Packages :-

|n2.sum|n3.add

java.lang - Language Package (Default Package)
java.util - Utility Package
java.io - Input/Output Streams and File Package
java.net - Networking Package
java.sql - DataBase Connection Package

→ **Operators** → Perform Operations

1). Arithmetic → +, -, *, /, %
2). Relational → >, >=, <, <=, ==, != (Compare 2 values / Boolean Result)
3). Logical → AND, OR, NOT
4). Increment/Decrement → ++, -- (Incl Dec value by 1)

Constructors NS → Constructor is a method having the same name of the Class, and executed while object creation, because constructor_call is available in object creation syntax attached with 'new' keyword.

Control Structures

→ ⇒ The structures which are used to control the Part of Program, for execution

1). Selection Statements → Are used to select one part of the Program for executed based on condition
↓
Simple if
if-else
Nested-if
Switch case → to select one, from multiple option

2). Iterative Statements → Are used to execute selected lines of Prog. repeatedly on some condition
↓
while loop → condition checked first, if condition is true, then the loop executed until the condition false.
do-while loop → loop executed first, then condition is checked
for loop → Initialization, condition and Incre/Decre declared in same line.

3). Branching Statements → Are used to transfer the control from one location to another location
↓
break;
continue
exit;
return;

s.nextInt() → to read Number
s.nextLine → to read String data from keyboard.

import → 'import' statement is used to make class or interface available from one package to another package.

Parameters → The variables which are used to transfer the data, from one method to another method.

Blocks → The set of statements, which are declared within the
S, NS flower brackets and executed automatically. without calling

Static Block → Static Block execute automatically, while class is loading in main method.
executed only once

Scanner class

⇒ Scanner class is a pre-defined class → 'java.util'
⇒ Provides methods to read data into Java Programs.
→ nextShort(), nextInt(), nextLong()

```
Scanner s = new Scanner(System.in);
```

⇒ While reading data from console, if we read String after numeric data, then the data is skipped. This can be overcomed using Parse methods.
⇒ byte b = Byte.p

```
import java.util.Scanner;
```

Variables → Data holders

Primitive Data Type	Non-Primitive Data Type
byte, short, int, long, float, boolean, char, double	Class, Interface, Array, Enum, object
↓ hold → values	↓ hold → Object References

⇒ Based on 'static' keyword, the variables are categorized into 2 Types
1). Static Variables
2). Non-static variables

1). Static Variables ✡

⇒ Declared with 'static' keyword
⇒ Static variables will get memory within the class, while class loading.
⇒ Can be accessed with Class_name directly
⇒ Also known as 'Class Variables'.

2). Non-static Variables ✡

⇒ Declared without ~~A~~ static keyword.

Two Types :-

Instance Variables	Local Variables
⇒ Declared outside the method	⇒ Declared Inside the method
⇒ Will get memory within the object while object creation	⇒ Will get memory within the method, while method execution.
	→ Local variables can't be static.

this :- 'this' keyword will hold the reference of object from where current method or constructor is executing.

✓ → 'this' keyword is used when we have same variable name in local variables and Instance variables.

Methods

⇒ methods are actions, which are Performed to generate result.

1). Static Method → Accessed with Class_name

2). Non-static Method → Accessed with Object_name

Scope of variables ✡

i). Scope of Static variables (Class Scope)

⇒ Static variables are accessed by the method of class and methods of Objects of same Class.

ii). Scope of Instance variables (Object Scope)

⇒ Instance variables are accessed by the method of same Object.

iii). Scope of Local variables (Method Scope)

⇒ Local variables are accessed inside the methods, where they are declared.

Access Modifiers ✡

⇒ Access modifiers specify the scope of Programming components within the Project.

i). public → Programming components → accessed within the Project.

ii). private → -||- → within the class

iii). protected → -||- → within the Packages

iv). default → -||- → within the Package

⇒ Relationship b/w Classes

→ The process of establishing Communication b/w Classes.

1). References → one object holding reference of another object

2). Inheritance → Process of Linking classes using 'extends' keyword

3). InnerClasses

Reference concept is also known as 'HAS-A relation'. (composition)

Inheritance concept is known as "IS-A relationship". (Aggregation)

IS → Strong (code reusability)

HAS → A → car Has an Engine

Method Area	Heap Area	Java Stack Area
main class loaded first then sub-classes loaded	Objects are created Objects are stored.	Method Executed

Method frame → The Partition of java stack Area, wher the method is copied for execution.

Methods → Executed on method call.

Blocks → Executed automatically, without calling.

Abstract class → Can hold block and constructors

Interface → Can not hold blocks and Constructors.

factory method → The method which is hide the object creation Process from the Us

Pillar of OOPS :-

①. Data Abstraction (Data Hide & showing functionalities)

②. Inheritance (Reusability) of code)

③. Polymorphism (Object to make many forms) < overriding Run time / Compile Time

④. Encapsulation (It provides security in object data)

✡ Static :- 'static' is a keyword.

It is a one copy storage.

→ static members are called by Class name. directly.

→ we can not call Non static members from static member directly.

we need to create a object of a class.

→ Static variables also called as class variables.

→ Static methods can not be overriden, as it is class level methods.

→ static says class related members only.

→ Static method will get memory within the class, while class loading.

→ Static methods can access static variables directly, but cannot access Instance variables

→ we don't need to create objects to call static method.

→ Instance methods can access both Static and Local variables.

→ Static blocks can access only static Variables.

→ Instance blocks can access both static and Instance variables.

→

Inheritance → code re-usability

→ The process of Linking classes using 'extends' keywords.

→ In Inheritance process the members of 'PClass' are available to 'CClass' and in this process, we create object for 'CClass' i.e. → B ob = new B();

✓ → In normal inheritance process one object is created, holding the members of PClass and CClass.

Note:-

→ In inheritance process, the members of PClass can be accessed by the CClass, but the CClass members cannot be accessed by the PClass.

case → 0-Parameter constructor from PClass/SuperClass

→ When we have 0-parameter constructor from the PClass then the compiler at compilation stage will add 'Super()' to the CClass constructor and which is PClass Con_call.

→ Parameterized constructor from the PClass/SuperClass. When we have parameterized constructor in PClass, then we must add 'super()' to the 'CClass' constructor to call PClass constructor and pass parameters.

→ In inheritance process we call PClass constructor through CClass constructor using 'super()'. ✓

Method Overriding ✓

→ The method with same method_signature in PClass and CClass, then PClass method is replaced by CClass method, while object creation......

{ Same return_type , same method_name
same para_List , same para_type }

Method Hiding

When we have same Static method signatures in PClass and CClass, then it is method Hiding Process

Method Overloading ✓

More than one method with same method_name but differentiated by their para_List or Para_type.

Super() → 'super()' is used to acess the constructors from the PClass/SuperClass.

this() → 'this()' is used to access the constructor from the same class.

Command-Line Argument program

The program in which we pass parameters to the Standard main() method is known as Command-Line Argument program.

void → Non-return_type method

Types:

i) Single Inheritance

The process of extracting the features from one class at a time is known as single Inheritance.

ii) Multiple Inheritance

→ The process of extracting the features from more than one class at a time.

Note:-
multiple Inheritance process using classes in java is not available, because

→ In java, the multiple-Inheritance process ✓ can be acheived using 'Interfaces'.

✓ Interfaces → achieve abstraction, 100% coupling

Interface is collection of variables and abstract methods upto java 7 version.

abstract method

Declared without method_body.

concrete method

Declared with method_body. ✓

Coding rules of Interface → No-constructor

1. We use 'interface' keyword to declare

2. The programming components which are declared within the interface are automatically 'public'.

3. The interface can be declared with both PDTV and Non-PDTV.

4. The variables which are declared within the interface are automatically 'static' and 'final' variables.

5. The methods which are declared within the interface are automatically Non-static abstract method

6. We cannot instantiate interfaces in java, because interfaces are 'abstract components'.

7. These interfaces are implemented to classes using 'Implements' keyword and the classes are known as Implementation classes.

8. These Implementation classes must construu the bodies for abstract method of Interf

9. There is no concept of declaring Blocks o constructors in Interface.

10. Interface can use the members of anoth interface using 'extends' keyword.

Abstract Class :

-> Declared with 'abstract' keyword.
-> Abstract class can hold variables, concrete methods, abstract methods, Blocks, constructors and features.
-> we must use 'abstract' keyword to declare abstract method.
-> Abstract classes in java are also 'abstract components' and which can not be instantiated.
-> These abstract classes are extended to classes known as 'Extention classes' and these extension classes must construct body for abstract methods of 'Abstract classes'.

✩ -> We can declare Inner Classes with in the Interface and which are automatically static member InnerClasses.
✩ -> We can declare InnerClasses within the AbstractClasses and which can be static or non-static member InnerClasses.

Generalization Process -> achieved using Interfaces

The process in which one object is created and the object will hold all the members of PClass and only Overriding members from the child class.

Syntax :- PClass ob = new CClass();

Note -> This generalization process can also be achieved using interfaces.
Syntax :- Interface_name ob = new ImplClass_name();

Anonymous InnerClasses

The InnerClasses which are declared without name are known as Anonymous InnerClasses.

Lambda Expression

The process of declaring the method without method_name is known as Lambda Expression or Anonymous method.

structure ->
```
(Para_List) ->
        {
          // method body
        }
```

✓-> Lambda Expression means para_List linked with method_body, without method name.
✓-> Interface will provide abstract method signature to hold Lambda Expression.

Abstraction :-

Process of hiding the implementation details while showing the functionalities is Abstraction

Encapsulation ✓

The process of binding data member into a single unit class is known as Encapsulation.

Encapsulation = Data Hiding + Abstraction

The main advantage of encapsulation is we can achieve security.
'Getter' and 'Setter' method :-
We declare variable as 'Private', to stop accesing them directly

Inheritance -> Reusability, data hiding, overriding.

Inheritance is a process, where one class acquires the properties of another.

PolymorPhism

PolymorPhism is the ability of a variable, function or object to take multiple forms.
Runtime polymorphism -> method Overriding

Exception Handling Process 'java.lang.Exception'

Exception !- The disturbance which is occured from the application.

. The process of used to handle the exception is known as Exception handling process.

We use following blocks in this process
i). try
ii). catch
iii). finally

i). try → 'try' block will hold the statements which are going to raise the exception.

when the exception is raised from the 'try' block, then automatically object is created for Exception_type_class and the object reference is thrown onto catch block.

ii). Catch →

'catch' block will hold Object_reference and the required msg is generated from catch block.

```
catch ( Exception_type_class obj_var )
{ // msg
}
```

iii). finally →

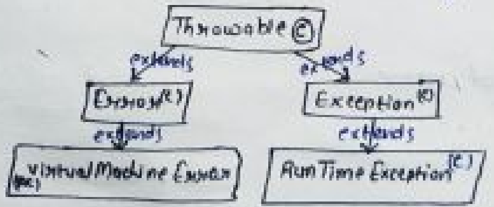
finally block is part of exception handling process, but executed independently without depending on exception.

→ In realtime finally block will hold resource closing operations. i.e exit; , return; , s.close();

Throwable

'Throwable' is a Built-in class from 'java.Lang' Package and which is root of exception handling process.

This 'Throwable' class is extended into the following two Sub Classes.
1). Error
2). Exception



1). Error

The disturbance which is occured from the Environment

get message() method !- getMessage() is from 'java.lang.Throwable' class and which is used to display the details of exception like message available in the object. Syntax !- String msg = ae.getMessage();

PrintStackTrace () method

It is used to display the complete details of exception like Exception_type_class_name, message, method_name, Line_no . . . .

Syntax !- ae.printStackTrace();

Exception

'java.lang.Exception' is the parent class of all the exceptions raised in the application.
1). Unchecked Exception
2). Checked Exception

1). Unchecked Exception

The exception which are not identified by the compiler at compilation stage. (Runtime Exception)

2). Checked Exception

The exception which are identified by the compiler and raised at compilation stage. (Compile Time Exception)

1). Unchecked Exception !-
i). Built-in Unchecked Exceptions
ii). User defined Unchecked Exceptions

i). Built-in Unchecked Exceptions

Ex !- java.lang.ArithmeticException
java.lang.NumberFormatException
java.util.InputMismatchException

ii). User defined Unchecked Exception

The Unchecked exceptions which are defined and raised by the programmer.

Steps to handle Exceptions

①. The user defined class must be extended from java.lang.Exception class.

②. we use try-catch blocks to handle exception

③. Declare the condition to raise the exception

④. If the condition is true, then raise the exception, which means create object for the User defined_class from where the exception is raised.

⑤. throw the object reference onto catch block using 'throw' keyword

⑥. Display the required msg from the catch block.

Note !- Pass Exception_details as parameter while object creation.

2). Checked Exception:

i). Built-in checked Exception
ii). User defined Checked Exception

i). Built-in checked Exception
java.lang.InteruptedException ✓
java.io.IOException ✓
java.sql.SQLException ✓
java.lang.ClassNotFoundException ✓

sleep() method :- Built-in method from java.lang.Thread class and which is used to stop the program execution process temporarily on sometime.

Syntax -> Thread.sleep(time-miliseconds);

→ 'throws' keyword specify to ignore the exception in current method and raised at method call.

ii). User defined Checked Exception :-

The Checked exceptions which are defined and raised by the programmer.

Note :- We use the following 2 steps to raise the exception

①. add 'throws' keyword to method signature to ignore the exception from current method.

②. Use throw keyword part of catch block and perform re-throwing process.

Annotation

The tag based information which is added to the programming components like Interfaces, classes, methods and variables is known as Annotation.

→ we use '@' symbol to represent annotation.

Ex. -> @Override
@SuppressWarnings

try-with-resource

try-with-resource statement is introduced by java7 version and which is used to close the resource automatically.
(which means finally block is not needed)

Null Pointer Exception

The process of using Non-Primitive data type variable, which is holding null-value, will raise Null Pointer Exception.

Exception Propagation

In exception re-throwing process, the exception is moved from one method to another method is known as Exception Propagation.

Multi-Threading

Task → The part of process is known as Task.

Multi-Tasking → Executing multiple tasks simultaneously.

Thread :- In the process of executing multiple tasks only some part of Task is executed.

Multi-Threading :- Executing Multiple Threads Simultaneously.

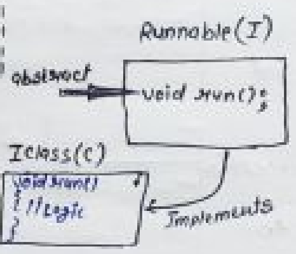
Creating & Executing Thread

⇒ In the process of creating thread the user defined class must be implemented from 'java.lang.Runnable' Interface.

Structure :-

```
Public Interface java.lang.Runnable
{
 public abstract void run();
}
```

```
IClass ob = new IClass();
Runnable ob = new IClass();
Runnable ob = new Runnable()
             {// IClass body
             };
Runnable ob = () ->
              {
               // method body
              };
```



```
try{
{thread.sleep(); }
} catch (InterruptedException
  ie) {ie.printStackTrace();}
```

⇒ We use the following steps to create Threads :-

①. The user defined class must be implemented from 'Runnable'.

②. The user defined class must construct body for run() method.

③. Create object for User defined class.

④. Create object for 'java.lang.Thread' class and pass the object_ref of User defined class as parameter while object creation.

⑤. execute run() method using 'start()' method.

Lambda Expressions in Threading concept

⇒ Lambda Expressions in Threading concept means declaring run() method as Lambda Expression. (run() method as Anonymous method).

Advantages of multi-threading :-

①. It doesn't block the User, because threads are independent you can perform multiple operations at the same time.

②. We can perform many operations together; so it saves time.

③. Threads are independent, so it doesn't affect other threads if an exception occurs in a single thread.

Thread Synchronization process

⇒ The process of ordering the threads for execution is known as Thread Synchronization Process.

⇒ Thread Synchronization can be performed in two ways :-

1). Mutual Exclusion Process
2). Thread Communication Process

1). Mutual Exclusion Process

⇒ The process of Locking the programming resources like Class, Object, and Method for ordering the threads for execution.

⇒ This Mutual Exclusion process can be Performed in three ways →

a). Synchronized block → Lock object
b). Synchronized method → Lock applied on Individual Instance Method
c). static synchronization → Lock on the Class

a). Synchronized block →

⇒ The process of declaring the statements with 'synchronized' keyword.

⇒ This synchronized block is used to apply the lock on the object.

Syntax →

```
synchronized (obj_ref)
{
 // statements
}
```

Limitation → when we apply the lock on the Object, then all the instance methods within the object, will be under the lock.

b). Synchronized method

The Instance method which is declared with synchronized keyword.

→ In this process the lock is applied on individual instance method and the method is available to one user at a time.

c). Static Synchronization

The static method which is declared with synchronized keyword is known as static...

⇒ In static synchronization process the lock is applied on the class
Also known as Class Locking Process.

3). Thread Communication Process

The process of establishing communication b/w threads using the following methods from 'java.lang.Object' class is known as Thread communication Process:

i). wait()
ii). notify()
iii). notifyAll()

i). wait() → ob.wait();

→ wait() is used to make the thread to wait until, it receives msg in the form of notify() or notifyAll().

ii) notify() → ob.notify();

→ notify() method will unlock the resource and send the msg to the next waiting thread.

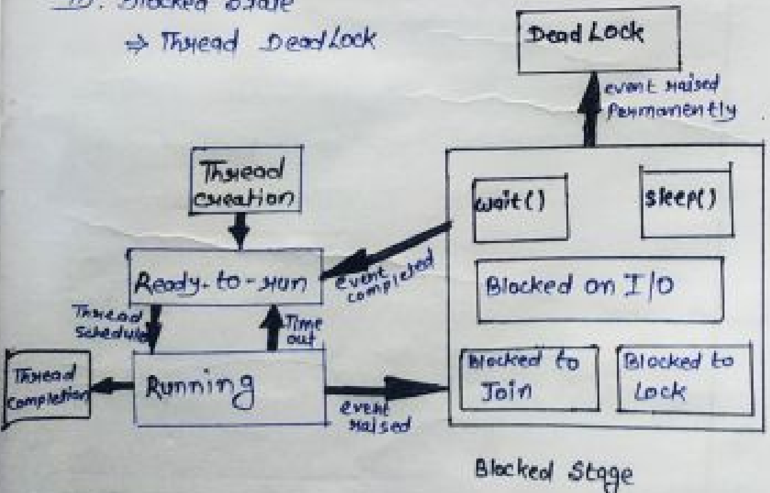
iii). notifyAll() →

→ notifyAll() method also unlock the resource and send the msg to all the multiple waiting threads.

Life-Cycle of thread

Life cycle of threads demonstrates different stages of thread from "thread creating to Thread termination" and "Thread Creating to Thread Completion".

1). Thread Creation
2). Ready-to-run
3) Running
  i). Thread completion
  ii). Blocked State
    → Thread DeadLock



Blocked Stage

Blocked State → (Thread Live-Lock)

→ The temporary blockage of thread

→ if any one of the following event is raised, then the Thread is under Blocked stage.

i). Wait() → Used to Block the thread, until it receives msg.

ii). sleep() → Block the thread on some timer.

iii). Blocked on I/O → Incomplete IO operation.

iv). Blocked to Join → One thread will wait to join the operation of another thread.

v). Blocked to Lock → The Incomplete locking operation.

Dead Lock

The permanent blockage of thread.

→ if any event of blocked state occurs permanently then the thread is under Dead_Lock.

Thread Scheduler

Thread Scheduler is an Internal algorithm to organize threads from 'Ready-to-run to Running' State based on the following algorithms:

1). Time Slicing Algorithm
2). Priority Based Algorithm

→ We use setPriority() method to assign priorities to the threads.

Syntax :- `t1.setPriority(6);`

Note :- The Thread Priorities must be from 1 to 10
  1 - Least Priority
  10 - Highest Priority
  5 - Normal Priority (Default)

→ We use getPriority() method to display the Priority of threads.

Syntax → `int p = t1.getPriority();`

Polymorphism many - forms

The process in which the programming components having many forms is known as PolyMorphism.

• Polymorphism categorized intoo two types :-

1). Dynamic Polymorphism

2). Static Polymorphism

1). Dynamic PolyMorphism

• The Polymorphism at execution stage is known as Dynamic PolyMorphism or Runtime PolyMorphism.

Ex:- method Overriding Process

Note:- Through method Overriding process we can provide more than one form to a method at runtime, because of this reason method Overriding comes under Dynamic PolyMorphism or Runtime Polymorphism.

2). Static PolyMorphism

⇒ The PolyMorphism at compilation stage is known as static PolyMorphism or Compiletime PolyMorphism.

Ex:- method Overloading Process

→ method Overloading means, more than one method with same name, but differentiated by their Para_list or Para_type.

```
Ex:-  add(int, int)
      add(int, int, int)
      add(int, float)
```

Private

⇒ The following are the private programming components

a). Private variables
b). Private methods
c). Private constructors
d). Private classes

⇒ There is no concept of Private blocks, Private Interfaces and Private abstract Classes.

a). Private variables → Accessed by the methods of same class.

Private variables are used in Bean classes and POJO classes.
(Plain old java object).

b). Private Methods → Accessed by the Non-Private methods of same class.

c). Private Constructors →

Private constructor is executed when the object is created in the same class, where the constructor is declared.

Single Ton Class

The class which generate only one object inside the same class.

d). Private Classes :-

Private classes are declared only as InnerClasses, which means outerclasses cannot be declared as private.

Wrapper Classes

Wrapper Classes are from 'java.lang' Package and which are used to make Primitive DataTypes available in the form of objects.

→ Every PMDT will have its own Wrapper class and there are 8 WrapperClasses

byte Byte
int Integer } → Wrapper Class
long Long (First letter is Capital)

Boxing Process :- The process of binding PMDT into WrapperClass objects.

Note:-
Boxing process is performed using Constructors.
List of Constructors from WrapperClasses :-

Wrapper Class	Constructors
Byte	byte, String
Short	short, String
----	------

Auto Boxing

Boxing process performed automatically.

→ AutoBoxing process means assigning PMDT values to Non-PMDT variables.

UnBoxing

The process of taking PMDT values out of Wrapper class objects.

Methods→
```
public byte byteValue();
Public int IntValue();
```

Auto UnBoxing

AutoUnBoxing Process means assigning NPDT variables to PDTV.

Note :-
All the Wrapper Class Objects are automatically Immutables objects.
(Secured Objects)

Strings

→ string is a sequenced collection of characters represented in double quotes.
Ex.→ "Lok", "java"

⇒ The following are the classes from java.lang package used to create string objects :-
1). String Class
2). StringBuffer Class
3). StringBuilder Class

1). String Class

⇒ The Objects generated from 'java.lang.String' class are Immutable objects.
⇒ 'java.lang.String' class is having 15 Constructors.
⇒ We use 2 syntax to create String objects :-

Syntax 1 → using String literal process
```
String s1 = "java";
```

Syntax 2 → Using new Operator Process
```
String s2 = new String("Program");
```

String Constant Pool

The Partition of Heap_Area where String objects are created is known as String Constant Pool.

(i). In String literal Process the execution control will check the 'String Constant Pool' is any object having same object data,
⇒ if Object not available, then create new_Object.
⇒ if Object is available, then use the reference of existing object without creating new object.

(ii). In new operator Process, one object is created Part of Heap_Area and the object will hold the reference of object created in String Constant Pool.

String Concatenation Process

⇒ The Process of combining multiple strings into a single String is known as String Concatnation Process.

Note:-
In concatnation Process, separate object is created to hold concatnated strings.

Factory Method :-

The method which hide object creation process from the user is known as factory method.

Types :- i). instance factory method
ii). Static factory method

String comparision process

The process of comparing two strings.

ways
✓ i). Using equals() method
✓ ii). Using compareTo() method
✓ iii). Using 'is equal to' (==) operator

✓ i). Using 'equals()' method

⇒ equals() method will compare two strings and generate boolean result.

⇒ In real Time equals() method is used in authentication process.

✓ ii). Using 'compareTo()' method

⇒ CompareTo() method will compare 2 strings and generate int result. (used in Sorting Process)

✓ iii). Using 'is equal to' (==) operator

⇒ 'is equal to' (==) operator will compare the object references and which will not compare the contents of an objects.

"trim()" → is used to remove the spaces before and after the string.

Note :- 'is equal to' (==) operator will compare the object references and which is not preferable to use on Non-Primitive data types, because which generate wrong result.

---

3). "StringBuffer" class

'StringBuffer' is a Built-in class from java.lang package and which generate 'Mutable Objects'. (can be modified)

⇒ The following are the four constructors from java.lang.StringBuffer. a. Public java.lang.StringBuffer();

case 1 :- Using 'StringBuffer()'

Syntax :- StringBuffer sb = new StringBuffer();

⇒ In this syntax the object is created with the default capacity 16 and the capacity increases dynamically at runtime by "dubling the capacity and adding 2".

16 ⟹ (16+16+2) ⇒ 34 ⟹ (34+34+2) ⟹ 70 ⟹ ....

⇒ We use "append()" method → to add the data to the StringBuffer object.

case 2 :- Using StringBuffer(int)

Syntax :- StringBuffer sb = new StringBuffer(6);

⇒ In this syntax the StringBuffer object is created with the capacity equal to the value passed as Parameter while object creation.

case 3 :- Using 'StringBuffer(java.lang.String)'

Syntax :- StringBuffer sb = new StringBuffer("Lok");

⇒ In this syntax the StringBuffer object is created with capacity equal to the "sum of default capacity and length of string passed as parameter".

Arrays

⇒ The sequanced collection of elements of same datatype is known as Array.

⇒ In Arrays, the elements are organized based on index values.

⇒ Arrays in java are sequanced collection of similar Objects, which means objects of same class.

Types :- a). Single Dimensional Arrays
b). multi Dimensional Arrays

Object Array

The Array which is declared with 'java.lang.Object' class is known as Object Array.

Note → java.lang.Object is the PClass of all the classes and the Object Array can hold dis-similar objects, which means objects of different classes.

Syntax → `Object o[] = new Object[3];`

Jagged Array

The Array which hold Array objects is known as Jagged Array.

Dis-Advantages of Arrays

⇒ Array size once defined cannot be modified at runtime or execution time, because of this reason Arrays are not per preferable in realtime to hold dynamic data or runtime data.

Note :-
This Dis-Advantage can be Overcomed using 'Collection<E>'.

Object Array → Hold dis-similar objects, which means object of different class.

Object o[] = new Object(3);

Wrapper Classes

Wrapper Classes are from 'java.lang' Package and which are used to make Primitive DataTypes available in the form of objects.

→ Every PMDT will have its own Wrapper Class and there are 8 WrapperClasses

byte Byte
int Integer
long Long
} → Wrapper Class
(First letter is Capital)

Boxing Process :- The process of binding PMDT into WrapperClass objects.

Note :-
Boxing process is performed using Constructors.
List of Constructors from WrapperClasses :-

Wrapper Class	Constructors
Byte	byte, String
Short	short, String
-----	------

Auto Boxing

Boxing process performed automatically.
→ Auto Boxing process means assigning PMDT values to Non-PMDT variables.

UnBoxing

The process of taking PMDT values out of Wrapper Class objects.

Methods → public byte byteValue();
Public int IntValue();

Auto UnBoxing

AutoUnBoxing Process means assigning NPDT variables to PDTV.

Note :-
All the Wrapper Class Objects are automatically Immutables objects.
(Secured Objects)

Strings

→ string is a sequenced collection of characters represented in double quotes.
Ex. → "Lok" , "java"

⇒ The following are the classes from java.lang package used to create string objects :-
1). String Class
2). StringBuffer Class
3). StringBuilder Class

1). String Class

⇒ The Objects generated from 'java.lang.String' class are Immutable objects.
⇒ 'java.lang.String' class is having 15 Constructors.
⇒ We use 2 syntax to create String objects :-
Syntax 1 → using String Literal process

```
String s1 = "java";
```

Syntax 2 → Using new Operator Process

```
String s2 = new String("Program");
```

String Constant Pool

The partition of Heap_Area, where String objects are created is known as String Constant Pool.

(i). In String Literal Process the execution control will check the 'String Constant Pool' is any object having same object data,
⇒ if Object not available, then create new Object.
⇒ if Object is available, then use the reference of existing object without creating new object.

(ii). In new operator Process, one object is created Part of Heap_Area and the object will hold the reference of object created in String Constant Pool.

String Concatenation Process

→ The Process of combining multiple strings into a single String is known as String Concatnation Process.

Note :- In concatnation Process, separate object is created to hold concatnated Strings.

|| final ||

⇒ The following are the final programming components ;

1). final variables

2) final methods

3). final Classes

⇒ There is no concept of final blocks, final constructors, final interfaces and final abstract classes.

1). final variables → These final variables must be initialized with values and once initialized can not be modified.

→ The final variables in classes can be Initialized using constructors.

2). final methods → final methods can not be Overrided.

3). final classes → final Classes can not be extended. (No inheritance for final classes)

Note :- Using final programming components we construct 'Immutable Classes'.

Immutable class

Rules :-

①. The class must be final class.

②. The variables within the class must be private and final variables.

③. The class must be declared with only 'Getter' methods.

④. These 'Getter' methods must be final methods.

Getter Method → The method which are used to get the data from the objects.

Setter method → The method which are used to set the data to the objects.

Immutable Objects

The objects which are generated from 'Immutable classes' are known as Immutable objects.

✡ → These Immutable objects once created can not be modified and which are known as secure objects.

TypeCasting

⇒ The process of converting one datatype value into another datatype value.

→ Not applicable

PMDT → Non PMDT ×

NonPMDT → PMDT ×

case 1 → TypeCasting Process on Primitive DataTypes :-

1). Widening Process → Lower value to Higher

2) Narrowing Process → Higher into Lower data type

case 2 → TypeCasting process can not be applied on Non-PMDT, but in inheritance process we can perform the following :-

1). Generalization Process

2). Specialization Process

i). Generalization Process :-

In Generalization Process one object is created holding all the members of ParentClass and only Overriding members from the ChildClass.

Syntax 1).
```
PClass ob = (PClass) new CClass();
```
2).
```
Interface_name ob = (Interface_name) new Impl_Class();
```

2). Specialization Process :-

. The process of constructing ChildClass by taking one feature from the PClass is known as.

Syntax :-
```
char ch2 = (char) z ;
CClass ob = (CClass) new PClass();
```

→ Specialization can not be applied on Interfaces.

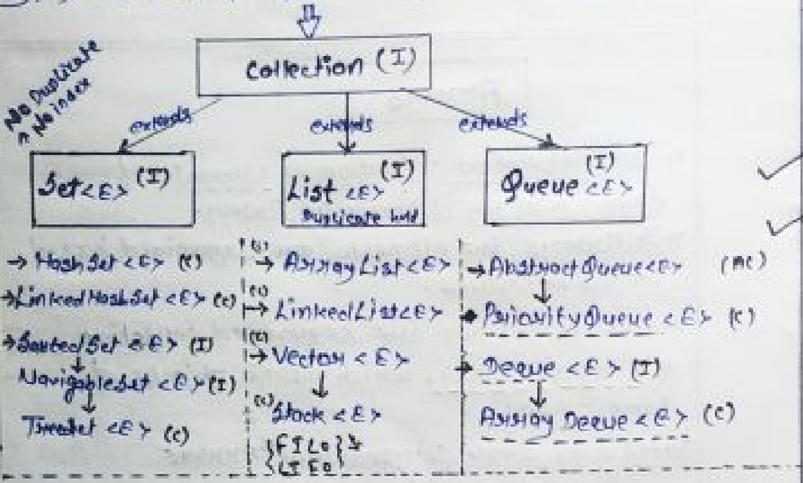
Collection Framework

Collection<E>

'Collection<E>' is an Interface from 'java.util' package and which is root of JCF.

⇒ This 'Collection<E>' interface is extended to the following Sub-Interfaces :-

1). Set <E> → (add(), remove(), clear(),
2). List <E> → we can Perform (add, remove, clear, addAll();)
3). Queue<E> → java.util, (add, offer, element(), Poll, Peek, remove)



Framework :-

The structure which is ready constructed and available for application development is known as Framework.

Generic Programming Components

The Programming components which are ready to accept any type of data at runtime.

following are Generic Programming Components →

①. Generic Types :-

The types which are ready to accept any type of data at runTime.

List :- T → Type
E → Element
K → Key
V → Value

②. Generic Method :-

The methods which are ready to accept any type of data as parameter. <T> return_type method_name (T)

③. Generic Classes :-

The objects of Generic Classes can hold objects of any type.

class Class_name <T>

④. Generic Interfaces :- Generic Interfaces are Implemented to Generic Classes.

1).

⇒ Set<E> organizes elements without 'index values' and can not hold duplicate elements.

⇒ Set <E> implemented into the following :-

a). HashSet <E> → No Order (Random)
b). LinkedHashSet <E> → Insertion Order
c). TreeSet <E> → Ascending Order automatically

2). List<E> 'Index values' 'can hold duplicate elements'

→ 'List<E>' organizes elements based on index values

→ List <E> is implemented into the following

a). ArrayList <E> → Sequenced & Non-Synchronized class
, al.add (new Integer(12)); sysout (al.toString);
b). LinkedList <E> → Non-Sequenced & Non-Synchronized class
c). Vector <E> → Sequenced and Synchronized class

Note :-
⇒ In realTime ArrayList <E> is not preferable in the applications, where we have more number of add() and remove() operations.

⇒ This Dis-Advantage of ArrayList <E> can be Overcomed using LinkedList <E>.

LinkedList :-

In LinkedList <E> the elements are available in "nodes".

→ This LinkedList <E> node is divided into the following parts →

i). Previous Node Address
ii). Data
iii). Next node Address

⇒ In realTime LinkedList <E> is preferable in applications, where we have more number of add() and remove() operations.

Vector <E> :- sequenced

Vector <E> is Synchronized and thread-Safe class

In realTime vector<E> is used in Connection Pooling Process and also used in multi-threading applications. (addElement, firstElement, insertElementAt, setElement

Stack <E> :-

Stack <E> is a childClass of vector <E> and which organizes elements based on the algorithm first-in-Last-out or Last in-first-out.

Stack < Integer> st = new Stack < Integer> ();
sysout (st.tostring

The following some methods of Stack <E>

push(E); → Used to add Element to the Stack <E>.

pop() → delete element

peek() → To display the element from the top-of-Stack

empty() → To check the stack is empty or not.

search() → To search element and display the position of an element.

⇒ In realTime Stack <E> and Queue <E> is used part of Algorithmic design.

---

3). $\boxed{\text{Queue <E>}}$ → FIFO LILO ✓

Queue <E> organizes elements based on algorithm first-in-first out or Last in - Last out.

⇒ PriorityQueue <E> is the implementation of queue which organizes elements based of elements Priority.

Note :- Pq. add (new Integer(1234)); Sysout (Pq. toString());

⇒ In realtime PriorityQueue <E> is used to hold multiple thread (users) executing on priorites.

Deque <E>

⇒ Deque <E> means Double-Ended-Queue and which is Sub-Interface of Queue <E>.

⇒ In Deque <E> the elements are organized on both ends, which means we can add elements on both ends and we can delete elements on both ends.

⇒ The following are the implementations of Deque <E> :-

a). $\boxed{\text{ArrayDeque <E>}}$ ad.add(new Integer (12)); ad.removeFirst(); ad.removeLast(); Sysout (ad. toString()); ⇒ Elements in Sequence

b). $\boxed{\text{LinkedList <E>}}$ (offer(), Poll(), Peek()) ⇒ Elements in Non- Sequence

Limitation of Collection <E>

⇒ In the process of holding DataBase table, data Collection <E> cannot differentiate Primary-key and Non-Primary_key values.

Note :-

This Limitation of collection <E> can be overcomed using map <K, V>.

$\boxed{\text{Map <K, V>}}$ Put() to add

⇒ Map <K,V> is an interface from java.util which organizes elements in the form of key-value Pairs.

K – key (Primary-key)

V – Values (Non- Primary key values)

⇒ The following are the implementations of map <K, V> :-

a). HashMap <K,V> (c) → Random order

b). LinkedHashMap <K,V> (c) → Insertion order

c). TreeMap <K,V> (c) → Ascending order

d). HashTable <K,V> (c) → Random order

HashMap And Hashtable <K,V> :- Random Order

Organizes elements without any order.

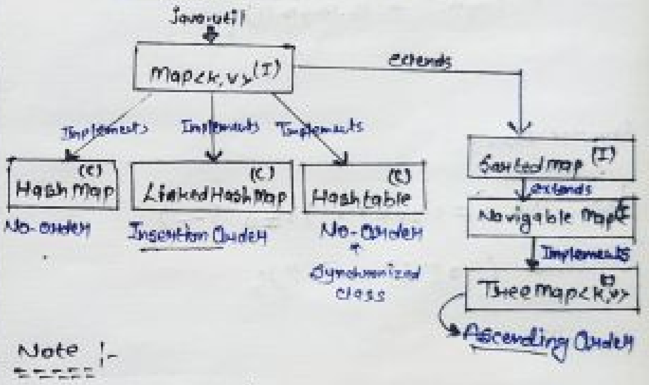
✓ LinkedHashMap <K,V> → Insertion Order

or

✓ TreeMap <K,V> → Ascending order Based on key

Hashtable <K,V> → Synchronized class ✓

remaining classes are Non- Synchronized classes.



Note :-

In the process of organizing DB-Table data using map<K,V>, we must construct one User defined class with variables equal to the Non – Primary _ key _ Values.

add() → Used to add data to Collection <E> Objects.

put() → Used to add data to Map<K,V> Objects.

Hashmap → hm.put (1, "Lok");
Sysout ("value of 1:" + hm. get (1);
Hashmap< Integer, String> hm = new HashMap <Integer, String>();

Serialization process :-

The process of converting Object state into Binary Stream.

Performed using "writeObject()"