**Exceptions:** exception is an error / abnormal condition that arises in the code. They can be generated by Java runtime system or we can manually generate by our code. Java exception handling has five keywords: try, catch, throw, throws and finally.

* **Try:** if you want to monitor certain lines of code that might throw exception they are contained within a try block. If an exception occur it is thrown.
* **Catch:** the exception that is thrown by the try block can be handled by catch block in some rational manner.
* **Throw:** any system generated exceptions can be thrown by Java runtime systems however we can manually throw an exception using throw keyword
* **Throws:** an exception that is thrown out of a method must be specified with throws keyword.
* **Finally:** you want certain pieces of code to be executed no matter what needs to be put in a finally block. Example database connection.

Most of the cases exceptions are caught in try and catch blocks, the finally is used in very few cases. For example, You wanted to access files from the database so you write the code to access or open files from database in try block whatever the exceptions that happened while accessing the database can be caught in catch block and you always wanted to make sure you close the connection to database so we close the connection in finally block.

**Checked versus unchecked exceptions**:

* Checked exceptions are exceptions that are checked at compile time by the compiler.
* They represent exceptional conditions that a well-behaved program should anticipate and handle.
* Checked exceptions must be either caught (handled) using a try-catch block or declared in the method signature using the throws keyword.

Examples of checked exceptions include IOException, FileNotFoundException, SQLException, etc.

**A diagram of an error

Description automatically generated**

A close-up of a computer error

Description automatically generated

* Unchecked exceptions are also known as runtime exceptions. These are usually caused by programming or logical errors in the code.

Examples of unchecked exceptions include NullPointerException, ArrayIndexOutOfBoundsException, ArithmeticException, etc.

* For checked exceptions (identified by the system), we either must deal with them using try catch block or we can directly use throws keyword in the method signature. Now the program doesn't show an error and lets us execute. However, since we are not catching those exceptions anywhere in the program it is thrown at runtime.
* For unchecked exceptions, Java LEXIS to execute the program however it explodes the compiler and throws us with the type of exception that the code has, we can also with it by using try catch block.

Creating own exception subclasses:

* use extends keyword and create a subclass of exception class you don't need to implement anything in the subclass, just their existence will allow you to use them as an exception

**Chained Exceptions:** whenever an exception occurs you can catch that exception and wrap it in a new exception object, this allows us to give more information about the circumstances leading to the error and also preserves the stack trace and provide additional context. Like this we can propagate the exception to higher level methods and handle it at appropriate level.

A computer screen shot of a program

Description automatically generated

* The runtime exception occurred in method one gives a high level detail
* it is actually occurred by arithmetic exception in method 2.
* here the stack trace provides the sequence of method calls and also the cause of the error that was shown in method 2 which was called by method 1 and which was called by mainA screen shot of a computer program

  Description automatically generated

**Java 7 Features**

* **Multi-Catch Feature:** instead of using catch block to handle each exception separately, we can separate each exception type with OR operator in the catch block catch(ArithmeticException | ArrayIndexOutOfBoundsException e) {
* **Final rethrow:** catching an exception and re throwing the exception for the caller to handle is precise final rethrow

**A screen shot of a computer program

Description automatically generated**

* The execution starts from the main method it calls for the divide method which attempts to perform division with the divisor 0
* inside the divide method we catch the edit arithmatic exception handle that exception using a print statement and then rethrow the exception using “throw e”.
* The exception now is caught by the main method catch block.
* In the output we see the error messages generated at both the levels of exception handling using rethrow exception.

A screenshot of a computer program

Description automatically generated

**Try with resources:** it is a Java 7 feature, which simplifies resource management. It ensures that each resource is closed automatically after the try block finishes execution whether normally or because of an exception.

In a traditional try-catch block, you are responsible for explicitly closing any resources that you open within the try block. This typically involves placing the close statements in the finally block.

* In contrast, try-with-resources simplifies resource management by automatically closing the resources that are opened within the parentheses of the try statement. You do not need to write explicit finally blocks to close resources.

A screen shot of a computer

Description automatically generatedA close up of text

Description automatically generated

In a try-with-resources statement, resources are closed in the reverse order of their creation or acquisition. This ensures that resources opened later will be closed first, which is often important for proper resource cleanup and avoiding potential resource leaks.

**Order of Resource Closure:**

With try-with-resources, resources are closed in the reverse order of their creation/acquisition. This ensures that resources opened later are closed first, which can be important for proper resource cleanup and avoiding potential resource leaks.

In traditional try-catch-finally blocks, you have explicit control over the order of resource closure, but you need to ensure proper ordering manually.

**Q/A:**

What is an exception in Java?

* An exception in Java is an event that disrupts the normal flow of a program's instructions during execution, typically caused by errors in code or unexpected conditions.

What are the types of exceptions in Java?

* Exceptions in Java are broadly categorized into two types: Checked exceptions and Unchecked exceptions. Checked exceptions are those that are checked at compile time, while Unchecked exceptions are checked at runtime.

What is the difference between checked and unchecked exceptions?

* Checked exceptions must be declared in a method's signature or handled using try-catch blocks, whereas unchecked exceptions do not need to be declared or handled explicitly.

A diagram of a computer error

Description automatically generated with medium confidence

What is the purpose of the try, catch, and finally blocks in exception handling?

* The try block is used to enclose the code that might throw an exception. The catch block is used to handle the exception, and the finally block is used to execute code that should always run, regardless of whether an exception is thrown or not.

Can you have multiple catch blocks for a single try block?

* Yes, you can have multiple catch blocks for a single try block, each handling a different type of exception. They are checked in the order they appear, and only the first matching catch block is executed.

What is the purpose of the throws keyword in Java?

* The throws keyword is used in a method declaration to indicate that the method may throw certain types of exceptions. It is used to delegate the responsibility of handling exceptions to the caller of the method.

What is the finally block used for, and is it always executed?

* The finally block is used to execute code that should always run, regardless of whether an exception is thrown or not. It is guaranteed to execute even if an exception occurs in the try block or a matching catch block.

What happens if an exception is not caught?

* If an exception is not caught, it propagates up the call stack until it is caught by an appropriate catch block or until it reaches the top-level exception handler, causing the program to terminate.

Can you throw an exception explicitly in Java?

* Yes, you can throw an exception explicitly using the throw keyword followed by an instance of the desired exception class.

How do you create custom exceptions in Java?

* Custom exceptions can be created by extending the Exception class or one of its subclasses. This allows you to define your own exception types with specific behaviors and properties.

What is an error?

Error can be defined as an issue in the program like infinite recursions happening, memory leakage or libraries incompatibility, all these can be defined as errors