Object Oriented Programming (OOPs)

OOPs is a Design principle which helps us connect/map code instructions with real world entities making the code short
and easy to understand

------Topics
OOPS Basics
classes and Objects
Constructor
constructor overloading
Blocks
this, final
Access Specifiers
java bean class
--Encapsulation
--Inheritance(Is-A Relationship)
Method overloading
Method overriding
type casting
generalization and specialization
--Polymorphism
Abstract class
interfaces
--Abstraction
has-a Relationship

---
class type definition is made up of two parts
1)class declaration/signature
2)class body

The class declaration specifies the
--accessSpecifier modifier class ClassName.

The body specifies variables and methods
Anything which is defined inside the body
of the class is known as members of the class

the list of variables defined inside the class is

known as member variable
The list of methods defined in the class body is
known as member methods

The members of the class are classified into 2 types
1)static member
2)non static members

1)static members : static members are declared with static keyword
Static members have single copy in the memory
-Static does not mean constant ,we can modify the value but they have single copy in the memory

2) non static members : non static member are declared without static keyword.
we can create multiple copy of the non static members by creating Objects

---

Q) can i write any number of class files in one source file ?
ans: yes , when we compile --for every class byte code will be
generated (if 3 classes 3 byte codes)

A class is eligible to execute only if class contains main method
Execution starts with main and end with main method

Note:
- -->In a single sourcefile we can define any number of java class type, each class name should be unique
- -->the compiler will generate separate byte code for each class
- ->In one sourcefile i can create only one "public" class
- ->with which name i create my sourcefile there only i have to write my main method for smooth execution

Accessing static members of one class in another class

1)Accessing static variable
syntax:
Classname.staticvariablename

2)Accessing static method

syntax:
Classname.staticmethodname()

Accessing Nonstatic members of one class in another class

1)Accessing Nonstatic variable
syntax:
new Classname().nonstaticvariablename

2)Accessing nonstatic method
syntax:
new Classname().nonstaticmethodname()

Members of a class can be accessed in any other class
by using . operator(Dot operator)

Local Variable:
A variable declared inside a method body or constructor body is know as local variable
(And local variables can be accessed only inside those methods)
local variable doesn't have static ,non-static concept

Global Variable :
A variable declared outside the method block and inside the class block is know as global variable
(And it can be accessed anywhere inside the class block)

Object

An Entity having states(Attributes/properties), behaviour and
unique identification is know as Object
(All real word entities are Object)

States : represent the information about the object
(variables).
The sates are also known as data members or attributes or fields

Behaviour: represent the actions which must be performed on the object (methods).

A class is a definition block used to define the states and behaviour
Class:
"Class is A BluePrint of an Object".

Object Creation /s instantiation (Object/instance)

Reference Variable

Its a non primitive type variable declared using java class type

(Classtype) Ex: Tesla t1 ; here Tesla is class ,"t1" is reference variable

"ReferenceVariable Contains ADDRESS of an Object"

The referencevariable is used to refer Object hence it is also know as Object reference
Using Object reference we can access datamembers and methods of the Object

---
Initialization of Object States/attributes

1)default Initialization
2)using Constructors
3)Using Blocks
4)using methods

Default values are only applicable for global variables ,
Default values are NOT-applicable for local variables

Constructor
-->used to Initialise attributes of an Object
--->It is internally called "at the Object creation time"
   (Not before , not After)
--->Constructor is a special member of class used for initialization of attributes of an Object

syntax:

Note:Constructor name will be same as class name
it is similar to method but constructor don't have any return type not even void.

Constructor is a special member of class used to initialize states of an Object ,during Object creation Time
The constructor is always called whenever we create the Object of a Class

When We define a Constructor ,The following Rules must be followed
1)The constructor name should be same as class name
2)The constructor should not have Return Type(not even void)
3)modifiers should not be used in constructor declaration
4)A constructor can be declared with parameter or without parameter
5)Defining a constructor in a class provides the benefits of initializing the Object states during the Object creation,
It makes Object Ready to consume

*** "this" keyword

java always refers the current Object through "this" keyword
The Object by which method/Constructor is invoked in known as current object
The current Object details is stored in 'this' Keyword

"this" keyword must be used either in non-static method of a class or Constructor

"this keyword holds the ADDRESS of Current Object"..

Constructor OverLoading

Defining more than one Constructor with different parameter-list is known as Constructor Overloading

The Parameter list should differ in any one of the following
1)parameter type
2)parameter length

3)Sequence of parameter

Constructor Overloading helps us create Object with different set of data.
The Overloaded Constructor are executed based on the parameter

---
Blocks

A block is a set of instruction used for initialization
----Blocks are categorized into two types
1)static block
2)non static block

Static Block/Multiline static initializer
static block is a set of instruction used for initializing "only static variables"

```
syntax:        static {
                   //set of instruction

                  }
```

--

#Non static Block //Multiline non-static initializer

```
syntax:
       {
     //set of instruction
      }
```

The non static block is set of instruction used to initilize both static
 and non-static variables
(but it is not good practice to initilize static variables in non static block)
We majorly use non static block to initilize non static variables only

Non static Blocks are executed at the time of Object Creation
When we have multiple non static blocks they also execute sequentially

Important:
1)static blocks runs only Once
2)Non static block runs whenever Object is created

Interview Question
In one class , we have both non-static Block and Constructor
if we create Object and initialize parameter both are called
,which will run first ? and which value will be there in the Object?

Ans) ---First non-static blocks will run
-----Then constructor will run
-----Constructor will run later so the value initialized in constructor will be stored

---
final Variable

Any variable declared by using 'final keyword' is known as final variable
--->final variables are 'constants' in java language
--->final variable must be initialized only once , It cannot be reinitialized

---
Access Specifiers

1)Public
If you declare any entity as public, then it can be accessed
by the classes present in the same package or different package
Public entities will have highest visibility and lowest security

2)Protected
If you declare any entity as protected ,then it can be accessed by the classes present in same package
Protected entity can be accessed by other classes present in different package through inheritance and creating the object of SUBCLASS(Child class) only.

3)Pkg-level (default)
if you declare any entity without using any access specifiers
then it is considered as pkg-level member (default member)
If you declare any entity as pkg-level ,then it can be strictly accessed only by the classes present in same package

4)private
If you declare any entity as private then it can be accessed only by the class within which they are declared

Private entities will have highest security and lowest visibility

---
Encapsulation

â€”â†’Binding the data members and related operation into the class is known as encapsulation

â€”â€”>Encapsulation is defined as the wrapping up of data and methods under a single unit is Know as Encapsulation .It also implements data hiding.

*The encapsulation specifies that a data and its related operation must be in the class body
*It also specifies that the data members should be protected(Hide) using access specifiers

Java bean class
(java bean class Format is Perfect Example for Encapsulation)
The characteristics of java bean class
1)The class must be public
2)the member variable should be private
3)constructors must be public
4)the class must define public getters and setters method

- the getters method are used to get the private variable values.
- the setters method are used to change/modify the private variable values.

get and set is not a keyword its a format

Getter Method
to get the value of private attribute of object

Format:
```
datatype  getAttributename()
{
return attributename;
}
```

Example:
```
private int age;

//getter method
```

```
int getAge(){
      return age;
    }
```

Setter method
Used to change value of private attribute of an object

Format:

```
void setAttributename(datatype (naya)attributename){
this.attributename = (naya)attributename;
}
```

Ex:
```
void setAge(int age){
this.age=age;
}
```

Inheritance
1)Is-A Relationship (Inheritance)
2)Has-A Relationship

[base Class or/ super class or/ parent class]

[derived class or/ sub class or/ child class]

- -extends Keyword is used for inheritance
--implements ---we will study in interface topic

""One class acquiring the properties of another class is called as Inheritance in java""

The class from which members are inherited is know as
Super class/Parent class
The class to which the members are inherited is know as sub class/child class

note:
Subclass inherits non static members from superclass
The class can inherit from another class by using "extends" keyword
Private non static members of super class will not be inherited to subclass

because of private access specifier
The constructor of super class will not be inherited to subclass

- **In order to achieve inheritance we make use of keyword called as "extends"**
- **We also refer Inheritance as Is-A relationship**
- **Blocks and constructor of the superclass will not be inherited to subclass (variables and methods are inherited)**

Implicit constructor call

Whenever we write inheritance program the subclass constructor must call the super class constructor to initialize the attributes

If the superclass contains non-parameter constructor ,the subclass constructor makes Implicit call.

Explicit Constructor call

The subclass constructor can make a call to super class constructor by using (super())super calling statement.
If the superclass contains parameterized constructor ,the subclass should make Explicit call. (using super())

Constructor calling statement
super() -----This makes a call to the superclass constructor
it can call with parameter or without parameter

Rules :
1)super calling statement must be used only inside the constructor body
2)Super calling statement(super()) must be the first statement in the constructor body
3)in one constructor body only one super calling statement is allowed

Constructor Chaining
1)"A phenomenon of running more than one constructor to initialize the attributes of an object is called as Constructor Chaining"
2)the constructor chaining should happen whenever we write inheritance program

Why Multiple inheritance is Not supported in java Class type

Ans) Java doesnâ€™t support multiple inheritance because constructor chaining will not happen from diff class, since more than one constructor calling statement is not supported ,It leads to ambiguity of Diamond Problem
---

Method Overloading

A class defining multiple methods with same name but with different parameter list
is known as method Overloading,
The parameter list should differ in any one of the following
1)parameter type 2)parameter length 3)parameter sequence

---In one class we can Overload both static and non static methods
---We can overload in the subclass with inherited methods

Whenever we want to perform same operation on different types of data or set of then we should apply method overloading

note:
We can overload main method,But the execution begins only if the main method (String[] args) String array parameter.

- --"The method overloading is used to achieve compile time Polymorphism"

Java Interview Questions!!!!!

What is a Variable ? and Explain different types of variables
What are data types and Explain different types of data types with there sizes
Explain this keyword
What are default values
Difference between Static and Non static
What are Blocks and Explain different types of Blocks
What are constructors and Explain constructors overloading
what are methods and Explain method Overloading
what is Encapsulation?
what is Inheritance?
Explain final keyword ?
Explain super calling statement with example
what are access specifiers?
Explain constructor chaining.

what is method overriding?
what is TypeCasting?
 a)Upcasting
 b)Downcasting
What is Generalization and Specialization

What is Polymorphism
a)Compiletime Polymorphism(method Overloading)
b)Runtime Polymorphism(method Overiding)

Whta is Abstraction?
a)what is Abstract class ,abstract method
a)what is Interface

Method OverRiding

"Inheriting method from superclass and changing its implementation in the subclass according to the subclass specification is known as method Overriding"

When a method inherits a superclass method with same declaration it is called override ,It is used to change/modify the method body..

While overRiding a method ,the subclass method should be written with same declaration defined in superclass

We can override only non static methods
The static methods of superclass cannot be overridden in the subclass, because the static members will not be inherited in subclass

The non static methods having private access specifier in the superclass cannot be Overridden ,the private access specifier restricts the access only to that class.

---Method Overriding is an example for "Run time Polymorphism".

Can we Overload and Override at the Same time?

Type Casting

Casting one type of information to another type is known as typecasting
The typecasting is divided into two Types
1)primitive typecasting and 2)non primitive typecasting

1)Primitive typecasting--Converting one primitive type of datatype into another primitive type of data is known as primitive typecasting
Ex: int x = (int)95.65; //narrowing
It is also known as data-type casting
It is further classified into two types
1)Widening 2)Narrowing
Casting lower datatype to any of the higher datatype is known as Widining
Ex: double d = 50; //int type data stored in double container

Widening is done by compiler ,it is known as autowidening /implicitly
The widening can be done explicitly in the code(user done ,but Widening will be done implicitly)

Narrowing : casting higher data-type to any of the lower DataType is known as Narrowing
The narrowing should be performed explicitly in the code ,Whenever we performing narrowing it leads to DATALOSS.

Non primitive typecasting
It is also known as class-Typecasting
Casting one class type to another class
type is known as class typecasting

It is further classified into two types
1)Upcasting
2)downcasting

The class type casting can be done only for the
classes which has Is-A relationship

1)Upcasting:
a)reference variable must be of superclass type
b)object must be of subclass type

casting subclass type to superclass type is know as "Upcasting"
Upcasting can be performed automatically (by compiler)
or we can do it Explicitly
When upcasting happens the subclass Object behaves like
superclass, the subclass Object shows the properties of superclass
,Subclass properties are hidden
The subclass Object can be casted to any of the superclasstype

2)Downcasting:

a)Object must be of super class type
b)reference variable must be of subclass type
(For downcasting upcasting is mandatory)
casting super classtype back to subclass type is
known as "Downcasting".
The downcasting must be done Explicitly in the program
The downcasting should be performed only on the object which is upcasted
orelse we will get "ClassCastException".

Generalization and Specialization

Generalization
Defining a function/method which operates on different
types of Object is known as Generalization

To achieve generalization the method parameter
must be of superclass type
for such methods we can pass subclass objects
The subclass type will automatically cast to superclass type

Specialization

to achieve Specialization the method/function parameter
must be of subclass type.
For such method we cannot pass different type of Objects

Generalization is used to work on common properties
and functionalities where as Specialization is defined
to work on specific properties of subclass

Polymorphism

Ability of a method to behave differently
,when different Objects are acting upon it,
is known as polymorphism

(Polymorphism means many forms)

An Object showing different behaviour
at different stages of the life cycle is known
as polymorphism

There are two types of polymorphism
1)compile time polymorphism(method Overloading)
2)Run time polymorphism(method Overriding)

Method Binding
Associating/mapping the method declaration/signature
to its method Implementation/body is called
as method Binding

Memory Allocation in java

The JVM uses 4 memory area during execution
1)Heap area 2)class area 3)method area 4)stack area

1)Heap area: heap area is used to store objects , the objects created in program gets a memory allocation in heap area
In other words Non static members of the class are loaded in the heap area

2)Class area / class static area: the class area is used to store the static members of the class

For each class created in the project a separate area is reserved in the class area and that area is known as static pool
Static pool contains only static members of the class

3)The method area stores ,the method body statements But the method declaration part of a method is always stored in either heap area or class area

If a method is non static method , the method declaration part/details is stored in heap area , if the method declaration is static, the declaration part /details is stored in class area.

4)The stack area is used for execution purpose
When a method is called , the method enters the stack memory/area executes the statements defined in the method body , after finishing of execution it exit from the stack area

JVM does all execution , Before it does execution it has to allocate memory

---

1)Compile time polymorphism

------> compile time polymorphism is
achieved with the help of method Overloading
Compile time polymorphism is referred as
Early Binding or static binding

Method Binding is happening in the compile time
,hence we call method Overloading as early binding
or static binding

Once the binding is done by the complier it cannot be rebinded
,Hence it is known as static polymorphism

Method Overloading is an Example for compile time polymorphism

In method Overloading ----> Out of so many methods, which method implementation/body should get executed is decided by the complier during Compile time....

Run time Polymorphism

----->Run time Polymorphism is achieved with the help of

1)Is-A relationship(Inheritance)
2)Method Overriding
3)Upcasting

1)When we call a Overridden method on a superclass reference,
the method implementation which gets executed is
dependent on the subclass acting upon it
Ex Animal a1 = new Cat();
a1.sound(): //Meoww Meoww

2)Out of so many overridden methods , which method implementation
should get executed is decided by JVM at Runtime
,Hence we call it as Runtime Polymorphism

3)Runtime polymorphism is also called as
late-Binding/dynamic-binding/ Dynamic method Dispatch

Note: If we call a Overridden method on a Superclass reference,
always the Overridden method (Subclass)implementation
only gets executed
Ex: Animal a1 = new Dog(); //upcasting
a1.sound(); //Boww Boww

Abstraction

â€œHiding the internal implementation logic of an Object(or application)
and showing only object Functionalities is known as Abstractionâ€•

The abstraction specifies that don't show internal
implementation-logic and show only object functionalities.

1)abstract is keyword which is used with class and method
2)A class which is not declared using abstract keyword is
called as concrete class
3)Concrete class can allow only concrete methods
4)A class which is declared using abstract keyword is called as
abstract class
5)abstract class will allow both abstract methods and concrete methods
6)Concrete class will not allow abstract methods
7)concrete methods will have both method declaration and method
implementation/body
8)abstract method will have only declaration part
but "No method body/implementation"

9)all abstract methods should be declared using abstract keyword

# Contract of abstract
or(what should we do when a class extends abstract class)

1Rule) When a class inherits an abstract class ,we have to override all the abstract methods and give implementation/body

2Rule) When a class inherits an abstract class ,and we don't want to override the inherited abstract method ,then make the subclass as abstract class

---
â€‹
Important Note:
1)can a class inherit an abstract class?
----->Yes
2)Can we create Object of abstract class
-----> No ...We cannot create Object of abstract class

-->abstract method cannot be private(if it is not
accessible outside the class how can i override and
give implementation so it cannot be private)

4)abstract methods cannot be static(static methods cannot be inhereted)
5)abstract methods cannot be final(because final methods
cannot be overridden)
6)can abstract class have constructors?
------->yes but it has to be invoked by the subclass-constructor
either implicitly or explicitly using super();

We can write abstract methods as well as concrete methods in abstract class , So we cannot achieve 100% abstraction using abstract class ,Hence we use interface to achieve 100% abstraction

---
Interface

1)interface is a java type definition which
has to be declared using " interface " keyword.
2)interface is a medium between two systems, where
one system is client/user another system is

Object with services
3)syntax:
interface InterfaceName{
//set if instruction
}
4)Interface can have variables ,Those variables are automatically
public static and final.
5)interface can achieve Is-A relationship with
class using implements keyword

6)Interface can allow only abstract methods, those methods
are automatically public and abstract.
7)when a class implements an interface, mandatorily we should
override the abstract methods

8)while overriding the method, access specifier should
be same or of higher visibility
ex: Public ----to public is Correct
public ------protected ,default ,private is Wrong

9)A class can implement any number of interfaces
(multiple inheritance)

10)A class can extend 1 class and implement
any number of interfaces

11)Interfaces doesn't contain Constructors
12)we cannot create object of an interface
13) In interface we can write static concrete methods(main method)

Example for abstraction using Interface

Abstraction
1)"The process of hiding the internal implementation
details(unnecessary details) and showing only the object
functionalities(necessary things/Behaviour)to the user with
the help of abstract class or interface is called as abstraction"

2)Abstraction can be achieved by the following rules below
a)abstract class/ interface
b)Is-A relationship(inheritance)
c)upcasting

d)method Overriding

Has-A Relationship
(An Object containing another Object)
1)Composition
Imagine there are two classes A and B ,
A class has-A B class in such a way that
A cannot function if B is not functioning
It is called as "Tight Coupling".
Ex: Mobile and Battery
Human and Heart
Human and Oxygen
Car and Tire
(Boys relationship with a girl---> if your not there I m not there)

2)Aggregation
A class has-A B class in such a way that
A can function if B is not functioning,
It is called as "Loose Coupling".
Ex: Mobile and Sim card
Human and food
Human and mobile
(Girl relationship with boy, if not u , someone else)

Exception Handling

Error:---> Error is a mistake or a problem
which occurs during the execution of a program
Error will occur during Compile time (due to Syntax mistakes)
(Error should always be debugged)

Exception-->It is an unwanted Event or interruption which stops the
execution of a program ,where in the below lines of code will
not get executed is called as Exception

-->Exception is a runtime interruption which has to be handled
-->The process of handling an exception is called as Exception handling

try and catch Block

In java we handle an exception using try and catch block
----->The critical lines of code which gives exception should be written inside try block
------> the solution message for exception occurred should be written inside catch block

- ------>try and catch block should always be together
------->catch block gets executed only when Exception occurs

- Syntax:

```
try{
 //critical line of code which can cause Exception
 } catch(ExceptionName refvar){
  //solution message for Exception occurred
 }
```

Note:
1)It is always necessary to handle an
exception with suitable try and catch block
2)We cannot have any executable lines of code
between try and catch block
3)Can we write comments there //?
Yes ...Comments are allowed between try and catch block
because comments are non executable line
4)If we do not know the exception name, we can always handle it
using superclass "Exception"

```
catch(Exception e)
{
}
```

--
Exception Hierarchy

Internal Working When we divide a number by Zero
----In java when we divide a number by zero internally an object of Arithmetic Exception is thrown, The Object thrown can be handled or caught using the same type (ArithmeticException) or Superclass type(Exception type)

```
try {
```

```
            System.out.println(10/0);
  }
  catch(ArithmeticException  e){     // or catch(Exception  e)
            System.out.println("Handled");
  }
```

```

//Specialization----> ArithmeticException e = new ArithmeticException();
//Generalization---->  Exception e = new ArithmeticException();   (Upcasting)
```

Note:

1. One try block can have multiple catch Blocks and that suitable catch block will get executed
2)When we have subclass catch blocks and superclass catch blocks
it is always a good practice to handle the superclass catch block as the last catch block

---

Types of Exception
1)Checked Exception
2)Unchecked Exception

1)Checked Exception: checked Exception are compiler known exceptions and the compiler will force you to handle immediately.
All Checked Exceptions will inherit(extends) Exception class
Ex: FileNotFoundException
InterruptedException etc.....

2)Unchecked Exception : Unchecked Exception are complier unknown exceptions and the compiler will not force you to handle it immediately.
All Unchecked Exceptions will inherit(extends) RunTimeException class
EX: ArithmeticException
ArrayIndexOutOfBoundsException

---

throws
---> throws is a keyword which is used to indicate the caller
about the possibility of an Exception (problem)
--->throws can be used with both checked and unchecked exceptions
but majorly used with checked exceptions
--->throws can be used with both methods and constructors but
majorly used with method declaration/signature

- --> throws is used to transfer the exception from
one end to another end

Note: with the help of throws we can declare multiple Exceptions

Example:
void dance() throws FileNotFoundException, ArithmeticException{
}

//Finally block

Finally Block
finally block is a set of Instruction which gets executed always irrespective of
Exception Occurred or not
Finally Block should be always written with try and catch block.

User Defined Exception Or Custom Exception

--->Based on the Project it is sometimes necessary to create
userdefined exception

Custom Exception
An Exception which is Created by the user or Programmer
explicitly is called as "Custom Exception"

---
throw: --->throw is a keyword which is used to invoke an Object
of Exception type
--->throw is generally used with custom Exception

throw : Syntax

throw ObjectofExceptionType or (throw i1;)
or

throw new Exceptionname();

Example:
10/0---> throw new ArithmeticException(); //Internally
invalid index ----> throw new ArrrayIndexOutOfBoundsException();

Rules for Creating or Working with Custom Exception

Rule 1) Create a Class with Exception name
Rule 2) class should inherit "RuntimeException class" to create Unchecked exception or class should inherit "Exception Class" to create Checked Exception
Rule 3)Invoke the Exception Object using "throw" keyword, handle it using try and catch block with suitable solution

0---
```
//Custom Exception    Unchecked Exception
public class InsufficientBalanceException extends RuntimeException{

	//private variable
	private String info;
					//info
	//constructor		//(String   "Paisa Khatam..Insufficient Balance")
	InsufficientBalanceException(String info) {
		this.info = info;
	}

	@Override
	public String getMessage() {
		return info;
	}
}
```

```
package javaBox4;

import java.util.Scanner;

public class ATM {

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

```
		Scanner sc = new Scanner(System.in);
		int balance=10000;
		System.out.println("Enter the amount you like to Withdraw");
		int amt = sc.nextInt();
		sc.close();  //this line will stop taking input from user

		if(amt<=balance) {
			System.out.println("Amount withdrawn successfully");
		}
		else {

			try {
			//	throw new InsufficientBalanceException("Insufficient
Funds");
				//0r                                          //passing info
				InsufficientBalanceException i1 = new
InsufficientBalanceException("Paisa Khatam..Insufficient Balance");
				throw i1;

			} catch(InsufficientBalanceException e1) {
				System.out.println(e1.getMessage());
			}
		}
	}
}
----------------------
```

File Handling or File Programming

The process of Storing the data into a file or reading the
data from a file is called as file handling . The data or information
from a java program can be stored into two platforms
1)Files (using java.io Package)
2)Database (using JDBC/Hibernate)

Important classes present in [java.io Package]
1)File--An abstract representation of file
2)FileReader--
Convenience class for reading character files.
3)FileWriter--Convenience class for writing character files.
4)FileInputStream--A FileInputStream obtains input bytes
from a file in a file system.
5)FileOutputStream--A file output stream is an output stream

for writing data to a File or to a FileDescriptor.
6)ObjectInputStream---An ObjectInputStream deserializes primitive
data and objects previously written using an ObjectOutputStream
7)ObjectOutputStream--An ObjectOutputStream writes primitive data
types and graphs of Java objects to an OutputStream.
8)BufferedReader--
Reads text from a character-input stream, buffering characters
so as to provide for the efficient reading of characters, arrays, and lines.
9)BufferedWriter---
Writes text to a character-output stream, buffering characters
so as to provide for the efficient writing of single characters, arrays, and strings.

Steps to Write the data into a file

step 1) create an instance of file by passing(specifying) the filename
step2)create an Object of FileWriter class and pass the Object
of the File to the constructor of FileWriter
step 3) Invoke Write() by passing the data to the stored file
step 4)Invoke the flush()
step 5)Close the FileWriter object using .close() within
finally block

note:
1)File Writer ---Opens the file in write mode
but the already existing data is Overridden
2)if File is not Present : creates a new file and opens in write mode
---

Steps to Read a data from a File

1)create an instance of file class by specifying the file name
2)create an instance of FileReader class and pass the Object
of File to the constructor of FileReader
3)define a business logic to read the data from a file using read();
4) Optionally close the FileReader Object within finally() block

---
note:

FileReader -----opens the file in read mode
1)File Present : opens the existing file in read
mode and data can be read successfully
2)File Not Present : throws an Exception called
as FileNotFoundException

String class

String is a predefined final class present in java.lang package
String class cannot be inherited because String is a final class
String objects are immutable in nature
String is a class as well as a datatype therefore we refer String as non primitive datatype
default value of String or any class type is null
String can also be referred as set of char enclosed within double cots

---String Object can be created in two different way
1)without new Operator (also called as "String literal")
ex: String s="Murtuza";
//this is creating a String Object without new operator

2)With new Operator
String s1 = new String("Om");

String class was introduced from JDK 1.0
String objects will get stored inside a memory location called as String pool
String pool is divided into two parts
a)String constant pool
b)String non-constant pool
String literal objects will get stored in string constant pool
and constant pool does not allow duplicates

String objects created using new-Operator will get stored inside non-constant pool and non-constant allows duplicates

---

String class also implicitly inherits the object class and has Overridden three methods

1)toString()
2)hashCode()
3)equals()

a)toString() should have returned string representation
of the Object(address) but in String class it is Overridden to
return the data

b)hashCode() should have returned a unique random number
but in string class as it is overridden we get
ASCII or Unicode values

c)equals() should have compared the address or references
of two objects but in String class it is overridden to
compare the data or content

Explain how String is Immutable in nature?

When we reinitialize a String Object , The existing object
is not modified
Rather than that a new Object gets created
The reference variable pointing to the old object will get dereferenced and
starts pointing to the newly created object
therefore String Object are immutable in nature

The mutable version of String are
1)StringBuffer
2)StringBuilder

(Difference between String class, StringBuffer class, StringBuilder class)

1)String (normal String which we use)

> String is a predefined final class present
in java.lang package
It was introduced from JDK 1.0
String is Immutable in nature
String implements --->Serializable, Comparable, CharSequence
Thread safe
String has Overridden toString(),hashCode() , equals()

We Can use + operator for concatenation
Synchronized

2)String Buffer
> String buffer is a predefined final class present
in java.lang package
It was introduced from JDK 1.0
StringBuffer is Mutable in nature
StringBuffer implements --->Serializable, Comparable, CharSequence
Thread safe
StringBuffer has Overridden only toString()
Cannot use + operator for concatenation(instead we should use append)
Synchronized
>

3)String Builder
StringBuilder is a predefined final class present in
java.lang package

> It was introduced from JDK 1.5
StringBuilder is Mutable in nature
StringBuilder implements --->Serializable, Comparable, CharSequence
Not Thread safe
StringBuilder has Overridden only toString()
Cannot use + operator for concatenation(instead we should use append)
Not Synchronized

```
public class Kabutar {

	public static void main(String[] args) {

		String  s ="Kabutar";

		char[] ch = s.toCharArray();

		//reverse of a String
		for(int i=ch.length-1; i>=0; i--) {
			System.out.print(ch[i]);
		}
		System.out.println();
		System.out.println("---------");
```

```
		//Reverse a String without charArray()
		String s1 = "Cherry";

		for(int i=s1.length()-1; i>=0; i--) {
			System.out.print(s1.charAt(i));
		}

	//	length ---->array
	//  length()---->String
		//To check Palindrome or not -->insted of == , use .equals() to
compare
	}
}
```

Java Object Class

Java Libraries
java libraries is a collection of predefined packages,
Each package is a collection of classes and interfaces
each class and interfaces is a collection of variables
and methods

All the predefined packages are present in zip file called as
src.zip
if not src it will have jar file rt.jar (rt== runtime)

Important packages
1)java.lang package = Object class, String class , Thread.....
2)java.util package = Scanner class , Arralist,HashSet..
3)java.io package = File, FileReader, FileWriter

java.lang package
the java.lang package is automatically imported in all the
classes and interfaces

Object Class
Object is a pre-defined class present in java.lang package
Object class is referred as super-most class in java because
every class will implicitly inherit Object class

Important methods present in the Object class

public String toString() -String representation of an object
public boolean equals(Object obj) --to compare the equality of objects
public int hashcode() --JVM generates unique number
public final void notify()
public final void notifyAll()
public final void wait()
public final void wait(long a)
public final void wait(long a, int b)
protected void finalize()
protected Object clone() --return the same object same as the current object
public final Class getClass() --to get the class name of given object

---

1)public String toString()

---------toString() method returns the String representation
of an Object

Syntax:
public String toString()
{
return null;
}

In java when we print the reference variable
(or Object reference) implicity toString() gets called

Format:
   fullyQualifiedClassname@hexadecimalValueofTheHashCode

ex: box18A.Dhuleti@b1bc7ed
    box24.Optimus@133314b

Fully Qualified class name means Considering class
name from package level

example: ClassName FullyQualifiedClassName
          Scanner java.util.Scanner
          Object java.lang.Object
          File java.io.File

---
equals()

equals() generally compares the address or reference of
two Object ,In order to compare the data of two Object
we have to Override equals()

Rules of Overriding equals method
1)Upcasting
2)downcasting
3)Comparison logic

synatx: public boolean equals(Object obj)
{
return false;
}

---
hashCode()

hashCode() return a unique random number for an Object
hashCode() is used to identify an Object uniquely

```
syntax:  public int hashCode(){
            return 0;
           }
```

Collection Frame Work

Data Structure: It is a way of Storing /Arranging /Organizing the data

In other words Data Structure is the Arrangement of a group of Data
Eg: Arrays, LinkedList ,Stack , Queue ,Graph.

Collection
Collection is a predefined interface present in java.util package
It was introduced from JDK 1.2
It is used to store a group of data

Difference between Array and Collection

Array
1)Homogenous in nature
2)Fixed Size
3)No predefined methods
4)Generics not supported
5)no underlined data Structure

Collection
1)Both homogeneous and heterogeneous in nature
2)Dynamic size
3)Many predefined methods available
4)generics supported , < >
5)Every collection will have underlined data structure

Generics â€”-It is a feature used to mention Element Type(Datatype of the Element)
It is represented as <E> element type or <T> type
Using Generics it is possible to Make a Collection as Homogeneous as well as Heterogeneous
---

Collection Hierarchy
-----Leave 1 page.....

IMPORTANT methods present in Collection Interfaces
1)add() -- It is used to add an Object into a collection
2)addAll() --- It is used to add 1 collection into another collection
3)contains()-- It is used to check if the object is present in the collection or not
4)containsAll()--- It is used to check if 1 collection is present in another collection or not
5)remove()--It is used to remove the Object from the Collection
6)removeAll() --It is used to remove 1 collection from another collection
7)size()--It is used to count no of objects present in collection
8)clear() ---It is used to remove all the Objects from the Collection
9) isEmpty()-- It is used to check if the collection is empty or not.

List interface

Specifications of List interface
1)Insertion order is maintained

2)Duplication is allowed
3)It is index based
4)Null values can be Stored

Important methods present in List Interface (<<List>>)
1)indexOf() --- it is used to find the index position of an object
2)lastIndexOf()-- it is used to find the last index position of an object
3)get()-- it is used to return the Object based on index position
4)add(index,value)--it is used to add the object in the specified index position ,already existing objects is shifted to the next position
5)set(index,value) --it is used to add the Object in the specified position ,already existing object is overridden/reinitialized

---
ArrayList

ArrayList is a predefined class present in java.util package
Arraylist was introduced from jdk 1.2
All the 4 specifications of List interface is followed by arraylist as well

The underlined datastructure of Arraylist is "Resizable array" or "Growable array".

ArrayList Implements following interfaces
1)List 2)RandomAccess 3)cloneable 4)serializable

Constructor Present in ArrayList are
1)ArrayList()
2)ArrayList(int initialcapacity)
3)ArrayList(Collection c)

Internal working of ArrayList

when we create an instance/object of ArraList internally an array is created based on initial capacity or custom capacity (10)

When we add an element into an ArrayList once it is full ,A new array gets created based on the Incremental capacity formula

Incremental Capacity formula =((current capacity*3)/2)+1

All the objects from old array is copied into newly created array,
the reference variable pointing to old array gets dereferenced and starts pointing to newly created array
---

LinkedList
LinkedList is a predefined class present in
java.util package
LinkedList is stored in the form of "Nodes",
where in each node will store the object, along with
address of previous node and address of next node
LinkedList doesn't have initial capacity and incremental capacity concept
The underlined dataStructure of LinkedList is "Doubly LinkedList"

Note:
-- We prefer linkedList when there is inserting and deleting of elements in between is involved , because in Arraylist when we insert or delete
shifting of objects are involved which reduces the efficiency therefore we choose linkedlist when there is inserting and deleting taking place

- ---We prefer Arraylist when there is sorting and retrieving of objects is involved ,because Arraylist stores in sequencial order and traversing is faster

---
Difference between ArrayList and LinkedList
ArrayList
1)Initial capacity of Arraylist is 10
2) IC =((cc*3)/2) +1
3)underlined datastructure of arraylist is "resizable array or growable array"
4)ArrayList implements
a)List b)RandomAccess c)Cloneable d)Serializable interfaces
5)Sequential memory allocation

LinkedList
1)No initial capacity concept
2)No incremental capacity concept
3)underlined datastructure is â€œDoubly linkedlistâ€ •
4)LinkedList implements
a)List b)Deque c)Cloneable d)Serializable interfaces

5)No Sequential memory allocation. (nodes)

Enhanced for loop or for-each loop
---->for each loop is generally used to traverse a group of Objects from an array or collection
---->for each loop is unidirectional

syntax:

```
for(dataTypeToBeTraversed  refvar : array/collection)
{
  //set of instruction    //sopln(refvar);
 }
```

Non Primitive datatypes

class , interface are all non primitive datatypes
we can have n number of non-primitive datatypes
they donâ€™t have fixed size
the best example is String class , and the default value of all non primitive datatype is null

Collection Framework doesnâ€™t support primitive datatype
therefor the alternative is called as Wrapper classes

Wrapper class: The non primitive version of primitive datatype is called as Wrapper classes
All wrapper classes are present in java.lang package
---toString() is overridden in all Wrapper classes and String class,
---for-each loop supports Wrapper classes.

Primitive Datatype	---> Wrapper cLass(NON-Primitive Datatype)
1)byte	Byte
2)short	Short
3)int ---	Integer
4)long	Long
5)float	Float
6)double	Double
7)char ---	Character
8)boolean	Boolean

AutoBoxing:

AutoBoxing is a process of converting primitive to non primitive type
int x=15; //primitive
Integer y = new Integer(x); //non primitive

AutoUnboxing:
AutoUnboxing is a process of converting NON-primitive to primitive type

---
Generics
generics is used to specify the element type of a collection
,It was introduced from JDK 1.5

syntax: <Elementtype>

1)<< >> -- Interface (to indicate)
2)< > ----Generics ---Element type

Why we do Auto boxing and Unboxing ?
To convert primitive datatype to Object
Some API's and methods require Objects instead of primitive datatype
To provide additional Functionality
To store primitive datatype in collection
to allow null values

Vector (Same Like ArrayList)
Vector is a predefined class present in java.util package
Vector was introduced from jdk 1.0 hence it is called as "Legacy Collection"
The initial Capacity of Vector is 10
The incremental Capacity of Vector is = CurrentCapacity * 2

IC = CC*2
The underlined datastructure of Vector is "Resizable array" or "Growable array".

Vector is Thread safe ----Old --Single Thread
ArrayList is not ThreadSafe ----Multi Threaded

All the 4 specifications of List interface is followed by Vector as well
Vector Implements following interfaces
1)List 2)RandomAccess 3)cloneable 4)serializable

Constructor Present in Vector are

1)Vector()
2)Vector(int initialcapacity)
3)Vector(int initialcapacity , int incrementalCapacity)
4)Vector(Collection c)

Working of Vector
1)When we create an Object of vector , internally an Array will be created with default initial capacity 10 (or Custom capacity)
Ex: Vector v = new Vector();

2)When We add an Object into a Vector Once it is Full a new Array will be created based on Incremental capacity formula
3)All the Objects from the array gets Shifted into newly created array
4)The reference variable pointing to the old array gets dereferenced and starts pointing to the newly created array

Difference Between vector and ArrayList
1)Introducted from JDK 1.o
2)IC = CC*2
3)Vector has 4 constructors
4)vector is Thread safe

Note :
1. Collection : Collection is predefined Interface present in java.util package
2)Collections : Collections is a predefined class present in java.util package

1)reverse() : reverse method is a predefined static method present in Collections class

2)sort() : sort() is a predefined static method present in Collections class used to sort a list of Homogeneous Objects
sort(list l1)
sort(list l, Comparator c)

Stack
Java Collection framework provides a Stack class that models and implements a Stack data structure. The class is based on the basic principle of last-in-first-out. In addition to the basic push and pop operations, the class provides three more functions of empty, search, and peek. The class can also be said to extend Vector

and treats the class as a stack with the five mentioned functions. The class can also be referred to as the subclass of Vector.

Set

Set is a predefined interface present in java.util package
JDK 1.2
The implementation classes of set interface are as follows
1)HashSet
2)LinkedHashSet
3)TreeSet

generalization/specialization

```
            new HashSet();
Set s =     new LinkedHashSet();       //upcasting for all three
            new TreeSet();
```

Specifications of Set interface

1)Insertion order is Not maintained
2)Duplication is not allowed
3)Set is not index based
//these three points are exactly opposite to our List
4)null values can be stored

HashSet
Initial capacity of Hashset is 16
the fill ratio or load factor of hash set is 0.75 75% (its noting but incremental capacity)

The underlined dataStructure of HashSet is HashMap

LinkedHashSet was introduced from jdk 1.4
Both HashSet and LinkedHashSet are similar but the the only difference is HashSet doesn't maintain insertion order and LinkedHashset maintains insertion order

---
Q) How Set maintains Uniqueness?
or How set doesnâ€™t allow duplicates?
Ans: Internally 2 methods are overridden in all the implimentation classes of set interface

those two methods are hashcode() and equals()
Set doesnâ€™t allow duplicate (HashSet ,LinkedHashSet ,Treeset)

TreeSet
TreeSet is a predifined class present in java.util package
JDK 1.2
TreeSet maintains natural sorting order(Ascending order)
Duplication is not allowed
null cannot be stored
the object you store should be of comparable type(Cannot store heterogeneous type of data)

Internal Working of TreeSet
1)When we add an Object into TreeSet internally
compareTo() gets called
2)return type of compareTo() is int
3)compareTo() returns

+1 when its > right node
-1 when its < left node
0 when its = same node
4)the underlined Datastructure of TreeSet is Balanced Binary Tree
5)The traversing order of Treeset is
leftnode --->Parentnode ---->Rightnode

Q)Why are we calling compareTo() compareTo()
Q)Where is compareTo() present?
compareTo() is present in All Wrapper classes and String class
When we add an Integer object ,compareTo() of Integer class gets called likewise for other classes.

MAP's

Map is a predefined interface present in java.util package
Map is a part of collection Framework but map doesnâ€™t extend collection interface

Map is used to store the Objects in the form of key Value pairs
Keys(enroll no, Adhaar, emp ID) cannot be duplicated but values(Name, age,salary) can be duplicated

The implementation classes of map interface are as follows
1)HashMap
2)LinkedHashMap
3)TreeMap

Entry:
One key and one Value Together is called as "ENTRY"
---->Map is a collection of Entries

Important methods present in map interface
1)put() ---it is used to add an Entry inside map
2)get() --- It is used to return the value based on the key
3)containsKey()--it is used to check if the key is present or not
4)containsvalue()--it is used to check if the value is present or not
5)isEmpty()--it is used to check if the map is empty or not
6)size() ---it is used to find the number of entries present in map
7)clear()---it is used to remove all the entries
8)remove()---it is used to remove the entry based on the key
9)keySet()--it is used to return a set of keys

HashMap
Underlying data structure is "Array of Bucket" (Set & List)
Predefined class present in java.util package
Introduced from jdk 1.2
Insertion order is not maintained (sorting is done internally)
null can be used as Key

LinkedHashMap:
Underlying data structure of LinkedHashMap is
HashTable and LinkedList (Hybrid data Structure)
Predefined class present in java.util package
Introduced from jdk 1.4
Insertion order is maintained
null can be used as Key

TreeMap
Underlying data structure of TreeMap is "Red Black Tree".
Predefined class present in java.util package
Introduced from jdk 1.2

Natural Sorting Order(Ascending Order) is maintained (keys)
null CanNOT be used as Key
No duplicates Allowed

HashTable
Predefined class present in java.util package
Introduced from jdk 1.0 (legacy Collection)
Both HasHMap and HashTable are Similar but major
difference is HashMAp is not Thread safe
and HashTable is Thread Safe

Q)
Create an Entity(class) called as Person ,
define 4 attributes (name , age , gender, dob)
create constructor , override tostring()
--Create another entity called as GovAadharCard
define main Method ,create 5-10 Object of Person class
and store that Objects inside a Map (TreeMap) and print the information
using keyset method in for each loop.

Singleton Design Pattern/Singleton Class

"The process of creating only one instance/Object
for a class is called as Singleton Design pattern"

- --Rules for developing Singleton Design Pattern
*Rule 1: Declare a Private Constructor
(Object creation Cannot be done Outside the class)

*Rule 2: declare a public static helper method which
will create an instance/Object and Return the instance
Example: public static void createObject(){
if(Obj==null)
 Obj= new Account(); //Account class
}

Q] Why helper method static?
Ans:The helper method is static because Object
creation is not possible, if it static we can access

using ClassName)

- Rule 3:Declare a private static nonPrimitive reference variable
Ex: private static Account obj:
(To make it visible only in that class)

---**********----------