Object oriented programming

- Introduction:

- Data and operators form a "class"

- But, class is abstract. We need objects to make any class functional.

- Theory, Java execution, multithreading, GUI and networks
  & Both console based, GUI awa socket based

- Hello world program:

```
public class Demo {
public static void main (String args [])
{
  System.out.println ("Hello World");
}
}
```

command line argument

Everything (including main) is inside a class.
↳ Java is purely object oriented.

- To run/compile:

① Save the file as filename.java

② javac Demo.java (compilation)
↓
on successful compilation, a class called Demo is created.

③ java Demo (execution).

On saving, file name is that of that "class" if keyword is public.
↳

So make sure execution is appropriately planned.

* Procedure-oriented programming: what to do

- List of instructions (in order)
- Eg. C, FORTRAN
- Divided into functions → which are accessible
- Data can be local/global
- Drawbacks:
  - Functions are reusable (only!)
    (In OOP, entire code itself is reusable)
  - Top down approach is used.

* Object-oriented programming: How to do

- Focus is on process rather than procedure
- Bottom top approach.
- Object: attribute + data } instance of class = obj
  operation
- Class is abstract → so instances are created for use in program

- Class:
  - user-defined data type → inside which there are primitive datatypes
    (≈ structure, but we get to add opr. as well)
  - Eg: class car
    {
    int price;
    string colour;
    string brand;
    } → member variable
    accelerate(); } → member functions
    break(); }
  - general format for defining object: car obj

- Member functions only have the access to member variables],
  Like, external functions can access member variables thro' member fns. {only}
  ↳ Data encapsulation

- Abstraction can be done using encapsulation
  ↳ hiding unnecessary details

- Inheritance: Deriving characteristics from a

"super" class / parent class → the attributes and methods basically.
(Both parent/grandparent)
Only public/protected variables are inherited (not private variables).
Types:
① Single - one class

② Hierarchical : A → B → C

③ Multiple : A → B → C
x ↗
} Not supported by Java

④ Multilevel : A → B
↓
C

- **Polymorphism**: one fn. will act in diff. forms.
Eg. operator overloading (fn. too)
↳ Not supported in Java.
can be runtime/compile time
Basically Java removed ambiguities in C++

- **Dynamic binding**: late binding; compilation during runtime

- **Message passing** ⇒ comm b/w objs.

Stages in development:
↓
Software development lifecycle:

Requirement → Analysis → Design → Implementation
(UML class/ object/activity/ sequential/ collaboration)
↓
coding
↓
testing

Note: common noun → class
proper noun → object
adjectives → attributes
verbs → methods/functions
} Techniques to identify

① "part of another class" → aggregation → A ✗, B survives
Eg:
class A:
{
int i; } primitive datatype
int j;

class B:
{
:

obj B; } complex datatype }

}

② composition: completely dependent on another class.
On deleting A, B dies as well.

③ Inheritance:

Eg:

Person { name, dob, ... }

/ \

Student Staff

{etc.} {etc.}

Analysis and Diagram:

Proper noun	common noun	Adj.	verbs
zonal head off.	Bank	savings	open - specification
acc.,	Branch	current	
loans	zone		loans
	Account		

classes:

Bank

Role → head, normal

customer

Account → curr.
→ savings

loans

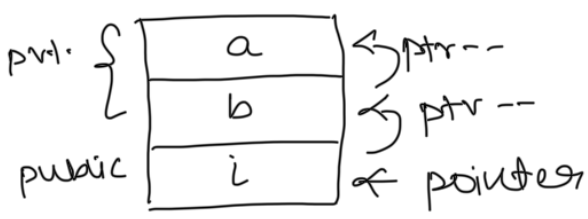
object oriented programming:

- Java is more of a correction / modification of C++
- Pointers are eliminated in JAVA
- satisfies all five major OOPS concepts.
- the byte code after compilation is platform-independent
- constructor → constructs the object and destructor → destroys objects

In C# objects must be destructed manually
In Java, there is automatic garbage collection

- Error prone features like multiple inheritance (in C++) is abdicated in Java. (Not directly atleast)

- Java is more secure as compared to C++
  ↓
  C++ allows pointers:



Private variables accessed thro' public variable's pointers.

- Similarly, Java also provides virus protection by sacrificing its JRE (Java Runtime Environment)

- "simultaneous" → but @ one instant of time, only one program will run.

- JDK - Java Development kit

Stack memory:

Stack memory : static

Eg.
```
main()
{
   arg();
   int arg1;
}

arg()
{
   int w;
}
```

Popped out once it is returned

Local variables

Stack overflow occure if program is not written properly.

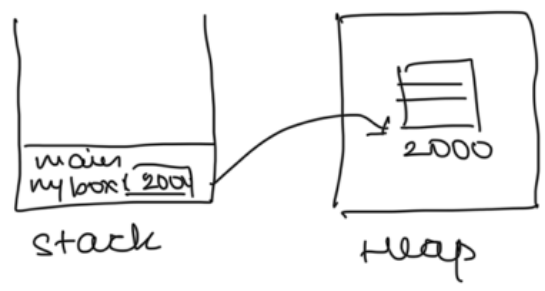
Heap memory: dynamic

Heap overflow occure if program is not written properly. → in C++ esp. when prog. is not destructed properly

Object creation in Java:

```
Box mybox1;
mybox1 = new Box;
```

Memory:



Stack → local to a particular fn.

Separate space to access global / static variables
↳ accessible to everything

Here, say we declare width as static:

```
/* A program that uses the Box class.

   Call this file BoxDemo.java
*/
class Box {
  double width;
  double height;
  double depth;
}

// This class declares an object of type Box.
class BoxDemo {
  public static void main(String args[]) {
    Box mybox = new Box();
    double vol;

    // assign values to mybox's instance variables
    mybox.width = 10;
    mybox.height = 20;
    mybox.depth = 15;

    // compute volume of box
    vol = mybox.width * mybox.height * mybox.depth;

    System.out.println("Volume is " + vol);
  }
}
```

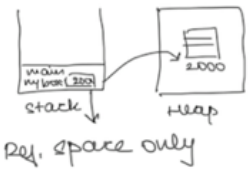
Ref. variable for Box

↳ Dynamic memory

If the two are stored as two class files,

`javac BoxDemo.java` will do as the compiler implicitly executes the other prog.

Memory:



Ref. space only

Here, irr. of no, only one w ⇒ last given value is taken

Static variable = Class variable (NOT specific to object as opposed to instance variables).

keywords:

* JRE itself cannot access private classes. Hence: "public" part of main.

* It has to be static so that JRE does not have to create a separate memory space/ alike each time. → class variable (not specific to obj, specific to class)
  ↓ other variables: obj vari / Instance vari

* void → return type

* String args[]
  ↳ variable name
  } CL Arguement. Stored in the format of string array.

* System.out.println
  ↳ method
  ↓ Inside lang package (comes by default)

  println is inside a predefined keyword: inside a class "output stream" → out is an instance of this class.

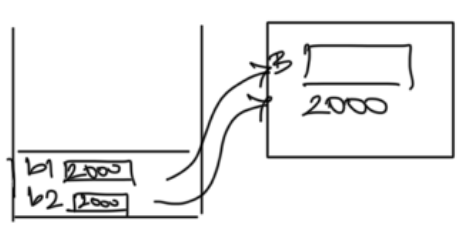
  "System" can refer to any computer/ printer/ etc. ↳ to designate that it is a system basically.

  Everything in Java is in a class!

Qn:

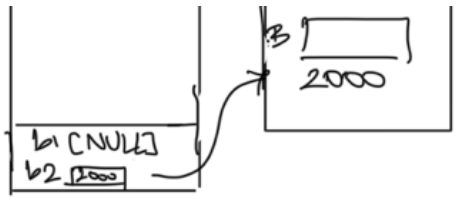
```
Box b1 = new Box;
Box b2 = b1;
```

one more copy will not be created.



(points to the same box)

Following this, if b1 = null,



Even if b2 = NULL, the object will still be in heap memory → automatic garbage collection will take care of it.

* Association:

(i) Simple / Regular

(ii) Aggregation
Class 'car' has an object of class 'Engine'

(iii) Composition / whole-part relationship
On del. one, the other gets deleted as well.
(whole) (part)

* Inheritance:

Inheriting class. ≠ Aggregation

* Dependence:

Execution of one class is dependent on another

Return my box → Address / ref. is rest.

Note:

demo.java → filename

```
class A
{
 }
```

javac Demo.java → compile
java A → run

But, if access specifier is "public", filename must be equal to class name.

Advised to have filename and classname same

Method signature → Gives the return type of

fn. and other details

↓

Ex. that method throws.

↓

Links different points of the code

Note:

this: my currently active instance.

Anonymous classes can be created in Java.

Class name → caps for every word.

Eg.

String, Float, String!

↓ ↓

Naming convention.

Method name: getName

Qn.: Write a java program to create class called "TrafficLight" with attr. for colour and duration and methods to change the colour and check for red/green.

```
public class TrafficLight

{
    public String colour;
    private int duration;

    public String change_clr (String colour)
    {
        this.colour = colour;
        System.out.println (colour)

    }

}

public static void main (String args[])

{
    TrafficLight obj = new Trafflight;

    System.out.println (Enter colour);
```

// get input from user and pass to fn.

}

Qn: write a program to create a class called employee

↓

with name, job title and salary attbr.
methods to calc. and update sal.

```
Public class Employee
{
    private string name;
    private string job;
    private double sal;

    public void get_sal (double sal)
    {
        this.sal = sal
    }
```

By default, java does 'pass by reference'.

one for each → stk. frame

Box b = new Box(); // Does the memory allocation

↓ Ready to store address of box variable

user def. complex datatype.

Similar to int *i in C where i stores address of integer

- Array in java:

int a[] = new int [10]

PASSING ARRAYS

```
class sortNumbers
{
    public static void main(String[] args)
    {
        int[] data={40,50,10,30,20,5};
        System.out.println("Unsorted List is :");
        display(data);
        sort(data);
        System.out.println("\nSorted List is :");
        display(data);
    }
    static void display(int num[])
    {
      for(int i=0; i<num.length;i++)
          System.out.print(num[i] + " ");
    }
    static void sort(int num[])
    {
      int i, j, temp;
      for(i=0; i<num.length-1;i++)
          {
              for(j=0; j<num.length-i-1;j++)
                  {
                      if(num[j]>num[j+1])
                          {
                              temp = num[j];
                              num[j] = num[j+1];
                              num[j+1] = temp;
                          }
                  }
          }
    }
}
```

common → Not in stk.

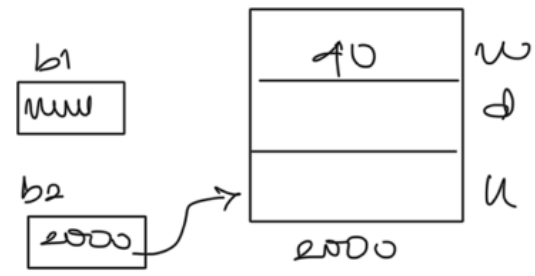
Box b1 = new Box()

Box b2 = b1

b1.width = 20

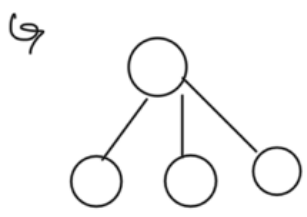
b2.width = 40

b1 = null

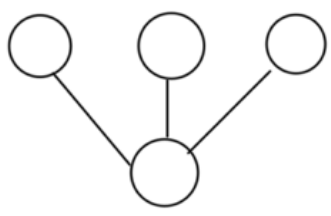


S.o.p. (b2.width) // Prints 40

One parent can inherit more than one child.



However, multiple inheritance can be achieved using interface.



using interface.

- super / this cannot be used inside 'static method'.

- super class ref. variable can be used to ref. subclass object.

- Aggregation: a class has the object of another class as its own attr. / member

  whole → one that is holding.

  when whole is del, part is also del → Composition
  part is not del → Aggregation.

(ref. to object)

Modifying code

```java
class Author
{
String authorName;
int age;
String place;
Author(String name,int age,String place)
{
 this.authorName=name;
 this.age=age;
 this.place=place;
}
public String getAuthorName()
{
 return authorName;
}
public int getAge()
{
 return age;
}
public String getPlace()
{
 return place;
}
}

class Book
{
String name;
int price;
Author auth;
Book(String n,int p,Author at)
{
 this.name=n;
 this.price=p;
 this.auth=at;
}
public void showDetail()
{
 System.out.println("Book is"+name);
 System.out.println("price "+price);
 System.out.println("Author is "+auth.getAuthorName());
}
}

class Test
{
 public static void main(String args[])
 {
  Author ath=new Author("Me",22,"India");
  Book b=new Book("Java",550,ath);
  b.showDetail();
 }
}
```

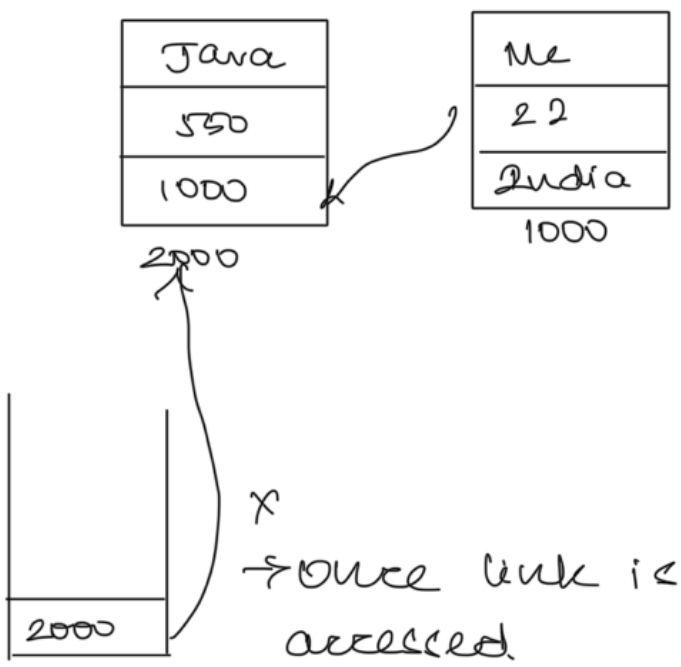
Two ways: change auth to NULL each time book is NULL.

Book (String name, int price, String authorName, int age, String place)
{

this. name = name;
this. price = price;
this. auth = new Author (authorName, age, place)
}

// Then author class is the same.

Before, loc in stack memory; now in heap:



→ once link is cut, 1000 cannot be accessed.

Method overloading is possible in parent-child as well.

[class is abstract; obj. is only real]

Abstract class has abstract method

↓

Method w/o body : only method signature.

↓ vcs

complex data type defn.

Local - no init; instance - init to 0

copy constructor ⇒ copy the constructor details.

Shallow - ref. for something existing already

deep copy - separate obj. one created.

== checks for address
.equals checks for values. } check!

' ' ⇒ treated as a character with ASCII value 32.

Anonymous object → no ref → so anon.

Null + string = null.

Built in string class has constructor overloading

↓

Accepts various constructors of strings.

'java api → api doc. of all classes in 'java.

Uneven arrays are possible.

Main can be overloaded ✓ But JRE accesses one with string only.

super: accesses super class variables of immediate parent.

import.util.Scanner

↓

Built-in package

Include the line ; package <name> ;
as the first one } code>

accessing: package.name.classname

compilation: javac /com/example/HelloWorld.java

Interpret: java com.example.HelloWorld

(04)

Javac -d <target_dir> <source file>

No modifier: package private.

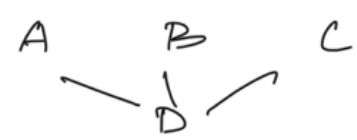
import mypack.*

No specific qn. on packages.

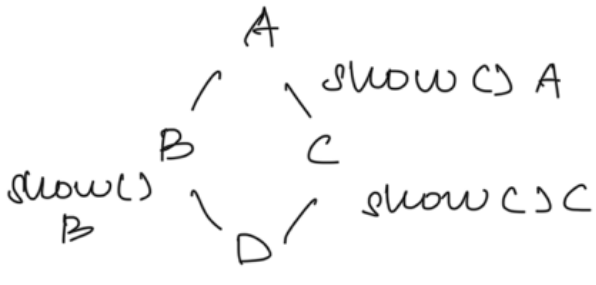
Abstract method ⇒ defines method signature

Interface facilitates multiple inheritance

A
↓
B Multilevel inheritance
↓
C



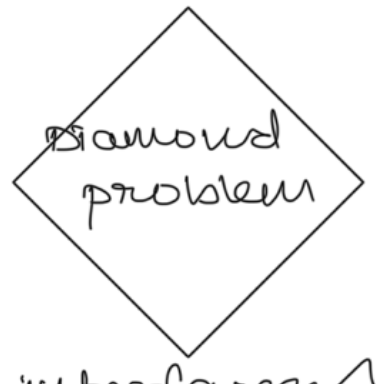
Multiple inheritance (Not possible dir. in Java)



D obj = new D()

obj.show()

Ambiguity (exists in C++)



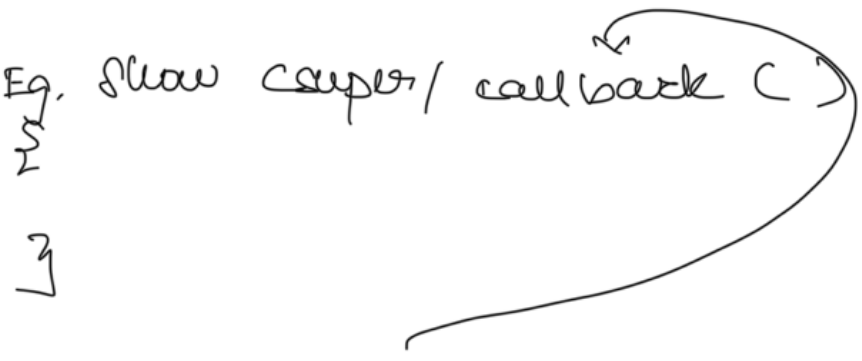
Inheritance from multiple classes & interfaces ✓

So, it now becomes D's responsibility.

Rewriting is compulsory (even for ones you do not (+add'n methods if need) needed.

otherwise, compilation occurs.

Eg. show super() callback ( )
{
}



obj.show (client)
obj1.show (emp)
obj2.show (stu).

First extends after implements.

Interface methods → pub, abs.

Default → package pvt.

superclass → looks for immediate parent.

Anonymous obj → no ref.

Pass by ref.

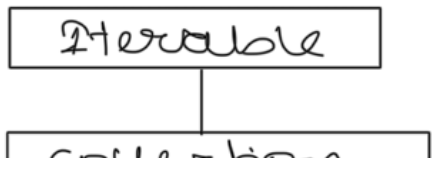
File permission class must be activated
↳ security related file op.

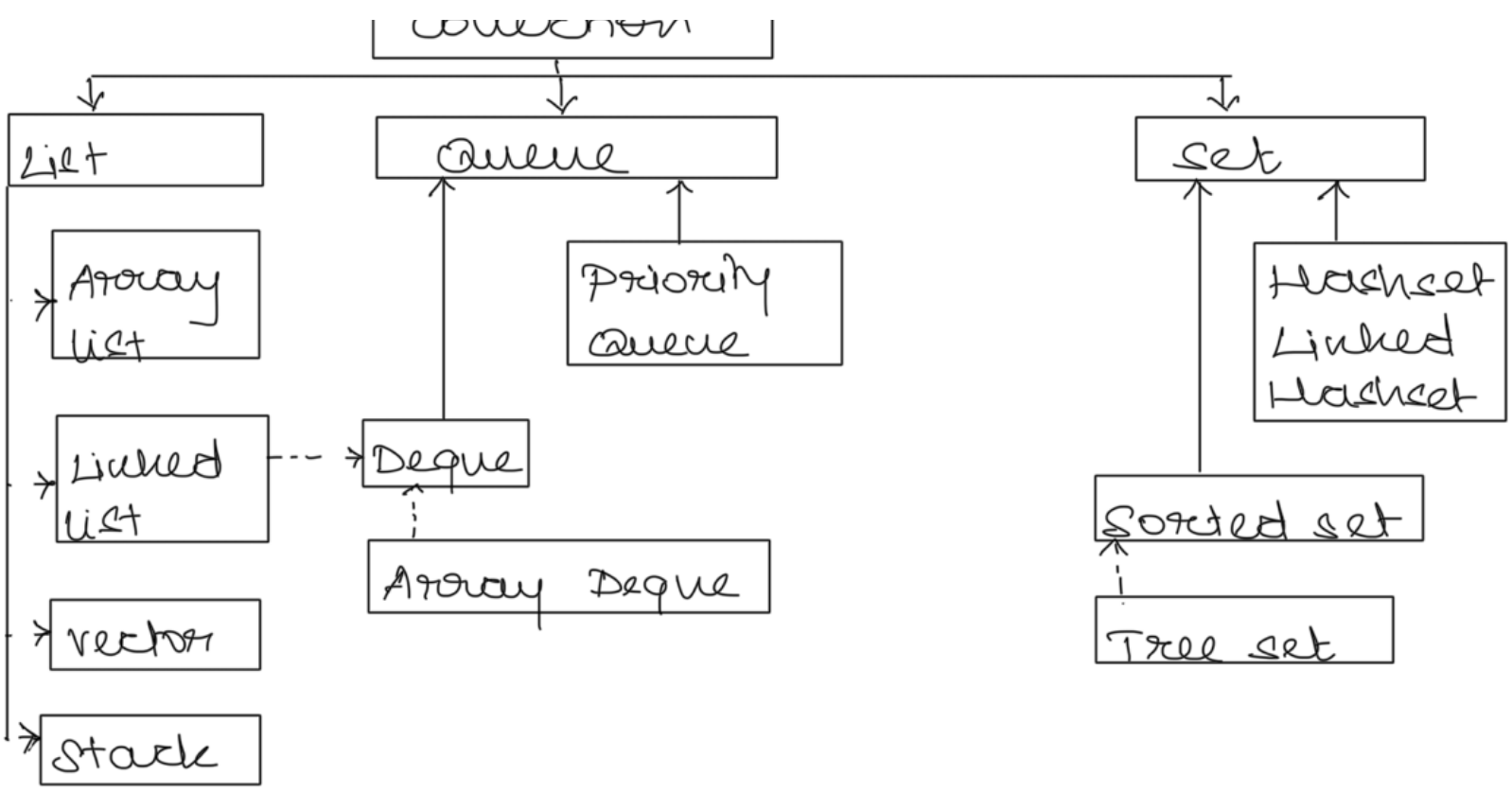
Collection framework: storing (programming without API).

Collections in Java:

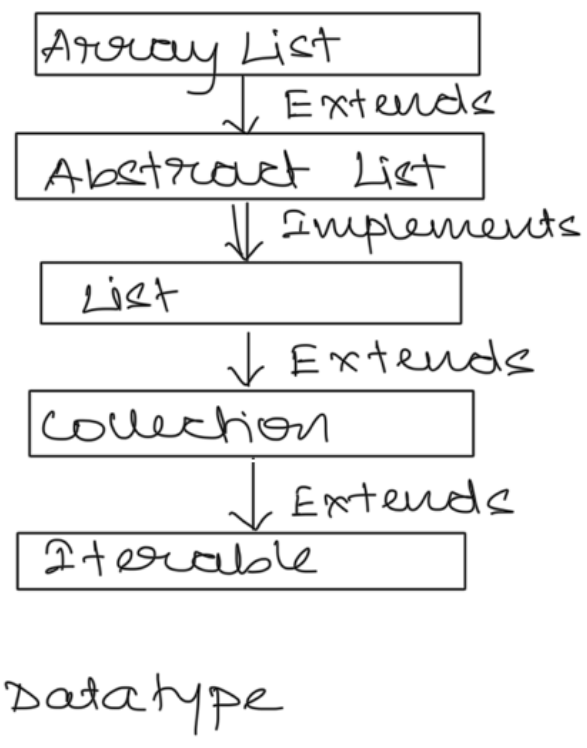
- storage and manipulation of data.
- Collections: single unit of objects. ↳ objects array received as
- Java collections has many interfaces: obj. type ↳ superclass
  Set, List, Queue, Deque

⁂ Hashtable, vector, stack, array, list and linked list





Representation Diagram



Generic class / Datatype
↓
Collection <? extends E > C

V → value; K → key; E → Elements in collection.

Note:
E: Type of element the linked list can hold.

Generic: Only elements of a particular type can be added to it; reducing the need for typecasting while retrieving elements.

Wrapper class and autoboxing
↳ create class and interface/methods.

Eg:

```
class Generic_Class <T> {
//variable of type T
private T data
```

```
public Generic_class {
    this.data = data;
}
```

```
Generic_Class<Integer> int Obj = new
Generic_Class<>(5)
```

An integer has been passed as input.

Wild cards in Generics:

- Allows flexibility while dealing with unknown types.

• Unbounded Wildcard	• Upper Bounded Wildcard	• Lower Bounded Wild Card
<?>	<? extends type>	<? super type>

public void pC (Collection<?> collection)
 	
for (object : item)
	public voird pN (List<? extends Number> list)	public void aN (List<? super Integer> list) {
		list.add(10);
		list.add(20); }

```
public class higher
{
  public void logEle (List<?> elements) {
      for (Object element : elements) {
      S.o.p. ("Logging:" + element);
    }
}
```

Iterators and hash methods: .:.

↳ List iterator is useful in traversing different elements.

↳ Methods in an iterator: hasNext(), next(), remove

Main collection	D	O	S	TS
ArrayList	✓	✓	x	x
LinkedList	✓	✓	x	x
Vector	✓	✓	x	✓

Hashset	✗	✗	✗	✗
Linked Hashset	✗	✓	✗	✗
Treeset	✗	✓	✓	✗
HashMap	✗	✗	✗	✗

Applets: GUI in Java & interface between logic and users.

<applet code = "filename">

component → container → panel → applet

Methods in applet : init(), start(), paint(), stop(), destroy()

↓

must be overridden based on their functionality.

awt → abstract window tool

g.drawString ("A First Applet", 50, 100)

↓

x and y

java api → graphics → drawing rectangle/circle/...

SetBackground / foreground → in superclasses.

Color ⇒ class

↳ Red: predefined (final) in colour class

Methods → camel case $(ff)_{16} = (255)_{10}$

repaint() → calls paint() method one more time

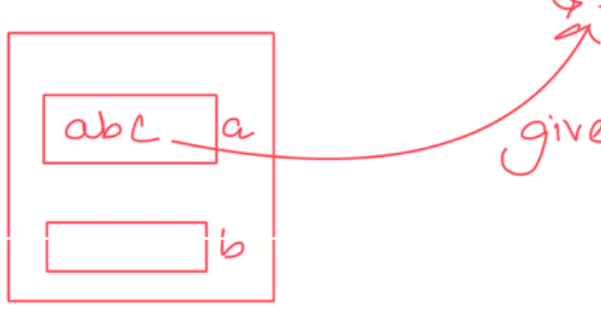
destroy() ⇒ for finalize.

showStatus() to display in status bar

<PARAM Tab>

↳ request.getparam (variable name)

↳ value of variable entered



Here param is used in the same applet

Integer.parseInt ⇒ str to int.

Event Handling :

Who generates the event will not handle - that's why

Delegation event model

Events — Click of a mouse / particular item

Event object — The source object will not handle
The Listener object will handle the event.

Event source → generates event

"Delegation event model" → source does not handle.

Add → to container Obj.

"Type" → must be added

After reg. stmt. → registration for event listener

class A implements "Listener"
↳ whichever listener interface
Eg. Mouse, key

→ Define methods
Eg. Mouse click method

"Action Listener" → interface
getActionCommand, getSource
...several constructors
here
→ Action performed — then here

| Steps: Implement → Define ALL the methods
→ Req. ones + Do nothing for others. |
|---|

without "b" → recognized thro' out applet
"this" → current event is recognized.

Mouse is moved → mouse entered is triggered

Only the applet area is checked for mouse

movement.

In case of applet, there are few restrictions
↳ "swing"

Form → logic to create object → write to file
(Front end)

Press → release → click.

Event source → Object
Add to container → Applet/ panel, ...
Register to corresponding typelist → addMouseListener (this)
Implement method in listener interface.
↳ Steps to be followed

Adapter class:

↳ To avoid the do-nothing functions.
↳ Use req. methods.

But it is a class → so extends applet and adapter class → not easy.

(Many interfaces can be implemented).

Anonymous class → for use in only one location
Obj. of inbuilt class freely.

'this' → recognizes all mouse-related activities
→ part. class → only that.

Swing → psvm → 'J'
↳ no life cycle.

Flow layout → same order → . set layout

Remove J for applet.

Create Event Source
Add Event Source
Register to Event Listener
Implement Methods.

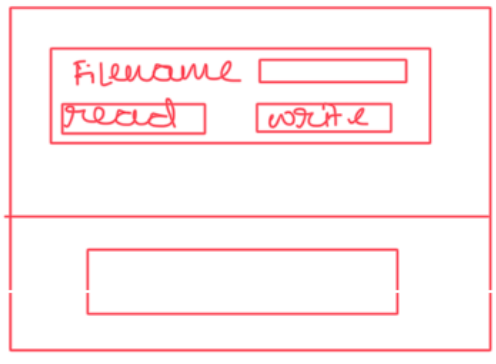
Anonymous class → (new WadapI())
↳ class defn. is given here

Layout for container classes.

// with this also.
↳ No adapter class here

Wrap: next line

"North" → north side of frame

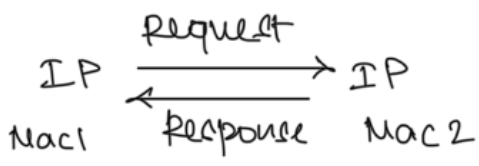


Source → button
command → on top of it

return byte → readline()

Network: more than one node is connected with each other

IP address → identify computer → unique
↳ to know who is being contacted



port number → identifies which runs in the machine
IP address → identifies which machine

Socket: End pt. of communication → where info can be read / written.

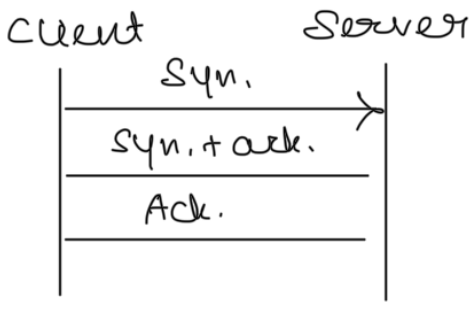
InetAddr : object with IP addr.
↳ has many methods to create.

New keyword is not used
↳ constructor ✗ } for Obj. creation
Method ✓ }
↳ in that class

∴ Method ⇒ factory method

factory methods are all static methods (converse is not true.)

Java Socket Programming



"Three-way handshake".

Loading...

running modified by any LOC → "volatile".