Avishek

8

* Inner classes (100 - 104)
* Serialization (125 - 131)
* Generics (173 - 177)
* Garbage Collection (178 - 180) Part-9 (2)

#100 Inner Class

05/May/25

* Inner Class
  → Sometime we declare class inside another class such type of classess are called innerClassess.
→ Came in 1.1 v. to fix GUI bugs.
→ Without existing one type of object if there is no chance of existing another type of object then we should go for inner classess.
→ University - department eg.
→ Car - Engine
→ Key - value
→ The relation blw Outer class & Inner class is not Is-A relation & it is Has-A relationship. (Composition / Aggregation)

* Based on Position of declaration & Behaviour. Division
1) Normal or Regular IC
2) Method Local IC
3) Anonymous IC
4) static nested classes.

* Normal (or) Regular I classess.

```
> class Outer {
        class Inner { }
  }
```

→ If we are declaring any named class directly inside a class without static modifier is called Normal (or) Regular Inner class

> Java Outer ↵ RE
Java Outer$Inner ↵ RE:

> class Outer {
    class Inner { }
    p s v m (S[] args) {
        Soln ("outer class"); }
}

Java Outer ↵ ✓
Java Outer$Inner ↵ RE.

> class outer {
    class Inner {
        p s v m (S[] args) {
            Soln ("Inner C"); }
    }
}

CE: Inner class cannot have static declaration

→ Inside inner class we can't declare any static members hence we can't declare main method & we can't run inner class directly from command prompt.

> class Outer {
    class Inner {
        public void m1() {
            Soln (" I.C.M"); }
    }
    p s v m (S[] args) {
        Outer o = new Outer();
        Outer.Inner i = o.new Inner();
        i.m1();
    }
}

→ Outer.Inner i = new outer().newInner();

C-1 Accessing inner class code from static area of outer class.

→ Above Example

C-2 Accessing inner class code from instance area of outer class

```
class Outer{
    class Inner{
        public void m1(){
            Sopln("ICM");}
    }
    public void m2(){
        Inner i = new Inner();
        i.m1();
    }
    p s v m (S[]args){
        Outer o = new Outer();
        o.m2();          }
}
```

C-3 Accessing inner class code from outside of outer class.

```
class Outer{
    class Inner{
        public void m1(){
            Sopln("ICM");}
    }
}
```

#101. Normal innerclass

05/May/23

```
> class Test {
      p s v m (S [] args) {
          Outer o = new Outer();
          Outer.Inner i = o. new Inner();
          i.m1();          }
  }
```

Accessing Inner class code

From Static Area of Outer.class (or) Outside of outer class

```
Outer o = new Outer();
Outer.Inner i = o.new Inner();
i.m1();
```

From Instance Area of outer class

```
Inner i = new Inner();
i.m1();
```

#101

```
> class Outer {
      int x=10;
      static int y = 20;
      class Inner {
          public void m1() {
              Sopln(x);
              Sopln(y);    }
      }
      p s v m(S[] a){
          new Outer().new Inner().m1();
      }
}
```

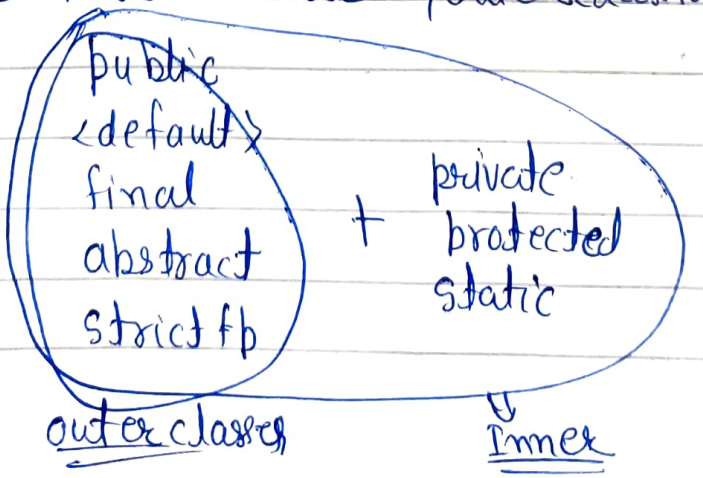
→ from normal inner class we can access both static or non-static member class directly.

```
Class Outer {
    int x = 10;
    class Inner {
        int x = 100;
        public void m1() {
            int x = 1000;
            Sop(x);          1000 ✓
            Sop(this.x);     100 ✓
            Sop(Outer.this.x); 10 ✓
        }
    }

    p s v main(S[] args) {
        new Outer().new Inner().m1();
    }
}
```

→ Within the inner class this always refers current inner class object. ~~gf~~

→ gf we want to refer current outer class ~~eta~~ object we have to use. {outerclassName.this}



* Nesting of Inner class

```
> class A {
      class B {
          class C {
              public void m1() {
                  Sopln("Inner most C"); } }
      }
  }
```

```
> class Test {
      p s v m(S[] a){
      A a = new A();
      A.B b = a.new B();
      A.B.C c = b.new C();
      c.m1();      }
  }
```

* Method local Inner classess.

→ Sometimes we declare a class inside method such type of inner classes are called method local Inner classess.

```
> class Test {
      public void m1() {
          public
```

50K

→ The main purpose of MLIC is to define method specific repeatedly required functionality.

→ MLIC is best suitable to meet nested method requirement.

→ We can access MLIC only within a method where we declare outside of method we can't access bcz of its less scope MLIC are rarely used.

```
class Test {
    public void m1() {
        class Inner {
            public void sum (int x;int y) {
                Sop1n("Sum" + (x+y)); }
        }
        Inner i = new Inner();
        i.sum(10,20)
        i.sum(100,200)
        i.sum(1000,3000)
    }
    p s v m(S[] a){
        Test t = new Test();
        t.m1();
    }
}
```

o/p

```
Sum 30
sum 300
sum 4000
```

→ We can declare MLIC both instance & static methods.

```
> class Test {
      int x = 10;
      Static int y = 20;        → CE: non-static var cannot be referenced from a static content
      public void m1() {
          class Inner {
              public void m2() {
                  Sop(x);
                  Sop(y);
              }
          }
          Inner i = new Inner();
          i.m2();
      }
      p s v m (S[] args) {
          Test t = new Test();
          t.m1();
      }
```

(static)

o/p
10
20

→ If we declare inner class inside instance method then from that method local class we can access both static & non-static member of outer class directly.

→ If we declare inner class inside static method then we can access only static member of outer class directly from m-MLIC.

*

```
> Class Test {
      public void m1() {
        int x = 10;  ---> final no CE.
        class Inner {
            public void m2() {
              Sop(x); }
        }
        Inner i = new Inner();
        i.m2();
      }
      p s v m(S[] a) {
        Test t = new Test();
        t.m1(); }
  }
```

CE:

→ from MLIC we can't access local variable of the methods in which we declare inner class.

→ If local var. is declared as final then we can access

→ If we declare 'x' as final then we will not get CE:

i ✓
J ✓
K ✗
m ✓

```
> Class Test {
      int i = 10;
      static int J = 20;
      public void m1() {
          int k = 30;
          final int m = 40;

      class Inner {
          public void m2() {
              Line ① }
```

#102 Anonymous inner class

05/May/23

* Anonymous Inner class
→ Sometime we declare inner class without name Such type of inner classes are called anonymous IC.
→ The main purpose of AIC is just for instant use (one time usage).
⇒ 3 Types → based on defination & beahivour.
① Anonymous Inner class that extends a class
② AIC that implements an inteface.
③ AIC that defined inside arguments.

* AIC that extends a class.

```
PopCorn p = new PopCorn();
PopCorn p = new PopCorn()
{
};
```

```
Thread t = new Thread();
Thread t = new Thread()
{
};
```

```
Runnable r = new Runnable()
{
};
```

```
> class PopCorn {
      public void taste() {
          Sopln("Salty"); }
      100 more methods.
  }
> class SubPopCorn extends PopCorn {
      public void taste() {
          Sopln("Spicy"); }
  }
> class Test {
      p s v m(S[]a){
          PopCorn p = new PopCorn() {
              public void taste() {
                  Sopln("Spicy"); }
          };
          p.taste(); // Spicy
          PopCorn p1 = new PopCorn();
          p1.taste(); // Salty
          PopCorn p2 = new PopCorn() {
              public void taste() {
                  Sopln("Sweet"); }
          };
          p2.taste(); //Sweet

          Sopln(p.getClass().getName())
          Sopln(p1.  "    "    )
          Sopln(p2.  "    "    )
```

PopCorn.class
Test.class
Test$1.class
Test$2.class

① Popcorn p = new Popcorn()
Just we creating popcorn object.

② Popcorn p = new Popcorn(){
{ ; ①
};

①→ we are declaring a class which extends popcorn without name (AIC).

②→ For that object we are creating an object with parent reference.

③ Popcorn p = new Popcorn()
{ public void taste(){
Sopln("spicy"); }
};

①→ we are declaring a class that extends popcorn without name (AIC)

②→ In the child class we are overriding taste() method.

③→ for that child class we are creating an object with parent reference.

* Defining a thread by Extending Thread class.

Normal C approach

```
>class MyT extends Thread {
    public void run() {
        for (int i=0; i<10; i++){
            Sopln("CT"); }
    }
}
```

AIC approach

```
>class ThreadDemo {
    p sv m (S[] a){
        Thread t = new Thread()
        {
            public void run()
```

```
class ThreadDemo {
    psvm(S[]a){
    MT t = new MT();
    t.start();
    for(int i=0;i<10;i++){
      Soplm("MT"); }
    }
}
```

```
    { for(int i=0;i<10;i++){
      Soplm("CT"); }
    }
};
t.start();
for(int i=0;i<10;i++){
    Soplm("MT") }
}
```

* AIC that implement an Interface.

⇒ defining a thread by implementing Runnable.

N.C. Approach

```
class MyRun implements Runnable
{ public void run(){
    for(int i=0;i<10;i++){
      Soplm("CT"); }
  }
}
class ThreadDemo {
    psvm(S[]a){
    MyRunable r = new MR();
    Thread t = new Thread(r);
    t.start();
    for(int i=0;i<10;i++){
      Soplm("MT"); }
    }
}
```

AIC App.

```
class ThreadDemo {
  psvm(S[]a){
  Runnable r = new Runnable
  { public void run(){
      for(int i=0;i<10;i++){
        Soplm("CT"); }
    }
  };
  Thread t = new Thread(r);
  t.start();
  for(int i=0;i<10;i++){
    Soplm("MT"); }
  }
}
```

* AIC that define inside argument.

```
> class ThreadDemo {
      p s v m (S [] args) {
      Thread t = new Thread ( new Runnable()
                              {
                                public void run() {
                                  for(int i=0;i<10;i++){
                                    Sop("CT"); }
                                }
                              }).start();
      for(i=0 ;i<10;i++){
        Sop("MT"); }
  }
}
```

#103 05/May/23

* Normal Java Vs Anonymous inner class.

1) NIJ can extend only one class at a time & AIC also can extend only one class at a time.
2) A NIJ can implement any no. of interfaces symaltiniously but AIC can implement only one interface at a time.
3) A NIJ class can extend a class & can implement any no. of interfaces symultiniously. but AIC can extend a class or can implement a interface but not simultaniously.
4) In NIJ class we can write any no. of constructors but in AIC we can't write any constructor explecitly

Note:- If requirement is standard & required several times then we should go for normal top level class.
If requirement is temporary & required only once (instant use) then we should go for anynomus inner class.

⇒ where AIC are best suitable.?

→ We can use AIC frequently in GUI based applications to implement Event handling.

* Static nested Classess (SNC)

→ Some we can declare inner class with static modifier, such type of inner classess is called static nested classess.

→ In the normal IC, without excising outer class object there is no chance of existing inner class object i.e inner class is strongly associated with outer class object.

→ But in case of SNC without existing outer class object there may be chance of existing nested class object hence SNC is not strongly associated with outer class object.

```
> class Outer {
      static class Nested {
          public void m1() {
              Sopln("SNC m");
          }
      }
  }
  psvm(S[]a){
      Nested n = new Nested(); > n.m1() }
```

→ If we want to create nested I Class object from outside of outer class then

> `Outer.Nested n = new Test.(Nested);`

→ In RIC we cant declare any static member.
→ In SNC we can declare static including main()
→ hence we can invoke SNC method directly from CMD prompt.

→

* Diff b/w NIC & SNC.

Normal Inner class	Static nested class
① Strongly associated	① Weakly associated
② We can't declare static mem.	② we can declare static mem.
③ We can't declare main method	③ we can declare main method
④ We can access both static & non-static members of outer class directly.	④ we can access only static members of of outer class.

#104 nested classes & Interfaces

05/May/23

* Various combination of nested classes & Interfaces

C-I Class inside class

Whole inner class sessions.

→ Without existing one type of object if there is no chance of existing another type of object then we can declare class inside a class.

→ University & Dept. Situtions

C-II Interface inside class

→
```
class VehicleTypes {
    interface vehicle {
        public int getNoOfWheels();
    }
    class Bus implements vehicle {
        public int getNoOfWheels() {
            return 6;
        }
    }
}
```

→ Inside a class, if we require multiple implementations of an interface & all these imple. are related to particular class then we can define a interface inside a class.

C-III Interface inside Interface

→ We can define interface inside interface.

→ Eg:- A Map is group of key-value pair & each key-value pair is called an entery. without existing map object there is no chance of existing entry object.

→ Entery interface is defined inside Map interface.

```
interface Outer {
    public void m1();
    interface Inner() {
        public void m2(); }
}
```

→ We can implement inner interface directly without implementing outer interface.

→ Similarly whenever we are implementing outer(I) we are not required to implement inner(I)

C-IV Class inside Interface

```
interface EmailServie {
    public void sendMain(EmailDetails e);
    class EmailDetails {
        str to_list;
        str cc_list;
        str subject;
        str body;
    }
}
```

→ If functionality of a class closely associated with interface then it is highly recommended to declare class inside interface.

→ To provide default implementation of interface we can use class inside interface.

→ The class which declared inside interface is public static by default.

* Conclusions :-

> class A {
    class B { }
}

> class A {
    interface B { }
}

> class A {
    class B { }
}

> Interface A {
    Interface B { }
}

> Interface A {
    Interface B { }
}

→ Amoung classes & interfaces we can declare anything inside anything.

→ Inner interface is by default public & static

→ ~~Inner~~ Inside interface
    ↳ Inner class is by default public & static.

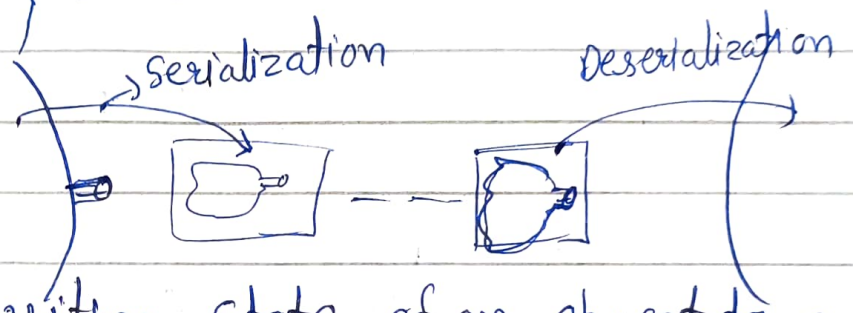
→ inner interface inside class is always static but need not be public.

#125 Serialization

06/May/23

① Introduction
② Object Graph in Serilization
③ Customized Serialization
④ Serialization wrt inheritance
⑤ Externalization
⑥ SerialVersionUID

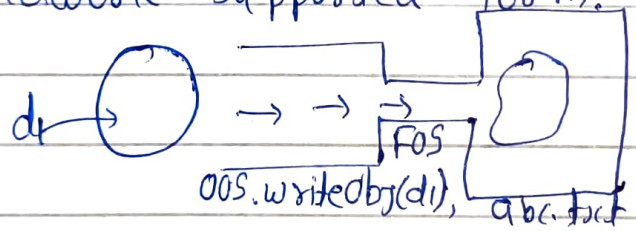
} Interview



* Introduction

* Serialization

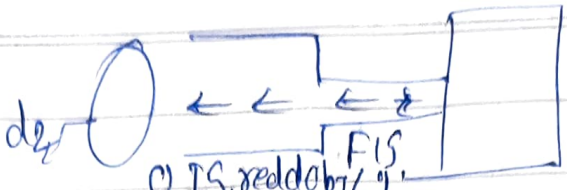
→ The process of writing state of an object to a file is called serialization but strictly speaking it is process of converting an object from java supported form into either file supported form or network supported form.



→ by using File Output Stream(FOS) & ObjectOutputStream classes we can achieve Serialization.

* Deserilazation

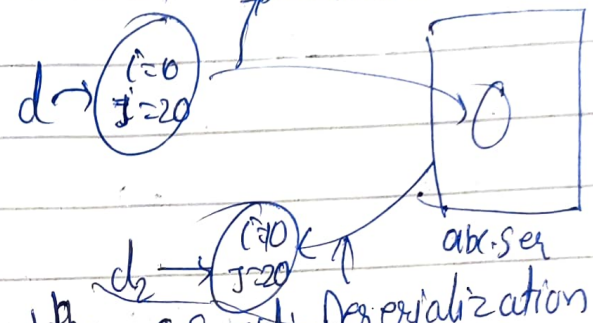
→ The process of reading state of an object from the file is called deserilazation. but strictly speaking it is the process of converting an object from either file supported form (or) Network supported form into java supported form.



→ By using FileInputStream(FIS) & ObjectInputStream classes we can implement deserialization.

> class Dog implements Serializable

```
    int i=10;
    int j=20;
}
```



> class serializeDemo {

```
    p s v m (S[] args) throws Exception
        Dog d1 = new Dog();
ser.    FOS fos = new FOS("abc.ser");
        OOS oos = new OOS(fos);
        oos.writeObject(d1);

Deser.  FIS fis = new FIS("abc.ser");
        OIS ois = new OIS(fis);
        Dog d2 = (Dog)ois.readObject();
        Sop ln (d2.i + d2.j);
    }
}
```

→ We can serialize only serializable objects.
→ An Object is said to be serializable iff the corresponding class implements Serializable (I) otherwise RE: NotSerializableException.
→ Serializable is a marker interface.

* transient

→ It is applicable only for variables.

→ At the time of serilazation If we don't want to save the value of a particular variable to meet security constraints then we should declare that variable as transient.

→ JVM ignores original value of transient variable & save default value to the file.

→ transient ⇒ not to searilaze.

* transient Vs Static

→ Static var. is not part of object state & hence it won't participate in serilazation.

→ So, [static transient] is of no use.

* final Vs transient

→ Final var. will be particaped in serilazation direct by the value hence declaring a final var. as transient there is no impact.

declaration	O/p
int i = 10; int j = 20;	10 -- 20
transient int I = 10; int J = 20	0 -- 20
transient static int i = 10 transient int J = 20	10 -- 0
transient int i = 10; transient final int J = 20;	0 --- 20
transient static int i = 10; transient final int J = 20;	10 --- 20.

#126 Serialization in the case of Object graphs

06/May/23

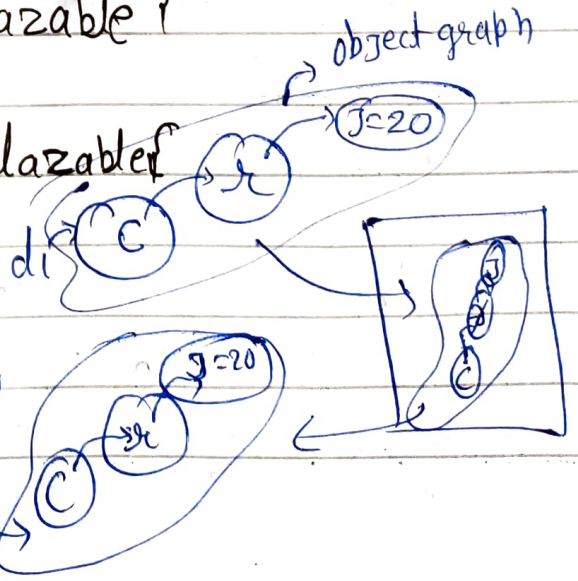
Note:- we can serialize any no. of objects to the file but in which order we serialize in the same order only we have to deserialize. i.e. order of object is important.

* If we don't know order of objects in serialazation

```
> FIS fis = new FIS("abc.ser");
  OIS ois = new OIS(fis);
  Object o = ois.readObject();
   if(o instanceof Dog){
        Dog d2 = (Dog) o;
   }else if(o instanceof Cat){
        Cat c2 = (Cat) o; }
   }else if(o instance of Rat){
   {  };
```

* Object graps in Serialazation.

```
> class Dog{ implements serialazable{
      Cat c= new Cat(); }
> class Cat{ implements serialazable{
      Rat r = new Rat(); }
> class Rat{ implements serialazable{
      int J = 20; }
> Dog d1 = new Dog();
  FOS fos = new FOS("abc.ser");
  OOS oos = new OOS(fos);
  oos.writeObject(d1);
```



#127.

```
FIS fis = new FIS("abc.ser");
OIS ois = new OIS(fis);
Dog d2 = (Dog) ois.readObject();
Sopln(d2.c.r.j); 20 ✓
```

→ Whenever we are serializing an object, the set of all objects which are reachable from the object will be serialize automically. This group of object is **object graph**.

→ In object graph every object should be serializable. if atleast one object is not serializable then we will get RE: NotSerializableException.

#127. Customized serialization

06-May-23

> Class Account implements Serializable{
>
> String username = "durga";
>
> transient String pwd = "anushka";
>
> }

```
Account a1 = new Account();
Sopln(a1.username+"---"+a1.pwd); durga --- anushka.
FOS fos = new FOS("abc.ser")
OOS oos = new OOS(fos);
oos.writeObject(a1);
FIS fis = new FIS("abc.ser");
OIS ois = new OIS(fis);
Account a2 = (Account) ois.readObject();
Sopln(a2.username+"---"+a2.pwd); # durga --- null.
```

→ During default serialization there may be chance of loss of information bcz of transient keyword.

→ In above example, we lost the information of pwd.

⇒ To overcome this issue we should go for Customized Serialization.

```
> class Account implements Serializable {
      String username = "durga";
      transient String pwd = "anushka";
  }
```

> private void writeObject(ObjectOutputStream os) throws Exception.

> private void readObject(ObjectInputStream is) throws E

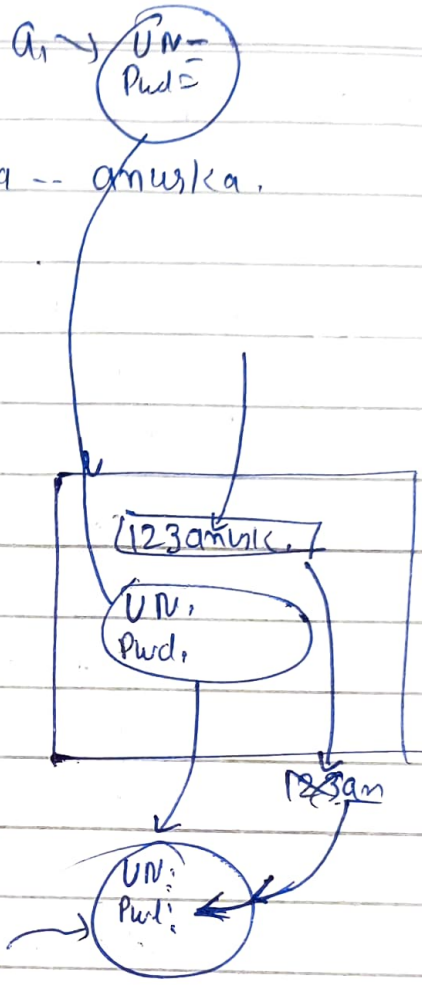
⇒ We can implement customized serialization by using above two methods.

① This method will be executed automatically at the time of serialization. hence at the time of serialization if we want to perform any activity we have to define that in this method only.

② This method will be executed automatically at the time of deserialization. hence at this time we can perform any activity.

> Account a1 = new Account();
Sop.ln(a1.Un + -- + a1.pwd);
FOS fos = new FOS("abc.ser");
OOb obs = new OOS(fos);
oos.writeObject(a1);

FIS fis = new FIS("abc.ser");
OIS ois = new OIS(fis);
~~ois.read~~
Account a2 = (Account) ois.readobj();
Sop.ln(a2.Un + -- + a2.pwd);
durga -- anu.



> Class Account imp. Ser. {
String username = "durga");
transient pstr. pwd = "anushka";
private void writeObject(OOS os) throws exc. {
os.defaultwriteObject();
String epwd = "123" + pwd;
os.writeobject(epwd);
}

private void readObject(OIS is) throws exc. {
is.defaultReadObject();
String epwd = (String) is.readObject();
pwd = epwd.substring(3);
}

}

#128 Customized Serialization

06 May 23

```
class Account imp. Ser. {
    String userName = "durga";
    transient Str. pwd = "anushka";
    transient Int pin = 1234;
    private void writeObject(OOS os) th. E {
        os.defaultWriteObject();
        Str epwd = "123" + pwd;
        os. writeObject(epwd);
        int epin = 4444 + pin;
        os. ~~writeObject~~
        os. writeInt(epin);
    }
    private void readObject(OIS. is) th. E {
        os.defaultReadObject();
        Str epwd = (Str) is.readObject();
            pwd = epwd.substring(3);
        int epin = (~~Int~~ is.readInt();
            pin = epin - 4444;
    }
    Account a1 = new Account();
    Sopln(___                    );
    FOS --
    OOS --
    oos.writeObject(a1);
    FIS --
    OIS ---
    Account a2 = (Account) oi. readObject();
```

#129 Inheritance serialization

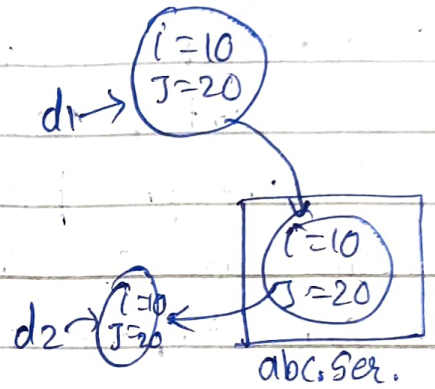
06 May '23

* Serialization wrt Inheritance

C-I

> class Animal implements Serializable {
    int i=10;
}

> class Dog extends Animal {
    int j=20;
}



> Dog d1 = new Dog();
FOS ---
OOS ---
oos.writeObject(d1);
FIS ---
OIS: ---
Dog d2 = (Dog) ois.readObject();
Sop(ln(d2.i + --- + d2.j); #10--20

→ Even though child class doesn't implement Serializable we can serialize child class object if parent class implement serializable(I).
i.e. Serializable nature is inheriting from parent to child.

Note:- Object class doesn't implement Serizable interface.

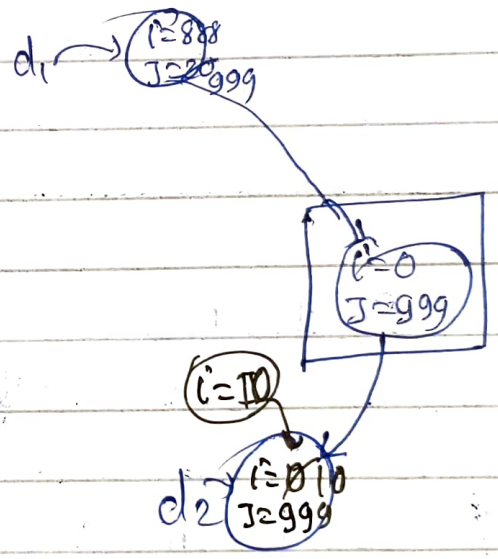
C-2

> class Animal {
    int i = 10; }

> class Dog extends Animal imp. Ser. {
    int J = 20; }

→ Eventhough parent class doesn't imp. serializable we can serialize child class object if child class implements Ser. (I).

> Dod d1 = new Dog();
    d1.i = 888;
    d1.J = 999;
    FOS --
    OOS --
    oos.writeObject(d1);
    Sop(m("Dec. started");
    FIS --
    OIS --
    Dog d2 = (Dog) ois.readObject();
    Sop(m(d2.i + --- + d2.J) 10 --- 999



→ At the time of serialization JVM will check is any var. inheriting from non-serializable parent or not, If any var. inheriting from non-Ser. parent then JVM ignores original value & save default value.

#130 Externalization

06|May|23

→ At the time of deserialization JVM will check is any parent class non-ser. or not,
If any parent C is non-ser. then JVM will execute instance control flow in every non-ser. parent & share it's instance var. to current obj.

→ While executing Instance control flow of non-ser parent JVM will always call no arg constructor hence every non-ser. class should compulsary contain no-arg constructor.

130

* Externalization

→ In seridlization everything takes care by JVM & programmer doesn't have any control.

→ In ser. it is always to save total object to the file & it is not possible to save part of the object which may create performance problems.

⇒ The main Adv. of Externalization is everything takes care by programmier & JVM doesn't have any control. Based on our requirement we can save either total object or part of the object, which improve performance.

> class Account imp. ~~Ser~~. Externailizable

* Methods

1) writeExternal()
2) readExternal()

Ser

→ Externalizable is child Interface of Ser.
→ Came in 1.1 V.

```
class ExtDemo imp. Externializable {
    String s;
    int i;   int j;
    ExtDemo(Str s, int i, int j) {
        this.s = s;
        this.i = i;
        this.j = j;
    public void writeExternal(Obj.Outer out) thr. Exc. {
        out.writeObject(s);
        out.writeInt(i);
    }
```

→ This method will be executed automatically at the time of ser.
→ within this method we have to write code to save required variables to the file.

```
public void readExternal(Obj.Input in) th. IE.
{   s = (String) in.readObject();
    i = in.readInt();
}
```

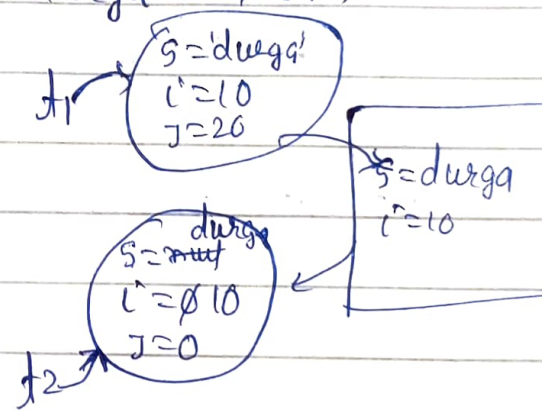
→ This method will be executed automatically at the time of deser.
→ within this method we have to write code to read required var of file & assign to current obj.

→ But strictly speaking at the time of deser. JVM will create seperate new object by executing public no-arg const., on that object JVM will call readExternal method.

→ Every Externalizable implemented class should contain public no-arg constructor.

```
> p s v m (S [] a) {
    ExtDemo t1 = new ExtDemo("durga", 10, 20);
    FOS fos = new FOS(abc.ser");
    OOS oos = new OOS(fos);
    oos.writeObject(t1);

    FIS fis = new FIS("abc.ser");
    OIS ois = new OIS(fis);
    ExtDemo t2 = (ExtDemo) ois.readObject();
    Sout(t2.s + "---" + t2.i + "---" + t2.j);
}
```



→ If the class imp. ser. then total obj. will saved to the file.

Note:→ In ser. transient keyword will play role but in External. it won't play any role.

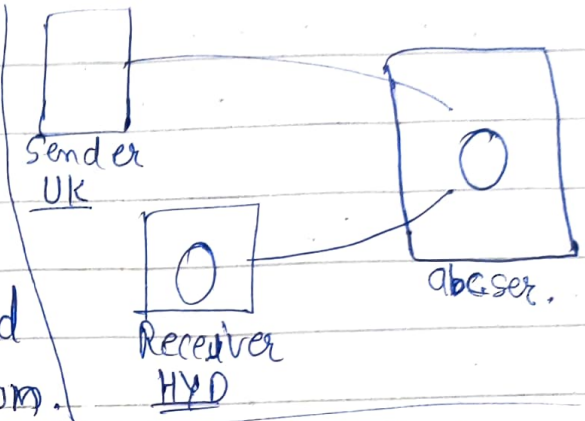
→ Ser	External
1) default ser.	1) Customized ser
2) Prog. ✗ JVM ✓	2) Prog ✓ JVM ✗
3) Total ✓ Part ✗	3) Total ✓ Part ✓
4) Perf. low	4) Per. high
5) Marker I ✓	5) Not marker I
6) no-arg not req	6) no-arg is required.
7) transient useful	7. Transient not-useful

#131 Serial Version UID

06/May/23

* Serial Version UID

→ In Serialization both Sender & reciever need not be same person, need not to use same machine & need not be from same location.



→ In ser. both sender & reciever should has .class file at the begining only. Just state of object is travelling from sender to reciever.

→ At the time of serialization with every object Sender side JVM will save a unique identifier

→ JVM is responsible to generate this unique identifier based on .class file.

→ At the time of de-serialization reciever side JVM will comare unique identifier associated with object with local class unique identifier.

→ If both are matched then only deserilization will be performed otherwise RE: InvalidClassExc.

⇒ This unique identifier is default SerialVersionUID.

* problems of depending on default SerialVersion UID.

→ Both Sender & reciever should use same JVM. vendor, platfor & version otherwise reciever unable to deserialize bcz of different SVUID.

→ Both Sender & reciever should use same .class version.

after serial. if there is any change in .class file at reciever side then reciever unable to de-serialize.

→ To generate serial VersionUID internally JVM use complex algorithm which may create performance problems

> class Dog {
→ private static final long serialVersionUID = 1L;
int i = 10;
int j = 10;
}

⇒ We can overcome problem by configuring our own serialVersion UID.

→ We can configure our own SVUID.

⇒ 3 class file

Dog Sender Reciever.

Note :- Some IDE prompt programmer to enter serialVersion UID explicitly.

→ Some IDE generate SVUID automitacally.

#173 Generics

11/May/23

① Introduction
② Generic classes
③ Bounded Types
④ Generic methods & wild-card character (?)
⑤ Communication with Non-Generic code
⑥ Conclusions

✱ Introduction

→ The main objective of generics are to provide type safety & to resolve type-casting problems.

Case-I : Type-Safety :

```
> String [] s = new String[10000];
  s[0] = "durga" ; ✓
  // s[1] = new Integer(10); ✗ → CE: IC types
  s[1] = "shiva" ; ✓
```

→ Arrays are type safe i.e we can give the gurantee for the type of elements present inside array.

```
> ArrayList l = new AL();
  l.add("durga"); ✓
  l.add(new Integer(10)); ✓

  String name1 = (String) l.get(0); ✓
  String name2 = (String) l.get(1); ✗ RE: CCE.
```

→ But collections are not type safe i.e we can't give the gurantee for the type of elements present in collection.

Case-II Type-Casting

⇒ String [] s = new String [10000];
s[0] = "durga"; → typecasting not required
String name1 = s[0];

⇒ ArrayList l = new AL();
l.add ("durga");
String name1 = l.get(0); → CE: Incompatible type.
String name1 = (String) l.get(0);
↳ Type-casting is mandatory.

→ In the case of arrays at the time of retrival it is not required to perform type-casting bcz there is a gurantee for the type of dements present inside array.

→ But in the case of collections at the time of retrival compulsary we should perform type-Castin bcz there is no gurantee for the type of elements present inside collection.

→ hence type casting is bigger headec in collections.

⇒ To overcome above problem of collections SUN people introduced Generics concept in 1.5 v.

→ Hence the main objectives of generics are
① To provide type-safety
② To resolve Type-casting problems.

#174 type safety || type casting

11/May/23

→ For example, to hold only String type of objects we can create Generic version of ArrayList as follows.

`ArrayList<String> l = new AL<String>();`

→ for this AL we can add only String type of object by mistake if we are trying to add any other type then we will get compile time error.

```
l.add("durga");  ✓
l.add(new Integer(10));  CE→X
l.add("shiva");  ✓
```

→ Hence through generics we are getting type-safety.

→ At the time of retrival we are not require to perform type-casting.

> `String name1 = l.get(0);`
>
> ↳ Type-casting not required.

→ hence through generics we can solve type-casting problem.

> AL l = new AL()	AL<String> l = new AL<Str>();
① Non-generic version	① Generic version
② Type-Safe X	② Type-Safe ✓
③ Type-casting is required.	③ Type-casting is not required.

~~* Conclusion~~ :~

* Conclusions :~> Parameter Type
Base Type ← AL<String> l = new AL<String>();

① Polymorphism is applicable only for the base type but not for parameter Type (Usages of parent reference to hold child object).

✓ List<String> l = new AL<String>();

✓ Collection<String> l = new AL<String>();

✗ AL<Object> l = new AL<String>();
→ CE: incompatible types.

② For the type parameter we can provide any class or Interface name but not primitive. if we are trying to provide primitive then we will get compile time error.

> AL<int> x = new AL<int>();
→ CE: Unexpected type.

* Generic Classes

1.4V >class AL {
add(object o)
object get(int index)
}

→ Untill 1.4 V a non-generic version of AL class is declared as follows

→ Then argument to add method is object & hence we can add any type of object to the arrlist due to this we are missing type safety.

→ The return type of get method is object hence at the time of retrival we have to perform type castin
→ But in 1.5V a generic version of AL class is declared as follows:-

1.5V

```
class AL<T> {          // T: Type Parameter
    add(T t)
    T get(int index)
}
```

⇓

```
class AL<String> {
    add<String s>
    String get(int index)
}
```

```
AL<Str> l = new AL<Str>();
l.add("durga");
l.add(new Integer(10));
CE: cannot find symbol
String name1 = l.get(0)
```

→ Based on our runtime requirement T will be replaced with our provided type.
→ For eg. to hold only string type of object a generic version of AL can be created as follows

→ In Generics we are associating a type parameter to the class such type of parameter raised classes are nothing but Generic classes @ Template classes.

```
> class Account<T> {
  }
```

> Account<Gold> a1 = new A<Gold>();
> Account<Platinum> a2 = new A<Platinum>();

→ Based on our requirement we can define our own generic classes also.

```
> class Gen<T> {
      T ob;                                   ob → (T)
      Gen(T ob) {
         this.ob = ob;
      }
      public void show() {
         Sop("Type ob:" + ob.getClass().getName());
      }
      public T getOb() {
         return ob;
      }
}
> class Test {
      p s v m(S[] a) {
        {Gen<String> g1 = new Gen<String>("durga");
         g1.show(); // Type ob: J.L.String
         Sopln(g1.getOb()); durga.
        {Gen<Integer> g2 = new Gen<Integer>(10);
         g2.show(); Type ob: J.L.Integer.
         Sopln(g2.getObj()); 10
      }
}
```

#175 type Safety || type Casting Excamples

11th May 23

* Bounded Types :-

```
> class Test <T extends Number> {
      public void m1() {
          T a, b;
          Sop(a+b);
          Sop(a*b);
          Sop(a/b);
      }
  }
```


```
class Test <T> {
}
```

Un-bounded (can pass any type)

→ We can bound the type parameter for a particular range by using extends keyword such types are called bounded types.

```
> class Test <T implements Runnable> {
  }
```

```
> class Test <T super String> {
  }
```

extends

: Syntax :-

```
> class Test < T extends X > {    }
```

→ X can be either class (or) Interface

→ If X is a class then as a type parameter we can pass either X type or it's child classes.

→ If X is an interface then as a type parameter we can pass either X type or it's implementation classes.

```
> class Test <T extends Number> { = }
✓ Test <Integer> t1 = new Test <Integer>();
✗ Test <String> t2 = new Test <String>();
        CE: Type parameter. not in it's bound.
```

> class Test<T extends Runnable>{ = }

> ✓ Test<Runnable> t = new Test<Runnable>();

> ✓ Test<Thread> t1 = new Test<Thread>();

> ✗ Test<Integer> t2 = new Test<Integer>();

↳ CE: not in it's bound.

→ We can define bounded types even in combination as,

> class Test<T extends Number & Runnable>{ }

→ As type para. we can take anything which should be child class of number and should implement Runnable interface.

✗ class Test<T extends Runnable & Comparable>

✓ class Test<T extends Number & Runnable & Comparable>

> ✗ class Test<T extends Runnable & Number>

→ we should tak class first int. next.

> ✗ class Test<T extends Number & Thread>

→ we can't extend more than one class simultaneously

Note :-

✓ class Test<T extends Number>{ }

✓ class Test<T extends Runnable>{ }

✗ class Test<T implements Runnable>{ }

✗ class Test<T super String>{ }

→ We can define bounded types only by using extends keyword. & we can't use implements & super keywords. but we can replace implements keyword purpose with extends.

#176 Generics method || Wildcard character (?)

11/May/23

Type Para

> class Test <T> { }
> class Test <X> { }

> class Test <Durga> { }

→ As a type-para, 'T' we can take any valid java identifier but it is convention to use T.

→

> class Test <A, B> { }
> class HashMap <K, V> { }

> class Test <X, Y, Z> { }
> HashMap <Integer, String> { }

→ Based on our requirement we can declare any no. of type parameters & all these para. should b seperated with ','

* Generics method and wildCard character.

① m1(AL<String> l)

→ We can call this method by passing AL of only str. type.
→ But within the method we can add only String type of the object to the list.
l.add("A"); ✓, l.add(null); ✓, l.add(10); X

② m1(AL<?> l)

→ We can call this method by passing AL of any type, ~~unknown~~
→ But within the method we can't add any thing to the list except null bcz we don't know the type exactly.
→ null is allowed bcz it is valid value for any type.
l.add(10.5) X l.add("A") X l.add(10); X
l.add(null); ✓

→ This type of method are best suitable for read only operation.

③ m1(AL<? extends X> l)

→ X can be either class or interface

→ If X is class then we can call this method by passing AL of either X type or it's child classes.

→ If X is an interface then we can call this method by passing arraylist of either X type or it's impl-class

→ But within the method we can't add anything to the list except null bcz we don't know type of X exactly.

→ This type of method also best suitable for read only oper

④ m1(AL<? super X> l

→ X can be either class or interface.

→ If X is a class then we can call this method by passing AL of either X type of it's super classes.

→ If X is an interface then we call this method by passing AL of either X type or super class of implementation class of X. Object, Runnable, Thread

→ But within the method we can add X type of obj. & null to the list.

l.add(X); , l.add(null);

42:00

#177 Generics method

10|May|23

→ AL<String> l = new AL<Str>();
→ AL<?> l = new AL<Str>();
→ AL<?> l = new AL<Inte>();
→ AL<?extends Num> l = new AL<Inte>();
✗ AL<?extends Num> l = new AL<Str>();
→ AL<?super String> l = new A<Object>();
→ AL<?> l = new AL<?>(); → CE: Unexpected type
AL<?> l = new AL<?extends Number>();
↳ CE:

✗ generic method
@ class level
→ class Test<T> {
we can use 'T' within this class based on our requirement }

@ method level
→ class Test {

public <T> void m1(T ob){
we can use 'T' anywhere within this method based on our requirement }
}

→ We can declare type para. either at class level or at method level.

→ we can define bounded type at method level also.

→ public <T> void m1()
<T extends Num>
<T extends Runnable>
<T extends Num & Runn>
<T extends Num & Comparable & Runn>
<T extends comp & Runn>

X <T extends Runn & Num>
X <T extends Num & Thread>

* Communication with non Generic Code

→ If we send generic object to non-generic area then it starts behaving like non-generic object. vise-versa.

→ i.e. the location in which object present based on that beaviour will be defined.

```
> Class Test {
      p s v m(S[] a){
          AL<String> l = new AL<Str>();
          l.add("durga");
          l.add("Ravi");
          // l.add(10); → CE
          m1(l);
          Sop(l); [dur, Ravi, 10, 10.5, tru]
          // l.add(10.5); → CE.
      }
```

Generic Area

```
p s v m1(AL l){
    l.add(10);
    l.add(10.5);
    l.add(true);
}
```

non-Generic Area

* Conclusions

→ The main purpose of generics is to provide type safety & to resolve type-casting problems.

→ Type safety & type-casing both are available at compile time. hence generic concept also ~~available~~ applicable only at compile time but not at run time.

→ At the time of compilation as last step generic syntax will be removed hence for the JVM generic syntax won't be available.

> AL l = new AL<String>();
l.add(10);
l.add(10.5);
l.add(true);
Sop ln(l) ; [10, 10.5, true]

> AL l = new AL<String>();
AL l = new AL<Integer>();
AL l = new AL<Double>();
AL l = new AL();
} => All are equal.

> AL<String> l = new AL<String>(); } equal.
> AL<String> l = new AL(); }

> class Test {
public void m1(int i) { => m1(int)
}
public int m1(int i) { m1(int)
return 10;
}
}

> class Test {
public void m1(AL<String> l) { } => m1(AL l)

public void m1(AL<Integer> l) { } => m1(AL l)
}. CE: name clash

At Compile time
① Compile code normally by considering Generic
② Remove Generic Syntax
* ③ Compile once again resultant code.

#178 Garbage Collection

11 May 23

① Introduction
② The ways to make an object eligible for GC.
③ The methods for requesting JVM to run GC.
④ Finalization.

* Introduction

→ In old languages (C++) programmer is responsible to create new object & to destroy useless objects. usually programmer taking very much care while creating objects & neglection destruction of useless objects bcz of his neglectance at certain point for creation of new object sufficient memory may not available (bcz total memory filled with useless obj. only) & total application will be down with memory problems hence outOfMemory Error is very common problem in old languages (C++).

→ But in Java Programmer is responsible only for creation of objects & Programmer is not responsible to destroy useless object. SUN people provided an assistant to destroy useless objects. This assistent is always running in background (Demon thread) & distroy useless object. Just bcz of this assistent chance of failing java program with memory problems is very-very low. this assistent is Garbage Collector.

→ hence the main objective of G.C. is to destroy useless object.

*179 The ways to make object eligible for GC

11/May/23

→ Even though programmer is not responsible to destroy useless objects it is highly recommended to make obj. available for GC. if it is no longer required.

→ A object is available for GC iff it doesn't contain any reference variable.

* Ways to make an object available to G.C

1. Nullifying the reference variable.

```
Student S1 = new S();
Student S2 = new S();
```

S1 → ○ S2 → ○ → No obj. elig. for GC

```
S1 = null;
```

→ one obj. elig for GC

```
S2 = null;
```

→ Two obj. eligible for G.C.

→ If an object no longer required then assign null to all it's ref. variables.

2. Reassigining the reference variable

```
Student S1 = new Student();
Student S2 = new S();
```

S1 → ○ S2 → ○ → no obj. ele. for GC

```
S1 = new S();
```

→ one obj. ele. for GC.

```
S1 = S2;
```

→ Two obj. ele. for GC.

→ If an object no longer required then re-assign its ref. var to some other object.

3. Objects created inside a method.

```
> class Test {
      p s v m(S[] args) {
          m1();  ----> Two obj ele. for GC
      }
      p s v m1() {
          Student s1 = new S();
          Student s2 = new S();
      }
  }
```

s1 ⇢ ○ s2 ⇢ ○

→ The objects which are created inside a method are by-default eligible for GC once method completes.

```
> class Test {
      p s v m(S[] a) {
          Student s = m1(); } ---> one obj. ele. for GC.
      p s Student m1() {
          Student s1 = new S();
          Student s2 = new S();
          return s1;
      }
```

s → s1 ⇢ ○ s2 ⇢ ○

```
> class Test {
      p s v m(S[] a) {
          m1(); } ---> Two obj. ele. for GC
      public static Student m1() {
          S s1 = new S();
          S s2 = new S();
          return s1;
      }
  }
```

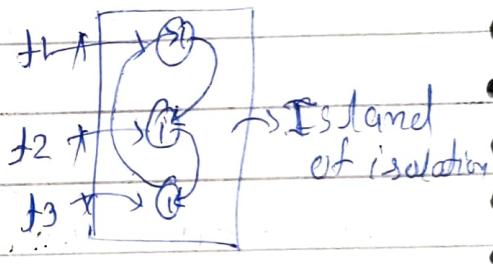
s1 ⇢ ○ s2 ⇢ ○

```
class Test {
    static Student s;
    p s v m(S[]a){
        m1();
    }
    p s v m1(){
        S = new S();
        S s1 = new S();
    }
}
```

← one obj. elig. for GC.

4. Island of Isolation

```
class Test {
    Test i;
    p s v m(S[]a){
        Test t1 = new Test();
        Test t2 = new T();
        Test t3 = new T();
        t1.i = t2;
        t2.i = t3;
        t3.i = t1;
        t1 = null
        t2 = null
        t3 = null
    }
}
```



No obj elig for gc

all 3 obj. elig for GC

Note:- → If an object doesn't contain any reference var then it is eligible for GC. always.

→ Even though obj. having references sometimes it is elegi for GC. (Island of isolation, all refs are internal)

180 The methods for requesting JVM to run GC

11 | May | 23

→ Once we made obj. elegible for GC. It may not be distroyed immediatly by GC. Whenever JVM runs GC then only the obj. will be distroyed. but when exactly JVM runs GC we can't expect it varied from JVM to JVM.

→ Instead of waiting untill JVM runs GC we can request JVM to run GC progamatically. but wether JVM accepts our request or not there is no gurantee.

→ 2 ways to request JVM.

① By using System class.

→ System class contains a static method .gc() for this purpose.

> `System.gc()`

② By using Runtime class

> `Runtime r = Runtime.getRuntime();`

① r.totalMemory()

② r.freeMemory()

③ r.gc();

→ Java app. can communicate with JVM by using Runtime object.

→ Runtime class present in java.lang pkg. & it is a Singlton class.

→ We can create runtime obj. by using Runtim. .getRuntime();

```
class RuntimeDemo {
    p s v m (S[] args) {
        Runtime r = Runtime.getRuntime();
        Sopln(r.totalMemory()); 100
        Sopln(r.freeMemory()); 60
        for (int i=1; i<=10000; i++) {
            Date d = new Date();
            d = null;
        }
        Sopln(r.freeMemory()); 40
        r.gc();
        Sopln(r.freeMemory()); } 30/40/50/60/80
}
```

① System.gc(); ✗ ② Runtime.gc();

✗ ③ (new Runtime).gc(); ④ Runtime.getRuntime().gc()

→ gc() method present in System class is a static method whereas gc() method present in Runtime class is instance method.

→
```
class System {
    p static void gc() {
        Runtime.getRuntime().gc();
    }
}
```

→ It is convinent to use System() class gc() method.

→ W.r.t. performance it is heighly recommended to use Runtime class gc() method. bcz System class gc() method internally call Runtime class gc() method.

#181 finalization

11/May/23

* Finalization

→ Just before destroying an object GC calls finalize() method to perform clean-up activities.

→ Once finalize() method complete automatically GC destroys the object.

→ finalize() present in Object class.

> protected void finalize() throws Throwable

→ We can override finalize() in our class to define our own clean-up activities.

Case-I

```
class Test {
    p s v m (S [] a){
        Test s = new Test();
        String s = new S("durga");
        s = null;
        System.gc();
        Sopln("End of main");
    }
    p void finalize() {
        Sopln("finalize method called");
    }
}
```

S → durga

{main → GC

End | final
final | End

→ Just before destroying an object GC calls finalize() method on the object whic is eligible for GC. then the corresponding class finalize() method will be executed. for eg. if String obj. eligible for GC then string class finalize method will be executed byt not Test class finalize method.

Case-II

```
class Test {
    p s v m (S[] a){
        Test t = new Test();
        t.finalize(); t.finalize()
        t = null;
        System.gc();
        Sopln("End of main"); }
    p void finalize() {
        Sopln("Finalize method"); }
}
```

o/p:
Final
Final
End
Final

→ Based on our requirement we can call finalize() method explicitly then it will be executed just like a normal method call & object wont be destroyed.

Note:- If we are calling finalize() method then it is like normal method call. & obj. won't be destroyed but if GC calls finalize() method then object will be destroyed.

Note:- (init(), service() & destroy() method are considered as lifecycle method of Servlet.

→ Just before destroying servlet object web container calls destroy method to perform clean-up activities ~~then~~ but based on our requr. we can call destroy() method from init() & service method then it will be executed like normal meth

#182 Finalization

11/May/23

Case-III

```
>Class FinalizeDemo {
    Static FinalizeDemo s;
    p s v m(S[] a)throws IE {
        FD f = new FD();
        Sopln(f.hashCode()); 100
        f = null;                          100
        System.gc();                       finalize
        Thread.sleep(5000);                100
        Sopln(s.hashCode()); 100           End
        s = null;
        System.gc();
        Thread.sleep(10000);
        Sopln("End of main"); }
    public void finalize() {
        Sopln("finalize method");
        s = this;                  }
}
```

→ Eventhough object eligible for GC multiple times but GC calls finalize() method only once.

Case-IV

```
Class Test {
    p s v m(S[] a){
    for (int i=0; i<10    ; i++){
        Test t = new Test();
        t.null);
    }                          }
}
```

```
static int count = 0;
public void finalize(){
    Sopln("Finalize m" + ++count);
}
```

→ We can't expect exact bevibour of GC. it is varied from JVM to JVM hence for following questions we can't provide exact answers

1) When exactly JVM runs GC?
2) In which order GC identifies eligible objects.
3) In which order GC destroyes eligible objs.
4) Wether GC destroyes all eligible objs or nat.
5) What is algarithm followed by GC. etc.

Note :-

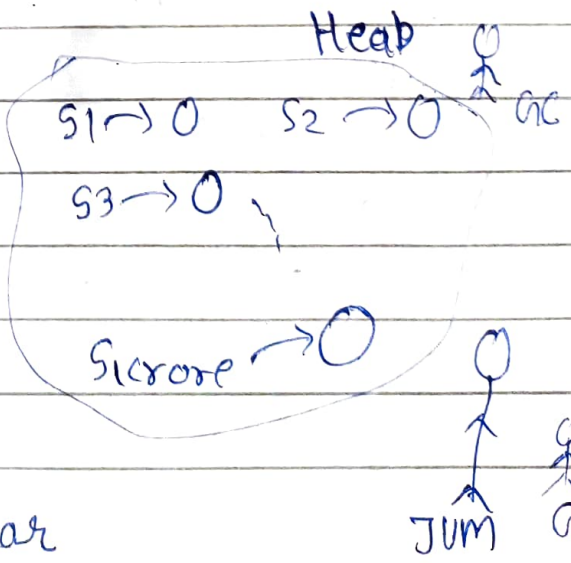
→ Whenever programes runs with low memory then JVM runs GC but we can't expect exactly atwhat time.

→ Most of the GC follow standard algo. mark & sweep algo. It does nat every GC follow the same algo.

Case - V

```
Student s1 = new S();
Student s2 = new S();
S
S sCrore = new S();
```

RE: Out Of Memory Error



* Memory leaks :-

The objects which are not used in our program & which are not eligible for GC such type of useless objects are called Memory leaks.

→ In our program if memory leaks present then the program will be terminated by raising OutOfMemory Error.

→ hence if object no longer required it is highly recommended to make that object eligible for GC.

* various third party memory management tools. to identify memory leaks.

HP OVO, HP J Meter, JProbe, Patrol, IBM Tivoli