9. Exceptions

Sisoft Technologies Pvt Ltd
SRC E7, Shipra Riviera Bazar, Gyan Khand-3, Indirapuram, Ghaziabad
Website: www.sisoft.in Email:info@sisoft.in
Phone: +91-9999-283-283

Exception Overview

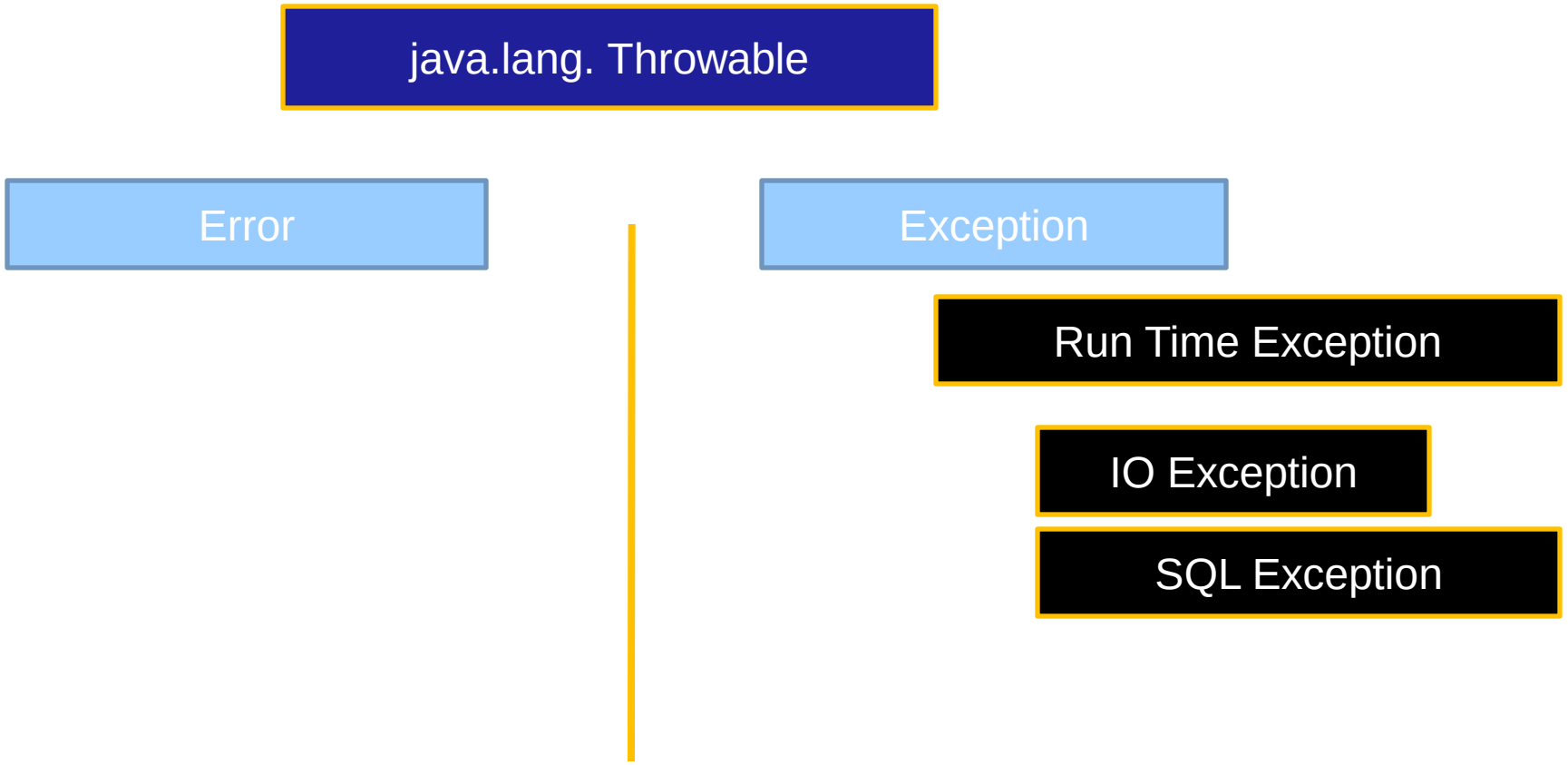
- An *exception is an* abnormal condition that arises in a code sequence at run time. In other words, an exception is a runtime error.
- **Checked exceptions:** A checked exception is an exception that application can anticipate and recover. For example, if a file is to be opened, but the file cannot be found, an exception occurs. These exceptions cannot simply be ignored at the time of compilation
- **Runtime exceptions:** A runtime exception is an exception that are internal to the application, and that the application usually cannot anticipate or recover from. As opposed to checked exceptions, runtime exceptions are ignored at the time of compilation. These usually indicate programming bugs, such as logic errors or improper use of an API

Exception Overview

- **Errors:** Error are used by the Java run-time system to indicate errors having to do with the run-time environment itself. Stack overflow is an example of such an error.
- These problems arise beyond the control of the user or the programmer. Errors are typically ignored in your code because you can rarely do anything about an error. For example, if a stack overflow occurs, an error will arise. They are also ignored at the time of compilation
- Errors and runtime exceptions are collectively known as *unchecked exceptions*.

Exception Class Hierarchy

- All exception types are subclasses of the java.lang.Throwable class. There are two subclasses of Throwable Class: Exception and Error.
- Exception class is used for exception conditions that user programs should handle.



Exception Class Methods

``	**Methods with Description**
1	public String getMessage() Returns a detailed message about the exception that has occurred. This message is initialized in the Throwable constructor.
2	public Throwable getCause() Returns the cause of the exception as represented by a Throwable object.
3	public String toString() Returns the name of the class concatenated with the result of getMessage()
4	public void printStackTrace() Prints the result of toString() along with the stack trace to System.err, the error output stream.
5	public StackTraceElement [] getStackTrace() Returns an array containing each element on the stack trace. The element at index 0 represents the top of the call stack, and the last element in the array represents the method at the bottom of the call stack.
6	public Throwable fillInStackTrace() Fills the stack trace of this Throwable object with the current stack trace, adding to any previous information in the stack trace.

Exception Handling

- Java exception handling is managed via five keywords: **try, catch, finally, throw and throws**
- Program statements that you want to monitor for exceptions are contained within a **try block. If an** exception occurs within the **try block,** it is thrown.
- Your code can catch this exception (using **catch) and handle it** in some rational manner. System-generated exceptions are automatically thrown by the Java runtime system.

Exception Handling

- **Any** code that absolutely must be executed after a **try block completes is put in a finally** block
- To manually throw an exception, use the keyword **throw**
- **Any exception that is** thrown out of a method must be specified as such by a **throws clause.**

The try Block

- The first step in constructing an exception handler is to enclose the code that might throw an exception within a try block
- In general, a try block looks like the following:

```
try {
code
}
catch and finally blocks . . .
```

The catch Block

- The exception handler are associated with **try** block by providing one or more **catch** blocks directly after the try block. No code can be between the end of the try block and the beginning of the first catch block.
- Code within a **try** block is referred to as protected code and code in **catch** block are exception handers

```
try {
 } catch (ExceptionType name) { }
catch (ExceptionType name) { }
```

- Each catch block is an exception handler that handles the type of exception indicated by its argument. The argument type, *ExceptionType*, declares the type of exception that the handler can handle and must be the name of a class that inherits from the Throwable class.

Multiple catch Blocks:

- A try block can be followed by multiple catch blocks. The syntax for multiple catch blocks looks like the following

```
try
{
 //Protected code }
catch(ExceptionType1   e1)
{ //Catch block }
catch(ExceptionType2  e2)
{ //Catch block }
catch(ExceptionType3  e3)
{ //Catch block }
```

The Unhandled Exception

- If an exception is not handled in the current catch blocks, it is thrown to the caller of the method.
- If the exception gets back to the main method and is not handled there, the program is terminated abnormally.

The finally Block

- The finally keyword is used to create a block of code that follows a try block. A finally block of code always executes, whether or not an exception has occurred.
- Using a finally block allows you to run any cleanup-type statements that you want to execute, no matter what happens in the protected code.
- A finally block appears at the end of the catch blocks and has the following syntax:

The finally Block

```
try {
//Protected code }
catch(ExceptionType1 e1) { //Catch block }
catch(ExceptionType2 e2) { //Catch block }
catch(ExceptionType3 e3) { //Catch block
   }finally { //The finally block always
   executes. }
```

The try-with-resources Statement

- The try-with-resources statement is a try statement that declares one or more resources
- A *resource* is an object that must be closed after the program is finished with it. The try-with-resources statement ensures that each resource is closed at the end of the statement
- Any object that implements java.lang.AutoCloseable, which includes all objects which implement java.io.Closeable, can be used as a resource

```
try (BufferedReader br = new BufferedReader(new
    FileReader(path)))
{
return br.readLine();
}
```

Notes try-catch-finally

- A catch clause cannot exist without a try statement.
- It is not compulsory to have finally clauses when ever a try/catch block is present.
- The try block cannot be present without either catch clause or finally clause.
- Any code cannot be present in between the try, catch, finally blocks.

The throw Keyword

- Any code can throw an exception: your code, code from a package written by someone else such as the packages that come with the Java platform, or the Java runtime environment. Regardless of what throws the exception, it's always thrown with the throw statement.
- The throw statement requires a single argument: a throwable object.

  throw *someThrowableObject*;

- Throwable objects are instances of any subclass of the Throwable class.

```
 try
{
----------
        throw new Exception("Not able to initialized");
}
catch (Exception ex)
{
Sop(ex.tostring);
 Logger.getLogger(ExceptionTest.class.getName()).log(Level.SEVERE, null,
    ex);
}
```

The throws Keyword

- If a method does not handle an exception, the method must declare it using the **throws** keyword. The throws keyword appears at the end of a method's signature.

```
public void psvm() throws Exception
{       }
```

Throws more then one exception

- A method can declare that it throws more than one exception, in which case the exceptions are declared in a list separated by commas
- For example, the following method declares that it throws a RemoteException and an InsufficientFundsException:

```
import java.io.*;
 public class className {
 public void withdraw(double amount) throws
   RemoteException, InsufficientFundsException { //
   Method implementation
   } //Remainder of class definition }
```

Declaring you own Exception

- User-defined exceptions are created by extending the **Exception** class
- The extended class contains constructors, data members and methods
- The Exception class does not define any methods of its own. It inherits methods provided by Throwable. These methods are available to user defined exceptions for overriding

Declaring you own Exception

```
Class MyException extends Exception{
 private int detail;

MyException(int a) {
 details = a ;
}

Public toString() {
   return “MyException:”+detail ;
}
}
```

Questions ?