**Exceptions Lab**

1. To demonstrate that “finally” **is always executed** regardless of whether or not an exception occurs in the try block.
   1. Code a method named *testMethodA*; return type String.
   2. In the method, code a try block that does not throw an exception. Code a catch for *Exception* and a *finally* block that outputs “finally section…”.
   3. Return “ok from finally” from the *finally* section.
   4. Call *testMethodA* from *main* and output the string returned.
2. To demonstrate that “finally” is executed even when you “return” from try block.
   1. Code a method named *testMethodB*; return type String.
   2. In the method, code a *try* block that returns “ok”. Code a catch for *Exception* and a *finally* block that outputs “finally section…” (no return statement in finally this time).
   3. Call *testMethodB* from *main* and output the string returned.
3. To demonstrate that “finally” is executed even when you “return” from exception handler.
   1. Code a method named *testMethodC*; return type *String*.
   2. In the method, code a *try* block that simply throws an *Exception* (only for demonstration purposes – you should never throw an *Exception* when you don’t actually have an error). Catch the exception within the method and output the toString() from the exception object. The last line of the exception handler is to return “exception”. The finally block is as in step 2 above.
   3. Call *testMethodC* from *main* and output the string returned.
4. To demonstrate that the (checked) exceptions you are trying to catch must be thrown in the *try* block you are checking. This step is about compilation and not runtime.
   1. Code a method named *testMethodD*; return type *String*.
   2. In the method, code a *try* block that simply returns “ok”. However, try and catch the following (checked) exceptions: *IOException*, *NoSuchMethodException* and *ClassNotFoundException*. Will they compile? why or why not? Catch *RuntimeExeption* – will it compile? why or why not? Catch *Exception* – will it compile? why or why not?
5. To demonstrate that what you return from *finally* replaces what you return from the exception handler.
   1. Code a method named *testMethodE*; return type *String*.
   2. In the method, code a *try* block that declare a String array with two strings in it “John” and “Peter”. Attempt to access index [55] of the array (this will cause an *ArrayIndexOutOfBoundsException*). The last line of the *try* block is to return “ok” (this will never be executed).
   3. The exception handler should output the *toString* of the exception object and then return “exception”.
   4. The *finally* block should oputput “finally section…” and return “ok from finally”.
   5. Call *testMethodE* from main and output the string returned.
6. To show how to throw exceptions and then catch them in *main*.
   1. Code a method *testMethodF*; return type *void*.
   2. This method throws an *Exception* (include this fact in the method signature).
   3. In the method throw an *Exception* (again, this is only for demonstration purposes – you should never throw an *Exception* when you don’t actually have an error).
   4. Catch the exception in *main* and print the stack trace.
7. Not the HDip.   
   This example is all about compilation and not runtime.
   1. Code a class *A* that has a method *process*() that throws an *IOException*
   2. Code a class *B* that inherits from *A*. The class *B* overrides the *process*() method inherited from *A* and it throws *IOException* and *NoSuchMethodException* (both are **checked** exceptions). Do you get an error? Why or why not? If you get an error, what are your options to fix this error?
   3. In *main*() create a new reference of type *A* that refers to an object of type *B*. Invoke *a.process*() – while the method executed will be in class *B* (via polymorphism, the signature is determined via the type of the object reference executing the method i.e. *A*). Thus, the Exceptions to catch are determined by the (static) reference type.
8. This example demonstrates the impact of throwing ***checked*** exceptions
   1. Code four methods *X1, X2, X3 and X4*. These methods take no parameters and return nothing. In *main*() call *X1*; *X1*() then calls *X2*(); *X2()* calls *X3*() and *X3* calls *X4*(). In *X4*() throw a *ClassNotFoundException* (for demo purposes). This exception is to be caught in *main*() only. What is the impact?
9. This example demonstrates the “impact” of throwing ***unchecked*** exceptions
   1. Code four methods *Y1*, *Y2*, *Y3* and *Y4*. These methods take no parameters and return nothing. In *main*() call *Y1*; *Y1*() then calls *Y2*(); *Y2*() calls *Y3*() and *Y3* calls *Y4*(). In *Y4*() throw a *RuntimeException* (for demo purposes). This exception is to be caught in *main*() only. What is the impact?
   2. Note that in *main(), Y1*() with **no** surrounding *try* catch block will compile…
10. Create the custom exception class *MyException*.
    1. Call method *a*() from *main()*; *a*() calls *b*() and *b*() calls *c*().
    2. In *c*() throw an *IOException* with the message “*An I/O exception occurred*”. Note that, the signature of *c*() says that it “*throws MyException*” (and not *IOException*).
    3. In *b()*, catch the *MyException* generated by *c*() and in this exception handler, output the exception caught i.e. *MyException* and details of the original exception that was generated in *c*() i.e. *IOException*.