Exception Handling

Exercises and solutions

1. What is an exception in Java? Name the two types of exceptions that Java supports.

**Answer:**

An exception is a condition that may arise during the execution of a Java program when a normal path of execution is not defined. Two types of exceptions supported by Java are checked exceptions and unchecked exceptions.

1. What is the superclass of all exception classes in Java?

**Answer:**

java.lang.Throwable

1. What type of statement/block would you use to place your code if a piece of code may throw an exception?

**Answer:**

A try-catch block.

1. How many exceptions can you catch in one catch block?

**Answer:**

One catch block can be used to handle multiple types of exceptions. There is no limit on the number of exceptions you can handle in one catch block.

1. Can you throw an exception from inside a catch block?

**Answer:**

Yes.

1. Name two constructs in Java that you may use to clean up resources after their use?

**Answer:**

fry-finally block and try-with-resources block.

1. What are checked and unchecked exceptions in Java? Is java.lang.ArithmeticException a checked exception? Is java.io.IOException a checked exception?

**Answer:**

The compiler makes sure that all checked exceptions are handled in the program or program (methods and constructors) declares them using a throws clause. All exceptions that are not checked exceptions are called unchecked exceptions. Handling or declaring unchecked exceptions is optional. java.lang.ArithmeticException is not a checked exception; it is a checked exception. Java.io.IOException is a checked exception.

1. What keyword do you use to in a method's declaration to declare that the method throws an exception?

**Answer:**

throws

1. What keyword do you use to throw an exception?

**Answer:**

throw

1. Will the following statement compile?  
     
   throw null;  
     
   If this statement compiles, what will happen when it is executed?

**Answer:**

Yes. This statement compiles fine. When it is executed, it throws a NullPointerException.

1. Can you throw a runtime exception from a method without specifying the exception in the throws clause of the method's declaration?

**Answer:**

Yes

1. Will the following method declaration compile? If no, describe the reason.  
     
   public void test() {  
    throw new RuntimeException("An error has occurred.");  
    System.out.println("Everything is cool!");   
   }

**Answer:**

No. Compilation fails because the second statement, System.out.println(“Everything is cool!”);, inside the test() method is an unreachable statement.

1. Complete the following snippet of code, so the error message associated with the exception is printed on the standard output.  
     
   try {  
    int x = 100 / 0;  
   } catch (ArithmeticException e) {  
    String errorMessage = e./\* You code goes here \*/;  
    System.out.println(errorMessage);  
   }

**Solution:**

try {  
 int x = 100 / 0;  
} catch (ArithmeticException e) {  
 String errorMessage = e.**getMessage()**;  
 System.out.println(errorMessage);  
}

1. What method of the Throwable class do you use to print the stack trace of the exception object?

**Answer:**

printStackTrace()

1. Describe the reason why the following try-catch block does not compile.  
     
   try {  
    // The following statement throws NumberFormatException  
    int luckNumber = Integer.parseInt("Hello");  
   } catch (Exception e) {  
    // Handle the exception here  
   } catch (NumberFormatException e) {  
    // Handle the exception here  
   }

**Answer:**

Multiple catch blocks for a try block must be arranged from the most specific exception type to the most generic exception type. Exception is the superclass of NumberFormatException, so the first catch block will also handle the NumberFormatException. You get a compile-time exception – NumberFormatException has already been caught.

1. Consider the following code inside a method, assuming that MyResource is a class that implements the AutoCloseable interface. The code does not compile. Describe the reason why the code does not compile and fix it, so it compiles.  
     
   MyResource res = new MyResource(1, false);  
   try (res) {  
    res.use();  
   }  
   res = null;

**Answer:**

The variable res is used in a try-with-resources block and it is neither final nor effectively final. You need to comment the last statement that assigns null to the res statement. This way, the res variable becomes effectively final. Optionally, you can also declare the res variable final.

MyResource res = new MyResource(1, false);  
try (res) {  
 res.use();  
}  
//res = null;