Task: Exception Handling

www.hyperiondev.com

Introduction

Figuring out where it all went wrong!

In this task you are going to learn about how to handle exceptions and problems in your code. This is essential when debugging and fixing problems you have with your programs. As your programs get more intense you will have more problems. Often more time is spent on debugging them actually writing the code in really big programs. Knowing how to find these problems and fix them is an important part of programming. This task will teach you how to save a lot of time when coding by teaching you how to identify issues easily and fix them efficiently. Let's get started!

-The Hyperion Team

A note from the Hyperion Team...

Learning from your mistakes
There are two types of exceptions – checked and unchecked. A checked exception is usually an exception which is not caused by fault of the programmer, but rather by unforeseen circumstances. For example, if your program connects to the Internet, but there is no internet connection available then a problem will occur. These exceptions are generally anticipated by a well-written program and handled appropriately. In the example above, the program could, for example, enter an offline mode.

All exceptions are checked exceptions unless they are Errors or Runtime Exceptions. All checked exceptions must be thrown or caught by a try-catch statement. A well-written program should handle all checked exceptions to the extent that they do not alter the output of the program in an unexpected way.

Note: *If you have a checked exception which you have not handled, Java will usually not let you compile and run your program until you have handled the exception.*

Unchecked exceptions can be further separated into two categories – compile time and runtime exceptions. Both of these are problems which will cause your program to terminate. Generally, unchecked exceptions are not required to be handled. It is expected for your program to fail (particularly with errors).

Compile time exceptions are mainly caused by syntactical and what is known as static semantic errors. This means that it is not to do with the logic of the program, but more the language. For example, if you were to write "If" instead of "if" this would be a compile time error as the keyword is spelt "if". The analyser goes through the program first to check if it encounters any errors before trying to actually run the program.

Runtime exceptions are errors generally caused by logical problems. For example, an ***ArithmeticException*** is caused by dividing by zero. This is (normally) the fault of the programmer. These types are exceptions are often caused by logical errors or by misuse of a certain method. Runtime errors will cause your program to crash and mean that the errors have to be fixed by changing code. An ***ArithmeticException*** is illustrated below:

```
int number = 10/0;
```

Here is what the output of an exception looks like:

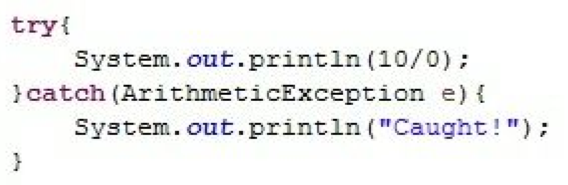
```
Exception in thread "main" java.lang.ArithmeticException: / by zero
        at Sort.t(Sort.java:9)
        at Sort.main(Sort.java:20)
```

Notice that the exception is highlighted in blue and underlined. The specific problem is specified after the colon (in this case '/ by zero'). The lines which follow specify where the exception occurs (which line specifically) as well as which function it occurs in. If the method which has the exception is called from multiple places the exception will occur multiple times. This does not mean that your code has to be changed in multiple places - the problem just has to be fixed at its root.

In the example above, the exception occurs in the method 't' as well as in the 'main' method. This is because the method 't' is called in the main method. The problem only has to be fixed in the method 't' and not in the main method. There will be times when there will be hundreds of lines like this. Knowing how to trace the problem back to its root is key to solving problems efficiently!

Now that you understand the different types of problems let's figure out how to deal with them. Let's go through the use of a try-catch statement.

Try-catch
A try-catch statement is one which encapsulates a block of code and handles a specific exception in a manner specified by you. Below is an example of a try-catch statement used on an unchecked exception:



```
try{
    System.out.println(10/0);
}catch(ArithmeticException e){
    System.out.println("Caught!");
}
```

Notice how the catch part is after the block of code which is being 'tried'. You essentially attempt to execute the code then if your code has a problem which you expect (as specified in the catch statement), then you handle it within the curly brackets after the catch statement. The statement above attempts to divide by 0 giving an arithmetic exception. This is caught in the catch statement and an appropriate statement is printed out ('Caught!').

There may be times when you might not be sure what type of exception to catch, so instead of specifying the type of exception (such as ArithmeticException) you can simply write Exception e which creates an object called e containing the details of the exception. You can print this out in a similar manner to the example above by using System.out.println(e) which will give you more detail as to what the exception is.

Note: *You can use 'System.**err**.println()' to print out a statement in red so it can be easily identified.*

We now need to cover the use of the **throws** statement. A **throws** statement is used when you do not want to handle an exception in a specific way as with try-catch statements. Any checked exceptions need to be thrown, however, some unchecked exceptions may be thrown as well. If a method is used within one of your methods and that method has certain exceptions that is throws, then these exceptions need to be handled either with a try-catch statement or with a **throws** statement. The **throws** statement does not change the way the code is executed, but is necessary when exceptions are not being handled by try-catch statements. Below is an example of a **throws** statement:

```
public void t() throws ArrayIndexOutOfBoundsException{
    int [] n = new int[3];
    n[8] = 9;
}
```

Note the placement of the word ‘throws’. Also, multiple exceptions may be thrown. If this is done, the exceptions are separated by commas. The code above attempts to access a position in an array which is larger than the size of the array. This produces the ‘ArrayOutOfBoundsException’ which is then thrown. Keep in mind that this is an unchecked exception so, in this case, the **throws** statement is unnecessary, but it is, nevertheless, an example of the usage of the **throws** statement.

Instructions

First read example.java, open it using jGRASP (Right click the file and select ‘Open with jGRASP”).

- In this folder there is a file called example.java
- Open this folder using the JGRASP program. You can either do this by right clicking on the example.java file, going to the ‘Open with’ menu, and selecting JGrasp.exe. Alternatively, you can run the JGRASP program on your computer, go to the top left corner of the program, select File->Open and navigate to example.java on your hard drive and double click on it to open it.
- Once example.java is open in JGRASP please read all of its content very carefully. This will help you understand the basic structure of a Java program.
- There is a compulsory task at the end of the example.java document. The instructions of what to do to complete the task can be found here. You must complete this exercise to continue onto the next task of this course.

Compulsory Task

Follow these steps:

- Create a new java file called ExceptionHandling.java
- Create a class called 'ExceptionHandling'
- Within this class, create a main method.
- Outside the main method, create a method called 'printOneByOne' which takes one String as a parameter. This method should iterate from 0 to 100 and print out the character of the String at these positions.
- Handle any possible exceptions with a try-catch statement. If an exception occurs, print out 'Caught the exception!'
- Within the main method, create an instance of the class and call the 'printOneByOne' method with the parameter 'EXCEPTION'.
- Compile, save and run your file.

Things to look out for

1. Make sure that you have installed and setup all programs correctly. You have setup **Dropbox** correctly if you are reading this, but **jGRASP or Java** may not be installed correctly.
2. If you are not using Windows, please ask your tutor for alternative instructions.

Still need help?

Just write your queries in your comments.txt file and your tutor will respond.

Task Statistics

Last update to task: 03/02/2016.
Authors: Jared Ping and Tason Matthew.
Main trainer: Umar Randeree.
Task Feedback link: Hyperion Development Feedback.