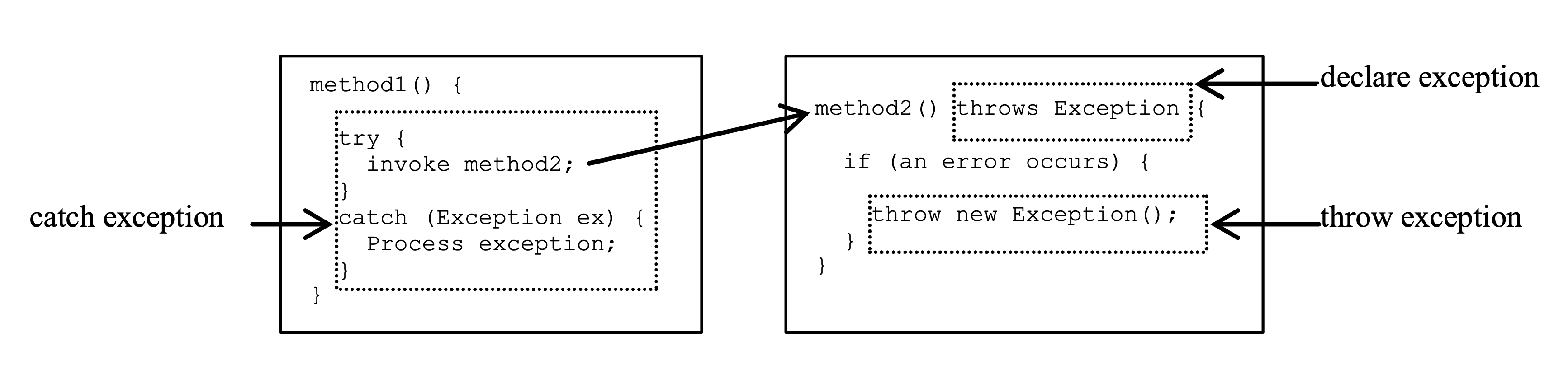
**Exception management:**

- Exceptions are used to manage run-time errors. We use try-catch for that matter.



- throw new TheException();   
TheException ex = new TheException();  
throw ex

- try {

statement;

}

catch (Exception exVar1) {

Handle1;

}

finally {

finalStatement; //This Block always executed.

}

**Files**:

* In Java, files are treated as sequential streams of bytes. Operating systems have mechanisms that determine each file break with an end-of-file-marker.
* Byte-based streams, input and output are in binary format. One char 2 bytes, one int 4 bytes, one double 8 bytes, etc.
* Character-based streams, input and output are in character format. Each character is 2 bytes. The number of bytes depends on the number of characters in a value. For example, 10 bytes for 20000 (5 characters, 2 bytes per character)
* The file created with byte-based streams is called a binary file, the file created with character-based streams is called a text file.
* Each file ends with an end-of-file marker that indicates the end of the file.   
  When a file is opened, a stream is associated with the file.  
  When a program starts running, 3 types of streams are automatically opened:

standard input (System.in),

standard output (System.out)

standard error (System.err)

* the java.io.\* library is used for files

**Multiple Choice Questions:**

1. Which of the following statements is true regarding Java exceptions?

A. An exception can only be thrown by a method, not caught.

B. Exceptions should always be caught and handled in the same method where they are thrown.

C. A try block can only have one catch block.

D. A catch block can catch multiple exceptions by separating them with a comma.

Answer: D. A catch block can catch multiple exceptions by separating them with a comma.

1. What will happen if you attempt to catch an exception that is not thrown?

A. The code will compile, but will not run.

B. The code will run, but will not catch any exceptions.

C. A compile-time error will occur.

D. A runtime error will occur.

Answer: A. The code will compile, but will not run.

1. Which of the following methods can be used to create a new file in Java?

A. file.createNewFile()

B. file.createFile()

C. file.makeNewFile()

D. file.makeFile()

Answer: A. file.createNewFile()

1. Which of the following statements is true regarding Java file handling?

A. A file can only be read sequentially from beginning to end.

B. A file can only be written sequentially from beginning to end.

C. A file can be read and written randomly using a file pointer.

D. A file can only be accessed if it is located in the same directory as the Java program.

Answer: C. A file can be read and written randomly using a file pointer.

**Practice Questions:**

**1.**

Write a Java program that reads a text file named "input.txt" and counts the number of words in it. Then, the program should create a new file named "output.txt" and write the word count to this file.

Assume that each word in the input file is separated by a space and there is no punctuation.

Here's a sample “input.txt” file:

“This is a sample input file for the coding question about files.”

Here's the expected output in the "output.txt" file:

“Number of words: 12”

Note: You can assume that both "input.txt" and "output.txt" files are located in the same directory as the Java program.

import java.io.File;  
import java.io.FileNotFoundException;  
import java.io.PrintWriter;  
import java.util.Scanner;  
  
public class Main {  
  
 public static void main(String[] args) {  
 // Open the input file that is next to the class file  
 File inputFile = new File("src/input.txt");  
 Scanner scanner = null;  
 try {  
 scanner = new Scanner(inputFile);  
 } catch (FileNotFoundException e) {  
 System.*err*.println("Error: The input file was not found.");  
 System.*exit*(1);  
 }  
  
 // Count the number of words in the input file  
 int wordCount = 0;  
 while (scanner.hasNext()) {  
 scanner.next();  
 wordCount++;  
 }  
  
 // Close the input file  
 scanner.close();  
  
 // Write the word count to the output file  
 File outputFile = new File("src/output.txt");  
 PrintWriter writer = null;  
 try {  
 writer = new PrintWriter(outputFile);  
 } catch (FileNotFoundException e) {  
 System.*err*.println("Error: Unable to create the output file.");  
 System.*exit*(1);  
 }  
  
 writer.println("Number of words: " + wordCount);  
  
 // Close the output file  
 writer.close();  
 }  
}

**2.**

Write a Java program that reads an integer from the user and prints its factorial. If the input is negative, the program should throw an IllegalArgumentException and print an error message instead of calculating the factorial.

Here's an example of the program output when the user enters a positive integer:

Enter an integer: 5

Factorial: 120

Here's an example of the program output when the user enters a negative integer:

Enter an integer: -5

Error: The input must be non-negative.

Note: You should use a try-catch block to handle the IllegalArgumentException.

import java.util.Scanner;  
  
public class Factorial {  
 public static void main(String[] args) {  
 Scanner scanner = new Scanner(System.*in*);  
 System.*out*.print("Enter an integer: ");  
 int n = scanner.nextInt();  
 scanner.close();  
  
 try {  
 int factorial = *calculateFactorial*(n);  
 System.*out*.println("Factorial: " + factorial);  
 } catch (IllegalArgumentException e) {  
 System.*err*.println("Error: " + e.getMessage());  
 }  
 }  
  
 public static int calculateFactorial(int n) {  
 if (n < 0) {  
 throw new IllegalArgumentException("The input must be non-negative.");  
 }  
  
 int factorial = 1;  
 for (int i = 2; i <= n; i++) {  
 factorial \*= i;  
 }  
  
 return factorial;  
 }  
}