Exception Handling

Shristi Technology Labs

Contents

- Define Exception
- Checked and Unchecked Exception
- Keywords for handling Exception
- throw and throws
- User defined Exception

What is Exception ?

- Abnormal condition that disrupts the normal flow of execution.

- Can be logical errors , database errors like
  - Connecting with the database
  - Attempting to access a file that does not exist
  - Inserting an element into an array at a position that is not in its bounds
  - Performing some mathematical operation that is not permitted
  - Declaring an array using negative values

Unhandled Exceptions

- Uncaught Exceptions
  - Runtime system throws this exception
  - Default handler of java runtime system(JVM) handles
  - Displays string describing exception
    - Prints stack trace
    - Terminates program
  - Stack trace stores sequence of method invocations that led to error

Example

```
class  Demo {
   public static void main(String args[]) {
       int x = 0;
       int y = 50/x;
       System.out.println("y  = " +y);
   }
 }
```

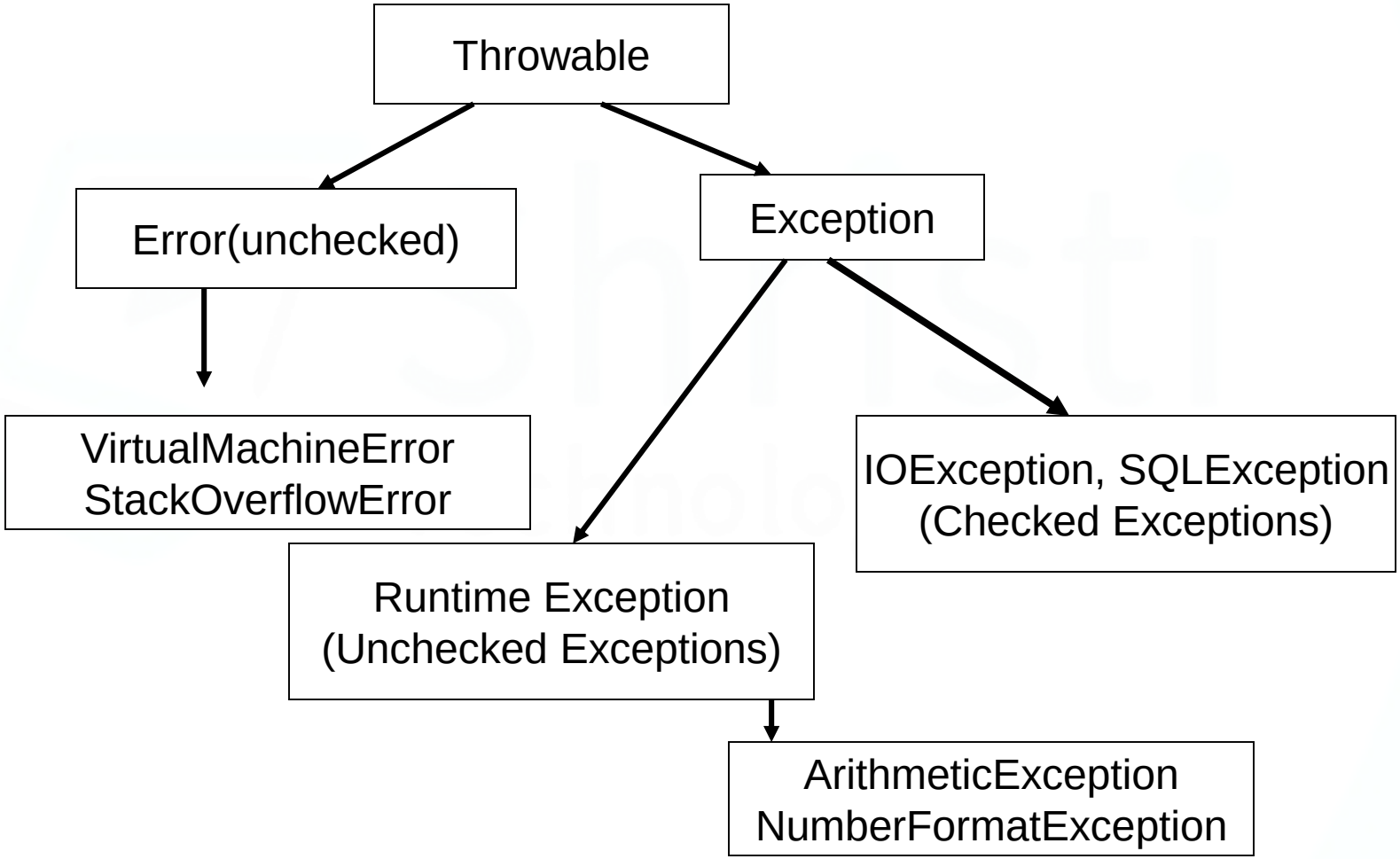
- This program compiles fine
- When executed the Java run-time-system will generate an exception
- Displays the following output on the console

java.lang.ArithmeticException: / by zero
at Demo.main(Demo.java:4)

What is exception handling ?

- The ability of a program to intercept run-time errors, take corrective measures and continue execution is referred to as exception handling
- All the exceptions must be either handled or declared.
- Five keywords used in handling exception are
  - try
  - catch
  - finally
  - throw
  - throws

Hierarchy of Exception



Types of Exception

- Unchecked Exception
  - Also called as RuntimeException
  - If not handled, will be handled by the JVM and terminates the application
- Checked Exception
  - Also called as CompiletimeException.
  - If not handled will throw a compiler error that it must be handled or declared

Error

- are exceptions that are not expected to be caught under normal circumstances by the program.
- are used by the Java runtime system to indicate errors having to do with the runtime environment.
- Stack overflow is an indication of such an error.
- Exceptions of type ***Error*** are beyond the control of the program

Examples of inbuilt Exceptions

- Errors:
  - OutOfMemoryError, InternalError
- Unchecked Exceptions:
  - ArrayIndexOutOfBoundsException, ArithmeticException, ClassCastException, NullPointerException
- Checked Exceptions:
  - ClassNotFoundException, IOException, SQLException

User-Defined Exception

- Can create your own exceptions also by extending the **Exception** class

- are called user-defined exceptions, and can be used in situations that are unique to the applications

Exception Handling keywords

Use of try, catch, finally

```
try {
  // all error prone code  here
  }
catch(TypeofException  obj) {
      //handle the exception
   }
 finally  {
   //clean up code written to release the resources like database
     connection closing, closing the file etc.
   }
```

More on try-catch

- No statements between try and catch
- A try can have any number of catch statements each catching a different type of exception
- A try block has to be with a catch or with a finally block.(ie)
  - try-catch
  - try-catch-catch
  - try-catch-finally
  - try-finally
  - Try –with resources

Used when you handle the exception in a different place

Example for try - catch

```
public class BasicEx {

    public static void main(String[] args) {
        try{
        System.out.println("welcome");
        String val = args[0];
        System.out.println("Value got");
        int num = Integer.parseInt(val);
        System.out.println("Converted");
        int total = 100/num;
        System.out.println("Total "+total);
        }catch(Exception e){
            System.out.println(e);
        }
        System.out.println("work done");
    }
}
```

Example for try-catch-catch

```
public class MultipleCatch {
    public static void main(String[] args) {
        System.out.println("welcome");
        try {
            String val = args[0];
            System.out.println("Value got");
            int num = Integer.parseInt(val);
            System.out.println("Converted");
            int total = 100 / num;
            System.out.println("Total " + total);
        } catch (ArithmeticException e) {
            System.out.println(" Dont enter 0");
        } catch (NumberFormatException e) {
            System.out.println(" input numbers not words!!!!!!");
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println(" enter a value");
        }
        System.out.println("main completed");
    }
```

Multiple Catch

- In multiple catch, the subclass exceptions must come before any of their superclass exceptions
- If a catch statement uses a superclass exception first, it will catch exceptions of that type as well as exceptions of its subclasses
- so a subclass exception would never be reached that manifests as an ***unreachable code error***

Example for try-catch(multi)

```
public class MultiCatch {
    public static void main(String[] args) {

        System.out.println("welcome");
        try {
            String val = args[0];
            System.out.println("Value got");
            int num = Integer.parseInt(val);
            System.out.println("Converted");
            int total = 100 / num;
            System.out.println("Total " + total);
            int m[] = null;
            System.out.println(m[7]);
        } catch (NumberFormatException | NullPointerException |
                ArithmeticException | ArrayIndexOutOfBoundsException e) {
            System.out.println(e);
        }
        System.out.println("main completed");
    }
}
```

finally

- Used for releasing resources(clean up code)
- Will be executed for both cases, with exception or without exception

when exception occurs(try+catch+finally)

- Statement after exception in try block will not be executed
- A matching catch will be executed
- All statements in finally executed
- Statements after finally will also be executed

when exception occurs(try+finally)

- Statement after exception in try block will not be executed
- All statements in finally executed
- Statements after finally will not be executed
- Searches for a matching catch in the calling part.
- If no catch is found JVM catches the exception

Example of try-catch-finally/try-finally

```
public class FinallyDemo {
    public static void main(String[] args) {
        System.out.println("welcome");
        try {
            String val = args[0];
            System.out.println("Value got");
            int num = Integer.parseInt(val);
            System.out.println("Converted");
            int total = 100 / num;
            System.out.println("Total " + total);
        } catch (Exception e) {
            System.out.println("technical error!!!!!!" + e);
        }finally {
            System.out.println("close connection in finally");
        }
        System.out.println("main completed");
    }
}
```

throw

- Is used to throw the exception to the place from where this method was called.
- Is used if the exception should not handled in the current method, but thrown.
- The general form of throw is:
  - throw ***ThrowableInstance***

where ***ThrowableInstance*** must be an object of type **Throwable**, or a subclass of **Throwable**

eg. ***throw new ArithmeticException();***

More on throw

RuntimeException(Unchecked)

- If a method throws runtime exception, you can either handle it with ***try/catch*** or allow ***JVM to handle*** it.

CompileException(Checked)

- If a method throws checked exception,
  - Declare the method with throws keyword
  - When you call this method call it within ***try/catch*** block
  - To allow JVM to handle it, declare the method which is calling also with throws keyword

Example for throw

```
public class Checker {
    void checkCredentials(String name) throws Exception {
        if (name.equals("admin")) {
            System.out.println("welcome");
            System.out.println("correct credentials");
        } else
            throw new Exception();
    }
}
```

```
public class ThrowDemo {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        String name = sc.next();
        Checker ch = new Checker();
        try {
            ch.checkCredentials(name);
            System.out.println("continue to site");
        } catch (Exception e) {
            System.out.println("wrong user");
        }
    }
}
```

throws

- Is used to declare that the method throws exception
- Mainly used for compile time Exception
- Can throw one to many exceptions

eg. public void withdraw(int amount) throws IOException, Exception

- When any method calls this method, it is mandatory
  - to either call within try/catch block or declare this exception in the calling method.
- or it will give compiler error

Example for throws

```
public class Bank {

    void withdraw(int x) throws Exception {
        try {
            System.out.println("in Bank");
            if (x > 1000) {
                throw new Exception("Not allowed");
            } else {
                System.out.println("withdrawn " + x);
            }
        } catch (Exception e) {
            System.out.println("error occured");
            throw e;
        } finally {
            System.out.println("close db");
        }
        System.out.println("Work done");
    }
```

```
public class ThrowsMain {

    public static void main(String[] args)  {
        System.out.println("welcome to ABCX bank");
        Scanner sc  = new Scanner(System.in);
        System.out.println("Enter Amount to withdraw");
        int amount = sc.nextInt();
        System.out.println("Checking");
        Bank bank = new Bank();
        try {
            bank.withdraw(amount);
            System.out.println("Amount withdrawn");
        } catch (Exception e) {
            System.out.println("Insufficient balance");
        }
        System.out.println("main completed");
    }
}
```

Custom Exceptions

- Custom Exceptions are used for throwing project specific errors
- Some examples can be like showing negative balance, age below the requiredlimit, taking Overdraft, checking password length etc
- To create a custom Exception make a class that extends Exception class

Example for custom exception

```
public class TooLongException extends Exception {
    private static final long serialVersionUID = 1L;

    public TooLongException() {
        super();
    }

    public TooLongException(String message) {
        super(message);
    }

}
```

Example for custom exception

```
String checkPassword(String password)
        throws TooShortException, TooLongException {
    if (password.length() < 4) {
        throw new TooShortException("Password is short");
    }
    if (password.length() > 10) {
        throw new TooLongException();
    }
    return "validated";
}
```

Rules for overriding Exception

- The overriding method must NOT throw checked exceptions that are new or broader than those declared by the overridden method

eg: A method that declares(throws) an SQLException in super class
can be overridden only by a method that is a subclass of SQLException

- if a method declares to throw a given exception, the overriding method in a subclass can only declare to throw the same exception or its subclass

- This rule does not apply for unchecked exceptions

Try with resources

- This technique helps to close the resources automatically, after the application gets completed.
- Using try-with-resources eliminates the task of adding finally to close resources
- **Example for a resource:** A file or a connection object or a scanner object.
- Any class that implements ***java.lang.AutoCloseable*** **or** ***java.io.Closeable*** is considered as a resource and can be used with try-with-resources construct.

Example for try with resources

```
public class TrywithResourcesDemo {

    public static void main(String[] args) {

        // try with resources can be with/without catch
        try(Scanner sc = new Scanner(System.in)){
            String name = sc.next();
            int salary = sc.nextInt();
            System.out.println("Name "+name);
            System.out.println("Salary "+salary );
        }
    }
}
```

Example for try with resources

- MyResource is a class which implements AutoCloseable and overrides *close()* method.
- *myMethod()* is the method of MyResource class

```
class MyResource implements AutoCloseable{

    @Override
    public void close() throws Exception {
        System.out.println("Closing the resources");
    }
    public void myMethod(){
        System.out.println("Doing some calculation");
    }
}
public class TrywithResources3 {

    public static void main(String[] args) {

        try(MyResource res = new MyResource()){
            res.myMethod();
        }
    }
}
```

Quiz

What will be the result, if we try to compile and execute the following code

```
class Test {
        public static void main(String[] args){
                for(int i=1;i<=args.length;i++)
           System.out.println(args[i]);
        }
    }
```

Ex:1 java Test

Ex:1 java Test Hello World

Quiz

What will be the result, if we try to compile and execute the following code

```
class Test{
        public static void main(String[] args) {
           try {
             int i= Integer.parseInt(args[0]);
             System.out.println(i);
           }System.out.println("Hello World");
           catch(NumberFormatException e) {
                   System.out.println(e);
           }
        }
         }
```

Ex:2 java Test 100

Quiz

What will be the result, if we try to compile and execute the following code

```
class Test {
        public static void main(String[] args) {
      try {
        int i= Integer.parseInt(args[0]);
        System.out.println(i);
      }
      catch(RuntimeException e) {
        System.out.println(e);
      }
      catch(NumberFormatException e) {
        System.out.println(e);}
    }
}
```

Ex:3 java Test 100

Quiz

What will be the result, if we try to compile and execute the following code

```
class Test{
    void throwOne()  {
            System.out.println("Inside throwOne.");
            throw new FileNotFoundException();
    }
    public static void main(String args[]){
      Test t = new Test();
        t.throwOne();            }
}
```

Ex:4 java Test

Quiz

What will be the result, if we try to compile and execute the following code

```
class  Test{
            public static void main(String[] args) {
          try {
            int i= Integer.parseInt(args[0]);
            System.out.println(i);
          }catch(NumberFormatException e) {
            System.out.println(e);
          }System.out.println("Exception Caught");
          finally { }
    } }
```

Ex:5 java Test

Quiz

What will be the result, if we try to compile the following code

```
class Super {
    void m1() throws ArithmeticException {
        int x = 100, y=0;
        int z=x/y;
        System.out.println(z);
    }}
class Sub extends Super {
    void m1() throws NumberFormatException {
        System.out.println(“Hello World");
    }
}
```

Quiz

What will be the result, if we try to compile the following code (FileNotFoundException is a subclass of IOException)

```
import java.io.*;
class Super {
    void m1() throws FileNotFoundException {
        FileInputStream fx = new FileInputStream("Super.txt");
    }
}
class Sub extends Super {
    void m1() throws IOException {
        FileInputStream fx = new FileInputStream("Sub.txt");
    }
}
```

Quiz

What will be the result, if we try to compile the following code (FileNotFoundException is a subclass of IOException)

```
import java.io.*;
class Super {
    void m1() throws IOException {
        FileInputStream fx = new FileInputStream("Super.txt");
    }}
class Sub extends Super {
    void m1() throws FileNotFoundException {
        FileInputStream fx = new FileInputStream("Sub.txt");
    }}
```

Summary

- Define Exception
- Checked and Unchecked Exception
- Keywords for handling Exception
- throw and throws
- User defined Exception
- Try with resources

Thank You