Exceptions

27 September 2024 19:37

What is Exception in Java ?

- *An **exception** in Java is an event that disrupts the normal flow of the program's execution. Exceptions can arise from various issues, such as invalid user input, file not found, network issues, or division by zero.*
- *If we doesn't handle the exception JVM handles it but if JVM handles the exception it terminate the program abnormally*

Types of Exception:

- ***Checked Exception:***
  - *These are exceptions that are **checked at compile-time**. The compiler ensures that these exceptions are either caught or declared in the method signature using the throws keyword.*
  - ***Examples include:***
    - ***IOException**: Raised when an input or output operation fails or is interrupted.*
    - ***SQLException**: Raised when there is an error with database access.*
    - ***ClassNotFoundException**: Raised when the Java Virtual Machine (JVM) cannot find a specified class.*

- ***Unchecked Exception:***
  - *These exceptions are **not checked at compile-time**. They occur during runtime and can be avoided with proper coding practices*
  - ***Examples include:***
    - ***NullPointerException**: Raised when an application attempts to use null where an object is required.*
    - ***ArrayIndexOutOfBoundsException**: Raised when an array has been accessed with an illegal index.*
    - ***ArithmeticException**: Raised when an illegal arithmetic operation occurs, such as division by zero.*

- ***Common Selenium Exceptions:** NoSuchElementException, TimeoutException, StaleElementReferenceException, ElementNotInteractableException, and WebDriverException*

Exception Handling Mechanism in Java

- *The exception handling mechanism in Java enables developers to manage errors gracefully. By using try, catch, and finally blocks, exceptions can be caught and handled appropriately, ensuring that the application continues to run smoothly. This leads to more robust and user-friendly applications by providing meaningful error responses instead of abrupt crashes.*

- ***Try-Catch Block:***
    - ***Try Block:*** *Code that might throw an exception is placed inside a try block. If an exception occurs, the flow of control is transferred to the catch or finally block.*
    - ***Catch Block****:*
        - *This block catches the exception thrown by the try block. Developers can specify what to do when an exception is encountered (e.g., log the error, show an error message, etc.).*
        - *This is a optional block*
        - *We can have multiple catch block also but only one will executed*
    - ***Finally block***: *Code that will execute regardless of whether an exception was thrown or caught. This is useful for releasing resources, such as closing files or database connections.*

- ***Throwing Exceptions:***
    - *Developers can explicitly throw exceptions using the throw statement. This is often done to indicate that a certain condition has been met (e.g., invalid input).* ***Example*** *thrownewIllegalArgumentException("Age must be 18 or older.");*

- ***Propagating Exceptions***
    - *Exceptions can be propagated up the call stack using the throws keyword. This allows a method to indicate that it may throw an exception, which can then be handled by calling methods.*

- ***Custom Exceptions:***
    - *Developers can create their own exception classes by extending the Exception class. This allows for more specific error handling tailored to the application's needs.*

- ***Benefits of Exception Handling:***
    - *Prevents Program Crashes*
    - *Improves Code Clarity*
    - *Customizable Responses*
    - *Resource Management*