COMPSCI 105

Principles of Computer Science

Flow of Control
Exception handling

Flow of control

http://en.wikipedia.org/wiki/Control_flow

flow of control refers to the order in which the individual instructions of a program are executed

Program Counter

- Special variable
- Keeps track of the address of the next instruction to be executed

Sequential execution

Statements are executed from top to bottom

```
private void sequenceExecution() {
      System.out.println();
      System.out.println( "Code is executed from" );
      System.out.println( "top to bottom," );
      System.out.println();
      int num1 = 10 - 5  + 4;
      int num2 = 10 - (5  +  4);
      int num3 = 10 + 15 * (5  -  4);
      int num4 = num1 + num2 + num3;
      System.out.println( num3+" " +num4);
}
```


Selection statements

Certain statements may be "skipped"

- Parts of the code are NOT executed

```
private void selectionStatements() {
      System.out.println();
      System.out.println( "Selection statements" );
      System.out.println();
      int num1 = (int)(Math.random()*100);
      if( num1 < 50 ) {
            System.out.println( "BAD LUCK" );
      } else if ( num1 > 80 ) {
            System.out.println( "GREAT" );
      }
      System.out.println( "BYE" );
}
```

Iteration statements

Blocks of statements are repeatedly executed

- Loop condition defines when the loop is executed
- Body of the loop is executed only when the condition is true

```
private void iterationStatements() {
	System.out.println();
	System.out.println( "Iteration statements" );
	System.out.println();
	int num1 = (int)(Math.random()*100);
	int num2 = (int)(Math.random()*100);
	while ( (num1-num2) > 10 )  {
		num1 = (int)(Math.random()*100);
	}

	System.out.println( num1 + "," + num2  );
}
```


Branching statements: `break`

`break`

- Terminates the execution of the enclosing statement
  - `for`
  - `while`
  - `do-while`
  - `switch`

```
private void branchBreak() {
	int tries = 0;
	int num = (int)(Math.random()*100);
	while (  num%2 == 1 )  {
		if( tries > 5  ) {
			break;
		}
		tries++;
		num = (int)(Math.random()*100);
	}
	System.out.println( tries + ": " + num  );
}
```


Branching statements: `continue`

`continue`

- Skips the execution of the current iteration of a loop
  - `for`
  - `while`
  - `do-while`

```
private void branchContinue() {
	System.out.println("Continue");
	for (int counter = -128; counter < 127; counter ++){
		if (counter == 0) {
			continue;
		}
		double x = 1/counter;
		System.out.println("Fraction: " + x);
	}
}
```


Branching statements: `return`

`return`

- Terminates execution of a method, returns control to the calling method
- Return with or without a value, depending on the return type of the method

```
public void branchReturn() {
	System.out.println();
	System.out.println("Return");
	System.out.println();
	String depart = getDepart();
	System.out.println(depart);
}

private String getDepart() {
	if (Math.random() < 0.25) {
		return "Bye";
	}
	return "Ciao";
}
```

Exercise

What output is produced when the `start()` method is called?

```
private void start() {
	for(int i = 0; i < 3; i++) {
		for(int j = 0; j < 5; j++) {
			if(j%2 == 0)
				continue;
			if(i%2 == 1)
				break;
			int result = calculate(i*j);
			System.out.println(result);
		}
	}
}

private int calculate(int n) {
	if(n%3 == 0)
		return 3;
	return 1;
}
```


Exceptions

An Exception is an abnormal condition that arises when you program is running. It causes the normal flow of control to be disrupted

The Java `Exception` class contains information about the kind of problem that occurred, why it occurred and where it occurred.

Exceptions come in two types

- Checked
- Unchecked

What kind of Java `Exceptions` do you already know of?

Unchecked Exceptions

Programmers do not have to handle the exception

- May not even know the code may generate an exception
- These are ones you are familiar with

RuntimeException

- Unchecked
- All subclass of RuntimeException are unchecked

Examples

- ArithmeticException
- ArrayIndexOutOfBoundsException
- NullPointerException
- NumberFormatException
- StringIndexOutOfBoundsException

Flow of control for Exceptions

Breaks the normal flow of control

Throwing an exception

- Code stops executing
- Create an Exception object
- Give the object to the runtime system

Handling the exception

- Runtime system searches for a block of code that can handle the exception
- Starts with the method that generated the exception
- Proceeds through the method call stack from the most recent called method
- When a handler is found, the runtime system passes the exception to the handler
- This block of code is called an "exception handler"
- Code starts executing from the handler onwards

Catching the exception

Block of code that handles the Exception is called the exception handler.

- Specify which kinds of exceptions it handles

Catching an exception

- If the type of Exception that the handler accepts matches the Exception thrown, then it will "catch" the exception and deal with it.
- If the runtime system searches all the methods on the call stack without finding an appropriate exception hander, then the program is terminated.

```
For Example,
Exception in thread "main" java.lang.ArithmeticException:
/ by zero
        at A1Q2Program.test01(A1Q2Program.java:40)
        at A1Q2Program.start(A1Q2Program.java:16)
        at A1Q2.main(A1Q2.java:13)
```


Syntax: try… catch

`try` block

- Includes the code that may generate an exception

`catch` block

- specify the kind of exception that we will handle
- Include the code that handles the exception (returns the program to a safe state).

```
try {

       //code which may cause the Exception

} catch( ExceptionWhichIsToBeHandled e ) {

       //code which returns the program to a safe state

}
```


NumberFormatException

An Exception could be generated by this code

- What line of code might generate an exception?
- When will the exception be generated?
- What happens when the exception is generated?

```
private void tryCatchEx01() {
    int num = 0;
    try {
        System.out.print( "Enter number: " );
        num = Integer.parseInt( Keyboard.readInput() );
        System.out.println( "Thank you" );

    } catch ( NumberFormatException e ) {
        System.out.println( "User error" );
        num = -1;
    }

    System.out.println( "Number: " + num );
}
```


ArrayIndexOutOfBoundsException

Consider how the exception handling code works in this example

- Exactly which statements are executed and which are not

```
private void tryCatchEx02() {
       int result;
       int[] nums = { 2, 3, 4, 2, 4 };

       try {

              result = nums[ nums.length ];
              System.out.println( "See you" );

       } catch( ArrayIndexOutOfBoundsException e ) {

              System.out.println( "Index error" );
              result = -1;

       }
       System.out.println( "Result: " + result );
}
```

NullPointerException

```
private void tryCatchEx03() {
	int result;
	int[] nums = null;

	try {

		result = nums.length;
		System.out.println( "See you" );

	} catch( NullPointerException e ) {

		System.out.println( "Object is null" );
		result = -1;
	}
	System.out.println( "Result: " + result );
}
```


Exercise

Add a try … catch block to this code

```
private void tryCatch01() {
	int number = 100;
	int result;


	result = number / (number - number);



	System.out.println( "Exception occurred" );



	System.out.println( "Number: " + result );

}
```


Exercise

What happens when we run this method?

```
private void tryCatch02() {
	int result = 0;
	int[] nums = null;

	try {
		result = nums.length;
		System.out.println( "See you" );

	} catch( ArithmeticException e ) {
		System.out.println( "Problem" );
		result = -1;
	}
	System.out.println( "Result: " + result );
}
```


Exercise

Rewrite the following code so that it uses a try… catch block to handle the exception.

```
private void tryCatch03() {
	int result = 0;
	String[] items = { "one", "two" };

	result = items[2].length();
	System.out.println( "Result: " + result );
}
```

Exercise

What is the problem with this code?

```
private void problemEx01() {
	try {
		int num = 0;
		System.out.print( "Enter number: " );
		num = Integer.parseInt( Keyboard.readInput() );
		System.out.println( "Thank you" );
	} catch( NumberFormatException e ) {
		System.out.println( "Input error" );
		num = -1;
	}

	System.out.println( "Number: " + num );
}
```


Exercise

What is the problem with this code?

```
private int problemEx02() {
	int result = 0;
	int[] nums = { 2, 3, 4, -1, 4 };
	try {
		result = nums[ nums[3] ];
		System.out.println( "See you" );
	} catch( StringIndexOutOfBoundsException e ) {
		System.out.println( "Index error" );
		result = -1;
	}
	return result;
}
```


Exercise

What is the problem with this code?

```
private void problemEx03() {
	int result = 0;
	int[] nums = { 2, 3, 4, -1, 4 };

	try {
		result = nums[ nums[3] ];
		System.out.println( "See you" );
	} catch {
		System.out.println( "Number error" );
		result = -1;
	}
	System.out.println( "Result: " + result );
}
```


Exercise

Correct this code by adding an appropriate catch block.

```
private void problemEx04() {
	int result = 0;
	int[] nums = { 2, 3, 4, -1, 4 };

	try {
		result = nums[ nums[3] ];
		System.out.println( "See you" );
	}
	System.out.println( "Result: " + result );
}
```

Multiple `catch` blocks

The `catch` block is the actual handler for the exception.

A `try` block can have multiple catch clauses associated with it, one for each type of exception which can occur in the try block.

The Java runtime attempts to match the thrown exception to a `catch` block in the order they appear in the code.

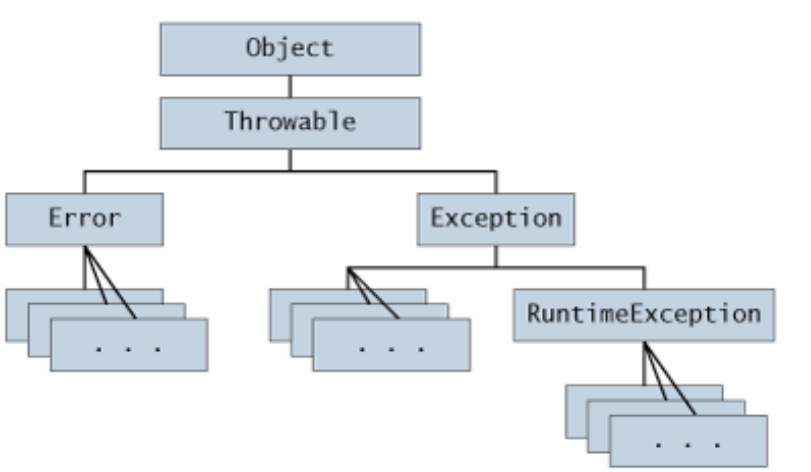
Example

Two possible exceptions generated by this code

- Handle each exception separately

```
private void tryCatchEx01() {
	int num = 0;
	try {
		System.out.print( "Enter number: " );
		num = Integer.parseInt( Keyboard.readInput() );
		num = 100 / num;
		System.out.println( "A" );
	} catch( NumberFormatException e ) {
		System.out.println( "User error" );
		num = -1;
	} catch( ArithmeticException e ) {
		System.out.println( "Division by 0" );
		num = -1;
	}
	System.out.println( "Number: " + num );
}
```


Java Exception class hierarchy



Exception hierarchy

Exceptions higher up in the class hierarchy will catch any subclass exception

- `Exception` is the highest type of exception, so it catches all other exceptions

```
private void tryCatchEx02() {
	int num = 0;
	try {
		System.out.print( "Enter number: " );
		num = Integer.parseInt( Keyboard.readInput() );
		num = 100 / num;
		System.out.println( "A" );
	} catch( Exception e ) {
		System.out.println( "User error" );
		num = -1;
	}

	System.out.println( "Number: " + num );
}
```

Exercise

What is the problem with this code?

```
private void tryCatchEx03() {
	try {
		…
	} catch( Exception e ) {
		System.out.println( "Error A" );
	} catch( ArithmeticException e ) {
		System.out.println( "Error B" );
	}
}
```


Exercise

Rewrite the following code so that it handles any exceptions correctly

```
private void tryCatchEx01() {
	String[] data = { "A", "B", "C", null};
	System.out.print( "Which element: " );
	int num = Integer.parseInt( Keyboard.readInput() );
	String value = data[num];
	System.out.println( "Element contains: " + value );
	System.out.println( "Length: " + value.length() );
}
```