Error at line 1:
ORA - 80984: "Salary": invalid identifier

cals select ename.

11/04/19

Java

abstractions

* abstraction is process of hiding all the unnecessary implementation and showcasing only the behaviour of the object
* to achive abstraction we have to make use of the following rules.
  1) use a interface or an abstract class
  2) achive IS-A relationship with the interface or abstract class
  3) achive method over ridding
* 4) the most important part in order to achive abstraction is upcasting that is creating a objects of subclass or implementation class & storing its address in super class or super interface

① rule	rule ② (IS-A relation)	rule ④

interface person
{
void display();
(public & abstract)
}
	class student implements person
	{ rule ③
	public void display() overridden
	{
	S.O.P ("name Harinya");
	}
	}
		class person
		{
		person - -
		person s = new student();
		s.display();
		}

Note :- if we not initialize to variable null will take place.

```
prg1 :- package com;
public interface person
{
  void display();
}
```

```
prg2 :- package com;
public class student implements person
{
  public string name;
  public student (string n)
  {
    name = n;
  }
  public void display() -> overridden
  {
    S.O.P ("name is: "+name);
```

```
package com; <prg-3
public class runner
{
public s vm (string []arg)
{ s.o.p("start");
person p = new student("diago"); -> upcasting.
p. display();
s.o.p("end");
}
}
```

using upcasting method the implementation method is hide.

student class.

```
prg1:- package com;
interface account
{
public void depait (int amt);
void withdraw (int amt)
void check balance ();
}
```

```
prg2: package com; -> implements
public class ATM extends account
{
  int balance = 5000;
  public void deposite (int amt)
  {
    balance = balance + amt;
  }
  public void withdraw (int amt)
  {
    balance = balance - amt;
  }
  public void checkbalance()
  {
    s.o.p(" account balance is :"+balance);
  }
}
```

```
prg3:- package com;
public class account holder
{
  p sum (string[] args)
  {
  account a = new ATM(); -> upcasting.
  a. check balance ();
  a. deposite (2500);
  s.o.p("after Deposite");
  a. check balance ();
  a. withdraw (4000);
  s.o.p("after withdrawing");
  a. check balance();
  }
}
```

```
o/p account balance is : 5000
after deposite
account balance is : 7500
after withdrawing.
account balance is : 3580
```

Java Bean class

(declared with private keyword)

```
public class student
{
private int age;        --> if the variable or method is declared with private keyword we use getter & setter method.
public void setage(int age)
{
  this.age = age;
}
public int getage()
{
  return age;
}
```

```
class mainclass
{ psvm...
}
student s = new student();
s.setage(30);
int age = s.getage();   --> not only age we can declare any variable
sop(age);
}
}
```

age : 30

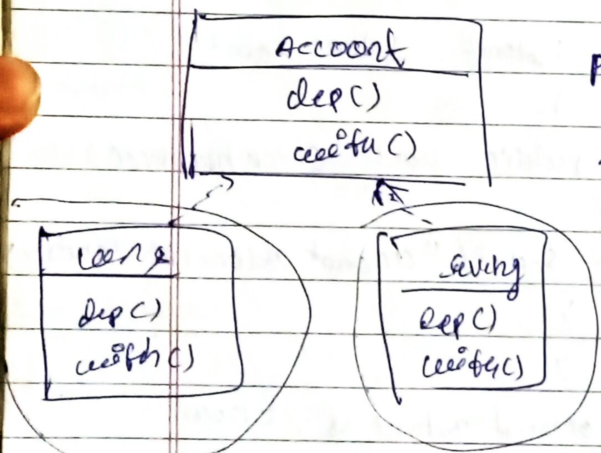
12/04/18

abstraction using factory design pattern

rule 1:- generalize the all the behaviour of implementation class and store them into a the database (abstract class)

rule 2:- creat an object of implementation classes and store it's adress in interface ref variable

rule 3:- use the interface ref variable inorder to access the methods of implements classes.



prg 1:- package com;

```
public interface account (body)
{
  void deposite(int amt);
  void withdraw(int amt);
}
```

prg 2:- package com

```
public class saving implements account
{
int balance = 5000;
public void deposite(int amt)
{
balance += amt;  // balance = balance + amt;     --> compound stmt
s.o.p("after depositing:" + balance);
}
```

```
public void withdraw(int amt)
{
    balance -= amt; // balance = balance - amt;
    s.o.p("after withdrawing: " + balance);
}
}
```

```
prg3 :- package com;
public class loan implement account
{
int balance = 10000;
public void deposite(int amt)
{
s.o.p("Before depositing: " + balance);
balance += amt;
s.o.p("after depositing: " + balance);
}
public void withdraw(int amt)
{
s.o.p("before withdrawing: " + balance);
balance -= amt;
s.o.p("after withdrawing: " + balance);
}
}
```

```
prg4 :- package com;
public class factory
{
account a; // ref = 2
public account openaccount(char type)
                    -> return type.
{
if (type == 'L')
{
a = new loans(); // role->
                  object creation
}
else                -> interface returnable
{
a = new savings(); // role->
}
return a;    a is type of account so the return type is account
}
}
```

Prg 5 :- package com ; } behind page
public class Runner
{ psvm (String[] args)

{ → return type.

Manager m = new Manager(); } to store value we use acc and data type of account
Account acc = m.openAccount ('L'); } this contains loan function if you want saving give any letters not L
acc.deposit (6000); // code - 3
S.o.p ("------ ");
acc.withdrawal (2000);
}
}

class Demo
{
void add (int a, int b) → arguments
}
s.o.p (a+b);
}
}

sop ("Start");
Demo d = new Demo ();
d.add (5, 6) → Parameterized
s.o.p (End)

o/p Start 11 End

we using sop start then so that why its printing

class Demo
{
int add (int a, int b) → arguments
{
return a+b;
} return type
}

return string value too.

sop ("Start")
Demo d = new demo ();
int sum = new demo();
Demo d = new demo ();
d.add (5, 6)
int sum = new demo ();
sop ("End") o/p = Start = 11 = End

if we don't use this the ans is not stored

String → non primitive class data type
BCZ predefined
↓ predefined

class demo
{ int age = 50;

psvm - -
{ demo d = new demo();
sop(d) // @123
sop(d.age) // 50
@123.age

age = 50 @123

object

working principle

Exp:-

com

google

gmail

inbox

draft

com.google.gmail.inbox → Fully qualified class name

package = folder & Better maintainance

class name → scanner

Fully qualified class name

java

util

Scanner.java

structure

pancard name

company name

application name

model 1

model 2

model n

class Employee → ①
{
String name;
double salary;
Employee (String name, double salary)
{
this.name = name;
this.salary = salary;
}

class Runner
{
void getHighest salary (Employee e1, Employee e2)
{
if (e1.salary > e2.salary)
{
e1
}
else
{
e2
}
psvm - - - - -
{
Employee e1 = new Employee ("Diego", 60.5);
Employee e2 = new Employee ("Gaddi", 200.75);
r.getHighest (e1, e2);
0x1, 0x2
}
}

0x01
e1

0x02
e2

① ② ③ ④

s1 = x = 0.x
s2 = y =

public class student
{
public String name;
public int marks;
student (String name, int marks);
this. name = name;
this. marks = marks;
}
}
public class Ranker
{
public student getHighestMarks (student s1, student s2, student s3)
{
if (x.marks > y.marks && x.marks > z.marks)
{ S.O.P (x.name + " has highest marks");
} return x;
else if (y.marks > x.marks && y.marks > z.marks) {
S.O.P (y.name + " has highest marks");
return y;
}
else {
S.O.P (z.name + " has highest marks");
} return z;
}

Notes: "if we use return type we use type student" — "you can also give any variable"

psvm (String [] args)
{ S.O.P ("start");
student s1 = new student ("Karan", 72);
student s2 = new student ("Raman", 62);
student s3 = new student ("Isha", 58);
Ranker r = new Ranker ();
student std = r.getHighestMarks (s1, s2, s3)
S.O.P ("name: " + std.name);
S.O.P ("end"); → SOP(std) → print the highest marks who has it

package is a collection of Java definition type which encapsulation the no. of class

Java libraries
* Java libraries
* Each and every

↓
Java.lang
String
System
object

java.lang package
java.lang package
that is user def

object class :- *ob
java.lang packa
* object class
* object class
that is predef

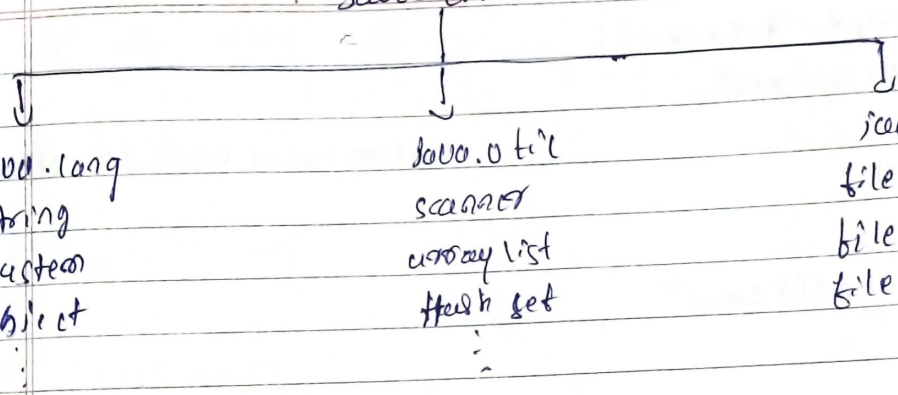
methods in obj
9) finalize metho
7) wait method
8) to String (

1) To String ( )
print the str
* internally
object addre
* once we
to latest
eg:- package
public class
{
public str
{
return
}

16/4/19

Java libraries

* Java libraries is a collection of predefined ~~libraries~~ packages
* Each and every package can contain any no classes & interfaces



java.lang package :-

java.lang package is automatically imported in all java classes that is user defined & predefined

object class :- object class is a predefined class which is present in java.lang package.

* object class is referred as super most class present in java
* object class is automatically inherited to all java class that is predefined and user defined.

methods in objects class :-> 1) equals method 2) getClass method 3) finalize method 4) # code method 5) notify method 6) notifyAll method 7) wait method 8) wait (long x) method 9) wait (long x, int y) method 10) to string ( ) method 11) clone() method

1) To string ( ) method :-> * to string ( ) method generally used to print the string representation of a specific object
* internally to string ( ) method is responsible for generating the object address in form of fully qualified classname@hexadecimal value
* once we override to string method it will always points to latest implementation.

Prg :- package org.example;

```
public class student
{
    public String toString()
    {
        return "Student@30";
    }
}
```

```
public static void main (String[] args)
{
    Student s = new Student();
    S.O.P(s);
}
```

@Override :- @override is an instruction to the programmer so, just that this specific method is inherited

* this @override has to be use before the declaration part

prg1 :- package org.example;
public class animal
{
void eat()
{
s.o.p("animal is eating");
}
}

return type of tostring is string

prg2 :- package org.example;
public class dog extends animal
{
@Override
void eat()
{
s.o.p("dog is eating");
}
void breed()
{
s.o.p("dog breed is bleu");
}
p s v m (String[] args)
{
dog d = new dog()
d.eat();
d.breed();
}
}

hash code() method :- hash code method is used to generate unique id for each and every object
return type of hash code method is int

prg1 :- package
public class
{
@override
public int
{
return
}
p s v m (
}
student
s.o.p
s.o.p
s.o.p
}
}

prg :-
package org.
public class
{ string
@ override
public
{
return
}
p s v m (
{
employee
s.o.p

prg :- pac
public cla
{
string
int
car
}
this.
this.

prg1:- package org.example;
public class student
{ @override
public int hashCode()
{
return 123;
}
psvm (String[] args)
{
student s=new student();
s.o.p (s);
s.o.p(s.toString());
s.o.p(s.hashCode());
}
}

both print address of object

capital

prg2:-
package org.example
public class ~~student~~ employee
{ string name = "jaheers";
@override
public string toString()
{
return name; (or) return "name is :"+name;
}
psvm (String[] args)
{
Employee emp = new Employee();
s.o.p(emp);

prg3:- package org.example.
public class car
{
string colour;
int price;
car (string colour, int price)
{
this.colour = colour;
this.price = price;

```
}
}                                  public String toString()
psvm (String[] args)               {
{                                    return colour + " " price;
  Car c1 = new Car("red", 25000);    }
  Car c2 = new Car("yellow", 5000);
  S.O.P(c1);
  S.O.P("------");
  S.O.P(c2);
}
}
```

Object

equals method :- * equals methods is generally used for content comparision

* equals method accepts an argument type of objects
* the content comparision logic has to written within equals() method

```
pgm1:- package org.Example;
public class Car
{
  int x=20;
  @Override
  public boolean equals(Object o)
  {
    Car c = (Car) o;
    boolean result = this.x == c.x;
    return result;
  }
  psvm (String[] args)
  {
    Car c1 = new Car();
    Car c2 = new Car();
    if (c1.equals(c2))
    {
      S.O.P("content are same");
    }
    else
    {
      S.O.P("content are not same");
    }
  }
```

Another method

```
if (c1==c2) // comparing address
{
  S.O.P("address same");
}
else
{
  S.O.P("address not same");
}
}
```

return type of equals method -> boolean

```
Obj o = new Car();
```

Car can only store type of Car

```
class Student
{
  int id;
  Student(int id)
  {
    this.id = id;
  }
  @Override
  public boolean
  {
    Student (ste) = (
    type of Student
    return this.id ==
                  10 ->
  }
```

Inner class (on

* class defined
* 4 types
1) local
2) Anonymous
3) local inner class

```
pgm1:- package
// outer class
public class Demo
{
  void test()
  {
    S.O.P("test()
    // local
class Sample1
{
  int x = 12;
  void m1()
  {
    S.O.P("
  }
}
Sample1 S1 =
S.O.P("X
S1.m1()
```

class student
{ int id;
Student (int id)
{ this.id = id;
}
@Override
public boolean equals (Object obj)
{
Student (std) = (Student) obj;
return this.id == std.id;
}

psvm
{
Student s1 = new Student[10];
s2 = ;
if (s1.equals(s2))
{
c.s
}
else
{ cws
}

classmate
Data page

o/p = cws

s1 → data
s2 → id: 20
s1 == s2 → F

17/4/19

Inner class (or) Nested class

* class defined inside another class
* 4 types
  1) local inner class 2) static inner class 3) instance (non static) class.
  4) Anonymous class

$ → innerclass
represents
it's use only inner class

1) local inner class

```
prg1:- package isolation;
// outer class
public class demo1
{
void test()
{
s.o.p("test() method started...");
// local inner class
class sample1 // no static member.
{
int x = 12;
void m1()
{
s.o.p("running m1() method...");
}
}
sample1 s1 = new sample1();
s1.x value:"+s1.x);
s1.m1();
```

access specifier is not allowed in local inner class

we can use static member in the inner class but declare as final static x = 12;

you should create object of local class in outer class only

here not allowed to static method

Prg2 :- package ispiders.p1;

class mainclass1
{
psvm(String[] args)
{
S.o.p("*****");
Demo1 D1 = new Demo1();
D1.test();
S.o.p("*****");

static inner class

prg1 :- package ispiders.s.p1;
public class demo2 -) it should be noactestDC
{
static int x = 10;
static void m1()
{
S.o.p ("running m1() method...");
}
// static inner class
1. static member are allowed
2. non ——— "
static class sample2
{
static int y = 34;
int z = 56;
static void m2()
{
S.o.p("running m2() method...");
}

void m3()
{
S.o.p ("running m3() method")...");
}
}
}

classmate
Date
Page

pg2:- package isidders.p1;
public class mainclass2
{
ps v m(string [] args)
{
s.o.p (" main method started");
s.o.p ("x value:" + demo2.x);
Demo2.m1();
s.o.p ("————————");
//ref static members of static inner class
s.o.p ("y value:"+ Demo2.sample2.y); → it shows ref the static inner class in normal class
Demo2.sample2.m2();
// ref non static members of static inner class
Demo2.sample2 ref1 = new Demo2.sample2(); -creating object of nonstatic
s.o.p ("z value:" + ref1.z);
ref1.m3();
s.o.p (" main method ended");
}
}

instance inner class

pg1:- package class Demo3
{
// instance inner class
1. static members are not allowed
2. non static members are allowed
class sample3
{
int z = 50;
void m3()
{
s.o.p ("running m3() method...");
}
}
}

pg2:- package isiders.p1;
public class mainclass3
{
public void main (string[] args)

```
{
s.o.p ("*****");
demo3 d1 = new demo3(); //created object of outer class
Demo3.Sample3 s1 = d1.new sample3(); //create object of inner
                                      class using outer class object ref
   -> (or)
//Demo3.Sample3 s2 = new Demo3().new sample3();
s.o.p ("z value:"+ s1.z);
s1.m3()
s.o.p ("*******");
}
}
```

anonymous is sub class

Anonymous inner class

we use in abstraction & runtime poli

anonymous class

real time example
real time screen you tube recommended
screen rotate

```
prg1:- package jspiders.p1;
public class Demo4
{
void test()
{
s.o.p ("test() method defined in Demo4 class");
}
}
```

f1 no name -> file name

```
prg2:- package jspiders.p1;
public interface pen
{
void write();
}
```

```
prg3:- package jspiders.p1;
public class mainclass4
{
psvm (String[] args)
{
s.o.p ("********");
Demo4 d1 = new demo4()
{
void test()
{
s.o.p ("test() overrided in anonymous class");
}
};
```

```
d1.test();
s.o.p ("--
pen p1 = new
{ public vo
{
s.o.p ("
}
void m
{
s.o.p ("
}
};
p1.writ
// p1.m
s.o.p (
}
}
```

* string i
* string
in herit
* string
* ...
1. a
2. usin
* even
the string
object c
1) hash c
2) to
3) equ
* Here
an c

```
d1.test();
s.o.p("-----");
Person p1 = new Person()
{
public void write()
{
s.o.p("writing with anonymous Person");
}
void m1()
{
s.o.p("m1 method running");
}
};
p1.write();
// p1.m1(); → we can't call because it's not present in the super class
s.o.p("Hi Hello");
}
}
```

String class

- -> java.lang → implicitly
- -> final class.
- -> hashCode() } overridden
- -> toString() } automatically
- -> equals() }

- if variable as final keyword we can't change value
- if the method has final keyword we can't override the method
- if the class has final keyword we can't inherit it.

* String is a predefined class present in java.lang package
* String is a final class so the String class can't be inherited
* String object are immutable
* two different ways of creating String object are
1. as a literal ex:- String s1 = "hello";
2. using new operator ex:- String s2 = new String("javatpoint");
* even the String class inherits the object class & by default String class will override the following method from the object class
1) hashCode method
2) toString method
3) equals method
* the hashCode method present in the String class we to generate an unique id based on the ASCII value of the data passed

* the tostring method in this string class is used to written the content
* the equal method in string class is used to check the content not the address.

classmate Page

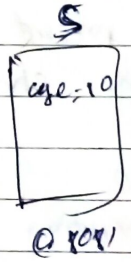
```
prg1:- package org;
  public class demo
}
psvm ( String [] args)
{
String s1=new string ("dinga");
string s2 = new string ("dinga");
s.o.p (s1);
string res = s1.tostring ();
s.o.p ("tostring value :"+res);
int val1 = s1.hashcode();
s.o.p (" hashcode code value :"+val1);
int val2 = s2.hashcode(); → we are storing the value s2 in
if (s1.equal (s2))                val2 for print purpose
{
s.o.p ("contents are same");
}
else
{
s.o.p ("contents are not same");
}
}
}
```

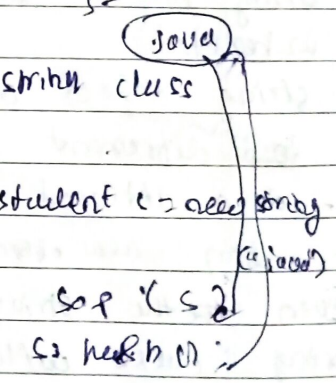
```
o/p
dinga
to string value : dinga
hashcode value : 66036483
contents are same
```

the override code is automatically in string class

normal class

```
class student
{ int age = 10;
  psvm (---)
{ student s1 = new student();
  s2 = new ---;
sop (s) → org stu@ ---
sop (s.tostring());
sop (s.hashcode); → 123
if (s1.equal(s2))
```






```
prg1:- package
public class
{ public static
  {
    s.o.p "ve
  }
  int a=10;
}
prg2:- package
public class
}
  @ override
  void start
  {
    s.o.p ("
  }
  int b = 2
}
prg3:- packa
public class
{
  @ override
  void start
  {
    s.o.p ("co
  }
  int c = 3
  }
prg4:- packa
public class
{
  void call
  {
    obj. sta
  }
  psvm (
{
  vehicle ve
  car ca
```

prg1:- package org;
public class vehicle
{ void start()
{
s.o.p ("vehicle started");
}
int a=10;
}

prg2:- package org;
public class bike extend vehicle
{
@override
void start()
{
s.o.p ("bike started");
}
int b=20;
}

prg3:- package org;
public class car extends vehicle
{
@override
void start()
{
s.o.p ("car started");
}
int c=30;
}

prg4:- package org;
public class Test
{
void callmethod (Vehicle obj)
{
obj.start();
}
psvm (String[] args)
{
Vehicle ve = new vehicle(); //ov1
car ca = new car(); //ov2

```
Bike bi = new Bike();   110x3                    O/P:-  Bike started.
Test t = new Test();                                     true
t.call method (bi);                                      true
                                                         true
s.o.p (cu instance of car);                              true
s.o.p ( bi instance of bike);   } its returns            false
s.o.p (car instance vehicle);   } boolean
s.o.p (bi instance vehicle);    }   value
s.o.p (veh instance of Bike);
}
}
```

prg1:- vehicle

prg2:- Bike   o/p

prg3:- package org.car   Bike started
                           10
                           20

```
public class test
{
 void callmethod (vehicle obj)
 {
  obj.start();        -> obj.a  (static)
  s.o.p (obj.v);      -> obj.a
  Bike bi = (Bike) obj;  -> casted to access the value of Bike class   (downcast)
  s.o.p ( bi.b);         B variable
 }
p s v m ( string [] args)
{
  Bike bi = new Bike();
  Test t = new test();
   t.callmethod (bi);
 }
}
```

instanceof operator is used to test whether the object is an instance of the specified type. it returns either true or false.

All the string objects ... as string pool.

string s1 = "java
string s2 = new str
string s3 = "java
string s4 = new
" s5 = "

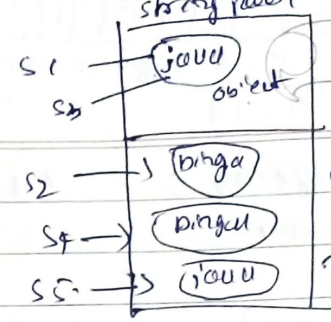
two overloaded construc
1) No arguement
2) string argumen
3) character arra
-> 2) Ex:- strin
Ex:- char[] c = {'j','a', ...}
-> 3) Ex:- string

Imp
why string ob
whether when ...
by ...
obj ... cr
obj reaching
collection.
Ex:- ① String s
②

prg1:- package
public class
{
psvm (stringZ
{
 string s = "S
 s.o.p (s.le
 s.o.p (s.
 s.o.p (s.

All the string objects are stored into a memory location called as string pool.

```
string s1 = "java";
string s2 = new string ("pinga");
string s3 = "java";
string s4 = new string ("pingal");
string s5 = new string ("java");
```



constant pool / literal
duplicates not allowed
non constant pool / new operator
duplicate value allowed

The overloaded constructor under string class are as follows

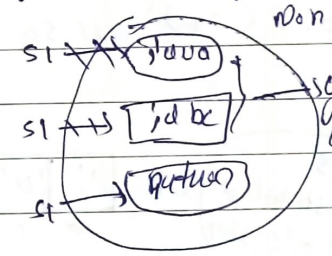
1) No argument constructor ⇒ Ex :- String s1 = new String ();
   sop(s1) -> creates empty string
2) String argument —"—
3) character array argument —"—

2) Ex :- String s2 = new String ("guldeal");
   Ex :- char[] ch = {'j','a','v','a'};
   sop(s2); -> creates object with the content
3) Ex :- String s3 = new String (ch); -> converts array into string obj
   sop. → java

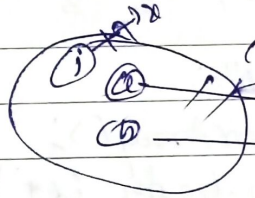
Why string objects are immutable in nature

whenever we tried to modify the data of string obj by reinitializing, then instead of modifying same obj a new obj is created and the ref will start point to the new obj leaving old obj. Old obj is eligible for garbage collection.

Ex :- ① String s1 = new String ("java");
s1 = new String ("jdbc");
s1 = new String ("python");



② String x = "j";
x = "a";
x = "b";




```
org:- package org;
public class runner
{
psvm (String[] args)
{
String s = "Software Engineer";
S.O.P (s.length());
S.O.P (s.charAt(2));
S.O.P (s.toUpperCase());
S.O.P (s.toLowerCase());
S.O.P (s.indexOf('e'));
S.O.P (s.substring(3,14));
}
}
```

o/p
17
SOFTWARE ENGINEER
software engineer
7
14
twore Engin

S | o | f | t | w | a | r | e | E | n | g | i | n | e | e | r → tware_ Engin

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

3, 14

```
prg1:- package org;
public class Demo
{
psvm(String[] args)
{
String s = "Software Engineer";
s.o.p(s.contains("are")); //true
s.o.p(s.lastIndexOf("e")); // 15
s.o.p("java".equals("Java")); //false
s.o.p("java".equalsIgnoreCase("Java")); // true.
s.o.p(s.replace('e', 'E'));
s.o.p(s);
s.o.p("========");
String s1 = "java";
s1 = s1.concat("language");
s.o.p(s1);
String s2 = "Hai";
s2.concat(" Diaga"); // created new object but not pointing to it
s.o.p(s2); // so output is Hai)
}
}
```

o/p →
true
software engineer team is ...
15
false
true
SoftwarE EnginEEr
Software Engineer
========
java language
Hai

s.o.p(s.startsWith("so")); → 28/6/19
s.o.p(s.endsWith("r"));

s → java
java program

i<=3
i<s.stringlength
i<ch.length

ch | 'j' | 'a' | 'v' | 'a'
0 1 2 3

```
String s2 "java";
char[] ch = s.toCharArray();
for (int i=0; i<ch.length; i++)
{ sop (ch[i]);
}
```

ch[0] j 0<4
ch[1] a 1<4
ch[2] v 2<4
ch[3] a 3<4

String
→ java.lang
→ jdk1.0
→ immutable
→ literal and new operator
→ not synch...
→ equals(+) op... can be used
→ for string ... hash code & eq... overridden ... class

prg:- packag...
public class ...
{
psvm(Str...
{
String s1 = ...
s.o.p(s1. ...
s.o.p(s1. ...
s.o.p(a == ...
String s2 = ...
s.o.p(s2 ...
s.o.p(s2. ...
s.o.p(s1. ...
s.o.p(" = ...
StringBuffer ...
s.o.p(sbu...
s.o.p(sa...
s.o.p("= ...
StringBuff...
s.o.p(s...
s.o.p(s...
s.o.p(...
}
}

string	string buffer	string builder
-> java.lang	java.lang	java.lang
-> jdk1.0	jdk1.0	jdk1.5
-> immutable	mutable	mutable
-> literal and new operator	new operator	new operator
-> not synchronize	synchronized	not synchronized
-> concat (+) operator can be used	can't be used (append keyword)	can't be used
-> to string method, hash code + equal are overridden in string class	only tostring method is overridden where as hashcode + equals method overridden	same

(left margin: ... inside. / EEE / peer / true. / 28/6/19 / pointing it)

prg:- package org;
public class runner
{
p.s.v.m(String[] args)
{
String s1 = new string ("java");
s.o.p (s1.tostring ());
s.o.p (s1.hash code());
s.o.p ("=======");
string s2 = new string ("java");
s.o.p (s2); //s2.tostring()
s.o.p (s2.hash code ());
s.o.p (s1.equals(s2));
s.o.p ("========");
String Buffer sbuf1 = new string Buffer ("jdbc");
s.o.p (sbuf1); //sbuf.tostring ()
s.o.p (sbuf1.hash code());
s.o.p ("========");
string Buffer sbuf2 = new string buffer ("jdbc");
s.o.p (sbuf2); // sbuf.tostring ()
s.o.p (sbuf2.hash code());
s.o.p (sbuf1.equals (sbuf2));
}

O/P :- java
2301506
=====
java
2301506
true
======
jdbc
3168322
======
jdbc
17225372
false

mutable   mutable.   c++ java
prg1:- package org;   python
Public class runner   return   progress
psvm(String[] args)
{
```
String s = new String("java");
s.concat("program");
s.o.p(s);
StringBuffer sbuf = new StringBuffer("jdbc");
sbuf.append("transactions");
StringBuilder sbui = new StringBuilder("python");
sbui.append("programming");
s.o.p(sbui);
}
}
```

error	exception 19/4/19
compile time	run time
run time	
debug the errors	handle this

finally block will executed whether exception occur or not occur

```
class Demo
{
psvm ...
{ s.o.p("start")
sop(10/0); => Arithmetic invoke exception occur so error occurs.
sop("end");
}
thread.sleep(2000);
```

try -> is used to avoid error problem
{
}
catch{} -> is used to solve to that error

2 types exception
1) checked
2) unchecked

```
package org;
public class mainclass
{
static void disp()
{
for (int i=1; i<=5; i++)
{
s.o.p(i);
try
{
thread.sleep(2000);
}
```

```
catch (exception e)
{
s.o.p("some problem");
}
}
}
```

checked -> handle immediately without execute the compiler
unchecked -> handle after execute the compiler

single tone class
1) private constructor
2) factory
3) static non-pri...

```
class Account
{
static account obj
private Account
{ s.o.p("obj
}
static Account
{
if (obj ==
obj = new
}
else
{ sop ("can
}
return (obj
```

prg1:- package
public class A
{
static Accoun
private accou
{
s.o.p("o
}
static ac
{
compiler pf (a=
{
a = ne
}
else
{
s.o.
}
}

single tone class

obj = null
0x1

1) private constructor
2) Factory
3) static non-primitive

class Account
{
(static account obj = null)
private Account()
{ s.o.p("obj created");
}
static Account create object() (return type)
{
if (obj == null)
obj = new Account();
}
else
{ sop ("cant create object")
}
return (obj);

class Test
{ psvm
{ (string return type value)
Account ref1 = Account.createobject();
ref2 = —— —

pgm:- package org;
public class account
{
static account a1 = null; → Account type
private account()
{
s.o.p("object of account created");
}
static account createobject()
{
if (a1 == null) static non primitive
{
a1 = new account();
}
else
{
s.o.p("canot create object");
} return a1;

19/4/19
to avoid error problem
is used to solve to that error
types exception
checked —
unchecked —
immediately without
after some time
by compiler
compiler execution

prg:- package org;
public class Test Account
{ p s v m (String[] args) → static method → ClassName.methodName
{ s.o.p ("Account.create object ()");
s.o.p ("-------------");
s.o.p ("-------------");
}
}

ASSIGNMENT
Date
Page

ob object of Account created.
org.account@1db9a8f2
=> =====
can't create obj
org.account@1db9a8f2

if it try to create object again, it redirects to same existing object

Singleton class is a type of class which allows to create only one object.

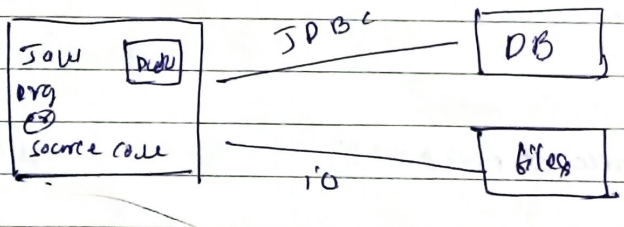
* rules for creating single tone class
1) have a private constructor → advantage: private constructor ie objects can't be created outside the class therefore we have to create objects indirectly
2) make use of factory method or helper method → this method helps to create an object & redirects to the same object again & again.
3) have static non primitive reference of class type which has to be initialize to null

20/4/19

catch block execute only when exception is occurred.

persistence = storing

File handling / programming.



With the help of java program it is possible to persistently store the data on two platform - 1) database 2) files.
* the process of reading & writting the data onto the files is called as file handling or file programming.
* to work around with files the ready made across pre defined package is called as java.io package
* the few important classes under java.io package are follows. 1) file
2) file writter
3) file reader
4) file input stream
5) file o/p stream
6) object o/p stream
7) object i/p stream
} serialization (or) de serialization

if a class pro
first whether
prg:- package or

import java.io.
public class De
{
psvm (String
{
file f=new
boolean resul
s.o.p (resul
if (f.exist
{
s.o.p ("direc
}
else
{
s.o.p ("direct
}
f.delete ();
// file f=n
}
}

prg1:- package
import java.i
public class

psvm (Strin
{
file f = new
try
{
s.o.p(
}
catch (Exc
{ s.o.p
}
}

if a class predefined we can create object but we should import first whether it's not located defaultly.

mkdir() → to create folder explicitly

```
import java.io.file;
public class Demo
{
psvm (String [] args)
{
file f=new file ("e:\\cinga");   // path to create folder, used to create folder name in E drive
boolean result = f.mkdir();   // create folder explicitly
s.o.p (result);   // storing boolean result.
if (f.exists())   // to check whether folder is present or not
{
s.o.p ("directory is present");   // true, directory is present
}
else
{
s.o.p ("directory is not present");
}
f.delete();  // folder deleted
// file f=new file("cinga");  // create folder within project itself, refresh then will show the folder
}
}
```

```
prg1:- package org;
import java.io.file;
public class demo
{
psvm (String [] args)
{
file f = new file ("e:/cinga.xlsx");   // path of file, folder name, Extension
try
{
s.o.p (f.createNewFile());   // Boolean value is returning, we're going to print directly and we can also store in variable and print out here we printing directly
}
catch (Exception e)
{ s.o.p ("some problem");
}
}
}
```

// file f = new file("cinga.xlsx"); creates file based on extension .txt, .xlsx, .docx etc within project

createNewFile() → it's going to create file (no space, capital)

Date class is in Util package

throws exception
io -II-
interrupt -II-

```
pgm:- package org;
import java.io.file;
import java.util.Date;
public class Demo
{
p s v m (String [] args)
{
file f = new file ("e:/clingu.txt");   // when its created and last modified
long milliseconds = f.lastmodified();
s.o.p (milliseconds); // unformatted date  -> o/p 1555731244784
// converts milliseconds to proper/standard date format -> sat Apr 20 09:05:24 IST2019
Date d = new Date (milliseconds);
s.o.p (d)
}
}
```

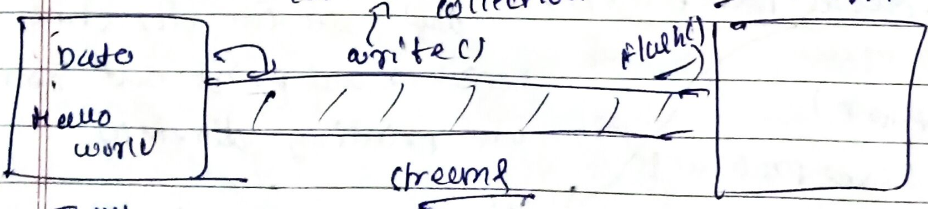
Stream means collection of byte sequence

① creation of user object
② Filewriter -> functions
(FW) -> create "w" mode
↳ file is already present in "w" mod.
opens existing file but data overrides -> means deletes the previous data on that folder

Steps to write the data into a file

1) Create an object of file class & pass the file name as the parameter to the constructor call
2) Create an object of file writer class & pass the file object as a parameter to the constructor call.
* file writer object creates a new file & open the file in write mode if there is no existing file
* if there is existing file into it's opens the same file in write mode but the data is overridden
3) call the write method and pass the data which has to be written on the file
4) Invoke the flush() method in order to write the streams of the data explicitly on to the file
5) close () the file writer object connection within the finally block

converts data into collection of stream (bytes).
-> write the data explicitly on to the file.



Java program   clingu.txt

```
Prg1:- package org;
import java.io.file;
import java.io.FileWriter;
import java.io.IOException;
public class demo
{
psvm(String[] args
{
file f = new file("demo1.txt");  -> step1
file writer fw = null;
try
{
fw = new file writer(f);  -> step2
fw.write("hello world");  -> step3
fw.flush()  -> -> step4
s.o.p("written succesfully");
}
catch(IOException e)
{
e.printstacktrace();
}
finally
{
try
{
fw.close();  -> step5
}
catch(IOException e)
{
e.printstacktrace();
}
}
}
```

Steps to read the data from a file

1) create an object of file class and pass the file name from which we have to read the data as a parameter to the constructor call

2) create an object of filereader & pass the object of the file as a parameter to the constructor

* file reader opens the existing file in read mode & throws file not found exception by the same if file is not present

3) call the read method & write respective business logic inorder to completly read the data of the file

4) it is good practice to close file reader objects & connection within the finally block

```
prgl :- package org;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
public class demo
{
psvm (String[] args)
{
File f = new File("diagram.txt");
try
{
FileReader fr = new FileReader(f);
int ch = fr.read();
while (ch != -1)
{
S.o.p ((char) ch);
ch = fr.read();
}
}
catch (FileNotFoundException e)
{
e.printStackTrace();
}
catch (IOException e);
{ e.printStackTrace();
}
}
```

o/p Hello world. Today date is also cool.

s.o.p (ch); -> instead of several times write this statement we use like this

-1 = means no data to read.

(read() return ASCII value of characters sequentially and return -1 if it has reached end of files (EOF) i.e. no data to be read

* thread is
* multiple th
* in java pr
present in
* thread cl
* common
thread Id
thread I
has been
* thread I
* thread Id
thread name
thread
* even thread
* it is possibl
thread prior
has to get
* the priorit
* 1 being th
* the default
* ACI the ab
methods
0 get Id
get Name
set Nam
get prior
set prior
-> sleep
-> run
-> start
-> com
different
① thread
② thread
③ thre
④ thr

Thread:

* A thread is like a weight scale process
* multiple threads are called perform a specific task
* in java perspective thread is predifined class present in java.lang package
* thread class have 4 overloaded constructor
* common properties are each & every thread are thread Id, thread name, & thread priority

thread Id :- thread Id is a unique number which has been assign for each & every thread
* thread Id is automatically generated
* thread Id can't be duplicated

thread name :- a thread name is a name given to each and every thread
* even threads names automatically generated
* it is possible to give user defined thread name

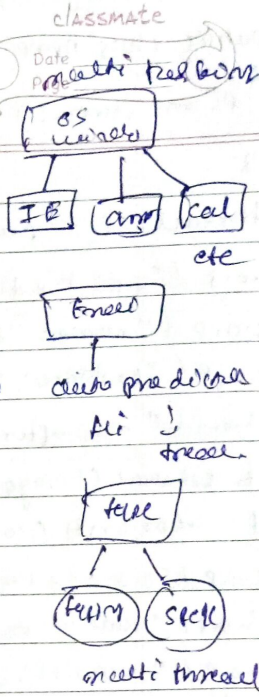
thread priority :- a thread priority is used to specify when that specific thread has to get executed
* the priorities of each & every thread lies blw the range of 1 to 10
* 1 being the lowest priority & 10 being the highest priority
* the default priority for each & every thread is 5.
* All the above properties can be access with the help of getter & setter methods

- methods under thread class

0 get Id ()
get Name ()
set Name ()
get priority ()
set priority ()
→ sleep () ← main method
→ run ()
→ start ()
→ currentThread ()

constructors under thread class are as follows

① thread ();
② thread (String name);
③ thread (Runnable r); → non primitive type and its interface type
④ thread (Runnable r, String name)



prg1:- Package org;
Public class Demo → Here we using getter & setter bcz Id, name, priority is a private
{
ps vm (String[] args)
{
thread t = new thread();
s.o.p ("ID: " + t.getId());
s.o.p ("name: " + t.getName());
s.o.p ("priority: " + t.getPriority());
s.o.p ("=== after modification ===");
t.setName ("Divya thread");
t.setPriority (8);
s.o.p ("ID: " + t.getId());
s.o.p ("name: " + t.getName());
s.o.p ("priority: " + t.getPriority());
}

o/p ID:-4
name: thread-0
priority: 5
=== after modification
ID: 4
name: Divya thread
priority: 10

prg2: package org;
public class Demo
{
psvm (String[] args)
{
thread t1 = new thread();
s.o.p ("ID: " + t1.getId());
s.o.p ("name: " + t1.getName());
s.o.p ("priority: " + t1.getPriority());
s.o.p ("-------");
thread t2 = new thread ("gulabi");
s.o.p ("ID" + t2.getId());
s.o.p ("name: " + t2.getName());
s.o.p ("priority: " + t2.getPriority());
}
}

o/p ID: 4
name: thread-0
priority: 5

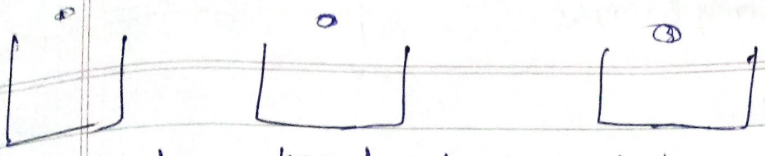
ID: 10
name: gulabi
priority: 5

→ you can pass parameter.

when we execute a java program (or) start the jvm internally 3 threads are created that is 3 separate & different stack has created.
1) main thread → main thread is used to specify entry point
(or) the starting point of the java program
2) thread scheduler → thread scheduler is used to allocate resource and manage all the threads

3) Garbage collector → dereference object
=> main thread
how to we
1) by extending th
2) by implem
Rules for creat
1) Extends th
2) override run
3) call start()
NOTE:- why do
when we cal
run method
internally
executed on
user defined
prg1:- package
Public class
{
static void
{
for (int i = 1
{
s.o.p ("
}
}
@ override
public void
{
super();
}
}

3) Garbage collector → garbage collector is used to automatically delete collect the dereference objects or unnecessary data from the memory



=> main thread thread scheduler Garbage collector

how to we can create user defined ~~class~~ threads in two ways.

1) by extending thread class

2) by implementing ~~Runnable~~ Runnable interface

Rules for creating a thread if it is extending thread class

1) extends thread class

2) override run method

3) call start() method run will execute existing stack

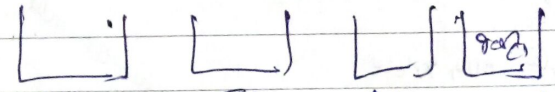
Note:- why do we call start() method not run() method

when we call the main method directly the thread and the run method gets executed on the existing stack itself

whereas when we call start method internally a new stack has got created & run() method gets executed on newly created stack therefore we have created user defined thread.


```
prg1:- package com;
public class demo extends thread
{
static void disp()
{
for (int i=1; i<=5; i++)
{
s.o.p ("i:" +i);
}
}
@override
public void run()
{
   disp();
}
}
```

after start() method



3 threads

int

nage

prg2:- package com
Public class main class
{
public static void main(String [] args)
{
demo d = new demo ();
d.start();
}
}

o/p i=1
i=2
i=3
i=4
i=5

prg1:- package com;
public class demo1 extends thread
{
start void disp1()
{
for(int i=1; i<=5; i++)
{
S.o.p("i:"+i);
try
{
thread.sleep(2000);
}
catch(interrupted exception e)
{
e.printstacktrace(); - used to what error is occurred.
}
}
}
@ override
public void run()
{
disp(); static method so we can call directly
or disp()
}
}

prg2:- package com;
public class demo2 extends thread.
{
static void disp2()
{
for(int j=6; j<=10; j++)

{ S.o.p("j:"
try
{ thread.sleep
}
catch(inter
{
e.printstac
}
}
}
@ override
public void
{
disp();
}
}

prg3:- packa
public class
{
public RSum(
{
Demo1 d1=
Demo2 d2=
d1.start()
d2.start
}
}

Sericalizat
* in java
* the object
to store
sericaliza
* the proc
byte * cal
* the process
* whenever
exteasion
* Even e
4 int

```
{ s.o.p ("i: "+i);
try
{ thread. sleep (2000);
}
catch (interruptedexception e)
{
  e.printstacktrace ();
}
}
}
@ override
public void run ()
{
disp ();          -> disp2()
}
}
```

o/p :- i=1
i=6
i=2
i=7
.
.
.

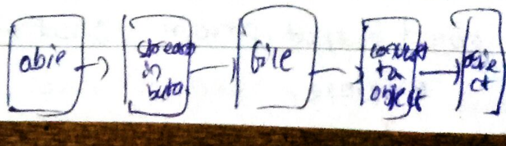
```
pro3 :- package com;
public class mainclass
{
  psvm (string [] args)
  {
  Demo1 d1= new Demo1 ();
  Demo2 d2= new demo 2 ();
  d1. start ()
  d2 start ();
  }
}
```

21|4|14

.ser - serial extension

Serialization

* in java objected are generally created inside heap memory
* the object created within the heap memory is not permanently stored therefore to store the object persistantly we come across a topic called as serialitation.
* the process of storing the objects into the file in the form of streams of byte is called as "serialization"
* the process of reading the objects from a file is called as the "de serializati-on"
* whenever we trying to serialize an objects we come across a file with the extension ".ser".
* Each and every class which objects has to be serialite as to implement a interface by the name serializable.



serializable is a marker interface present in java.io package

file output stream & file write same function

steps to serialize an object

1) create file object with the file name using the extension ".ser"
2) create an object of file outstream and pass file an object as parameter to the constructor call
3) create an object of object output stream class & pass file output stream object as parameter to the constructor call.
4) invoke write object method & pass the object that has to be serialize
5) invoke the flush() method
6) close the connection within the finally block

storing & writing object

prg1-> package org;

```
import java.io.serializable;
public class employee implement serializable   — it is agreement to the file creating
{
string name;
int id;
public employee(string name, int id)
{
this.name = name;
this.id = id;
}
}
```

prg2:- package org;

```
import java.io.file
import java.io.fileoutputstream;
import java.io.objectoutputstream;   here we going to operate with "object"
public class demo
{
psvm(string[] args)
{
file f = new file("demo.ser");   we can pass directly demo.ser to here
try
{
file output stream fout = new file output stream(f);
object output stream oout = new objectoutput stream(fout);
employee emp = new employee("ofagu", 101);
oout.writeobject(emp);
oout.flush();
s.o.p("object stored");
}
catch(exception e)
{
s.o.p("some problem");
}
}
}
```

steps to de serializ...

1) create an object ... the object read as a
2) create an object of ... the constructor call
3) create an object of ... as parameter to the cons...
4) invoke the read object ...
upcasted address reference
5) return type
6) to retrive specif... down casted
7) on the down casted ... specific class
8) it is good standard ... block
→ ...

prg1:- package org;

```
import java.io.file
import java.io.file...
import java.io.objec...
public class demo
{
public s v m (string [] ...
{
file f = new file ...
try
{
file input stream
object input str...
```

oout. writeobject (emp); — we writing objects → o/p → object stored
oout. flush();
s.o.p ("object stored");
}

→ we can read ~~from~~ employee class & object
write several time

catch (exception e)
{
s.o.p ("some problem");
}
}
}

steps to de serialize an object

1) create an object of file class & pass the file name from which the object read as a parameter to the constructor call
2) create an object of file input stream & pass file object as parameter to the constructor call
3) create an object of object input stream & pass file input stream object as parameter to the constructor call
4) Invoke the read object method which ~~returns~~ written object address in upcasted ~~address~~ reference
5) ~~written~~ return type of read object method is object class
6) to retrieve specific object details upcasted reference has to be down casted
7) on the downcasted reference call the data members with respect to that specific class
8) it is good standard to close the stream connections within the finally block → receiving the objects

prg1:- package org;

```
import java.io.file
import java.io.file input stream;
import java.io.object input stream;
public class Demo
{
public s v m (string [] args)
{
file f = new file ("demo.ser");
try
{
fileinput stream fin = new file inputstream (f);
object input stream oIn = new object input stream (fin); → stream to object convertor
```

object obj = OIn.readobject(); → I am called

// s.o.p(obj) → here we can't access the properties of employee

Employee e = (Employee)obj; → class because down casting not done

s.o.p("name:"+e.name);

s.o.p("Id:"+e.Id);

o/p name: Vinay
Id: 101

}

catch(Exception e)

{

e.printStackTrace();

}

}

extends → all properties inherited.

implements → some

not all properties.

class Test {

}

class son extends Test implements Runnable

Thread

Runnable

Demo1 IS Thread Demo1

Demo1 ——— Runnable

creation also in thread class son

class

A

B extends

Thread

B implements A

rules for creating a thread with the help of runnable interface

1) A class has to implement runnable interface present in java.lang package.

2) it is mandatory to override abstract method present in runnable interface that is overrided run method

3) in order to create new stack & execute that ~~thread in~~ specific thread in that particular stack we have to call start method since the implementation class is type of runnable interface we can't call start() method directly therefore we have to pass an object of runnable interface has a parameter to the thread object & call the start() method on the thread object reference.

```
prg3:- package org;
public class Demo1 implements Runnable
{
  @Override
  public void run()
  {
  for (int i=1; i<=5; i++)
    {
     S.O.P(i);
    }
  }
}
```

```
prg2:- package org;
public class Demo
{
  public static void
  {
    Demo d = new
    Thread t =
    t.start();
    S.O.P(t.get
    // Thread t =
    // explicitly
    (S.O.P(t.get
  }
```

Note:- the above
or inherited

program to retr

prg1:- same

prg2:- package

public class

{

psvm(String

{

Thread t=new

return

S.O.P(t);

S.O.P(t.get

S.O.P(t.ge

S.O.P(t.g

}

}

Demo

String

{

return "

(left page margin, partially visible:)
...rties of superclass ...thing not done

o/p name: Diaga
Id: 101

...properties inherited.

...all properties.

...Thread Demo1

Runnable

[Thread]

implements

...nterface

java.lang package.
in runnable

specific
...tart method
...ble interface
...we have to
...meter to the
...d object reference.

(right page:)

o/p: Diaga thread.

prg3:- package org.
public class main class
{ public static void main (String args)
{ Demo1 d1 = new Demo1();
thread t = new thread(d1); } create thread class object to run specific purpose
t. start();
s.o.p (t.get name());
// thread t = new thread (d1, "Diaga thread");
// explicitly passing runnable object and name of the thread
// s.o.p (t.get name()); //o/p Diaga thread

Note:- the advantage of implementing runnable interface is subclass can inherited another class simultaneously

program to retrieve current thread detail

prg1:- same prev prg demo1
prg2:- package com;
public class mainclass
{
psvm (string [] args)
{
thread t= thread. current thread();
// return instance of the currently executing thread
s.op (t);
s.o.p (t.get name());
s.op (t.get Id()); the field has private key access specifier
s.o.p (t.get priority());
}
}

o/p
thread [main, 5, main]
main
1
5

Demo

string data ()
{
return "diaga";
}

Demo d = new Demo
string msg = d.data()
sop (msg. length());
sop (d. data(). length()

String s = helloworld
s. length

diag. length

prg1 :- package prg

prg2 :- package com;

public class <name> mainclass

{

psvm (String[] args)

{

thread t = thread.currentThread();

S.O.P(t.getName());

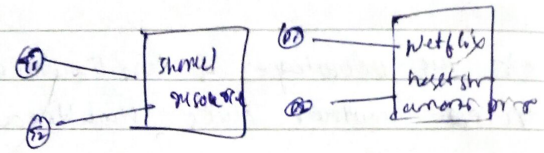
S.O.P(thread.currentThread().getName()); → directly printing

String data = thread.currentThread().getName().toUpperCase();

S.O.P(data); → storing & printing

}

}



Synchronization

When two or more threads trying to access the same shared resource at a time it gives erroneous result. (data inconsistent) or ambiguity.

* this process of multiple threads accessing the same recource simultaneously is called as "race condition".
* to overcome this condition we have to come across thread safe concept
* if one thread is accessing the shared resource the other threads are in Blocked or waiting state.
* we can make a thread safe by using synchronize keyword

prg1 :- package com;

public class NetflixAccount

{

synchronized void watching (String name)

{

S.O.P(name + " now logged in and watching some web series");

try

{

thread.sleep(10000);

}

catch (InterruptedException e)

{

e.printStackTrace();

}

without this key word thread are executing at a time

Dinga has stopped watching

Gelalu

Dinga logged out

Gelalu

Shows directly

prg2 :- package com;

public class person ext

{

String name;

NetflixAccount ac;

person (String name,

{

this.name = name;

this.acc = acc;

}

@Override

public void run()

{

acc.watching (

}

}

prg3 :- package com

public class runner

{

p.s.v.m (String[] arg

{

Netflix account.obj =

// S.O.P(obj);

person p1 = new

// person ("Dinga"

person p2 = new

// person ("Gelalu

p1.start();

p2.start();

}

}

Diagram

Thread

final

```
S.O.P (name + " has successfully logged out ");
}
}
```

ojr dinga has logged in and whatever some won series.
dinga has successfully logged out
Gulaloo has logged in and watching some web series.
Gulaloo has successfully logged out.

prg2 :-

```
package com;

public class person extend thread
{
  String name;
  Netflix Account acc;
  person (String name, Netflix Account acc)
  {
    this.name = name;
    this.acc = acc;
  }
  @override
  public void run()
  {
    acc.watching (name);
  }
}
```

prg3 :-

```
package com;
public class runner
{
  p.s.v.m ( String[] args)
  {
    Netflixaccount obj = new Netflix account();
    // s.o.p (obj);
    person p1 = new person ("dinga", obj);
    // person ("dinga", @802L);
    person p2 = new person ("Guldu", obj);
    // person ("Guldu", @802L);
    p1.start ()
    p2.start();
  }
}
```

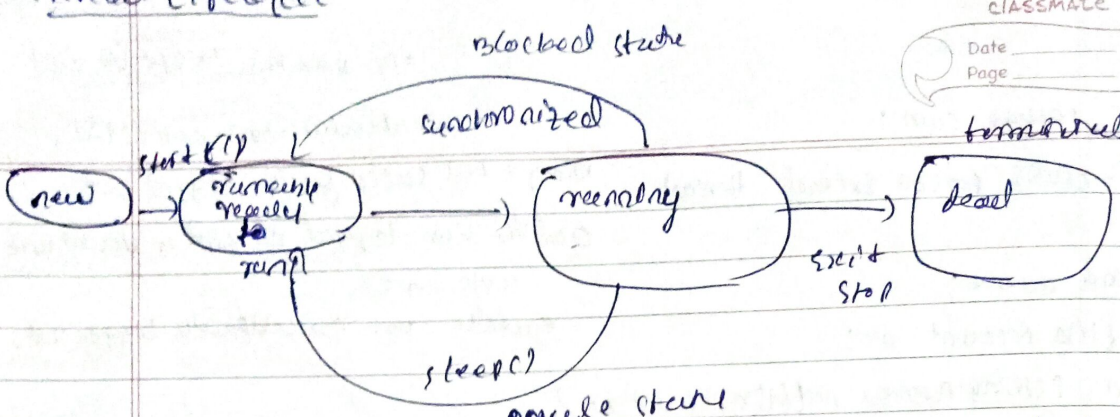
Assignments

ATM also

ATM ATM — Joint Account — Dtyme

song playing

Thread Lifecycle



Enum type

```
string s1;
s1 = "ramesh";
(or)
s1 = new string ("suresh");
(or)
s1 = "1234
```

```
eg: string dayname;
day name = "wednesday"
(or)
day name = "thursday"
(or)
dayname = "monday";
(or)
dayname = "4567";
```

24/4/19

```
int month;
month = 4;
(or)
month = 10;
(or)
month = 12;
(or)
month = 43;
```

Enum class is used to initialize set of values in predefined in it.

```
prg1:- package isplcdemo.p1;
public enum day
{
MONDAY, TUESDAY, WEDNESDAY, THURSDAY, FRIDAY, SATURDAY, SUNDAY;
   //values are static and final
}
prg2:- package isplcdemo.p1;
public class meinclass1
{
psvm (String[] args)
{
S.op("*********");
Day dayname; // Enum type corrected
dayname = day.WEDNESDAY;
S.op("today is" + dayname);
S.o.p("*********");
}
}
```

capital should be

without reference variable we call dereference the unused object

class normal
{ } { member
}

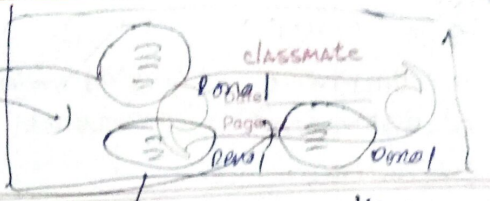
Garbage colle
→ removing
→ call de-re

Garbage c
* it is
* set last
* system.
{ class
static
How do you
modify
S.g :- pane
:
d1 =
s1.ge

objective
* IT project
* OOPS co
* collection
* Exception
* glimpse o

Book my show
* no UI
* no IO

```
class normal
{
    p s v m(){
        normal d1 = new demo1();
        new demo1();
        S.o.println(new demo1().x);
    }
}
```



dereferenceable

new operator acquire memory

Garbage collection process
- removing garbage (collection) eligible objects
- all de-referred object are eligible for G.C.

Garbage collector
* it is a low priority thread
* at last
* System.gc();- explicit call may be or may not be
* static class

How do you make object eligible for G.C?
- nullify the reference

```
e.g:- demo1 d1 = new demo1();
      :
      :
      d1 = null;
      Sl@.gc();
```

destroy object
by calling
finalize()
final vs finally vs finalize
Exception

finalize():- it is used to perform some final operations or clean up operations on an object before it is removed from the memory

Wednesday 25/1/19
Prerequirement

1) High level Requirement
HLD - High level Design
2) low level requirement
LLD
Low Level Design

Objective of project class
* IT project work
* OOPS concepts
* collection Framework Library
* Exception Handling
* Glimpse of design principles

Book my show
* No GUI (Graphic user interface)
* No JDBC

pre. List high functionalities in Bms
1) list of movies
2) Booking ticket
3) payment
4) Location
5) theatre
6) user
7)

Ex:- House
(1) Living room → Living room without
(2) Bedroom
(3) kitchen
(4) Bathroom
:

package
com.bms.user
com.bms.movies
com.bms.theatre
com.bms.ticket

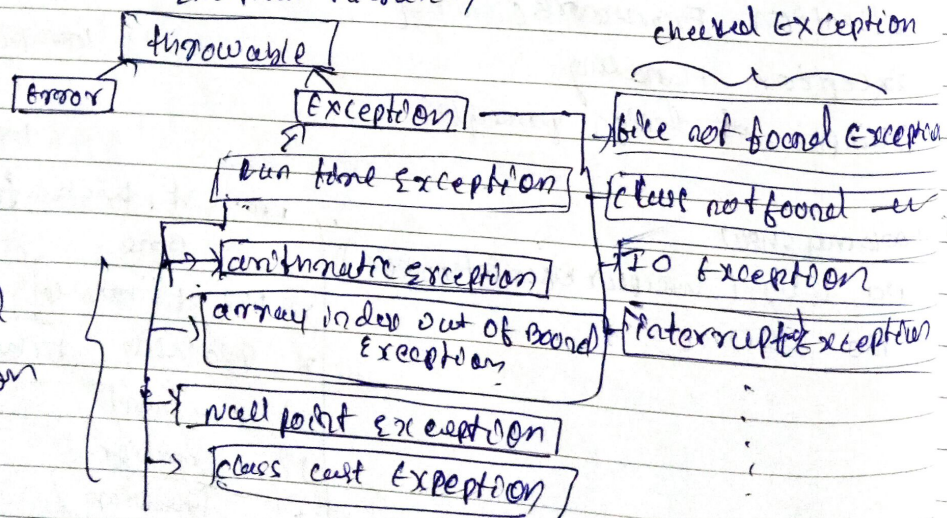
BMS1.0 → minor
BMS1.1
BMS2.0 → major

25/1/24

EXCEPTION HANDLING

interpreted means line by line execution

* exception one real time interruption which stops the program execution.
* errors also some interruption or problem which occurs two scenario 1) compile time 2) real time
* compilation error occurs due to syntactical mistakes
* runtime error occurs when we try to execute a class without main method
* the major difference btw error and exception is error has to be debug whereas exception has to be handle.
* in process of handling an exception it called as Exception handling
* typically an exception is handled using try and catch block



* try block
which might
* in simple
it generate
* the
written
* catch block

prgl:- package
import
public class
{
psvm (String
{
s.o.p("START
Scanner s=
s.o.p("
int a = s.
int b = s.
try
{
s.o.p
}
catch (Arithm
{
s.o.p("
}
s.o.p("
}
}

Note:-
prgl:- package
public class
{
psvm (strin
{
s.o.p("stu
int[] a

* try block is a block which is used to specify the set of lines which might probably give an exception.
* in simple terms any no lines which might give an exception it generally written into try block
* the solution part if an exception occurs has to be written into the catch block
* catch block gets executed only if an exception occurs

prgl:- package com
import java.util.scanner;
public class Demo
{
psvm (String [] args)
{
s.o.p("START");
Scanner s= new Scanner (System.in);
s.o.p ("enter a and b value");
int a = s.nextInt();
int b = s.nextInt();
try
{
s.o.p (a/b);
}
catch (Arithmetic Exception e)
{
s.o.p ("enter proper denominator");
}
s.o.p ("end");
}
}

o/p - start
enter a & b value
10
0
enter proper denominator
END

Note:- one try block can have multiple catch blocks

prgl:- package com
public class Demo
{
psvm (String [] args)
{
s.o.p ("start");
int[] a = new int[5];

interpreted means line by line execution
n which stops the
or problem when
2) real time
ed mistakes
to execute a class
exception is
exception has to be
ll called as
using try and catch
checked Exception
file not found exception
class not found ex
IO exception
Interrupted exception

```
try
{
 s.o.p( a[10]);
 }
catch (ArithmeticException e)
{
s.o.p ("dabba fellow, enter valid denominator");
}
catch (ArrayIndexOutOfBoundException e)
{
 s.o.p ( "mad fellow, enter valid index position");
}
   s.o.p ( "end");
  }
 }
```

it don't know exception so we can use the declare exception e

always the super class exception need to be handle last
it is a good practice to use super class exception or the default exception if we are not sure about the exception name

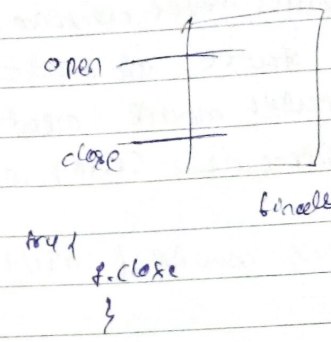
prg3 :- package demo
child exception (or) sub exception
public class demo
directly access to the exception
{
psvm (String [] args)
{
try
{
  s.o.p(10/0);
}
catch (ArrayIndexOutOfBoundException e)
{
 s.o.p ("mad fellow, enter valid index position");
}
catch (ArithmeticException e) -> sub class exception
 {
    s.o.p(" dabba, fellow, enter valid denominator");
 }
  catch (Exception e) ---> super class (or) default class
  { s.o.p ("some problem");
}}

Finally Block -
even if an exception occurred block
prg4 :- package
public class
{
psvm (String
{
 s.o.p ("start
try
 {
  s.o.p (10/0
 }
catch (Arith
 }
 s.o.p ("dab
}
finally
 {
  s.o.p ["
)
  s.o.p ("
}
creation
-> create
user create
create user
prg2 :-
package
public
{
public st
}
```
```

Finally Block - Finally Block is a Block that gets executed even if an exception occurs or not occurs.

finally block is generally use to close costly resource

```
prg:- package com
public class demo
{
psvm ( String [] args)
{
s.o.p ( "start ");
try
{
s.o.p ( 10/0);
}
catch ( ArithmeticException e)
{
S.O.P ("abba fellow, enter valid denominator ");
}
finally
{
S.o.p ("I am finally Block");
}
s.o.p ("end");
}
```



creation class

-> create objects of the specified location

user creation
createuser (-,-,-,-,-):User

movie creation
create movie (-,-,-,-,-,-) : movie

movie
user
screen
theater
ticket

```
prg:-
package com.bns.movies;
public class movies creation
{
Public static movie createmovie ( int mId, String mTitle, int
    mDuration, String mDirector, String mLang)
{
    return new movie (mId, mTitle, mDuration, mDirector,
        mLang);
}
```

Keshavsir

26/4/19

user mean class

to generate auto Id.

package com. bms. movies;

```
public class movie creation {
private static int mId = 1001;
public static movie create movie ( string mTitle, int mDuration,
string mDirname, String mlang)
{
 return new movie ( mId++ , mtitle, mDuration, mDirname, mlang);
 }
 }
```

```
package com.bms. movies ;
public class Testclass
{
 psvm( string [] arg)
{
 s.o.p ("*** ***");
movie m1 = movie creation. createmovie( "kgF", 120, "prashanth", "kumar");
movie m2 = movie creation. createmovie ( "kgF", 120, "prashanth", "kumar");
 s.o.p (m1.getmId());
 s.o.p (m2. getmID());
 s.o.p (" welcome");
 }
 }
```

mov 1001 forhead

tune

```
private static final string mporIdname;
private static int mporId2 = 1001;
private static string mId = mporIdname+ mporId2++;
```

---

checked Exception

26/4/19

uncheckEd Exception are those Exception which the compiler forces to the handle it immediately

- unchecked Exception → unchecked Exception are those Exception which the compiler does not force you to handle it immidiately

-

checked

prg1 :- pac
public c
{
psvm
{
for (
{
s.o
try
{
thread
}
catch (
{
s.o.p
}
}
}
}

prg1 :- pa
public
{
psvm (
{
s.o.p (
}
}

→ Handle
→ compile
→ interrup
tile ne

Checked Exception

```
prg1:- package com;
public class Demo
{
psvm (String [] args)
{
for (int i=1; i<=5; i++)
{
s.o.p (i);
try
{
Thread.sleep (1000);
}
catch (InterruptedException e)
{
s.o.p ("some problem");
}
}
}
}
```

o/p
1
2
3
T
5

Unchecked Exception

```
prg1:- package com;
public class Test
{
psvm (String [] args)
{
s.o.p (10/0);
}
}
```

Checked Exception	Unchecked Exception
-> Handle immediately	-> will not handle immediately
-> compiler understand CE	-> compiler does not understand UCE
-> InterruptedException	-> ArithmeticException
file not found	-> array

methods in throwable class

1) print stack trace method

* print stack trace method is used to print the complete information about the exception occurred.
* this method generally used by the developers in order to retrieve the exception name, predefined msg & the line no where the exception had occured.

prg 1:- package com;
public class Demo
{
public static void main (String args)
{
S.O.P ("START");
try
{
S.O.P (10/0); - Exception object created from ArithmeticException
}
catch (Exception e)
{
e. print stack trace ();
}
S.O.P ("END");
}
}

o/p = start
java. lang. ArithmeticException: / by zero
at com. Demo. main (Demo.java:10)
end

internally Exception is created and thrown on arithmetic exception in o/p execution

2) get message method -> this method is used to retrive the predefined message about the exception.

prg 1:- package com
public class Demo
{
p sum (String [] args)
{
S.O.P ("START");
try
{
S.O.P (10/0);
}
catch (Exception e)
{
String msg = e. get message ();

Def in this prg first I have to use e. print stack trace () then run it and delete this then it will print stack trace with its msg

o/p Start
/ by zero
END

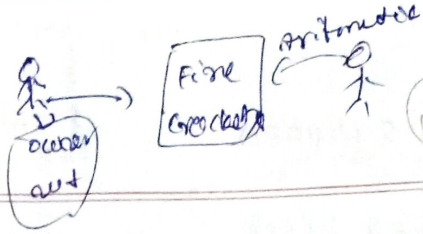
S.O.P (msg);
}
}

throws

* throws is a
of exception
with
* throws key
handle within
that Exceptio
* throws key

prg 1 :- package com
public class De
{
void div() th
{
S.O.P (10/0
}
public static
{
S.O.P ("ST
Demo d = ne
try
{
d.div();
}
catch (Arithm
{
S.O.P (" DO
}
S.O.P ("E
}
}

```
s.o.p (msg);
}
s.o.p s.o.p ("END")
}
}
```



throws keyword

* throws is a keyword which is used to indicate the possibility of exception to the caller. throws keyword is generally used with methods
* throws keyword in simple terms means that exception not been handle within the method and indicating the caller to handle that exception at the caller side
* throws keyword are generally used with checked exception

prg1 :- package com

o/p :- start
Do not divide by zero
end

```
public class Demo
{
void div() throws ArithmeticException → you can give super class
{
  s.o.p (10/0);
}
public static void main(String[] args)
{
  s.o.p ("START");
  Demo d= new demo();
  try
  {
    d.div();
  }
  catch (ArithmeticException e)
  {
    s.o.p(" Do not divide by zero");
  }
  s.o.p("END");
}
}
```

```
pg 1: package com;
public class Demo
{
 void disp() throws Exception
 {
  for(int i=1; i<=5 ; i++)
  {
  s.o.p (i);
  thread.sleep(1000);
  }
 }
public static void main (String []args)
 {
  s.o.p ("start");
  Demo d = new Demo();
  try
  {
   d.disp();
  }
  catch( Exception e)
  {
  s.o.p ( "some Issue");
  }
  s.o.p( "end");
 }
}
```

o/p -> start
1
2
3
4
5
end

is declared
not apply

27/4/19 (lecture)

String objects

1) array	limited in size	—	temporary	traditional knowledge	—
	no size limit fixed	simple & easy			not centralized
2) collections Framework Library	-"-	Little complex	-"-	-"-	-"-
3) Files (unstructured)	-"-	-"-	permanent	file handling program	-"-
4) Database (structure)		-"-	-"-	SQL query & DBMS	centralized

↓ means data access at both user & admin

user software ->

UserDB
- userdB : User[size]   - size : int   - count : int
add user (user) : void   remove user (uId) : void   get user (uId) : User   get user count () : int

user type object -> - userdB
private static (size, count)
operation on user database { (methods)

class User D…
{
private st…
private s…
private st…
public u…
{
this.s…
userdb =…
}
void add…
{
if …
userDb[…
else
sop("…
}
void remo…
{
for(int…
{
if(user…
}
userDb[i…
ucount--…
break…
}
}
user get…
{
user …
for (i…
{
if (…
{…
}
}

classmate

```
class userDB
{
private static user[] userdb;
private static int ucount;
private state int size;
public userdb (int size)
{
  this.size = size;
  userdb = new user[this.size]
}
void adduser (user u1)
{
 if if (ucount < size)
  userDb[ucount++] = u1;
 else
  Sop("No space in db")
 }
void remove user (int uId)
{
 for( int i=0 i< count ; i++)
 {
  if(userDb[i].getuId() == uId);
 {
  userDb[i] == null;
  ucount--;
   break;
 }
 }
user getuser( int uId)
{
  user u1 = null;
 for (int i=0; i<ucount ;i++)
  {
  if (userDb[i].getuId() == uId)
  {
    u1 = userDb[i];
    break;
  }
  }  return u1;
}
```

classmate

Date
Page

```
userDB udb = new userDb(10);
udb.adduser (----);
udb.adduser (---);
udb.adduser (---);
int getuser count()
{
 return ucount)
}
```

27] 4/19

decentralised knowledge	not centralized
—"—	—"—
handling program	—"—
query & errors	centralized ↓

access of both user & admin

we can't ride any excuseable piece of code b/w try & catch blocks

classmate
Date 07/7/19
Page

throw keyword

throw is a keyword which is used explicitly to invoke an exception.

* By invoking an exception we have to invoke the object of exception type

throw: invoking a exception object

```
prg1:- package (no);
public class Demo
{
psvm(String[] args)
{
s.o.p ("start");
int age=5;
if (age>=18)
{
s.o.p ("issue DL");
}
else
{
try
{
ArithmeticException obj = new ArithmeticException();
throw obj;
}
catch (ArithmeticException e)
{
s.o.p ("no not issue DL");
}
}
s.o.p ("end");
}
}
```

o/p start
no not issue DL
end

Unchecked exception

custom Exception (or) user define exception

* based on the project it is sometime is necessary to create user defined exception & those exception called as custom exception

steps for creating custom exception

① create a class with the exception name
② make that class extends runtime exception if you want to create unchecked exception

③ if you want create checked exception extends the exception class directly

prg1:- package com;
public class AgeInvalidException extends RuntimeException
{
}

prg2:- package com;
import java.util.Scanner;
public class Demo
{
psvm (String[] args)
{
s.o.p ("start");
Scanner s = new Scanner (System.in);
s.o.p ("enter age");
int age = s.nextInt ();
if (age >= 18)
{
s.o.p ("Apply for DL");
}
else
{
try if you want unchecked exception declare object creation and throw keyword also in try block
{
AgeInvalidException e = new AgeInvalidException ();
throw e;
}
catch (AgeInvalidException e)
{
s.o.p ("have patience, wait until you're 18");
}
}
s.o.p ("end")
}
}

o/p start
enter age
have patience, wait until you're 18
end

Prg 1:- ~~insufficient Balance Ex~~

package com

public class insufficient balance Exception extends Exception

{

string msg;

insufficientbalance Exception (string msg)

{ this.msg = msg;

}

}

O/P

Enter Amount to be withdrawn

5000

cannot withdraw Amount

com.insufficientbalanceExcep.

at com.ATM.main

Prg 2:- package com

import java.util.scanner;

public class ATM

{

PSVM(string[] args)

{

int balance = 5000;

scanner s=new scanner(system.in);

s.o.p("Enter amount to be withdrawn");

int amt =s.next();

if(amt <= balance)

{

s.o.p("withdrawal successfull");

}

else

{

if you want checked exception you have to declare throw key word in try block

Insufficient balanceexception i=new Insufficient balanceexception("cannot withdrawmoney")

try

{

throw i;

}

catch (insufficient balanceException x)

{

s.o.p(x.msg)

s.o.p(i.msg)

x.printstacktrace();

}

}

}

catch con

package com;

public class Recu

{

PSVM (string [

{

s.o.p ("stor

try

{

s.o.p (10/

int[] a = n

s.o.p (a[a

}

catch(Arithm

{

4. printsta

}

s.o.p ("

}

}

array is

(1) datatype [

int [

(2) arrayn

(3) string [

(4) s[]

(5) Double

Prg1:- packag

public class

{

int id;

String name

Student (

{ this.

this.

}

catch can declare N no of exception

```
package com;
public class Demo
{
psvm (String [] args)
{
S.O.P ("start");
try
{
S.O.P (10/5);  // if it (10/0) then it will show arithmetic exception
int[] a = new int [2];  // if it (10/5) it not exception so then second it it gives exception it will show the exception when it one is not occurred.
S.O.P (a[20]);
}
catch (ArithmeticException | ArrayIndexOutOfBoundsException e)
{
e.printStackTrace();
}
S.O.P ("End");
}
}
```

o/p
start
2
End
array index out of Bounds Exception

Arrays

array is used to store the group of data.

default value of non-primitive is zero

① datatype [] arrayName ; → declaration
int [] a ;

② arrayName = new datatype [size] ; → creation
a = new int [3] ;

a = | 0 | 0 | 0 |
0 1 2

③ String [] s = new String [3]; → s = | null | null | null |
0 1 2

④ s[1] = "divya"

⑤ Double d = {1.21, 3.56, 4.75} ; d = | 1.21 | 3.56 | 4.75 |
0 1 2

```
mgl!- package com;
public class student
{
int id;
String name;
student ( int id, String name)
{ this.id = id;
this.name = name;
}
```

```
p s v m( string []args)
{
student s1 = new student (1, "prasad");
student s2 = new student( 2, "crk");
student s3 = new student (3, "tom");
s.o.p ( s1);
s.o.p ( s2);
s.o.p ( s3);
student []std = new student [3];
std [0] = s1;
std [1] = s2;
std[2] = s3;
for (int i=0 ; i<std.length; i++)
{
s.o.p ( std[i]);
s.o.p (std[i].Id + " " +std[i].name);
}
}
}
```

o/p -> com.student
@1234
com.student
12 crk
com.stu
3 tom

to convert string to char pass which hold values
character. toString( )

to convert char to string pass which hold the values
toCharArray( )

```
prg: package com;
import java.util.scanner;
public class Demo
{
s.o.p( start)
scanner s = new scanner (system.in);
s.o.p ( "enter number");
int a = s.nextInt ();
s. nextByte();
s.next short();
s. next long();
s.next float();
s.next double();
s.next();
s.next Boolean();
s.next().charAt(0);
s.o.p ( a);
s.o.p ("End");
}
}
```

```
prg: package com;
public class Demo
{
psvm ( string[] ar
{
double[]d = { 10.5
for (int i=0;
{
s.o.p ( d[i]);
}
s.o.p ("===
for (double i:d
{
s.o.p (i);
}
s.o.p ("===
string[]name = {
for (int i=0;
{
s.o.p (name
}
s.o.p ("===
for (string s:
{
s.o.p(s);
}
}
}
```

Data

* Data structure
* in simple te
representing
Collec
collection frame
object
* the major
1) collection
2) to mani
are perfor

```
pkg1:- package com;
public class demo
{
psvm (String[] args)
{
double[] d = { 10.5, 11.6, 12.7 };
for (int i=0; i<d.length; i++)
{
s.o.p (d[i]);
}
s.o.p ("=========");
for (double i:d)
{
s.o.p (i);
}
s.o.p ("=========");
String[] names = {"dinga", "gulabu"};
for (int i=0; i<names.length; i++)
{
s.o.p (names[i]);
}
s.o.p ("=========");
for (String s:names)
{
s.o.p(s);
}
}
}
```

for(data type to be retrieved or iterable) { traversing array variable or name or collection object }

Data structure:

30/4/19

* data structure is use to represents the structure of the specific data.
* in simple terms data structure is a way of arranging (or) representing the data.

Collection Framework

* collection frame work use to store a group of data (or) collection of object
* the major advantages of collection are as follows
1) collection can store homogenies and heterogenies of data.
2) to manipulate to arround data insertion & deletion are performed using pre defined method.

3) collection is dynamic in nature ~~that it~~ used it item increase the size & decrease the size based on the requirement.

method under

return

header

Iterator

boolean

List interface

* list is a
* list is a
sorted from
* insertion
* duplication
* null inserti
* List inter

Array

multithreaded

* array list
* array list
* array list
* the initial
* the internal
re.sizeable
* the increase

NOTE:- * add
* size

method under collection interface

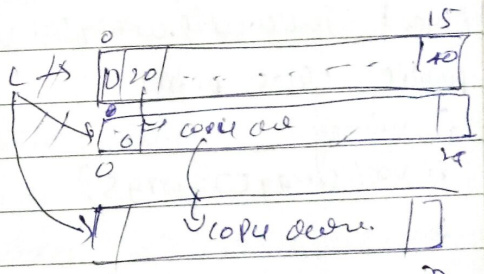
return type	method name
boolean	add (Object e)
"	addAll (Collection c)
"	remove (Object o)
"	removeAll (Collection c)
"	contains (Object o)
"	containsAll (Collection c)
int	size ()
void	clear ()
Iterator<E>	Iterator ()
boolean	isEmpty ()

List interface specification

* List is a predefined interface present in java.util package.
* List is a sub interface of collection and was started from Jdk 1.2.
* Insertion order is maintain in terms of list.
* Duplication of values are possible in the List.
* Null insertion is also possible.
* List interface one index base. → ArrayList

multithreaded

```
ArrayList l = new ArrayList();
l.add(10);
l.add(20);
:
l.add(150);
l.add(120);
```



* ArrayList is implementation class of the List interface.
* ArrayList was introduced from Jdk 1.2.
* ArrayList is present in java.util package.
* the initial capacity of the ArrayList object is 10.
* the internal data structure of ArrayList is resizeable array or growable array
* the incremental capacity of the ArrayList object is

$$\frac{\text{current capacity} \times 3}{2} + 1$$

Note:- * add method() is used to insert the elements in
* size() method is used to find the length of the object.

* get() method is used to retrive the value stored on the index position

```
prg1:- package org;
import java.util.Arraylist;
public class demo
{
psvm(string[] args)
{
arraylist l = new arraylist();
l.add(10);
l.add(20);
l.add(10);
l.add(null);
l.add(100);
s.o.p("size:"+l.size());
s.o.p(l);
for(int i=0; i<l.size(); i++)
{
s.o.p(l.get(i));
}
}
}
```

o/p :- size:5
[10, 20, 10, null, 100]
10
20
10
null
100

classmate Date Page 00

```
prg1:- package org;
import java.util.arraylist;
public class demo
{
psvm(string[] args)
{
arraylist l = new arraylist();
C.add(10);                    // adding the data into arraylist
l.add("durga");
l.add(null);
l.add(20.26);
l.add(10);
s.o.p(l.isEmpty());  // checks if the arraylist is empty or not
s.o.p("-----==>===");
for(int i=0; i<l.size(); i++)  // size() is used to find the length
{                              of the collection objects
s.o.p(l.get(i));
}
```

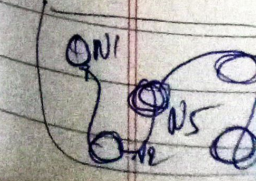
o/p : false
=====
10
durga
null
20.26
10
=====
true

```
l.clear();
s.o.p("==
s.o.p(l
```

```
prg1:- packag
import java
public class
{
psvm(stri
{
arraylist l
l.add(10
l.add("ra
l.add(20.
s.o.p(l);
s.o.p("=
s.o.p(l.
s.o.p("=
l.add(1,25
s.o.p(l);
```

* Linke
data is
* each node
as well as
* linked list
and queue
* it acce
* it is

null



l.clear(); // removes all the data from the array list
s.o.p("=====");
s.o.p(l.isEmpty());

pkg1 :- package org;
import java.util.arraylist;
public class demo
{
psvm(String[] args)
{
arraylist l = new arraylist();
l.add(10);
l.add("vinay");
l.add(20.67);
s.o.p(l);
s.o.p("======");
s.o.p(l.contains(20.69)); // used to check if the specified data is present in the array list or not
s.o.p("======");
l.add(1,250); // adds 250 in index pos 1 and the existing elements will get shifted
s.o.r(l);

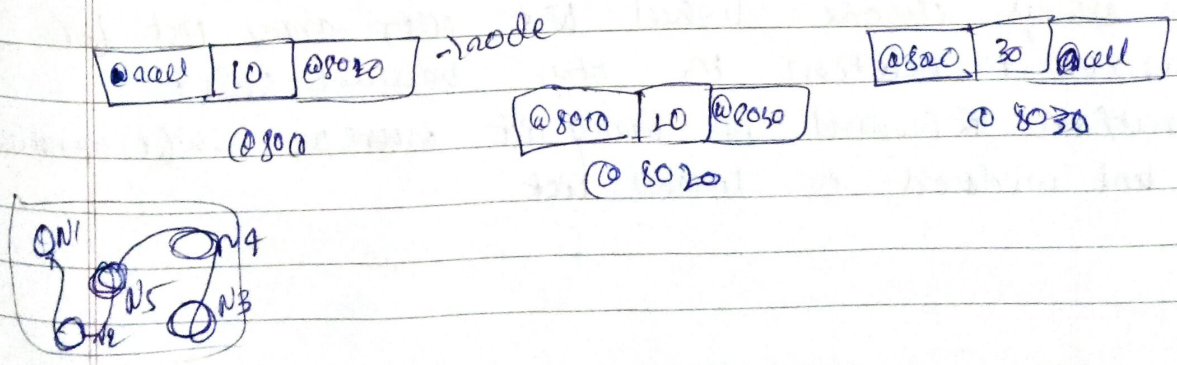
o/p [10, vinay, 20.67]
---
true
===
[10, 250, vinay, 20.67]

1/5/19

~~memory allocation~~

Linked list

* Linked list is a data structure which stores the data in the form of nodes.
* Each node is going to store the address of previous node as well as next node
* linked list is an implementation class of the list interface and queue interface.
* it was introduced from JDK 1.2
* it is present in a package is called as java.util



prg 1:- package org;
→ import util.LinkedList;

```
public class Demo
{
  psvm(String[] args)
  {
    LinkedList l = new LinkedList();
    l.add(10);
    l.add(20);
    l.add(30);
    l.add(45.24);
    for (int i=0; i<l.size(); i++)
    {
      s.o.p (l.get(i));
    }
    l.remove(2);
    s.o.p ("--------");
    for (int i=0; i<l.size(); i++)
    {
      s.o.p (l.get(i));
    }
  }
}
```

2 type's

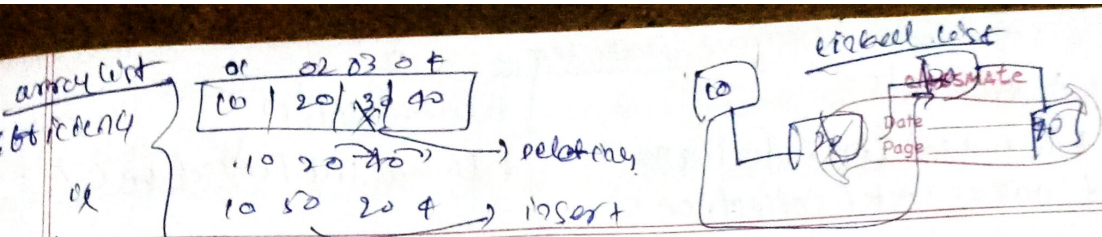
single → [|]

double → [| |]

the underlined data structure for linked list is double linked list

Array list	Linked list
→ jdk 1.2	→ jdk 1.2
→ java.util	→ java.util
→ growable / resizable array	→ Double linked list
→ 10 & $\left(\frac{c \times 3}{2}+1\right)$	→ No concept of looping
→ implementation of list	→ implement class of list & queue
→ sequential memory allocation	→ non-sequential memory allocation
→ storing & retrieving the data together	→ insertion or deletion in btw

we always choose linked list over array list for insertion & deletion in btw because shift operation is involved in array list whereas shift operation is not involved in linked list



we choose array list over linked list for completely storing retrieving the data

Vector → single threaded

* Vector is an implementation class of the list interface.
* vector is a legacy collection which has started from JDK 1.0
* vector is present in java.util package
* the underlined data structure for vector is resizable array (or) growable array
* the initial capacity of vector is 10.
* the incremental capacity of vector is current capacity X 2
* vectors where single threaded therefore is comparitably lesser use in the industry.

```
pkg:- package org;
import java.util.Vector;
public class Demo
{
psvm(String[] args)
{
Vector v = new Vector();
v.add(10);
v.add(20);
v.add(30);
S.O.P(v);
}
}
```

o/p [10, 20, 30]

+ -> public
- -> private

array list constructor

3 Vectors Constructors

+ ArrayList()
+ ArrayList(int initialCapacity)
+ ArrayList(Collection c)

2

+ LinkedList()
+ LinkedList(Collection c)

4

+ Vector()
+ Vector(int initialCapacity)
+ Vector(int initialCapacity, int incrementalCapacity)
+ Vector(Collection c)

prog 1:-

```
package org;
import java.util.*;
public class Demo
{
psvm(String[] args)
{
Vector v = new Vector();
v.add(10);
v.add(20);
v.add(30);
S.o.p("Vector Values:"+v);
S.o.p("Vector size:"+v.size());
S.o.p("==========");
ArrayList a=new ArrayList(v);
a.add("Divya");
S.o.p("ArrayList values:"+a);
S.o.p("ArrayList size:"+a.size());
a.remove(1);
S.o.p("ArrayList values:"+a);
}
}
```

o/p

Vector values: [10, 20, 30]
Vector size: 3
ArrayList values: [10, 20, 30, Divya]
ArrayList size: 4
ArrayList values: [10, 20, 30, Divya]

prog 1:-

```
Same as above
Vector v = new Vector
v.add(10);
v.add(20);
ArrayList a = new ArrayList(v);
a.add(v);
```

o/p ArrayList values: [[10, 20]]
ArrayList size: 1

prog:-

```
Same as above
v.add(10);
v.add(20);
a.add("Divya")
a.addAll(v);
a.add(10,40);
```

→

do above sc
object of a
used to ac

· Stack

* Stack is a
* Stack be
* Stack is

prog 1:- package
import java.
public class
{
psvm(Str
{
Stack s=ne
s.push(10
s.push("h
s.push(3
s.o.p(s);
s.o.p(s.pop(
s.o.p(s);
s.o.p(s.pee
s.o.p(s);
}
}

wrapper

* primitive
key words
* in Java
* if a data
data type
* the best ex
* the collecti
∴ java
for the
generalized
data type

in above scenerio it is good to add the value of vector object at a single entity i.e together where as add() is used to add the values of an object independent.

stack

* steack is also legecy collection introduced from jdk 1.2
* steack follows last in first out approach.
* steack is presents in java.util package.

```
prg1:- package org;
import java.util.steack;
public class demo
{
  psvm (string[] args)
{
steack s = new steack();
s.push(10);
s.push("hai");
s.push(30.56);
s.o.p(s); //display the stack data.
s.o.p(s.pop()); //delete or remove top most data or last inserted data.
s.o.p(s);
s.o.p(s.peek()); //displays top most data/object
s.o.p(s);
}
}
```

o/p [10, hai, 30.50)
30.56
[10, hai]
hai
[10, hai]

collection not support primitive type

wrapper classes

* primitive data types are those data type which are predifined key words
* in java there are 8 primitive data type
* if a data type is also a class, then it is consider as non primitive data type
* the best example for non primitive data type is string class
* the collection frame work will not support primitive datatype
∴ java people have to create an alternative or wrapper for the respective primitive data type. therefore generalized name for all the non primitive version of the primitive data type are called as wrapper class.

Primitive data type	non primitive data type
byte	Byte
short	Short
int	Integer
long	Long
double	Double
float	Float
char	Character

wrapper class → all class upper class

generics < >

* generics<> is used to specify the element type.
* In simple terms it is used to tell what type of data that has to be stored into the collection.
* generics <> has to be specify with help of angular braces
* generics was introduced from JDK 1.5.

```
prg1:- package arr;
import java.util.ArrayList;
public class demo
{
psvm ( String[] args)
{
ArrayList<String> l = new arraylist<String> ();
   l.add ("dinqu");
   l.add ("tom");
   l.add ("guldu");
for (int x=0 ; x< l.size() ; x++)
  {
   s.o.p (l.get(x));
  }
   s.o.p ("------");
   for ( String s: l)
      {
        s.o.p (s);
      }
  }
}
```

all dinqu tom guldu dinqu tom guldu

prg1:- package
import java.
public class

psvm (s
LinkedList
l.add(26)
l.add(29)
l.add(45)
for (Integer
{
s.o.p(
// non primi
one called
}
}
}

prg1:- // sam
// LinkedLis
// by def
LinkedList
l.add(26)
l.add(2
l.add(
l.add(
for(Obje
{
s.o.p
}
}
}

prg1
Public
}
String
int id
Employ
{ busi
tusk

prg1:- package org;
import java.util.LinkedList;
public class Demo
{
psvm(String[]args)
{LinkedList<Integer>L=new LinkedList<Integer>();
L.add(26);
L.add(24);
L.add(45); → auto boxing
for(Integer i:int i:l) // for(int i:l)
{
s.o.p(i);
// non primitive version or the class version of primitive datatypes are called as wrapper classes all wrapper are present in java.lang package
}}
}

prg1:- save and above prg.
// LinkedList<Object>l=new LinkedList<Object>();
// by default generic type is object
LinkedList l=new LinkedList();
l.add(26);
l.add(24.56);
l.add("45");
l.add('true');
for(Object o:l) // object can traverse any type of ele.
{
s.o.p(o);
}
}

storing objects into collection

prg1:- package org;
public class Employee
{
String name;
int id;
Employee(String name, int id)
{this.name=name;
this.id=id;}

```
Pry2:- package org;
import java.util.arraylist;
public class multiclass
{
  psvm (String args[])
{
Employee e1 = new Employee ("Tom", 102);
Employee e2 = new Employee ("Jerry", 109);
Employee e3 = new Employee ("Bheem", 104);
s.o.p ("address of all the object");
s.o.p (e1);
s.o.p (e2);
s.o.p (e3);
arraylist <Employee> l = new Arraylist <Employee> ();
//restricted but only Employee objects can be stored so generic
type is employee
l.add(e1); // adding the object into arraylist i.e object
l.add(e2);                                    address
l.add(e3);
s.o.p ("traversing using for-each loop");
for (Employee emp : l)
  }
  s.o.p(emp);                    // print addresses
  s.o.p (emp.id+"\t"+emp.name); // print correct address
}
}
```

Queue interface 3/5/19

* Queue is a data structure which follows first in first out (FIFO) traversing.
* queue is the sub interface of the collection interface
* It was started from JDK 1.5
* queue interface present in java.util package
* the process of adding an element into queue is called as ENQUEUE
* the process of removing an element from queue is called as DEQUEUE
* first element inside the queue is always referred head & the last element is referred as tail.

priority queue

* priority queue is
* it was started fro

* pry1:- package org
import java.util.pri
public class tes
{
psvm (string[] ar
{
priorityQueue q
q.add (10);
q.add (20);
q.add (30);
s.o.p (q);
s.o.p (q.size());
s.o.p (q.isEmpty
q.clear();
s.o.p ("=====
s.o.p (q);
s.o.p (q.size
s.o.p (q.isEm
}
}

	element
working	remove
	value

if queue	no such
for empty	exceptio

org1:- package
import java.u
public class test
{
return
}
priority

priority queue

* priority queue is an implementation class of ~~the queue~~ interface
* it was started from jdk 1.5.

```
* pry1: package org;
import java.util.PriorityQueue;
public class test
{
psvm(String[] args)
{
PriorityQueue<Integer> q = new PriorityQueue<Integer>();
q.add(10);
q.offer(20);
q.add(30);
s.o.p(q);
s.o.p(q.size());
s.o.p(q.isEmpty());
q.clear();
s.o.p("=========");
s.o.p(q);
s.o.p(q.size());
s.o.p(q.isEmpty());
}
}
```

o/p = [10, 20, 30]
3
false
[]
0
true

we can use add and offer also

	element()	peek()	remove()	poll()
working	remove head value & return	return head value	removes & return head value	removes & return head value
if queue is empty	No such element exception	null	No such element exception	null

```
org1: package org;
import java.util.PriorityQueue;
public class test
{
psvm
{
PriorityQueue<Integer> q = new PriorityQueue<Integer>();
```

o/p [10, 20, 30]
10
null

```
q.add(10);
q.add(20);
q.add(30);

S.o.p(q);
S.o.p(q.poll());
q.clear();
S.o.p(q.poll());
}
}
```

Auto boxing → P → WP (conversion)
Un boxing → WP → P

Auto Boxing and Auto Unboxing

* to string method is explicitly overridden in all wrapper classes

```
prg1:- package org;
public class test
{
int a=10;          // primitive representation
integer b=20;      // non primitive representation
integer c = new Integer(30);  // non primitive representation
S.o.p (a+" "+b+" "+c);
}
}
```

o/p 10, 20, 30

* the process of converting primitive data type into non primitive data type is called as auto boxing.
* the process of converting non primitive data type into primitive data type is called as auto unboxing

```
Prg2:- package org1;
public class test
{
psvm(String[] args)
{
S.o.p("===Boxing===");
int a=10;
Integer b=a;
Integer c= new Integer(a);
S.o.p(a+" "+b+" "+c);
S.o.p("===Unboxing===");
Double d = new Double(3.6)
double e = d;
S.o.p(d+" "+e);
}
}
```

o/p
===Boxing===
10 10 10
===Unboxing===
3.6 3.6

```
pkg:- package org;
public class student
{ int id;
  String name;
  student (int id, String name)
  {
    this.id = id;
    this.name = name;
  }
  @Override
  public String toString()
  {
    return "student [id=" + id + ", name=" + name + "]";
  }
}
pkg:- package org;
import java.util.ArrayList;
public class Test student
{
  psvm (String[] args)
  {
  student s1 = new student (1, "anya");
  student s2 = new student (2, "golden");
  ArrayList <student> l = new ArrayList <student> ();
  l.add (s1);
  l.add (s2);
  l.add (new student (3, "bingi"));
  l.add (new student (4, "toon"));
  for (student std : l)
  {
    S.O.P (std);
  }
  }
}
```

or student [id=1, name=anya]

to add string object collection

```
package org;
public class Car
{ int id = 10;
public void main (---)
Car c = new Car();
S.o.p(c);
S.o.p (c.id);
S.o.p (new Car().id);}
```

IMP → creation of objects

```
pry1:- package org;
public class Car
{
private string company;
public Car(string company)
 {
   this.company = company;
}
public string getCompany()
{
  return company;
  }
}
```

obj address is: org.Car@12ea47b2

Bwm

:

```
pry2:- package org; import java.util.*;
public class TestCar
public Con (---)
{
 List<Car> l = new LinkedList<Car>();
 l.add( new Car("Bmw"));
 l.add (new Car("suzuki"));
 l.add(new Car("tesla"));
 for (Car c : l)
 {
   S.o.p ("obj address is: "+c);
   S.o.p ( c.getCompany());
  }
 }
}
```

```
pry1:- package org
import java.util
public class Test
{
  psvm (String
  {
   int a=20;
   double
   ArrayList l = n
   l.add(a); //
   l.add(b);
```

Set

* Set is
* it was intr
* it is
* set colle
* insertion
* there is
  set int
* since index
  we have
* the implem
  2) Link
* it i

1) Hash
of

* it is pre
* it allows
* Hash set
* only one
* duplicate
* the inte
  where to
  pred
* Unique

org :- package org
import java.util.*;
public class Test extr
{
psvm (String[] args)
{
int a=20;
double b=9.56;
ArrayList l=new ArrayList();
l.add(a); // inserting primitive but internally it's like new Integer(a); ie Auto boxing
l.add(b); // internally it's like new Double(b);

int a=10;

internally

l.add(a) → new Integer(a);

array list folder

Set interface 6/5/19

* Set is a sub interface of the collection interface
* it was introduced from JDK 1.2
* it is present in java.util package
* set collection it will not allow duplicate
* insertion order is not maintain
* there is no concept of indexing Applicable on set interface
* since indexing is not possible we can't use get method therefore we have to use for each loop (or) iterators.
* the implementation classes are 1) Hash set 2) Linked Hashset 3) tree set
* it is a hetero genious

1) Hash set :- * hash set is an implementation class of the set interface.

* it is present in java.util. package
* it was started from JDK 1.2
* Hash set doesn't maintain insertion order
* only one null value is allowed
* duplication is not possible
* the internal (or) underlined data structure is Hash table where in the data is stored in terms of key and value pair
* Uniqueness of Hash set is dependent on the equals method

* the initial capacity of hashset is 16. and the fill ration is 0.75 @ 75%.

* pgm:- package set programs;

```
import java.util.hash set;
public class demo
{
 psvm (String [] args)
{
HashSet s= new HashSet();
s.add(10);
s.add("durga");
s.add("null");
s.add(10);
s.add(10.5);
s.add(null);
s.o.p("size:"+s.size());
for(Object obj: c)
{
    s.o.p(obj);
} Uniqueness is maintained using equals() and hashcode()
} overridden
}
```

key	value
#code	10

Fill ratio = 75%
0.75

LinkedHash Set

* LinkedHash set was started from jdk 1.4
* it is present in java.util package
* LinkedHash set is similar to hash set but for one major difference. Like hash set maintains insertion order whereas hash set will not maintain insertion order

Iterators

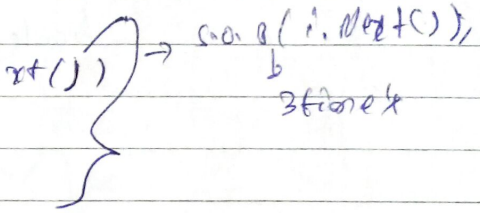
* Iterators is process of traversing the object one by one
* the important methods under Iterators are
① iterator():- this method is used to return iterable object address
② hasNext():- has next() method is used to check if there is an element present in the next position or not.

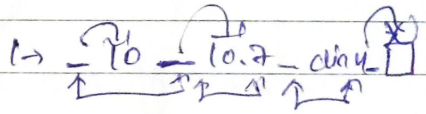
3) next():- next one by one.

pgm:- package
import java.util
import java.u
public class De
{
psvm(sring
}
LinkedHashset
l.add(10);
l.add(10.5);
l.add("durga
l.add(10);
Iterator i=
while
{
s.o.p(i.N
}
}

3) next():- next() method is used to return the data or object one by one.

prg1:- package set program;
import java.util.*; Iterator;
import java.util.LinkedHashSet;
public class Demo
{
psvm(String args[])
{
LinkedHashSet l = new LinkedHashSet();
l.add(10);
l.add(10.7);
l.add("diagg");
l.add(10);
Iterator i = l.iterator();
while (i.hasNext())
{
s.o.p(i.next());
}
}





Iterator itr = l.iterator();
while (itr.hasNext())
{
s.o.p(itr.next());
}

10
10.7
diagg