Set 25: Exceptions

Skill 25.01: Explain what an exception is in JAVA

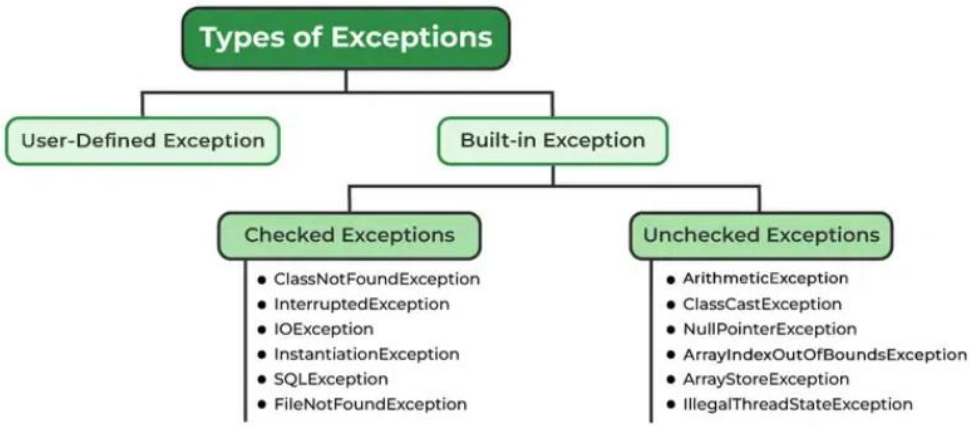
Skill 25.01 Concepts

All exception and error types are subclasses of the class *Throwable*, which is the base class of the hierarchy. One branch of the *Throwable* class is headed by *Exception* class. This class is used for exceptional conditions that user programs should catch. *NullPointerException* is an example of such an exception. Another branch, *Error* is used by the Java run-time system(JVM) to indicate errors having to do with the run-time environment itself(JRE). *StackOverflowError* is an example of such an error. Errors are usually beyond the control of the programmer, and we should not try to handle errors.

The subclasses of the Throwable class and their purposes are summarized below,

Aspect	Error	Exception
Definition	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
Cause	Caused by issues with the JVM or hardware.	Caused by conditions in the program such as invalid input or logic errors.
Examples	OutOfMemoryError StackOverFlowError	IOException NullPointerException

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



Exceptions can be categorized in two ways:

- Built-in Exceptions
    - Checked Exception
    - Unchecked Exception
- User-Defined Exceptions

Built-in Exception

Build-in Exception are pre-defined exception classes provided by Java to handle common errors during program execution.

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:

1. ClassNotFoundException: Throws when the program tries to load a class at runtime but the class is not found because its not present in the correct location or it is missing from the project.
2. InterruptedException: Thrown when a thread is paused and another thread interrupts it.
3. IOException: Throws when input/output operation fails
4. 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.
5. SQLException: Throws when there's an error with the database.
6. FileNotFoundException: Thrown when the program tries to open a file that doesn't 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 didn't handle or declare it, the program would not give a compilation error. Examples of Unchecked Exception are listed below:

1. ArithmeticException: It is thrown when there's an illegal math operation.
2. ClassCastException: It is thrown when you try to cast an object to a class it does not belongs to.
3. NullPointerException: It is thrown when you try to use a null object (e.g. accessing its methods or fields)
4. ArrayIndexOutOfBoundsException: It occurs when we try to access an array element with an invalid index.
5. ArrayStoreException: It happens when you store an object of the wrong type in an array.

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.

Skill 25.01: Exercises 1 & 2

Skill 25.02: Write a try-catch statement to catch an exception

Skill 25.02 Concepts

An exception (or exceptional event) is a problem that arises during the execution of a program. When an Exception occurs, the normal flow of the program is disrupted and the program/Application terminates abnormally, which is not recommended, **therefore exceptions should be handled by the programmer**.

An exception can occur for many different reasons. The following are some scenarios where an exception occurs.

- A user has entered an invalid data.
- A file that needs to be opened cannot be found.
- An array that needs to be accessed is null
- A user tries to access an index in a string that does not exist

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

```
try {

   // Code that may throw an exception

} catch (ExceptionType e) {

   // Code to handle the exception


}
```

Below are examples of how to catch a checked and unchecked exception,

Checked

```
File f = new File("file.txt");

try {
       Scanner s = new Scanner(f);

} catch (FileNotFoundException e) {

       System.out.println("file not found");
}
```

Unchecked

```
try{

      System.out.println(10/0);

}catch(ArithmeticException e){

      System.out.println("cannot divide by zero");
}
```

In the examples above, we created custom messages to print our catch statements. JAVA also provides the following built-in messages,

Method	Description
printStackTrace()	Prints the full stack trace of the exception, including the name, message, and location of the error.
toString()	Prints exception information in the format of the Name of the exception.
getMessage()	Prints the description of the exception.

Other examples of unchecked exceptions include

- NullPointerException
- ArrayIndexOutOfBoundsException
- IndexOutOfBoundsException
- IllegalArgumentException
- ArithmeticException
- ClassCastException

Skill 25.02: Exercise 1

Skill 25.03: Combine try-catch statements

Skill 25.03 Concepts

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

```
try {
   // Code that may throw an exception
} catch (ArithmeticException e) {
   // Code to handle the exception
} catch(ArrayIndexOutOfBoundsException e){
   //Code to handle the anothert exception
}catch(NumberFormatException e){
    //Code to handle the anothert exception

}
```

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

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

```
try {
    // Code that may throw an exception
} catch (ExceptionType e) {
    // Code to handle the exception
}finally{
// cleanup code

}
```

Skill 25.03: Exercise 1