Features of Java

**Simple**: Bcuz Syntax is overridden from C and C++ and java does not support the concept of Pointer

**Secure**: Bcuz java program run inside a virtual machine sandbox and it’s not communicated directly with OS

**OOPs:** It’s a methodology that simplify the development and maintenance of our Software

1. **Abstraction:** Means hiding the implementation and showing the functionality to the user this we can achieve in java by using abstract keyword or else using concept of interface
2. **Encapsulation**: It’s a process of binding data and code together into a single unit by making data private and accessing it through public environment
3. **Polymorphism**: Poly means many and morph means form one object can perform multiple tasks as per argument we pass there are 2 type of polymorphism we have compile time and runtime.

Method and Constructor Overloading is Compile time and Method Overriding is runtime

1. **Inheritance**: Inheritance is a mechanism in which a one class acquired all the property and behavior of another class there are 5 type of inheritance we have

a) **Single** b) **Multilevel** c) **Hierarchical** d) **Multiple** e) **Hybrid**

Multiple and Hybrid not supported due to diamond problems but this we can achieve in java by using Interface

**Platform Independent**: Java convert its code into bytecode which can be run on any JVM enabled platform

**Portable:** Java convert its code into bytecode which can be run on any JVM enabled platform

**Multithreaded:** One Application can perform multiple tasks at same time that’s we called Multithreaded and with the help of java we can build that type of Application

**Robust:** Means Strong, Java is Strong because it having automatic exception handling technique automatic garbage collection and java does not support the concept of pointer which is unsafe for security

**Distributed**: Two Application can communicate with each other which build on same technology that is possible in java by using applets, servlets, aglets, remote method invocation (RMI) etc...

**Dynamic:** A source code written in one platform the same code can be executed in any platform. And it also loads the class files at runtime. anything that happens at runtime is considered as Dynamic, so the java is considered as Dynamic.

**High Performance**: Java is HP due to JIT (Just-in-Time) compiler, it’s also an interpreter means it convert our bytecode into binary code.

**Type Conversion of Data type?**

There are two type we have **implicit** and **explicit *Ex...***

**int** **i**; **char** **c**; **double** **d**; **byte** **b**; **float** **f**;

Expression: **i + b – f**

So first *i + b* will get converted into **int** because **int** is of 4 byte and **byte** is of 1 byte, so if we pass 2 in overloaded method it will call **int** args method because compiler give first priority to higher available

Now i + b will be something in **int** and now we add **int** to **float** it will convert our data into **float** because **float** contains some decimal values also, so to avoid loss of data it will convert into **float**, no matter if they both are 4 bytes

**Binary in Java?**

It’s a representation of our number like **75** into machine format which is **1001011**

**Find Binary of number 75?**

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **128 ends** | **128** | **64** | **32** | **16** | **8** | **4** | **2** | **1** |
| **75** |  | **1** | **0** | **0** | **1** | **0** | **1** | **1** |
| **142** | **1** | **0** | **0** | **0** | **1** | **1** | **1** | **0** |

Now **75** will falls in between **128** and **64** so we need to select lesser one that is **64** and after select we need to do minus like

**75 – 64 = 11**

Now we need to check **11** is falls between **16** and **8** so we need to select lesser one that is **8** and after select we need to do minus like

**11 – 8 = 3**

Now we need to check **3** is falls between **4** and **2** so we need to select lesser one that is **2** and after select we need to do minus like

**3 – 2 = 1**

Now the next highest is **1** so we need to stop here, so finally the number we have selected is

**64 + 8 + 2 + 1** is **75** So the number we have selected we need to consider it **1** and not select is **0** check above table

***Now for 142 next lesser number will 128 and so on.***

**Declare number as Binary in Java?**

// Binary literal in int type

**int** i1 = 0b10001110; // Using b0, the b can be lower or upper case

**int** i2 = 0B10001110; // Using B0

System.***out***.println("i1 = "+i1);

System.***out***.println("i2 = "+i2);

**What is Class?**

Class is Blue Print of our Object.

**What is Object?**

Object is having its own state and behavior state means Instance variable and behavior means Method

**Explain System.out.println()?**

System is a class in java available in java.lang package and out is a public static final Object of PrintStream class inside a System class and println is a method of PrintStream class.

**What is Constructor?**

1. Constructor must be same name as class name
2. Constructor is use to initialize the instance variable of a class
3. There are 2 type of Constructor we have parameterized and non-parameterized
4. If we don’t provide any Constructor in our class by default JVM will provide will one default constructor in our class
5. It does not have return type but its implicitly return the instance of current class
6. Constructor will get called implicitly whenever we create object of our class otherwise, we can call explicitly with the help of **this** and **super** keyword.
7. We cannot override Constructor but we can overload it that we called compile time polymorphism
8. Constructor can be private, protected, default and public

**Use of this keyword?**

1. with the help of this keyword, we can call current class method and instance variable within same class.
2. This can be used to call one constructor from another constructor but it should be the first statement
3. This can be used to refer current class object also
4. This can be used to differentiate between instance and local variable inside method and constructor
5. With this keyword we can call superclass data inside subclass because subclass acquired all property of superclass so now all property of superclass became subclass property

Employee.java

String eName;

**public** Employee getEmp(Employee e) {

**return** e; }

If we want Employee Object from another class so we are creating a method and taking that obj as a arguments in our method like above but instead of that we can use below approach to do above things So below return this will that object in which you are calling that method **getEmp()**

**public Employee getEmp() {**

**return this; }**

Client.java

Employee e = **new** Employee();

e.eName = "NSE";

System.***out***.println(e);

System.***err***.println(e.getEmp(e));

**What is method overloading?**

IF a class have multiple method with same name but different type parameter that’s we called method overload. And it’s also called Compile time polymorphism

1. We can overload method by changing its return type.
2. Compiler will come to know about duplicity with the help of parameters
3. Two methods can have same number of parameter but it should be of different data type.
4. In java only constructor and method can be overloaded.
5. We can overload main method also

Client.java

(1) **public** **void** name(Object o) {

System.***out***.println("O");

(2) **public** **void** name(String s) {

System.***out***.println("S");

(3) **public** **void** name(Integer s) {

System.***out***.println("I");

**new** Client().name(**null**); // "S" if we just have **1** and **2** method

**new** Client().name(**null**); // if we all **3** then Ambiguous compile error

**new** Client().name("ABC"); // "S"

**new** Client().name(**new** Thread()); // "O"

**What is static in Java?**

Static is a keyword in java, if we want to define some common properties amongst all object, we can use static keyword

1. In java static can be Instance variable, method, block and Inner class only
2. Static having its separate memory location inside heap and that we called static context or perm generation area.
3. Static data will get loaded inside a memory at the time of class loading
4. To call static data we don’t required object we can call it via class name we can call static with null reference also.
5. We can call static inside instance method directly via class-Name but if we want to call instance method inside static, we need to create object of our class.
6. We cannot override static method but we can Overload it.

**What is Instance Initializer and Static block?**

1. Instance Initializer is use to Initialize the instance variable of a class or else if we want to run some code each time whenever we create an object of our class, we can use Instance Initializer block.
2. Static block is use to initialize the static variable of a class or else if we want to run some code only once at the time of class-Loading or when first time we create an object of our class, you can call static block by Class.forName also.
3. If we have Instance Initializer block, static block and constructor in our class then first static block will get called then Instance Initializer then constructor at last. But if we again create object of that class on that time only Instance Initializer and constructor will get called.

**Variable in Java?**

1. Variable is use to hold different type of object
2. 3 type of variable we have Local variable, Instance variable and static variable
3. **Local variable** is resided inside some sort of block and the scope of local variable within that block only, Local variable can be final only apart from final we cannot able to use any modifier, Local variable will go inside a stack memory in java Local variable will not have any default value.
4. **Instance variable** are resided inside a class and outside any block, we can use all modifier with instance variable, to call instance variable we required object, Instance variable will go inside a heap memory, Instance variable will have some default value
5. **Static variable** is also resided inside a class and outside of any block, static variable can be called with the help of class name and can be with null reference also, static variable will go inside static context in memory, we can use any modifier with static variable if we do public static final to static variable then it will become Constant variable, static data will get memory only once whereas Instance and local will get Multiple time.

**Flow of Program?**

1. We have .java file that .java file we get converted into .class file with the help of java compiler (it resides outside JVM) Bytecode is reside in .class file
2. Class-Loader is a subsystem of JVM that is use to load class and interface in side JVM, After Pulling
3. That file will go to bytecode verifier, it will verify the bytecode that its proper or not After this...
4. JIT (Just-In-Time) compiler. It’s also called Interpreter it will convert our bytecode into machine understandable format

**Memory Allocation?**

1. Object of new keyword, Instance variable will go inside heap
2. Methods, local Variable will go inside a stack
3. Static variable will go inside a static context
4. String without new keyword will go inside a literal-pool and String with new keyword will go Inside Heap and Literal pool both but heap value is preferred.

**Inheritance starts from PDF....**

**Use of super keyword?**

1. We can call super class method, variable and Constructor of super class inside sub-class
2. We can call super class data directly inside sub without using super also we need to use super when super-class and sub-class both have same variable or method but we want super-class data.

**Constructor call in Inheritance?**

1. Whenever we have multiple constructors in our sub-class so then in each and every constructor of sub-class, we have to call one Constructor of super-class with super keyword.
2. When we have non parameter constructor in our super class then compiler will automatically that non parameter constructor in sub-class Constructor but if we just have parameter constructor in Super-class so then we have to call it manually.

**Blocks calling of Super and Sub with Constructor?**

1. First Super class static block and then Sub-class Static block called
2. Second Super class Instance Initializer block and then Super-class constructor will get called
3. Third Sub class Instance Initializer block and then constructor will get called

**Up-casting in Inheritance?**

The reference variable of super class refers to the object of sub class is called Up-Casting in java

1. We can access everything from super class and from sub class only overridden method, means Overridden method will called from sub class.
2. We cannot access the own properties of sub-class
3. If both the class have common instance variable then Super class variable will come bcuz variable calling is depend on type. Means we cannot override variables.
4. It is same as Left join of MySQL.
5. Down-Casting is vice-versa of up-casting but it not supported in java.

**Packages**

1. **default**: within a class and within a package
2. **private**: we can access within a class
3. **protected**: within a class within a package and all subclass of another package
4. **public**: Everywhere inside project context
5. Static Import: **import** **static** java.lang.Thread.*sleep*;
6. We can create java file without contain any single word in it, these types we create usually give some info like related to package or something else

**Rules while overriding the method.**

When a subclass is overriding the method of superclass then following factors must be considered for overriding method in subclass.

1. **Access modifier / Scope:** must not be more restrictive than original method i.e., a sub class cannot reduce visibility of overridden method of superclass
2. **Return Type:** there are following rules if return type is
3. **primitive:** should be same like original method
4. **non-primitive:** co-variants are allowed. Means if super class method is returning any object, then subclass overriding method can return object of its subclass type. It is added from Java 5
5. **Method name:** should be same as overridden method
6. **Type signatures:** both the number of parameters and data type of parameters should be same as original method
7. **Exception handling:**
8. If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.
9. If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

**What is final?**

1. It’s a keyword in java
2. If we declare any class as final, we cannot Inherit it
3. If we declare any method as final, we cannot override it
4. If we declare any variable as final, we cannot change value of it
5. It is use to achieve Immutability in java

**final** Integer eid;

String eName;

**public** Demo(String eName) {

**this**.eName = eName;

}

1. We are not initializing eid and its final field we will get compilation error Initialize final field in constructor, because final field need to have some value somehow

**Custom Immutable class in java:** Immutable class means that once an object is created, we cannot change its content. In Java, all the [wrapper classes](https://www.geeksforgeeks.org/wrapper-classes-java/) and String class is immutable.

1. The class must be declared as final (So that child classes can’t be created)
2. Data members in the class must be declared as final (So that we can’t change the value of it after object creation)
3. Allowed parameterized constructor
4. Getter method for all the variables in it
5. No setters (Do not have the option to change the value of the instance variable)

**Custom Immutable class with another class reference**

We have Employee class and inside we have Address class reference in it, our employee class is Immutable but Address can be changed because we are providing getter method to access.

Employee.java

**final** **class** Employee {

**private** **final** String empName;

**private** **final** **int** age;

**private** **final** Address address;

***// All parameters constructor and getters***

**public** Address getAddress() **throws** CloneNotSupportedException {

**return** (Address) address.clone();

}

}

Address.java

**class** Address **implements** Cloneable {

**public** String addressType;

**public** String address;

**public** String city;

***// All parameters constructor and getters***

**public** Object clone() **throws** CloneNotSupportedException {

**return** **super**.clone();

}

}

Client.java

**public** **static** **void** main(String[] args) **throws** CloneNotSupportedException {

Employee emp = **new** Employee("Adithya", 34, **new** Address("Home", "Madhapur", "Hyderabad"));

Address address = emp.getAddress();

System.***out***.println(address);

address.setAddress("Hi-tech City");

address.setAddressType("Office");

address.setCity("Hyderabad");

System.***out***.println(emp.getAddress());

}

In the above example, instead of returning the original **Address** object we will return a **deep cloned copy** of that instance. The address class must implement the **Cloneable** interface.

**Custom Immutable class with Date field**

**class** Person {

**private** String name;

**private** Date dob;

**public** Person(String name, Date dob) {

**this**.name = name;

**this**.dob = **new** Date(dob.getTime());

}

**public** String getName() {

**return** name;

}

**public** Date getDob() {

**return** **new** Date(dob.getTime());

We are creating a new copy of Date field otherwise reference to dob field may leak

We are returning defensive copy of Date field instead of directly returning the reference of instance variable.

**Abstract in Java?**

1. Abstract is a keyword in java Abstract can be class and method only
2. If we declare any method as abstract, we need to declare class also abstract if we declare class as abstract then we don’t need to do method as abstract
3. We can create everything inside an abstract class along with abstract method
4. Abstract method can be protected, default and public
5. We cannot create Abstract method as static, final and private bcuz we need to override this method
6. We cannot create Abstract class as final
7. We cannot create Instance of Abstract class and all data of abstract class will get called via Subclass
8. If we have 10 methods in Abstract class and we want only 5 in subclass then we have to make our subclass also abstract
9. Abstract class can extend another Abstract class

**Interface in Java?**

1. Interface is a concept to achieve some default amongst all class.
2. Inside an interface we can declare class, interface, Constant Variable, static, default and abstract method
3. Whatever we declare inside interface that will become public
4. If we declare variable, it will become public static and final means constant variable.
5. If we declare class or interface, it will become public static
6. Default and static method will become public.
7. While we implement any method of interface we cannot add **@Override** annotation on top of that.

**INNER CLASS**

**What is Inner Class?**

A class Inside a class we call them as Inner class we can create

1. Class inside Class (can be abstract)
2. Interface inside Class
3. Static class inside Class (can be abstract)
4. Interface inside Interface

**Types of Inner class?**

1. Inner class: a class inside a class
2. Static Inner class (Nested class): a static class inside class
3. Anonymous Inner class: a class with no name
4. Local Inner class: a class inside a method

**Inner Class**

1. We can access any data from outer to Inner directly without creating an object but to call Inner to Outer we required an Object in Outer class
2. If both the class have same variable and we want outer in Inner method the we can do Outer.this.a;
3. Static variable is not allowed if we want, we can create constant
4. If we are creating class inside an interface then static variable are allowed because inside an interface variable will be default **static final**
5. We can create Abstract class, Interface, concreate class inside a class
6. It can extend any class and implement any interface
7. Class inside a Class **private**, **protected**, **default** and **public** but if class inside an Interface its by default public
8. Object of Inner class will be

**Outer o = new Outer();**

**Outer.Inner i = o.new Inner();**

**Static Inner class (Nested Class)**

1. A class inside a class but it’s must be with static keyword
2. It can be abstract class or can be interface also
3. It can extend any class and implement any interface if it’s a class not interface
4. We can create any type of variable inside a static class
5. We cannot access Outer class non static data inside static Inner class directly if we want, we need to create Object of Outer class inside Inner class.
6. Object of static Inner class will be

**Outer.Inner i = new Outer.Inner();**

**Anonymous Inner Class**

1. It is use to minimize the code, now we can override method of class, Interface and abstract class without creating a separate class
2. Its name will be decided by compiler at Runtime.
3. .class file is also get created at runtime if we don’t want that .class file we can use Lambda Expression of java 8
4. Object of Anonymous Inner class will be

**Outer o = new Outer(){**

**public void one(){}**

**public void two(){}**

**}; o.One(); o.Two();**

**Local Inner Class**

1. A class inside a method or any local block is called Local Inner Class
2. Inside a method we can create class, abstract class only not interface bcuz is default in nature so it should be visible to all
3. It can extend a class and implement any number of interfaces
4. The object of Local Inner class must be inside a method in which it’s got created
5. We can create any type of variable in Local-Inner Class but not static if we want static, we can create a Constant

**Interface inside Interface**

1. It’s become static inner interface
2. If we want Outer interface method, we can implement Outer and if we want Inner interface method, we can implement Outer.Inner

**Class inside Interface**

1. It will become static inner class
2. If we want Outer interface method, we can implement Outer and if we want Inner static class method, we can extend Outer.Inner and remaining will be same as static inner class above
3. We can create interface also inside a class.

**What is String?**

String is a sequence of character its implemented Serializable, Comparable and CharSequence interface.

**String is Immutable in java means once we created a String, we cannot change it if we want changeable String, we can use StringBuffer or StringBuilder**

We can create a String in java by using two ways first is Literal pool and second is by using new keyword.

**By literal pool.**

1. String is having its separate memory area inside heap and that’s we called constant pool or literal pool.
2. When we do String s = “Sam”; then s will go in stack and SAM will go in literal pool memory.
3. If we do String s1 = “Sam”; again, previously already created for (s) then s1 will also go in stack but it will not create separate object of Sam instead of that it will point to previously created. That will save our memory.
4. Means s and s1 will point to same object (Sam)

**By new Keyword?**

1. When we do String s = new String(“Pam”); then s will go in stack and one object called Pam will go in literal pool and one object will go in heap s will point to heap and literal pool object is un referred later we can use intern method to refer it.
2. If we do String s1 = new String(“Pam”); then again one object of Pam should go to literal pool but it will not bcuz previous Pam of s is already there but it will again create a new object of Pam inside heap.

**NOW Practice String Methods....**

1. **Intern():** Intern method is use to refer the referred object of literal pool

**Programs in String**

1. Reverse character by character
2. Palindrome String
3. Reverse word by Word
4. Anagram String

**Ex:** LISTEN > SILENT

TRIANGLE > INTEGRAL

**Another word which we are creating from following character present in first word is called Anagram**

String s1 = "LISTEN";

String s2 = "SIFENT";

**char**[] c1 = s1.toCharArray();

**char**[] c2 = s2.toCharArray();

Arrays.*sort*(c1);

Arrays.*sort*(c2);

System.***out***.println(Arrays.*toString*(c1));

System.***out***.println(Arrays.*toString*(c2));

String x = String.*copyValueOf*(c1);

String y = String.*copyValueOf*(c2);

System.***out***.println(x.contentEquals(y));

1. Domain from Email

**String Compare**

String s1 = "SAM";

String s2 = "SAM";

String s3 = **new** String("SAM");

String s4 = s3.intern();

String s5 = "SAMPAM";

String s6 = s1 + "PAM";

**(s5 == s6) // false**

**(s1 == s2) // true** bcuz == will check ref and both pointing to same Object(ref) in literal pool.

**(s1 == s3) // false** bcuz s1 pointing to literal and s3 pointing to heap

**(s1 == s4) // true** bcuz intern method is use to refer the unrefereed object of literal pool and s4 is pointing to literal pool object

**s1.equals(s2) // true** bcuz Equals method will check content

**s1.equals(s3) // true** checking content

Same for equalIgnoreCase checking content with ignoring case

**StringBuffer vs StringBuilder**

1. String is immutable whereas StringBuffer and StringBuilder are mutable classes.
2. StringBuffer is thread safe and synchronized whereas StringBuilder is not, that’s why [StringBuilder is faster than StringBuffer](https://www.journaldev.com/137/stringbuffer-vs-stringbuilder).
3. String concat + operator internally uses StringBuffer or StringBuilder class.
4. For String manipulations in non-multi-threaded environment, we should use StringBuilder else use StringBuffer class.

**Difference between String.equals() and String.contentEquals() method?**

The **equals()** method is to compare the contents of **two Strings** are same whereas contentEquals method is to compare **String with StringBuffer or StringBuilder**. But both are to compare the contents.

**Exceptions Handling....**

**Keywords**

1. **Try:** try is use to monitor exception
2. **Catch:** catch is use to handle exception
3. **Finally:** will always executed weather exception occur or not, it will not execute when we are calling Infinite loop before finally or **System.exit(0);** or not putting checked exception into a suitable catch.
4. **Throw:** if we want to throw some exception Intentionally, we can use Throw keyword, we can throw Runtime and Compile time exception by using throw
5. **Throws:** Forwarding exception object to the invoking method is called exception propagation in java and this we can achieve by using throws keyword

**Custom Exception....**

1. To create compile time exception, we need to extends Exception class
2. To create Runtime exception, we need to extends RuntimeException class

**Expected Values....**

1. Only try allowed (**try with resources from JDK 1.5**)
2. Try finally allowed
3. Try catch finally allowed
4. Try catch catch.... are allowed
5. Only finally not allowed
6. Only catch not allowed
7. Only Catch with Finally not allowed
8. Try finally catch not allowed
9. Try catch finally finally.... not allowed

**If we are using multiple catch then Super type of exception we need to put at last**

**Try-Catch with or | operator:** Now from java 7 we can handle multiple exception in single catch using or

**try**{

**int** a = 3;

Thread.*sleep*(100);

FileReader fw = **new** FileReader("");

}**catch**(InterruptedException | FileNotFoundException e) {}

Now we cannot add **IOExeception** in above catch bcuz FileReader throws **FileNotFoundEx** only which is already handled and **IOExce** will also do same thing for **FileReader** so 2 **Exception** doing same thing is not possible in same catch with OR operators EX……

**catch**(InterruptedException **|** FileNotFoundException **|** NullPointerException **|** NumberFormatException **e**)