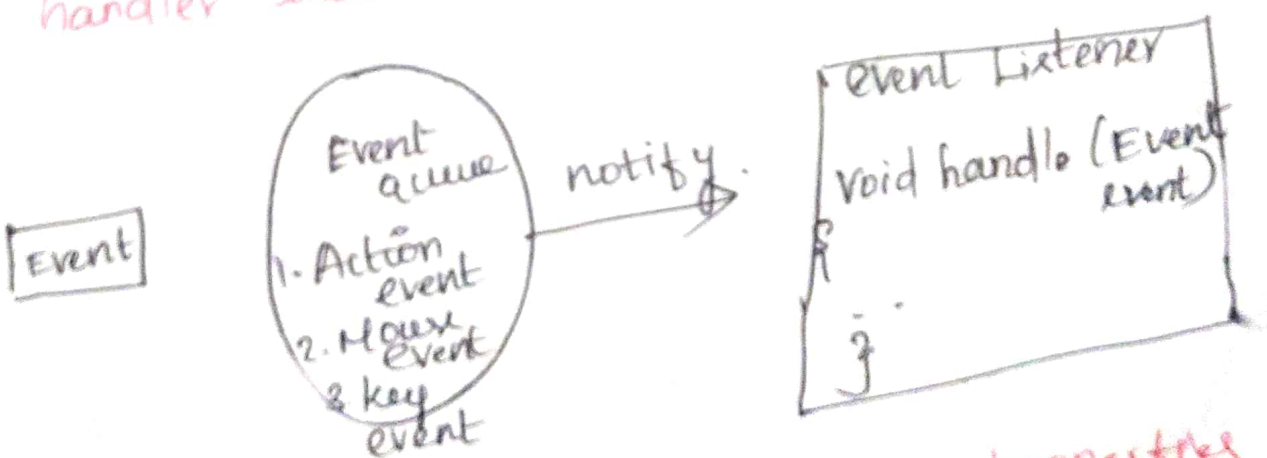
Touch Event → indicates a touch screen action.

Types of events.



Event handling:

Event handling is a Mechanism that controls the event and decides what should happen if an event occurs It has one code which is known as event handler that is executed when a event occurs



Every event in JavaFx has three properties

1. event source
2. Event target
3. Event type.

Property	Description
Event source.	It denotes source of the event ies Origin which is reponsible for generating the event.
Event target	It denotes the node on which an event is created.
Event type	It is the key of the event that is being generated.

Phases of event handling in JavaFx

1. Target selection → Depends on the particular event type
2. Route construction → Specified by the event target
3. Event capturing → event travels from the stages to the event target
4. Event Bubbling → event travel back from the target to the stage

Three methods.

1. Convenience methods
   - ↙ Set onkey pressed
   - ↙ Seton mouse clicked
2. Event handler / filter Registration methods
   - add Event handler
   - add Event filter
3. Event dispatcher property.

Event filter.

addEventFilter()

Syntax

```
node.addEventFilter(<EventType> new EventHandler<EventType>()
{
public void handle(EventType)
 {
  actual logic
  });
```

Removing event filter

```
$node.removeEventFilter(<input-event> filter);
```

Event handlers.

Event filters provides the way to handle the events generated by the keyboard Actions, mouse actions, scroll actions and many more event sources.

Adding event handler to a node.

```
node.addEventHandler(<EventType>, new
        EventHandler<Event-Type>()
{
public void   handle(<Event-Type> e)
{
  // Handling code
  }).
```

Removing Event Handler

Syntax.

node.removeEventHandler (<EventType>, handler);

A node can register for more than one filter and handler.

the interface javafx.event.EventHandler must be implemented by all the event filters and event handlers.

Handling key and Mouse Events:

As a general rule, the events you will most often handle are generated by controls.

Key events:

It is an input event that indicates the key stroke occured on a node.

It is represented by class named KeyEvents

The event includes actions like key pressed, key released and key typed.

Types of Key Event in java.

Key Pressed → key on the keyboard pressed the event will be triggered.

Key RELEASED → this event will be excuted

Key TYPED → The event will be triggered when a unicode character is entered.

Methods in the KeyEvent class to get the key details:-

Keycode getcode() - This method returns the key information or the keycode enum constant Linked with the pressed.

String getText() - This method returns a string description of the keycode Linked with the key pressed and key Released events

d

String get character - This method returns a string representing a character

Example

```
import java.applet.Applet;
import java.awt.*;
import java.awt.event.*;

Public class KeyboardEvent extends Applet implements
                                    KeyListener
{
  TextArea tpress, trel
  Textfield t;
  public void init()
  {
  t= new TextField(20);
  t.addKeyListener(this);
  t press = new TextArea(3,70)
  tpress.SetEditable(false);
  trel = new TextArea(3,70);
  trel.setEditable(false)
  add();
  add(tpress);
  add(trel);
  }
  public void KeyTyped(KeyEvente)
  {
  disppress(e"KeyTyped");
  }
  public void keypressed(KeyEvent e)
  {
  disppress(e," key pressed");
  }
```

```
public void KeyReleased (KeyEvent e)
{
disppress (e, "key pressed:");
}
public void
KeyReleased (KeyEvent e)
{
String
char string, key codeString, modString;
char c= e.getkeychar();
int keyCode = e.getkey code();
int modifiers = e.get Modifiers();
char String = "key character ="'+c";
keycodeString = "key code + "("
(+KeyEvent.getkeyText (keyCode)+");
mod String = "modifiers = "+ modifiers;
trel.setText ("key released"+
charString + keycode String + mod string);
}
protected void
disppress (KeyEvent e, String s)
{
String
charString, keycode String, mod string, tmpStr
String;
Char c = e get key char ()
int
keycode = eg et Code ();
```

```
int modifiers = e.getModifiers();
if (Character.IsIsoControl(c))
    {
char string = "key character (an unprintable
      control character)";
    }
else
{
charString = "key character = '" + c + "'";}
mod string = "modifiers = " + modifiers.
tmpString = KeyEvent.getKeyModifiers
            Text (modifiers);
 if (tmpString.length()>0)
    {
     modString += " (" + tmpstring + ");
    }
else
 { modstring += "(no modifiers)";
 }
keyCode String = " key code = " + keycode + "
   " + keyEvent.getKeyText getKeyText
        (keycode) + "");
tpress.SetText (St charString + keyCod
          String + mod String);
}}
```

Mouse Events

Mouse Events are represented by the MouseEvent class which is packaged in javafx.scene.input

Node
Scene } → defines convenience methods for mouse events

When a mouse event is handled by a Node, mouse events are received only when that node has input focus.
When a mouse event is handled by a Scene, mouse events are received when the scene has input focus.

For example:

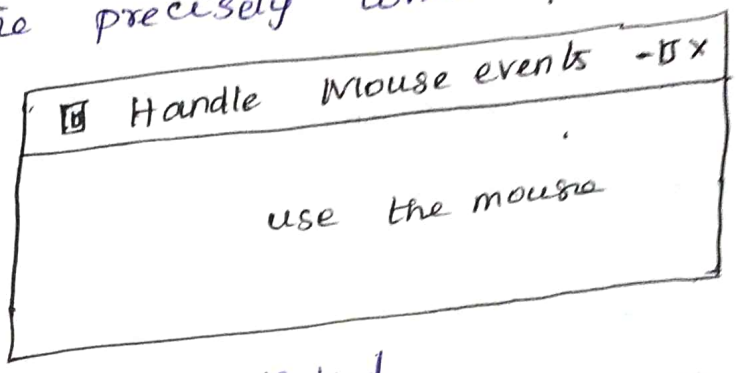
```
import javafx.application.*;
import javafx.Scene.*;
import javafx.Scene.control.*;
import javafx.Stage.*;
import javafx.Scene.Layout.*;
import javafx.event.*;
import javafx.geometry.*;
import javafx.Scene.input.*;
Public class MouseEventDemo extends Application
{
Label ShowEvent, ShowLoc;
@override
public void start (Stage primaryStage) throws
                                Exception {
Primarystage.SetTitle ("Handle Mouse Events");
Flowpane rNod = new Flowpane (orientation.
                    VERTICAL, 0, 10);
r Node.SetAligment (POS.CENTER);
Scene msce = setsune (msce);
```

```
Stack
Primary Stage . SetScene (msc);
ShowEvent = new Label ("use the Mouse");
ShowLoc = new Label (" ")
msce. Setonmouse clicked (new EventHandler<MouseEvent>()
{
int cont = ae.getclick Count();
  int cnt = ae.getclick count();
   String t = "time";
   if (cnt > 1) t += "s";
   Switch (ae getButton())
  {
  case PRIMARY:
ShwEvent SetText ("Primary button clicked" + cnt+
                                  " "+t);
   break;
   case MIDDLE:
ShwEvent.SetText ("MIDDLE button clicked "+cnt+
                                  " "+t);
   break;
   case SECONDARY;
   shwEvent.SetText (" secondary button clicked
                  " cnt+" "+t);
   break;
  } }
  });
msce Seton MouseMoved ( new EventHandler<MouseEvent>()
  {;
public void handle (MouseEvent ae)
{
ShowLoc.SetText ("Mouse at " + ae.getsceneX()
      +", "+ae.getsceneY());
```

```
    }
});
r.Node.getChildren().addAll(showEvent, showLoc);
primaryStage.show();
    }}
```

As a result, a number EventType objects are defined by MouseEvent that represent the events such as Mouse-clicked and Mouse_moved.

MouseEvent defines a number of methods that help you determine precisely what has occurred

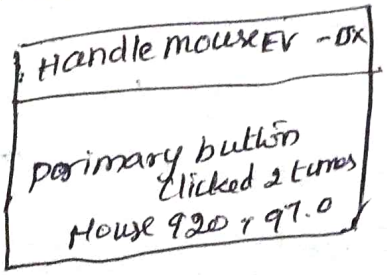


when the mouse is clicked, you can find out which button was used by calling getButton();

final MouseButton getButton().

In general if the left button was clicked, MouseButton.PRIMARY is returned.

If the right button was clicked the return value is MouseButton.SECONDARY

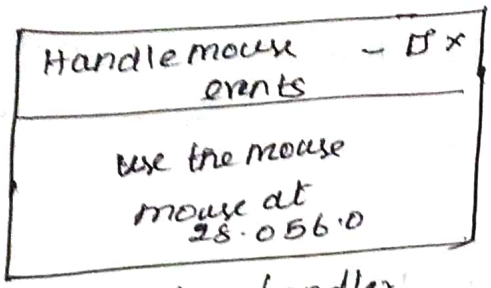


getclick count() shown here

final int getclickcount().

final double getSceneX()

final double getSceneY()



O/p of Key handler.

CONTROLS.

JavaFX defines a rich set of controls that are packed in javafx.scene.control.

CheckBox :

checkbox encapsulates the functionality of a checkBox.

immediate superclass of checkBox is ButtonBase

checkBox represents a special type of button

CheckBox supports three states

1) First two are checked or unchecked

2) Third state is indeterminate.

CheckBox:

two constructors

1. First is the default constructor.

2. Second lets you specify a string that identifies the box as shown below

CheckBox (String str)

It creates a checkBox that has ~~button~~ text specified by str as a Label.

Layouts - Flowpane - HBox and VBox - Borderpane - Grid pane.

In JavaFX, Layout defines the way in which the components are to be seen on the stage. It basically organizes the scene graph nodes.

Layouts

Layout panes are containers which are used for flexible and dynamic arrangements of UI controls with in a scene graph of a JavaFX application

package used. JavaFX.scene.Layout package

1. pane
2. VBox
3. HBox
4. Borderpane
5. Flowpane
6. Grid pane
7. Stackpane

Pane:

Basic class for Layout panes

It contains the getChildren() method for returning a list of nodes in the pane

VBox

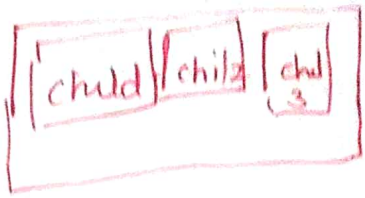
places the nodes in sigle column.

VBox

places the nodes in a single column

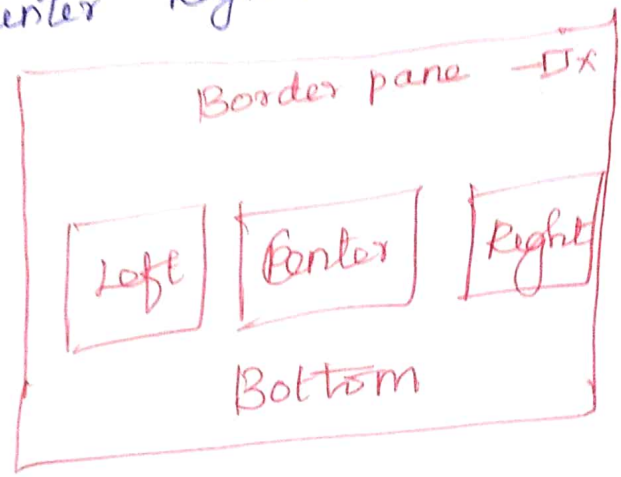
HBox

places the nodes in single row



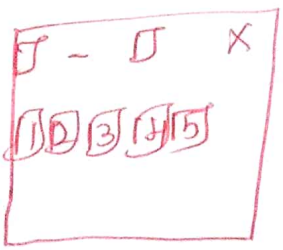
Borderpane

places the nodes in the top, right button left and center Region

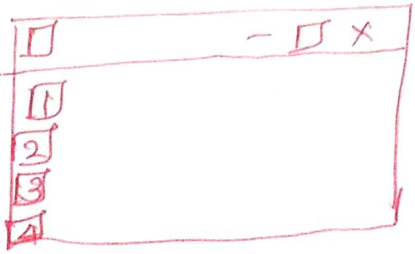


Flow pane

places the nodes row by row horizontally or column by column vertically.



Horizontal

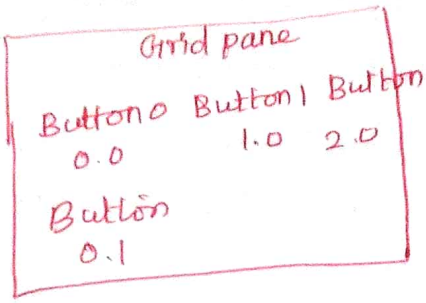


Vertical

Grid pane

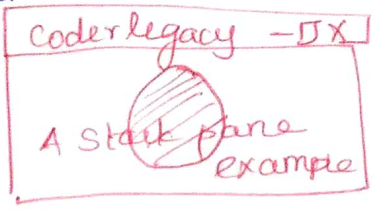
places the nodes in the cells in a two dimensional grid.

S



Stack pane

places the nodes on the top of each other in the center of the pane



Layouts — Constructors.

VBOX

1) VBox()
2) V box (Double spacing
3. Vbox (Double spacing Node? children)
4 Vbox (Node? children)

HBox

1. new HBOX()
2. new HBOX [Double spacing)

Borderpane

1) Borderpane()
2. Border pane (Node center)
3. Border pane (Node Center / Node top, Noderight, left.)

Flowpane

1. Flowpane()
2. Flowpane(Double Hgap, Double Vgap)
3. Flowpane (Node ... children.

Grid pane : public Grid pane()

Stack pane

1. Stackpane()
2. Stack pane (Node ?. children)

Menus - Basics - menu - menubars - menuitem

5.6.1

JAVAFX Menus - menu item and menu Bar

Definition

Menu is a popup menu that contains several menu items that are displayed when the user clicks a menu.

menubar is usually placed at the top of the screen which contains several menus.

Constructor of the menubar class

menubar () → creates the new empty menubar

MenuBar (Menu... m)

Constructor of the menu class are

1. menu()
2. menu (String s)
3. Menu (String s , Node n)

~~Controls~~ :

~~CheckBox~~ ~~Up~~.

4. Menu (String s, Node n, MenuItem ... i)

Commonly used methods

getItems()

hide()

Show()

getMenus()

isUseSystemMenuBar()

SetOnHidden (EventHandler v)

JavaFX Menu

In the Javafx application in order to creat a menu, menu items, and menubar. menu, menuitem, menubar class is used

Example.