**String in Data Structure**

A string is a sequence of characters. The following facts make string an interesting data structure.

* Small set of elements. Unlike normal array, strings typically have smaller set of items. For example, lowercase English alphabet has only 26 characters. ASCII has only 256 characters.
* Strings are immutable in programming languages like Java, Python, JavaScript and C#.
* Many String Problems can optimized using the fact that the character set size is small. For example sorting can be done faster, counting frequencies of items is faster and many interesting interview questions are based on this.

https://www.geeksforgeeks.org/dsa/string-data-structure/

**String vs StringBuilder vs StringBuffer in Java**

A **string** is a sequence of characters. In Java, String objects are immutable, which simply means once created, their values can not be changed. In Java, String, **StringBuilder**, and **StringBuffer** are used for handling strings. The main difference is:

* [String](https://www.geeksforgeeks.org/java/string-class-in-java/)**:** Immutable, meaning its value cannot be changed once created. It is thread-safe but less memory-efficient.
* [StringBuilder](https://www.geeksforgeeks.org/java/stringbuilder-class-in-java-with-examples/): Mutable, not thread-safe, and more memory-efficient compared to String. Best used for single-threaded operations.
* [StringBuffer](https://www.geeksforgeeks.org/java/stringbuffer-class-in-java/): Mutable and thread-safe due to synchronization, but less efficient than StringBuilder in terms of performance.

Difference Between String, StringBuilder, and StringBuffer

|  |  |  |
| --- | --- | --- |
| Feature | Action Class | JavaScriptExecutor |
| **Purpose** | Simulate real user keyboard/mouse actions | Execute JavaScript directly in browser |
| **Keyboard Support** | ✅ Yes (e.g., sendKeys(Keys.ENTER)) | ⚠️ Limited (manual event dispatching) |
| **Real User Simulation** | ✅ Yes | ❌ No |
| **DOM Manipulation** | ❌ No | ✅ Yes |
| **Use Case** | Form filling, keyboard shortcuts | Hidden elements, custom JS events |
| **Reliability** | ✅ High (if element is interactable) | ⚠️ Depends on JS and browser behavior |

Let us consider the code below with three concatenation functions with three different types of parameters, String, StringBuffer, and StringBuilder. Let us clear out the understanding between them via a single Java program below, from which we will be drawing out conclusions from the output generated, to figure out the main differences between String vs StringBuilder, vs StringBuffer in Java.

class Geeks {

// Method 1

// Concatenates to String

public static void concat1(String s1)

{

s1 = s1 + "forgeeks";

}

// Method 2

// Concatenates to StringBuilder

public static void concat2(StringBuilder s2)

{

s2.append("forgeeks");

}

// Method 3

// Concatenates to StringBuffer

public static void concat3(StringBuffer s3)

{

s3.append("forgeeks");

}

// Method 4

// Main driver method

public static void main(String[] args)

{

// Custom input string

// String 1

String s1 = "Geeks";

// Calling above defined method

concat1(s1);

// s1 is not changed

System.out.println("String: " + s1);

// String 1

StringBuilder s2 = new StringBuilder("Geeks");

// Calling above defined method

concat2(s2);

// s2 is changed

System.out.println("StringBuilder: " + s2);

// String 3

StringBuffer s3 = new StringBuffer("Geeks");

// Calling above defined method

concat3(s3);

// s3 is changed

System.out.println("StringBuffer: " + s3);

}

}

**Output**

String: Geeks

StringBuilder: Geeksforgeeks

StringBuffer: Geeksforgeeks

**Explanation:**

* **Concat1**: In this method, the string "Geeks" is passed, and we perform s1 = s1 + "forgeeks". Since String is immutable, a new string is created, and s1 in concat1() points to it. The original string in main() remains unchanged.
* **Concat2:** Here, s2.append("forgeeks") modifies the original StringBuilder object. Since StringBuilder is mutable, it updates the string directly in main() to "Geeksforgeeks".
* **Concat3**: StringBuffer and StringBuilder are similar, but StringBuffer is thread-safe due to synchronized methods, while StringBuilder is not, making it thread-unsafe.

**When to Use Which?**

Geeks now you must be wondering when to use which one, do refer below as follows:

* If a string is going to remain constant throughout the program, then use the String class object because a String object is immutable.
* If a string can change (for example: lots of logic and operations in the construction of the string) and will only be accessed from a single thread, using a StringBuilder is good enough.
* If a string can change and will be accessed from multiple threads, use a StringBuffer because StringBuffer is synchronous, so you have thread-safety.
* If you don't want thread-safety than you can also go with StringBuilder class as it is not synchronized.

**Conversion Between String, StringBuilder, and StringBuffer**

Sometimes there is a need for converting a string object of different classes like String, StringBuffer, StringBuilder to one another. Below are some techniques to do the same. Let's cover all use cases as follows:

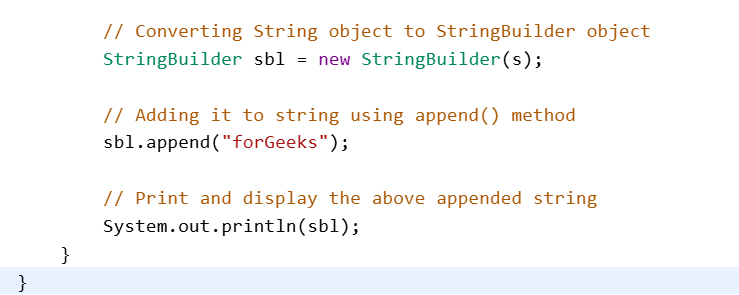
* From String to StringBuffer and StringBuilder
* From StringBuffer and StringBuilder to String
* From StringBuffer to StringBuilder or vice-versa

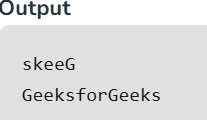
**Case 1: Convert String to StringBuffer and StringBuilder**

This one is an easy way out as we can directly pass the String class object to StringBuffer and StringBuilder class constructors. As the String class is immutable in java, so for editing a string, we can perform the same by converting it to StringBuffer or StringBuilder class objects.

A screenshot of a computer program

AI-generated content may be incorrect.





**Explanation:**The above example shows how to convert a String into StringBuffer and StringBuilder. We are reversing the String using StringBuffer and then adds more text using StringBuilder.

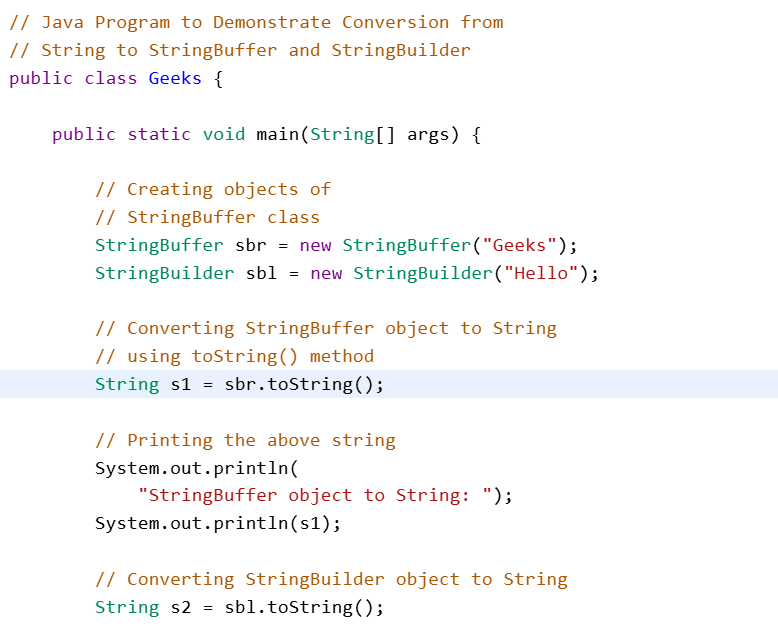
**Case 2: Convert StringBuffer or StringBuilder to String**

This conversion can be performed using [**toString() method**](https://www.geeksforgeeks.org/java/class-tostring-method-in-java-with-examples/#:~:text=The%20toString()%20method%20of,returns%20the%20formed%20string%20representation.) which is overridden in both **StringBuffer and StringBuilder** classes. Below is the Java program to demonstrate the same.

**Note**:

While we use *toString()* method, a new String object(in Heap area) is allocated and initialized to the character sequence currently represented by the StringBuffer object, which means the subsequent changes to the StringBuffer object do not affect the contents of the String object.

**Example:**



A screen shot of a computer code

AI-generated content may be incorrect.

**Output**

StringBuffer object to String:

Geeks

StringBuilder object to String:

Hello

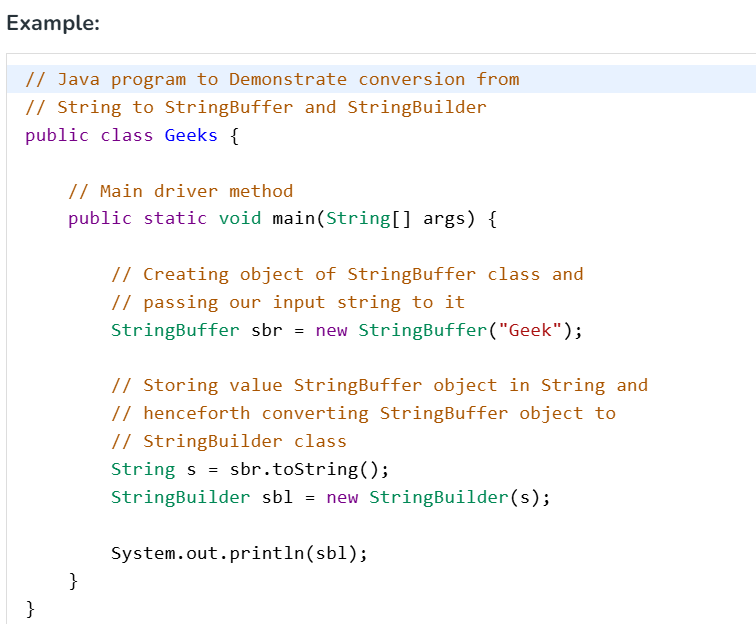
GeeksForGeeks

Geeks

**Explanation:**The above example shows how to convert StringBuffer and StringBuilder objects into String with the help of toString() method.

**Case 3: Convert StringBuffer to StringBuilder or vice-versa**

This conversion is tricky. There is no direct way to convert the same. In this case, We can use a String class object. We first convert the StringBuffer/StringBuilder object to String using ***toString()* method** and then from String to StringBuilder/StringBuffer using constructors.



**Output**

Geek

**Explanation:**The above example shows how to convert a StringBuffer into a String and then converting the same String into a StringBuilder.

**Note:**From the above three use-cases we can conclude out below pointers:

* Objects of String are immutable, and objects of StringBuffer and StringBuilder are mutable.
* StringBuffer and StringBuilder are similar, but StringBuilder is faster and preferred over StringBuffer for the single-threaded program. If thread safety is needed, then StringBuffer is used.

**JAVA\_ Collections**

**Collections in Java**

A collection in Java is a group of individual objects that are treated as a single unit. In Java, a separate framework named the "**Collection Framework"** was defined in JDK 1.2, which contains all the Java Collection Classes and interfaces.

In Java, the Collection interface (**java.util.Collection**) and Map interface (**java.util.Map**) are the two main “root” interfaces of Java collection classes.

**Needed for a Collection Framework**

Before the Collection Framework (before JDK 1.2), Java used Arrays, Vectors, and Hashtables to group objects, but they lacked a common interface. Each had a separate implementation, making usage inconsistent and harder for developers to learn and maintain.

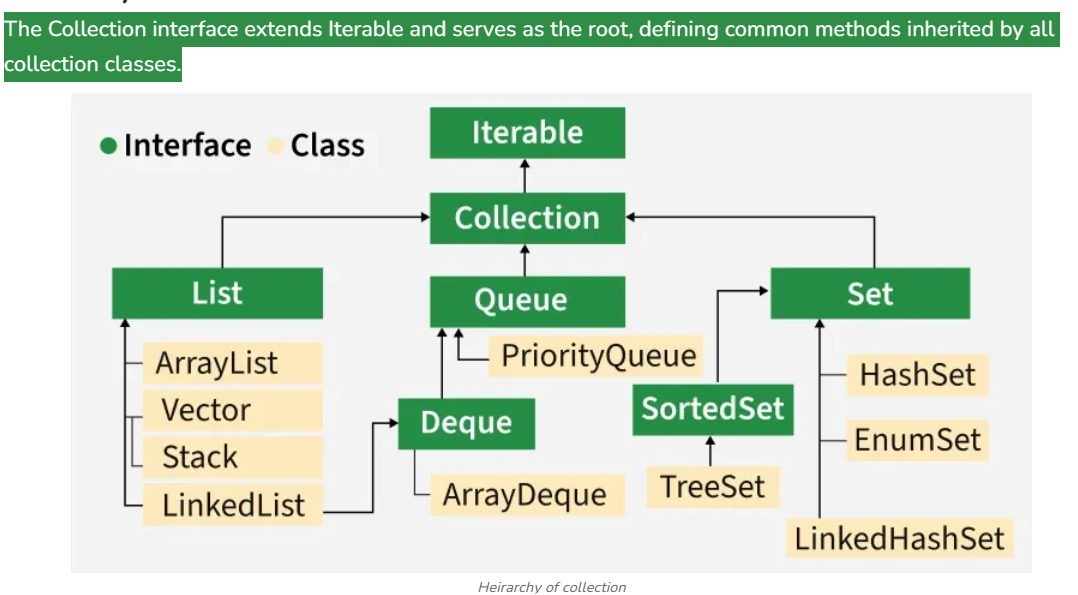
**Advantages of the Java Collection Framework**

Since the lack of a collection framework gave rise to the above set of disadvantages, the following are the advantages of the collection framework.

1. **Consistent API:**Interfaces like List, Set, and Map have common methods across classes (ArrayList, LinkedList, etc.)**.**
2. **Less Coding Effort:** Developers focus on usage, not designing data structures—supports OOP abstraction.
3. **Better Performance:** Offers fast, reliable implementations of data structures, improving speed and quality of code.

**Hierarchy of the Collection Framework in Java**

The Collection interface extends Iterable and serves as the root, defining common methods inherited by all collection classes.



**Linked List Data Structure**

A linked list is a fundamental data structure in computer science. It mainly allows efficient insertion and deletion operations compared to [arrays](https://www.geeksforgeeks.org/dsa/introduction-to-arrays-data-structure-and-algorithm-tutorials/). Like arrays, it is also used to implement other data structures like stack, queue and deque. Here’s the comparison of Linked List vs Arrays

A diagram of a computer algorithm

AI-generated content may be incorrect.

Linked List:

* Data Structure: Non-contiguous
* Memory Allocation: Typically allocated one by one to individual elements
* Insertion/Deletion: Efficient
* Access: Sequential

Array:

* Data Structure: Contiguous
* Memory Allocation: Typically allocated to the whole array
* Insertion/Deletion: Inefficient
* Access: Random

**Stack Data Structure**

A **Stack** is a linear data structure that follows a particular order in which the operations are performed. The order may be **LIFO(Last In First Out)** or **FILO(First In Last Out)**. **LIFO** implies that the element that is inserted last, comes out first and **FILO** implies that the element that is inserted first, comes out last.

It behaves like a stack of plates, where the last plate added is the first one to be removed. **Think of it this way:**

* Pushing an element onto the stack is like adding a new plate on top.
* Popping an element removes the top plate from the stack.

A screenshot of a computer

AI-generated content may be incorrect.

**Queue Data Structure**

A **Queue Data Structure**is a fundamental concept in computer science used for storing and managing data in a specific order.

* It follows the principle of "**First in, First out**" **(FIFO)**, where the first element added to the queue is the first one to be removed.
* It is used as a buffer in computer systems where we have speed mismatch between two devices that communicate with each other. For example, CPU and keyboard and two devices in a network
* Queue is also used in Operating System algorithms like CPU Scheduling and Memory Management, and many standard algorithms like Breadth First Search of Graph, Level Order Traversal of a Tree.

A screenshot of a computer

AI-generated content may be incorrect.

**Java OOP(Object Oriented Programming) Concepts**

Before Object-Oriented Programming (OOPs), most programs used a procedural approach, where the focus was on writing step-by-step functions. This made it harder to manage and reuse code in large applications.

To overcome these limitations, Object-Oriented Programming was introduced. Java is built around OOPs, which helps in organizing code using classes and objects.

* **Key Features of OOP in Java:**
* Structures code into logical units (classes and objects)
* Keeps related data and methods together (encapsulation)
* Makes code modular, reusable and scalable
* Prevents unauthorized access to data
* Follows the DRY (Don’t Repeat Yourself) principle
* **Characteristics of an OOP(Object Oriented Programming)**

The diagram below demonstrates the Java OOPs Concepts



**1. Class**

A [Class](https://www.geeksforgeeks.org/java/classes-objects-java/)is auser-defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. Using classes, you can create multiple objects with the same behavior instead of writing their code multiple times. In general, class declarations can include these components in order:

* **Modifiers**: A class can be public or have default access (Refer to [this](https://www.geeksforgeeks.org/java/access-modifiers-for-classes-or-interfaces-in-java/) for details).
* **Class name:** The class name should begin with the initial letter capitalized by convention.
* **Body:** The class body is surrounded by braces, { }.

**2. Object**

An [Object](https://www.geeksforgeeks.org/java/object-class-in-java/)is a basic unit of Object-Oriented Programming that represents real-life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. The objects are what perform your code, they are the part of your code visible to the viewer/user. An object mainly consists of:

* **State**: It is represented by the attributes of an object. It also reflects the properties of an object.
* **Behavior**: It is represented by the methods of an object. It also reflects the response of an object to other objects.
* **Identity**: It is a unique name given to an object that enables it to interact with other objects.
* [**Method**](https://www.geeksforgeeks.org/java/methods-in-java/)**:** A method is a collection of statements that perform some specific task and return the result to the caller.

**Example:**

A screenshot of a computer program

AI-generated content may be incorrect.**Output**

Employee: Geek

Salary: 10000.0

**3. Abstraction**

**Abstraction in Java** is the process of hiding the implementation details and only showing the essential details or features to the user. It allows to focus on what an object does rather than how it does it. The unnecessary details are not displayed to the user.

**Note:** In Java, abstraction is achieved by [interfaces](https://www.geeksforgeeks.org/java/interfaces-in-java/) and [abstract classes](https://www.geeksforgeeks.org/java/abstract-classes-in-java/). We can achieve 100% abstraction using interfaces.

**Example:**

A screenshot of a computer program

AI-generated content may be incorrect.

**A white background with black text

AI-generated content may be incorrect.**

**4. Encapsulation**

Encapsulation is defined as the process of wrapping data and the methods into a single unit, typically a class. It is the mechanism that binds together the code and the data. It manipulates. Another way to think about encapsulation is that it is a protective shield that prevents the data from being accessed by the code outside this shield.

* Technically, in encapsulation, the variables or the data in a class is hidden from any other class and can be accessed only through any member function of the class in which they are declared.
* In encapsulation, the data in a class is hidden from other classes, which is similar to what **data-hiding** does. So, the terms "encapsulation" and "data-hiding" are used interchangeably.
* Encapsulation can be achieved by declaring all the variables in a class as private and writing public methods in the class to set and get the values of the variables.

A diagram of a method and data

AI-generated content may be incorrect.

Encapsulation

**Example:**

A screen shot of a computer code

AI-generated content may be incorrect.

A computer code with text

AI-generated content may be incorrect.  
**Output**

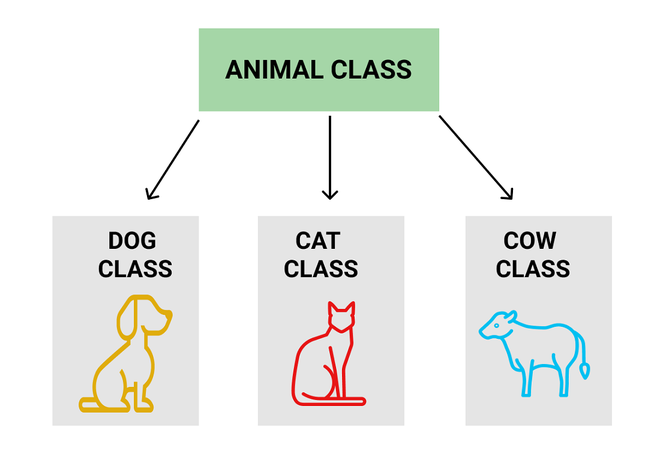
Employee ID: 101

Employee Name: Geek

**5. Inheritance**

Inheritance is an important pillar of OOP (Object Oriented Programming). It is the mechanism in Java by which one class is allowed to inherit the features (fields and methods) of another class. We are achieving inheritance by using **extends** keyword. Inheritance is also known as "**is-a**" relationship.

**Example**: Dog, Cat, Cow can be Derived Class of Animal Base Class.



Inheritance

Let us discuss some frequently used important terminologies:

* **Superclass:**The class whose features are inherited is known as superclass (also known as base or parent class).
* **Subclass:** The class that inherits the other class is known as subclass (also known as derived or extended or child class). The subclass can add its own fields and methods in addition to the superclass fields and methods.
* **Reusability:**Inheritance supports the concept of "reusability", i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class. By doing this, we are reusing the fields and methods of the existing class.

A screenshot of a computer program

AI-generated content may be incorrect.  
**Output**

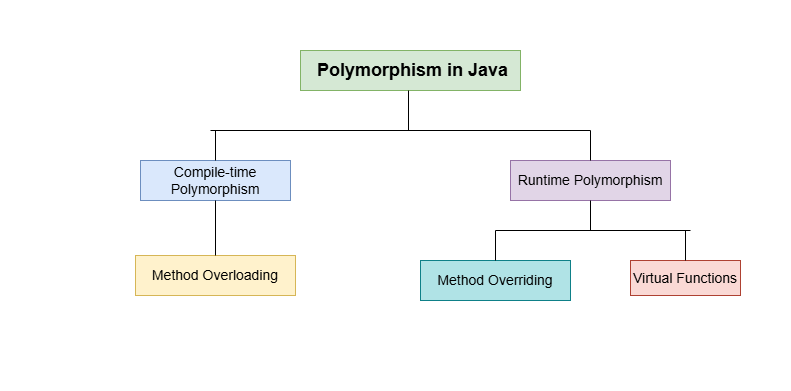
Animal is eating...

Animal is sleeping...

Dog is barking!

**6. Polymorphism**

The word [**polymorphism**](https://www.geeksforgeeks.org/java/polymorphism-in-java/)means having **many forms**, and it comes from the Greek words **poly (many)** and **morph (forms)**, this means one entity can take many forms. In Java, polymorphism allows the same method or object to behave differently based on the context, specially on the project's actual runtime class.



**Types of Polymorphism**

Polymorphism in Java is mainly of 2 types as mentioned below:

1. [Method Overloading](https://www.geeksforgeeks.org/java/method-overloading-in-java/)
2. [Method Overriding](https://www.geeksforgeeks.org/java/overriding-in-java/)

* **Method Overloading and Method Overriding**

**1. Method Overloading:** Also, known as **compile-time polymorphism**, is the concept of Polymorphism where more than one method share the same name with different signature(Parameters) in a class. The return type of these methods can or cannot be same.

**2. Method Overriding:**Also, known as run-time polymorphism,is the concept of Polymorphism where method in the child class has the same name, return-type and parameters as in parent class. The child class provides the implementation in the method already written.

**Below is the implementation of both the concepts:**

// Parent Class



**A screenshot of a computer program

AI-generated content may be incorrect.**

**Output**

Parent.func()

Parent.func(int): 10

Child.func(int): 20

Child.func(int): 30

**Advantage of OOPs over Procedure-Oriented Programming Language**

Object-oriented programming (OOP) offers several key advantages over procedural programming:

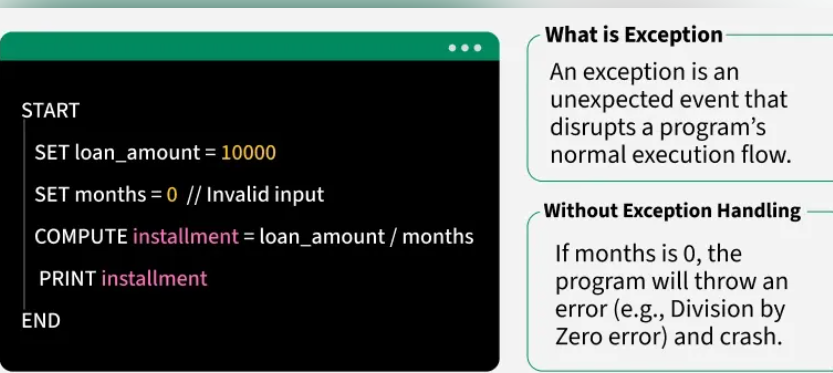
* By using objects and classes, you can create reusable components, leading to less duplication and more efficient development.
* It provides a clear and logical structure, making the code easier to understand, maintain, and debug.
* OOP supports the DRY (Don't Repeat Yourself) principle.This principle encourages minimizing code repetition, leading to cleaner, more maintainable code. Common functionalities are placed in a single location and reused, reducing redundancy.
* By reusing existing code and creating modular components, OOP allows for quicker and more efficient application development

**Disadvantages of OOPs**

* OOP has concepts like classes, objects, inheritance etc. For beginners, this can be confusing and takes time to learn.
* If we write a small program, using OOP can feel too heavy. We might have to write more code than needed just to follow the OOP structure.
* The code is divided into different classes and layers, so in this, finding and fixing bugs can sometimes take more time.
* OOP creates a lot of objects, so it can use more memory compared to simple programs written in a procedural way.

**Java Exception Handling**

Exception handling in Java is an effective mechanism for managing runtime errors to ensure the application's regular flow is maintained. Some Common examples of exceptions include ClassNotFoundException, IOException, SQLException, RemoteException, etc. By handling these exceptions, Java enables developers to create robust and fault-tolerant applications.



A black and white screen with white text

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**Example: Showing an arithmetic exception or we can say a divide by zero exception.**

import java.io.\*;

​

A screenshot of a computer

AI-generated content may be incorrect.

**Output:**



***Note:*** *When an exception occurs and is not handled, the program terminates abruptly and the code after it, will never execute.*

**Example**: The below Java program modifies the previous example to handle an ArithmeticException using try-catch and finally blocks and keeps the program running.

import java.io.\*;

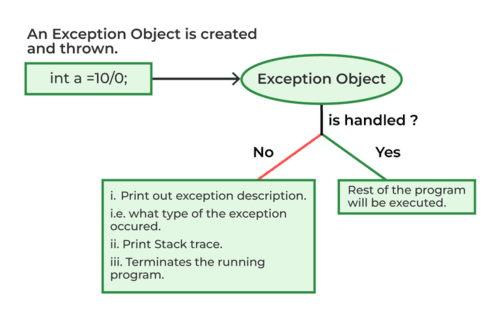
​

A screenshot of a computer program

AI-generated content may be incorrect.  
**Output**

Error: Division by zero is not allowed!

Program continues after handling the exception.

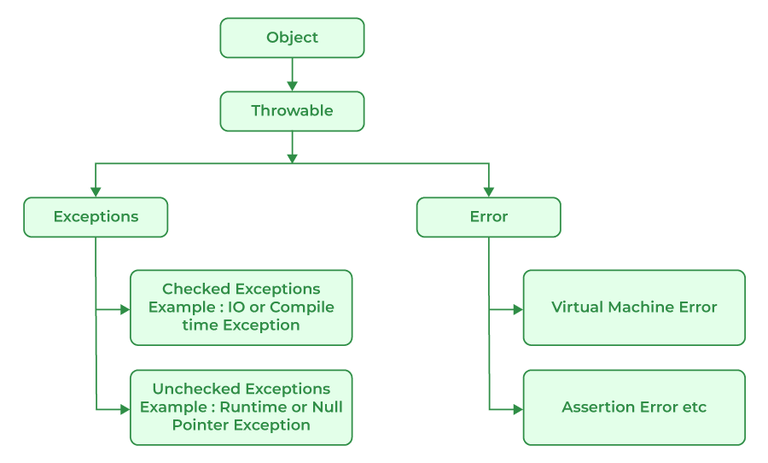


**Java Exception Hierarchy**

In Java, all exceptions and errors are subclasses of the Throwable class. It has two main branches

1. Exception.
2. Error

The below figure demonstrates the exception hierarchy in Java:



Heirarchy of exception

**Major Reasons Why an Exception Occurs**

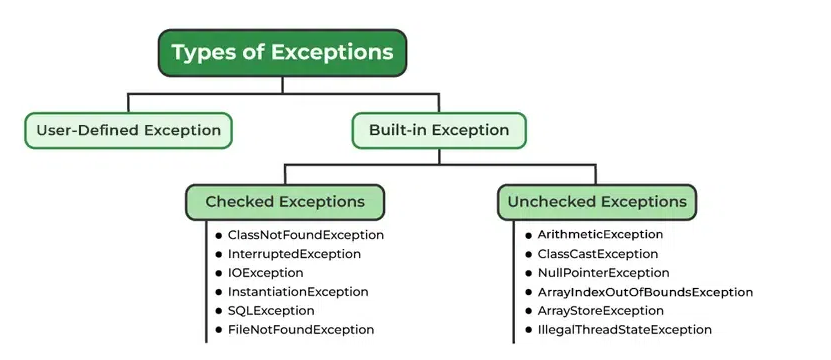
Exceptions can occur due to several reasons, such as:

* Invalid user input
* Device failure
* Loss of network connection
* Physical limitations (out-of-disk memory)
* Code errors
* Out of bound
* Null reference
* Type mismatch
* Opening an unavailable file
* Database errors
* Arithmetic errors

Errors are usually beyond the control of the programmer and we should not try to handle errors.

**Types of Java Exceptions**

Java defines several types of exceptions that relate to its various class libraries. Java also allows users to define their it's exceptions.



Exception

**Exceptions can be categorized in two ways:**

1. Built-in Exceptions

* Checked Exception
* Unchecked Exception

2. user-defined Exceptions

**1. Built-in Exception**

Build-in Exception are pre-defined exception classes provided by Java to handle common errors during program execution. There are two type of built-in exception in java.

* **Checked Exceptions**

Checked exceptions are called compile-time exceptions because these exceptions are checked at compile-time by the compiler. Examples of Checked Exception are listed below:

* **ClassNotFoundException:** Throws when the program tries to load a class at runtime but the class is not found because it's belong not present in the correct location or it is missing from the project.
* **InterruptedException:** Thrown when a thread is paused and another thread interrupts it.
* **IOException:** Throws when input/output operation fails.
* **InstantiationException:**Thrown when the program tries to create an object of a class but fails because the class is abstract, an interface or has no default constructor.
* **SQLException:** Throws when there is an error with the database.
* **FileNotFoundException**: Thrown when the program tries to open a file that does not exist.
* **Unchecked Exceptions**

The unchecked exceptions are just opposite to the checked exceptions. The compiler will not check these exceptions at compile time. In simple words, if a program throws an unchecked exception and even if we did not handle or declare it, the program would not give a compilation error. Examples of Unchecked Exception are listed below:

* **ArithmeticException**: It is thrown when there is an illegal math operation.
* **ClassCastException:** It is thrown when we try to cast an object to a class it does not belong to.
* **NullPointerException:** It is thrown when we try to use a null object (e.g. accessing its methods or fields).
* **ArrayIndexOutOfBoundsException:** This occurs when we try to access an array element with an invalid index.
* **ArrayStoreException:** This happens when we store an object of the wrong type in an array.
* **IllegalThreadStateException:** It is thrown when a thread operation is not allowed in its current state.

**2.User-Defined Exception**

Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, users can also create exceptions, which are called "user-defined Exceptions".

* **Methods to Print the Exception Information**

1. [**printStackTrace()**](https://www.geeksforgeeks.org/java/throwable-printstacktrace-method-in-java-with-examples/)**:** Prints the full stack trace of the exception, including the name, message and location of the error.
2. [**toString()**](https://www.geeksforgeeks.org/java/throwable-tostring-method-in-java-with-examples/)**:**Prints exception information in the format of the Name of the exception.
3. [**getMessage()**](https://www.geeksforgeeks.org/java/throwable-getmessage-method-in-java-with-examples/)**:**Prints the description of the exception

**Try-Catch Block**

A try-catch block in Java is a mechanism to handle exception. The try block contains code that might thrown an exception and the catch block is used to handle the exceptions if it occurs.

**Internal working of try-catch Block**

* Java Virtual Machine starts executing the code inside the try block.
* If an exception occurs, the remaining code in the try block is skipped and the JVM starts looking for the matching catch block.
* If a matching catch block is found, the code in that block is executed.
* After the catch block, control moves to the finally block (if present).
* If no matching catch block is found the exception is passed to the JVM default exception handler.
* The final block is executed after the try catch block. regardless of whether an exception occurs or not.

**A close-up of a computer code

AI-generated content may be incorrect.**

**Nested try-catch**

In Java, you can place one try-catch block inside another to handle exceptions at multiple levels.

public class NestedTryExample {

A screen shot of a computer program

AI-generated content may be incorrect.

**finally Block**

The finally block is used to execute important code regardless of whether an exception occurs or not.

**Note**: finally block is always executes after the try-catch block. It is also used for resource cleanup.

**A computer code with text

AI-generated content may be incorrect.**

**Handling Multiple Exception**

We can handle multiple type of exceptions in Java by using multiple catch blocks, each catching a different type of exception.

**A screen shot of a computer code

AI-generated content may be incorrect.**

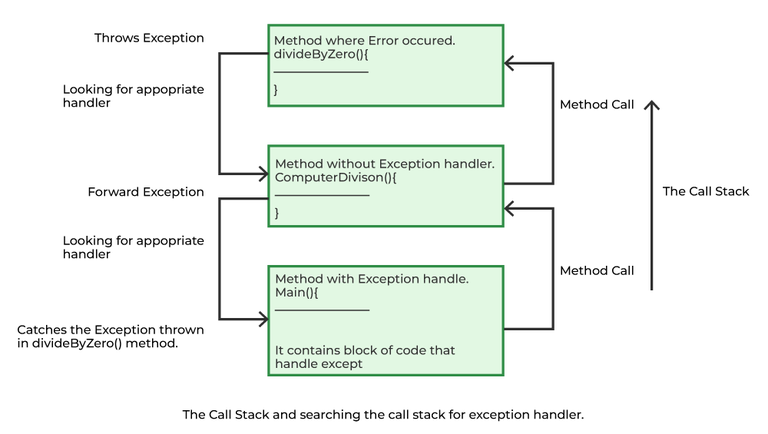
**How Does JVM Handle an Exception?**

When an Exception occurs, the JVM creates an exception object containing the error name, description and program state. Creating the exception object and handling it in the run-time system is called throwing an exception. There might be a list of the methods that had been called to get to the method where an exception occurred. This ordered list of methods is called call stack. Now the following procedure will happen:

* The run-time system searches the call stack for an exception handler
* It starts searching from the method where the exception occurred and proceeds backward through the call stack.
* If a handler is found, the exception is passed to it.
* If no handler is found, the default exception handler terminates the program and prints the stack trace.

**

Look at the below diagram to understand the flow of the call stack:

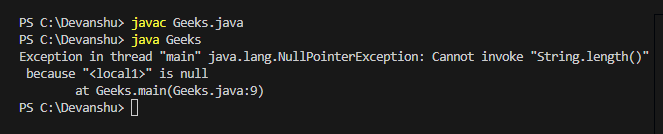
Exception flow

**Illustration**:

**A computer code with text

AI-generated content may be incorrect.**

**Output:**



Let us see an example that illustrates how a run-time system searches for appropriate exception handling code on the call stack.

**Example:**

A screenshot of a computer program

AI-generated content may be incorrect.

A computer code with text

AI-generated content may be incorrect.  
**Output**

A black and white text

AI-generated content may be incorrect.

**How Programmer Handle an Exception?**

Java exception handling uses five keywords such as try, catch, throw and throws and finally.

* Code that might cause an exception goes in the try block.
* If an exception occurs, it is caught using catch.
* We can throw exceptions manually with throw and methods must declare exceptions they can throw using throws.
* The finally block is used for code that must run after try, whether an exception occurs or not.

***Tip****: One must go through* [*control flow in try catch finally*](https://www.geeksforgeeks.org/java/flow-control-in-try-catch-finally-in-java/) *block for better understanding.*

* **Need for try-catch clause (Customized Exception Handling)**

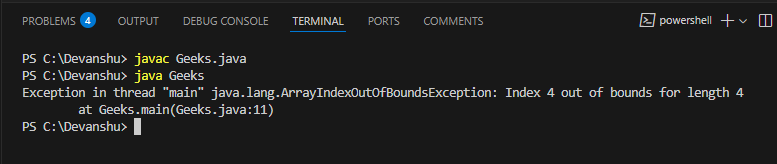
Consider the below program in order to get a better understanding of the try-catch clause.

**Example: Java Program to Demonstrate Need of try-catch Clause**

**A computer code on a white background

AI-generated content may be incorrect.**

**Output:**

output

**Advantages of Exception Handling**

* Provision to complete program execution.
* Easy identification of program code and error-handling code.
* Propagation of errors.
* Meaningful error reporting.
* Identifying error types.

**Difference Between Exception and Error**

|  |  |
| --- | --- |
| **Error** | **Exception** |
| An Error indicates a serious problem that a reasonable application should not try to catch. | Exception indicates conditions that a reasonable application might try to catch |
| This is caused by issues with the JVM or hardware. | This is caused by conditions in the program such as invalid input or logic errors. |
| **Examples**: OutOfMemoryError, StackOverFlowError | **Examples**: IOException, NullPointerException |