**Write a program to swap 2 numbers without a temporary variable? Swap 2 strings**

**without a temporary variable?**

**//swap numbers**

**int** a = 10;

**int** b = 20;

a = a + b;// first this should be there a=10+20=30

b = a - b; // b= 30-20=10

a = a - b; // a=30-10=20

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

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

**//swap strings**

String x="Hello";

String y="Welcome";

x=x+y; //HelloWelcome

y=x.substring(0,(x).length()-y.length());

x=x.substring(y.length());

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

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

**2. Write a java program to check whether a given number is prime or not?**

public class primNumber {  
  
 public static void main(String[] args) {  
 int num = 9; *// example number* boolean isPrime = true;  
 if (num <= 1) {  
 isPrime = false;  
 } else {  
 for (int i = 2; i < num; i++) {  
 if (num % i == 0) {  
 isPrime = false;  
 break;  
 }  
 }  
 if (isPrime){  
 System.*out*.println(num+" number is prime");  
 } else {  
 System.*out*.println(num+" number is not prime try again");  
 }  
 }  
 }  
}

**Write a Java Program to print the first 10 numbers of Fibonacci series.**

The Fibonacci series is a sequence of numbers where each number is the sum of the two preceding ones, starting from 0 and 1. The series typically starts like this:  
  
public class fibonacci {  
  
 public static void main(String[] args) {  
 int a, b, c;  
 a = 0; *// First number* b = 1; *// Second number* System.*out*.print(a + " " + b + " "); *// Print the first two numbers* for (int i = 3; i <= 10; i++) { *// Starting from 3rd number* c = a + b; *// Next number in the sequence* System.*out*.print(c + " "); *// Print the next number* a = b; *// Update a to the next number* b = c; *// Update b to the next number* }  
}}

**Find out how many alpha characters are present in a string? Find the number of**

**words in a string?**

public static void main(String[] args) {  
  
 String words="234$%#($345mvdFHF";  
 int characters=words.replaceAll("[^A-z]","").length();  
 System.*out*.println(characters);  
  
 }

---------------------------------------------------------

String myString="Today is Wednesday and it is Java Class";

String[] array=myString.split(" ");

**int** words=array.length;

System.***out***.println("Total words in string:"+myString+" = "+words);

**Write a java program to reverse String?**

public static void main(String[] args) {  
  
 String word= "Hello Syntax";  
 StringBuilder rev=new StringBuilder(word);  
 System.*out*.println(rev.reverse());  
}

**Reverse a string word by word?**

public static void main(String[] args) {  
 String word="please reverse each word but dont change the place of word";  
 StringBuilder obj=new StringBuilder(word);  
 String one=obj.reverse().toString();  
 String [] two=one.split(" ");  
 for (int i =two.length-1; i >0 ; i--) {  
 System.*out*.print(two[i]+" ");  
  
 }

**Write a Java Program to find whether a String is a palindrome or not.**

public static void main(String[] args) {  
 String word="level";  
 StringBuilder obj=new StringBuilder(word);  
 String reve=obj.reverse().toString();  
 if (word.equals(reve)){  
 System.*out*.println("is palindrom");  
 } else {  
 System.*out*.println("not palidtrom");  
 }  
}

**Write a java program to find the second largest number in the array? Maximum**

**and minimum number in the array?**

int [] number={34,5,6,89,78,64,065,4,556,75,4};  
Arrays.*sort*(number);  
System.*out*.println("maximum number in array is "+number[number.length-1]);  
System.*out*.println("second larges number is "+number[number.length-2]);  
System.*out*.println("minimum numbers is "+number[0]);

**How can you remove all duplicates from ArrayList?**

ArrayList<String> aList = **new** ArrayList<>();

aList.add("John");

aList.add("Jane");

aList.add("James");

aList.add("Jasmine");

aList.add("Jane");

aList.add("James");

// 1 way

HashSet<String> set = **new** HashSet<>(aList);

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

**Versions of java you worked with? What version of java do you currently use in**

**your framework? Difference between JRE, JDK ?**

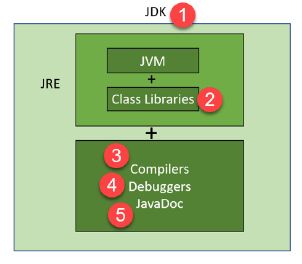
I've worked with various versions of Java, including Java 8, 11, and 17. Currently, in my framework, I am using Java 8, as it is widely supported and stable for most automation projects.

Why not use latest version

Might have bugs

Training cost

Takes time

****Need to upgrade other tools

**JDK (Java Development Kit):**

* **Definition** jdk consists of all the tools for creating , running and debugging Java applications. It contains the JRE (Java Runtime Environment), the Java compiler (javac), a debugger, and other utilities required for developing Java applications.
* **Role**: Provides everything necessary for Java development,
* **Jre (jvm , class libraries)**
* **compiler,**
* **debuggers**
* **javadoc creating, debugging, and running Java applications.**
* **Usage**: Used by developers to write and compile code, debug, and build Java applications.

**SDK (Software Development Kit):**

* **Definition**: A set of software tools and libraries provided for developing applications for a specific platform or framework.
* **Role**: Provides the resources needed to develop applications on a particular platform, including compilers, debuggers, libraries, documentation, and sometimes hardware emulation tools.
* **Usage**: SDKs exist for many platforms (like Android, iOS, Java) and are used by developers to create software tailored for a specific environment.

**Compiler:**

* **Definition**: A software tool that translates source code written in a programming language (like Java) into machine code (binary) or bytecode that can be executed by a computer or a virtual machine (e.g., JVM).
* **Role**: Converts human-readable code into a format that machines can process and run.
* **Usage**: Java's compiler (javac) takes .java files and converts them into bytecode (.class files) which the JVM can run.

**Debugger:**

* **Definition**: A tool used by developers to test and debug their programs by identifying and fixing bugs in the source code.
* **Role**: Allows developers to step through their code, set breakpoints, inspect variables, and understand the execution flow to find and fix errors.
* **Usage**: Java comes with a debugger called jdb that can be used to troubleshoot issues in the code.

**Java Libraries:**

* **Definition**: Collections of pre-written code (in the form of classes and methods) that developers can use to simplify common tasks like file I/O, network communication, data manipulation, and more.
* **Role**: Provides reusable functionality to avoid rewriting commonly used operations from scratch.
* **Usage**: Java libraries include standard libraries (part of the Java Standard Edition) and third-party libraries like Apache Commons, Google Guava, etc.

### JVM (Java Virtual Machine):

* **Definition:** The JVM is a runtime environment that runs Java bytecode, making Java programs platform-independent.
* **Role:** The JVM converts the platform-neutral bytecode into machine-specific instructions, enabling Java applications to run on any device with a JVM.
* **Usage:** It acts as the execution engine for Java applications, ensuring platform independence by interpreting or compiling the bytecode into native machine code.

**JRE (Java Runtime Environment):** It contains everything needed to run Java programs, including the Java Virtual Machine (JVM) and 2. libraries, but it does not include development tools like a compiler or debugger.

**10. What is a method? What is the main method? Why do we need one in java? Do we**

**have to have a main method in java?**

**Method** is a collection of statements that are grouped together to perform an operation.

A method can be called (invoked) at any point in a program by the method's name.

**Main method is the starting point** of an application. JVM starts execution by invoking

the main method of some specified class, passing it a single argument, which is an array

of strings. Whenever we execute a program, the main() is the first function to be

executed. We can call other functions from the main to execute them. It is not mandatory

to have

**Explain public static void main (String args[])?**

**public:** it is an access specified that means it will be accessible by any Class.

**static:** is a keyword to call this method directly using class name without creating an

object of it.

**void:** it is a return type i.e. it does not return any value.

**main():** it is the name of the method which is searched by JVM as a

starting point for an application with a particular signature only.

It is the method where the main executions occur.

**string args[]:** it’s a command line argument passed to the main method.

**What are Access Modifiers (Private,public,protected)? How did you use them?**

Java provides access modifiers to set access levels for classes, variables, methods and

constructors.

**public:** A class or interface may be accessed from outside the package. Constructors,

inner classes, methods and field variables may be accessed wherever their class is

accessed.

**protected:** Accessed by other classes in the same package or any subclasses of the same

package or different package.

**default:** When no access modifier is specified for a class , method or data member – It is

said to be having the default access modifier by default.

**private:** Accessed only within the class in which they are declared.

In our framework we follow page object model design pattern, in page classes we store

WebElements as public to give visibility to our test classes in different package

@FindBy(id="txtUsername")

**public** WebElement username;

**13. What is an instance variable and how do you use it? What is the difference between**

**local and instance variables?**

Variables which are declared **inside a method or constructor or blocks are called local**

**variables.** Local variables are created when a method is called and destroyed when the

method exits.

Variables which are declared **inside the class, but outside a method, constructor or**

**any block are called instance variables.** We can access instance variables by creating an

Object of the class they belong to. Instance variables are created when an object is

created with the use of the keyword 'new' and destroyed when the object is destroyed.

**Example of Instance Variable from the framework**

**@FindBy(xpath="//img[contains(@src, 'logo')]")**

**public WebElement logo;**

**How can we access variables without creating an object instance of it? Difference**

**between Instance Variable and static Variable? What is a static keyword in java?**

**Where did you use static in your framework?**

**1.** Static variables are declared with the **static** keyword in a class, but outside a

method, constructor or a block. By declaring a variable as a static we can access it

from different classes without creating an Object - those variables called class

variables and also known as static variables.

Whereas, **Instance** variables are declared in a class, but outside a method,

constructor or any block. To access instance variables we need to create an object

of the Class they belong to.

**2. Class variables only have one copy** that is shared by all the different objects of a

class, whereas every object has its own **personal copy of an instance variable**.

So, **instance variables across different objects can have different values** and

when we make changes to the instance variable they don't reflect in other

instances of that class **whereas class variables across different objects can have**

**only one value.**

**3.** Static variables are created when the program starts and destroyed when the

program stops whereas instance variables are created when an object is created

with the use of the keyword 'new' and destroyed when the object is destroyed.

**Static keyword in java:**

● Static keyword means that the variable or method belongs to the class and shared

between all instances.

● Using static keyword we can access class variables and methods without object

reference

● Static methods can not call/refer Non Static members

**Usage of static keyword in framework:**

In our utility package we have a class where we store common methods, such as wait,

switch between frames, clicking on buttons, selecting values from drop down. So

those methods are written using static keyword and we can easily access them in our

program.

**public static void click(WebElement element) {**

**waitForClickability(element);**

**element.click();**

**}**

In our Constants Class we have static variables

**public static final String CONFIGURATION\_FILEPATH =**

**System.getPrope 15. What is a constructor? Use of constructor in class? Can you make the constructor**

**static? What is the difference between constructor and method? Can we overload a**

**constructor?**

**What is constructor**

A constructor in Java is a block of code similar to a method. Constructor is called when

an instance of a class is created. A constructor is a “special method” whose task is to

initialize the object of its class.

**Constructors cannot be abstract, final, static.**

Rules to create constructor:

1. Constructor name class name must be the same.

2. Constructor do not have any return type.

3. Constructor may or may not have parameters.

**Usage of Constructor**

The primary use of constructor is to **initialize the instance variables.**

Constructors are special functions which are called automatically when we create objects

of the class. So once we create the object of the class all the variables get initialized, and

we don't need to write extra code for initialization of variables.

Constructor is the property of an object while static has nothing to do with the object.

That's why there is nothing like a static constructor. But we have a static block to do a

similar task as constructor i.e. initialization of static fields etc.

**Difference between Constructor and Method**

● Constructor must not have a return type whereas methods must have a return type.

● Constructor name is the same as the class name where the method may or may not

be the same class name.

● Constructor will be called automatically whenever an object is created whereas

the method invokes explicitly.

● Compiler provides a default constructor when no constructor is created whereas

the compiler doesn’t create any methods.

**Example of constructor from framework**

creating constructor to initialize instance variables

**public class AddEmployeePage {**

**@FindBy(id="firstName")**

**public WebElement firstName;**

**public AddEmployeePage() {**

**PageFactory.initElements(CommonMethods.driver, this);**

**}**

**}**

**WE CAN OVERLOAD CONSTRUCTOR (using different number or type of**

**parameters)rty("user.dir")+"/src/test/resources/config/config.properties";**

**16. Super vs super()? this vs this()? Can super() and this() keywords be in the same**

**constructor?**

**this vs this()**

● **this keyword** is used to refer to the current object and differentiate between local

and instance variables

**public class ThisKeyword {**

**String name;**

**int age;**

**ThisKeyword(String name, int age){**

**this.name=name;**

**this.age=age;**

**}**

**}**

● **this()** is used to access one constructor from another where both constructors

belong to the same class.

**public class ThisKeyword4 {**

**int z;**

**ThisKeyword4() {**

**System.out.println("This a default constructor");**

**}**

**ThisKeyword4(String a) {**

**this();**

**System.out.println("Parameterized constructor);**

**}**

**}**

**super vs super()**

Both are used in a subclass as a way to invoke or refer to its superclass.

● **super keyword** is used to call super class(parent class/ base class)

variables and methods by the subclass object when they are overridden by

subclasses.

● **super()** is used to call a superclass constructor from a subclass constructor.

**public class SuperKeyword1 extends SuperKeyword{**

**SuperKeyword1(){**

**super(4);**

**System.out.println("This is a child default constructor");**

**}**

We can use **super()** and **this()** only in the constructor, not anywhere else, any attempt to

do so will lead to a compile-time error. **This()** and **super()** always have to be in the first

line within the constructor and for that reason we **CANNOT** use them within the same

constructor. We have to keep either super() or this() as the first line of the constructor but

NOT both simultaneously.

**Difference between an abstract class and interface? Can we create an object for an**

**abstract class? interface? When to use abstract class vs interface in Java?**

**Interface** is a blueprint for your class that can be used to implement a class.

Interface is a collection of public static methods and public static final variables

**An abstract class** is a class that is declared with abstract keyword and can

contain defined(concrete) and undefined(abstract) methods.

**Abstract Class Interface**

The abstract keyword is used to declare

abstract class

The interface keyword is used to declare

interface

Abstract class does not support multiple

inheritance

Interface support multiple inheritance

Abstract class contains Constructors Interface doesn't contain Constructors

An abstract class Contains both incomplete An interface Contains only incomplete

(abstract) and complete member and Abstract

class can have abstract and non-abstract

methods.

members (signature of member) and

Interface can have only abstract methods.

(Since Java 8 we can have static and

default methods)

An abstract class can contain access

modifiers for the methods,

properties(variables)

An interface cannot have access modifiers

by default everything is assumed as public

Abstract class can have final, static and

non-static variables.

Interface has only final static variables.

Their properties can be reused commonly in a

specific application.

Their properties are commonly usable in

any application of java environment.

Abstract class may contain either variables or

constants. Interface should contain only constants.

The default access specifier of abstract class

methods is default.

The default access specifiers of interface

methods are public.

These class properties can be reused in other

classes using extends keyword.

These properties can be reused in any other

class using implements keyword.

For the abstract class there is no restriction

like initialization of variables at the time of

variable declaration.

For the interface it should be compulsory to

initialize variables at the time of variable

declaration.

There are no restrictions for abstract class

variables.

For the interface variable can not declare

variable as private, protected

There are no restrictions for abstract class

method modifiers that means we can use any

modifiers.

For the interface method cannot declare

method as protected, private, final

**We cannot create an object of interface or an abstract class !**

● An abstract class is good if you think you will plan on using inheritance since it provides

a common base class implementation to derived classes.

● An abstract class is also good if you want to be able to declare non-public members. In an

interface, all methods must be public.

● If you think you will need to add methods in the future, then an abstract class is a better

choice. Because if you add new method headings to an interface, then all of the classes

that already implement that interface will have to be changed to implement the new

methods. That can be quite a hassle.

**Practical Example of an Interface:**

Basic statement we all know in Selenium is

**WebDriver driver = new FirefoxDriver();**

WebDriver itself is an Interface. We are initializing Firefox browser using Selenium

WebDriver. It means we are creating a reference variable (driver) of the interface

(WebDriver) and creating an Object. Here WebDriver is an Interface and FirefoxDriver

is a class.

**Practical Example of an Interface:**

**public static byte[] takeScreenshot() {**

**TakesScreenshot ts = (TakesScreenshot) driver;**

**byte[] screen = ts.getScreenshotAs(OutputType.BYTES);**

**return screen;**

**}**

**public static void scrollDown(int pixels) {**

**JavascriptExecutor js = (JavascriptExecutor) driver;**

**js.executeScript("window.scrollBy(0," + pixels + ")");**

**}**

**Practical Example of an Interface:**

**public class Listener implements ITestListener {**

**@Override**

**public void onTestStart(ITestResult result) {//for method**

**System.out.println("Starting Test: "+result.getName());**

**}**

**@Override**

**public void onTestSuccess(ITestResult result) {**

**System.out.println("Test case passed: "+result.getName());**

**}**

**@Override**

**public void onTestFailure(ITestResult result) {**

**System.out.println("Test case failed: "+result.getName());**

**}**

**@Override**

**public void onTestSkipped(ITestResult result) {**

**System.out.println("Test case skipped: "+result.getName());**

**}**

**}**

**18. Explain OOPS concepts? Is java 100% object oriented?**

OOP concepts in Java are the main idea behind Java’s Object Oriented Programming.

OOP mainly focuses on the objects that are required to be manipulated instead of logic

4 main OOPS concepts: **inheritance, polymorphism, abstraction and encapsulation.**

**Inheritance** is a mechanism in which one object acquires all the properties and behaviors

of a parent object. It provides code reusability.

**Polymorphism** is the ability of an object to take on many forms. The most common use

of polymorphism in Java is when a parent class reference type of variable is used to refer

to a child class object.

**Abstraction** is a process of hiding the implementation of internal details and showing the

functionality to the users. Abstraction lets you focus on what the object does instead of

how it does it.

**Encapsulation** is a mechanism of binding code and data together in a single unit. We can

hide direct access to data by using a private key and we can access private data by using

getter and setter methods.

No, **Java is not 100% object oriented**, since it has primitive data types, which are

different from objects.

**19. What is inheritance and benefits of it? Types of inheritance? How do you use it in**

**your code?**

**Inheritance**

**●** The process of acquiring properties (variables) &methods (behaviors) from one class to

another class is called inheritance.

● We are achieving the inheritance concept by using extends keyword. Also known as is-a

relationship.

● Extends keyword is providing a relationship between two classes.

● The main objective of inheritance is code extensibility whenever we are extending the

class automatically code is reused.

**Types of Inheritance:**

● **Single Inheritance** - single base class and single derived class.

● **Hierarchical Inheritance** - when a class has more than one child classes (sub

classes)

● **Multilevel Inheritance** - single base class, single derived class and multiple

intermediate base classes.

● **Multiple Inheritance** - multiple classes and single derived class (Possible

through interface only)

● **Hybrid Inheritance** - combination of both Single and Multiple Inheritance

(Possible through interface only)

**Usage of inheritance in real time project**

In our current Cucumber framework we have BaseClass where we initialize the

WebDriver interface. And after we extend the Base Class in other classes such as

PageInitializer and to the Common methods where we have functions to work with Web

Browser.

**What is polymorphism? Types of polymorphism?**

Polymorphism is the ability of an object to take on many forms. Polymorphism allows

us to perform a task in multiple ways.

**Combination of overloading and overriding is known as Polymorphism.**

There are two types of Polymorphism in Java

**1. Compile time polymorphism (Static binding) – Method overloading**

**2. Runtime polymorphism (Dynamic binding) – Method overriding**

**21. Method overloading & overriding? How do you use it in your framework? Any**

**example or practical usage of Run time polymorphism?**

**Method overloading** in Java occurs when two or more methods in the same class have

the exact same name but different parameters (remember that method parameters accept

values passed into the method).

**Overloading:** Same method name with different arguments **in the same class**

**Practical Example of Overloading**

**public static void selectDdValue(WebElement element, String textToSelect) {**

**try {**

**Select select = new Select(element);**

**List<WebElement> options = select.getOptions();**

**for (WebElement el : options) {**

**if (el.getText().equals(textToSelect)) {**

**select.selectByVisibleText(textToSelect);**

**break;**

**}**

**}**

**} catch (UnexpectedTagNameException e) {**

**e.printStackTrace();**

**}**

**}**

**public static void selectDdValue(WebElement element, int index) {**

**try {**

**Select select = new Select(element);**

**int size = select.getOptions().size();**

**if (size > index) {**

**select.selectByIndex(index);**

**}**

**} catch (UnexpectedTagNameException e) {**

**e.printStackTrace();**

**}**

**}**

**Practical Example of Overloading**

**public static void switchToFrame(String nameOrId) {**

**try {**

***driver*.switchTo().frame(nameOrId);**

**} catch (NoSuchFrameException e) {**

**e.printStackTrace();**

**}**

**}**

**public static void switchToFrame(WebElement element) {**

**try {**

***driver*.switchTo().frame(element);**

**} catch (NoSuchFrameException e) {**

**e.printStackTrace();**

**}**

**}**

**public static void switchToFrame(int index) {**

**try {**

***driver*.switchTo().frame(index);**

**} catch (NoSuchFrameException e) {**

**e.printStackTrace();**

**}**

**}**

**Method overriding**

Declaring a method in child class which is already present in the parent class is called

Method Overriding.In simple words, overriding means to override the functionality of an

existing method.

With method overriding a child class can give its own specific implementation to an

inherited method without modifying the parent class method. Assume we have multiple

child classes. In case one of the child classes wants to use the parent class method and the

other class wants to use their own implementation then we can use overriding features.

**Practical Usage:**

1. Implementation of WebDriver interface.

**WebDriver driver = new FirefoxDriver();**

**WebDriver driver = new ChromeDriver();**

2. Implementation of iTestListener interface.

**public class Listener implements ITestListener {**

**@Override**

**public void onTestStart(ITestResult result) {//for method**

**System.out.println("Starting Test: "+result.getName());**

**}**

**@Override**

**public void onTestSuccess(ITestResult result) {**

**System.out.println("Test case passed: "+result.getName());**

**}**

**@Override**

**public void onTestFailure(ITestResult result) {**

**System.out.println("Test case failed: "+result.getName());**

**}**

**@Override**

**public void onTestSkipped(ITestResult result) {**

**System.out.println("Test case skipped: "+result.getName());**

**}**

2. Selenium WebDriver provides an interface WebDriver, which consists of abstract

methods getDriver() and closeDriver(). So any implemented class with respect to the

browser can override those methods as per their functionality, like ChromeDriver

implements the WebDriver and can override the getDriver() and closeDriver().

**22. Can we override/overload the main method? Explain the reason? Can you override**

**the static method? Can we overload and override private methods?**

We cannot override a static method, so we cannot override the main method. However,

you can overload the main method in Java. But the program doesn't execute the

overloaded main method when you run your program; you have to call the overloaded

main method from the actual main method.

Practically I do not see any use of it and we don’t use it in my framework.

**public class MainMethodOverload {**

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

**main(5);//if comment this line nothing will get executed**

**}**

**public static void main(int r) {**

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

**}**

**}**

Static methods are bound with class **it is not possible to override static methods**

**class Parent {**

**static void m1() {**

**System.out.println("parent m1()");**

**}**

**}**

**class Child extends Parent {**

**static void m1() {**

**System.out.println("child m1()");**

**}**

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

**Parent p = new Child();**

**p.m1();**

**}**

**}**

In java not possible to override private methods because these methods are specific to

classes, not visible in child classes.

**23. Can we achieve 100% abstraction in JAVA?**

In JAVA abstraction can be achieved with the help of Abstract Classes and Interfaces.

Using abstract class we can achieve 0 to 100% or partial abstraction.

Using interfaces we can achieve 100% or full abstraction.

**24. What is encapsulation?**

It is the technique of making the fields in a class private and providing access to the

fields via public methods. If a field is declared private, it cannot be accessed by anyone

outside the class, thereby hiding the fields within the class. Therefore encapsulation is

also referred to as data hiding. The main benefit of encapsulation is the ability to modify

our implemented code without breaking the code of others who use our code.

Encapsulation gives maintainability, flexibility and extensibility to our code.

Example from my current framework

**public class ExcelUtility {**

**private static Workbook *wbook*;**

**private static Sheet *sheet*;**

**public static Workbook getWorkbook(String filePath) {**

**try {**

**FileInputStream fis = new FileInputStream(filePath);**

***wbook* = new XSSFWorkbook(fis);**

**} catch (FileNotFoundException e) {**

**e.printStackTrace();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**return *wbook;***

**}**

**public static Sheet getSheet(String filePath, String sheetName) {**

**return getWorkbook(filePath).getSheet(sheetName);**

**}**

**}**

**25. What are the primitives and wrapper classes?**

Primitives are data types in Java. There are a total of **8 primitive data types in Java:**

**byte, short, int, long, float, double, char, boolean.**

Every primitive data type has a class dedicated to it and these are known as wrapper

classes. **These classes wrap the primitive data type into an object** of that class.

**26**

**26. What is collection in Java and what type of collections have you used?**

Java provides Collection Framework which defines several classes and interfaces to

represent a group of objects as a single unit.

**Maps are not part of collection but built based on the collection concepts**

Mostly in my current project we use List and Map.

**Practical Example of List & Map usage**

**public static List<Map<String, String>> excelIntoListOfMaps(String filePath,**

**String sheetName) {**

**List<Map<String, String>> mapList = new ArrayList<Map<>>();**

**for (int r = 1; r < *rCount*(); r++) {**

**Map<String, String> rMap = new LinkedHashMap<>();**

**for (int c = 0; c < *colsCount*(r); c++) {**

**String key = *getCellData*(0, c);**

**String value = *getCellData*(r, c);**

**rMap.put(key, value);**

**}**

**mapList.add(rMap);**

**}**

**return mapList;**

**}**

**Practical Example of List & Map usage**

**public static List<Map<String, String>> storeDataFromDB(String sqlQuery) {**

**try {**

**st = conn.createStatement();**

**rset = st.executeQuery(sqlQuery);**

**ResultSetMetaData rsetData = rset.getMetaData();**

**listData = new ArrayList<>();**

**while (rset.next()) {**

**Map<String, String> rowMap = new LinkedHashMap<>();**

**for (int i = 1; i <= rsetData.getColumnCount(); i++) {**

**rowMap.put(rsetData.getColumnName(i),**

**rset.getObject(i).toString());**

**}**

**listData.add(rowMap);**

**}**

**} catch (Exception e) {**

**e.printStackTrace();**

**}**

**return listData;**

**a) Difference between ArrayList vs LinkedList?**

ArrayList and LinkedList, both implement List interface and provide capability to store

and get objects as in ordered collections. Both are non synchronized classes and both

allow duplicate elements.

**ArrayList**

● ArrayList internally uses a dynamic array to store the elements.

● Manipulation with ArrayList is slow because it internally uses an array. If any

element is removed from the array, all the bits are shifted in memory.

● ArrayList is better for storing and accessing data.

**LinkedList**

● LinkedList internally uses a doubly linked list to store the elements (consist of

value + pointer to previous node and pointer to the next node)

● Manipulation with LinkedList is faster than ArrayList because it uses a doubly

linked list, so no bit shifting is required in memory.

● LinkedList is better for manipulating data.

**//Create linked list**

**LinkedList linkedList = new LinkedList();**

**//Add elements**

**linkedList.add("A");**

**linkedList.add("B");**

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

**//Add elements at specified position**

**linkedList.add(2, "C");**

**linkedList.add(3, "D");**

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

**//Remove element**

**linkedList.remove("A"); //removes A**

**linkedList.remove(0); //removes B**

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

**b) ArrayList vs Vector?**

Both implement List Interface and maintains insertion order

ArrayList is not synchronized, so it is fast.

Vector - is synchronized, so it is slow.

**28. What is the difference between HashSet vs HashMap ?**

**HashSet**

1. HashSet class implements Set interface

2. In HashSet, we store objects(elements or values).

3. HashSet does not allow duplicate elements that mean you cannot store duplicate

values in HashSet.

4. HashSet permits to have a single null value.

5. HashSet is not synchronized.

**//create HashSet**

**HashSet hset = new HashSet();**

**hset.add("BMW");**

**hset.add(2018);**

**How to print all values from hashSet**

**// 1. advance loop**

**for (Object obj : hset) {**

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

**}**

**// 2. using iterator**

**Iterator itr = hset.iterator();**

**while (itr.hasNext()) {**

**Object words = itr.next();**

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

**}**

**HashMap**

1. HashMap class implements the Map interface

2. HashMap is used for storing Key, Value paired objects.

3. HashMap does not allow duplicate keys however it allows having duplicate

values.

4. HashMap permits a single null key and any number of null values.

5. HashMap is not synchronized.

**Bonus:**

**ArrayList vs HashSet?**

Both ArrayList and HashSet are non synchronized collection class

Both ArrayList and HashSet can be traversed using Iterator

**ArrayList**

● ArrayList implements List interface

● ArrayList allows duplicate values

● ArrayList maintains the order of the object in which they are inserted

● In ArrayList we can add any number of null values

● ArrayList is index based

**HashSet**

● HashSet implements Set interface

● HashSet doesn’t allow duplicates values

● HashSet is an unordered collection and doesn’t maintain any order

● HashSet allow one null value

● HashSet is completely object based

**29. What is a Map? How did you use it in your framework?**

Java Map Interface. A map contains values on the basis of key, i.e. key and value pairs.

Each key and value pair is known as an entry. Map is a collection of entry objects.

A Map Contains unique keys. A Map is useful if we have to search, update or delete

elements on the basis of a key.

The Map interface is implemented by different Java classes, such as HashMap,

HashTable, LinkedHashMap and TreeMap.

**HashMap**: it makes no guarantees concerning the order of iteration, HashMap doesn’t

maintain the insertion order of elements.

**LinkedHashMap**: It orders its elements based on the order in which they were inserted

into the set (insertion-order).

**TreeMap**: It stores its elements in a red-black tree, orders its elements based on their

values; it is substantially slower than HashMap.

**HashTable**: it also stores the data in a key-value pair but the HashTable store only

non-null objects i.e any key or value can’t be Null. Hashtable is similar to HashMap

except it is synchronized or we say thread safe.

**Map<String, Integer> groceryMap=new LinkedHashMap<>();**

**groceryMap.put("Milk", 1);**

**groceryMap.put("Bread", 2);**

**groceryMap.put("Ice Cream", 10);**

**groceryMap.put("Cookie", 5);**

**groceryMap.put("Tea", 3);**

**for(Map.Entry<String, Integer> entry:groceryMap.entrySet()) {**

**if (entry.getKey().contains("Tea")) {**

**entry.setValue(2);**

**}**

**}**

**for (String key:groceryMap.keySet()) {**

**if(key.contains("Cookie")) {**

**groceryMap.replace(key, 3);**

**}**

**}**

**System.out.println(grocery**

**30. What is the difference between HashTable vs HashMap ?**

Both **HashMap** and **Hashtable** implement Map Interface

**HashMap**

● HashMap is non synchronized, so it is not-thread safe

● HashMap is fast

● HashMap allows one null key and multiple null values

**HashMap <Integer, String> hmap=new HashMap<Integer, String>();**

**hmap.put(1, "January");**

**hmap.put(1, "January");**

**hmap.put(2,"February");**

**hmap.put(null, "February");**

**hmap.put(null, "February");**

**for (Map.Entry obj: hmap.entrySet()) {**

**System.out.println(obj.getKey()+" "+obj.getValue());**

**}**

**Output:**

null February

1 January

2 February

**Hashtable**

● Hashtable is synchronized, so it is thread-safe

● Hashtable is slow

● Hashtable doesn’t allow any null key or value

**Hashtable <Integer, String> htable=new Hashtable<Integer, String>();**

**htable.put(1, "January");**

**htable.put(1, "January");**

**htable.put(2, "February");**

**htable.put(null, "February");**

**htable.put(null, "February");**

**for (Map.Entry obj: htable.entrySet()) {**

**System.out.println(obj.getKey()+" "+obj.getValue());**

**}**

**Output:**

Exception in

**31. How can you handle exceptions? Types of exceptions you faced in your project?**

**What is the parent of all exceptions?**

**An Exception is a problem that can occur during the normal flow of execution.**

Depending on the situation, we can use try and catch blocks.

In **try block:** Code that might throw some exceptions

In **catch block:** We define exception type to be caught and what to do if an exception

happens in TRY block code

**Throwable class is parent of all Exceptions:**

**try {**

**int a=10;**

**int b=0;**

**int c=a/b;**

**}catch (ArithmeticException e) {**

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

**}**

**Types of Exception:**

**1. Checked Exception -** are the exceptions that are checked at compile time.

Example of checked exceptions:

● **ClassNotFoundException** - Class not found

● **InstantiationException** - Attempt to create an object of an abstract class or

interface

**● FileNotFoundException -** Attempt to open file that doesn’t exist or open file to

write but have only read permission

**2. Unchecked Exception -** are the exceptions that are not checked at compile time, they are

Runtime Exceptions.

**Exception faced as part of java perspective:**

● **ArithmeticException** - Arithmetic error, such as divide-by-zero.

● **ArrayIndexOutOfBoundsException** - Array index is out-of-bounds.

● **NullPointerException** - Invalid use of a null reference.

● **IllegalArgumentException** - Illegal argument used to invoke a method.

**Example of exception handling from current framework:**

**private static Properties *prop*;**

**public static void readProperties(String filePath) {**

**try {**

**FileInputStream fis = new FileInputStream(filePath);**

***prop* = new Properties();**

***prop*.load(fis);**

**fis.close();**

**} catch (FileNotFoundException e) {**

**e.printStackTrace();**

**} catch (IOException e) {**

**e.printStackTrace();**

**}**

**}**

**32. How many catch blocks can we have? Which catch block will get executed if you**

**get ArithmeticException?**

There can be any number of catch blocks for a single try block and it is not necessary that

each try block must be followed by a catch block. It should be followed by either a catch

block or a finally block.

However only the catch block encountered first on the call stack that satisfies the

condition for the exception will be executed for that particular exception, rest will be

ignored.

**try {**

**int a=10;**

**int b=0;**

**int c=a/b;**

**}catch (ArithmeticException e) {**

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

**}catch (Exception e) {**

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

**}**

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

throw and throws are two keywords related to Exception feature of Java.

**Throws :**

**●** is used to declare an exception, which means it works similar to the try-catch

block.

● is used in method declaration.

● is followed by exception class names.

● you can declare multiple exception with throws

● throws declare at method it might throws Exception

● used to handover the responsibility of handling the exception occurred in the

method to the caller method.

**1 Example:**

**public void readPropFile() throws FileNotFoundException, IOException{**

**Properties prop=new Properties();**

**FileInputStream fis=new FileInputStream("fileNamePath.properties");**

**prop.load(fis);**

**}**

**2 Example:**

**public class Test {**

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

**Test test = new Test();**

**test.company();**

**}**

**void studentDetails() throws InterruptedException {**

**System.out.println("Sumair is sleeping");**

**Thread.sleep(3000);**

**System.out.println("Please do not disturb.....");**

**}**

**void syntax() throws InterruptedException {**

**studentDetails();**

**}**

**void company() throws InterruptedException {**

**syntax();**

**}**

**}**

**Throw :**

● is used in the method body to throw an exception

● throw is followed by an instance variable

● you cannot declare multiple exceptions with throw

● The throw keyword is used to handover the instance of the exception created by

the programmer to the JVM manually.

● throw keyword is mainly used to throw custom exceptions.

**Example**

**public static WebDriver setUp() {**

**ConfigsReader.*readProperties*(Constants.*CONFIGURATION\_FILEPATH*);**

**switch (ConfigsReader.*getProperty*("browser").toLowerCase()) {**

**case "chrome":**

**System.*setProperty*("webdriver.chrome.driver",**

**Constants.*CHROME\_DRIVER\_PATH*);**

***driver* = new ChromeDriver();**

**break;**

**case "firefox":**

**System.*setProperty*("webdriver.gecko.driver",**

**Constants.*GECKO\_DRIVER\_PATH*);**

***driver* = new FirefoxDriver();**

**break;**

**default:**

**throw new RuntimeException("Browser is not supported");**

**}**

**return *driver*;**

**}**

**35. What is the difference between String and StringBuffer? String and StringBuilder?**

**What is mutable and immutable? StringBuffer vs StringBuilder?**

The most important difference between **String and StringBuffer** in java is that **String**

**object is immutable** whereas **StringBuffer object is mutable.** Once a String Object is

created **we cannot change** it and everytime we change the value of a String there is

actually a new String Object getting created. For example we cannot reverse string

directly, only through using StringBuffer class.

There are **2 ways to make String mutable:** 1. by using **StringBuffer** 2. by using

**StringBuilder**.

The StringBuffer and StringBuilder Class are mutable means we can change the value of

it without creating a new Object. Objects of StringBuilder and StringBuffer Classes live

inside **heap memory.**

**immutability vs. mutability**

✓ String is an immutability class. It means once we are creating String objects it is not

possible to perform modifications on existing objects. (String object is fixed object)

✓ StringBuffer and StringBuilder are mutability classes. It means once we are creating

StringBuffer/ StringBuilder objects on that existing object it is possible to perform

modification.

**class Test {**

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

**String a="Hello";**

**String b="Hello";**

**StringBuffer sb=new StringBuffer("Hello Syntax");**

**a=a.concat(" Syntax");**

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

**sb=sb.append(" Technologies");**

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

**}**

**}**

**StringBuffer vs StringBuilder?**

Both Classes are mutable, except **StringBuffer is thread-safe (synchronized)** and

**StringBuilder is not thread-safe (non synchronized)** which makes **StringBuilder**

**faster compared to StringBuffer.**

**36. What is singleton and have you used the singleton concept in your project ?**

A singleton class is a class that can have only one object (an instance of the class) at a

time. After the first time, if we try to instantiate the Singleton class, the new variable also

points to the first instance created. So whatever modifications we do to any variable

inside the class through any instance, it affects the variable of the single instance created.

● Singleton pattern restricts the instantiation of a class and ensures that only one

instance of the class exists in the Java virtual machine.

● The singleton class must provide a global access point to get the instance of the

class.

● Singleton pattern is used for logging, drivers objects

Example:

**public class SingletonExample {**

**//static member holds only one instance of the singleton class**

**private static SingletonExample singletonInstance;**

**//creating private constructor to prevent instantiation**

**private SingletonExample(){**

**}**

**//create public method to return an instance of the class**

**public static SingletonExample getInstance() {**

**singletonInstance=new SingletonExample();**

**return singletonInstance;**

**}**

**}**

**In my current project we use a singleton concept for our page classes where we store**

**page elements.**

**37. What is a garbage collector and how to make a call to the Garbage Collector?**

Garbage collection is the process of looking at heap memory and identifying which

objects are in use and which are not and deleting unused objects. Once an object is

created it uses some memory and the memory remains allocated until there are

references for the use of the object.When there are no references to an object, it is

assumed to be no longer needed. There is no explicit need to destroy an object as Java

handles the deallocation automatically by using the Garbage Collection process.

Garbage collection in Java happens automatically during the lifetime of the program.

You can call Garbage Collector explicitly, but JVM decides whether to process the call

or not. Ideally, you should never write code dependent on call to garbage collector.

JVM internally uses some algorithm to decide when to make this call. When you make

call using System.gc(), it is just a request to JVM and JVM can anytime decide to ignore

it

In the context of **thread safety**, multiple threads run on the CPU. If a program has multiple threads, the CPU manages their execution, potentially allowing them to run in parallel (on multiple cores) or switching between them quickly (on a single core), which can lead to thread safety issues if not managed properly.