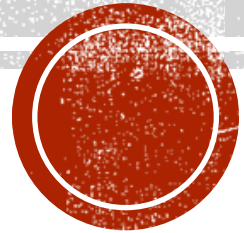
EXCEPTION HANDLING



Lecture 20

Prof. Anita Agrawal

BITS Pilani, Goa

COMMON JAVA EXCEPTIONS

- A scenario where ArithmeticException occurs.

  Example: **int** a=50/0; //ArithmeticException

- A scenario where NullPointerException occurs.

  Example: String s=**null**;

  System.out.println(s.length());

- A scenario where NumberFormatException occurs

  Example: String s="abc";

  int i=Integer.parseInt(s); //NumberFormatException

- A scenario where ArrayIndexOutOfBoundsException occurs-

  Example: int a[ ]=new int [5];

  a [10]=50; //ArrayIndexOutOfBoundsException

TRY-CATCH BLOCK SYNTAX

- try {

  //code that may throw an exception

  } catch(Exception_class_Name ref){ }

- try block is used to enclose the code that might throw an exception. It must be used within the method.
- catch block is used to handle the Exception by declaring the type of exception within the parameter. The declared exception must be the parent class exception (i.e., Exception) or the generated exception type

MULTIPLE CATCH CLAUSE

- Exception subclasses must come before any of their superclasses
- Otherwise, **catch** statement that uses a superclass will catch exception of that type and its subclasses
- Subclass would never be reached if it came after its superclass
- In Java, unreachable code is an error

- Unreachable code

NESTED TRY

- The try block within a try block is known as nested try block.
- Sometimes a situation may arise where a part of a block may cause one error and the entire block itself may cause another error. In such cases, exception handlers have to be nested.
- Each time a **try** statement is entered
- The context of that exception is pushed on the stack
- If an inner **try** statement does not have a **catch** handler for a particular exception
- The next **try** statement's **catch** handlers are inspected for a match
- This continues until one of the **catch** statements succeed
- If no **catch** statement matches
- Then, the Java run-time system will handle the exception

NestC, Nested1

```
//Main try block
try {
  statement 1;
  statement 2;
  //try-catch block inside another try block
  try {
    statement 3;
    statement 4;
    //try-catch block inside nested try block
    try {
      statement 5;
      statement 6;
    }
    catch(Exception e2) {
      //Exception Message
    }
  }
  catch(Exception e1) {
     //Exception Message
  }

}
//Catch of Main(parent) try block
catch(Exception e3) {
    //Exception Message
}
```

FINALLY

- Java finally block is a block that is used to execute important code such as closing connection, stream etc.
- **finally** creates a block of code that will be executed after a **try /catch** block has completed and before the code following the **try/catch** block
- The **finally** block will execute whether or not an exception is thrown
- If an exception is thrown, the **finally** block will execute even if no **catch** statement matches the exception
- Useful for closing file handles and freeing up any other reserved resources
- The **finally** clause is optional
- However, each **try** statement requires at least one **catch** or a **finally**

THROW

- We are only **catching** exceptions that are **thrown by the Java run-time system**
- It is possible for your program to throw an exception explicitly
- The throw keyword in Java is used to explicitly throw an exception from a method or any block of code.
- We can throw either checked or unchecked exception.
- The throw keyword is mainly used to throw custom exceptions.
- The general form of **throw** is

  throw *ThrowableInstance*

THROW

- *ThrowableInstance* must be an object of type **Throwable** or a subclass of **Throwable**
- Primitive types and object of **String** and **Object** cannot be used as exceptions.
- The flow of execution stops immediately after the **throw** statement
- The nearest enclosing **try** block is inspected
  - If a **catch** statement that matches the type of exception is available
    - If it does, control is transferred to that statement
    - If not, then the next enclosing **try** statement is inspected, and so on

- If no matching **catch** is found
  - Then the default exception handler halts the program
  - Prints the stack trace

- Throw1
- Throw2

THROW

- Most Java's built-in runtime exceptions have at least two constructors
- One with no parameter and
- One that takes a string parameter
- When the second form is used, the argument specifies a string that describes the exception
- This string is displayed when the object is used as an argument to **print( )** or **println( )**

THROWS

- As we know that there are two types of exception checked and unchecked.
- Checked exception (compile time) force you to handle them, if you don't handle them then the program will not compile.
- On the other hand unchecked exception (Runtime) doesn't get checked during compilation.
  - **Throws** keyword is used for handling checked exceptions .
  - By using throws we can declare multiple exceptions in one go.
- The throws does the same thing that try-catch does but there are some cases where you would prefer throws over try-catch.

- Throw4

THROWS

- If a method is capable of causing an exception that it does not handle
- This behavior must be specified to the callers of the method
- A **throws** clause lists the types of exceptions that a method might throw
- General form of a method declaration with a **throws** clause:
  - *type method-name*(*parameter-list*) throws *exception-list*{// body of method}

CREATING YOUR OWN EXCEPTION SUBCLASSES

- Create your own exception types to handle situations specific to your applications
- Just define a subclass of **Exception** (which is, of course, a subclass of **Throwable**)
- Your subclasses don't need to actually implement anything
- Their existence in the program allows you to use them as exceptions
- The **Exception** class does not define any methods of its own
- It inherits the methods provided by **Throwable**
- You may also wish to override one or more of these methods

- **Exception** defines multiple public constructors
- Two most commonly used constructors
  - *Exception( )* -creates an exception that has no description
  - *Exception(String msg)* -lets you specify a description of the exception