Java Swing: Java Swing is a part of Java used to create (GUIs), like buttons, text boxes, and windows, for desktop applications. It provides tools (called components) like:

JFrame: A main window for your application.

JButton: A button that can be clicked.

JTextField: A box for user input.

JPanel: A container to organize components.

Swing makes it easier to design user-friendly apps by letting developers add and arrange these components.

Ex: you can use Swing to create a calculator app with buttons and a display screen.

Swing is lightweight, flexible, and works on different platforms (windows, macOS, etc.)

Applet

→ Sp. type of prog — web pages embeded in — generate dynamic content
→ works inside browser — works at client side.

Adv: less response time
more secure
executed by browser running under many platforms
Linux, Windows, MacOS, etc

disad: - Plugin — req. — client browser to execute applet

Lifecycle: - Applet is initialized.
Started
Painted
stopped
destroyed

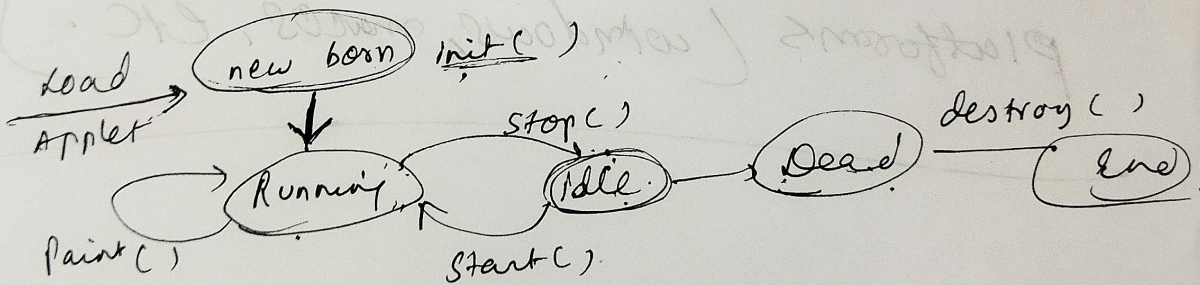
Lifecycle method:

Public void init( ) → initialize applet
public void start( ) → invoked after the init( ) method / browser is maximised.
→ start the applet.

public void paint (graphics g) → pain the applet.
Provides graphics class object — used for drawing oval, rectangle, arc, etc.

public void stop( ) → stop the applet
→ applet is stopped / browser is minimised.

public void destroy( ) → used to destroy the applet.



Types → Local Applet → created locally in local m/s.
→ can be easily accessed without any network connection.

Remote Applet → designed & developed by another developer located on remote computer that is connected to the internet

Local
no URL req.
available on comp.
no internet connec.
written on our own
no need to download

Remote
URL req.
not a.v.
ext. connection
written by another developer.
need to download.

Method Table :-> dispatch table.

=> Internally Java was a method
=> manage method call
=> each class in Java has method table & contains entries for each method declar". in that class.
=> Object - created, a ref: to the method table of its actual class is stored in object's header.

Dynamic Method dispath ->

-> method to be invoked is determined at runtime.
-> based on actual type not reference type.
-> flexibility in methods invoked
-> supporting polymorphism.

Association

- Relationship -> where one obj. can be connected to another.
- Object - independent of each other.
  (exist without each other)
- Teacher teaches Student - but - can exist independently.
- 'Just connected' not dependent

Aggregation

Relationship - "has-a" relationship
one class is part of another
Part (child) can exist independent of whole (Parent)
Strong (whole - Part relationship)
Library & Books.
(list, set)

GUIs → user interface
→ allows user to interact with a comp. sw using graphical elements such as button, windows, icons, and menus instead of text-based commands.

→ Graphical Elements
- Windows → main containers for applic°. interface
- Button → for user interac°. like submitting / cancelling actions.
- Text fields → input areas for users to enter text.
- Labels → Display static text / image.
- Menus → Provide navigation / options.
- ~~Labels~~ Sliders → allow selec°. of values within a range.
- Dialogs → Pop-up windows for specific tasks / message.

→ Events & Event Handling.

GUI Libraries & Frameworks.
→ Java Swing → Lightweight GUI toolkits
→ Java FX
→ Tkinter (Python)
→ PyQt / PySide (Python)
→ HTML/CSS/JS
→ C# Windows forms. & WPF (Windows Presentation Foundation)

* user-friendly
* interaction
* visual appeal
* efficiency

disadv. → resource intensive
complexity
platform dependences.

AWT → Graphical User Interface (GUI)
→ user interactions.
→ Abstract window toolkit
→ collections of components & event handling features for creating GUIs.

Event listener → object-detect, respond-events.
like button clicks,
Mouse movt.
Keyboard input.
→ java.awt.event package.
→ receive notifn. when mouse is moved/dragged.

Virtual Method Table → Polymorphism
↓ Java VMT term-don't term- → Dynamic method dispatch.
→ Method Table & runtime polymorphism.

Collections → framework that provides a set of classes & interfaces to store & manipulate a group of object.

→ Part of `java.util package` & are widely used for handling dynamic data str. like lists, sets, maps.

→ JCF (Java Collection Framework) simplifies programming by providing ready-to-use data str. & algo.

→ **Collection Interface** :-
- Root interface in collection hierarchy.
- Subinterfaces :- List, Set, Queue, etc.

Collection Framework components :-
- Interfaces :- Define behaviour of a collection (List, Set, Queue, Map).
- Classes - Implements these interfaces (eg. ArrayList, HashSet, HashMap)
- Algorithm - Static utility methods in Collections class for tasks like sorting, searching, etc.

List - Ordered collection.
- duplicates are allowed
- Implementations → ArrayList, LinkedList, Vector

Set - no duplicate elements are allowed
- Implementations: HashSet, LinkedHashSet, TreeSet

Queue - FIFO
- Implementations: PriorityQueue, LinkedList

Map
HashMap → unordered
LinkedHashMap → maintains insertion order.
TreeMap → sorted order of keys.

Common Operations
→ Add elements: add(), put()
→ Remove Elements: remove(), clear()
→ Access Elements: get(), iterator()
→ Check existence: contains(), containsKey()
→ Size of collection: size()

Tree Map Set List

Generics type & collections GUIs

→ Generics allow you to create classes, interfaces & methods that operate on types specified by the user.
This means you can write code that works with any data type without sacrificing type safety.

Why Generics?

a) Code Reusability - use same code for diff. data types.
b) Type safety - catch errors at compile-time instead of runtime.
c) Avoid type casting - no need to manually convert data types.

Without generics

```
List list = new ArrayList();
list.add("Hello");
String s = (String) list.get(0); // needs casting.
```

With generics

```
List<String> list = new ArrayList<>();
list.add("Hello");
String s = list.get(0); // No casting needed.
```

→ Declaring a Generic type :-
declared with a type parameter (eg. <T>).
T stands for Type, it can be any letter E, K, V.

→ `class GenericClass<T>{ }`

→ Multiple type parameters → multiple generic type parameters in a single class/method

`class Pair<K, V>{ }`

→ Bounded type parameters - you can restrict the types that can be used with a generic type by using extends.

`class Bounded<T extends Number>{ }`

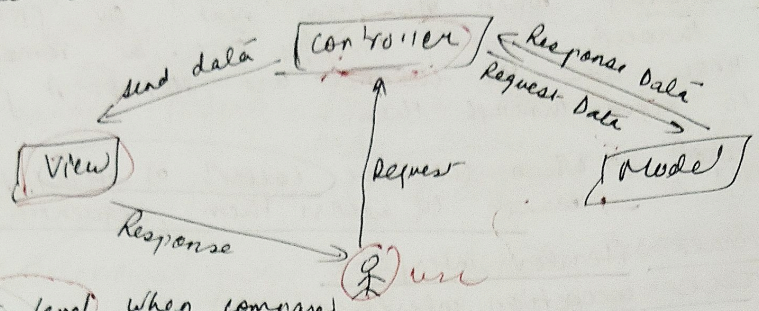
<T>

class GenericClass<T>{ }

class Pair<

* Model - View - controller Pattern

- primarily used to separate applic into (3) main comp. Model, View and controller.



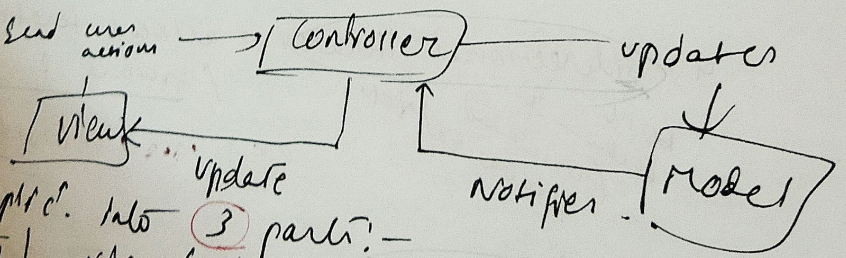
Model → lowest level when compared to view & controller.
→ Represents data to user & defines the storage of all the applic's data, objects.
→ Applications logic

View → majorly associated with User Interface (UI) & used to provide the visual represent. of MVC model.
→ deals with displaying the actual output to user.
→ handles communic' b/w user (input, request) & the controller.

Controller → takes care of the request handler.
→ completes the cycle of taking the user o/p & converting it into desired messages.

Benefits of MVC: → Organizes large size web applic'.
→ easy modifiable
→ easy planning & maintenance.
→ support TDD (Test Driven Development)

MVC Design Pattern:



- MVC breaks applic' into (3) parts:- Model (handles data) View (which is what user see) & controller (which connects the two).
Makes it easier to work on each part separately.
So you can update / find things without messing up the whole app.

Iterator Pattern

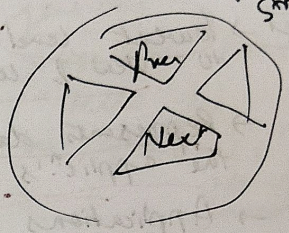
→ used to access elements of a collection (list, array, set) one by one without exposing the underlying str.

→ (ex remote control) - when you press "next" or "previous" buttons, you move through channels one by one. The remote doesn't need to know how the channels are stored - it just provides a way to move through them.

→ when to use? → when we have collection of items & want to access them sequentially.

→ code → (1) create iterator interface
(2) create collection interface
(3) create a class for the collection
(4) use the iterator

(forEach() Stream API)



→ Benefits → simplifies navigation
→ hides details
→ reusability.

→ Variations → 1) External iterator. (When client needs full control over iteration) — managed externally by client using the iterator explicitly.
- use Java's Iterator interface from java.util package.

Ext Iterator.

2) Internal iterator (When you want iteration process to be simpler & abstracted away) — handled internally by collection.
ex - Java's forEach method or Stream API.

3) Reuse iterator — iterates through collection in reverse order.
- When you need to process element in reverse order.

4) Bidirectional Iterator (both backward & forward traversal req.) — allows iteration both forward & backward.
ex - Java's ListIterator interface. (ListIterator)

5) Null Iterator - represents an empty collection
- Avoids null logic.

Design Pattern

(1)

- Sol^n to common s/w design pattern
- "Cheat codes" for solving common coding prblms.
- instead of figuring out everything from scratch, you can use these pre-defined sol^n.
- Types:
  - Creational Pattern – focus on how to create objects in a smart way
  - Structural pattern – help organize & connect objects / classes to make a syst. work well
  - Behavioural pattern – how objects talk to each other & share responsibilities

ex:- Singleton Pattern (Creational)
- makes sure only one object of a class is created.
- everyday ex:- A company has only one CEO.

ex:- Adaptor Pattern (Structural)
- Acts like a translator b/w 2 things that can't handle directly work together.
- A power adapter helps you plug a foreign device into your local socket.

ex:- Observer Pattern (Behavioural)
- notifies all connected parts when something changes.
- A Youtube channel notifies subscribers when a new video is uploaded.

adv →
- Saves time → don't solve same pattern again & again
- makes code clean → easy to understand & work with
- handles complexity → Good for big projects

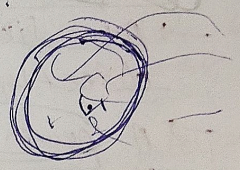
* we can't do implement() in abstract for that we create child class & do the implement() there. In abstract, object creation not possible. So, after creating child class, create object there.

ex - abstract class Test

```
{
    abstract void m1();
                  m2();
                  m3();
    void m4(){SOP("m4 method");}
}
class Test extends Test
{
    void m1(){SOP("m1 method");}
    void m2(){SOP("m2 method");}
    void m3(){SOP("m3 method");}
    psvm(S[] args)
    {
        Test t = new Test();
        t.m1();
        t.m2();
        t.m3();
        t.m4();
    }
}
```

Output

m1 method
m2 method
m3 method
m4 method



* Concrete State Space :- complete list of all possible conditions (States) a system can be in and the actions or event (transitions) that can move the system from one state to another. Shows how system works step-by-step in a clear & detailed way.

Advantages / dis

→ Overhead → implementing ADT can add overhead in term of memory & processing which can affect performance.

→ complexity → ADTs can be complex to implement

→ learning curve → using ADTs req. knowledge of their implem". & effort to & usage which can take time & effort to learn.

→ cost → ADTs may req, addn". resource & investment, which can ↑ cost of development.

• Concrete invariant → Concrete invariant in Java refers to a rule/condn". that must always be T for an object of a class, no matter what operation are performed on it.
It ensures the objects consistency & correctness.

Key points
1) Always true condn". → Concrete invariant is a condn". that is always true for a class objects like "a bank account balance should never be negative."
2) defines class Rules
3) checked in methods → set rules what makes an object "valid".
4) private variables → constructor, setters.
5) improve reliability.

Invariant :- The balance must always be greater than or equal to 0.

Concrete value → map. → abst. space

Abstracn". funcn" :- is a comment that maps concrete value to abstract space, describes loss of informn" that occurs when abstract data types are represented by concrete values.

Abstraction - Process of highlighting set of services & hiding the implementation. eg. Bank ATM.
By using abstract class & interfaces, we achieve abstraction.

In Java :-

Normal class
Contains method decl" & implement".

```
void m1()
{
  // logics here
}
```

ex:- class Test
{
  void m1(){}
  void m2(){}
  void m3(){}
}

all normal method are present.
→ possible to create object

Abstract class
only method declar". (no) implemn".
must end with semi-colon ;
To represent abstn". method use abstract modifier.

```
Abstract void m1();
```

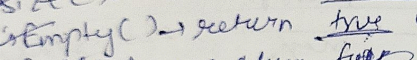
ex:- Abstract class Test
{
  void m1(){}
  void m2(){}
  Abstract void m3();
}

if a class contains atleast one abstract method.
→ Object creation not possible.

ADTs → List ADT
Specific. → Stack ADT
→ Queue ADT.

List ADT
get( ) → return an element from list at any given pos^n.
insert( ) → insert an element at any pos^n in the list.
remove( ) → remove the 1st occurrence of any element from non-empty list.
removeAt( ) → Remove the element at specified location from non-empty list.
replace( ) → replace an element at any pos^n with another element.
Size( ) → return the no. of element in the list.
isEmpty( ) → return true if the list is empty; otherwise return false.
isFull( ) → return true if list is full, otherwise false.

Stack ADT 
. Order of insertion & deletion should be acc. to LIFO or FILO principle.
. elements are inserted & removed from same end called top of the stack.
push( ) → insert element at one end of stack called top of the stack.
pop( ) → remove & return the element at top of stack if it is non-empty.
peek( ) → return the element at top of stack without removing it.
size( ) → return the no. of elements in the stack
isEmpty( ) → return true if stack is empty; otherwise false.
isFull( ) → return full " " " full, " false.

Queue ADT.


. Queue ADT follows a design similar to stack ADT
. Order of Insert^n & deletion change to FIFO.
. Elements inserted at one end (rear) & removed from other end (front).

Operations:
enqueue( ) - insert an element at end of queue.
dequeue( ) - remove & return the 1st element of the queue, if queue is non-empty.
peek( ) - return the element of the queue without removing it, if queue is not empty.
size( ) → return the no. of elements in queue.
isempty( ) - return T if queue is empty; Otherwise Return false.

Features of ADT:-
ADTs are a way of encapsulating data & operations on that data into a single unit.

Key features (Adv-)
1) Abstraction → ADT allow users to work with DS without having to know the implement^n details
2) Encapsulation → encapsulate data & operat^n into single unit, making it easier to manage & modify DS.
3) Data Abstraction
4) Data str. independence → ADTs can be implemented using diff. DS, which can make it easier to adapt to changing needs & requirement.
5) modularity → ADT can be combined with other ADTs to form more complex DS which can ↑ flexibility & modularity in prog.
6) Info. hiding → ADT can protect the integrity of data by controlling access & preventing unauthorized modific^n.

① Procedular lang:

- Follows a procedural / step by step approach.
- focus on functions & procedure.
- Data & functions are separate.
- Less secure - data is globally accessible.
- Limited Reusability
- C, fortan
- Less efficient - handling complex prblm.
- Linear / hierarchical str.
- Inheritance - not supported
- function overloading - not supported.

```
void main() {
    printf("Hello");
}
```

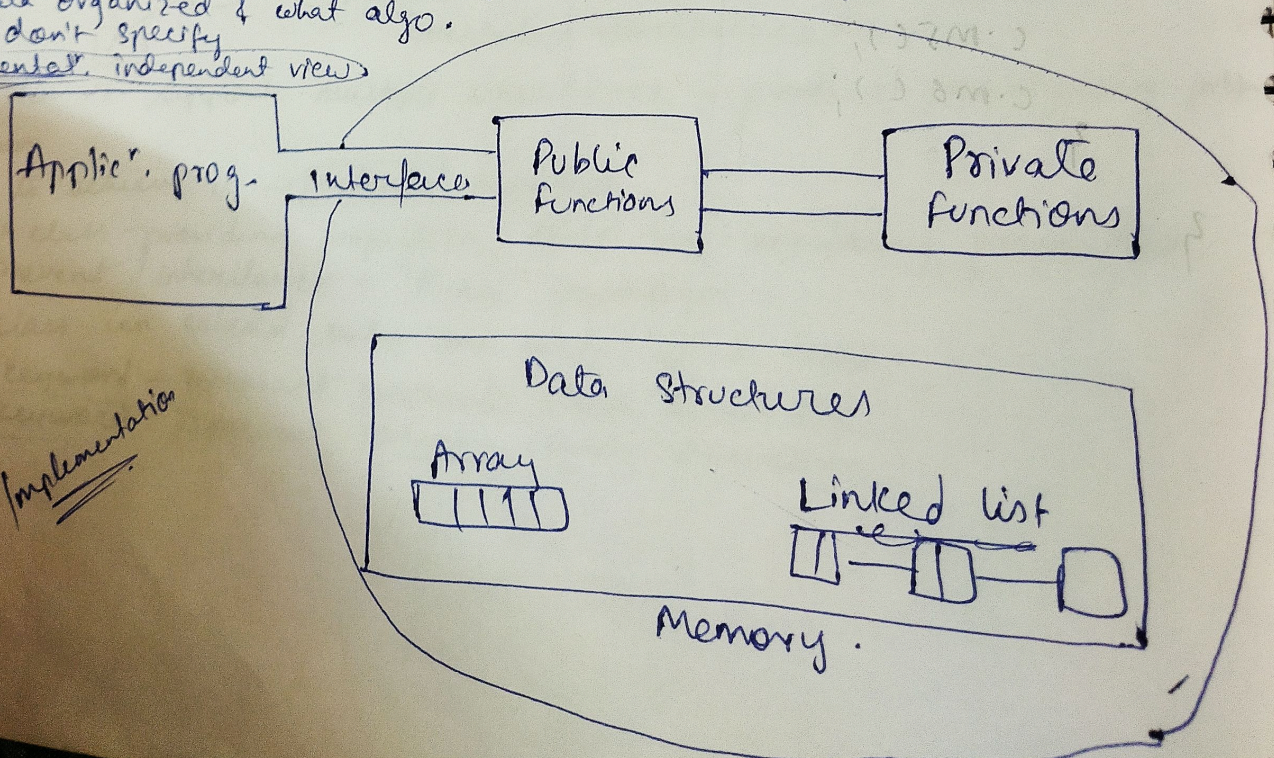
OOL

- Follows an Object-based approach with classes & objects.
- focus on objects & data encapsulation.
- Data & fun: are encapsulated in objects.
- More secure - data is hidden using encapsulation.
- High reusability through inheritance & polymorphism.
- C++, Java, Python, C#.
- Better suited - complex problems.
- Object-based str. with relationship b/w objects.
- Supported.
- function & operator overloading supported.

```
class Hello
{
    void sayHello()
    {
        System.out.println("Hello");
    }
}
```

② ADT → only mention what operations are to be performed but not how these operations will be implemented.

- how data organized & what algo. used → don't specify
- implementation independent view



Code :-

```
class A
{
    void m1() {S.o.p ("m1 method "); }
    void m2() {      ("m2       "); }
}

class B extends A
{
    void m3() {   }
    void m4() {   }
}
class C extends B
{
    void m5() {   }
    void m6() {   }
}
psvm (S[] args)
{
  A a = new A();
  a.m1();
  a.m2();
  B b = new B();
  b.m1();
  b.m2();
  b.m3();
  b.m4();
  C c = new C();
  c.m1();
  c.m2();
  c.m3();
  c.m4();
  c.m5();
  c.m6();
}
}
```

Output

```
m1
m2
m1
m2
m3
m4
m1
m2
m3
m4
m5
m6.
```

Module - 3

(1) Inheritance →

- feature of OOP
- allows a class (subclass / derived class) to inherit properties & behaviours (methods) from another class (superclass / base class).
- enables creation of hierarchy
- more specialized class can reuse, extend & modify the functionality of a more general class.

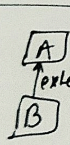
1. Reusability → subclass inherits methods from its superclass, avoiding code duplication.
2. Extendability → subclass can add new methods / modify inherited ones, providing more specialized behaviour.
3. Polymorphism → subclass object can be treated as an instance of its superclass, enabling dynamic method dispatch.

- UML Terminology :- "is-a" relationship

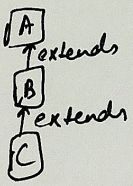
(2) Design Pattern

- help developers create flexible & maintainable code by abstracting common tasks & providing well-structured ways of solving problems.
- classification :-

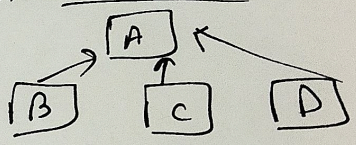
Types: (1) Single



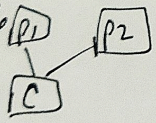
(2) Multilevel



(3) Hierarchical



(4) multiple



(5) Hybrid
multiple + hierarchical

- Java doesn't support multiple inheritance — produces ambiguity problem
- how to achieve it → extends keyword
- Parent class - providing properties, Child class - acquiring properties.
- to prevent inheritance - "final" modifier.
- one class can extend only one class at a time.
- this keyword - represent current class members
- Super keyword - represent parent class members.

⑦ Setting Priority of Threads in Java

- using `setPriority(int priority)` method of the Thread class.
- priority of a thread influences the scheduler's decision on which Thread to execute first, but it doesn't guarantee the order of execution.

Range of priority values

from 1 to 10.

1. MIN_PRIORITY: 1 (lowest priority)
2. NORM_PRIORITY: 5 (default priority)
3. MAX_PRIORITY: 10 (highest priority)

How to set priority?

Thread.setPriority() Method :- Sets the priority of a thread.
Thread.getPriority() Method :- returns the current priority of a thread.

Exception Handling →
If we pass a priority value outside the valid range (1 to 10), Java throws an IllegalArgumentException.

⑧ Exceptions

An event that disrupts the normal flow of a prog's execution.
occurs during runtime
can be caused by invalid user input, hw failures or bugs in code.

Types → checked → known at compile time compiled in JVM
→ must be handled using try-catch or declared with throws.
→ ex:- IOException, SQLException.

unchecked → known at runtime
→ subclasses of RuntimeException.
→ ex. NullPointerException, ArrayIndexOutOfBoundException.

Errors → irrecoverable issues that can't be handled.
→ ex. OutOfMemoryError, StackOverflowError.

Keywords to handle Exceptions

① try → defines a block of code to test for exceptions try { } risky code
② catch → defines a block of code to handle a specific excep.. catch(Exc.Type e) { } handle exc.
③ finally → defines " " " that executes whether / not an exception occurs. finally { // cleanup code }
④ throw → used to explicitly throw an exception
throw new Exception("Error message");
⑤ throws → declares exceptions that a method might throw.
public void method() throws Exception {
// method code
}

* use multiple catch bl
* finally block is op
* except: improve - rob
errors.

ref. to notes - Rata

③ JAVA - not 100

1. Primitive Data t
2. Static member
3. Wrapper class

~~Use of Non~~
Freeform lang

~~2024~~

constructor overloading.
multiple constructors in same class with diff: parameter lists.
enables creating objects in various ways.

Object - pwim
Body → colour, shape, height, weight

```
class OverloadedConstructor {
    int x;
    OverloadedConstructor() {
        x = 0;
    }
    OverloadedConstructor(int value) {
        x = value;
    }
    void display() {
        S.o.p("Value of x: " + x);
    }
}
public class Main {
    psvm(S[] args) {
        OverloadedConstructor obj1 = new OLC();
                              obj2 = new OLC(25);
        obj1.display();
        obj2.display();
    }
}
```

if no constructor defined - java provides default constructor.
if any " is explicitly " - default con. is no longer provided.
const. can be overloaded to support multiple initialize scenarios.

Constructors
① Default constructor - const. with no parameters.
- used to initialize default values for an object.
- automatically provided by Java if no other cons. is defined.

② Parameterized const.
- Accepts arguments to initialize object attributes
- provides flexibility to assign specific values during object creation.

③ copy const. - const. that creates a new obj. by copying an existing object's attributes.
- not explicitly provided in Java but can be implemented manually.

code ref. to notes by Rafatar Sir

ex: public class MainExample {
    psvm(String[] args) {
        S.o.p ("Main method is entry point!");
    }
}

output :- Main method is entry point!

ex: public class NoMainExample {
    static {
        S.o.p ("Static block");
    }
}

output :- Error: Main method not found in class NoMainExample, pls. define main method as psvm(S[] args)

~~5 WAP sort 10 names entered by user in ascending~~

5 Command Line Arguments in Java for user.
- are inputs provided to a Java prog during its execution via command line.
- passed to the main method as Array of String.
- Syntax: Run the prog by passing arguments:
  java ProgramName arg1 arg2 arg3.
- CLA stored in args array.
- args array length represents no. of arguments passed.
- args are always treated as strings.

6 Constructors
Special method used to initialize objects.
It is called automatically when an object of a class is created.
Features → Same Name as class → const. has same name as class.
No Return Type (not even void).
Called Automatically (when object is created)
Types of constructors → (3 - default, parameterized, copy)
Able to take parameters

class Test {
}
Test t = new Test();

Test → class name.
t → ref. var. (object name)
= → assign. operator
new → keyword (used to create object)
Test() → constructor.

constructore
multiple cons
enables creati
class Overl
int
Overl
x=
}
Over
x=
}
voi
}
public cla
psv
}
}
# if no
# if any
# const.
Constru
code
ref. to note
byteco

③ JAVA API →

i> Java API (Applic° prog. Interface) → collection of pre-written classes, interfaces, and methods provided by Java to simplify development.
Includes core libraries like - java.lang, java.util, java.io, etc.

ii> Predefined class → provides ready-to-use functionality (ArrayList, HashMap).
Extensibility - programmers can create their own APIs to extend java functionality.
Ease of use - Simplifies complex tasks like file handling, networking & multithreading.

iii> Java API package → java.lang → basic classes like String, Math, object.
java.util → utility classes like ArrayList, HashMap
java.io → file & I/O handling
java.net → networking capabilities
java.sql → DB connectivity using JDBC.

iv> using JAVA API → import the req. pck.
import java.util.*; // importing all classes
import java.util.ArrayList; // specific class.

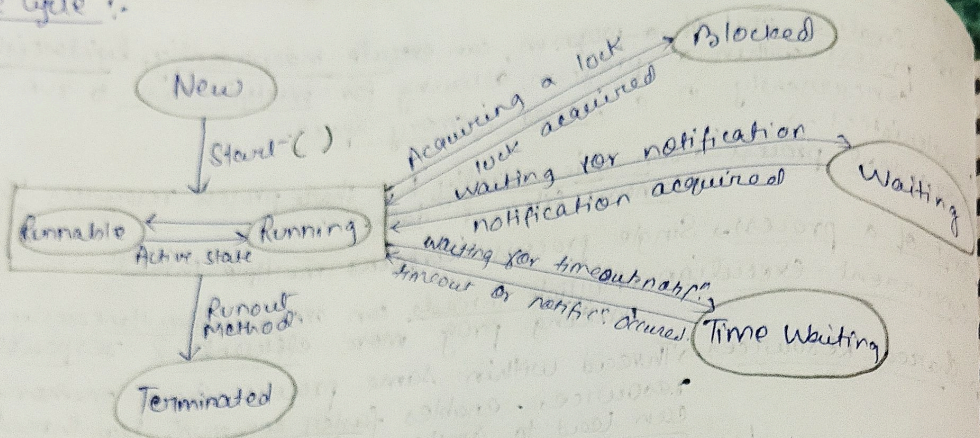
v> Java API Documentation - provides detailed description of all classes, methods & interfaces.
*Developers save time & effort by leveraging prebuilt functionality.

JDK → notes. JRE, JVM → notes

Byte code → intermediate code generated by Java compiler
→ executed by Java Virtual M/c (JVM).
→ platform independence - can run on any system with JVM.
→ file format - it is stored in .class files created after compiling .java source files.
→ execution - JVM interpretes or JIT compiles byte code into native m/c code for execution.
→ Security - runtime checks - enhance security & prevent malicious code execution.

④ Main method essential for Java?
- Yes, JVM will not execute the code if main method is missing.
- Act as starting point for Java Applic°.
- Signature must be:- public static void main(String[] args).
- Frameworks like Java EE or JavaFx might not req. main method explicitly bcz. container /runtime env. manages execution.

Life Cycle :-



States of a Thread :-

a) New state → A Thread is in this state after it is created using Thread class but before start() method is invoked.
Thread t = new Thread();

b) Runnable state → thread enters runnable state after start() is invoked
→ ready to run but waits for CPU to schedule it.
t.start();

c) Running state → gets the CPU & starts executing.
→ managed by thread scheduler, which decides when a Thread should run.

d) Blocked / waiting state → (timed waiting) - until another thread signals / notifies it.
when waiting for a resource / another thread to complete.
→ waits for sp. amount of time (eg. sleep(), join()).

e) Terminated state → once it completes its execution or is explicitly stopped.

Methods that transition to this state

✓ Thread() constructor
✓ start()
CPU scheduler picks the thread.
wait(), join(), sleep()
sleep(ms), join(ms), wait(ms)
run() method completes / thread stops.

State
New
Runnable
Running
Waiting.
Timed Waiting.
Terminated.

③ JAVA API →
i> Java API (Applic
interfaces, and
Includes core libr
ii> Predefined class
Extensibility - Pr
fun
Ease of use - Si
mu
iii> Java API packa
iv> using JAVA A
v> Java API Doc
*Developers
JDK → notes
Byte code → In
→ exe
→ pla
→ fil
→ ex
→ S
④ Main met
- Yes, JVM
- Act as st
- Signature
- framework
method
execution

(2) Thread - Life cycle

Thread → Smallest unit of a program, can execute independently, lightweight process within a program, allowing for multiple task to run concurrently in a shared memory space.

Characteristics →

1) Independent process → execution → run independently but share process resources like memory files & data.
2) Part of a process → Single process - contains multiple thread.
3) Concurrent execution → Multiple threads can execute simultaneously, making prog more efficient & responsive.
4) Shared Resources → Thread within same process share memory & resources - enables faster comm". b/w threads but can lead to synchronization issues.

Ex → Java provides [Thread] class and [Runnable] ~~class~~ interface to create thread.

```
class MyThread extends Thread {
    public void run() {
        S.o.p("Thread is running.");
    }
}
public class ThreadExample {
    psvm(String[] args) {
        MyThread thread = new MyThread();
        thread.start();
    }
}
```

```
class MyRunnable implements Runnable {
    public void run() {
        S.o.p("Thread is running.");
    }
}
public class ThreadExample {
    psvm(String[] args) {
        Thread thread = new Thread(new MyRunnable());
        thread.start();
    }
}
```

Common thread operations :-

- Start a thread :- [start()] - begins thread execution
- Sleep a thread :- [Sleep (milliseconds)] - pause thread for specified duration
- Interrupt a thread :- [interrupt()] → interrupts thread execution.
- Joins Thread :- [join()] → waits for a thread to finish before continuing prog. execution.

Advantages (use)

→ creating high-performance, responsive, and efficient programs in modern sw.

CA-3

(1) Method Overloading (Late Binding)

- Occurs when 2/more methods in the same class have same name but diff. parameters.
- happens within a (single class).
- parameters must differ in no., type or order.
- Return type can differ b/w overloaded methods.
- Resolved at (compile-time)
- @Override keyword not used.
- To perform similar operations with diff. inputs.

Code

```
class OverloadingExample {
    void show(int a) {
        S.O.p("Integer:" +a);
    }
    void show(String s) {
        S.O.p ("String:" + s);
    }
    void show(double d) {
        S.O.p ("Double:" +d);
    }
    psvm (String[] args) {
        OverloadingExample obj
        = new OverloadingExample();
        obj.show(10);
        obj.show("Hello");
        obj.show(5.5);
    }
}
```

- enables flexibility within same class

(Early binding)

Method overriding

- occurs when a subclass provides a specific implementation for a method already defined in parent class.
- involves inheritance, occurs in subclass
- must have same name, parameters and return type as in parent class
- Return type must be the same as overridden method in parent class
- Resolved at (run-time)
- Require @Override annotation for clarity & correctness.
- To provide specific behaviour in a subclass.

Code

```
class ParentClass {
    void display() {
        S.O.p("Display method in
            ParentClass.");
    }
}
class ChildClass extends ParentClass {
    @Override
    void display() {
        S.O.p("Display method in
            ChildClass.");
    }
}
public class OverridingExample {
    psvm(String[] args) {
        ParentClass parent = new ParentClass();
        parent.display();
        ParentClass child = new ChildClass();
        child.display();
    }
}
```

- ensures - subclass can modify/extend behaviour of a method defined in its parent class.

(2) Thread - Life

Thread → Smallest process concurr

Characteristics

1> Independent

2> Part of a

3> Concurrent

4> Shared

Ex: → Java create

class MyTh

publi

public

comm

- Start
- Slee
- Inter
- Joi

Adv

Inheritance

- Inheritance - Process of inheriting properties from 1 class into another.
- how to achieve it - by using `extends` keyword.
  - ↳ defines a relationship b/w 2 classes
- Parent class - class which is providing properties. Child class → who is acquiring properties.
- It is possible to create the object of both parent class & child class. But it is always recommended to create object of child class only.
- it is possible to access parents class members by using child class object.
- If you are extending the class, class becomes parent class. If not extending then object class will become parent class.
- Root class of all java classes - object class - present in java.lang package.
- Every class of java is child class of object class directly or indirectly.
- In Java every class contains parent class except object class.
- To prevent inheritance - declare the class with "final" modifier.
- 5 types of inheritance - single, multiple, multi-level, hierarchical, hybrid
- Java supports only 3 types - single, multi-level, hierarchical don't support other two bcz. it generates ambiguity problems
- In Java, one class is able to extend only single class at a time.
- To represent current class members - this keyword
  - parent class " - super keyword

Polymorphism

→ One thing in many forms. / ability to appear in more than 1 form. / one person with different behaviours

→ 2 types of polymorphism in Java:-

i> Compile-Time polymorphism / static binding / Early binding.
ex:- we achieve this by method-overloading

ii> Run-Time polymorphism / Dynamic Binding / Late Binding.
ex:- we achieve this by method-overriding

Generally - 3 types of overloading in Java:-

i> method overloading
ii> constructor overloading
iii> Operator overloading.

method overloading → class Test

When class contains more than one method with same name but diff. no. of arguments, it is called overloaded methods.

```
{
void m1 (int a)
void m2 (int a, int b)
}
{
void m2 (int a)
void m2 (char ch)
}
```

or

When class contains more than one method with same name, same no. of arguments but diff. data types.

Class contains → ① variables ← local / static / instance.
② Methods ← local / static / instance
③ Constructors ← default cons / user-defined cons
④ instance block
⑤ static block.

Creating objects by following ways:
① new keyword
② instance factory method
③ static factory method
④ pattern factory method
⑤ newInstance() method
⑥ clone() method
⑦ Deserialization process

Constructors

```
class Test
{
}
Test t = new Test();
```

Test ---> class name
t ---> ref var (object-name)
= ---> assignment operator
new ---> keyword (used to create the object)
Test() ---> constructor.

* Inside the class, if we are not declaring at least one constructor, then only default constructor will be generated.

Rules to declare constructors in Java:-
1) Constructor-name & class-name must be same.
2) Constructors are able to take parameters.
3) Constructors not allowed return types. (not even void)

Types of constructors:-
1) default constructors (0-argument const. // empty implementation)
(generated by compiler) (Run-time executed by JVM)
2) User defined const. (0-arg., parameterized const.)

ex-① → default constr. execution.

```
class Test
{
  void m1()
  {
    SOP("m1 method");
  }
  /* default cons
  Test()
  {
    // empty impl
  }
  */
  psvm(String[] args)
  {
    Test t = new Test();
    t.m1();
  }
```

ex-② Parameterized Constr.

```
class Test
{
  void m1()
  { SOP("m1 method");
  }
  Test()
  {
    SOP("0-arg cons");
  }
  Test(int a)
  {
    SOP("1-arg cons");
  }
  psvm(String[] args)
  {
    Test t = new Test();
    Test t1 = new Test(10);
    t.m1();
    t1.m1();
  }
```

Methods → Inside class, directly writing business logics are not allowed.
Inside class, we declare method & inside method, we write logics.

```
class Test
{
    int a = 10;
    int b = 20;
    SOP(a+b); ✗
    void add()
    {
        SOP(a+b);
    }
}
```

~~Method Name (functionality)~~

2 types of methods in Java:
- Instance: `void m1()` { // logic }
- Static: `static void m1()` { // logic }

Every method contains method-name which is executed when parameter-list is passed after the method is compiled it returns the value through return-type, to specify the permission is. modifier-list and to handle the exception - throws exception.

Syntax of Method:-

modifier-list Return-type method-name (parameter-list) throws exception

Ex:-
```
public void m1(int a, int b)
private int m2()
private int m3(int a) throws exception
```

method signature → method name & its corresponding parameter list.

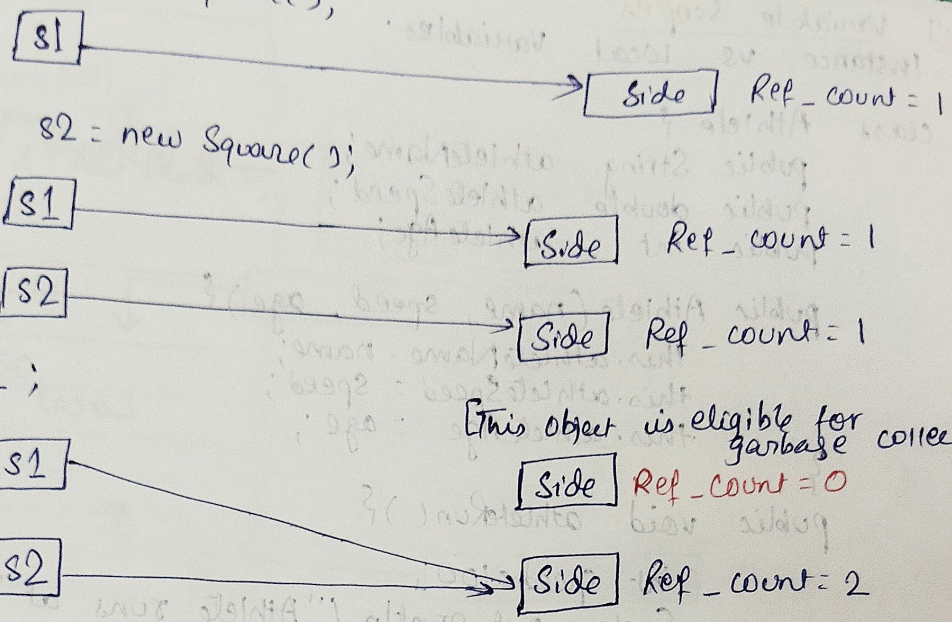
```
method-name (parameter-list)
m1 (int a, int b)
m2 ()
m3 (int a)
```

void add(int a, int b)

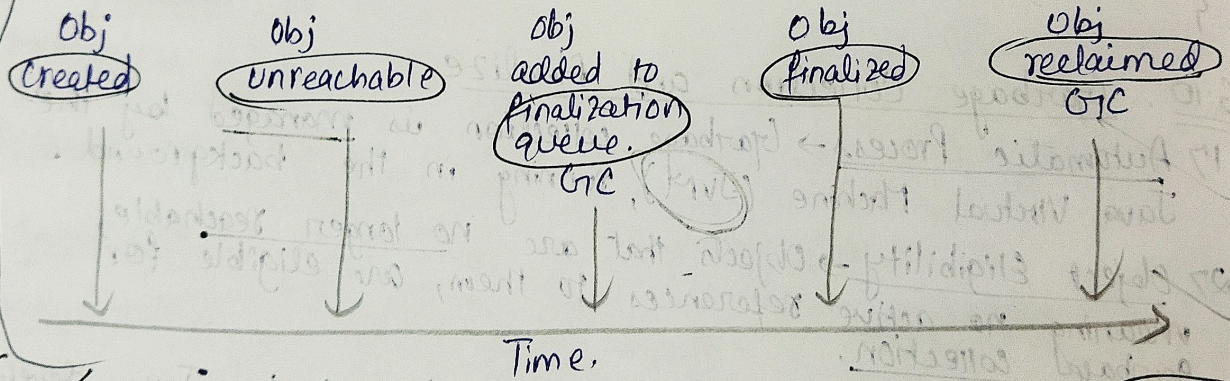
Every method contains 3 parts → declaration, implementation, calling

```
void m1() // ① method declaration
{
    logics here // ② method implementation.
}
Test t = new Test
t.m1() // ③ method calling.
```

Square s1 = new Square();



Life time of finalizable object obj



Finalization → Before an object is garbage collected, the finalize() method may be called, giving the object a chance to clean up resources. However, it's not guaranteed to run, and its usage is generally discouraged.

Optimization → Developers can optimize the process indirectly through code practices, like setting unnecessary object references to null.

System.gc() Call: While System.gc() suggests that the JVM performs garbage collection, it's not a guarantee.

6.9 Variable Scopes

Instance vs Local Variables.

```
class Athlete {
    public String athleteName;
    public double athleteSpeed;
    public int athleteAge;

    public Athlete(name, speed, age){
        this.athleteName = name;
        this.athleteSpeed = speed;
        this.athleteAge = age;
    }
    public void athleteRun(){
        int speed = 100;
        System.out.println("Athlete runs at "+speed+"
                            Km/hr");
    }
}
```

→ Instance Variable

→ Local Variables.

6.10. Garbage Collection and Finalize

1) Automatic Process → Garbage collection is managed by the Java Virtual Machine (JVM), running in the background.

2) Object Eligibility → Objects that are no longer reachable, meaning no active references to them, are eligible for garbage collection.

3) No Manual Control → Unlike languages like C++, Java developers can't explicitly deallocate memory. Garbage collection is automatic and non-deterministic.

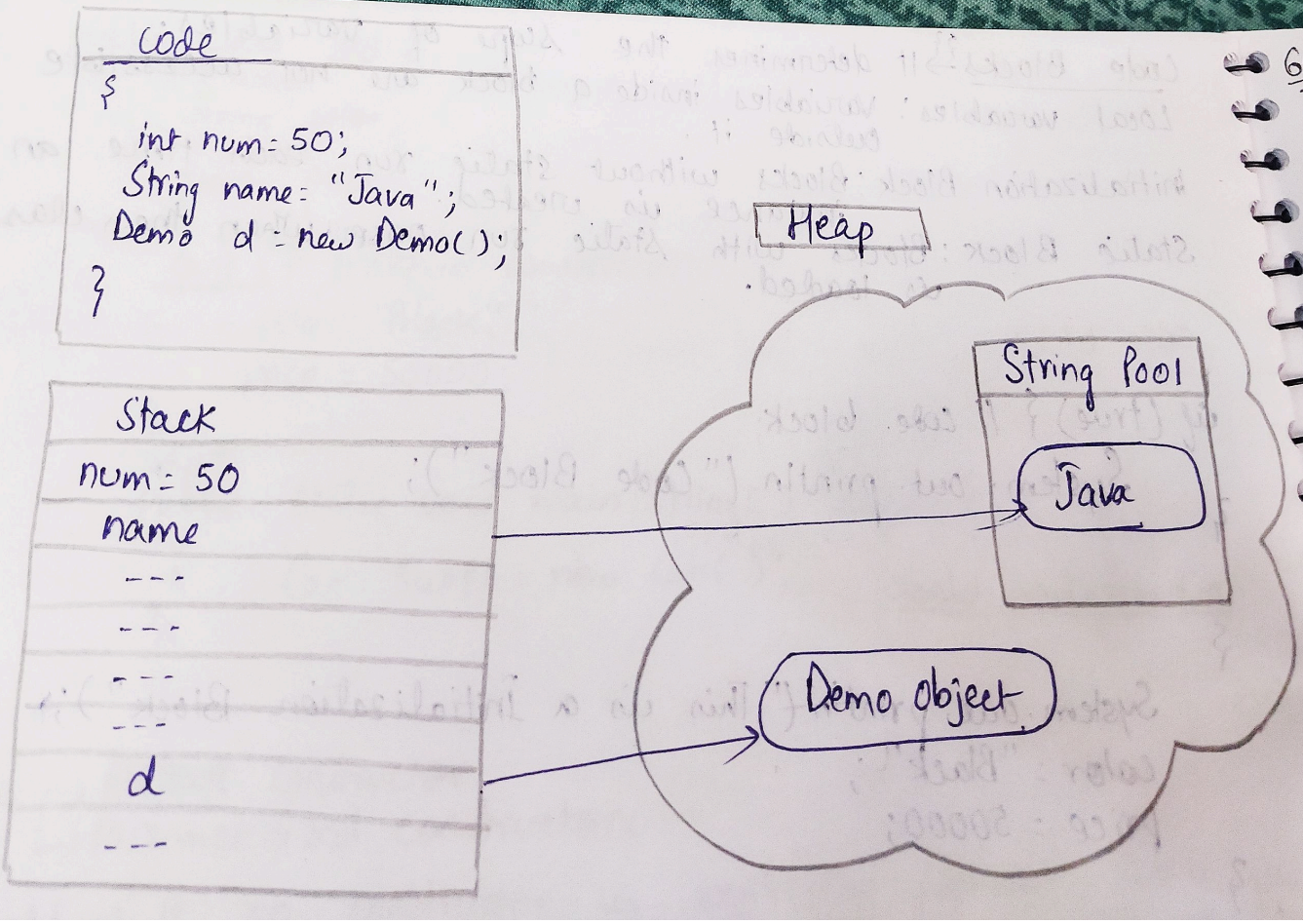
4) Generational Collection → Java uses a generational garbage collection strategy, which divides memory into diff. regions (young, old, permanent generations) based on objects ages.

5) Heap Memory → Garbage collection occurs in the heap memory, where all Java objects reside.

6) Performance Impact → Garbage collection can affect application performance, particularly if it runs frequently ~~for~~ or takes a long time to complete.

Code

```
{
  int num = 50;
  String name = "Java";
  Demo d = new Demo();
}
```



6.9 Varia
Instan
class

Code Blocks {} → It determines the scope of variables.

Local variables: variables inside a block are not accessible outside it.

Initialization Block: Blocks without static run each time an instance is created.

Static Block: Blocks with static run once when the class is loaded.

Code block

```
if (true) { // code block
    System.out.println("Code Block");
}
```

Initialization block

```
{
    System.out.println("This is a Initialization Block");
    color = "Black";
    price = 50000;
}
```

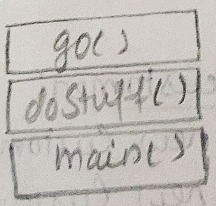
Static block

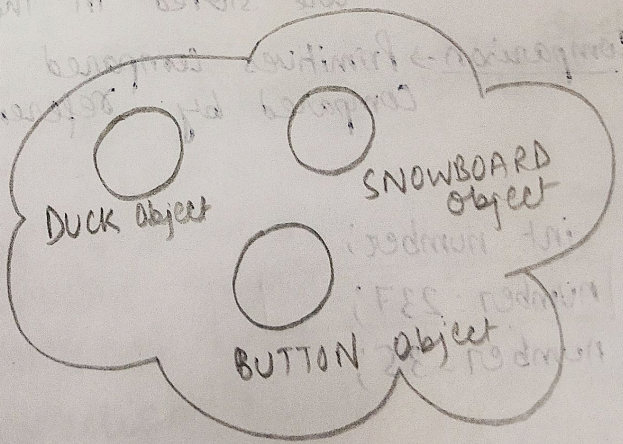
```
static {
    System.out.println("This is a Static Block");
}
```

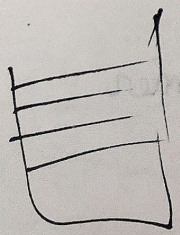
6.7 Stack vs Heap Memory

Heap → Dynamic
- No particular order.
- Where ALL objects live.

Stack
- Ordered on top of eachother!
- Where method invocations and local variables live.







ex: Invoking Other Constructors → can use this() to call another constructor within the same class. This is known as constructor chaining.

```
public class Person {
    private String name;
    private int age;
    public Person (String name) {
        this (name, 0); // calls the other constructor
    }
    public Person (String name, int age) {
        this.name = name;
        this.age = age;
    }
}
```

6.6 Static keyword

make it class method so that it can be called using class name without creating an object of the class

name of the method which is called by JVM.

public | static | void | main(String[] args)

1. Static Variables → Belong to the class, not individual instances. shared among all instances of the class.
2. Static Methods → can be called without creating an object of the class. can only directly access static variables and other static methods.
3. No Access to Non-static Members → static methods and blocks can't directly access non-static members (variables and methods) of the class.

6.6 Constructors

Purpose: Constructors initialize new objects & set initial state for the object's attributes.

Naming: A constructor must have the same name as the class in which it is declared.

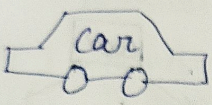
No Return Type: constructors don't have a return type, not even void.

Automatic calls: A constructor is automatically called when an instance of a class is created.

6.4 Class vs object

Class is a blueprint; objects are real values in memory.

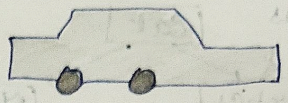
Class



Properties
- color
- price
- Km
- model

Methods-behaviour
- Start() ✓
- backward() ✓
- forward() ✓
- Stop() ✓

Object



Property values
- color: red
- price: 23,000
- Km: 1,200
- model: Audi

Methods
- Start()
- backward()
- forward()
- Stop()

In Java, class and objects are fundamental concepts in OOP.

	Class	Object
Defin":	Blueprint for creating objects.	Instance of a class. Represents a Sp. ex. of the class.
Purpose	Defines str. (fields) and behaviour (methods)	
Declar".	Declared using (class) keyword	using (new) keyword.
State	doesn't hold state/data	Holds state/data (instance variables)
Behavior	defines behaviour (methods)	Performs behaviour defined by the class methods.
example	public class Car(...)	Car myCar = new Car ("Toyota", "Corolla", 2022);

6.5 This keyword

There can be a lot of usage of Java This keyword. In Java, this is a reference variable that refers to the current object.

1. this can be used to refer current class instance variable.
2. this can be used to invoke current class method (implicitly)
3. this() " " " " " " class constructor.
4. this can be passed as an argument in the method call.
5. this can be passed as argument in constructor call
6. this can be used to return the current class instance from the method.

ex:- Referring to Instance Variables

```
public class Person {
    private String name;
    private int age;
    public Person(String name, int age) {
```

4) Object-Oriented Programming Keywords:

class → declares a class.
interface → declares an interface → ref. type like class but contain only abstract methods
extends → indicates that a class is inheriting from a superclass.
implements → indicates that a class is implementing an interface.
this → Refers to the current instance of a class.
super → Refers to the parent class / superclass
new → creates new objects
abstract → declares an abstract class or method.
final → defines a const. / prevents inheritance / method Overriding.
static → defines class-level methods and Variables.
instanceof → tests whether an object is an instance of a class or subclass.

5) Exception-Handling:

try → defines a block of code to test for exceptions.
catch → catches exceptions thrown in try block.
finally → defines code that will execute regardless of whether an exception is thrown.
throw → throws an exception.
throws → Declares that a method can throw one or more exceptions.

6) Miscellaneous:

void → Specifies that a method doesn't return any value.
enum → declares an enumerated type.
package → Specifies that package of the class.
import → imports other Java packages / classes into a class.
Synchronized → ensures that a method / block of code is accessed by only one thread at a time.
volatile → indicates that a variable may be changed unexpectedly in multithreading.
transient → Prevents a variable from being serialized.
assert → tests a condition for debugging purposes.

7) Reser
true →
false →
null

3.6 Es
Escape
allow
chara
They
deter
It

3.7

1) Control Flow Statements

if → conditional statement that execute a block of code if the condition is true.

else → executes a block of code if condition in if state. is false.

switch → Select one of many code blocks to execute based on a condition.

case → Defines code to be executed when the case matches in a switch statement.

default → defines a default block of code in switch statement.

while → executes a block of code repeatedly while the cond? is true.

do → similar to while, but checks the cond^n after the loop body executes.

for → executes a block of code a certain no. of times.

break → exits a loop or switch statement.

continue → skips the current iteration in a loop and continues with the next iteration.

return → exits from a method and optionally returns a value.

2) Access Modifiers :

public → makes the class, method, or variable accessible to all other classes.

private → makes the class, method, or variable accessible only within its own class.

protected → allows access within its own package and subclasses.

3) Data Types :

int → defines an integer data type.

byte → defines a byte data type.

short → defines a short integer data type.

long → defines a long integer data type.

float → " a floating-point data type.

double → " double-precision floating-point data type.

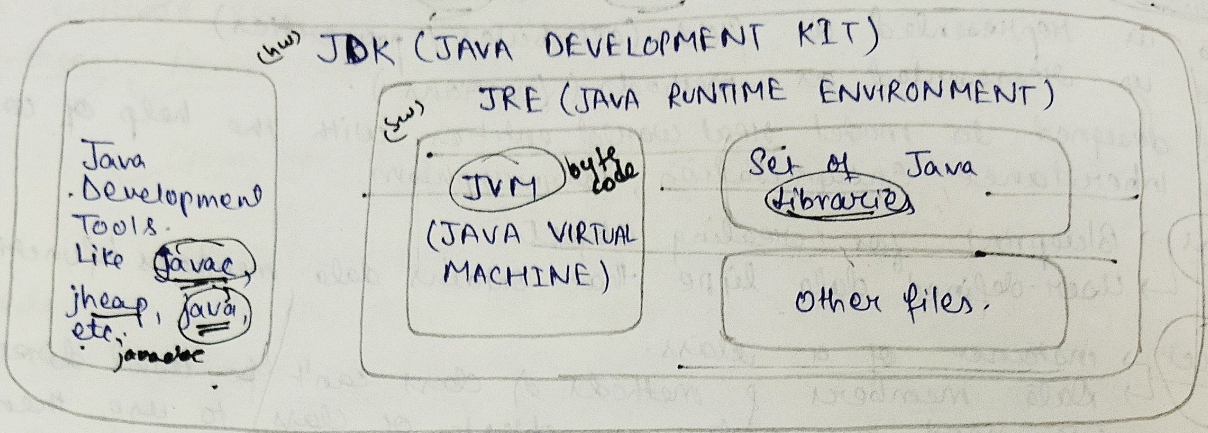
char → " character data type.

boolean → " boolean data type (T or F)

File extensions

- .java - Contains Java source code, High level human Readable, used for development, file is editable.
- .class - contains Java Bytecode, for consumption of JVM, used for execution, not meant to be edited.

2.6 JDK vs JVM vs JRE



JDK → Software development kit, used to develop java applications.
→ includes JRE, an interpreter/loader (java), a compiler (javac), a doc generator (javadoc) and other tools needed for java dev.
→ Essentially, JDK is superset of JRE.

JRE → Part of JDK but can be downloaded separately.
→ Provides libraries, JVM and other components to run applic.
→ Doesn't have tools and utilities for developers like compilers or debuggers.

JVM → It's a part of JRE and responsible for executing bytecode
→ Ensures Java's Write-Once-Run-anywhere capability.
→ Not platform-independent. A diff. JVM is needed for each type of OS.

2.8 Importance of main method.

```
public  static  void  main (String args[])
```

public → access specifier
static → keyword
void → return type
main → method
String args[] → array of String type.

Entry Point → It's entry point of Java prog., execution starts. Without main method, JVM doesn't know where to begin running code.

Public and Static → ensuring its accessible to JVM without needing to instantiate the class.

(5) Distributed → Java is inherently distributed, designed to facilitate network based application development and interaction, seamlessly integrating with internet protocols and remote method invocation.

1.3 Object-oriented Programming

OOPs is a programming paradigm based on the concept of Objects which can contain data and code → method.
Data is represented as fields (attributes / properties)
Code is represented as methods (functions).
OOPs - designed to model real-world entities with the help of concepts like Inheritance, Encapsulation, Polymorphism.

Class → Blueprint for creating objects.
↳ User-defined data type that contains data members / functions

Object → instance of a class.
↳ Data members & methods of class can't be used directly we need to create an object of class to use them.

2.1 Installing JDK from Oracle website.

2.2 First class: (using text editor)

```
import java.lang.*;
public class First {
    public static void main (String[] args) {
        System.out.println ("Welcome to KGiCoding");
    }
}
```

2.3 Compiling and Running

Program.java → JAVA compiler → Program.class → JVM → Program.

.java Source file → Javac → bytecode .class file → java → output.

2.4 Anatomy of a class

```
public class MyFirstApp {
    public static void main (String[] args) {
        System.out.print ("I Rule!");
    }
}
```

- public class MyFirstApp { → opening curly brace of the class
- This is a class
- The name of this class
- public so everyone can access it
- public static → void → The return type, void means no return value
- main → name of this method
- (String[] args) { → This method must be given an array of strings, and the array will be called args. → opening brace of the method
- System.out.print ("I Rule!"); → Print to standard output (defaults to command-line) → The string you want to print. → every statement must end in a semi-colon.
- } → closing brace of main method.
- } → closing brace of MyFirstApp class

Standardized into diff. editions for various computing platforms.

1.6 What is Bytecode?

Form of intermediate code - used in programming lang. to be executed by a virtual machine (VM) rather than directly by the hardware.
High level source code is compiled into bytecode by a compiler.
Virtual machine interprets / compiles the bytecode.
Java bytecode executed by JVM files have a .class extension.

Advantages: Portability → written once & run anywhere, as long as compatible VM is available (Platform independence)
Optimization → optimized by VM at runtime through techniques like Just-In-Time (JIT) compilation.
Security → run in a controlled environment.

Bytecode vs. Source code vs. Machine code :-

Source code → written in high-level prog. lang., human-readable and needs to be compiled or interpreted.
Bytecode → Intermediate Representation, portable, needs to be executed by Virtual Machine.
Machine code → low-level code, directly executed by CPU, specific to a particular type of hardware.

1.7 How Java changed the Internet?

. Portability with Write Once Run Anywhere.
. Security because of code running on virtual machine.

1.8 Java Buzzwords (Features of Java)

(1) Robust → Strong memory management, exception-handling, type-checking mechanisms, helps in preventing system crashing and ensures reliable performance.

(2) Multithreaded → ability of a CPU to execute multiple threads concurrently, allowing for more efficient processing and task management.

(3) Architecture Neutral → Java is architecturally neutral bcz. its compiled code (bytecode) can run on any device with a Java Virtual Machine (JVM), regardless of the underlying hardware architecture.

(4) Interpreted and high performance → Java combines high performance with interpretability, as its bytecode is interpreted by the Java Virtual Machine (JVM), which employs Just-In-Time (JIT) compilation for efficient and fast execution.

(5) Distribute

1.9 Object-

OOPs is

Objects

Data is

code is

OOPs- de

like In

Class

Objec

2.1

2.2

2.3

10.09.24

JAVA

1.1 Why you must learn Java?
1. One of the most popular language. Java currently runs on 60,00,00,00,00,000 devices.
2. Wide usage (web-apps, backend, Mobile apps, enterprise software).
3. High paying and a lot of jobs.
4. Object oriented
5. Rich APIs and community support. Application Programming Interface.

1.2 What is a programming language?
Humans use natural lang. like Hindi/Eng. to communicate with computers only understand 0/1 or on/off.
Programming languages are giving instructions to a computer.
Instructions: tells computer what to do, these instructions are called code. Human instructions are given in High Level Languages.
Compiler converts HLL to LLL or machine code.

1.3 What is an Algorithm?
Step-by-Step procedure or a set of rules used to solve a specific task. It provides a clear seq. of instructions that can be followed to reach a desired outcome.
Characteristics:- Definiteness → each step clearly defined.
Input → takes input values
Output → produces output values
Finiteness → must terminate after a finite no. of steps.
Effectiveness → each step must be basic enough to be performed mechanically.

1.4 What is Syntax?
Structure of words in a sentence.
Rules of a language for programming, exact syntax must be followed.

1.5 History of Java.
Developed by James Gosling at Sun Microsystems (Early 1990s);
Originally named Oak, later renamed Java in 1995
First Release - 1995 - Introduced "Write Once, Run Anywhere" concept with cross-platform compatibility.
Developed with vision of backward compatibility
↓
Should not break with new version release.
Rapid Growth and Diversification (Late 1990s - 2010s).
→ Expanded from web applets to server-side applications;