CMPS 12A
Introduction to Programming
Spring 2010

Programming Assignment 2

Due Saturday April 17, 10:00 pm

In this project you will make some minor alterations to programming assignment 1 to make it more robust with respect to user input. The updated `lawn.java` program will check its input for correct formatting and logical consistency, and quit with informative error messages if certain conditions are not satisfied. The name of the altered program will be `lawn2.java`. Start by creating a copy of `lawn.java` from `pa1` called `lawn2.java`, then alter it to implement the functionality described below.

Java provides a special mechanism for handling situations that depart from the expected normal program operation. In such cases we say that the program "throws an Exception". An Exception is a special type of class that, when "thrown", can have the effect of shutting down the program, or otherwise altering the normal flow of execution, depending on whether or not the Exception is "caught". The java standard libraries contain Exception classes designed to handle many different common error states. The programmer can also tailor new Exception classes to fit the special needs of a program. A full discussion of the various Exception classes and their uses must wait until we have learned more about java classes. After all, we still don't quite know what a class is. At this point however, there is one type of catch-all Exception class that we can easily use, called a `RuntimeException`. When this Exception is "thrown" (and not "caught" elsewhere in the program), it first prints an error message specified by the programmer, then shuts down the program. Its general usage is

```
if( abnormal_situation )
   throw new RuntimeException( error_message );
```

In this generic example, `abnormal_situation` is a boolean expression, and `error_message` is a string expression (usually a string literal) giving useful information to the user as to what went wrong. The following example does a "safe" division of two doubles entered by the user.

```
// div.java
import java.util.*;
class div{
   public static void main( String[] args ){
      Scanner sc = new Scanner(System.in);
      double x, y, q;

      System.out.print("Enter the numerator and denominator: ");

      if( sc.hasNextDouble() ) x = sc.nextDouble();
      else throw new RuntimeException("Numerator must be a number.");

      if( sc.hasNextDouble() ) y = sc.nextDouble();
      else throw new RuntimeException("Denominator must be a number.");

      if( y==0 ) throw new RuntimeException("Denominator must be non-zero.");

      q = x/y;
      System.out.println("The quotient is: " + q);
   }
}
```

The method `hasNextDouble()`, belonging to the `Scanner` class, returns the `boolean` value `true` if the next token in the standard input stream can be interpreted as a `double`, and returns `false` otherwise. When `hasNextDouble()` returns true, we can call `nextDouble()` to extract that value and assign it to a `double` variable. This is done on lines 9 and 11 above. In the case that `hasNextDouble()` returns false, we throw a `RuntimeException` with descriptive error message, then quit the program. This is done on lines 10 and 12. Line 13 tests whether or not the denominator is zero, and if it is, we throw another `RuntimeException`. Several sample runs of `div.java` follow.

```
% java div
Enter the numerator and denominator: 125 321
The quotient is: 0.3894080996884735

% java div
Enter the numerator and denominator: 125xxx 321
Exception in thread "main" java.lang.RuntimeException: Numerator must be a number.
        at div.main(div.java:10)

% java div
Enter the numerator and denominator: 125 32gg1
Exception in thread "main" java.lang.RuntimeException: Denominator must be a number.
        at div.main(div.java:12)

% java div
Enter the numerator and denominator: 125 0
Exception in thread "main" java.lang.RuntimeException: Denominator must be non-zero.
        at div.main(div.java:13)
```

Notice that java prints the error message, points to the particular line in the source code that generated the Exception, then quits the program. Also study the program `Area2.java` on webpage to see another example of the proper use of `RuntimeException`.

The modified program `lawn2.java` will check each of its five input values to make sure they can be interpreted as doubles, before they are read from standard input. All five values will also be checked to make sure they are positive. In addition, the lot and house dimensions will be checked to make sure that a house of the given size will fit on the given lot. It is to be assumed here that the rectangular sides of the house are parallel to the sides of the lot. In all cases, if any of these checks fails, your program will throw a `RuntimeException` with appropriate error message, formatted as in the following examples.

```
% java lawn2
Enter the length and width of the lot, in feet: 150xyz 135
Exception in thread "main" java.lang.RuntimeException: Length and width of lot must be numbers.
        at lawn2.main(lawn2.java:14)

% java lawn2
Enter the length and width of the lot, in feet: 150 135
Enter the length and width of the house, in feet: 75 -60
Exception in thread "main" java.lang.RuntimeException: Length and width of house must be positive.
        at lawn2.main(lawn2.java:27)

% java lawn2
Enter the length and width of the lot, in feet: 30 50
Enter the length and width of the house, in feet: 35 40
Exception in thread "main" java.lang.RuntimeException: House does not fit on the given lot.
        at lawn2.main(lawn2.java:31)

% java lawn2
Enter the length and width of the lot, in feet: 150 135
Enter the length and width of the house, in feet: 75 60
The lawn area is 15750.0 square feet.
Enter the mowing rate, in square feet per second: 2.5fps
Exception in thread "main" java.lang.RuntimeException: Mowing rate must be a number.
        at lawn2.main(lawn2.java:40)
```

Notice that it is not enough to check that the area of the house is less than the area of the lot, since as the above example illustrates, this condition can be satisfied, while the house still does not fit on the lot. Let *a*, and *b* are the dimensions of the house, with *a* being the smaller side, (so $a \le b$), and let *c*, and *d* be the dimensions of the lot (where $c \le d$). It is then necessary and sufficient for the house to fit on the lot that both $a \le c$ and $b \le d$. A moments thought will show that there are least 12 distinct logical pathways the program can take to an Exception. Test your program in each of these situations. As usual, start early and ask questions if anything is unclear. Submit your `lawn2.java` program to the assignment name `pa2`.