JAVA FEATURES

1. **What is JAVA**?
Java is a high-level programming language and is platform-independent.
Java is a collection of objects. It was developed by Sun Microsystems.
2. What are the features of JAVA?
a) **OOP concepts:**
- Object-oriented
- Inheritance
- Encapsulation
- Polymorphism
- Abstraction

b) **Platform independent**: A single program works on different platforms without any modification.
c) **High Performance:** JIT *(Just In Time compiler)* enables high performance in Java. JIT converts the bytecode into machine language and then JVM starts the execution.
d) **Multi-threaded:** A flow of execution is known as a Thread. JVM creates a thread which is called the main thread. The user can create multiple threads by extending the thread class or by implementing the Runnable interface

Q1. Explain JDK, JRE and JVM?

JDK	JRE	JVM
It stands for Java Development Kit.	It stands for Java Runtime Environment.	It stands for Java Virtual Machine.
It is the tool necessary to compile, document and package Java programs.	JRE refers to a runtime environment in which Java bytecode can be executed.	It is an abstract machine. It is a specification that provides a run-time environment in which Java bytecode can be executed.
It contains JRE + development tools.	It's an implementation of the JVM which physically exists.	JVM follows three notations: Specification, **Implementation,** and **Runtime Instance**.

Disadvantages of Java Language
1- Performance
Java programs take much longer time to run compared to C/C++.
2- Memory
Since Java Programs run on top of Java Virtual Machine, it consumes more memory.
3- Cost
Since memory and processing requirements higher, hardware cost increases.
4- Low level programming
There is no support for low level programming in Java, like pointers are missing.
5- Garbage collection

There is no control over garbage collection in Java. That is programmer does not have any right to control the garbage collection. Java does not provide functions like delete(), free().

6- No Unsigned Types Unlike C/C++, Java does not support unsigned int, unsigned char, etc. However, in Java 8, API for unsigned long and unsigned int is introduced

3. **What is meant by the Local variable and the Instance variable?**

Local variables are defined in the method and scope of the variables that exist inside the method itself.
Instance variable is defined inside the class and outside the method and the scope of the variables exists throughout the class.

4. **Difference between Array and Array List.**

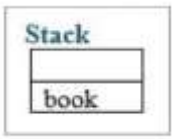
Array	Array List
Size should be given at the time of array declaration. String[] name = new String[2]	Size may not be required. It changes the size dynamically. ArrayList name = new ArrayList
To put an object into array we need to specify the index. name[1] = "book"	No index required. name.add("book")
Array is not type parameterized	ArrayList in java 5.0 are parameterized. Eg: This angle bracket is a type parameter which means a list of String.

5. **Difference between String, String Builder, and String Buffer.**

String variables are stored in a "constant string pool". Once the string reference changes the old value that exists in the "constant string pool", it cannot be erased.

String Buffer:
Here string values are stored in a stack. If the values are changed then the new value replaces the older value.
The string buffer is synchronized which is thread-safe.
Performance is slower than the String Builder.

```
String Buffer name ="book";
```



String Builder:
This is the same as String Buffer except for the String Builder which is not threaded safely that is not synchronized. So obviously the performance is fast.

6. What is Java thread-safety?
thread-safety or thread-safe code in Java refers to code which can safely be used or shared in concurrent or multi-threading environment and they will behave as expected. any code, class, or object which can behave differently from its contract on the concurrent environment is not thread-safe.

7. **Difference between HashMap and HashTable.**

HashMap	HashTable
Methods are not synchronized	Key methods are synchronized
Not thread safety	Thread safety
Iterator is used to iterate the values	Enumerator is used to iterate the values
Allows one null key and multiple null values	Doesn't allow anything that is null
Performance is high than HashTable	Performance is slow

8. Difference between HashSet and TreeSet.

HashSet	TreeSet
Inserted elements are in random order	Maintains the elements in the sorted order
Can able to store null objects	Couldn't store null objects
Performance is fast	Performance is slow

9. **Abstraction class and Methods in Java:**

Data abstraction is the process of hiding certain details and showing only essential information to the user.
Abstraction can be achieved with either abstract classes or interfaces (which you will learn more about in the next chapter).

The abstract keyword is a non-access modifier, used for classes and methods:

Abstract class: is a restricted class that cannot be used to create objects (to access it, it must be inherited from another class).
Abstract method: can only be used in an abstract class, and it does not have a body. The body is provided by the subclass (inherited from).

```
// Abstract class
abstract class Animal {
  // Abstract method (does not have a body)
  public abstract void animalSound();
  // Regular method
  public void sleep() {
    System.out.println("Zzz");
  }
}

// Subclass (inherit from Animal)
class Pig extends Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig(); // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

When should a class be abstract?
In general, a class should be abstract when you have absolutely no reason to create an instance of that class. For example, suppose you have a Shape class that is the superclass of Triangle, Square, Circle, etc

When to Use an Abstract Class
If we have specified requirements and only partial implementation details. While classes that extend abstract classes have several common fields or methods (that require non-public modifiers) If one wants to have non-final or non-static methods to modify the states of an object.

10. **Interfaces in JAVA:**
Interfaces is Another way to achieve abstraction in Java, is with interfaces.
An interface is a completely "abstract class" that is used to group related methods with empty bodies.

```
// interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void run(); // interface method (does not have a body)
}
```

Implementation:

```
// Interface
interface Animal {
  public void animalSound(); // interface method (does not have a body)
  public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
  public void animalSound() {
    // The body of animalSound() is provided here
    System.out.println("The pig says: wee wee");
  }
  public void sleep() {
    // The body of sleep() is provided here
    System.out.println("Zzz");
  }
}

class Main {
  public static void main(String[] args) {
    Pig myPig = new Pig();  // Create a Pig object
    myPig.animalSound();
    myPig.sleep();
  }
}
```

Notes on Interfaces:

- Like abstract classes, interfaces cannot be used to create objects (in the example above, it is not possible to create an "Animal" object in the MyMainClass)
- Interface methods do not have a body - the body is provided by the "implement" class
- On implementation of an interface, you must override all of its methods
- Interface methods are by default abstract and public
- Interface attributes are by default public, static and final
- An interface cannot contain a constructor (as it cannot be used to create objects)

Why And When To Use Interfaces?
1) To achieve security - hide certain details and only show the important details of an object (interface).
2) Java does not support "multiple inheritance" (a class can only inherit from one superclass). However, it can be achieved with interfaces, because the class can implement multiple interfaces. Note: To implement multiple interfaces, separate them with a comma

11. **Difference between Abstract class and Interface.**

Abstract Class	Interface
An Abstract class doesn't provide full abstraction	Interface does provide full abstraction
Using Abstract we can not achieve multiple inheritance	using an Interface we can achieve multiple inheritance.
We can declare a member field	We can not declare a member field in an Interface
An abstract class can contain access modifiers for the subs, functions, properties	We can not use any access modifier i.e. public , private , protected , internal etc. because within an interface by default everything is public
An abstract class can be defined	An Interface member cannot be defined using the keyword static, virtual, abstract or sealed
A class may inherit only one abstract class.	A class may inherit several interfaces.
An abstract class can provide complete, default code and/or just the details that have to be overridden.	An interface cannot provide any code, just the signature.

12. **Explain public static void main(String args[]) in Java.**
main() in Java is the entry point for any Java program. It is always written as public static void main(String[] args).

- public: Public is an access modifier, which is used to specify who can access this method.
- Public means that this Method will be accessible by any Class.
- static: It is a keyword in java which identifies it is class-based. main() is made static in Java so that it can be accessed without creating the instance of a Class. In case, main is not made static then the compiler will throw an error as main() is called by the JVM before any objects are made and only static methods can be directly invoked via the class.
- void: It is the return type of the method. Void defines the method which will 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 execution occurs.
- String args[]: It is the parameter passed to the main method.

13. **Why Java is not 100% Object-oriented?**
Java is not 100% Object-oriented because it makes use of eight primitive data types such as boolean, byte, char, int, float, double, long, short which are not objects.

14. **What are wrapper classes in Java?**
Wrapper classes convert the Java primitives into the reference types (objects). Every primitive data type has a class dedicated to it. These are known as wrapper classes because they "wrap" the primitive data type into an object of that class. Refer to the below image which displays different primitive type, wrapper class and constructor argument.

15. **What is singleton class in Java and how can we make a class singleton?**
Singleton class is a class whose only one instance can be created at any given time, in one JVM. A class can be made singleton by making its constructor private.

16. **What is the difference between equals() and == in Java?**
Equals() method is defined in Object class in Java and used for checking equality of two objects defined by business logic.
"==" or equality operator in Java is a binary operator provided by Java programming language and used to compare primitives and objects. public boolean equals(Object o) is the method provided by the Object class. The default implementation uses == operator to compare two objects. For example: method can be overridden like String class. equals() method is used to compare the values of two objects

17. What is a package in Java? List down various advantages of packages.
Packages in Java, are the collection of related classes and interfaces which are bundled together. By using packages, developers can easily modularize the code and optimize its reuse. Also, the code within the packages can be imported by other classes and reused. Below I have listed down a few of its advantages:

- Packages help in avoiding name clashes
- They provide easier access control on the code
- Packages can also contain hidden classes which are not visible to the outer classes and only used within the package
- Creates a proper hierarchical structure which makes it easier to locate the related classes

18. **Why pointers are not used in Java?**
Java doesn't use pointers because they are unsafe and increases the complexity of the program. Since, Java is known for its simplicity of code, adding the concept of pointers will be contradicting. Moreover, since JVM is responsible for implicit memory allocation, thus in order to avoid direct access to memory by the user, pointers are discouraged in Java.

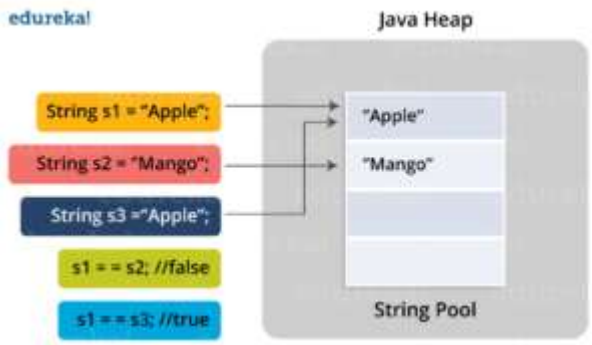
19. **What are access modifiers in Java?**
In Java, access modifiers are special keywords which are used to restrict the access of a class, constructor, data member and method in another class. Java supports four types of access modifiers:

- Default
- Private
- Protected
- Public

Modifier	Default	Private	Protected	Public
Same class	YES	YES	YES	YES
Same Package subclass	YES	NO	YES	YES
Same Package non-subclass	YES	NO	YES	YES
Different package subclass	NO	NO	YES	YES
Different package non-subclass	NO	NO	NO	YES

20. **What is Java String Pool?**

Java String pool refers to a collection of Strings which are stored in heap memory. In this, whenever a new object is created, String pool first checks whether the object is already present in the pool or not. If it is present, then the same reference is returned to the variable else new object will be created in the String pool and the respective reference will be returned.



Difference between String, StringBuilder, and StringBuffer.

Factor	String	StringBuilder	StringBuffer
Storage Area	Constant String Pool	Heap Area	Heap Area
Mutability	Immutable	Mutable	Mutable
Thread Safety	Yes	No	Yes
Performance	Fast	More efficient	Less efficient

21. **How do you use ‘this’ keyword?**

You can use ‘this’ to refer to a current object, invoke the current class method or class constructor. You can also pass it on as an argument into your methods or constructors

22. **What is annotation?**

Annotation is a tag you use to symbolize metadata that represents your class, interface, and fields among others.They are used by the compiler and the JVM and don’t directly influence the operations.

23. **What is aggregation?**

It is a type of weak relation you can create between two classes, where one contain references to another class contained within it

24. What is the purpose of composition?

You can use composition to hold the reference of one class within another class, and in this case, the contained object cannot exist without the class containing it. It is a type of aggregation.

25. **What are the different types of garbage collectors in Java?**

Garbage collection in Java a program which helps in implicit memory management. Since in Java, using the new keyword you can create objects dynamically, which once created will consume some memory. Once the job is done and there are no more references left to the object, Java using garbage collection destroys the object and relieves the memory occupied by it. Java provides four types of garbage collectors:

- Serial Garbage Collector
- Parallel Garbage Collector
- CMS Garbage Collector
- G1 Garbage Collector

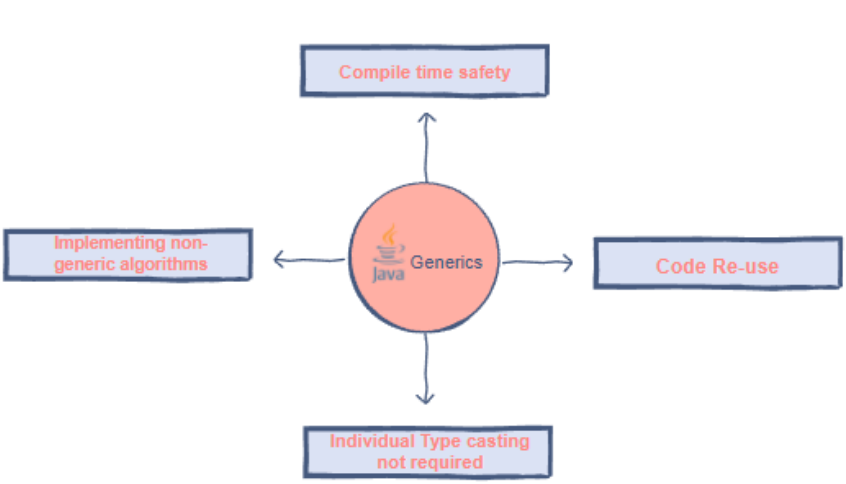
JAVA GENERICS

1. **What are Generics in Java?**

Java Generics is a set of related methods or a set of similar types. Generics allow types Integer, String, or even user-defined types to be passed as a parameter to classes, methods, or interfaces. Generics are mostly used by classes like HashSet or HashMap.

Advantages of using generics

- Generics ensure compile-time safety which allows the programmer to catch the invalid types while compiling the code.
- Java Generics helps the programmer to reuse the code for whatever type he/she wishes. For instance, a programmer writes a generic method for sorting an array of objects. Generics allow the programmer to use the same method for Integer arrays, Double arrays, and even String arrays.
- Another advantage of using generics is that Individual typecasting isn't required. The programmer defines the initial type and then lets the code do its job.
- It allows us to implement non-generic algorithms.

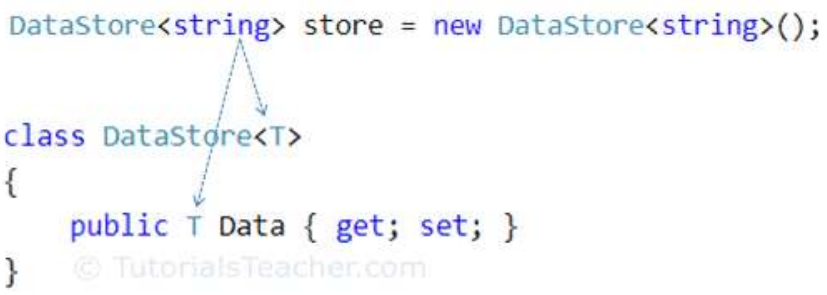


Types of Java Generics

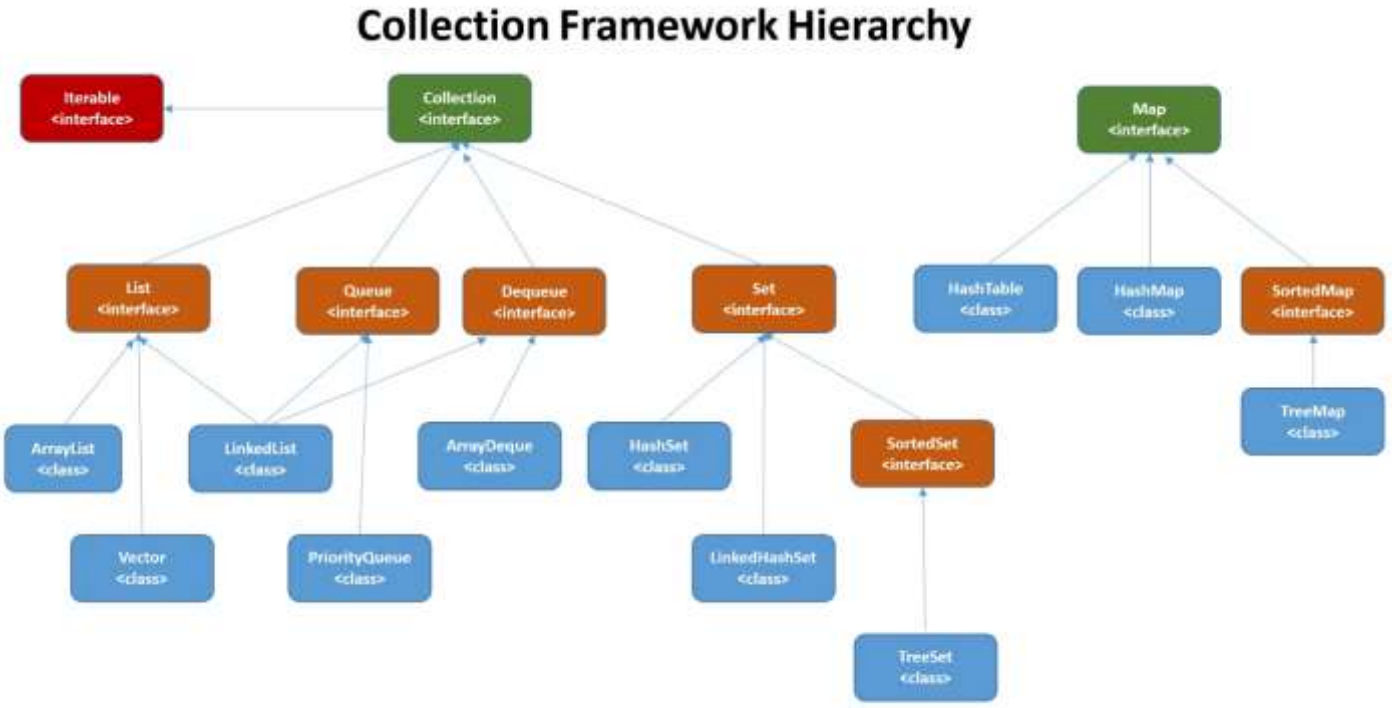
Generic method: Generic Java method takes a parameter and returns some value after performing a task. It is exactly like a normal function, however, a generic method has type parameters which are cited by actual type. This allows the generic method to be used in a more general way. The compiler takes care of the type of safety which enables programmers to code easily since they do not have to perform long, individual type castings.

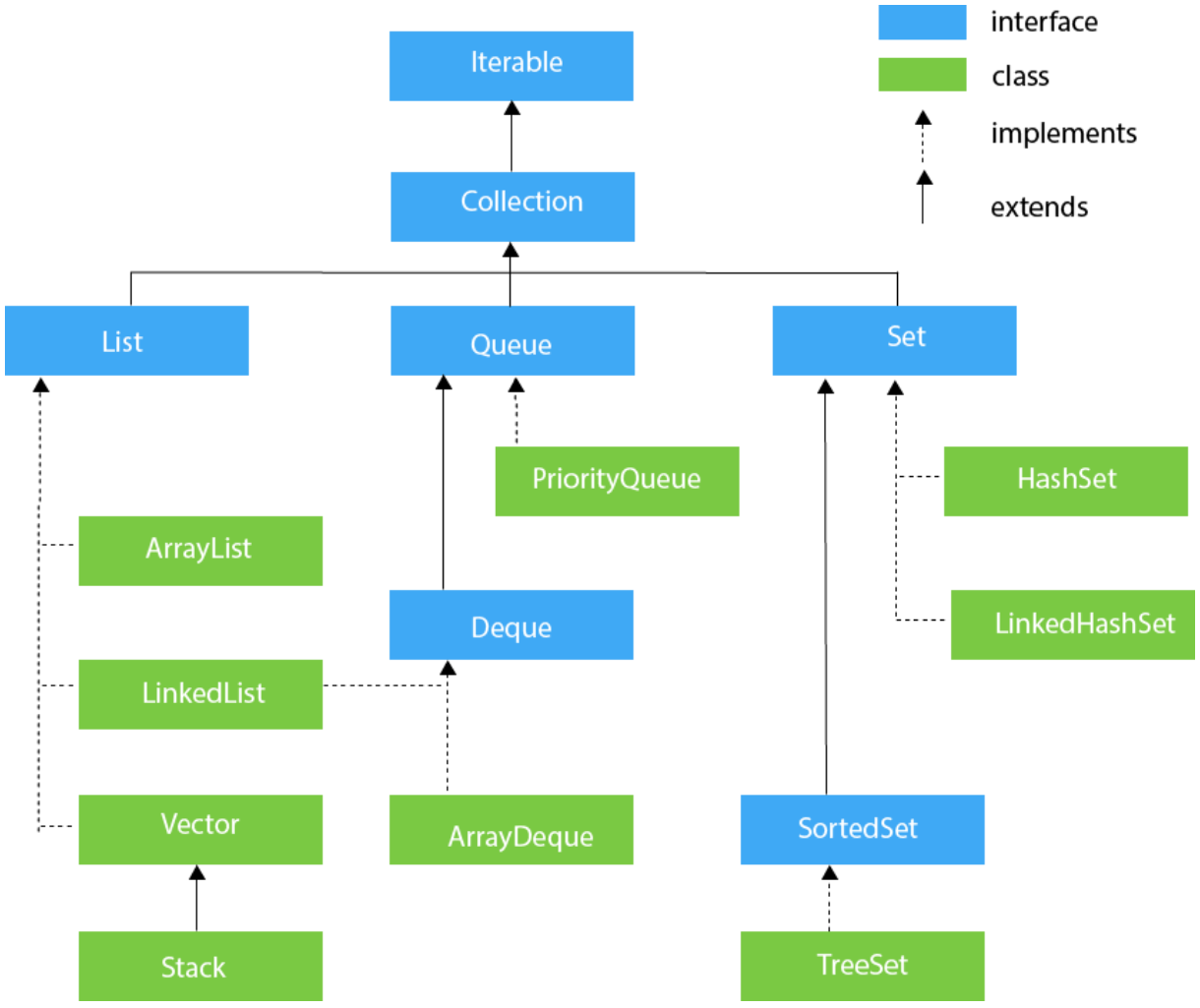
```java
public class GenericMethodTest {
   // generic method printArray
   public static < E > void printArray( E[] inputArray ) {
      // Display array elements
      for(E element : inputArray) {
         System.out.printf("%s ", element);
      }
      System.out.println();
   }

   public static void main(String args[]) {
      // Create arrays of Integer, Double and Character
      Integer[] intArray = { 1, 2, 3, 4, 5 };
      Double[] doubleArray = { 1.1, 2.2, 3.3, 4.4 };
      Character[] charArray = { 'H', 'E', 'L', 'L', 'O' };

      System.out.println("Array integerArray contains:");
      printArray(intArray);   // pass an Integer array

      System.out.println("\nArray doubleArray contains:");
      printArray(doubleArray);   // pass a Double array

      System.out.println("\nArray characterArray contains:");
      printArray(charArray);   // pass a Character array
   }
}
```

Generic classes: A generic class is implemented exactly like a non-generic class. The only difference is that it contains a type parameter section. There can be more than one type of parameter, separated by a comma. The classes, which accept one or more parameters, are known as parametrized classes or parameterized types.



JAVA COLLECTION FRAMEWORK





1. **What are Java Collections?**
The Java Collections Framework is a unified architecture for representing and manipulating collections. It contains interfaces, their implementation classes, and algorithms to process the data stored in a collection. The Collection interface is extended by other interfaces like List, Set, and Queue.

The Java Collections Framework components

Interfaces: These interfaces supply the abstract data type to represent the collection. The **java.util.Collection** is the root interface of the framework. It's at the top of the framework hierarchy and contains important methods, like size(), iterator(), add(), remove(), and clear().

The iterable interface is the root of the whole collection framework. It allows the iterator to iterate through all of the collections. All classes and interfaces use this interface. The collection interface extends the iterable interface and is executed by the classes in the collection framework. The list interface inhibits a list type data structure where we can store ordered collections of objects.

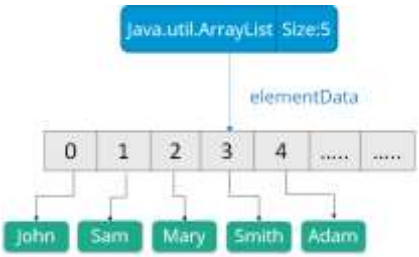
Java collections: List

A List is an ordered Collection of elements which may contain duplicates. It is an interface that extends the Collection interface. Lists are further classified into the following:

- ArrayList :
- LinkedList
- Vectors

Array list:

- Fast iteration and fast Random Access.
- It is an ordered collection (by index) and not sorted.
- It implements the Random Access Interface.

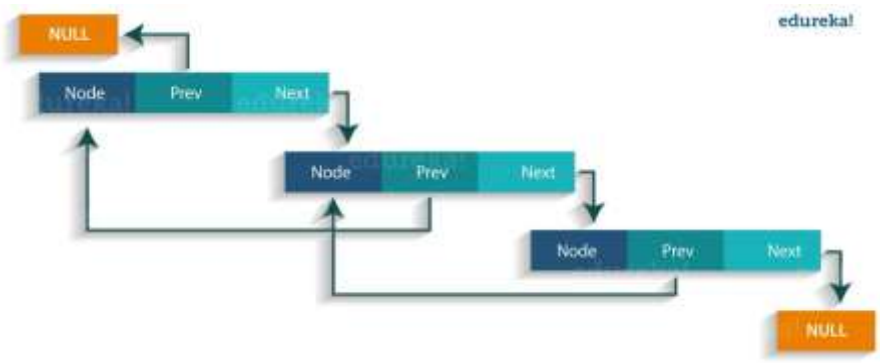


Linked List:

- Elements are doubly linked to one another.
- Performance is slower than the Array list.
- Good choice for insertion and deletion.

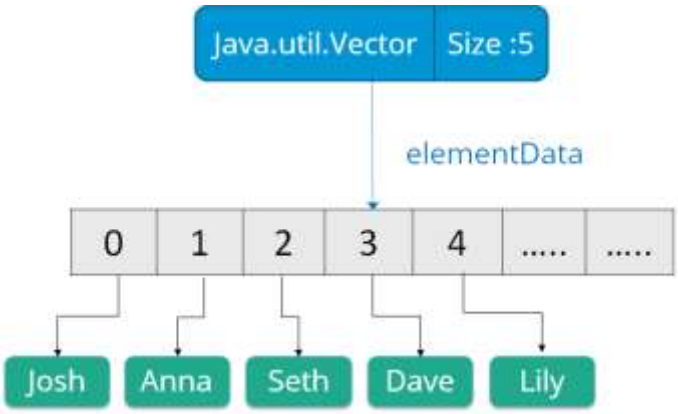
Singly Linked List : In a singly Linked list each node in this list stores the data of the node and a pointer or reference to the next node in the list. Refer to the below image to get a better understanding of single Linked list.

Doubly Linked List : In a doubly Linked list, it has two references, one to the next node and another to previous node.



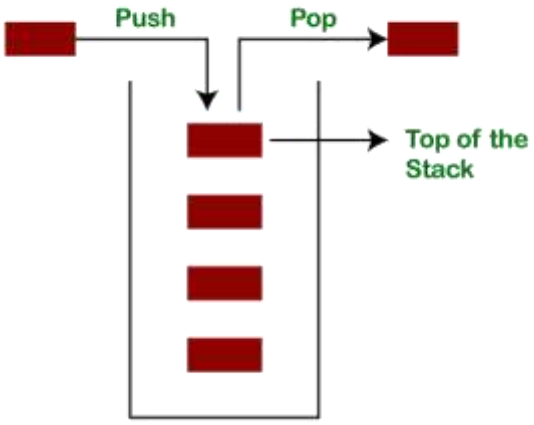
Vectors :

- Vector methods are synchronized.
- Thread safety.
- It also implements Random Access.
- Thread safety usually causes a performance hit.



Stack:

The stack is a linear data structure that is used to store the collection of objects. It is based on Last-In-First-Out (LIFO). Java collection framework provides many interfaces and classes to store the collection of objects. One of them is the Stack class that provides different operations such as push, pop, search, etc.



Java collections: Queue

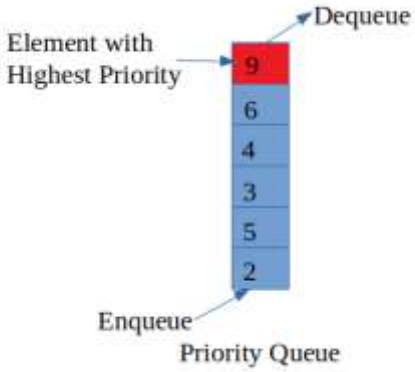
Queue in Java follows a FIFO approach i.e. it orders the elements in First In First Out manner. In a queue, the first element is removed first and last element is removed in the end. Each basic method exists in two forms: one throws an exception if the operation fails, the other returns a special value.

Queue interface can be instantiated as:

```
Queue<String> q1 = new PriorityQueue();
Queue<String> q2 = new ArrayDeque();
```

Priority Queue

The Priority Queue class implements the Queue interface. It holds the elements or objects which are to be processed by their priorities. Priority Queue doesn't allow null values to be stored in the queue.

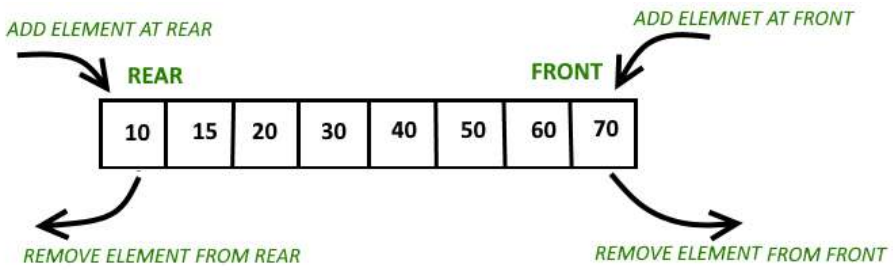


Deque Interface:

Deque interface extends the Queue interface. In Deque, we can remove and add the elements from both the side. Deque stands for a double-ended queue which enables us to perform the operations at both the ends.

Array Deque

Array Deque class implements the Deque interface. It facilitates us to use the Deque. Unlike queue, we can add or delete the elements from both the ends.
Array Deque is faster than ArrayList and Stack and has no capacity restrictions.



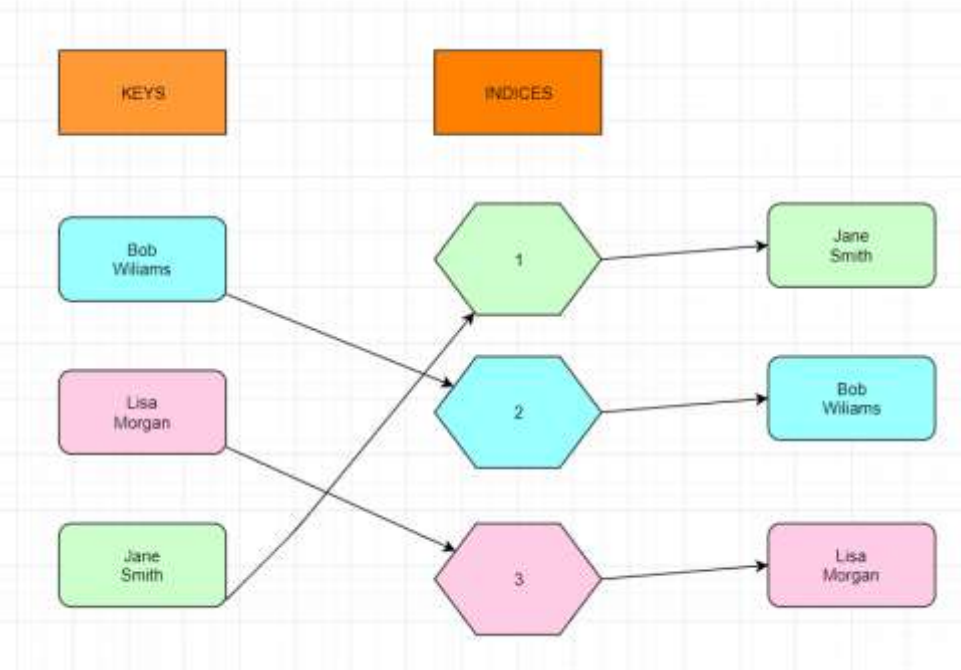
Java collections: SET

Set Interface in Java is present in java.util package. It extends the Collection interface. It represents the unordered set of elements which doesn't allow us to store the duplicate items. We can store at most one null value in Set. Set is implemented by HashSet, LinkedHashSet, and TreeSet.

```
Set<data-type> s1 = new HashSet<data-type>();
Set<data-type> s2 = new LinkedHashSet<data-type>();
Set<data-type> s3 = new TreeSet<data-type>();
```

HashSet:

- Unordered and unsorted.
- Uses the hash code of the object to insert the values.
- Use this when the requirement is “no duplicates and don’t care about the order”.

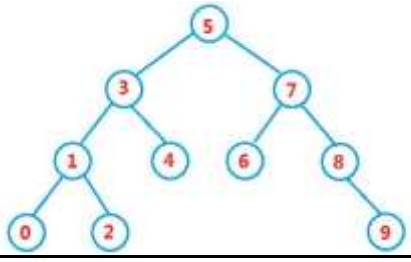


LinkedHashSet:

- An ordered version of the hash set is known as Linked Hash Set.
- Maintains a doubly-Linked list of all the elements.
- Use this when an iteration order is required.

TreeSet:

- It is one of the two sorted collections.
- Uses the “Read-Black” tree structure and guarantees that the elements will be in ascending order.
- We can construct a tree set with the constructor by using a comparable (or) comparator.



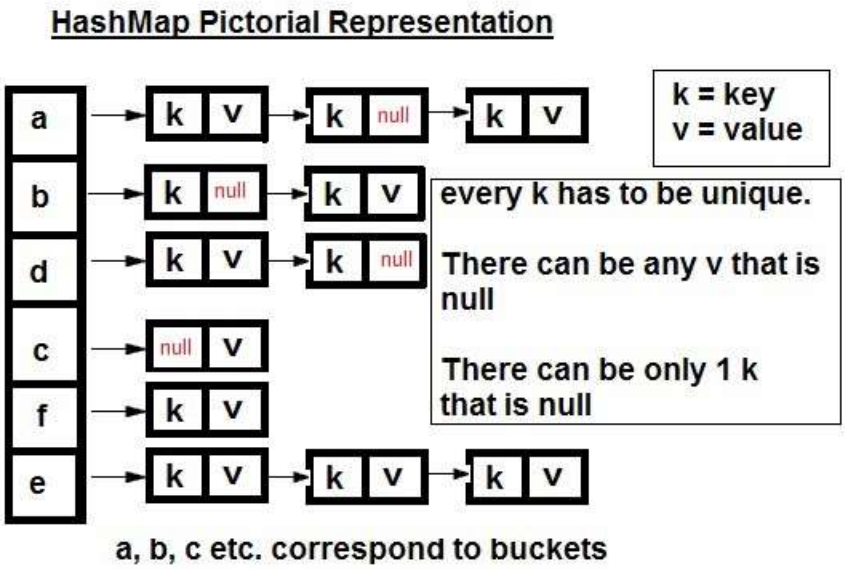
HashSet	LinkedHashSet	TreeSet
HashSet internally uses HashMap to store its elements.	LinkedHashSet internally uses LinkedHashMap to store its elements.	TreeSet internally uses TreeMap to store its elements.
HashSet doesn't maintain any order of elements.	LinkedHashSet maintain insertion order of elements.	TreeSet maintains default natural sorting order.
HashSet gives better performance than LinkedHashSet and TreeSet.	The performance of LinkedHashSet is between HashSet and TreeSet.	The TreeSet gives less performance than HashSet and LinkedHashSet.
HashSet allow maximum one null element.	LinkedHashSet also allow maximum one null element.	The TreeSet doesn't allow even single element.

Java collections: MAP

Map cares about the unique identifier. We can map a unique key to a specific value. It is a key/value pair. We can search a value, based on the key. Like the set, the map also uses the "equals ( )" method to determine whether two keys are the same or different.

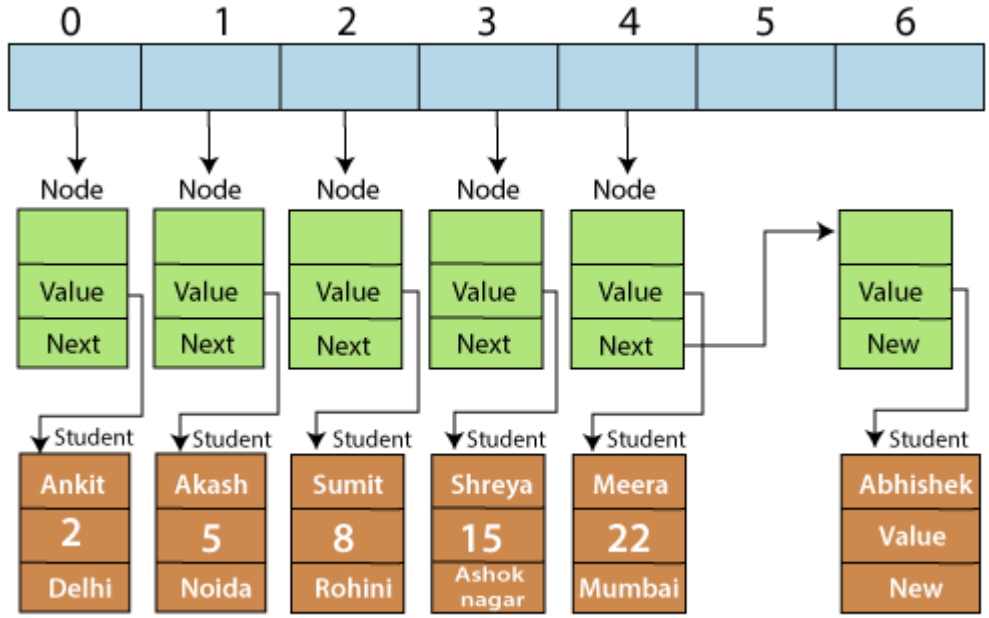
Hash Map:

- Unordered and unsorted map.
- Hashmap is a good choice when we don't care about the order.
- It allows one null key and multiple null values.



Hash Table:

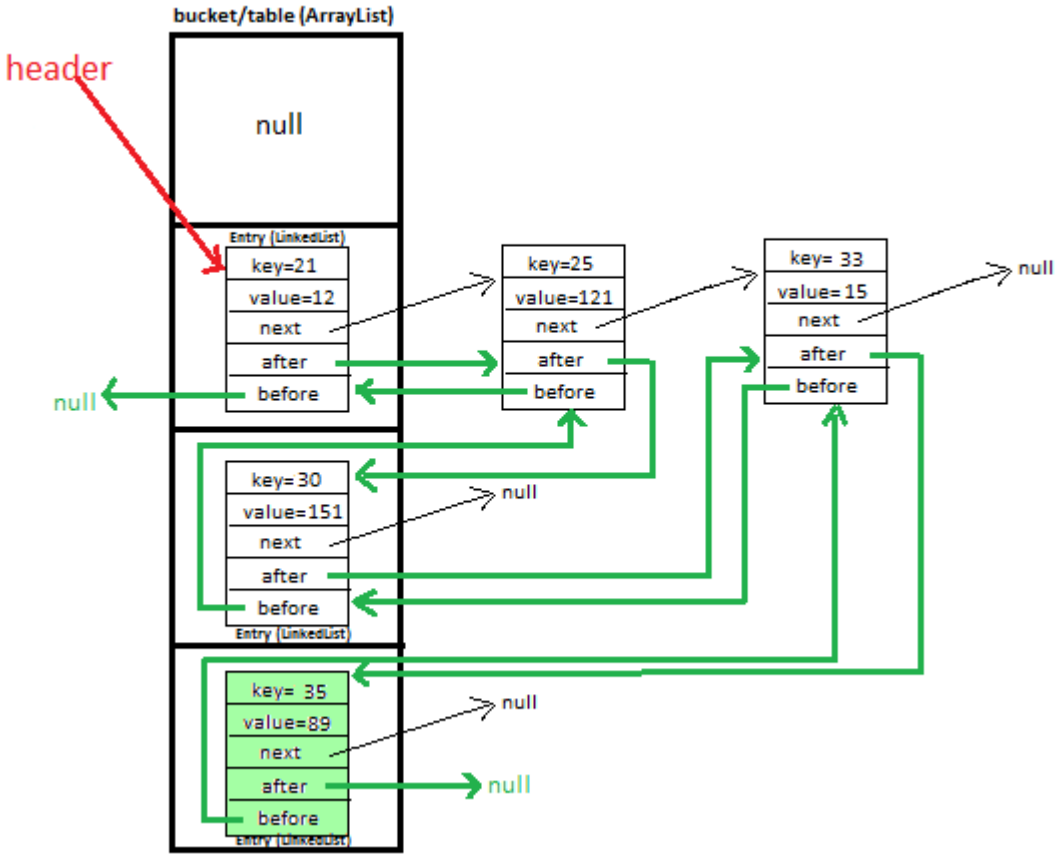
- Like the vector key, methods of the class are synchronized.
- Thread safety and therefore slows the performance.
- It doesn't allow anything that is null.



HashTable	HashMap
HashTable is **synchronized** and **thread- safe** and it can be shared with many threads.	Whereas, **HashMap** is **non-synchronized** and **not thread-safe** hence it can't be shared without synchronization code between multiple threads.
It doesn't allow any null value or null key.	HashMap allows multiple null values and one null key as well.
HashTable is internally synchronized so it can't be unsynchronized.	We can make **HashMap** synchronized by using the code as follows: **Map m1=Collections.synchronizedMap(hashMap);**
HashTable is a legacy class in Java.	It is a new class introduces in JDK 1.2.
This inherits Dictionary class.	**HashMap** inherits AbstractMap class.
HashTable is slow.	**HashMap** is faster as it can't be shared with multiple threads.

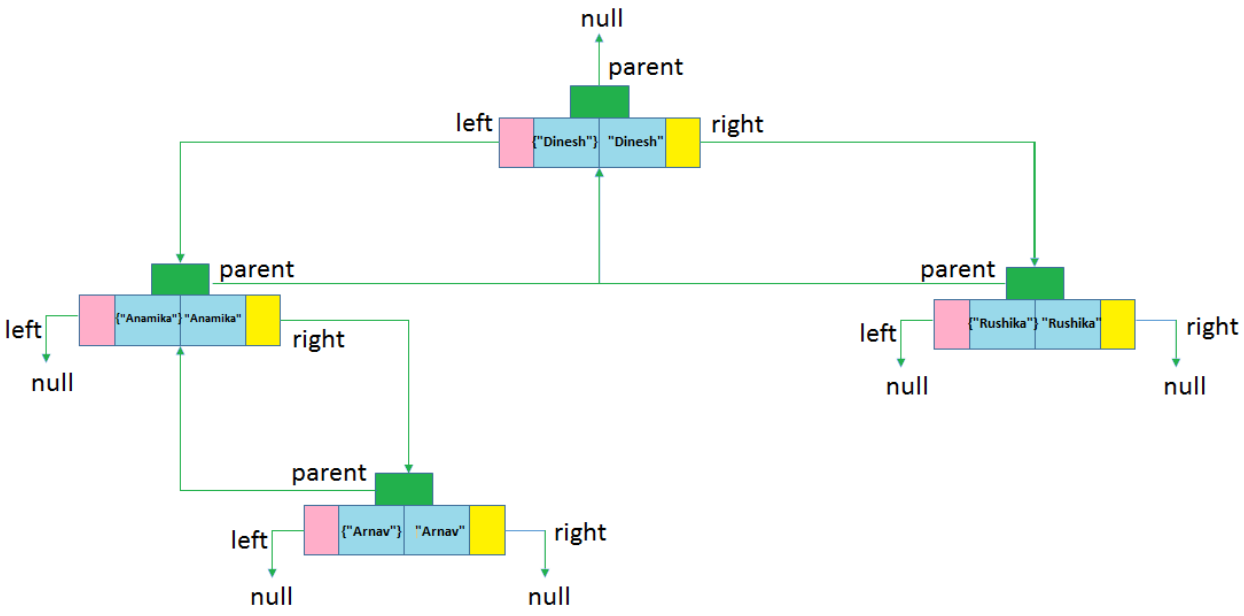
Linked Hash Map:

- Maintains insertion order.
- Slower than Hash map.
- I can expect a faster iteration.



TreeMap:

- Sorted Map.
- Like Tree set, we can construct a sort order with the constructor.



Property	HashMap	TreeMap	LinkedHashMap	HashTable
Iteration Order	Random	Sorted according to natural **order of keys**	Sorted according to the **insertion order**.	Random
Efficiency: Get, Put, Remove, ContainsKey	$O(1)$	$O(\log(n))$	$O(1)$	$O(1)$
Null keys/values	allowed	Not-allowed*	allowed	Not-allowed
Interfaces	Map	Map, SortedMap, NavigableMap	Map	Map
Synchronized	Not instead use Collection.synchronizedMap(new HashMap())			Yes but prefer to use ConcurrentHashMap
Implementation	Buckets	Red-Black tree	HashTable and LinkedList using doubly linked list of buckets	Buckets
Comments	Efficient	Extra cost of maintaining TreeMap	Advantage of TreeMap without extra cost.	Obsolete

3. Contiguous memory locations are usually used for storing actual values in an array but not in ArrayList. Explain.

An array generally contains elements of the primitive data types such as int, float, etc. In such cases, the array directly stores these elements at contiguous memory locations. While an ArrayList does not contain primitive data types. An arrayList contains the reference of the objects at different memory locations instead of the object itself. That is why the objects are not stored at contiguous memory locations.

4. Explain the term "Double Brace Initialization" in Java?

Double Brace Initialization is a Java term that refers to the combination of two independent processes. There are two braces used in this. The first brace creates an anonymous inner class. The second brace is an initialization block. When these both are used together, it is known as Double Brace Initialization. The inner class has a reference to the enclosing outer class, generally using the 'this' pointer. It is used to do both creation and initialization in a single statement. It is generally used to initialize collections. It reduces the code and also makes it more readable.

5. Why is it said that the length() method of String class doesn't return accurate results?

The length() method of String class doesn't return accurate results becauseit simply takes into account the number of characters within in the String. In other words, code points outside of the BMP (Basic Multilingual Plane), that is, code points having a value of U+10000 or above, will be ignored.The reason for this is historical. One of Java's original goals was to consider all text as Unicode; yet, Unicode did not define code points outside of the BMP at the time. It was too late to modify char by the time Unicode specified such code points.

6. ArrayList, LinkedList, and Vector are all implementations of the List interface. Which of them is most efficient for adding and removing elements from the list? Explain your answer, including any other alternatives you may be aware of.

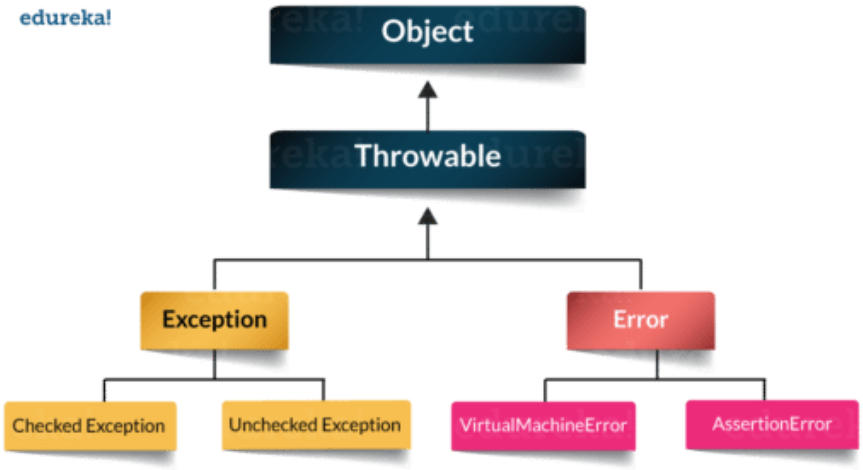
Of the three, LinkedList is generally going to give you the best performance. Here's why:

ArrayList and Vector each use an array to store the elements of the list. As a result, when an element is inserted into (or removed from) the middle of the list, the elements that follow must all be shifted accordingly. Vector is synchronized, so if a thread-safe implementation is not needed, it is recommended to use ArrayList rather than Vector.

LinkedList, on the other hand, is implemented using a doubly linked list. As a result, an inserting or removing an element only requires updating the links that immediately precede and follow the element being inserted or removed.

JAVA EXCEPTION HANDLING

1. **What is meant by Exception?**
An **Exception** is a problem that can occur during the normal flow of execution. A method can throw an exception when something wails at runtime. If that exception couldn't be handled, then the execution gets terminated before it completes the task.



2. What are the types of Exceptions?
Checked Exception: These exceptions are checked by the compiler at the time of compilation. Classes that extend Throwable class except Runtime exception and Error are called checked Exception.
For Example, ClassNotFound Exception
(ClassNotFoundException is a checked exception which occurs when an application tries to load a class through its fully-qualified name and cannot find its definition on the classpath)

Unchecked Exception:
These exceptions are not checked during the compile time by the compiler. The compiler doesn't force to handle these exceptions. It includes:

- Arithmetic Exception
- ArrayIndexOutOfBounds Exception
  (The ArrayIndexOutOfBounds exception is thrown if a program tries to access an array index that is negative, greater than, or equal to the length of the array.)

3. **What are the different ways to handle exceptions?**

a) **Using try/catch:**

The risky code is surrounded by try block. If an exception occurs, then it is caught by the catch block which is followed by the try block.

```
class Manipulation{
public static void main(String[] args){
add();
}
Public void add(){
try{
addition();
}catch(Exception e){
e.printStacktrace();
}
}
}
```

b) **By declaring throws keyword:**

At the end of the method, we can declare the exception using throws keyword.

```
class Manipulation{
public static void main(String[] args){
add();
}
public void add() throws Exception{
addition();
}
}
```

4. **What are the advantages of Exception handling?**

The advantages are as follows:

- The normal flow of the execution won't be terminated if an exception gets handled
- We can identify the problem by using catch declaration

5. **What are the Exception handling keywords in Java?**

a) **try**:

When a risky code is surrounded by a try block. An exception occurring in the try block is caught by a catch block. Try can be followed either by catch (or) finally (or) both. But any one of the blocks is mandatory.

b) **catch:**

This is followed by a try block. Exceptions are caught here.

c) **finally**:

This is followed either by try block (or) catch block. This block gets executed regardless of an exception. So generally clean up codes are provided here.

```
try {
    int budget = 1000;
    System.out.println("Success");
}
catch(Exception ex) {
    System.out.println(ex);
}
finally {
    System.out.println("This always runs");
}
```

6. **What is the final keyword in Java?**
Final variable: Once a variable is declared as final, then the value of the variable could not be changed. It is like a constant.
Final method: A final keyword in a method, couldn't be overridden. If a method is marked as a final, then it can't be overridden by the subclass.
Final class: If a class is declared as final, then the class couldn't be subclassed. No class can extend the final class.

6. What is the difference between Error and Exception?

An error is an irrecoverable condition occurring at runtime. Such as OutOfMemory error. These JVM errors you cannot repair them at runtime. Though error can be caught in the catch block but the execution of application will come to a halt and is not recoverable.

While exceptions are conditions that occur because of bad input or human error etc. e.g. FileNotFoundException will be thrown if the specified file does not exist. Or a NullPointerException will take place if you try using a null reference. In most of the cases it is possible to recover from an exception (probably by giving the user feedback for entering proper values etc.

7. **What are the differences between throw and throws?**

throw keyword	throws keyword
Throw is used to explicitly throw an exception.	Throws is used to declare an exception.
Checked exceptions can not be propagated with throw only.	Checked exception can be propagated with throws.
Throw is followed by an instance.	Throws is followed by class.
Throw is used within the method.	Throws is used with the method signature.
You cannot throw multiple exception	You can declare multiple exception e.g. public void method()throws IOException,SQLException.

JAVA THREADING

1. **What is a Thread?**

In Java, the flow of execution is called Thread. Every java program has at least one thread called the main thread, the main thread is created by JVM. The user can define their own threads by extending the Thread class (or) by implementing the Runnable interface. Threads are executed concurrently.

2. **How do you make a thread in Java?**

There are two ways available to make a thread.

a) **Extend Thread class**: Extending a Thread class and override the run method. The thread is available in java.lang.thread.

```
Public class Addition extends Thread {
public void run () {
}
}
```

The **disadvantage** of using a thread class is that we cannot extend any other classes because we have already extended the thread class. We can overload the run () method in our class.

b) **Implement Runnable interface**: Another way is by implementing the runnable interface. For that, we should provide the implementation for the run () method which is defined in the interface.

```
Public class Addition implements Runnable {
public void run () {
}
}
```

3. **Explain about join () method.**

Join () method is used to join one thread with the end of the currently running thread.

```
public static void main (String[] args){
Thread t = new Thread ();
t.start ();
t.join ();
}
```

Based on the above code, the main thread has started the execution. When it reaches the code t.start() then 'thread t' starts the own stack for the execution. JVM switches between the main thread and 'thread t'.

Once it reaches the code t.join() then 'thread t' alone is executed and completes its task, then only the main thread starts the execution.

It is a non-static method. The Join () method has an overloaded version. So we can mention the time duration in join () method also ".s".

4. **What does the yield method of the Thread class do?**

A yield () method moves the currently running thread to a runnable state and allows the other threads for execution. So that equal priority threads have a chance to run. It is a static method. It doesn't release any lock.

Yield () method moves the thread back to the Runnable state only, and not the thread to sleep (), wait () (or) block.

```
public static void main (String[] args){
Thread t = new Thread ();
t.start ();
}
public void run(){
Thread.yield();
}
}
```

5. **Explain about wait () method.**

wait () method is used to make the thread to wait in the waiting pool. When the wait () method is executed during a thread execution then immediately the thread gives up the lock on the object and goes to the waiting pool. Wait () method tells the thread to wait for a given amount of time.

Then the thread will wake up after notify () (or) notify all () method is called.

Wait() and the other above-mentioned methods do not give the lock on the object immediately until the currently executing thread completes the synchronized code. It is mostly used in synchronization.

```
public static void main (String[] args){
Thread t = new Thread ();
t.start ();
Synchronized (t) {
Wait();
}
}
```

6. **Difference between notify() method and notifyAll() method in Java.**

notify()	notifyAll()
This method is used to send a signal to wake up a single thread in the waiting pool.	This method sends the signal to wake up all the threads in a waiting spool.

7.What is thread pool in Java what is its use?

A thread pool helps mitigate the issue of performance by reducing the number of threads needed and managing their lifecycle. Essentially, threads are kept in the thread pool until they're needed, after which they execute the task and return the pool to be reused later

8. **How to stop a thread in java? Explain about sleep () method in a thread?**
We can stop a thread by using the following thread methods:

- Sleeping
- Waiting
- Blocked

Sleep: Sleep () method is used to sleep the currently executing thread for the given amount of time. Once the thread is wake up it can move to the runnable state. So sleep () method is used to delay the execution for some period.
It is a static method.

9. **When to use the Runnable interface Vs Thread class in Java?**

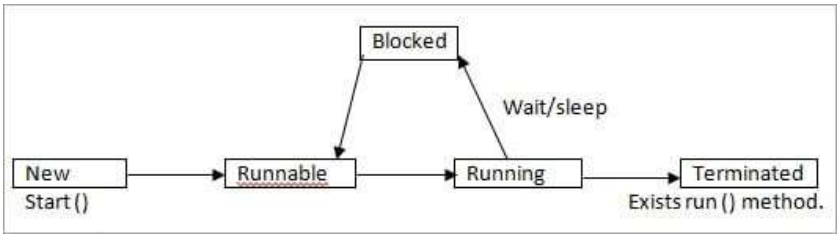
- If we need our class to extend some other classes other than the thread then we can go with the runnable interface because in java we can extend only one class.
- If we are not going to extend any class then we can extend the thread class.

10. **Difference between start() and run() method of thread class.**
Start() method creates a new thread and the code inside the run () method is executed in the new thread. If we directly called the run() method then a new thread is not created and the currently executing thread will continue to execute the run() method.

11. **What is Multi-threading?**
Multithreading is a Java feature that allows concurrent execution of two or more parts of a program for maximum utilization of CPU. Each part of such program is called a thread. So, threads are light-weight processes within a process.

Advantages of Java Multithreading
1) It doesn't block the user because threads are independent and you can perform multiple operations at the same time.
2) You can perform many operations together, so it saves time.
3) Threads are independent, so it doesn't affect other threads if an exception occurs in a single thread.

12. **Explain the thread life cycle in Java**.



- **New**: In New state, a Thread instance has been created but start () method is not yet invoked. Now the thread is not considered alive.
- **Runnable**: The Thread is in the runnable state after the invocation of the start () method, but before the run () method is invoked. But a thread can also return to the runnable state from waiting/sleeping. In this state, the thread is considered alive.
- **Running**: The thread is in a running state after it calls the run () method. Now the thread begins the execution.
- **Non-Runnable**(Blocked): The thread is alive but it is not eligible to run. It is not in the runnable state but also, it will return to the runnable state after some time. Example: wait, sleep, block.
- **Terminated**: Once the run method is completed then it is terminated. Now the thread is not alive.

JAVA SYNCHRONIZATION

1. **What is Synchronization?**

Synchronization makes only one thread to access a block of code at a time. If multiple threads access the block of code, then there is a chance for inaccurate results at the end. To avoid this issue, we can provide synchronization for the sensitive block of codes.
The synchronized keyword means that a thread needs a key in order to access the synchronized code.
Locks are per objects. Every Java object has a lock. A lock has only one key. A thread can access a synchronized method only if the thread can get the key to the objects to lock.
For this, we use the "Synchronized" keyword.

```
public class ExampleThread implements Runnable{
 public static void main (String[] args){
 Thread t = new Thread ();
 t.start ();
 }
 public void run(){
 synchronized(object){
 {
 }
}
```

2. **What is the disadvantage of Synchronization?**

Synchronization is not recommended to implement all the methods. Because if one thread accesses the synchronized code then the next thread should have to wait. So it makes a slow performance on the other end.

3. **What is meant by Serialization?**

Converting a file into a byte stream is known as Serialization. The objects in the file are converted to bytes for security purposes. For this, we need to implement a java.io.Serializable interface. It has no method to define.
Variables that are marked as transient will not be a part of the serialization. So we can skip the serialization for the variables in the file by using a transient keyword.

4. **How to not allow serialization of attributes of a class in Java?**

The NonSerialized attribute can be used to prevent member variables from being serialized. You should also make an object that potentially contains security-sensitive data nonserializable if possible. Apply the NonSerialized attribute to certain fields that store sensitive data if the object must be serialized. If you don't exclude these fields from serialisation, the data they store will be visible to any programmes with serialization permission.

JAVA SERVLETS

1. **What is a servlet?**
   - Java Servlet is server-side technologies to extend the capability of web servers by providing support for dynamic response and data persistence.
   - The javax.servlet and javax.servlet.http packages provide interfaces and classes for writing our own servlets.
   - All servlets must implement the javax.servlet.Servlet interface, which defines servlet lifecycle methods. When implementing a generic service, we can extend the GenericServlet class provided with the Java Servlet API. The HttpServlet class provides methods, such as doGet() and doPost(), for handling HTTP-specific services.
   - Most of the times, web applications are accessed using HTTP protocol and thats why we mostly extend HttpServlet class. Servlet API hierarchy is shown in below image.

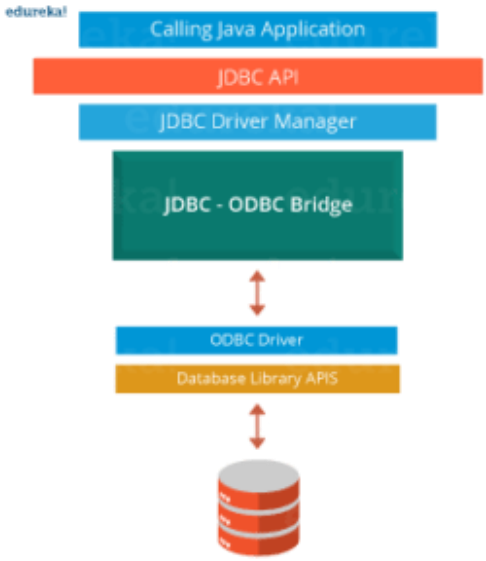
2. **What are the differences between Get and Post methods?**

Get	Post
Limited amount of data can be sent because data is sent in header.	Large amount of data can be sent because data is sent in body.
Not Secured because data is exposed in URL bar.	Secured because data is not exposed in URL bar.
Can be bookmarked	Cannot be bookmarked
Idempotent	Non-Idempotent
It is more efficient and used than Post	It is less efficient and used

JAVA JDBC

1. **What is JDBC Driver?**
JDBC Driver is a software component that enables java application to interact with the database. There are 4 types of JDBC drivers:

- JDBC-ODBC bridge driver
- Native-API driver (partially java driver)
- Network Protocol driver (fully java driver)
- Thin driver (fully java driver)



2. **What are the steps to connect to a database in java?**
- Registering the driver class
- Creating connection
- Creating statement
- Executing queries
- Closing connection

3. **What are the JDBC API components?**
The java.sql package contains interfaces and classes for JDBC API.

Interfaces:
- Connection
- Statement
- PreparedStatement
- ResultSet
- ResultSetMetaData
- DatabaseMetaData
- CallableStatement etc.

Classes:

- DriverManager
- Blob
- Clob
- Types
- SQLException etc.

4. **What do you understand by JDBC Statements?**
JDBC statements are basically the statements which are used to send SQL commands to the database and retrieve data back from the database. Various methods like execute(), executeUpdate(), executeQuery, etc. are provided by JDBC to interact with the database.

JDBC supports **3** types of statements:

- **Statement**: Used for general purpose access to the database and executes a static SQL query at runtime.
- **PreparedStatement**: Used to provide input parameters to the query during execution.
- **CallableStatement**: Used to access the database stored procedures and helps in accepting runtime parameters.

JAVA SPRING

1. **What is Spring?**
Wikipedia defines the Spring framework as "an application framework and inversion of control container for the Java platform. The framework's core features can be used by any Java application, but there are extensions for building web applications on top of the Java EE platform." Spring is essentially a lightweight, integrated framework that can be used for developing enterprise applications in java.

3. **Name the different modules of the Spring framework.**
Some of the important Spring Framework modules are:

- Spring Context – for dependency injection.
- Spring AOP – for aspect oriented programming.
- Spring DAO – for database operations using DAO pattern
- Spring JDBC – for JDBC and DataSource support.
- Spring ORM – for ORM tools support such as Hibernate
- Spring Web Module – for creating web applications.
- Spring MVC – Model-View-Controller implementation for creating web applications, web services etc.

Prepared By: Shakhawat Hossain
Email: Shakhawathossain.se@gmail.com
Github: https://github.com/Imshakhawat