**Introduction to Java**

**1.  What do you know about Java?**

Java is a high-level programming language. It is platform independent. The java code  runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX. Add answers of **Q:3 also**

**2. What are the supported platforms by Java Programming Language?**

Java runs on a variety of platforms, such as Windows, Mac OS, and the various versions of UNIX/Linux like HP-Unix, Sun Solaris, Redhat Linux, Ubuntu, CentOS, etc.

**3. List any five features of Java?**

Object Oriented, Platform Independent, Robust, Multi-threaded language, Distributed

**4. Why is Java Architecturally Neutral language?**

 When a source code is compiled, the javac  compiler generates an architecture-neutral file format called as byte code(class file). This byte code is platform independent. It can be executed on any machine running in any OS if JVM is available

**5. What is JVM ?**

When Java is compiled, it is compiled into platform independent byte code. This byte code is distributed over the web and interpreted by Java Virtual Machine (JVM) on whichever platform it is being run.

**6. Is JVM platform independent?**

Yes. It is designed to adapt to an evolving environment. Java programs can carry extensive amount of run-time information that can be used to verify and resolve accesses to objects on run-time.

**7. What is JIT Compiler?**

JIT compilers interact with the JVM at run time and compile appropriate bytecode sequences into native machine code.

It is enabled by default, and is activated when a Java method is called.

It compiles the bytecode of that method into native machine code, compiling it "just in time" to run. Now the JVM calls the compiled code of that method directly instead of interpreting it.

**8.  List some Java keywords(unlike C, C++ keywords)?**

 Some Java keywords are final, abstract, finalize, interface, implements, enum.

**9. Can a java file have more than one java class?**

Yes, you can have any number of java classes in a single file. Only one class can be public.

**10. Can I have more than one main method in a class?**

Yes, but with different parameters. But JVM will check for the method with public static void main(String a[]) as parameter. If not available will throw an exception as Main method not found

**Datatypes, Controlflow, Arrays**

1. **What are Shift operators?**

A shift operator performs bit manipulation on data by shifting the bits to the right or left

Types – left, right, right shift zero fill

1. **What is the default value of boolean?**

false

1. **What is the default floating point type in Java?**

double. It takes 64 bits.

If you want declare float use float a= 10.5f;

1. **What type of parameter does switch accept?**

switch accepts variable that can be of type byte, short, and int. switch from java 7 also accepts String also.

1. **Is it mandatory to have default case in switch statement?**

No. It is optional.

1. **What is the use of char?**

‘char’ is a data type which is a Unicode representation of characters. It can be used for for giving backspace, newline, tab etc

1. **What is typecasting?**

**Assigning a value of one datatype to another is called typecasting. Two types - Upcasting and Downcasting**

**Upcasting – The conversion happens automatically. Called as automatic conversion**

**Downcasting – The conversion has to be done explicitly. This is explicit conversion**

1. **Explain arrays in Java?**

* **Arrays are used to store multiple like types values in a single variable**
* **Arrays come under reference data types**
* **When an array is declared and initialized using new keyword, memory gets allocated**
* **eg. int[] nums = new int[4];**
* **The index starts from 0**
* **The length of the array can be obtained using length property**

1. **When does an ArrayIndexOutOfBoundException occur in an array?**

**When you try to populate an array with value out of the bounds, this exception occurs**

**eg. int[] nums = new int[4];**

**num[4]=90;**

**When the second statement is given, it will compile fine. But it will throw AIIOB exception during the runtime**

1. **When does a NullPointerException occur in an array?**

**After declaring an array, if you try to populate it without allocating memory(without using new keyword), it will throw NullPointerException**

**eg. int[] nums; //array declared but not initialized**

**num[2]=90; //populating, before initializing**

**when the below statement is given, it will compile fine. But it will throw NullPointerException during the runtime**

**Classes and Object**

**1. What is an object?**

Anything can be an object. A car, pen, laptop. An object has its own state, behaviour

The state is the properties(variables) of an object.

(eg) A pen has colour, size, shape. A car has model, price, mileage.

The behaviors are the functionalities (the methods) that can be performed on an object (ie) getFeatures(), getPriceDetails() etc.

**2. Define class?**

A class is a collection of objects. A class is a blueprint from which objects are created. A class is a collection of objects with common properties , behavior , relationship and semantics.

(eg) collection of car objects- Car class

**3. What is a local variable**?

The variables within methods, blocks and constructor are called local variable. Their scope is within the method only. They don’t take any default value. They must be initialized before using them. If not will give compiler error.

**4. What are instance variables** **?**

These are the properties of an object. Each object has its own set of instance variables(values are different). The values are initialized to default values on creation of the object

**5. What is a static variable ?**

These variables are available as a single copy in a class. All the objects in a class share the same copy. The static variables are initialized to their default values when the class gets loaded.

**6. What is a constructor?**

A constructor is used to initialize the instance variables

**7. What is the use of constructor?**

To initialize the instance variables. It comes with the class name. It is invoked automatically when an object is created.

**8. Can a class have more than one constructor?**

Yes. With different parameters

**9. Can a class have private constructor?**

Yes. But you cannot create objects from another class using this

**10. How will you create an Object?**

Use **new**operator to create an object.

**11. What package gets imported automatically? - java.lang**

**12. What is the top level class of all the classes? - Object.**

**Polymorphism - Overloading**

1. **What is polymorphism?**

One name taking many forms is called polymorphism.

one method taking many forms –(method with one parameter, two parameter and so on)

move() is a method having different functionalities for different objects under same hierarchy

Types - Compile time, Runtime

1. **What is Overloading?**

Overloading is a concept where the method or constructor is overloaded with zero to any number of parameters

Types - Method Overloading, Constructor Overloading

1. **What is Method Overloading?**

Method Overloading is a concept where the method is overloaded to zero to any number of parameters

Here

***access- specifier* *returntype* methodname(parameter list)**

***access -specifier, return type***– any

***methodname*** – same

***parameterlist***  – the no of parameters or type is different

1. **Why Overloading is called as Compile time polymorphism?**

The binding of the method and the calling happens during the compile time. (if not available gives compiler error)

1. **Can we overload static methods? --** Yes
2. **What is constructor overloading?**

Having more than one constructor with zero to many parameters is called as constructor overloading

1. **How to invoke one constructor from another in the same class?**

Using this() keyword with matching parameter.

1. **What is the use of this(), this keyword?**

**this()** keyword is used to call one constructor from another by passing matching parameter in the same class.

**this** refers to the current object. It prevents the local variable hiding the instance variable

1. **What is Var-args?**

Var args is variable argument list. It can used within a method as parameter. It can values as zero to many. It similar to an array

**Rules**

* Can have only one vararg
* var args can be the last argument only.
* datatype must be followed by 3 dots(…)

**Example**

***public void area(int… x)*** 🡪 takes 0 to many values of x

1. **When to use overloading and when to use var-args?**

**Var-args -** If the logic(functionality) is same but the parameter list is different use varargs.

**eg. double calcAvg(int … x)**

can be used for

**double calcAvg(90,80)**

**double calcAvg(90,80,100)**

**double calcAvg(90)**

**Overloading -** If the logic and the parameter list is different, use overloading

**eg. double calcArea(int x) 🡪(sq)**

**double calcArea(double x) 🡪(circle)**

**Inheritance**

1. **What is Inheritance?**

Is a concept in which one object inherits the properties of another

1. **Can the subclass access all methods and variables of super class?**

Sub class can use all the properties and methods of the super class except private members.

1. **Can the subclass have its own methods and variables? --** Yes
2. **Why do we need to use Inheritance?** -- Reusability.
3. **What type of inheritance does Java support?**

Simple, hierarchial & multilevel inheritance.

1. **What is the keyword to use with inheritance?** extends
2. **What is the use of super keyword?**

Is used to call the constructor of the super class

Has to be in the first line of a constructor

super() is added in the first line by default by the compiler

The parameters within super must match atleast one constructor of the super class

With super keyword, the super class variables are initialized first and then the sub class variables.

1. **What is Overriding?**

If the subclass has the same method signature of the superclass, overriding happens. It is also called as DynamicMethodDispatch or Runtime Polymorphism.

Overriding is done to get *different implementation* from the sub classes.

**Syntax**

**access-specifier returntype methodname(parameterlist)**

**access-specifier** 🡪 same or level above(less restrictive)

**methodname, parameterlist** 🡪 same

**returntype** 🡪same except co-variant return types

**class Employee{**

**public void calcBonus(){ }**

**}**

**class Manager extends Employee{**

**public void calcBonus(){ }**

**}**

1. **How to call the overridden method?**

Using super-class reference = sub-class object

Employee employee = new Manager();

employee.calcBonus(); ---🡪 This will call the method in the subclass

1. **Why overriding is called as Runtime Polymorphism?**

When you refer super-class reference = sub-class object

The method gets dispatched dynamically during the runtime. The super class decides which subclass method to call only during the runtime.

1. **What is co-variant return type?**

It refers to the return types of overridden methods. It can be only non-primitive type. The return type of the subclass method should be same as that of the super class method return type (as in Employee) or sub classes of that class(Manager, Programmer)

**Example:** Classes - Employee, Manager, Programmer

**Method in super class**

**public Employee getEmployeeById(){ }**

**Method in sub class**

**public Employee getEmployeeById(){ }**

**public Manager getEmployeeById(){ }**

**public Programmer getEmployeeById(){ }**

1. **Can static methods be overridden?**

No. static methods cannot be overridden

1. **What is the meaning of “IS-A” and “HAS-A” relationship?**

* **IS- A** means Inheritance
* **HAS-A** means Association

**class Employee{**

**}**

**class Manager extends Employee{**

**Address address;**

**}**

Here Manager is an Employee(Inheritance). Manager has an Address instance(Association)

1. **What is the difference between this() and super()?**

**this:**

* **‘this()’** is used to call the constructor in the same class
* **this.** refers to the current object. **this.property** is used inside the constructor to initialize the instance variables

**super:**

* ‘super()’ is used to call the constructor in the super class
* super.methodName() is used to call the super class overridden method
* Either this or super has to be in the first line of the constructor
* They cannot be used inside the static methods

1. **Difference between Overloading and Overriding**

**Overloading:**

* Happens in the same class
* Compile time polymorphism
* Parameter list or parameter type should be different
* Return type can be any

**Overriding:**

* Happens in superclass sub class
* Runtime polymorphism
* Parameter list and parameter type should be same
* Return type should be same except co-variant types

**Abstraction**

1. **What is Abstraction?**

Abstraction is a concept in which the internal or unwanted details are hidden. It is achieved with the help of abstract classes and interfaces

1. **What is an abstract class?**

* A class that may or may not have abstract method
* The class must have abstract keyword.
* The abstract class should be extended.
* The subclasses provide the functionality for the abstract methods

**abstract class Employee{**

**abstract void calcBonus();**

**}**

1. **Can an abstract class be instantiated ( create objects)?**

* No. They cannot be instantiated. (ie) can’t create objects

1. **What is an abstract method?**

* A method with declaration and no definitions
* The abstract method must use abstract keyword
* The abstract method must be implemented in the sub classes.
* The subclasses provide different functionality for the abstract methods

**abstract void calcSalary(int x);**

1. **Can an abstract class have concrete methods ?** --Yes
2. **Can an abstract class be without abstract methods?**

Yes. The purpose is for future extensibility

1. **Can an abstract class have private methods?**

Yes. But it can be used only within the class

**abstract class Employee{**

**public void printDetails(){}**

**abstract void calcBonus();**

**final double payment(){checkCredentials() }**

**private void checkCredentials(){}**

**}**

1. **Can an abstract class have final methods?**  -- Yes.
2. **Can an abstract class have constructor?**

Yes. But it should be public, so that it can be used by sub classes to create objects. As abstract class cannot be instantiated

1. **What is a final class?**

A final class is one which cannot be extended

If a class should not be extended, use the keyword final

**final class PensionDetails{ }**

Many inbuilt classes are final **eg. String, Math**

1. **What is the use of final class?**

* To prevent inheritance
* To create immutable classes

1. **Name few inbuilt classes that are final in java.**

java.lang.String , java.lang.Math

1. **What type of methods can a final class have?**

* A final class cannot have abstract methods
* A final class can have concrete, private and final methods

**final class PensionDetails{**

**public void printDetails(){}**

**final double payment(){checkCredentials() }**

**private void checkCredentials(){}**

**}**

1. **What is a final method?**

A method with final keyword cannot be overridden in the subclasses

**final void payment(int x){ }**

1. **What is the use of final method?**

Is used to prevent Overriding. Say you have few methods in your application, you allow the sub classes to use them, but do not want the subclasses to override them. In such scenarios use final methods – so all the subclasses will get the same functionality from the super class

1. **How to declare constants in Java?**

* Use the keyword final with the variable to make it constant
* The variable should be initialized while declaring itself
* It is created mostly as public to allow external access.
* By convention - variable name has to be in uppercase, as it is constant

**public final double BONUS = 1000;**

A variable declared as final inside a method, constructor or block is called local final variable

1. **What is the difference between abstract, final and private methods?**

**private method**

* A private method in the super class is not even visible in the sub class

**final method**

* A final method cannot be overridden in the sub class.
* All the subclasses will get the same functionality of the method
* It is visible in the subclass and can be used, but not overridden

**abstract method**

* An abstract method should be overridden in the sub class
* The subclasses provide different functionality for the abstract methods

1. **What is the difference between abstract and final?**

**abstract**

* An abstract class should be extended
* An abstract class can have abstract, concrete, final methods
* An abstract class cannot be instantiated
* An abstract method should be overridden in the sub class

**final**

* A final class cannot be extended
* A final class can have only concrete and final methods
* An final class can be instantiated
* A final method cannot be overridden in the sub class
* A variable declared with final keyword is constant

**Encapsulation**

1. **What is a java bean?**

A java bean is a public class with private instance variables and public getter and setter methods.

1. **What is the use of native keyword in java?**

The native keyword is used with methods. If a method is declared as native, then it means the implementation of the method is given in C language, not in java

**Example:** public static native java.lang.Thread currentThread();

1. **What is a package?**

Package is a container for a class. It is just the folder in which the class is created.

1. **What are the different access specifiers in java?**

public, protected, default(not a keyword), private

1. **What is the meaning of default / friendly?**

default/ friendly is not a keyword. A representation of a class /method/variable without any keyword is default. The scope of this is only within a package.

1. **What is the access specifier used for class?**

public & default only. An inner class can be private, as it acts like a class member.

1. **What is the correct access specifier used for instance variables?**

**protected** – if the access specifier is protected, then the variables can be used by the sub classes also.

1. **What is the access specifier used for method?**

public, protected, default(not a keyword) & private

1. **What is the scope of the different access specifiers?**

**private:**

If a variable or a method is private, only the class in which it is declared can access it

**default: (not a keyword)**

If a variable or method has no access specifier, all the classes in the same package can access it

**protected:**

If a variable or a method is protected, all the classes in same package and the sub classes in different packages can access it

**public:**

If a variable or a method is public, all the classes in same package and different packages can access it

1. **What is the access specifier that can be used for local variable?**

final

**static**

1. **What is static keyword?**

static is a keyword. It can be as block, variables, method, and as top level of nested class. The method/variable marked as static is available as a single copy only. All the objects share the same value only.

1. **In which sequence the static keyword is executed?**

First, the JVM calls the static variables and initializes them to default values. Then the static blocks are called in the order written. Here the static variables are given values. Then PSVM is called. From here, the other static methods can be invoked.

1. **How to call a static method?**

The static method/variables can be called without creating an object. They can be called using ***classname.method name*** or ***classname.variablename***.

1. **Name few inbuilt static method**

Integer.parseInt(“200”);

Thread.sleep(1000);

Double.valueOf(“900.2”);

1. **How to call a non static(normal) method from a static method?**

Use object to call a non static method

(eg) calling a method from PSVM using object

1. **What is the use of static import?**

The static import is used to import the static member of one class to another class. The syntax

***import static pkg.subpkg.classname.\*;***

This imports all the static members of a class into the calling class

1. **Can you refer to a super class object or a current object inside a static method?**

No. super and this keyword cannot be used inside a static method.

1. **Can a static method be overridden?**

No. Even if you have the same method signature in the subclass, while calling using dynamic method dispatch [*super class ref= sub class object*], since the method is static only the super class method only will be called.

1. **Can I have more than one static block? --** Yes
2. **In which sequence the static blocks will run? --**  In the order of creation.

**Interfaces**

1. **What is an interface?**

* Interface is similar to abstract class with methods and variables
* Interfaces must be implemented by classes to give functionality to the methods

1. **What is the use of interface?**

* Helps to achieve abstraction
* Helps to provide multiple implementations
* Interfaces can’t be instantiated
* A class implements many interfaces to get multiple functionalities
* Multiple classes can implement an interface to come under a common category

1. **What type of methods does an interface have?**

**public abstract methods**

* The classes that implement the interface provide the functionality

**static methods**

* used to provide common functionality for all the implementing classes
* They can be called using interface name. No need of implementing class
* Cannot be overridden in the subclasses

**public default methods**

* helps to add new functionality to the existing interfaces of the application
* Adding a default method to an existing interface does not break the contract
* default methods are implicitly public

**interface interfacename{**

**// public static final variables**

**// abstract methods**

**// default methods**

**//static methods**

**}**

1. **What type of variables does an interface have?**

Variables are public, static, final by default

**interface BonusCalculator{**

**int AMOUNT = 100;**

**void fly();**

**default void payment(){ }**

**}**

1. **Can multiple classes implement an interface?**

Multiple classes can implement an interface to be grouped under one category.

**eg**. If class A and class B implements Serializable the objects of these classes can be serialized.

1. **Can a class implement multiple interfaces?**

Yes. A class can implement multiple interfaces to get many different functionalities.

1. **What is a marker interface?**

An interface without methods is called ***Marker or Tag interface***

**eg. Serializable, Cloneable**

The purpose of this interface is to group multiple classes under a common category

1. **What is a functional interface?**

* An interface with only one abstract method is called as **Functional Interface**

**eg. Comparable, Runnable**

* They may or may not be annotated with @FunctionalInterface
* Functional interfaces can also have default and static methods
* The abstract method can be implemented using Lambda expressions

1. **Name few functional interfaces from java 8**

Predicate, Consumer, Supplier, Function

1. **Can interfaces be extended? If so what is the use?**

Yes. Interfaces can be extended. Whenever you have to add more functionality, extend the interface and add the new functionality in the sub interface. The class implementing the sub interface will have the functionality of both the interfaces

**interface Calculator {**

**void add(int x, int y);**

**}**

**interface Scientific extends Calculator{**

**void square(int x);**

**}**

1. **What is the use of default Methods?**

* Helps to add new functionality to the existing interfaces of the application
* Adding a default method to an existing interface does not break the contract
* Helps to add a common behavior across all implementing classes of the interface.

**Example:**

**interface Calculator {**

**void calculate(int x, int y);**

**//adding a new method to an interface**

**default void square(){}**

**}**

1. **What is the difference between abstract class and interfaces?**

**Abstract classes:**

* Can have abstract, final, concrete methods
* Can have instance variables
* A class can extend another class and implement multiple interfaces
* Use in case of IS-A relationship

**Interfaces**

* Has abstract, default, static methods they are implicitly public
* The variables are by default public static final
* An interface can extend multiple interfaces
* Use in case of complete abstraction

**Exception handling**

1. **What is an exception?**

Exception is an abnormal condition that disrupts the normal flow of execution. It can be logical errors, database errors like

* Connecting with the database
* Attempting to access a file that does not exist
* Inserting an element into an array at a position that is not in its bounds
* Performing some mathematical operation that is not permitted
* Declaring an array using negative values

1. **What will happen if exception is not handled?**

If an exception is not handled,

* The runtime system throws this exception.
* The default handler of the java runtime system(JVM) handles the exception. It displays the string describing the exception
* This is the stack trace that shows sequence of method invocations that led to error and then terminates the program

1. **What is the meaning of exception handling?**

The ability of a program to intercept run-time errors and and take corrective measures to continue the execution of the program is exception handling. All the exceptions must be either handled or declared.

Five keywords used in handling exception are

* 1. try
  2. catch
  3. finally
  4. throw
  5. throws

1. **What are the types of exception?**

UnChecked and Checked

1. **What are Unchecked Exceptions?**

* All the classes that come under RuntimeException are called as Unchecked exceptions
* They are also called as Runtime Exceptions
* If the exception is not handled, it will be handled by the JVM and terminates the application

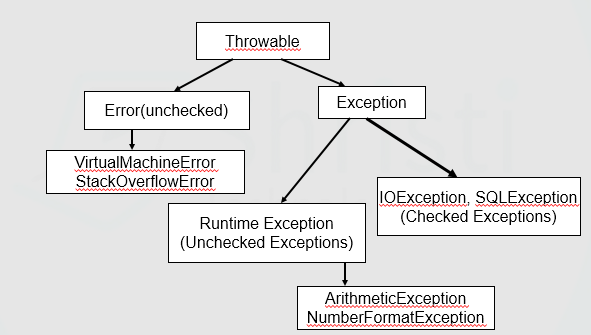
**eg. NullPointerException, NumberFormatException**

1. **What are Checked Exceptions?**

* All the classes that come under Exception directly are called as Checked exceptions
* They are also called as Compile time Exceptions
* These exceptions should be handled by using try and catch or declared using throw, throws
* If the exception is not handled, It will give compiler error

**eg. IOException, SQLException**

1. **What is the hierarchy of exception?**



1. **What is error?**

* These are exceptions that are not expected to be caught under normal circumstances by the program.(They are NOT compiler errors)
* They are used by the Java runtime system to indicate errors having to do with the runtime environment.
* Stack overflow is an indication of such an error.
* Exceptions of type ***Error*** are beyond the control of the program

**eg. StackOverflowError, OutOfMemoryError**

1. **What is the use of try and catch?**

**try**

* All the error prone statements are inside try block
* The try block should be followed by catch or finally
* No statements between try and catch
* A try can be with try-catch, try-catch-catch, try-catch-finally, try-finally

**catch**

* Is used to handle the exception
* Can have more than catch statements
* Always the subclass exceptions must be handled and then the superclass exceptions

1. **What is finally and why it is used?**

* finally is a block and is called when there is an exception and also when there is no exception
* finally is used to **release the resources** – close the files, database connection etc.
* finally can be with try or with try-catch
* finally will not be executed if you call **System.exit(0)** inside the catch block

1. **What is the use of throw keyword and when it is used?**

* **throw** is used to throw the exception from the current method to the calling class
* The syntax is **throw Throwable instance;**

**eg. throw new IOException();**

* An instance of error also can be thrown
* Whenever you throw a checked exception from a method, declare the exception using throws keyword

**public void readFile() throws IOException{**

**throw new IOException();**

**}**

1. **What is the use of throws keyword and when it is used?**

* **throws** is used to declare that the method may or may not throw an exception
* You can declare many exceptions in throws clause

**eg.**

**public void readFile() throws IOException, SQLException{ }**

* Whenever you throw a checked exception from a method, declare the exception using throws keyword

1. **What is the difference between throw and throws?**

**throw**

* **throws the exception** from the current method to the calling class
* Can throw only one exception
* Has to be the last statement in a block (if, try, catch)

**public void readFile() throws IOException{**

**throw new IOException();**

**}**

**throws**

* **Declares the exception** that the method may or may not throw
* Can declare many exceptions in throws clause

1. **What are custom exceptions?**

These are exceptions specific to projects. You can create custome exceptions by extending RuntimeException or Exception class

eg.

Validator

**public class NamenotFoundException extends Exception{ }**

**public class TooShortException extends RuntimeException{ }**

1. **What is the difference between final, finally and finalize?**

**final**

* The class declared final cannot be extended
* The method declared final cannot be overridden
* The variable declared final is a constant

**finally**

* Can be used only with try
* It is used for releasing the resources – files, database connection

**finalize**

* Used in the context of garbage collection
* This is used to close the files, db connection, making the references to null before gc ing
* The GC will call this method, before starting the garbage collection

**Multithreading**

1. **What is the Multithreading?**

Multithreading is a concept where are two or more parts run simultaneously. Each part is called as a thread. This concept is used to utilize the ideal time of the CPU

1. **What is the thread?**

A thread is a separate path of execution. Multiple threads can run simultaneously.

1. **How will you create a thread?**

By extending Thread class or by implementing Runnable interface.

1. **What is the default thread that runs, when the JVM starts the execution?**

The main thread. It has a priority of 5

1. **How to identify the current thread that runs in the method?**

* Use the method Thread.currentThread(). This is a static method and returns the thread instance.
* To get the name of the current thread use the method Thread.currentThread() .getName()

1. **What are the priorities of thread?**

Thread.MAX\_PRIORITY- 10

Thread.MIN\_PRIOIRITY-1

Thread.NORM\_PRIORITY-5

These are public static final variables from java.lang.Thread class

1. **Explain the life cycle of a thread?**

* When a thread is first created, it is in the NEW State

**Thread t = new Thread();**

* + - When you invoke start method, it gets ready to get the CPU.

**t.start()**

* + - The thread changes to RUNNABLE state ( eligible for execution ) depending on its priority

• When sleep/wait method is called on a RUNNABLE thread, it may enter the NOT RUNNABLE state.

**Thread. Sleep();**

* + - When a thread is BLOCKED, it is still alive, but it is not eligible for execution.
    - A BLOCKED thread becomes ready to run again when the sleeping thread wakes up.
* This thread occupies the CPU depending on its PRIORITY
* When a thread terminates, it is said to be DEAD

1. **What method is used to identify the thread that is running currently?**

**Thread.CurrentThread()** is the method that is used to get the thread that is occupying the CPU currently. It returns a thread reference.

1. **What are Daemon Threads?**

* They are like a service providers for other threads or objects running in the same process as the daemon thread.
* They are used for background supporting tasks and are only needed while normal threads are executing.
* If normal threads are not running and remaining threads are daemon threads then the interpreter exits.
* They do not prevent the JVM from exiting when the program finishes even if the thread is still running.
* **Example** for a daemon thread is the garbage collection.

1. **How to create daemon thread?**

By using setDaemon(true) method on the thread instance.

1. **What is Synchronization?**

* When two or more threads need access to a shared resource, they need some way to ensure that the resource will be used by only one thread at a time.
* Synchronization in Java can be achieved using the keyword – synchronized

1. **Where can I use synchronized keyword?**

• In methods

• as synchronized blocks

1. **What is a synchronized block?**

* This is a synchronized block

synchronized(Object o){

o.anymethod();

}

* It performs two actions:
  + After getting a reference to an object, it locks that object
  + The thread entering this block gets the lock of this object
  + The method is called on the object
  + After execution of the body has completed, either normally or abruptly, it unlocks that same lock.

1. **What happens if a thread is in a synchronized method?**

If a thread is in synchronized method, all the other threads trying to call this method or any other synchronized method using the same object have to wait.

1. **What is the use of sleep method?**

* The sleep method holds the lock of the object and releases the CPU, so that any other thread can use the CPU.
* The thread wakes up automatically after the time elapses and goes to the thread pool.

**java.lang**

1. **Name few classes in java.lang package**

Object, Math, System, Exception, Thread, Throwable, String, StringBuffer, StringBuilder, Number, All Wrapper classes – Integer, Float, Double

1. **Name few interfaces in java.lang package**

Runnable, Cloneable, Comparable

1. **What are Wrapper classes?**

Wrapper classes are classes that wrap the primitive data types into objects. Each primitive data type is having the corresponding wrapper class.

**eg**. float primitive has Float Wrapper class.

double primitive has Double Wrapper class.

Name the Wrapper classes ?

Integer, Float, Byte, Short, Long, Double, Boolean, Character

Name the Wrapper classes that come under Number class?

Integer, Float, Byte, Short, Long, Double

1. **What is Autoboxing?**

Autoboxing is used to convert primitive data types into objects

***eg. int x =10; Integer y = x;***

*‘x’ is converted into Integer object;*

1. **What is Autounboxing?**

Autounboxing is used to convert objects into primitive datatypes

***eg. Integer y = new Integer(90); int x = y;***

*‘y’ object is converted into int primitive data type.*

1. **What is the use of hash code() and equals() method?**

* Both the methods are from java.lang.Object class
* Every object has access to the *equals()* method because it is inherited from the *Object* class. The default implementation - compares the memory addresses of the objects.
* To emphasize that two objects are same based their instance variables, you can override the default implementation of the  *equals()* method from *Object class* and give your own implementation. While overriding equals must also override *hashCode()*. Otherwise a violation of the general contract for Object.hashCode() will occur.
* Set implementation classes override equals method to identify duplicate objects.

1. **What is Cloning ?**

Cloning is creating a copy of the original object. It is field by field copy. The cloned object has separate memory address. The types of cloning supported by Java - Shallow Cloning, Deep Cloning

In Cloning if x2 = x1.clone(), then

x2== x1 returns false

x2.getClass()==x1.getClass() returns true

1. **What are steps involved in cloning?**

The Class must implement Cloneable interface and override clone method

No constructor is called on the object being cloned.

1. **What is Shallow Cloning?**

**In Shallow Cloning**

* If the class has only primitive data type members, then a completely new copy of the object will be created
* Then the reference to the new object copy will be returned.
* If the class contains members of any class type, then only the object references to those members are copied
* Hence the member references in both the original object as well as the cloned object refer to the same object.

1. **What is Deep Cloning?**

**In Deep Cloning**

* The object is copied along with the objects it refers to.
* Deep clone copies all the levels of the object from top to the bottom recursively.
* All the member classes in original class should support cloning
* In clone method of original class, call super.clone() on all member classes.
* If any member class does not support cloning , then in clone method, one must create a new instance of that member class and copy all its attributes one by one to new member class object. This new member class object will be set in cloned object.
* Or use Serialization for deep copy

1. **What is the use of toString()?**

toString() method is from Object class. It is used to give string representation of an object. To get this, it is mandatory to override toString() method in the required class and give the implementation.

1. **What is a String?**

* String is sequence of characters.
* String is a final class from java.lang package
* It is fixed and immutable, (ie) once created the value cannot be changed.
* Operations like finding index of characters, getting characters from index, concatenation, checking equality and substring can be performed
* Operations like – insert, delete, append are not in String.
* With ‘+’ though it appends only a new String is created

1. **Why is String immutable?**

Once a string object is created, it cannot be changed. So it can be safely shared across threads.

**Example**

**String s =”hello”;**

**s.toUpperCase();**

**System.out.println(s);**  // will print ”hello” only.

“HELLO” is created and it will be lost as it is not referenced by any variable.

1. **What is the difference between == and equals method while using with String variables?**

**equals()** This compares the string literal values only

**==** This compares the String Object references

**Example**

**String s1="hello"; String s2 = "hello" ;**

**String s3=new String ("hello world"); String s4 = new String("hello world");**

**s1. equals (s2)** ---------->returns true as both have same value with same case,

**s3. equals (s4)** ---------->returns true

**s1==s2** -------> returns true as both references point to the same object

**s3==s4** --------> returns false as both references point to two different objects

1. **Which method will be called by the garbage collection before doing the garbage collection?**

finalize() is the method that is called before doing gc.

1. **What does finalize () do?**

The resources associated with the object are released inside the method, before garbage collecting the object.

1. **What methods are used to call garbage collector explicity?**

System.gc()

Runtime.getRuntime().gc();

1. **What is StringBuffer?**

* StringBuffer is a mutable sequence of characters, (ie) it can be modified.
* It contains some particular sequence of characters at any point of time
* The length and content of the sequence can be changed
* It is thread-safe , as the methods are synchronized. So can be used in multi-threaded applications
* Operations like insert, append, delete can be performed on the Stringbuffer object which returns the modified object

1. **What is StringBuilder?**

* StringBuilder is a mutable sequence of characters,(ie) it can be modified.
* It contains some particular sequence of characters at any point of time
* The length and content of the sequence can be changed
* They cannot be used in multi-threaded applications, as the methods are not synchronized.
* They are faster in performance
* Operations like insert, append, delete can be performed on the Stringbuffer object which returns the modified object

1. **Difference between StringBuffer and StringBuilder**

**StringBuffer:**

* It is thread-safe , as the methods are synchronized. So can be used in multi-threaded applications

**StringBuilder:**

* They cannot be used in multi-threaded applications, as the methods are not synchronized.

**java.io**

1. **What is an IO Stream?**

Stream is flow of data between source and destination. IO Streams represent different sources and destination for reading and writing data.

java.io package provides many classes to perform operations on byte streams and Unicode character streams.

An **inputstream** reads data from source. eg., System.in

An **outputstream** writes data to the destination. eg., System.out

1. **What are ByteStreams? Name classes that support character streams**

* In Byte Streams, the data is in the form of bytes.
* The top level interfaces that support performing operation on byte streams are **InputStream** and **OutputStream**.
* They provide different implementation to read byte data from different sources and write byte data to different destinations

**eg.,**

**FileInputStream** – for reading from a file

**FileOutputStream** – for writing into a file

1. **What are Character Streams? Name classes that support character streams**

* In Character Streams, the data is in the form of Unicode characters.
* The top level interfaces that support performing operation on character streams are Reader and Writer.
* They provide different implementation to read character data from different sources and write character data to different destinations

**eg.,**

**FileReader** – for reading from a file

**FileWriter** – for writing into a file

1. **What is Serialization?**

Serialization is saving the state of an object into a file. It is the process of converting an object into stream of bytes in order to store the object into memory. The reverse process is called as deserialization

1. **How serialization is achieved?**

To achieve serialization, the class must implement java.io.Serializable interface

1. **What are the classes that are used for serialization?**

java.io.ObjectInputStream and java.io.ObjectOutputStream

1. **Identify the methods used for serilaization / deserialization?**

**Serialization:** public void writeObject(Object o)

**Deserialization:** public Object readObject()

1. **What keyword is used to prevent serialization of certain instance variables?**

**'transient'** is the keyword used with the instance variables to avoid serialization of that particular field.

1. **What happens to the sub class objects if only super class implements Serializable?**

The subclass object also will be serialized.

1. **What happens to the super class object if only sub class implements Serializable?**

The super class object will not be serialized.