Exception Handling

Exception Handling

- Exception is an error event that can happen during the execution of a program and disrupts its normal flow.
- Java provides a robust and object oriented way to handle exception scenarios, known as **Java Exception Handling**.
- Exception can arise from different kind of situations such as wrong data entered by user, hardware failure, network connection failure, Database server down etc.
- Java being an object oriented programming language, whenever an error occurs while executing a statement, creates an exception object and then the normal flow of the program halts and JRE tries to find someone that can handle the raised exception.
- The exception object contains a lot of debugging information such as method hierarchy, line number where the exception occurred, type of exception etc.

Exception Handling

- When the exception occurs in a method, the process of creating the exception object and handing it over to runtime environment is called “**throwing the exception**”.
- Once runtime receives the exception object, it tries to find the handler for the exception.
- Exception Handler is the block of code that can process the exception object.
- The logic to find the exception handler is simple – starting the search in the method where error occurred, if no appropriate handler found, then move to the caller method and so on.
- If methods call stack is **A->B->C** and exception is raised in method C, then the search for appropriate handler will move from **C->B->A**.

Exception Handling

- If appropriate exception handler is found, exception object is passed to the handler to process it. The handler is said to be “**catching the exception**”.
- If there are no appropriate exception handler found then program terminates printing information about the exception.
- Note that Java Exception handling is a framework that is used to handle runtime errors only, compile time errors are not handled by exception handling framework.

keywords

- **throw** – We know that if any exception occurs, an exception object is getting created and then Java runtime starts processing to handle them. Sometime we might want to generate exception explicitly in our code, for example in a user authentication program we should throw exception to client if the password is null. throw keyword is used to throw exception to the runtime to handle it.
- **throws** – When we are throwing any exception in a method and not handling it, then we need to use throws keyword in method signature to let caller program know the exceptions that might be thrown by the method. The caller method might handle these exceptions or propagate it to it's caller method using throws keyword. We can provide multiple exceptions in the throws clause and it can be used with main() method also.

keywords

- **try-catch** - We use try-catch block for exception handling in our code. try is the start of the block and catch is at the end of try block to handle the exceptions. We can have multiple catch blocks with a try and try-catch block can be nested also. catch block requires a parameter that should be of type Exception.
- **finally** - finally block is optional and can be used only with try-catch block. Since exception halts the process of execution, we might have some resources open that will not get closed, so we can use finally block. finally block gets executed always, whether exception occurred or not.

try-with-resources

- One of the Java 7 features is **try-with-resources** statement for automatic resource management.
- A resource is an object that must be closed once your program is done using it, like a File resource or JDBC resource for database connection or a Socket connection resource.
- Before Java 7, there was no **auto resource management** and we should explicitly close the resource once our work is done with it.
- Usually, it was done in the **finally block** of a try-catch statement.
- This approach used to cause memory leaks and performance hit when we forgot to close the resource.

try-with-resources

Before Java 7:

```
try{
    //open resources like File, Database connection, Sockets etc
} catch (FileNotFoundException e) {
    // Exception handling like FileNotFoundException, IOException etc
}finally{
    // close resources
}
```

Java 7 implementation:

```
try(// open resources here){
    // use resources
} catch (FileNotFoundException e) {
    // exception handling
}
// resources are closed as soon as try-catch block is executed.
```

try-with-resources - benefits

- More readable code and easy to write.
- Automatic resource management.
- Number of lines of code is reduced.
- No need of finally block just to close the resources.
- We can open multiple resources in try-with-resources statement separated by a semicolon.
- When multiple resources are opened in try-with-resources, it closes them in the reverse order to avoid any dependency issue.
- Java 7 has introduced a new interface **java.lang.AutoCloseable** which is extended by java.io.Closeable interface.
- To use any resource in try-with-resources, it must implement **AutoCloseable** interface else java compiler will throw compilation error.