



JAVA Q&A

PROGRAMMING KNOWLEDGE IS A MUST-HAVE NOW



“Test Automation & Programming skills will surely give a boost to your Testing career.”



Is Java a compiled language or an interpreted one?

Both. Yes, Java is both compiled as well as interpreted language. This is what makes it platform-independent. Java Code >> Compile (javac) >> Bytecode (portable code) >> Interpret (platform specific) >> Machine Code >> Execute.

Normal:

1. Java Compiler (javac) compiles Java code to Bytecode (which is platform-independent).
2. Java Interpreter (java) interprets this bytecode (line-by-line) & convert it to Machine language.
3. Execute.

Exception: JIT (Just In Time) Compiler

1. JVM maintains the count for no of times a function is executed.
2. If it exceeds the limit then JIT directly compiles the Java code into Machine language. No Interpretation.

In General,

- Compile: Source code >> Optimized Object Code (can be machine code or other optimized code)
- Interpret: Source Code >> Machine Code (to be executed)

What are bin & src folders in Java Automation?

'bin' and 'src' are not exclusive to Test automation. These are generic Java folders created to segregate the written-code and the compiled-code.

- **src**: Folder where our written code resides. Human readable code files.
- **bin**: Folder where the compiled files reside, that the Java Virtual Machine (JVM) executes... E.g. .class, .jar etc.

In simple words: Developer writes the Java code, i.e. .Java files >> These .Java files are stored in src folder >> Compile the code >> Java code is converted to Byte-code which can be read by JVM, i.e. .class files stored in bin folder.

JDK vs JRE vs JVM

JDK: Java Development Kit which provides the environment to develop and execute (run) the Java program.

- Development Tools (provide an environment to develop your java programs)
- JRE (to execute your java program).

It includes - Java Runtime Environment (JRE), an interpreter/loader (Java), a compiler (javac), an archiver (jar), a documentation generator (Javadoc) and other tools needed in Java development.

JRE: Java Runtime Environment provides the minimum requirements for executing (not develop) a Java application; it consists of the Java Virtual Machine (JVM), core classes, and supporting files. Note: You might have noticed that you associate a JRE [which contain all the libraries] with your test automation framework.

JVM: Whatever Java program you run using JRE or JDK goes into JVM and JVM is responsible for executing the java program [byte code] line by line.

Understanding < **public static void main(String[] args)** >

- **public:** Access modifier, making the main() method available globally so that JVM can invoke it from outside the class [as it is not present in the current class].
- **static:** Keyword, making the main() method class-related [not object] so that JVM can invoke it without instantiating the class.
- **void:** Keyword, used to specify that a method doesn't return anything. As soon as the main() method terminates, the java program terminates too.
- **main:** Identifier, that the JVM looks for as the starting point of the java program. It's not a keyword.
- **String[] args:** Parameter, stores Java command line arguments in an array of type String class. Note: the name <args> is not fixed, user can use any name.

Java 'This' keyword

For the Page Object classes, we generally define the constructor like,

```
public class BasePage
{
    WebDriver driver;
    public BasePage(WebDriver driver) throws Exception
    {
        this.driver = driver;
    }
}
```

Any idea why we use 'this' keyword? 'this' keyword in Java is used to refer to the current Object whose Method or constructor is being invoked. Simply stated – to distinguish between current object's instance variable and method parameter. i.e.

- static WebDriver driver;
- public BasePage(WebDriver driver)

Both statement contains the 'driver' variable. 'this' keyword here is used to refer to the current object's instance variable (static WebDriver driver), i.e. to refer current object's variable within the method.

What's 'Static' in Java? Why to use it?

Static, as in fixed. Applying it to Object-Oriented tech - something that doesn't change for every object, it's fixed for all objects.

Generally, fields are created separately in memory for each instance of a class, i.e., Object variables. But anything declared using static keyword belongs to the class instead of individual instances (objects). What does it mean? That every object of this class shares this same copy of the variable, method, etc. We can apply static keyword with variables, methods, blocks and nested class. The benefit – memory management of course.

```
public class Student{
    private String Name; //Object variable
    private int Age; //Object variable
    private String StudentId; //Object variable
    public static int NumberOfStudents = 0; //Class variable

    public Student(String name, int age, String studentId) {
        this.Name = name;
        this.Age = age;
        this.StudentId = studentId;
        NumberOfStudents++; //Increase the no of students whenever an object is
        created.
    }
}
```

The most common example is `<< public static void main(String args[]) >>` declared static because it must be called before any object exists. Making a method static in Java is an important decision. Does it make sense to call a method/variable, even if no object has been constructed yet? If so, it should be static.

Static entity,

- Will be initialized first, before any class objects are created.
- Is accessed directly by the class name and doesn't need any object.
- Can access only static data. It cannot access non-static data (instance variables).
- Can call only other static methods and cannot call a non-static method.

Caution: Generally, it is bad practice to set the WebDriver instance as static. Instead create a base class that each test classes extend so that each test class has its own instance of WebDriver to be used (this is especially important with parallel execution), then just declare/define your WebDriver variable within the base class.

Can Static methods be over-ridden in Java?

Over-riding [run-time polymorphism] - A subclass provides an implementation of a method in superclass and which implementation to execute is decided at run-time according to the object [subclass OR superclass object] used for call.

No! Static methods cannot be over-ridden. Though you can declare a method with the same signature in a subclass.

Strange? Isn't that over-riding? No! It won't be overridden in the exact sense, instead, that is called 'method hiding'.

Why? Because there won't be any run-time polymorphism. Method call will be resolved at compile time itself depending on the class used to create object.

Note: Don't confuse it with overloading [compile-time polymorphism]. Static methods can be overloaded with same name, but different parameters.

What does 'Abstract' mean in Java?

Abstraction is a process of hiding the implementation details and only show important functionality. 'abstract' keyword is used to create an abstract class and method. The purpose of an abstract class is to specify the default functionality of an object and let its sub-classes to explicitly implement that functionality. Thus, it stands as an abstraction layer that must be extended and implemented by the corresponding sub-classes.

- Abstract class can't be instantiated (it needs to be extended) << abstract class A {} >> | an abstract method contains a method signature, but no method body. << abstract void methodName(); >>
- Abstract class is used to provide default/common method implementation to all the subclasses.
- If you are extending any abstract class that have abstract methods, you must either provide the implementation of all abstract methods or make this sub-class abstract.
- An abstract class can have both abstract and non-abstract (or concrete) methods.
- If a class have abstract methods, then the class should also be abstract.
- An abstract class can have data member, abstract method, concrete method (body), constructor and even main() method.
- An abstract class may have static fields and static methods.
- Abstract method can never be final and static.

Abstract class and methods are usually declared where two or more subclasses are expected to do a similar thing in different ways through different implementations. Most common e.g. Shape as the abstract class >> its implementation provided by the Rectangle and Circle classes.

Note: Abstract classes are not Interfaces. They are different.

Some pointers about Interface!

- Interface can't have constructor. Cannot create object for interface.
- The fields of interfaces are public-static-final. Cannot declare as private or protected and must be initialized (final).
- Interface cannot implement another interface. Interface cannot extend a class. But can extend other interfaces.
- Interface cannot be declared as final (otherwise it will block extending them in other interfaces).
- An interface allows only public, abstract, default & static modifiers in a method declaration. No final. static & default methods added since Java 1.8.
- Cannot reduce the visibility of the methods [public] while implementing (overriding a method) an interface.
- A class can implement multiple interfaces.

Why can't we create constructor for interface?

No, interface cannot have constructors. In order to call any method, we need an object – and since there is no need to have object of interface, there is no need of having constructor in interface.

Note: Constructor is being called during creation of object.

The impact of Java final keyword on variable-method-and-class.

- **final variable**: nothing but constants. We cannot change the value of a final variable once it is initialized. It will result in compilation error in case you try to modify the value.

Note: final variable that is not initialized at the time of declaration must be initialized in the constructor otherwise it will throw a compilation error.

- **final method**: cannot be overridden. Even though a sub class can call the final method of parent class, but it cannot override it [compilation error].
- **final class**: cannot extend a final class.

Trivia: final, finally and finalize are three different terms. finally is used in exception handling and finalize is a method that is called by JVM during garbage collection.

The impact of Java static keyword on variable-method-and-class.

Static members belong to the class instead of a specific instance, i.e., access it without object.

- **static variable:** also known as class variables, i.e., common to all the instances (or objects) of the class. Only a single copy of static variable is created and shared among all the instances of the class.
- **static method:** can access static variables without using object (instance) of the class. Also, they can directly call only other static methods.
- **Static block:** to do the computation in order to initialize static variables, i.e., declare a static block that gets executed exactly once, when the class is first loaded.
- **static class:** A class can be made static only if it is a nested class. It doesn't need a reference of Outer class. Note: a static class cannot access non-static members of the Outer class.

Note: static is used for a variable or a method that is same for every instance of a class.

Java Collections

Collections. A 'Group of things'. It can be a group of variables, constants, classes, objects, anything. Since it is all about 'Objects' in 'Object Oriented programming' - it has to be a collection of Objects.

Set, List, Queue, ArrayList, Vector, LinkedList, HashSet, LinkedHashSet, TreeSet, etc. are all Collections. These form a part of the 'Java Collection Framework', a set of interfaces & classes to ease your implementation of different data structures. Each collection holds the data of some type structured as per its naming convention.

When to use different Collection types?

Collection framework provides interfaces and class implementations that enable easy data handling, i.e. store, retrieve and manipulate the data very effectively and easily. The basic types are Lists, Set & Maps.

Lists: no of items written one-by-one, usually ordered. Can be accessed using index. Say you want to store a list of web elements (e.g., all the hyperlinks). Implementations - ArrayList, LinkedList, Vector.

Set: a collection of unique items, i.e., no duplicates. Say you don't want duplicates in a database. Implementations - HashSet, LinkedHashSet, TreeSet.

Maps: a collection of 'Key - Value' pairs. Say you want to save key-value pairs like,
• Username – STS | Password - ***** | Environment - SIT URL | Browser - Firefox
Implementations - HashMap, LinkedHashMap, TreeMap.

The specific type to use depends on Test requirement, Test data management, multi-threading, sync-async, performance, etc.

What's the difference between Java Map and Hashmap?

Simply stated – Map is an interface, whereas HashMap is one of the implementations of this Map Interface.

- **Map Interface:** public interface Map<K,V>

An object that maps keys to values. A map cannot contain duplicate keys; each key can map to at most one value. The Map interface provides three collection views, which allow a map's contents to be viewed as a set of keys, collection of values, or set of key-value mappings.

- **Hashmap:** public class HashMap<K,V> extends AbstractMap<K,V> implements Map<K,V>, Cloneable, Serializable

Hash table-based implementation of the Map interface. This implementation provides all of the optional map operations, and permits null values and the null key. (The HashMap class is roughly equivalent to Hashtable, except that it is unsynchronized and permits nulls.) This class makes no guarantees as to the order of the map; in particular, it does not guarantee that the order will remain constant over time.

Difference between Java array and ArrayList?

'ArrayList' is one of the class in Java Collection framework which was introduced as a dynamic array. Since an 'array' is static in nature i.e., you cannot change its size once created, So, if you need an array which can resize itself then you should use the ArrayList. Add OR Delete objects from an ArrayList as required.

ArrayList provides more ways for iteration i.e., accessing all elements one by one. You can only use loop e.g., for, while, enhanced for and do-while to iterate over an array but you can also use Iterator and ListIterator class to iterate over ArrayList.

Difference between Java Set and List?

- List is an ordered collection that maintains the elements in insertion order while Set is a type of unordered collection, so elements are not maintained any order.
- List allows duplicates while Set doesn't allow duplicate elements.
- List permits any number of null values while Set permits only one null value in its collection.

List implementations: ArrayList, LinkedList etc.

Set implementations: HashSet, LinkedHashSet, TreeSet etc.

Return type of Array and List

Both Array and List returns an Object of respective classes. Array extends the "Object" class.

The use of Hashmap

Hashmap collection type is used extensively in an automation framework & script. Why? Because of its simplicity & easy usage.

A Map-based collection class like Hashmap used for storing <Key & value> pairs makes it easy to capture & access the values. For example – using a Hashmap to store/access application's page labels & values. Tool-Selenium | Scripts-300 | Status-Automated | Environment-QA. As simple as that.

It also resembles how we, as humans, actually read values – we look at the field label or a header and then the value. Easy to access the value via field label.

How important is 'Exception Handling' in an automation framework?

Yeah, you don't want your test script to end abruptly without any info to debug. It leads to a lot of frustration later on.

The best approach is to use 'exception handling' to properly handle any exceptions thrown during the test execution.

You might want to continue execution despite minor exception, or follow an alternate path, or print a custom message or simply fail the test script with a screenshot.

Java's exception handling keywords – try, catch, throw, throws and finally – comes to rescue here.

It is important that you use them intelligently in order to build a robust framework which is easy to debug and report any exceptions.

Java Catch, Try, Finally, Throw

Exception handling is really important. You don't want your script execution to end abruptly without any info to debug.

Java's exception handling keywords – try, catch, throw, throws and finally comes to rescue here.

- **Try:** try block contain statements which may generate exceptions.
- **Catch:** catch block defines the action to be taken, when an exception occurs.
- **Throw:** used to throw an exception explicitly. Execution stops on encountering throw statement, and the closest catch statement is checked for matching type of exception.
- **Throws:** Any method that is capable of causing exceptions must list all the exceptions possible during its execution, so that anyone calling that method gets a prior knowledge about which exceptions are to be handled. Otherwise, you get compile time error saying unreported exception XXX must be caught or declared to be thrown.
- **Finally:** exception occur or not, finally block will always execute. I.e., run any cleanup type statements, no matter what happens in the protected code.

Java Exception class hierarchy

All objects within the Java exception class hierarchy extend from the Throwable superclass.

- indirectly thrown by the JVM
- can be directly thrown via a throw statement.
- only Throwables (or an inherited subclass) can be caught via a catch statement.

Throwable > Errors | Exceptions

- **Error:** "indicates serious problems that a reasonable application should not try to catch.", i.e., general practice is not to explicitly catch Error classes in code, since they should be dealt with through a change in the application architecture or refactoring.

E.g., AssertionError | LinkageError | ThreadDeath | VirtualMachineError

- **Exception:** “conditions that a reasonable application might want to catch.”, i.e., typical errors that occur from time to time in most applications. E.g., `ArithmeticExceptions` and `IllegalArgumentException` are found in the `Exceptions` subclass category.

E.g., `CloneNotSupportedException` | `InterruptedException` |
`ReflectiveOperationException` | `RuntimeException`

How to create a custom exception in Java?

Java exceptions cover almost all general exceptions; however, we sometimes need to supplement these standard exceptions with our own – say Business logic exceptions OR if you wish to catch and provide specific treatment to a subset of existing Java exceptions.

How to create? Create a class which extends `java.lang.Exception` [checked] OR `java.lang.RuntimeException` [unchecked] class and create a constructor which calls the ‘super’ constructor.

Example,

```
public class CustomException extends Exception
{
    public CustomException(String errorMessage)
    {
        super(errorMessage);
    }
}
```

Note: We can call super constructor without any parameter or can also add a `java.lang.Throwable` parameter.

What's POJO?

POJO - Plain Old Java Object. As the name says – it's an ordinary Java Object, not a special object (Enterprise JavaBean).

POJO is usually used to describe a class that doesn't need to be a subclass of anything, or implement specific interfaces, or follow a specific pattern. I.e., only have a default constructor, private property and corresponding setter and getter methods.

Why use POJO? For effective encapsulation & abstraction - in public classes use setter/getter methods instead of public fields. An example would be to create a POJO (with set/get methods) to store the test data for a test script.

Strings are immutable in Java, why? And how to make them mutable?

Immutable, i.e., unchanging over time or unable to be changed. As simple as that.

Once created, you cannot change the value of String object in Java. It's final. It is set aside in a memory area called "String constant pool". You can just have multiple reference variables pointing to the same object – but you cannot change its value.

```
String s1 = "java";  
s1.concat(" rules");  
System.out.println(s1); --- Yes, it will still print "java".
```

The benefit?

- **Security:** it's easier to operate with sensitive code when values don't change.
- **Synchronization:** immutable objects can be shared across multiple threads running simultaneously. They're thread-safe.
- **Hashcode:** the hash is calculated and cached during the first hashCode() call and the same value is returned ever since.
- **Performance:** "String constant pool" enhances the performance by saving heap memory and faster access.

How to make them mutable? use StringBuffer [thread-safe] OR StringBuilder class instead.

```
StringBuffer s1 = "java"; s1.append(" rules"); System.out.println(s1);  
StringBuilder s1 = "java"; s1.append(" rules"); System.out.println(s1);
```

This shall give you "java rules".

Difference between defining `s = "text"` and `new String("text")`

The basic difference is,

- `new String("text");` explicitly creates a new and referentially distinct instance of a String object;
- `String s = "text";` may reuse an instance from the string constant pool if one is available.

You very rarely would ever want to use the `new String(anotherString)` constructor.

"String(String original): Initializes a newly created String object so that it represents the same sequence of characters as the argument; in other words, the newly created string is a copy of the argument string. Unless an explicit copy of original is needed, use of this constructor is unnecessary since strings are immutable".

Any Java design pattern used in your automation framework?

A design pattern systematically explains a general design that addresses a recurring design problem in object-oriented systems. You might not be aware, but we do use some popular design patterns in test automation,

- **Factory pattern:** The most common – PageFactory - returns a Page Object with its fields fully initialized. It follows the Java Factory pattern - create object without exposing the creation logic.
- **Builder pattern:** Ever used `'(new Actions(driver)).moveToElement(element).click().perform()'`? Yeah! That's the builder pattern. Built on top of method-chaining [invoke multiple methods on the same object as a single statement, since methods return 'this' reference for a class instance] to ensure the thread-safety and atomicity of object creation. The inner static class is used for setter methods and then build to return an instance of outer class.
- **Singleton pattern:** restrict the instantiation of a class and ensures that only one instance of the class exists in the JVM. How? Make the constructor private and return a static class-object from your `getInstance()` method. Say, I login to rest [start services] before the start of TC execution and then use the same object [getInstance] to perform rest operations within the TC.

Java Collections – Map | Set | List.

The Collections Framework [collection of interfaces and classes] help in storing and processing the data efficiently.

- **List:** an ordered Collection (a sequence) where elements can be inserted or accessed by their position in the list, using a zero-based index. E.g., ArrayList | LinkedList | Vector. You might have used List to store a list of Web Elements.
- **Set:** A Collection that cannot contain duplicate elements. E.g., HashSet | TreeSet | LinkedHashSet. Might have used Set to operate on Window handles (since it doesn't allow duplicates).
- **Map:** A Collection that maps keys to values [cannot contain duplicate keys]. E.g., HashMap | TreeMap | LinkedHashMap. Might have used HashMap for data storage-read-and-update operations.

How is Set/HashSet implemented in the backend to achieve uniqueness?

Set: a collection of distinct objects, i.e., a set will never contain duplicate elements.

Basic: When we try to add a duplicate element to a set using add() method, it returns false, and element is not added to hashset, as it is already present.

How add() method checks whether the set already contains the specified element?

Logic: A HashSet internally creates a HashMap | The add() method of HashSet actually calls put() method on internally created HashMap object – key: specified element + value: constant Object "PRESENT".

- **HashMap put(Key, Value)** - returns the previous value associated with key OR null [if there is no mapping].
- **HashSet add()** - check the return value of map.put(key, value) with null value to identify if the element is already present or not.

How are Java objects stored in memory?

The Java heap space is the area of memory used for dynamic memory allocation to store objects instantiated by applications running on the JVM.

- **Young Generation** – where all new objects are allocated and aged. A minor Garbage collection occurs when this fills up.
- **Old or Tenured Generation** – where long surviving objects are stored. When object in young generation reach its threshold age, the object is moved to the old generation.

Some pointers,

- If heap space is full, Java throws `java.lang.OutOfMemoryError`
- Heap space needs a Garbage Collector to free up unused objects.
- Use command "jmap" to take Heap dump and "jhat" to analyze that heap dump. [Use Profiler and Heap dump Analyzer tool to understand Java Heap space and how much memory is allocated to each object]
- For strings, we have a 'String Constant Pool' memory area allocated within Heap space.

Note: Java Heap space is different than Stack which is used to store call hierarchy and local variables.

Explain what collections you have used in project?

- **List:** to store a list of web elements (when a locator returns more than 1 web element using *driver. Findelements*)
- **HashMap:** to store config key-value pairs like Env-URL | Username: name | Password: pwd | Client: XYZ | etc.
- **Set:** Generally used to store WindowHandles since it doesn't allow duplicate values.

Which mechanism is used in HashMap?

It works based on the hashing principle - mechanism of assigning unique code to a variable or attribute using an algorithm to enable easy retrieval. A true hashing mechanism should always return the same `hashCode()` when it is applied to the same object.

HashMap has an inner class called an Entry Class which holds the key and values.

- **Get:** First, it gets the hash code of the key object, which is passed, and finds the bucket location.
- **Put:** applies the hashcode to the key >> indexOf() method is used to get the exact location to store the Entry object >> if key.equals(k) is true, it will replace the value object inside the Entry class and not the key. This way, it prevents the duplicate key from being inserted.

Can we put duplicate values in any Set, Will it give any error?

The meaning of "sets do not allow duplicate values" is that when you add a duplicate to a set, the duplicate is ignored, and the set remains unchanged. This does not lead to compile or runtime errors: duplicates are silently ignored.

Difference between ArrayList and HashMap in Java?

- ArrayList implement List Interface while HashMap is an implementation of Map interface.
- ArrayList maintains the insertion order while HashMap does not maintain insertion order.
- ArrayList allows duplicate elements while HashMap doesn't allow duplicate keys but does allow duplicate values.

Does list maintain insertion order?

Both the ArrayList and LinkedList maintain the elements insertion order which means while displaying ArrayList and LinkedList elements the result set would be having the same order in which the elements got inserted into the List.

Difference between an array and an ArrayList in Java?

The main difference between array and ArrayList is that the array is static and the arraylist is dynamic. We cannot change the size of the array once created, but ArrayList can increase its size automatically.

Differences between ArrayList and LinkedList?

- LinkedList can be iterated in reverse direction using `descendingIterator()` but to iterate over the ArrayList in reverse direction we need to write our own code.
- Insertions and Deletions are faster in LinkedList as compared to ArrayList because there is no need of resizing array and copying content to new array if array gets full which makes adding into ArrayList of $O(n)$ in worst case, while adding is $O(1)$ operation in LinkedList in Java. So, if we have more add or delete operations then we should go with Linked list.
- LinkedList has more memory overhead as each node holds both data and address of next and previous node but in ArrayList each index only holds actual object or we can say data.
- ArrayList internally uses a dynamic array while LinkedList internally uses a doubly linked list.
- ArrayList get method is faster than Linked list so fetching data or search operations is good in arraylist.

Can we extend multiple classes in Java? Or does Java support multiple inheritance?

Multiple Inheritance is a feature of object-oriented concept, where a class can inherit properties of more than one parent class.

NO, Java doesn't support multiple inheritance.

Why? Simple, **to prevent ambiguity**. Consider a case where class B extends class A and Class C and both class A and C have the same method `display()`. Now java compiler cannot decide, which display method it should inherit. To prevent such situation, multiple inheritances is not allowed in java. Example, the diamond problem – Class B & C extend A and then Class D extends both B and C.

How can we achieve multiple inheritance in Java?

Java supports multiple inheritance of 'type', which is the ability of a class to **implement more than one interface**. An object can have multiple types: the type of its own class and the types of all the interfaces that the class implements. This means that if a variable is declared to be the type of an interface, then its value can reference any object that is instantiated from any class that implements the interface.

Update: Java 8 supports default methods where interfaces can provide default implementation of methods. And a class can implement two or more interfaces. In case both the implemented interfaces contain default methods with same method

signature, the implementing class should explicitly specify which default method is to be used or it should override the default method.

How to implement a Singleton design pattern?

Used to restrict the instantiation of a class and ensures that only one instance of the class exists in the JVM. In other words, a singleton class can have only one object at a time per JVM instance.

Say you want to make a rest API connection and then use the same session details for further requests, i.e., create an object of rest connection and then use the same object for subsequent requests.

How?

1. We want to restrict object creation using constructor > **Make constructor private.**

```
private SingletonClass() {}
```

2. Make a **private static instance** (class-member) of this singleton class. Static because there should only be one copy.

```
private static SingletonClass SINGLE_INSTANCE = null;
```

3. Write a **static/factory method** that checks the static instance member for null and creates the instance. At last, it returns an object of the singleton class.

```
public static SingletonClass getInstance()
{
    If (SINGLE_INSTANCE == null)
    {
        Synchronized (SingletonClass.class)
        {
            If (SINGLE_INSTANCE == null)
            {
                SINGLE_INSTANCE = new SingletonClass();
            }
        }
    }
}
```

Why double check for null? Because there might be two threads running which gets inside the first if statement concurrently when the instance is null.

Java: Can we have a return statement in the catch or, finally blocks?

Yes! We can write a return statement in both catch and finally block.

How does it work? 'finally' block is always executed (the only exception is System.exit()) hence any return statement in try/catch is overridden by the one in finally block.

Note: a rule of thumb – never return from finally. Eclipse, for example, shows a warning for that snippet: *"finally block does not complete normally"*.

Any idea about HashMap Collision concept?

Collision, i.e., crash between two things – hash code in this case!

Background: HashMap stores key-value pairs in what is known as buckets. When a value is added, the hashCode() and hash() methods are called to compute the hash value which ultimately boils down to an index in the internal array or what we call a bucket location.

Problem: A hash code collision is a situation where two or more keys produce the same final hash value and hence point to the same bucket location or array index. Why? Coz two unequal objects in Java can have the same hash code.

Solution: Java implements a hash code collision resolution technique. If the hash codes of any two keys collide, their entries will still be stored in the same bucket. After finding the bucket location with the final hash value, each of the keys at this location will be compared with the provided key object using the equals API.

Java: What's the difference between extend and implement keywords?

Extends: to indicate that a class is derived from the base class using inheritance, i.e., extend the functionality of the parent class to the subclass. In Java, multiple inheritances are not allowed due to ambiguity. Therefore, a class can extend only one class.

Implements: used to implement an interface. To access the interface methods, the interface must be "implemented" by another class. A class can implement any number of interfaces at a time.

I.e., *Extends* is used when inheriting a class, and *Implements* is used when a class is implementing an interface.

Note: An interface can 'extend' any number of other interfaces.

Java Programs

Commonly asked

Java Program to Find Odd or Even number

Logic: Iterate through the array, check if the no is divided by 2 or not. How? By getting the remainder when divided by 2.

```
public static void main(String args[])
{
    int a[] = {1,2,5,6,3,2};
    for(int i=0;i<a.length;i++)
    {
        If ( a[i] % 2 != 0)
            System.out.println(a[i]+" : Odd");
        else
            System.out.println(a[i]+" : Even");
    }
}
```

Java Program to Swap two numbers without using third variable

Logic: The arithmetic operators for addition and subtraction can be used to perform the swap without using a third variable.

```
public static void main(String args[])
{
    int x = 10, y = 50;
    x = x + y;
    y = x - y;
    x = x - y;
    System.out.println("After swap x=" + x + "and y=" + y);
}
```

Write a java program to find Prime number

Logic: Looping from 2 to number/2, since a number is not divisible by more than its half. Inside the for loop, check if the number is divisible by any number in the given range (2...num/2).

- If divisible, flag is set to true and we break out of the loop. I.e., not a prime number.
- If it isn't divisible by any number till the loop ends, flag is false and it's a prime number.

```
public static void main(String[] args)
{
    int num = 29;
    boolean flag = false;
    for (int i = 2; i <= num / 2; ++i)
    {
        if (num % i == 0)
        {
            flag = true;
            break;
        }
    }
    if (!flag)
        System.out.println(num + " is a prime number.");
    else
        System.out.println(num + " is not a prime number.");
}
```

Java Program to Find Factorial of a Number

Logic: Factorial of n is the product of all positive descending integers, denoted by n!.

$$4! = 4 * 3 * 2 * 1 = 24$$

Using loop,

```
public static void main(String args[])
{
    int i, fact=1;
    int number=5;
    for(i=1;i<=number;i++)
    {
        fact=fact*i;
    }
    System.out.println("Factorial of "+number+" is: "+fact);
}
```

Using recursion,

```
static int factorial(int n)
{
    if (n == 0)
        return 1;
    else
        return(n * factorial(n-1));
}

public static void main(String args[])
{
    int i,fact=1;
    int number=4;
    fact = factorial(number);
    System.out.println("Factorial of "+number+" is: "+fact);
}
```

Java Program to Reverse Number

Logic:

- Take the number's modulo by 10
- Multiply the reverse number by 10 and add modulo value into the reverse number.
- Divide the number by 10.
- Repeat above steps until number becomes zero.

```
// Function to reverse the number
static int reverse(int n)
{
    int rev = 0; // reversed number
    int rem; // remainder
    while(n>0)
    {
        rem = n%10;
        rev = (rev*10) + rem;
        n = n/10;
    }
    return rev;
}
```

- $rev = 0 \mid n = 1234$
- $rev = (0 * 10) + 4 = 4 \mid n = 123$
- $rev = (4 * 10) + 3 = 43 \mid n = 12$
- $rev = (43 * 10) + 2 = 432 \mid n = 1$
- $rev = (432 * 10) + 1 = 4321 \mid n = 0$

Java Program to print Fibonacci Series

The Fibonacci series is a series of elements where, the previous two elements are added to get the next element, starting with 0 and 1.

Input: N = 10

Output: 0 1 1 2 3 5 8 13 21 34

Using Recursion,

// Function to return the sum of previous 2 numbers

```
static int fib(int n)
```

```
{
```

```
    if (n <= 1)
```

```
        return n;
```

```
    else
```

```
        return fib(n - 1) + fib(n - 2);
```

```
}
```

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

```
{
```

```
    int N = 10;
```

```
    for (int i = 0; i < N; i++)
```

```
    {
```

```
        System.out.print(fib(i) + " ");
```

```
    }
```

```
}
```

Logic: Recursively iterate from value N to 1:

- Base case: If the value called recursively is less than 1, the return 1 the function.
- Recursive call: If the base case is not met, then recursively call for previous two value as: $fib(N - 1) + fib(N - 2)$;
- Return statement: At each recursive call (except the base case), return the recursive function for the previous two value as: $fib(N - 1) + fib(N - 2)$;

Write a java program to find Palindrome number

Logic: First reverse digits of number, then compare the reverse of number with actual number. If both are same, then return true, else false.

```
public static void main(String args[])
{
    int r, sum=0, temp;
    int n=454;
    temp=n;
    while(n>0)
    {
        r=n%10;
        sum=(sum*10)+r;
        n=n/10;
    }
    if(temp==sum)
        System.out.println("palindrome number ");
    else
        System.out.println("not palindrome");
}
```

Write a java program to find number of digits in given number.

Logic: Create an integer (count) > initialize it with 0. Divide the number with 10, till the number is 0 and for each turn > increment the count.

```
public static void main(String args[])
{
    Scanner sc = new Scanner(System.in);
    int count = 0;
    System.out.println("Enter a number:");
    int num = sc.nextInt();
    while(num!=0)
    {
        num = num/10;
        count++;
    }
    System.out.println("Number of digits: "+count);
}
```

Write a java program to find the duplicate words and their number of occurrences in a string.

Logic:

- Convert string to lowercase & split using space delimiter.
- Create empty HashMap of type String & Integer
- Iterate through String array
- Check whether particular word is already present in the HashMap using containsKey method.
- If it contains, then increase the count value by 1 using put(K, V) method.
- Otherwise insert using put() method of Map with count value as 1.
- Finally, print Map using keySet() or entrySet() method.

```
public static void main(String[] args)
{
    String str = "This is a program to find duplicate words in a string, again! a
program";
    String[] words = str.toLowerCase().trim().split(" ");
    Map<String, Integer> duplicateString = new HashMap<>();
    int count = 1;
    for (String x : words)
    {
        if (duplicateString.containsKey(x))
        {
            duplicateString.put(x, duplicateString.get(x) + 1);
        }
        else
        {
            duplicateString.put(x, count);
        }
    }
    System.out.println("Duplicate Words in a String : ");
    for (Map.Entry a : duplicateString.entrySet())
    {
        int val = (Integer) a.getValue();
        if (val > 1)
        {
            System.out.println(a);
        }
    }
}
```

Write a java program to count the number of words in a string

Using String.split() method

Logic: Solution uses the regular expression "\\s+" to split the string on whitespace. The split method returns an array, the length of the array is your number of words in a given String.

```
public static int countWordsUsingSplit(String input)
{
    if (input == null || input.isEmpty())
    {
        return 0;
    }
    String[] words = input.split("\\s+");
    return words.length;
}
```

Note: \\s+ will find one more space and split the String accordingly.

Write a java program to count the total number of occurrences of a given character in a string. Or to find duplicate characters in a string.

Using charAt() method and loop,

- Iterate throughout the length of the input String
- Check whether each character matches the character to search
 - If yes, increment the count
 - else do nothing

```
public static void main(String args[])
{
    String input = "aaaabbccAAdd";
    char search = 'a';
    int count=0;
    for(int i=0; i<input.length(); i++)
    {
        if(input.charAt(i) == search)
            count++;
    }
    System.out.println("The Character '"+search+"' appears "+count+" times.");
}
```

Using Hashmap,

For each character in the input string, we will put it in our Map with value 1, if it is seen for the first time. If the character repeats or the character is already present in our Map, we update the value of the character in the map by adding 1 to it.

```
public static void main(String args[])
{
    String input = "aaaabbAAAAcccd";
    char search = 'a';
    Map<Character,Integer> hash = new HashMap<Character,Integer>();
    for(int i=0;i<input.length();i++)
    {
        if(hash.containsKey(input.charAt(i)))
            hash.put(input.charAt(i), hash.get(input.charAt(i))+1);
        else
            hash.put(input.charAt(i), 1);
    }
    int result = hash.get(search);
    System.out.println("The Character '"+search+"' appears "+result+" times.");
}
```

Write a java program to reverse a string.

Using StringBuilder/StringBuffer,

```
public static String reverseString(String str)
{
    StringBuilder sb=new StringBuilder(str);
    sb.reverse();
    return sb.toString();
}
```

Using Reverse iteration,

```
public static String reverseString(String str)
{
    char ch[]=str.toCharArray();
    String rev="";
    for(int i=ch.length-1;i>=0;i--)
    {
        rev+=ch[i];
    }
    return rev;
}
```

Write a java program to reverse each word of a given string.

Logic: By using reverse() method of StringBuilder class, we can reverse given string. By the help of split("\\s") method, we can get all words in an array. To get the first character, we can use substring() or charAt() method.

Input: my name is khan

Output: ym eman si nahk

```
public static String reverseWord(String str)
{
    String words[] = str.split("\\s");
    String reverseWord="";
    for(String w:words)
    {
        StringBuilder sb=new StringBuilder(w);
        sb.reverse();
        reverseWord+=sb.toString()+" ";
    }
    return reverseWord.trim();
}
```

Write a java program to reverse a given string with preserving the position of spaces

Input: "abc de"

Output: "edc ba"

Logic:

- Create a string to store results. Mark the space position of the given string in this string.
- Insert the character from the input string into the result string in reverse order.
- While inserting the character check if the result string already contains a space at index 'j' or not. If it contains, we copy the character to the next position.

string reverses(string str)

```
{
    int n = str.size();
    string result(n, '\0');
    for (int i = 0; i < n; i++)
        if (str[i] == ' ')
            result[i] = ' ';

    int j = n - 1;
    for (int i = 0; i < str.length(); i++)
    {
        if (str[i] != ' ')
        {
            if (result[j] == ' ')
                j--;
            result[j] = str[i];
            j--;
        }
    }
    return result;
}
```

Write a Java Program to find the second-highest number in an array.

Logic-1: sort the array in descending order and then return the second element which is not equal to the largest element from the sorted array.

```
static void print2largest(int arr[], int arr_size)
{
    int i, first, second;
    if (arr_size < 2)
    {
        System.out.printf(" Invalid Input ");
        return;
    }
}
```

```

    }
    Arrays.sort(arr);
    for (i = arr_size - 2; i >= 0; i--)
    {
        if (arr[i] != arr[arr_size - 1])
        {
            System.out.printf("The second largest " + "element is %d\n",
                arr[i]);
            return;
        }
    }
    System.out.printf("There is no second " + "largest element\n");
}

```

Logic-2:

- Initialize highest and secondHighest with minimum possible value.
- Iterate over array.
- If current element is greater than highest
 - Assign secondHighest = highest
 - Assign highest = currentElement
- Else if current element is greater than secondHighest
- Assign secondHighest =current element.

```

public static int findSecondLargestNumberInTheArray(int array[])
{
    int highest = Integer.MIN_VALUE;
    int secondHighest = Integer.MIN_VALUE;
    for (int i = 0; i < array.length; i++)
    {
        if (array[i] > highest)
        {
            secondHighest = highest;
            highest = array[i];
        }
        else if (array[i] > secondHighest && array[i] != highest)
            secondHighest = array[i];
    }
    return secondHighest;
}

```

Write a Java Program to remove all white spaces from a string.

Logic: We can either use “replaceAll” method to remove all spaces OR iterate over the char array to add only the non-space values to new String.

```
public static void main(String[] args)
{
    String str = "India Is My Country";

    //1st way
    String noSpaceStr = str.replaceAll("\\s", ""); // using built in method
    System.out.println(noSpaceStr);

    //2nd way
    char[] strArray = str.toCharArray();
    StringBuffer stringBuffer = new StringBuffer();
    for (int i = 0; i < strArray.length; i++)
    {
        if ((strArray[i] != ' ') && (strArray[i] != '\t'))
        {
            stringBuffer.append(strArray[i]);
        }
    }
    String noSpaceStr2 = stringBuffer.toString();
    System.out.println(noSpaceStr2);
}
```

Write a Java program to find the longest substring from a given string which doesn't contain any duplicate characters?

Logic: We start traversing the string from left to right and maintain track of:

- the current substring with non-repeating characters with the help of a start and end index
- the longest non-repeating substring output
- a lookup table of already visited characters
- For every new character, we look for it in the already visited characters. If the character has already been visited and is part of the current substring with non-repeating characters, we update the start index. Otherwise, we'll continue traversing the string.

```
String getUniqueCharacterSubstring(String input)
{
    Map<Character, Integer> visited = new HashMap<>();
    String output = "";
    for (int start = 0, end = 0; end < input.length(); end++)
    {
        char currChar = input.charAt(end);
        if (visited.containsKey(currChar))
        {
            start = Math.max(visited.get(currChar)+1, start);
        }
        if (output.length() < end - start + 1)
        {
            output = input.substring(start, end + 1);
        }
        visited.put(currChar, end);
    }
    return output;
}
```

Java program to calculate SUM of numbers in an array.

Logic: Create an empty variable (sum) > Initialize it with 0 > Traverse through each element of the array and add each element to sum > Print sum.

```
public static void main(String[] args)
{
    int [] arr = new int [] {1, 2, 3, 4, 5};
    int sum = 0;
    for (int i = 0; i < arr.length; i++)
    {
        sum = sum + arr[i];
    }
    System.out.println("Sum of all the elements of an array: " + sum);
}
```

Java program to find the sum of digits of a number.

Input: 95318

Output = 9+5+3+1+8 = 26

Logic:

- Declare a variable (sum) to store the sum of numbers and initialize it to 0.
- Find the remainder by using the modulo (%) operator. It gives the last digit of the number (N).
- Add the last digit to the variable sum.
- Divide the number (N) by 10. It removes the last digit of the number.
- Repeat the above steps (2 to 4) until the number (N) becomes 0.

```
public static void main(String args[])
{
    int number, digit, sum = 0;
    Scanner sc = new Scanner(System.in);
    System.out.print("Enter the number: ");
    number = sc.nextInt();
    while(number > 0)
    {
        digit = number % 10;
        sum = sum + digit;
        number = number / 10;
    }
    System.out.println("Sum of Digits: "+sum);
}
```

Java program to multiply 2 numbers without using * operator.

Suppose, we want to multiply 3 by 4 which gives 12 as the result. The same can be achieved by adding 3 four times i.e. $(3 + 3 + 3 + 3 = 12)$ or by adding 4 three times i.e. $(4 + 4 + 4 = 12)$. Both give the same result. Therefore, we can implement the logic using recursion.

```
public static void main(String args[])
{
    int x=4, y=4, sum=0;
    for(int i=1;i<=x;i++)
    {
        sum=sum+y;
    }
    System.out.println("The multiplication of "+x+" and "+y+" is: "+sum);
}
```

Find substring in a string without using contains.

`String.indexOf()`: The `.indexOf()` method is a bit more crude than the `.contains()` method, but it's nevertheless the underlying mechanism that enables the `.contains()` method to work. It returns the index of the first occurrence of a substring within a `String`, and offers a few constructors to choose from:

- `indexOf(int ch)`
- `indexOf(int ch, int fromIndex)`
- `indexOf(String str)`
- `indexOf(String str, int fromIndex)`

Find duplicates in an array.

Logic: Duplicate elements can be found using two loops. The outer loop will iterate through the array from 0 to length of the array. The outer loop will select an element. The inner loop will be used to compare the selected element with the rest of the elements of the array. If a match is found which means the duplicate element is found then, display the element.

```
public static void main(String[] args)
{
    int [] arr = new int [] {1, 2, 3, 4, 2, 7, 8, 8, 3};
    System.out.println("Duplicate elements in given array: ");
    for(int i = 0; i < arr.length; i++)
    {
        for(int j = i + 1; j < arr.length; j++)
        {
            if(arr[i] == arr[j])
                System.out.println(arr[j]);
        }
    }
}
```

Checking anagram strings.

An anagram of a string is another string that contains the same characters, only the order of characters can be different. LISTEN – SILENT are anagrams.

Logic: Sort both strings > Compare the sorted strings.

```

static boolean areAnagram(char[] str1, char[] str2)
{
    int n1 = str1.length;
    int n2 = str2.length;
    if (n1 != n2)
        return false;
    Arrays.sort(str1);
    Arrays.sort(str2);
    for (int i = 0; i < n1; i++)
        if (str1[i] != str2[i])
            return false;
    return true;
}

```

Find first non-repeated character in a given string.

Input: TesterTested

Output: 'R'

Logic:

- Make a hash_map which will map the character to their respective frequencies.
- Traverse the given string using a pointer.
- Increase the count of current character in the hash_map.
- Now traverse the string again and check whether the current character has frequency=1.
- If the frequency>1 continue the traversal.
- Else break the loop and print the current character as the answer.

```

class STS
{
    static final int NO_OF_CHARS = 256;
    static char count[] = new char[NO_OF_CHARS];
    static void getCharCountArray(String str)
    {
        for (int i = 0; i < str.length(); i++)
            count[str.charAt(i)]++;
    }

    static int firstNonRepeating(String str)
    {
        getCharCountArray(str);
        int index = -1, i;
    }
}

```

```

        for (i = 0; i < str.length(); i++)
        {
            if (count[str.charAt(i)] == 1)
            {
                index = i;
                break;
            }
        }
        return index;
    }

    public static void main(String[] args)
    {
        String str = "TesterTested";
        int index = firstNonRepeating(str);
        System.out.println(index == -1 ? "Either all characters are repeating or string" + "is empty" : "First non-repeating character is " + str.charAt(index));
    }
}

```

How to find largest and smallest element in Array?

Logic: Traverse the array iteratively and keep track of the smallest and largest element until the end of the array.

```

public static void main(String args[])
{
    int large, small, i;
    int a[] = new int[]{1, 2, 3, 4, 5};
    int n = a.length;
    large=small=a[0];
    for(i=1; i<n; ++i)
    {
        if(a[i]>large)
            large=a[i];

        if(a[i]<small)
            small=a[i];
    }
    System.out.print("\nThe smallest element is " + small );
    System.out.print("\nThe largest element is " + large );
}

```

Find two numbers of which the product is maximum in an array.

Logic-1: A Simple Solution is to consider every pair and keep track of the maximum product.

```
static void maxProduct(int arr[], int n)
{
    if (n < 2)
    {
        System.out.println("No pairs exists");
        return;
    }

    int a = arr[0], b = arr[1];
    for (int i = 0; i < n; i++)
        for (int j = i + 1; j < n; j++)
            if (arr[i] * arr[j] > a * b)
            {
                a = arr[i];
                b = arr[j];
            }

    System.out.println("Max product pair is {" + a + ", " + b + "}");
}
```

Logic-2: Sort input array in increasing order,

- If all elements are positive, then return the product of the last two numbers.
- Else return a maximum of products of the first two and last two numbers.

```
static void maxProduct(int arr[], int n) {
    Arrays.sort(arr);
    int num1, num2;
    int sum1 = arr[0] * arr[1];
    int sum2 = arr[n - 1] * arr[n - 2];
    if (sum1 > sum2) {
        num1 = arr[0];
        num2 = arr[1];
    }
    else {
        num1 = arr[n - 2];
        num2 = arr[n - 1];
    }
}
```

```
        System.out.println("Max product pair = " + "{" + num1 + "," + num2 + "}");  
    }
```

Java program to read from file line by line.

Using the Java BufferedReader class is the most common and simple way to read a file line by line in Java. It belongs to java.io package. Java BufferedReader class provides readLine() method to read a file line by line.

```
public static void main(String args[]) {  
    try {  
        File file=new File("Demo.txt");  
        FileReader fr=new FileReader(file);  
        BufferedReader br=new BufferedReader(fr);  
        StringBuffer sb=new StringBuffer();  
        while((line=br.readLine())!=null)  
        {  
            sb.append(line);  
            sb.append("\n");  
        }  
        fr.close();  
        System.out.println("Contents of File: ");  
        System.out.println(sb.toString());  
    }  
    catch(IOException e) {  
        e.printStackTrace();  
    }  
}
```