CSN-103: Fundamentals of Object Oriented Programming

Instructor: Dr. Rahul Thakur

Assistant Professor, Computer Science and Engineering, IIT Roorkee

Multiple catch Clauses

- In some cases, more than one exception could be raised by a single piece of code
- To handle this type of situation
  - We can specify two or more **catch** clauses
  - Each catching a different type of exception
- When an exception is thrown
  - Each **catch** statement is inspected **in order**
  - The first one whose type matches is executed
- After one **catch** statement executes, the others are bypassed
- Execution continues after the **try /catch** block

Multiple catch Clauses

- It is important to remember
  - 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

Nested try Statements

- The **try** statement can 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 succeeds
  - If no **catch** statement matches
    - Then, the Java run-time system will handle the exception

Nested try Statements

- Nesting of **try** statements: When method calls are involved
- We can enclose a call to a method within a **try** block
  - Inside that method is another **try** statement
- The **try** within the method is still nested inside the outer **try** block
- NOTE: **toString()**
  - A method to get string representation of an object
  - Declared in **Object** class and usually overloaded
  - Whenever we try to print an object using **println()** or **toString()** method is invoked internally

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
  - Using the **throw** statement
- The general form of **throw** is

  throw *ThrowableInstance*;
- *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

throw

- 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

ThrowDemo.java

- Illustrates how to create one of Java’s standard exception objects

```
throw new NullPointerException("demo");
```

- 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

- If a method is capable of causing an exception that it does not handle
  - This behavior must to specified to the callers of the method
- A **throws** clause lists the types of exceptions that a method might throw
  - **throws** is necessary for all exceptions
  - Except **Error** or **RuntimeException** (and their subclasses)
- General form of a method declaration with a **throws** clause:

type method-name(*parameter-list*) throws *exception-list*
{
        // body of method
}

ThrowsDemo .java (Incorrect Program)   ThrowsDemo2.java (Correct Program)

finally

- **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** clause

Creating Your Own Exception Subclasses

Custom Exception

- 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
  - There existence in the program allows you to use them as exceptions
  - Sometimes it is better to override **toString( )** to display a description of your exception
    - Display a cleaner output

ExceptionDemo.java