# No BS Java Notes

Approaches of Programming:

1. Procedural : subset of structured programming and is based on concept of procedure call, they contain a series of computational steps to be performed eg: BASIC, PASCAL
2. Structured : enforces a logical structure making it easy to understand and efficient, the whole code is divided into smaller modules or units but it does not allow reusability of code! Separates programs data from its functionality eg: C
3. Modular: code is divided into modules which performs functions which can be reused and debugged easily eg: header files of C
4. Object Oriented: revolves around objects which have unique attributes (variables) and behavior (methods) eg: Java, Python, C++

Differences b/w C++ and Java

1. C++ is procedural and OOP whereas Java is purely OOP
2. C++ uses pointers whereas Java uses References
3. C++ supports header files and pre-processor directives, whereas there are no Pre-processor directives in Java and rather uses Packages which are imported
4. C++ uses new and delete keywords for storage allocation whereas Java uses Garbage Collector.
5. C++ does not support DB connectivity whereas Java does.
6. C++ is a compiled Language and Java is an interpreted language
7. C++ uses operator overloading whereas Java does not
8. C++ does not support Multithreading whereas Java does
9. C++ supports inheritance, java supports too, but Java does not support Multiple Inheritance

Java

1. It a high level, object oriented, robust and secure language and can be run on any-platform
2. It was named after coffee called Java, was previously known as Oak and was developed by James Gosling and Sun Microsystems in 1995
3. Features of Java:
    1. It is easy to learn and has simple syntax
    2. It is object oriented language
    3. It is portable as the code is converted to byte code which can run on any platform

4. It is WORA language (Write Once, Run Anywhere)
5. It is a secure language because it does not use explicit pointer and runs inside a virtual machine
6. It is robust because it uses strong memory management and deals with exception handling
7. It is architecture neutral because there are no implementation dependent features like size of primitive data types
8. It supports multithreading : separate programs running concurrently and its main advantage is that it does not occupy memory for each thread and shares a common memory area.
9. It is faster than other interpreted languages but is slower than compiled languages like C++
10. It supports dynamic loading of classes, meaning that classes can be loaded on demand.

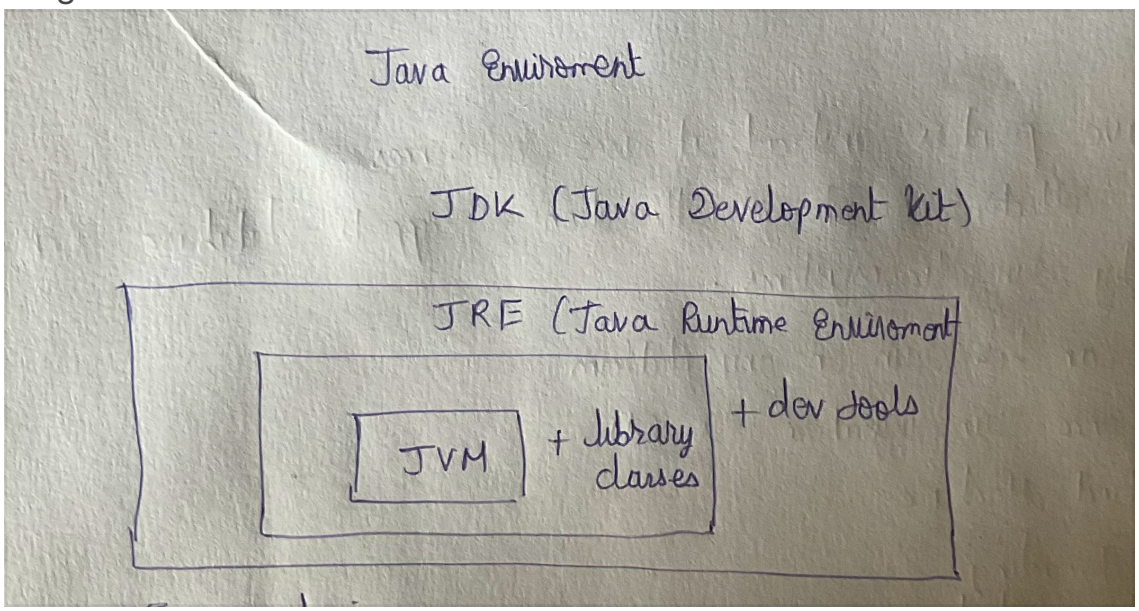
OOP Concepts

1. A class is a logical entity whereas an Object is a physical as well as a logical entity.
2. An entity which has state and behavior and is an instance of a class is known as an object
3. State refers to data of the object and behavior refers to the functionality
4. JVM assigns ID to each object for identification purposes
5. A class is a blueprint from which objects can be created.
6. Object is a real-world entity/runtime entity/entity having state and behavior/instance of a class
7. Inheritance:
    1. It is a mechanism through which one object can acquire all properties and behaviors of a parent class. It allows you to create new classes which are an extension of existing class and allows you to add new methods and data to the child class, thereby supporting reusability and thereby increasing efficiency.
    2. It can be represented as “IS-A” relationship or “parent-child” relationship.
8. Polymorphism:
    1. It is a way in which you can perform an action in multiple ways.
    2. There are two types of polymorphism in Java: compile-time polymorphism(method overloading) and runtime polymorphism (method overriding)
9. Abstraction:
    1. It is the process of hiding the implementation details from the user and showing the required functionality!
10. Encapsulation:
    1. It is the process of wrapping code and data into a single unit,

which is achieved technically by the means of making a class

11. Message passing refers to sending message from one thread to another thread

Coupling & Cohesion

1. Coupling refers to the interdependence of modules
2. Cohesion refers to how closely related the functions within a single module are.
3. Achieve low coupling and high cohesion.
4. Diagram of how Java works
5. 


Namespace:

1. It is a collection of related names or identifiers which helps to separate these identifiers from similar identifiers in other namespaces or global namespaces
2. In C++ namespace can be used in three ways: :: operator, the using declaration and the using directive

Type Casting & Type conversion:

1. In Type casting, a data type is converted into another data type by the programmer and in this the destination data type may be smaller that source data type and is hence known as narrowing conversion.
2. In Type conversion, a data type is converted automatically into another data type by the computer and in this the destination data type cannot be smaller than the source data type and is hence known as widening conversion.

Function Overloading

1. It is a feature of C++ in which two or more functions can have the same name but must have different parameters, either they should have different data type or different number of parameters.

Constructors:

1. It is a special method which is invoked at the time an object of a class is created and is used to initialize new objects
2. It has the same name as the class
3. They do not have a return type.
4. Types of constructors:
    1. Default : it does not have any parameters and uses default values. (Aka Zero argument constructor)
    2. Parametrized: it takes parameters and used to create objects which have specific initial values
    3. Copy: It takes reference to another object of same class and is used to create a copy of it
    4. Move: Take the reference to another object and transfers it using temporary storage and does not create extra copies (only in C++)

Destructors:

1. It is an instance member function and is invoked whenever an object is going to be destroyed.
2. It destroys the class objects created by constructor and is preceded by (~) sign
3. There can be only one destructor and it cannot be overloaded as it is the only way to destroy the object created by the constructor
4. It cannot be declared as static or constant and does not require any argument and does not return any value either.

Scope of variables:

1. Instance Variables: They are created when objects are instantiated and are hence associated with the objects
2. Class Variables: They are global to a class and belong to the entire set of objects created by the class and only one memory location is created for each class variable
3. Local Variables: They are declared and used inside methods and are not available outside a method.

For-each loop:

1. It is exclusively used to loop through array elements and its syntax is
    for (String i:cars){

```
            sout(i);
        }
```

Break: used to exit loop and in switch case
Continue: used to move to next iteration

Method Overloading:

1. In this methods have different parameters or definitions na dis sued to perform similar tasks but Ising different input parameters. When a method is called, Java matches the method name first and then the number and type of parameters and this process happens at compile time, hence is known as compile time polymorphism or static polymorphism.

Static Members:

1. Every time a class is instantiated, a new copy of instance variables and methods is created which are accessed using objects. To define a variable/method which can be used globally without needing an object, we can do that by adding static keyword. They can be used even by other classes!

Inheritance:

1. Forms of inheritance:
    1. Single
    2. Multiple
    3. Hierarchical
    4. Multilevel
2. You use the extends keyword, which allows the subclass to have access to own methods/variables along with that of parent class.
3. A subclass constructor is used to construct the instance variables of both subclass and superclass, it uses the keyword super to invoke constructor of super class.
4. Super keyword may be used within a subclass and must be the first statement with the subclass constructor and if default constructor is there in super class, then it is mandatory to use super keyword

Method Overriding:

1. If a subclass defines a method that has the **same name**, **return type**, and **parameters** as a method in its superclass, the subclass method **overrides** the superclass method.If a subclass defines a method that has the **same name**, **return type**, and **parameters** as a method in its superclass, the subclass method **overrides** the superclass method.

2. When you call the method on an object of the subclass, the **subclass's version** of the method is executed, even though the method is defined in both the superclass and the subclass.
3. All methods and variables can be overridden by subclasses, to prevent overriding we can declare the methods/variables as final using the final keyword

Abstract methods and class:

1. It is used to indicated that a method must always be redefined in a subclass, which makes overriding compulsory and this can be done by adding the keyword abstract before the method definition.

Dynamic Method Dispatch (Late Binding):

1. Method overriding is one of the ways in which Java supports Runtime Polymorphism.
2. Dynamic Method dispatch is the mechanism by which a call o an overridden method is resolved at run time rather than at compile time.
3. This happens because java determines which version of the method is to be executed at the time the method call occurs, hence is determined at run time.
4. A superclass reference variable can refer to subclass object and this is known as upcasting which is used by Java to resolve calls to overrriden methods at run time.

Instance-of operator

1. It allows the programmer to verify whether an object is an instance of a specific subclass, class or interface, it is useful when code has inheritance in it
2. The instanceof operator takes the object name on the left side of it and takes class name on the right side (<objectname> instanceof <classname>) and it results a boolean value of true/false

Garbage Collection in Java

1. It is the process by which Java performs automatic memory management.
2. When java programs are run, the objects are created in heap and after sometime, few objects might not be needed, hence the garbage collector finds these unused objects and deletes them to free up memory
3. Its main objective is to free heap memory by deleting unreachable objects.

4. It is an automatic process, the garbage collector looks at the heap memory and identifies the objects which are in use (referenced object) and which are not in use (unreferenced objects) and the memory used by unreferenced objects is reclaimed.
5. The garbage collection implementation lives in the JVM
6. An object is said to be unreachable if it does not contain any reference to it, or is a part of island of isolation.
7. Island of isolation in Java is a group of objects which have reference to each other but to whom the active program has no reference (extra stuff, might skip but hehe)
8. Ways to make object eligible for garbage collector
    1. Although it is not necessary for programmer to mark objects eligible for garbage collection but there are ways to do it still:
        1. Nullifying the reference variable
        2. Re-assigning the reference variable
        3. An object created inside the method
        4. Island of Isolation
9. An unreferenced object is destroyed only when JVM runs the Garbage Collector
10. Two ways to run Garbage Collector: (still there is no guarantee that it will work)
    1. Using system.gc()
    2. Using Runtime.getRuntime().gc():
11. Before destroying an object, the garbage collector calls the finalise() method on the object to perform cleanup activities, and once done, it destroys the object, it is present in the object with the following syntax: protected void finalize() throws Throwable
12. It is recommended to override the finalize()method
13. Advantages: makes java memory efficient and removes/cleans up memory automatically

Arrays

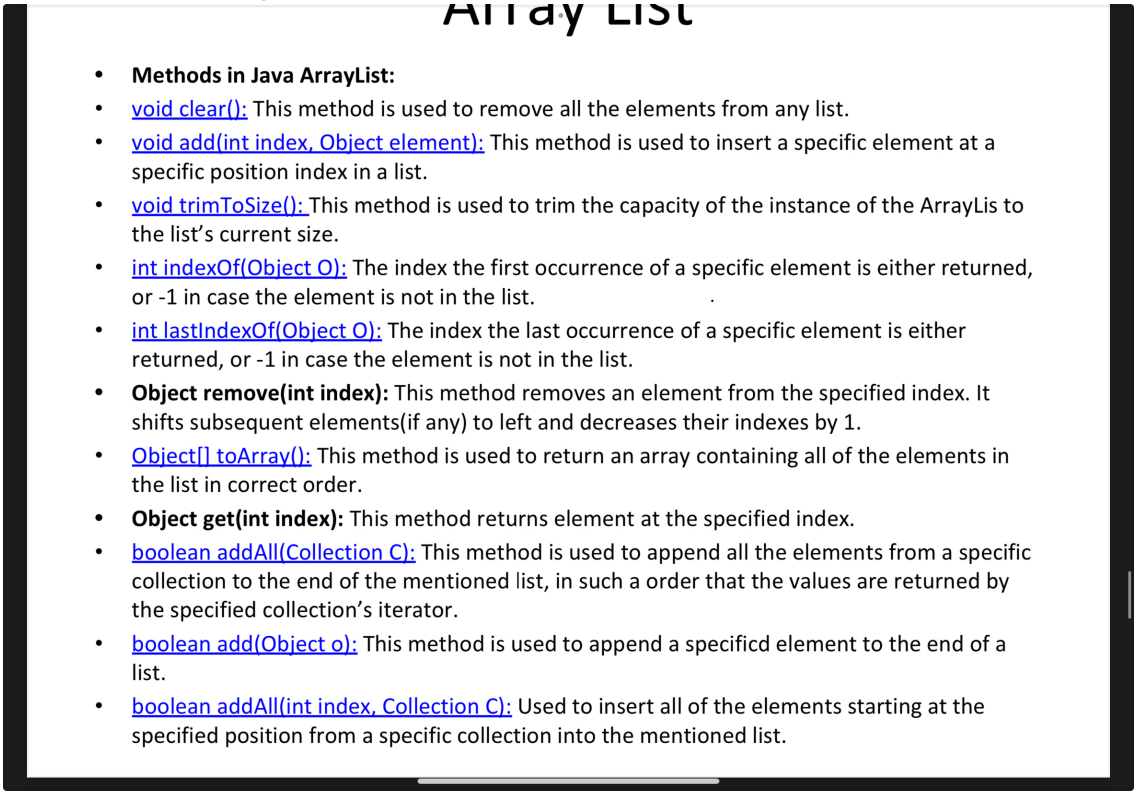
1. Stores data in contiguous memory locations
2. You know how to play with arrays
3. Do 2D Array/Jagged Array questions

Java Collections Framework

It is a collection of many useful data types like linked list, re-sizeable array lists, red-black trees, hash-based maps, dictionary, set etc. It provides abstractions, so that you can work with the functionality rather than focussing on its implementation. It allows you to use the right data structure. The framework promotes code reusability, consistency, and flexibility by offering reusable

algorithms for sorting, searching, and shuffling. Its versatile design enables developers to choose the best data structure based on specific needs, ensuring optimal performance and ease of use in applications.

Collections in Java

1. We know that arrays are of fixed size and the Java Collection Framework provides lots of different useful data types like linked list etc
2. Different collection interfaces are: List, Set, Queue, Maps
3. List is an interface with Arraylist, vector and linked list, array list is used when there are more retrievals and linked list is used when there are more removals
4. Arraylist provides us dynamic arrays in Java. It inherits AbstractList class, implements List interface and is based on array data structure
5. Arraylist is initialized by a size, but can increase or decrease if collection grows/shrinks
6. Constructors in ArrayList
    1. ArrayList(): used to build an empty array list
    2. ArrayList(Collection c): used to build an array initialized with elements from collection c
    3. ArrayList(int capacity): used to build an array list with initial capacity being specified
7. ArrayList declaration: ArrayList<String> xyz = new ArrayList<String>();
8. Methods of ArrayList:
9. 

| Array | ArrayList |
| --- | --- |
| Array is a fixed length data structure that is you can not change length of Array once created in Java | ArrayList is a variable length Collection class that is ArrayList re-size itself when gets full depending upon capacity and load factor |
| We can not use Generics along with Array, as Array instance knows about what kind of type it can hold and throws ArrayStoreException, if you try to store type which is not convertible into type of Array | ArrayList allows you to use Generics to ensure type-safety |
| Array can contain both primitive data types as well as objects of a class depending on the definition of the array | ArrayList only supports object entries, not the primitive data types |
| Java provides add() method to insert element into ArrayList | We use assignment operator to store element into Array |
| Its mandatory to provide size of Array | We can create ArrayList with default size of 10 |
| Each array object has the length variable which returns the length of the array | Length of the ArrayList is provided by the size() method |
| Array can be multi dimensional | ArrayList is always single dimensional |
| Use Array, when you know that the size is fixed | Use ArrayList, when you don't know about the size of elements |

How to sort an ArrayList in ascending order? Collections.sort(numbers);
How to sort an Arraylist in descending order? Collections.sort(numbers, Collections.reverseOrder());

**Look into Vector Functions from PPT or any website**

A comparator interface is used to order the objects of user-defined classes. A comparator object is capable of comparing two objects of the same class. (Check its implementation once)

## Difference in String & StringBuffer

| no | String | StringBuffer |
|---|---|---|
| 1) | The String class is immutable. | The StringBuffer class is mutable. |
| 2) | String is slow and consumes more memory when we concatenate too many strings because every time it creates new instance. | StringBuffer is fast and consumes less memory when we concatenate t strings. |
| 3) | String class overrides the equals() method of Object class. So you can compare the contents of two strings by equals() method. | StringBuffer class doesn't override the equals() method of Object class. |
| 4) | String class is slower while performing concatenation operation. | StringBuffer class is faster while performing concatenation operation. |

## StringBuffer & StringBuilder class difference

| No. | StringBuffer | StringBuilder |
|---|---|---|
| 1) | StringBuffer is *synchronized* i.e. thread safe. It means two threads can't call the methods of StringBuffer simultaneously. | StringBuilder is *non-synchronized* i.e. not thread safe. It means two threads can call the methods of StringBuilder simultaneously. |
| 2) | StringBuffer is *less efficient* than StringBuilder. | StringBuilder is *more efficient* than StringBuffer. |
| 3) | StringBuffer was introduced in Java 1.0 | StringBuilder was introduced in Java 1.5 |

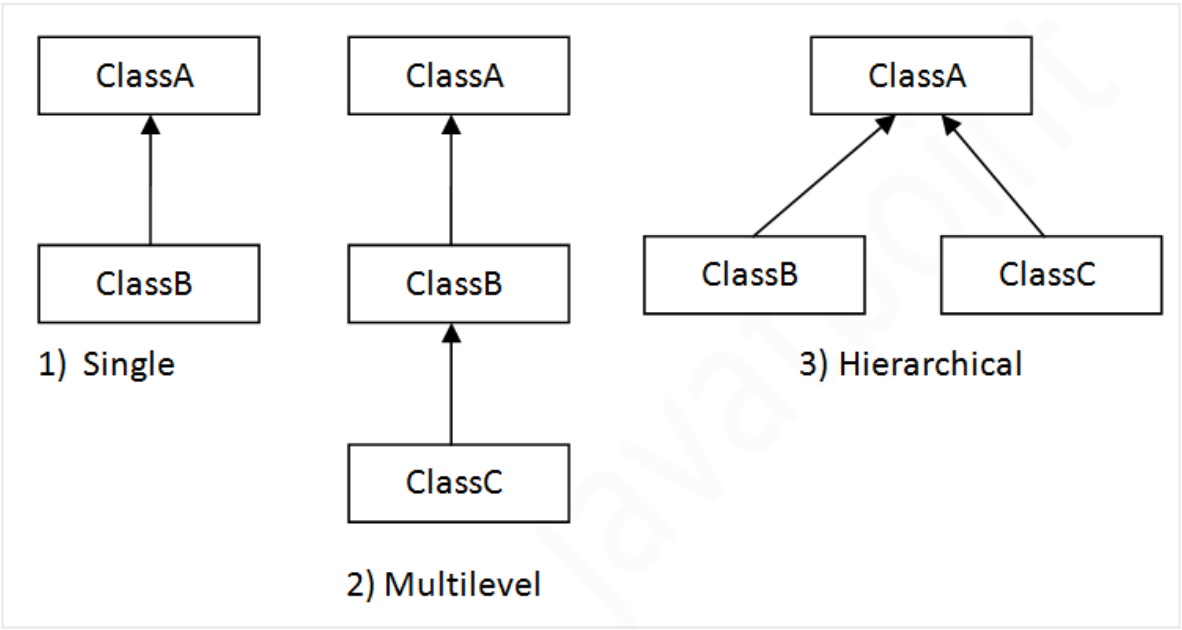
A Wrapper class is a class that encapsulates types so that those types can be used to create object instances and methods in another class that need those types.

Multiple Inheritance

1. Java does not support Multiple inheritance and hence it provides interfaces to support it. Although a class cannot be a subclass of more than one class but with interface, we can create classes that build on existing classes without the problems caused by multiple inheritance.
2. Interface:
    1. It is basically a kind of class and has methods/variables but the only major difference is that interface defines only abstract methods and final fields, it does not specify any code to implement the methods and hence it becomes the responsibility of the class to define the method while implementing an interface.
    2. A interface can be sub interfaced from other interfaces using the “extends” keyword and it will inherit all the members of the super interface  but a class can implement interface using “implements” keyword only and will inherit everything of the interface
    3. 

## Difference in Interface and class

| Sr No | Class | Interface |
|---|---|---|
| 1. | The members of a class can be constant or variables. | The members of an interface are always declared as constant that is their values are final. |
| 2. | The class definition can contain the code for each of its methods. That is, the methods can be abstract or non-abstract. | The methods in an interface are abstract in nature i.e. there is no code associated with them. It is later defined by the class that implements the interface. |
| 3. | It can be instantiated  by declaring objects. | It cannot be used to declare objects . It can only be inherited by a class. |
| 4. | It can use various access specifiers like public, private or protected. | It can only use public access specifier. |




Packages:

1. It is a way in which you can group a number of related classes/ interfaces together into a single unit.
2. This allows us to reuse the functionality of classes included in the package. It helps us to increase our efficiency of coding while maintaining a clean code
3. Also the name of classes can be unique as compared with classes in other packages. That two classes in two different packages can have the same name and if there is a clash them ,classes can be accessed with their fully qualified name
4. There are six foundational java packages:
    1. LANG: contains classes for math function, thread, primitive types
    2. UTIL: contains vectors, scanners, date
    3. IO: input output stream
    4. awt: for implementing GUI
    5. Net: for networking
    6. Applet: for creating applets

Access Specifiers

1. Public: makes It visible everywhere
2. Private: only visible in that class and cannot be inherited
3. Protected: only visible in a class, subclass (through inheritance) and also with all classes in the same package

# Visibility Modifiers

| Accessible to: | public | protected | Package (default) | private |
|---|---|---|---|---|
| Same Class | Yes | Yes | Yes | Yes |
| Classin package | Yes | Yes | Yes | No |
| Subclassin different package | Yes | Yes | No | No |
| Non-subclass different package | Yes | No | No | No |

Types of Errors:

1. Compile time errors : Syntax errors
2. Run - time errors: logical errors

Exception Handling

1. It can be defined as a way in which we deal with errors, in this technique, if we want the program to continue the execution of the remaining code, then we try to catch the exception object thrown by the error condition and then display an appropriate mediate for taking correct actions
2. Way in which it is done:
    1. Find the problem (HIT)
    2. Inform that error has occurred(Throw the exception)
    3. Receive  the error information(Catch the exception)
    4. Take corrective action(handling it)
3. There are two types of exceptions:
    1. Checked Exception: exceptions checked at compile time
    2. Unchecked Exception: exception handled by JVM
4. Throws is a keyword in Java which is used in the signature of method to indicate that this method might throw one of the listed exceptions

## How to handle Java exceptions?

This is accomplished using the keywords: **try**, **catch**, **throw**, **throws**, and **finally**.

- You **try** to execute the statements contained within a block of code. *(A block of code is a group of one or more statements surrounded by braces.)*
- If you detect an exceptional condition within that block, you **throw** an exception object of a specific type.
- You **catch** and process the exception object using code that you have designed.
- You optionally execute a block of code, designated by **finally**, which needs to be executed whether or not an exception occurs. *(Code in the **finally** block is normally used to perform some type of cleanup.)*

---

Multithreading

1. It is similar to multi-processing
2. A thread is the smallest unit in multithreading
3. It supports execution of multiple parts of a single program simultaneously
4. If one thread is executing a long process, then the whole entire application need not wait for it to finish
5. Multithreading is used in any application which requires tasks to be performed independently/used in GUI/network/data applications
6. How does it work?
    1. Each thread is given its context
    2. The scheduler decides which thread executes at any given time
    3. The scheduler maintains a list of ready threads and a list of threads waiting
    4. Each thread has a priority and the one with highest priority is run first
7. There are two ways to create own threads:
    1. To subclass the thread class and override the run method
    2. By using an interface called runnable
8. The correct way to stop a thread is to have run method terminate by adding a boolean variable which indicates whether the thread should continue or not or provide a set method for that variable which can be invoked by another thread

Java Iterator:

1. An iterator is an object that can be used to loop through collections like ArrayList
2. Methods of Java Iterator:
    1. hasnext(): returns true if iteration has more elements left
    2. next(): return next element in the iteration, throws NoSuchElementException if no more element present
    3. remove(): removes the next element in the iteration

| No | Method Overloading | Method Overriding |
|---|---|---|
| 1) | Method overloading is used *to increase the readability* of the program. | Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class. |
| 2) | Method overloading is performed *within class*. | Method overriding occurs *in two classes* that have IS-A (inheritance) relationship. |
| 3) | In case of method overloading, *parameter must be different*. | In case of method overriding, *parameter must be same*. |
| 4) | Method overloading is the example of *compile time polymorphism*. | Method overriding is the example of *run time polymorphism*. |
| 5) | In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter. | *Return type must be same or covariant* in method overriding. |

- Consist of abnormal conditions that can be handled explicitly.
- If one handles the exception then our code will continue to execute smoothly.

## Difference between Checked and Unchecked Exception

| Checked Exceptions | Unchecked Exceptions |
|---|---|
| Occur at compile time. | Occur at runtime. |
| The compiler checks for a checked exception. | The compiler doesn't check for exceptions. |
| Can be handled at the compilation time. | Can't be caught or handled during compilation time. |
| The JVM requires that the exception be caught and handled. | The JVM doesn't require the exception to be caught and handled. |
| Example of Checked exception- 'File Not Found Exception' | Example of Unchecked Exceptions- 'No Such Element Exception' |

```
    }
}
```

## Java Throw vs Throws

| Throw | Throws |
|---|---|
| This keyword is used to explicitly throw an exception. | This keyword is used to declare an exception. |
| A checked exception cannot be propagated with throw only. | A checked exception can be propagated with throws. |
| The throw is followed by an instance and used with a method | Throws are followed by class and used with the method signature. |
| You cannot throw multiple exceptions. | You can declare multiple exceptions |

## User-defined Exceptions

```
}
}
```

The BigDecimal class provides operations for arithmetic, scale manipulation, rounding, comparison, hashing, and format conversion.

## Java Exception Keywords

| Keyword | Description |
|---|---|
| try | The "try" keyword is used to specify a block where we should place an exception code. It means we can't use try block alone. The try block must be followed by either catch or finally. |
| catch | The "catch" block is used to handle the exception. It must be preceded by try block which means we can't use catch block alone. It can be followed by finally block later. |
| finally | The "finally" block is used to execute the necessary code of the program. It is executed whether an exception is handled or not. |
| throw | The "throw" keyword is used to throw an exception. |
| throws | The "throws" keyword is used to declare exceptions. It specifies that there may occur an exception in the method. It doesn't throw an exception. It is always used with method signature. |

## The Finally Block