COMPSCI 105

Principles of Computer Science

Exception handling

Revision

System of resolving unexpected situations that arise when your program is running.

- Breaks the normal flow of control
- Exception is thrown at the point of the problem
- Exception is caught at the point when it can be resolved

```
try {
      //try block

}
catch (ExceptionType exceptionIdentifier) {
      //catch block
}
catch (AnotherExceptionType exceptionIdentifier) {
      //catch block
}
```


Nested `try... catch` blocks

Exception is caught by the first `catch` clause that it matches

If no matching `catch` clause is found

- Exception is propagated to the next higher enclosing block of code
- Proceeds as normal through the method call stack in reverse order

```
try {
      try {

            //Exception generated here

      }
      catch (FirstException e) {  }
      catch (SecondException ae) {  }
}
catch (ThirdException te) {  }
catch (FourthException fe) {  }
```


Example 01

```
private void example01() {
      int num = 0;
      String s = null;
      try {
            System.out.println( "A" );
            try {
                  num = s.length();
                  num = 200/num;
                  System.out.println( "B" );
            } catch( NullPointerException e ) {
                  System.out.println( "C Error" );
            }
            System.out.println( "D" );
      } catch( ArithmeticException e ) {
            System.out.println( "E Error" );
      }
      System.out.println( "F" );
}
```

Exercise 01

What is the output produced by this code?

```
private void exercise01() {
      int num = 0;
      String s = "";
      try {
            System.out.println( "A" );
             try {
                  num = s.length();
                  num = 200/num;
                  System.out.println( "B" );
            } catch( NullPointerException e ) {
                  System.out.println( "C Error" );
            }
            System.out.println( "D" );
      } catch( ArithmeticException e ) {
             System.out.println( "E Error" );
      }
      System.out.println( "F" );
}
```


Exercise 02

What is the output produced by this code?

```
private void exercise02() {
      int num = 0;
      String s = null;
      try {
            num = s.length();
            System.out.println( "A" );
            try {
                  num = 200/num;
                  System.out.println( "B" );
            } catch( NullPointerException e ) {
                  System.out.println( "C Error" );
            }
            System.out.println( "D" );
      } catch( ArithmeticException e ) {
            System.out.println( "E Error" );
      }
      System.out.println( "F" );
}
```


Example 02

```
private void start() {
      try {
            example02( 0, null );
            System.out.println( "A" );
      } catch( ArithmeticException e ) {
            System.out.println( "B Error" );
      }
}
private void example02(  int num, String s ) {
      System.out.println( "C" );
      try {
            num = s.length();
            num = 200/num;
            System.out.println( "D" );
      } catch( NullPointerException e ) {
            System.out.println( "E Error" );
      }
      System.out.println( "F" );
}
```


Practice Exercise

```
private void practice01() {
      try {
            tryMethod( 0, "" );
            System.out.println( "A" );
      } catch( ArithmeticException e ) {
            System.out.println( "B Error" );
      }
}
private void tryMethod(  int num, String s ) {
      System.out.println( "C" );
      try {
            num = s.length();
            num = 200/num;
            System.out.println( "D" );
      } catch( NullPointerException e ) {
            System.out.println( "E Error" );
      }
      System.out.println( "F" );
}
```

Quick Exercise

Where do you put the try catch if you ALWAYS want the statement C to be executed even if there is an exception in the statement after statement B which is caught?

```
private void tryCatch03() {

            try03( 0, null );

            System.out.println( "A" );

}
private void try03(  int num, String s ) {
            System.out.println( "B" );
      try {
            num = s.length();
      } catch (Exception e) {
            System.out.println( "C" );
      }
}
```


Quick Exercise

Where do you put the try catch if you DON'T want the statement C to be executed if there is an exception after statement B which is caught?

```
private void tryCatch04() {
      try {
            try03( 0, null );
      } catch (Exception e) {
            System.out.println( "A" );
      }
}
private void try04(  int num, String s ) {
            System.out.println( "B" );

            num = s.length();

            System.out.println( "C" );

}
```


Finally

An optional component of the try-catch structure that is sometimes useful is the `finally` block.

Code within a `finally` block is guaranteed to be executed if any part of the associated try block is executed, regardless of an exception being thrown or not.

If an exception occurs in the related `try` and there is a matching `catch` in the local block, the `finally` is executed after the `catch`.

Example 03

```
private void tryCatchFinallyEx03() {
      int num = 0;
      String s = "";
      System.out.println( "A" );
      try {
            num = s.length();
            System.out.println( "B" );
      } catch( NullPointerException e ) {
            System.out.println( "C Error" );
      } finally {
            System.out.println( "D Finally" );
      }
      System.out.println( "E" );
}
```

Example 04

```
private void tryCatchFinallyEx04() {
	int num = 0;
	String s = null;
	System.out.println( "A" );
	try {
		num = s.length();
		System.out.println( "B" );
	} catch( NullPointerException e ) {
		System.out.println( "C Error" );
	} finally {
		System.out.println( "D Finally" );
	}
	System.out.println( "E" );
}
```


Finally

Code in a finally block is also guaranteed to be executed even when the code in the associated `try` or `catch` blocks executes a `return` or `break`.

The `try` block can have an associated `finally` block even if there is no associated `catch` block.

Reminder: code within a `finally` block is guaranteed to be executed if any part of the associated try block is executed, regardless of an exception being thrown and not caught.

Example

```
private void tryCatchFinallyEx03() {
	int num = 0;
	System.out.println( "A" );
	try {
		num = 200/num;
		System.out.println( "B" );
	} catch( ArithmeticException e ) {
		System.out.println( "C Error" );
		if ( num==0 ) {
			return;
		}
		System.out.println( "E" );
	} finally {
		System.out.println( "D Finally" );
	}
	System.out.println( "E" );
}
```

Example

Exception is not generated.

```
private void tryCatchFinallyEx04() {
	int num = 2;
	System.out.println( "A" );
	try {
		num = 200/num;
		System.out.println( "B" );
	} finally {
		System.out.println( "D Finally" );
	}
	System.out.println( "E" );
}
```

Example

Exception is generated, but not caught

```
private void tryCatchFinallyEx05() {
	int num = 0;
	System.out.println( "A" );
	try {
		num = 200/num;
		System.out.println( "B" );
	} finally {
		System.out.println( "D Finally" );
	}
	System.out.println( "E" );
}
```


Exercise

What is the output?

```
private void tryCatch05() {
	try {
		try05( 0, null );
		System.out.println( "A" );
	} catch ( NullPointerException e ) {
		System.out.println( "B" );
	}
}
private void try05(  int num, String s ) {
	System.out.println( "C" );
	try{
		num = s.length();
		System.out.println( "D" );
	} finally {
		System.out.println( "E" );
	}
	System.out.println( "A" );
}
```


Exercise

What is the output?

```
try {
	String a = "0";
	String b = "10";
	String c = "abcde";
	int r1 = Integer.parseInt(c);
	System.out.println( "A" );
	int r2 = Integer.parseInt(b)/ Integer.parseInt(a);
	System.out.println( "B" );
} catch (ArithmeticException e) {
	System.out.println("Arithmetic Error!");
} catch (NumberFormatException e) {
	System.out.println("Incorrect Format");
} finally {
	System.out.println( "C" );
}
System.out.println( "D" );
```


Exercise

What is the problem?

```
private void problemEx01() {
	int num = 0;
	try {
		System.out.print( "Enter number: " );
		num = Integer.parseInt( Keyboard.readInput() );
		System.out.println( "A" );
	}
	System.out.println( "B" );
	catch( NumberFormatException e ) {
		System.out.println( "C" );
	}
}
```

```
Program.java:431: 'try' without 'catch' or 'finally'
		try {
		^
Program.java:437: 'catch' without 'try'
		catch( NumberFormatException e ) {
```

Exercise

What is the problem?

```
private void problemEx02() {
      int result = 0;
      int[] nums = { 2, 3, 4, 2, 4 };
      try {
            result = nums[ nums.length ];
            System.out.println( "A" );
      } catch( RuntimeException e ) {
            System.out.println( "B" );
      } catch( ArrayIndexOutOfBoundsException e ) {
            System.out.println( "C"  );
      }
      System.out.println( "D" );
}
```

```
Program.java:461: exception
java.lang.ArrayIndexOutOfBoundsException has already been
caught        } catch( ArrayIndexOutOfBoundsException e ) {
                  ^
1 error
```


Exercise

What is the problem?

```
private void problemEx03() {
      int[] nums = { 2, 3, 4, 2, 4 };
      try {
            int result = 0;
            result = nums[ nums.length ];
            System.out.println( "A" );
      } catch( ArrayIndexOutOfBoundsException e ) {
            System.out.println( "B" );
            result = -1;
      }
      System.out.println( "C" );
}
```

```
P:\Programs\Exercise.java:15: cannot find symbol
symbol  : variable result
location: class Exercise01
            result = -1;
            ^
1 error
```


Exception handling so far 1

What is the output?

```
private void tryCatchFinally01() {
  try {
      String a = "0";
      int r2 = Integer.parseInt(a)/Integer.parseInt(a);
      System.out.println( "After division" );
  } catch ( ArithmeticException e ) {
      System.out.println( "Calculation Error" );
  } catch ( Exception e ) {
      System.out.println( "General Exception" );
  } finally {
      System.out.println( "Finally" );
  }
  System.out.println( "Finished" );
}
```


Revision exercise

What is the output?

```
private void tryCatchFinally02() {
  try {
      String a = "0";
      int r2 = Integer.parseInt(a)/Integer.parseInt(a);
      System.out.println( "After division" );
  } catch ( ArithmeticException e ) {
      System.out.println( "Calculation Error" );
      return;
  } catch ( Exception e ) {
      System.out.println( "General Exception" );
  } finally {
      System.out.println( "Finally" );
  }
  System.out.println( "Finished" );
}
```

Exception handling so far 3

What is the output?

```
private void tryCatchFinally03() {
  try {
       String a = "0oops";
       int r2 = Integer.parseInt(a)/Integer.parseInt(a);
       System.out.println( "After division" );
  } catch ( ArithmeticException e ) {
       System.out.println( "Calculation Error" );
  } catch ( NullPointerException e ) {
       System.out.println( "General Exception" );
  } finally {
       System.out.println( "Finally" );
  }
  System.out.println( "Finished" );
}
```


Remainder to be covered in tutorials

More information about Exceptions

- Can create your own type of exception (i.e. define a new class of exception)
- Use throw to throw a new exception

Checked exceptions

- Covered when we do I/O

Constructing Exception objects

`Exception()`

Constructs a new exception with `null` as its detail message.

`Exception( String message )`

Constructs a new exception with the specified detail message.

`throw` statements

So far we have only been catching exceptions that are thrown by the Java run-time system.

But an exception can be created and explicitly thrown using a `throw` statement, For example:

```
throw new ArithmeticException( "A problem occurred")
```

When the `throw` statement is reached:

- the program stops immediately after the `throw` statement;
- the exception is thrown
- the flow of control continues as for any other exception

`throw` statements example 1

```
private void throwEx01() {
   try {
       System.out.print( "Input any number except 2: " );
       int num = Integer.parseInt( Keyboard.readInput() );
       if ( num == 2 ) {
         throw new ArithmeticException( "Incorrect input" );
       }
       System.out.println( "A" );
   } catch( ArithmeticException e ) {
       System.out.println( e );
   }
   System.out.println( "B" );
}
```


`throw` statements example 2

```
private void throwEx02() {
   try {
       System.out.print( "Enter code: " );
       String numS1 = Keyboard.readInput();
       System.out.print( "Enter code again: " );
       String numS2 = Keyboard.readInput();
       if ( !numS1.equals(numS2) ) {
         throw new Exception( "Incorrect input" );
       }
       System.out.println( "Password: GetIn45" );
   } catch( Exception e ) {
       System.out.println( e.getMessage() );
   }
   System.out.println( "Finished" );
}
```


Re-throwing an Exception

If the `catch` handler catches an exception and cannot process it or if the exception needs further processing outside of the handler, the `catch` handler can simply re-throw the exception with a `throw` statement.

Such a `throw` statement re-throws the exception to the next enclosing `try` block.

`throw` statements example 3

```
private void throwEx03() {
    try {
            throwEx03( 123 );
     }  catch( Exception e ) {
            System.out.println( "Exception: " + e );
     }
}
private void throwEx03( int num ) {
    try {
            throwEx03( num, 346 );
            System.out.println( "A" );
     }  catch( RuntimeException e ) {
            throw e;
     }
}
private void throwEx03( int num1, int num2 ) {
     if ( num1 != num2 ) {
         throw new RuntimeException( "Bad parameters" );
     }
}
```