EECS1720
Worksheet 1 – Exceptions

1) Which types of expressions may raise exceptions?

   a) class instance creation expressions e.g.,
      `new MyClass();` -> if no memory available
   b) method invocation expressions e.g.,
      `myObject.myMethod(arg);` -> if myObject null (or not instantiated)
   c) expressions with integer division or integer remainder with a zero divisor e.g.,
      `78 % 0` -> % 0 implies divide by zero occurs before calculating remainder

   d) **all of the above**

2) Suppose an expression potentially raises an exception and you wish to add exception handling. What are your options?

   a) use a `try-catch` block
   b) use a `throws` expression in the method header
   c) neither A nor B; just hope for the best
   d) **either A or B; either is an exception handler.** -> in (b) exception is expected to be handled elsewhere (i.e. in calling method)

3) Suppose an exception is raised at run-time and my code is set up to handle the exception in a catch block.

```
catch (RuntimeException  theCaughtException) {
   // here is the code to handle the exception,
   // the variable theCaughtException will hold
   // the reference to the exception object
}
```

   What things can I do with this object reference?
   a) there are no possible actions because it is a variable
   b) there are some actions, but it a mystery and there is no way to tell
   c) **these are some possible actions, to understand look in the API** -> need to look at API of RuntimeException class
   d) there are infinite possible actions, your imagination is the limit

4) Given two example applications.. In one example, the compiler does not give a compile time error. For the other, the compiler does give a compile-time error. Why is this?

a) The compiler is unpredictable;
b) It might work differently another time and we should try again later
c) The compiler is enforcing a rule that is conditional; it depends on whether the expression is a method invocation or an instantiation
d) **The compiler is enforcing a rule that is conditional; it depends on the type of the exception being thrown**
e) None of the above

-> compiler only acts if exception is checked.. Unchecked exceptions are dealt with by user at Runtime (handled) or dealt with by JVM otherwise (crash)

5) Consider the following code segment (you may need to refer to the Java API to answer)

```
output.println ("Enter a fraction (x/y) and I will give you the
quotient");
String str = input.nextLine();
int slash = str.indexOf("/") ;
String left = str.substring(0, slash);      ──> can throw IndexOutOfBoundsExeception
String right = str.substring(slash + 1);
int numer = Integer.parseint(left);         ──> can throw NumberFormatExeception
int denom = Integer.parseint(right);
int quotient = numer/denom;                 ──> can throw ArithmeticExeception
output.println("Quotient = " + quotient);
```

How many different exceptions could potentially be raised?

a) 0
b) 1
c) 2
d) **3**

6) A compiled program consists of a series of bytecode instructions. One of these instructions may be invalid, even though the compiler did not issue an error.

a) TRUE
b) FALSE

-> e.g. variable might hold a value used in an int division.. at runtime the value may cause divide by zero, this is not caught at compile time because we dont know the state of the variable

7) If an app that is invoked using in one particular running environment (e.g. on one computer) **does not crash**, then that same app, when invoked in some other running environment (i.e. another different computer), will also **not crash**.

a) TRUE
b) FALSE

-> some exceptions are thrown based on the environmental state (e.g. out of memory, etc)

8) Thrown exceptions are like run-time errors: they are bad and a sign that something went wrong.

a) TRUE
b) **FALSE**

-> not bad, since they can be caught and handled gracefully (with a clean exit). Run time errors cause program crash which can result in loss of data, corruption of a file, etc..

9) What is a crash?

a) When the compiler issues an error due to syntactic problems
b) **When the JVM terminates at run-time because an exception was never handled**
c) When the app is incorrect due to logical errors
d) When the flow of control in an app reaches the end of the main method

10) Why doesn't the compiler check for run-time errors and prevent them from happening?

a) The compiler *does* check for run-time errors. This is a trick question!
b) The compiler does not check for run-time errors because this checking is not specified in the Java Language Specification.
c) **The compiler does not check for run-time errors because this is not possible, from both a practical and theoretical perspective.**
d) The compiler does not check for run-time errors because it is desired for programmers to develop stronger design skills.

-> it is impossible to know the state of an object/variable at compile time (that may lead to an error), nor the state of the environment (computer resources, like available memory, or if a file exists, or is corrupted/not)..

11) What is the typical strategy for building exception handling into your code?

Assume that any inputs to the code observe all necessary pre-conditions (i.e. there are no errors).. then test with inputs that do have errors, see what exceptions are caused, then build handlers accordingly)

12) What is the difference between an Error and an Exception?

An error is usually external to the application and represents a condition the program cannot usually recover from, while an exception may internal/external to the application and generally may be recovered from (so that the program can still keep excecuting)

13) Explain how the JVM deals with exceptions at runtime

The JVM will consider any exception only if not caught by the program and dealt with via a handler. If the exception is not handled, the JVM treats it as a run-time error, and crashes the program (unclean exit)