Exception Handling Exercises

1. Write a java program using multiple catch blocks. Create a class CatchExercise inside the try block declare an array a[] and initialize with value a[5] =30/5; . In each catch block show Arithmetic exception and ArrayIndexOutOfBoundsException.

    Test Data:
    a[5] =30/5;
    Expected Output :
    ArrayIndexOutOfBoundsException occurs
    Rest of the code
2. Create a program to ask the user for a real number and display its square root. Errors must be trapped using "try..catch".
    Sample Output:
    Enter a real number: 25
    The square root of 25.0 is 5.0
    Enter a real number: -9
    Cannot compute the square root of a negative number.
    Enter a real number: abc
    Error: Invalid input. Please enter a valid real number.
3. Write a Java program that throws an exception and catch it using a try-catch block.
    Sample Output:
    Exception caught: This is a custom exception message!
4. Write a Java program to create a method that takes an integer as a parameter and throws an exception if the number is odd.
    Sample Output:
    The number 7 is odd!
    The number 8 is even.
5. Write a Java program to create a method that reads a file and throws an exception if the file is not found.
    Sample Output:
    Error: The file 'sample.txt' was not found.
6. Write a Java program that reads a list of numbers from a file and throws an exception if any of the numbers are positive.
    Sample Output:
    Suppose you have a file named numbers.txt with the following content:
    -5
    -2
    3
    -8
    0
    When you run the program, it will throw an exception as soon as it encounters the positive number 3 and print:
    Error: The file contains a positive number: 3
    If the file only contains non-positive numbers, it will print:
    The file contains only non-positive numbers.
7. Write a Java program that reads a file and throws an exception if the file is empty. If the file has no content, the program will throw a custom exception indicating that the file is empty.

Exception Handling Exercises

Sample Output:

- If the file sample.txt is **empty**, the program will output:
  Error: The file is empty: sample.txt

- If the file is **not empty**, it will print the content of the file line by line.
- If the file does not exist, the output will be:
  Error: The file 'sample.txt' was not found.

Example Files:

1. **Non-empty file (sample.txt)**:
   Hello, world!
   This is a test file.

   Output:
   Hello, world!
   This is a test file.
   File read successfully!

2. **Empty file (empty.txt)**: (File has zero length, no content)

   Output:
   Error: The file is empty: empty.txt

3. **File not found (nonexistent.txt)**:

   Output:
   Error: The file 'nonexistent.txt' was not found.

8. Write a Java program that reads a list of integers from the user and throws an exception if any numbers are duplicates.
   Sample Output:
   ```
   Enter integers (enter a non-integer to stop):
   Enter a number: 10
   Enter a number: 20
   Enter a number: 30
   Enter a number: 20
   Duplicate number found: 20
   ```
9. Write a Java program to create a method that takes a string as input and throws an exception if the string does not contain vowels.

   Sample Output:
   ```
   The string 'rhythm' does not contain any vowels.
   The string 'hello' contains vowels.
   ```

Exception Handling Exercises

10. *(Catching Exceptions with Superclasses)* Use inheritance to create an exception superclass (called ExceptionA) and exception subclasses ExceptionB and ExceptionC, where ExceptionB inherits from ExceptionA and ExceptionC inherits from ExceptionB. Write a program to demonstrate that the catch block for type ExceptionA catches exceptions of types ExceptionB and ExceptionC.
11. *(Catching Exceptions Using Class Exception)* Write a program that demonstrates how various exceptions are caught with catch (Exception exception) This time, define classes ExceptionA (which inherits from class Exception) and ExceptionB (which inherits from class ExceptionA). In your program, create try blocks that throw exceptions of types ExceptionA, ExceptionB, NullPointerException and IOException. All exceptions should be caught with catch blocks specifying type Exception.
12. *(Order of catch Blocks)* Write a program that shows that the order of catch blocks is important. If you try to catch a superclass exception type before a subclass type, the compiler should generate errors.
13. *(Constructor Failure)* Write a program that shows a constructor passing information about constructor failure to an exception handler. Define class SomeClass, which throws an Exception in the constructor. Your program should try to create an object of type SomeClass and catch the exception that’s thrown from the constructor.
14. *(Rethrowing Exceptions)* Write a program that illustrates rethrowing an exception. Define methods someMethod and someMethod2. Method someMethod2 should initially throw an exception. Method someMethod should call someMethod2, catch the exception and rethrow it. Call someMethod from method main, and catch the rethrown exception. Print the stack trace of this exception.
15. *(Catching Exceptions Using Outer Scopes)* Write a program showing that a method with its own try block does not have to catch every possible error generated within the try. Some exceptions can slip through to, and be handled in, other scopes.