Exception Handling
Part 2

Prof. Siddharth Shah
Department of Computer Engineering
Dharmsinh Desai University

Outline

- Multiple **catch** Clauses
- Nested **try** Statements
- **throw** Statement
- **throws** Clause
- **finally**
- ***multi-catch* feature**

Multiple ***catch*** Clauses - 1

- In some cases, more than one exception could be raised by a single piece of code.
- To handle this type of situation, you 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, and the first one whose type matches that of the exception is executed.
- After one catch statement executes, the others are bypassed, and execution continues after the try / catch block.

Multiple ***catch*** Clauses - 2

- When you use multiple catch statements, it is important to remember that exception subclasses must come before any of their superclasses.
- This is because a catch statement that uses a superclass will catch exceptions of that type plus any of its subclasses.
- Thus, a subclass would never be reached if it came after its superclass.
- Further, in Java, unreachable code is an error.

Nested *try* Statements - 1

- The try statement can be nested. That is, a try statement can be inside the block of another try.
- 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 stack is unwound and the next try statement's catch handlers are inspected for a match.
- This continues until one of the catch statements succeeds, or until all of the nested try statements are exhausted. If no catch statement matches, then the Java run-time system will handle the exception.

Nested *try* Statements - 2

- Nesting of try statements can occur in less obvious ways when method calls are involved.
- For example, you can enclose a call to a method within a try block.
- Inside that method is another try statement.
- In this case, the try within the method is still nested inside the outer try block, which calls the method.

throw Statement - 1

- It is possible for your program to throw an exception explicitly, using the throw statement. The general form of throw is shown here:
- **throw ThrowableInstance;**
- ThrowableInstance must be an object of type Throwable or a subclass of Throwable.
- Primitive types, such as int or char, as well as non-Throwable classes, such as String and Object, cannot be used as exceptions.
- There are two ways you can obtain a Throwable object:
- using a parameter in a catch clause
- creating one with the new operator

throw Statement - 2

- The flow of execution stops immediately after the throw statement; any subsequent statements are not executed.
- The nearest enclosing try block is inspected to see if it has a catch statement that matches the type of exception.
- If it does find a match, control is transferred to that statement. If not, then the next enclosing try block is inspected, and so on.
- If no matching catch is found, then the default exception handler halts the program and prints the stack trace.

throws Clause - 1

- If a method is capable of causing an exception that it does not handle, it must specify this behavior so that callers of the method can guard themselves against that exception.
- You do this by including a **throws** clause in the method's declaration. A throws clause lists the types of exceptions that a method might throw.
- This is necessary for all exceptions, except those of type Error or RuntimeException, or any of their subclasses.
- All other exceptions that a method can throw must be declared in the throws clause. If they are not, a compile-time error will result

throws Clause - 2

- Below is general form of a method declaration that includes a throws clause:

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

- Here, exception-list is a comma-separated list of the exceptions that a method can throw.

finally - 1

- **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.

finally - 2

- Any time a method is about to return to the caller from inside a try/catch block, via an uncaught exception or an explicit return statement, the finally clause is also executed just before the method returns.
- This can be useful for closing file handles and freeing up any other resources that might have been allocated at the beginning of a method with the intent of disposing of them before returning.
- The finally clause is optional. However, each try statement requires at least one catch or a finally clause.

multi-catch

- The **multi-catch** feature allows two or more exceptions to be caught by the same catch clause.
- It is not uncommon for two or more exception handlers to use the same code sequence even though they respond to different exceptions.
- Instead of having to catch each exception type individually, you can use a single catch clause to handle all of the exceptions without code duplication.
- To use a multi-catch, separate each exception type in the catch clause with the OR operator.
- Each multi-catch parameter is implicitly final, and hence it can’t be assigned a new value.