CSN-103: Fundamentals of Object Oriented Programming

Instructor: Dr. Rahul Thakur

Assistant Professor, Computer Science and Engineering, IIT Roorkee

Exception

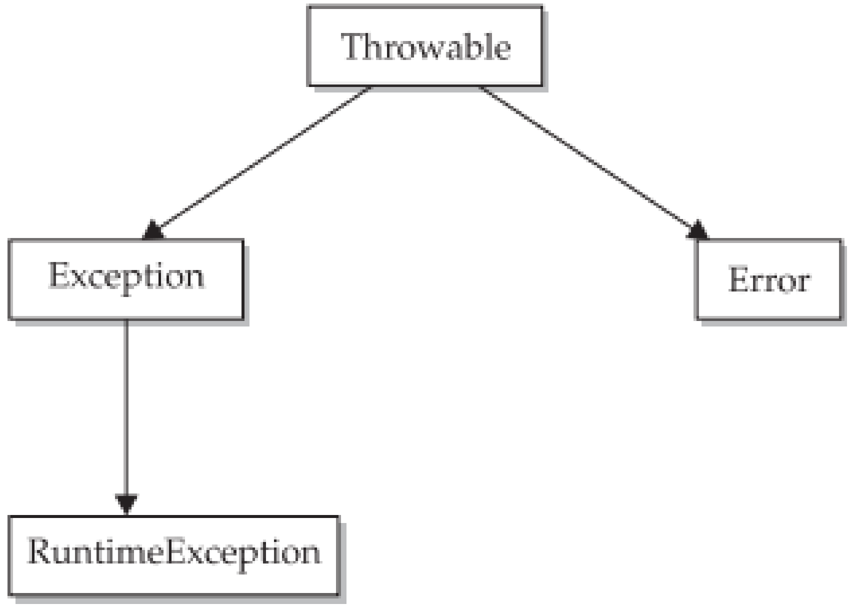
- An ***exception*** is an abnormal condition that arises in a code sequence at run time
- In other words, an exception is a runtime error
- Some computer languages that do not support exception handling
  - Runtime errors must be checked and handled manually
  - Writing additional code: Error codes
- Java's exception handling brings run-time error management into the **object-oriented** world

Fundamentals

- In Java, an exception is an object
  - Describes an exceptional condition occurred in a piece of code
- When an exceptional condition arises
  - An **object** representing that exception is created
  - Then, the **object** is *thrown* in the method that caused the error
  - The method can choose to handle the exception or pass it on
  - In the end, at some point, exception is *caught* and processed
- Who generates the Exceptions?
  - Can be generated by the Java run-time system
  - Can be manually generated by your code

Exception Types

- Java has a build-in class **Throwable**
- All exception types are subclasses of class **Throwable**
- Top-level exception hierarchy



Exception Types

- **Exception** class is used for exceptional conditions that **user programs** should catch
- You will subclass **Exception** to create your own custom exception types
- An important subclass of **Exception** is:
  - **RuntimeException**: Exceptions of this type are automatically defined for the programs you write
  - Includes: division by zero and invalid array indexing

Exception Types

- Class **Error** defines the exceptions that are not expected to be caught by your program
- Exceptions of type **Error** are used by the Java run-time system
  - For errors having to do with the run-time environment
  - Stack overflow is an example
  - We will not be dealing with exceptions of type **Error**
  - Used in response to catastrophic failures
    - Usually cannot be handled by your program

Uncaught Exceptions

- What happens when you don’t handle exceptions in your program

```
class Exc0 {
      public static void main(String args[]) {
      int d = 0;
      int a = 42 / d;
      }
}
```

- No compile time error
- Runtime error (Exception)

```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Exc0.main(Exc0.java:4)
```

Uncaught Exceptions

- When the Java run-time system detects the attempt to divide by zero
  - It constructs a new exception object and then *throws* this exception
- What happens next?
  - The execution of **Exc0** stops
  - Exception is *caught* by an exception handler and dealt with immediately
  - Where is the exception handler?

Uncaught Exceptions

- Our program should provide the exception handler to catch the exception and process them
- If we haven't supplied any exception handlers of our own
  - The default handler provided by the Java run-time system
  - Prints a stack trace from the point at which the exception occurred
  - Terminates the program

Uncaught Exceptions

- The stack trace will always show
  - The sequence of method calls that led up to the error

```
class Exc1 {
     static void subroutine() {
     int d = 0;
     int a = 10 / d;
     }
public static void main(String args[]) {
     Exc1.subroutine();
     }
}
```

Stack trace from default exception handler:

```
java.lang.ArithmeticException: / by zero
at Exc1.subroutine(Exc1.java:4)
at Exc1.main(Exc1.java:7)
```

Try and Catch

- The default exception handler provided by the Java run-time system is useful for debugging
- Usually, we want to handle an exception ourselves
- Benefits:
  - It allows you to fix the error
  - It prevents the program from automatically terminating
- To handle a run-time error
  - Enclose the code that you want to monitor inside a **try** block
  - Immediately following the **try** block, include a **catch** clause
    - Specify the exception type that you wish to catch

Try and Catch

In **Exc2.java** example:

- **println( )** inside the **try** block is never executed
- Once the **catch** statement has executed
  - Program control continues with the next line in the program following the entire **try/catch**
- A **try** and its **catch** statement form a unit
  - A **catch** statement cannot catch an exception thrown by another **try** statement

    (except in the case of nested **try** statements)
  - The statements that are protected by **try** must be surrounded by curly braces
- The goal of most well-constructed **catch** clauses should be
  - To resolve the exceptional condition
  - Then continue on as if the error had never happened