MODULE 2

29/09/22

- Programming models
  - Process oriented - Focus on procedure / algo. rather than data
    - follows top down approach
    - Program → fn → then this fns are performed in sequential manner.
    - concept of class is absent
    - comm° through functions.
    - Used where data security is not major concern

(Program : fn + data)

  - Object orient programming
    - Focus is on data security
    - Fn & data are tied together in a class
    - Program is divided into objects

      Ex: HR & Account dept of one company

    - comm° is through objects

Features of OOP

- Objects - Instance of class which contains states (O = I) and behaviour (common properties)
- Class - A blueprint from which instance of class is created
  - Group of object with same behaviour & state.
  - Ex: Class - student
    object - Harshal, Ankit, Bhushan, ...
    Class - Employee
    object - Emp1, Emp2 ----

1) Abstraction

- Hiding complexity & showing only necessary data.
  Ex: website, OS
  (Data hiding is diff - privacy concerns are not adressed here)

2) Encapsulation

- Binding of data & function together

3) Inheritance

- Acquiring properties of another object

5) Polymorphism

- One interface multiple methods / one interface to be used for a general class of actions.

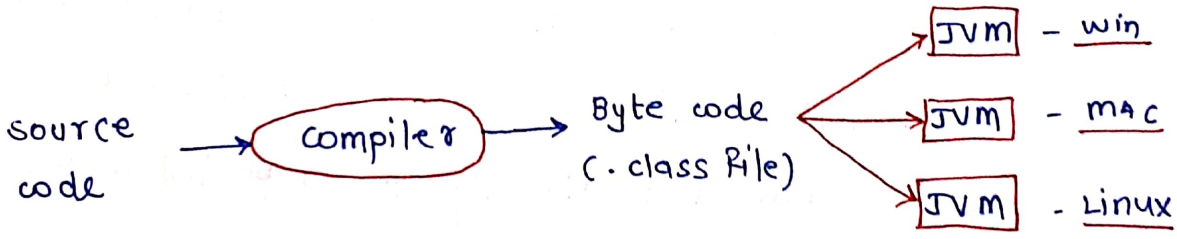
  Eg: water , UPI

- Java is cross platform OOP language

- Java code ⇒ Intermediate code (bytecode) ⇒ Java interpreter ⇒ machine code

javac Program.java → Program.class → java Program

- WORA - Write once Run anywhere



Features

- Portable
- Object oriented
  - Garbage coll^n
- Robust
  - Early detn of error
  - Exceptn handling
- Secure - Program is confined to java execut^n environment
- multithreading
- Distributed
- Dynamic
- High performance
  - Bytecode
  - JIT

JIT compiler

30/09/22

- It neutralises the disadvantage of interpreter.
- JIT compiles repeatative code into native code
  (compiles entire bytecode into native code)
- Native code used directly for repeated method.

JDK

JDK = JRE + Dev tool
JRE = JVM + Library classes

JVM = Interpreter + JIT

Edn

- Java SE (Standard / Software Edn)
  - ↳ Used for desktop env. apps
- Java EE (Enterprise Edn)
  - ↳ for larges appln, web services & web based app.
- Java ME (micro Edn)
  - ↳ for embeded

- Hello world java program
- comments - // single line
  - /* */ - multiline
  - /** documentation **/

01/10/22

Data types

- All data types are well defined
- Java is strongly typed language
- Total - 8 data types (primitive)
  - Integars - byte, short, int, long
  - Floating point no. - float, double
  - characters - char
  - Boolean - boolean

. why they are called primitive ?

↳ because they have already assigned data values

- Non-primitive - Reference data types

Scope of variable

- It defines the life & accessability of a variable.

  Locality.

- variable is restricted to particular fn

- (Local variable) must be initialised
  ↳ variables declared within fn

- Literals - values which are assigned to variable

```
int a = 5;
 ↑   ↑   ↑
Data  variable  Literal
type
```

- Printing o/p will be decimal always if it can be stored in any value (decimal, octal, hexal)

Decimal → 47

8 | 47
  | 5 7

Octal → 057

16 | 47
   | 2 15(F)

Hexa → 0x2F

- Default datatype of float is double (ie. 0.0)
  literal

Type conversion & casting

- Assigning value of one data type to another data type

- If 2 types are compatible → Automatic type conv^{n} by java (implicit)

- If 2 types are incompatible → Typecasting (Explicit)

- Widening - Small data type variable → Big data variable

- Narrowing - Big data variable → Small data variable

Binary of 260

~~100000100~~ So 1

03/10/22

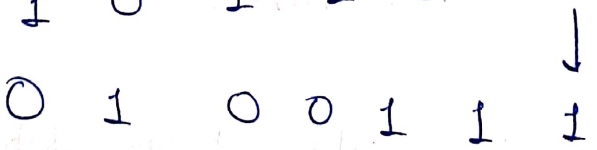
Ex :

89 ~~35~~ - Binary - 1 0 1 1 0 0 1

- finding 2's (two's complement

- find first 1 from right end then flip

then



Type casting exceptions

- Byte to char
- short to char

1's complement = find out binary number & flip each bit

2's complement = start from right side & keep all bits same till first 1 & after that flip each bit

- Leftmost bit is MSB (most significant bit)
  - It represents 1 = -ve
    0 = +ve
  - It is sign bit

- Ex : -35 - Binary → 10000111

into byte →

?

Type promotion

1. If any arthmatic operation is taking place betn two diff data types then in that case the lower data type is promoted to higher data type. And result will also be in higher data type.

2. Java promotes each byte & short data to int while doing arithmatic evaluation (Byte, short)

b * s → int * int ; s * s → int * int

b * b → int * int

Soln. - change data type of resultant to a int data type

Identifiers

- A name in a program that might be assigned to variable, class, method, object.

Rules for Identifier

- Name cannot be started with number
- It must start with letter, currency sign, (_)
- No limit to length of identifiers
- It is case sensitive

Upper camel case → For class, interface name

Lower camel caye → For methods, objects name

Ex: Uppercc → class StudentName ✓
class AreaOfShape ✓

lowercc → void calculateArea ← method name

Operators

increament & decrement operator

```
x ++        // x = x + 1
```

- Unary operator - ++, --
  - As the number of operand is one
  - operator ++, --
  - operand x

Variant operator

a = a + b can be written as a += b ;

a = a * b ——— a *= b ;

Bitwise operator

① ~ (NOT)

```
int a = 10
int b = 18

int res ;
res = ~a ;
```

It will flip the bit

// binary of a = 0000 1010

11110101

↳ -11

② & (AND)

```
a = 0000 1010
b = 0001 0010
-------------
& 0000 0010   (Decimal value - 2)
```

[1 & 1 = 1 , 0 & 1 = 0 , 1 & 0 = 0,

0 & 0 = 0]

③ | (OR)

		1	&
0	0	0	0
1	0	1	0
0	1	1	0
1	1	1	1

(Practise this operators in lab section)

Leftshift operator (<<)

- shifting 1 bit to left

5 = 0 0 0 0 0 1 0 1

0 0 0 0 1 0 1 0 ← Here zero replaces last shifted bit

Ex:
```
char a = 'A';
char b = (char)(ch << 1);
s.o.pln("b = " + b);
```

- Left shift giving multiplication by 2.

Rightshift (>>)

Ex:
```
int a = 6;
int b = a >> 1;
sop(b)
```

6 = 0 0 0 0 0 1 1 0 ← vanished

0 0 0 0 0 0 1 1

Black space filled by sign bit

(As this number was +ve - so 0)

- Each right shift result in division by 2.

- vacant bit is 0

Unsigned Right shift operator >>>

- for each unassigned right operator vacant bit is assigned as '0'

Relational operators

- == , != , > , < , >= , <=

- output of this always going to be in boolean values

- != - not equal to (Ex : 5 != 6
  - yes (As it is not equal to)

(refer SS - 04/10/22 - @15.31)

Boolean logical operators

~ (not), & (and), | (or), ^ (xor)

The above operator are used for bitwise operatn for the binary numbers & are also used for boolean operations.

~ - Gives opposite value i.e. for true it will give false & for false it will give true

& - If anyone of operand is false answer will be false

| - If anyone of two operands is true then it will return true

^ - Say if both the operands are same then it will return 0 otherwise 1.

- && (short circuit AND operator / logical AND operator)
  - (only for boolean values)
- & (Bitwise / Boolean AND operator)

Ex:

(exp1) & (exp2) ; // In this cases both the expressions will be evaluated

(exp1) && (exp2) ; // In this it is not mandatory that both the expr^n will be evaluated
- Time is saved
- If 1st exp is false wont check further
- If 1st exp true then it will check further

(practise its program)

Ex:

```
int x = 5;
int y = 10;
if (x>y & y++ < 15)        /    &&
    {
        s.o.p. ("Hi")
    }
else {
        s.o.p ("Bye")
    }
```

|| - short circuit logical OR

- Used where o/p can be derived from one expressions evaluath

| - Both expressions evaluated

Ex:
```
{ int x = 5 ;
  int y = 15 ;
  if ( x < y || y++ > 15)
    {
      Sop (Hi)
    }
  else {
      sop ( Bye)
    }
  sop (x)
  Sop (y)
}
```

Ternary operator

-
```
int x = 10 ;
int y = 10 ;
int bigger = x > y ? x : y ;
```

conditional statement ? if true return this : return this for false

- Parentheses
  - It raises the precedence of the operations and sometimes clarify the meaning of expr.

Java control statements

ss @ 19:15 / 4 Oct 22

- If else ✓
- If elseIf

```
if (condn true)
{
    then control will come here
}
else if (condn true)
{
    then control will come here
}
else if (condn true)
{
    ——————
}
multiple lines can be written here
else {
    then here (if all above are false)
}
```

Switch statement

Ex:

```
int choice = 2;

Switch (choice)
{
    case 1 :
        S.O.p.(1)
        break;
    case 2 :
        Sop(2)
    }
default:
    S.O.pln(wrong choice)
}
```

[Breaks helps in exiting loop]

06/10/22

Loops

- Initialisatn, condn, increament/decreament

1) Do
2) Do while
3) For

Printing 1 to 10

1) While loop

```
int i = 1;
while (i<=10) ← condn
{
    S.O.p(i);
    i++; {w/o this it will be infinite loop)
}
```

2) Do while loop

- In while & for loop, first condn is checked and then control comes inside the loop.
- In do while condn is check lastly. So it atleast execute once

```
do {
    // code
}
while (condition);
```

```
do {
    S.O.p(i)
    i++
} while (i<=10);
```

3) For loop

- for (initialisatn ; condn ; increment / decrement)
- Variables defined in method are local variables

- w/o condn loop will go into infinite loop

Break & continue

Break - takes control out of loop

Continue - it will skip to execute further inside loop

Ex:

```
int i = 1;
while (i<=10)
{
  s.o.p (i)
  if (i==5)
  {
    break;   // continue
  }
  i++
}
```

→ o/p

1
2
3
4
5
(out of loop)

[In so output → 1 2 3 4 _ 6 7 8 9 10)

- Loop can be labelled with any name

Ex: outerLoop : for (int ....

- Labelled break statement

```
{
  break __ ; ← label name
}
```

ARRAY

- Arrays in java - storing similar type data into one variable

- Array is a collection of similar type of data

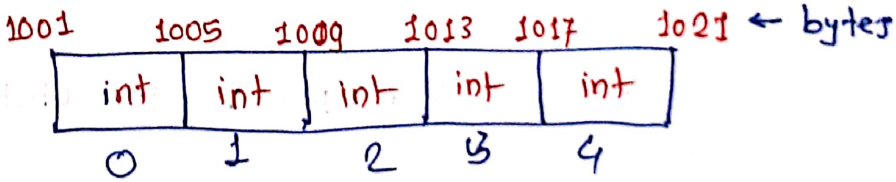
Ex: Storing student marks 40, 65, 50, 75, 80

↳ new operator is used to allocate memory for array

↳ new int [5] // it will allocate memory to store 5 int values

(Syntax: new DATATYPE [size]

↳ memory allocatn is like,



↳ It will allocate memory to store 5 int and will return the reference (base adress) of allocated memory.

- Storing memory locations / return reference into some variable

Ex: `int [] marks ;` mark is a variable which is used to keep the reference of any integar array

`new int [] ;`

It can be written as,

`marks = new int [5] ;`

marks → Base adress of allocated memory

new int [5] → creath of Array

```
marks[0] = 10 ;
marks[1] = 20 ;
marks[2] = 30 ;
marks[3] = 40 ;
marks[4] = 50 ;

s.o.pln (mark[])
```

or

```
int mark[] = { 1 };
```

values

2D Array

- Accessing 2D arrays

Ex: arr [1] [2]
Row, column

	0	1	2
0	0 0	[0][1]	0 2
1	1 0	[1][1]	1 2
2	2 0	[2][1]	2 2

arr [2][0] = 5 ; //example

For 1D,

```
int marks [] = new int [5];
```

marking variable for array

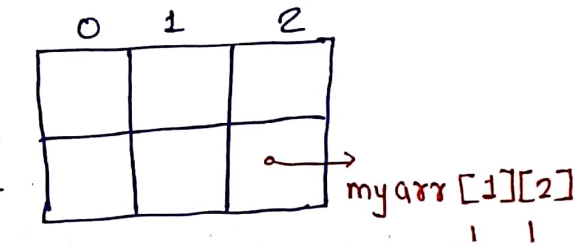
memory allocation

For 2D,

new int [2] [3]; // creation of array with 2 Rows & 3 columns

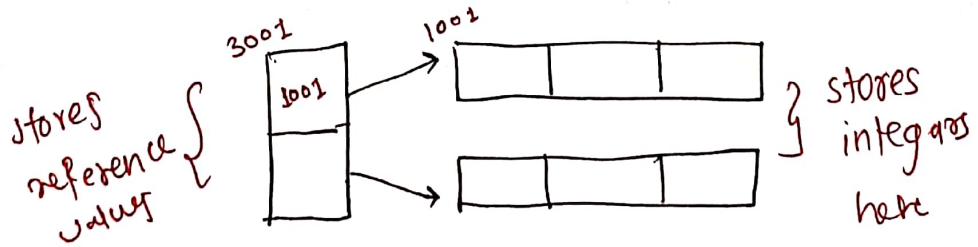
syntax:

```
int my arr[][] = new int [2][3]
```



my arr [1][2] = 5;

S.o.pln ( ); //o/p = 5



- Refer program - Array2D.java (MOD2)

Length

```
myarr.length ;          // return - 2

myarr[0].length         // return 3

myarr[1]                // return 3

myarr[1][0].length      // Error.
```

Accessing 2D array

↳ myarr[row index][column index]

Jagged Array

↳ Each row will have diff number of elements or different number of columns

↳
```
int jarr[][] = new int[2][]

jarr[0] = new int[2];
jarr[1] = new int[4];
```

↑ keep this empty

Refer SS @ 12:00 7th oct

Direct initialised in 2D array

~~int myarr [][] = new~~

```
int myarr [][] = {
                    { 5, 10, 15 },
                    { 20, 25, 30 }
                 };
```

(Refer: SS @12:52 7/10)

Class & Object

object → Entity with having state and behaviour
- Instance of class
- It is real world entity

class → Group of objects which have common properties.
- class is not a real world entity.

Class = Data + method

↳ creating & defining class

↳

```
class student
{ data
  + method
}
```

object

new student() // new obj created

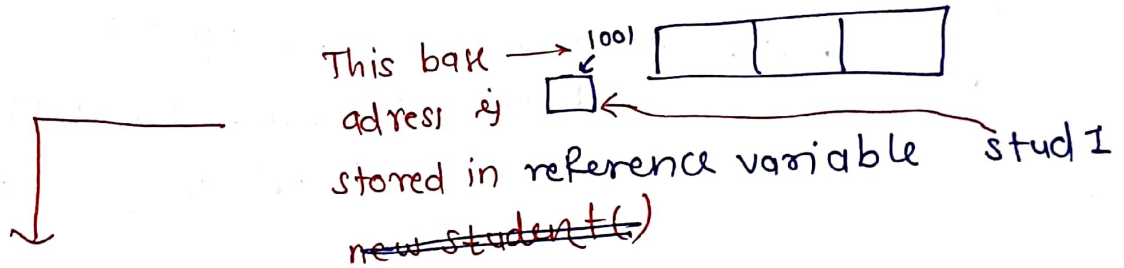
new → keyword

↳ memory is allocated for data members of class

student stud1// → reference variable

↳ As student has ← roll no, Age, marks

↳ It will ~~generate~~ allocate memory



student stud1;

stud1 = new student();

// object is created & it is stored in ref. variable stud1

storing reference here

storing object

(Student stud1)

1) Stud1 is reference variable, it stores reference value of → only the reference of student class object

- Default value of any reference variable is NULL.

Ex: 2 ~~Employee emp1~~

```
Class Employee
{
    int empId ;
    long contact ;
    double salary ;
}
Employee emp1 ;
emp1 = new Employee()
```

```
void printdetails()
{
    s.o.p(id + contact + salary)
}
```

(. dot → member access operator
└ Job: To access the data & method in objects)

obj 1

```
emp1 . empId = 5 ;
emp1 . contact = 8552947795 ;
emp1 . salary = ∞ ;
```

obj 2

```
Employee emp2 ;
    emp1 . empId =
Employee emp3 )
```

emp1.printdetails(); // object calling method

emp2.printdetails();

08/10/22

- object is known by there reference name
- A java file can have multiple classes

- Refer Code in files /MOD2/ student.java + Employee.java

object using method calling

```
class student
 {
    int roll;
    int age;
    void setDetails(int r, int a)
     {  roll = r;
        age = a;
     }
    void printDetails()
     {
        s.o.p(roll + " " + age);
     }

Student stud1 = new Student();
 stud1.setDetails(1,20);
Student stud2 = new Student;
 stud2.setDetails(1,30);
} stdi.printdetails()
```

local variables

values will be saved in that object

method calling

// obj creath

// setting / initi. value in obj.

Constructor

→ It is used to initialise an object immediatly upon creation.

→ constructor must have the same name as class name

→ constructor do not return any type

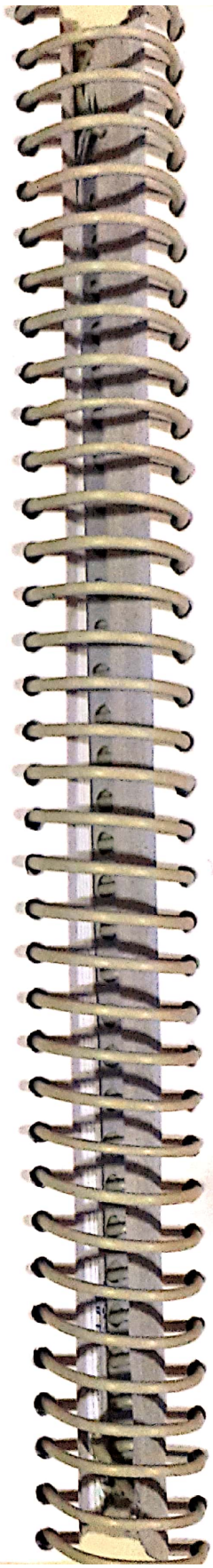
→ constructors are called only once at the time of object creatn (automatically)

→ It can be used to set initial values for object attributes

→ multiple constructors can be called in a class

Ex:

```
// zero-arg constructor
Student()
{
}
// 1-arg constructor
student(int a)
{
}
```



Ex:

class
{

cla
{

Ex:

```
class student
{
    int roll;
    int age;
    studen()
    { s.op("zero Arg constr"); }
    void printdetails()
    { s.op. (roll + " " + age);
    }
}
class student demo 1
{
    p.s.v.m ( )
    {
        student stud 1 = new student();
        stud1 . printdetails ();
    }
}
```

zero argument constructor / default

Here it is zero arg constructor

```
new student ( ) // zeroarg constr
new student (1) // 1 arg constr
new student (1,2) // 2 arg constr
```

this keyword

- 'this' is a reference to the current object.
- current obj → object which has made the funcn/method call recently.

Ex: obj1.printdetails();
obj2.printdetails(); ← this is current object here

Instance variable

- It is the property of class / it becomes a part of the an object
- They are inside a class but outside the method

Ex:
```
int marks;
int roll;
double age;
```
} This are 'instance variables'

- It is part of every string object

Local variable

- variables which are declared within a method

- Conflict of Local variable vs Instance variable
  - Local variable ✓ (This will prevail)
  - Instance variable ✗
- variable shadowing - Local variable shadows instance variable

Method overloading

- Inside a class we can define multiple methods with the same name.
- This methods will differ in number of arguments or type of arguments.

– Not possible to overload with same method name & signature

Ex:

```
Class Demo
{
    void myFun(int x)
    {
    }
    void myFun(string s)
    {
    }
    int myFun(int x)
    {
    }
}
class Ov
{
    p.s.v.m
    {
        Ov o1 = new Ov();
        o1.myFun(↑)
    }
}
```

(signatures)

According to value we put here the method will be executed

H.W. Add Calculator

5,15 - int int
5, 15.5 - int, d
5.5, 15 - d, int
5.7, 10.5 - d, d

class AddCalculator
{ void add();
};

Static members

- class = Data + method
- Data & method
  - Static - Belong to entire class
  - Non-Static - Belong to objects

→ Static - AKA class data/variable & class methods

→ Non-static - AKA instance data/variable & instance methods

Ex:

```
Class Demo
{
    int a ;      // non-static , instance var
    int b ;      // non-static , i—
    static int c ;  // class variable

void myFun ()   // non-static, instance method
{
}
static void anotherFun() // static method
{
}
```

object will only save this data

- To make any call to non-static method of a class, we must need an object of that class to call that method using that object.

- If a method is static we dont need any object to call that static method.
  ↳ To call any static method, we can use
  classname.staticmethodname.

refer ss @ 1745 hrs

- Memory is allocated to static at the time of class loading

Scanner

Scanner sc = new Scanner (System.in);

- Scanner ↑ class
- System.in ↑ it represents standard input (keyboard)

Scanner class methods

```
nextInt();
nextLong();
}
```

next() // to read single string/word

nextLine() // ~~to read byte value from the keyboard~~

↳ To read entire line i.e. string with spaces

Ex:

```
class Demo
{
    p.s.v.m
    {
        Scanner sc = new Scanner(Sy.in);
        int a = sc.nextInt();
    }
}
```

09/10/22

Q. can we access instance variable within static method ?

→ ~~Yes~~ No

- Because there is no call through object and static variable are accessed with class name

Q. can we access static variable in non-static

→ ~~no~~ yes

- Inside static method only static variables can be accessed
  - compilation error may occur
- From static method only another static method can be called

2. static data & method can be accessed from any non-static method

---

Array of objects

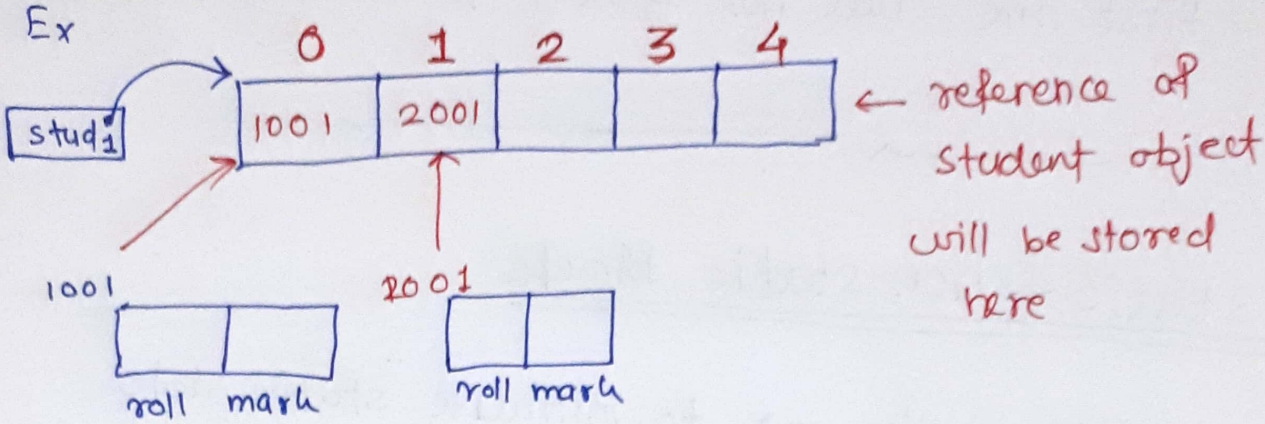
For int, `int myarr[] = new int[5]` //array of primitive type

`Student stud[] = new Student[5];`

- stud: ref var
- new Student[5]: Array creation

- It will ~~new~~ create an array of student reference which will be used to store 5 student object reference.

Ex



← reference of student object will be stored here

- Refer code (comments are in codes)
  ↳ StudentArrayObj.java

- Nullpointexcepth → when null reference calling method.

  Ex: stud1[2].printDetails().
  ↑ consider here this ref. has not have any values stored

Constructor calling using 'this'

- We can call overloaded constructed from another constructor of same class using 'this'.

- only possible with same class name

- constructor calling using this shall be in first line of constructor

- first line rule not applies to calling method

Static & non-static Block

- static block → to initialise static data
- Non-static block → to initialise non-static data/ instance variable
  (AKA init / initialiser)

```
class
{
    int a ;
    int b ;
    int c ;
    // static block
       static ABC
       {
       }
    // non-static
       ABC
       {
       }
```

→ non-static block are alway executed before the constructor

→ non-static block will only be executed when any object is created

→ static block is loaded first in main memory

→ when class defin? is loaded in memory static block will be executed

- order of execution

1. Static block is executed (only once) & then
2. Non-static block then constructor is executed

- If there are multiple static/non-static blocks inside a class then block will be executed in block sequence

---

Call by value & Call by reference

- In method we pass the arguments by value or reference

10/10/2022

Packages

- Package is a collection of related classes
- To create any package,

  package <package_name>

  examples
  - java.io;
    java.sql;
    java.util;
    java.lang; ← default package
- Package need to be imported to use any particular package
- Creating our own package
  1) Creating one package - mypack
  2) creating one class first inside
  3) creating another class second
  4) Using it in class

- Package mypack; // whichever classes we define in this folder will be a part of mypack package

↳ mkdir mypack

↳ mypack → First.java // 1st statement - package mypack;
second.java //

↳ come out of mypack folder

↳ JAR file (Java Archives) - Package File Format

↳ jar cf mypack.jar mypack/ or complete directory/ path

javac -cp <path to jar file>;

Inheritance

1. Parent class → Base class / ~~who~~ super class
2. child class → Derived class / subclass

Ex:
```
class Cricketer
{   string name;
    string country;
    int totNoofmatches;
};
class Batsman (extends) Cricketer   // Batsman is child class of cricketer
{   int totalruns;
};
```

Class Batsman is child class, it will have all properties from its parent class & have its own properties as well

- when we create object of child class, then ~~parent class will~~ it will occupy space to store inherited values & own properties

Syntax :

```
class childclassname extends Parent class
{
```

- Refer → Cricketer1.java

- Constructor is never inherited (refer sir's note)

method overriding

- method of parent class be redifed in the child class with the same & signature

- In this case of method overriding method name, argument type and number of arguments everything will be same in the child class.

```
Ex:.  class First
      {
          void myFun (int x, int y)        // overridden method
          {     s.o.pln ("myfun first");
          }
          void anotherfun (int x)
            {   s.o.p ("another fun");
            }
      }
class  second extends First
       void myfun (int a, int b)   // overriding method
       {  s.o.pln("myfun of second");
       }

Public class over {
        p.s.v.m ( ) {
            second s = new second ();
              s.myFun (); (2,5)
```

Overloading vs Overriding

- Overloading happens inside the same class but overriding occurs inside the child class.
- In both the cases method name will remain same but in the case of overloading signature will differ (argument will differ)
- In case of overriding everything remains same (type as well as number of arguments)
- In method overriding return type can be changed in the child class but that return type must be subtype of return type in P. class (read more)

'super' keyword

- super keyword can be used to access superclass or parent class. from child class.

Ex:

```
class first
    {
    }
class second extends first
    {
    }
class Demo
    {
    first myFun()
        { first f = new first();
          return f;
        }
class Demo1 extends Demo
    {   second myFun()  // As second is child of first / we can say subtype of first
        { // code
        }
    }
```

- In method overriding return type must be covariant

Constructor calling using "Super"

10/10/2022

Packages

- Package is a collection of related classes
- To create any package,

  package <package_name>

  examples
  - java.io;
  - java.sql;
  - java.util;
  - java.lang; ← default package
- Package need to be imported to use any particular package
- Creating our own package
  1) Creating one package - mypack
  2) creating one class first inside
  3) creating another class second
  4) Using it in class

- Package mypack; // whichever classes we define in this folder will be a part of mypack package

- mkdir mypack
- mypack → First.java // 1st statement - package mypack;
  Second.java //
- come out of mypack folder

- JAR file (Java Archive) - Package file format
- jar cf mypack.jar mypack/ or complete directory path

javac -cp <path to jar file>;

[-d

Inheritance

1. parent class → Base class / ~~who~~ super class
2. child class → Derived class / subclass

Ex:
```
class Cricketer
{   string name;
    string country;
    int totNoofmatches;
}
class Batsman^Cricketer (extends)   // Batsman is childclass of cricketer
{   int totalruns;
}
```

[class Batsman is child class, it will have all properties from its parent class & have its own properties as well]

- when we create object of childclass, then ~~parent class will~~ it will occupy space to store inherited values & own properties

Syntax :

```
class childclassname extends Parent class
{
```

- Refer → Cricketer1.java

- Constructor is never inherited (refer sir's note)

method overriding

- method of parent class be redifed in the child class with the same & signature

- In this case of method overriding method name, argument type and number of arguments everything will be same in the child class.

```
Ex:  class First
     {
        void myFun (int x, int y)        // overridden method
        {
            s.o.pln ("myFun first");
        }
        void anotherFun (int x)
        {
            s.o.p ("anotherFun");
        }
     }
class second extends First
     {
        void myFun (int a, int b)        // overriding method
        {
            s.o.pln ("myFun of second");
        }

Public class over {
        p.s.v.m () {
            second s = new second ();
            s.myFun (); (2,5)
        }
```

Overloading vs Overriding

- Overloading happens inside the same class but overriding occurs inside the child class.
- In both the cases method name will remain same but in the case of overloading signature will differ (argument will differ)
- In case of overriding everything remains same (type as well as number of arguments)
- In method overriding return type can be changed in the child class but that return type must be subtype of return type in P. class (read more)

'super' keyword

- super keyword can be used to access superclass or parent class. from child class.

Ex:

```
class first
    {
    }
class second extends first
    {
    }
class Demo
    {
    first myFun()
        { first f = new first();
          return f;
        }
class DemoI extends Demo
    {   second myFun()  // As second is child of first / we can say subtype of first
        { // code
        }
    }
```

- In method overriding return type must be covariant

Constructor calling using "Super"

Types of Inheritance

1. Single inheritance
2. Multilevel inheritance (not allowed in java)
3. Heirarchial inheritance

Final keyword

'final' is a keyword & it can be used with class, data member and member function

- Final variable → once data initialised, no change can be made
- Final method → Final method can't be overriden by child class
- Final class → A final class can not be inherited

Abstract class & method

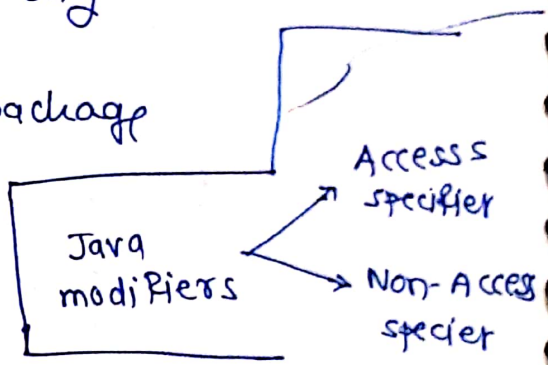
Abstract method - method without body or without defination

Ex
```
class Demo {
    abstract void myFun(); // abstract method
}
```

Access specifiers

11/10/22

- It specifies the access, from where data member/ function can be accessed. (Java has modifiers)
- It can apply to class, method, data
- 4 specifiers - Public, Private, Protected, Default
  1. Public - can be accessed from anywhere
  2. Private - Inside a class only
  3. Default - Inside same package
  4. protected -



- Refer sir's notes for sample code for access modifiers

Abstract class

- objects of abstract class cant generated
- An abs method can have zero or more abs method

concrete method

- If there is any abstract method in a class then that class will be considered Abstract class.

Can we make a class/method final as well as abstract ?

→ No.

- final - It is final & can not be overridden
- Abstract - It is abstract & it must be overridden in child class
- final - class can not be inherited
- Abstract - class should be inherited, if it is to be used

↳ we can store child class object in parent class reference type

Ex: cl First

cl second

cl demo

```
First f;
f = new second();
f.fun1();
f.fun2();
```

obj = of second

ref = of first

← stored in second class

- Any class reference type can keep the reference of,
  - ↳ its own object
  - ↳ reference of its child class object

Take a note from sir's notes

↳

Binding

- Association (linking) between method call & method defination.
- Binding takes place either at compiler or run time
  - If binding happens at a compile time then it is static binding.
  - Binding at runtime is dynamic binding.

— All final, private, static — Binding @ compilatn

— In method overriding dynamic binding occurs

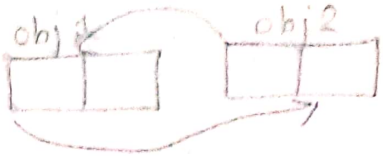
Method Hiding (static method overriding)

- static method of parent class cant be overridden by child class
  - It can be done so by redifining that static method with the same signature i.e. method hiding

12/10/2022

Island of Isolation

- object refer each other internally but their reference are not stored anywhere



- whenever garbage collecth of any day is being performed then 'finalise()' method is called for that object to perform some pretask / freeup the resources occupied by that object.

- Refer : Garbagecollection 2 java

---

changing access modifier of the overriden method in the child day

- Access modifier of the overridden method in the child class must be same as parent class or less restrictive

Public protected default private → Afold.

↳ Private method can not be overridden

parent (dfawt) = child (protected / public)

parent (procted) = child (public)

parent (Public) = child (public)

INTERFACE

↳ It tell 'what to do' not 'how to do it'

↳ It is mechanism to achiere abstraction

↳ In interface it will only have abstract method not body of that method.

↳ A class that implements interface must implement all the methods declared in interface (If not all defined in class A then class A = Abstr)

- Interface < methods - All type
  Data - All types
- No object, no constructor
-

Ex:

```
interface print {
    void print() {}   — empty body
}
class A implements print {
    public void print()   — method filled by class
    {
        s.o.pln ("Hello");
    }
}
```

st 1

st 2

Difference between Abstract class & Interface

Interface	Abstract class
1) Interface has all the methods abstract	1) Abstract can have one or more methods abstract
2) Can-not have non-static data members	2) It can have non-static data members
3) Cant contain concrete method	3) It can have some concrete methods
4) Can't have constructor	4) It can have constructor
5) Cant create obj with new keyword, it can be taken as ref. type	5)

Relationship between interfaces

- One interface can extend (inherit) one or more interface.
- interface extends interface / multiple interfaces exhibits (not possible in class)
- class extends interface

A class can extend only one class & can implement one or more than one interfaces

- Extension - 1
- Implementn - 1 + ∞

class B extends A implements myInterface 1, myInt 2 ✓

class B implements myInterface extends A ✗

class C extends A, B ✗

interface myInterface extends myInt 1, myInt 2 ✓

class B implements myIntF 1, myInt 2 extend A ✗

class B implements myInt 1 ✓

class B implements

class B extends ↑ ✓

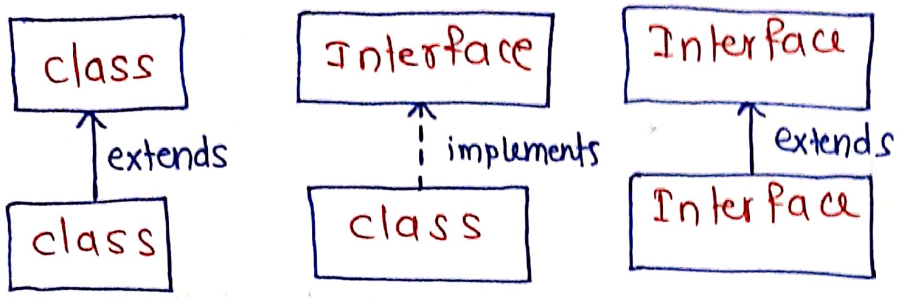
class B extends A implements myInt 1 ✓

Interface:

data - public, final, static

methods - Public abstract

interface ref variable = implementing class 'objects' ref.



Relationship betn objects

↳ AKA Association

1) Aggregation (HAS A r/s) - one object can exist without another obj.

2) Composition (PART OF) - Child class is associated with parent class

3) Is a r/s - Inheritance r/s of 2 classes

Object class methods

- There is a class named 'object' in java
- This Object class is the parent class of all the classes in java.
- Any class in java is the child of Object class.

- Object class methods
  - String toString(); 1
  - boolean toString(); 2
  - int hashCode(); 3
  - protected void finalize(); 4

- What is hashcode?
  - It is identification number for the object
  - It is not an adress.
  - It can be negative
  - If two object are equal using equal().

Refer code : HashCodeDemo.java

- objname.toString();

↳ o/p will be objectClassName @ Hashcode in HexaFormat

---

equals() method

↳ It does comparison based on references not on the data. mean it does shallow comparison

refer - code.

[Overriding object class' method]

```
Demo d1 = new Demo();
s.o.p(d1)    // It will internally call d1.
```

(refer code + Notes)

- If two object are equal using equals() method then their hashcode should also be same.

↳ So whenever we override equals() method we shall must override hashcode method

---

Wrapper Classes

- Some classes in java only work with object / we can say at some place we must have objects to perform operations.
- Here wrapper class plays a role. In collection framework & in generics, only object is required there to do any opn. These can not work with primitive type
- wrapper classes are basically used to wrap the primitive type into objects.

- Number → super class
  ↳ Class of all wrapper classes implement this class

- Double & float wrapper class

Ex:

```
Double d = new Double (53.5); // prim to double wrapper
Double d = new Double("53.5"); // string to double wrapper
```

wrapper class

Obj of Integer

```
Integer obj = new Integer (56);
Integer obj = new Integer("56");
```

wrapper class

```
int ip = Integer.parseInt("555")
```

value of method

↳ `Double d = Double.valueOf("535");`

Parse

```
double dp = Double.parseDouble("53.5");
// string to primitive double
```

obj1.compareTo(obj2)

↳ return 0 → if both are same

1 → obj1 > obj2

-1 → obj1 < obj2

Summary

Double

Double

- wrapper classes override toString() & equals() method. Here equal method compares the content not reference.

13/10/22

- constructor never get inherited like method

Integer

```
new Integer(5);                  // returns Integer obj
Integer.valueOf("5");            //
Integer.parseInt("5");           // converts string numeric
                                    to int primitive
toString();                      // returns string representn
                                    of number
intValue();                      // Returns int primitive from Int
                                    obj
```

For Byte, Short, Long, Float, Double - same

Integer and Long class have following methods

- toBinaryString();
- toHexString();
- toOctalString();

[check manual program of all]

Boxing and Unboxing

Integer iobj = new Integer(

Boxing - Encapsulating primitive data into wrapper object

Unboxing - Extracting primitive data from wrapper object

int val = iobj.intValue();

Auto Boxing & Auto Unboxing (feature of Java)

Integer iobj = 10 // w/o using method, AutoBoxing

int value = iobj // ——— Auto unboxing

Integer = int ;
int = Integer ; } Auto-boxing, both are compatible here

— Wrapper class objects can also perform arithmatic operations alike primitive data type

```
Integer iobj ;
iobj++          ✓
int a = 10      ✓
iobj = 10       ✓
int x = a + iobj ; ✓
```

Each object has a header which contains infor

For housekeeping work,

in 32 bit jvm - Header occupies 8 byte

in 64 bit jvm - Header occupies 12 byte

int takes 4 byte.

Integer takes 16 byte (for 64 bit jvm)

~~String Handling~~

JVM - memory management

method Area → class defination

Hip Area → ~~met~~ objects

~~Native method~~ stack → reference variable

Native method stack → native resources

PC register → Adress of new instrn

Shallow comparison - comparison based on object reference

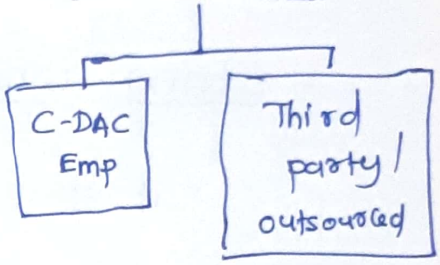
Deep comparison - comparison based on value

Assignment

Q.1. Printing CDAC employees details

↳ I/p - Parent - CDAC Employee

child - Regular emp & contract Based



- Employee - ID, Name

Regular Emp - ~~Reg~~ ~~PayScale~~ -

Basic Pay - ~~10000~~ Double type

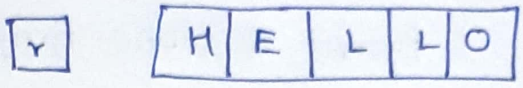
getSalary () = Basic + 70% Basic pay

(refer ss @

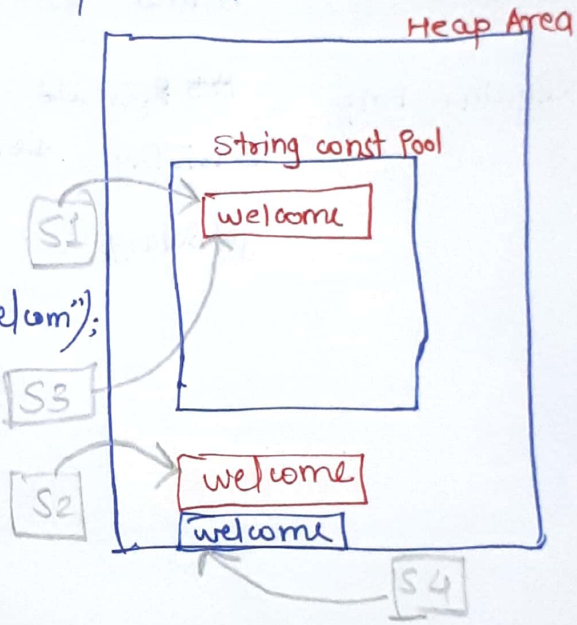
done ✓

String Handling

- String are basically sequence of characters
- String literals are kept always inside double quote. Ex: "Hello"



- String constant pool
  - It is a part inside heap area.



- string s1 = "welcome";
- string s2 = new string("welcom");
- string s3 = "welcome";

(As welcome exists in constant, so new space will be not occupied same literal will be use)

- string s4 = new string ("welcome");
- In constant pool, there will be no duplicate.

```
S1==S3;
// true

S2==S4;
// false
```

String class overrides the equals method & does the deep comparison

- s1.equals(s3); // true
- s2.equals(s4); // true
- s1.equals(s2); // true

→ Strings are immutable objects

- once string object is created. It can never be changed so it is immutable.

— System.out.println("s1.charAt(3) returns: " + s1[3]);

s.o.p/n ("s1.charAt(3) returns ;" + s1.charAt(3))

// return char 'c'

Q. Print the number of occurances of each character in a given string

→ welcome

w - 1 , e - 2 , l - 1 , c - 1, o - 1 , m - 1

- Read about for each loop ?

- equals() — // case sensitivy

- equalsIgnoreCase (case insensitive)

  ↳ It will ignore if it is upperCase/lowerCase

  ↳ Ex: str5.equalsIgnoreCase(str6);

- public char[] toCharArray()

  ↳ This converts string into array

Ex:

```
s1.toCharArray();
char charr[] = s1.toCharArray();
```

boolean startsWith(String suffix);

boolean endsWith(String suffix);

- Experiment with string methods.

14/10/22

String buffer Class

- It is used to make string mutable
- stringbuffer used to store string but it is mutable
- stringbuffer not allowes direct initialisation
- Experiment with stringbuffer methods
- string and stringbuffer difference

3) StringBuilder

- It is also to create mutable string
- Stringbuilder is not synchronised
- see diff bet stringbuffer & stringbuilder

EXCEPTION HANDLING

- mechanism to handle runtime errors, so normal flow of appln can be maintained
- To avoid termination of program from error point

2 Exception & Error

- 5 keywords to handle the exceptions

1) try 2) catch 3) Finally

4) Throw 5) throws

-
```
try
{
    // code which might show exception
}
catch (Exception Type)
{
    // Handler code
}
```

- when any excepth is generated inside try block, it ~~throw~~ makes object of excepth & throws it to catch block handler

- Each excepth in java is object in java

1) Try

- Try block can have multiple catch box
- only try block can not exist
- If try block has multiple catch blocks, only one catch block will be executed & other will be ~~to~~ skipped

-

finally block

- A try block can or can not have a finally block
- finally block will alway be executed & it does not matter excepth occured or not

→
```
try
{
}
finally
{
}
```

```
try
{
}
catch
{ //Handler
}
finally
{ //not handler
}
```

- If try block + catch + finally are present
- ~~Error~~ excepth occurs then to catch then to finally

- If no except"

- In finally block → codes which need to be executed with exception or w/o exception before end of the code

- Universal exception handler

```
catch (Exception e)
    {
    }
```

- Child class handler must come before parent class and universal handler

- Nested try-catch block

'throws'

- It is a keyword used by the methods to tell the calling function that it might throw some specific type of exceptions, so that the calling method can put the handler for those exceptions.

15/10/22

Q. Final, finally, finalize diff

→ final is a keyword which is used with class, method & varible to make it final

→ finally - exception handling

→ finalise - for garbage collection

Checked Exception → checked at compile time

Unchecked Exception → Not checked at compile time

Defining & creating our exception classes

Exception methods

1. getMessage()
2. e.printStackTrace()
   - Prints the exception details and helpful in identifying the actual reason / point of exception
   - we should use this method always whenever

Typecasting

- In case of non-primitive data type, typecasting will alway be possible betn parent & child class only.

i.e. If we do typecasting betn two class they must have r/s of inheritance

Ex: First f = new First ();
Second s = new Second ();

connected by inheritance r/s

f = s // parent ref = child ref
f = (First) f // ✓
s = f // ✓

→ Any reference type variable can the reference of its own class object / object of its child class.

Ex: First f
↳ It can have reference of its own reference or child class objects

→ Parent = child // here it is upcasting (Parent ref assigned to child)
f = (First) s

→ child = parent // here it is downcasting (child to parent)
s = (Second) s

Generics

(2) It is used to work with different type of data & object

(1) Generics is used to define such ~~dat~~ classes & methods which will be .... (2)

- Type safety is provided in generics
- Generic implementation does not

Type Erasure

↳ Compiler does the typecasting and other works and all type info is erased ie. AKA Type Erasure

Collection framework

17/10/22

- CF is used to work with group of similar types of objects i.e. objects belonging to same class
- Collectn is an object which contains reference of other objects with same type
- AKA container clases / objects
- Examples of container classes
  1) ArrayList
  2) LinkedList
  3) HashSet

- CF contains number of classes & interfaces
- work only with class types
  - prim type ✗

Steps -
1. Container class object creation
2. Then adding object to that container
3. Then accessing & using those objects stored in container

PS : Storing & accessing mechanism is diff for different container

1) Arraylist

refer Collection.java

collectn is interface, Collections is a class

List

- In list elements are added / accessed in their insertion order

Hash

[watch 7:00 pm onwards]

- Hashset internally uses Hash Map
- Insertn order is not maintained
- It stores the obj. in the keyvalue form
- key shall be some unique value

18/10/2022

- collections is a utility class, it has few methods
- static method sort() - Is used to sort array list elements
- Need to implement generic interface name

① comparable < class name > {{

then,

② override compareTo() method of comparable interface

then, sort using Collections.sort(list1);

- comparable and comparator are used to compare two objects based on some argmnts.
- comparable → we can compare objects based on one field

Comparator → To compare objects based on more than one field.

	method
Comparable	compareTo()
comparator	compare()

Hashmap

- It stores the object in the key value form
- key should be some unique value
- map declaration
  - Map <KeyType, ValueType>

OR

Ex: Map <Integer, student> map1 = new HashMap<>();

Adding obj/element to map

Student s1 = new student(1, "Harsh", 765, 21);
s2 = (2, "Rahul");

map1.add(1, s1);

map1.add(2, s2);

- map1.add(3, new Studen (3, "Geeta", 62.0)

- Iterator & for-each can not be used with map to iterate it.

- Each map object has a method entrySet() which gives set of map.Entry object for resp. map

- map.Entry is an interface
- Read about entry set method

```
for (map.Entry<Integer, Student> entry : set1)
    {
        // entry.getKey();    — returns key
    }   // entry.getValue();   — returns value
```

2) keySet() method

```
set<Integer> kset1 = map1.keySet();
for (Integer k : kset1)
{
    S.o.p(k);
}
```

entrySet() // gives set of map.Entry object for resp map

keySet() // It gives set of keys of resp. map

values() // It gives collectn of values / objects of resp map

Treemap / Treeset

↳ To store data in sorted order.

↳ In treemap on the basis of keys

- List & set difference

List	set
- Allows duplicate elements to add	- Not allows duplicate elements to add
- Insertion order is list	- Insertion order is not maintained
- Any num of null can be added	- only one null value can be added
- getmethod() ✓	- getmethod() ✗

Different utility class

- Synchronised & unsynchronised set

↳ Set <String> hs1 = new HashSet <string> ()

// Not synched so multiple threads can access

↳ Set < string > SynchSet = Collections. synchronizedSet(hs1)

// now synchset is synchronized set

↳ List <String> synchList = Coll^ns. syzed List (list I)

// Now sghchronized

→ ArrayList is not synchronised. faster than vector

→ Hashtable - Synchr.
Hashmap - Not synch.

- Non-synchronized will always be faster as it wont have overload of synching threads

- concurrent collection

  - In concurrent ~~processy~~ collections includes a set of collections apart from java collect API.
  - This are optimised and designed specifically for synchronised multithreated acces

I/O

File Class

methods - f.exists() // return is boolean

f.isDirectory() //

f.list()

- I/O is used to process the input & produce the output
- File handling in java by I/O API
- stream - It is a sequence of data like a river stream
- outputstream - To write data to destinn
- Inputstream - To read data from a source
- Bytestream & characterstream
- In java folder is also considered as file
- file obje - new file(filepath);

new file(Directory);

- FileInputStream – To read from file

- FileOutputStream – To write to the file

- FileReader, FileWriter → For unicode values

→ when data is not binary format

For char stream

int read() // reads one ch. at a time. Returns -1 at end

String readLine() // this method is available in char stream only & not in byteStream

can be used with bufferreader

- Reads whole line at a time

returns null at end

- write() - Used to write the specified string on the stream

Reading from console

System.in - keyboard

sytem.out - monitor / console

system.error -

InputStreamReader is = new InputStreamReader (Sy.in);

BufferedReader br = new BufferedReader (is);

S.o.p (Enter an number)

String strInput = br.readLine();

Serialisation

- Using it we can store object's state in permanent storage (hard disk) & can retrieve it later

- Its use is restricted to objects of class which implements ~~serialisation~~ Serializable interface