CSci 4651 Fall 2003
Problem Set 7: Exceptions.
Due Friday, November 14th

**Problem 1.** For this problem you may use a (very simple) implementation of a linked list available on epoxy at ~elenam/pub/4651/LinkedList.java or copy-paste the code from the link on the assignments page. For simplicity the instance variables of a node (`item` and `next`) are made `public`.

Write a Java function `addLists` that takes two lists of integers and returns another list of integers whose $i$-th element is the sum of the $i$-th elements of the two given lists, provided the lists are of the same length.

Define an exception class `DifferentListLengthException` which extends the class Exception (not RuntimeException). If the list are not of the same length, the function `addLists` should throw this exception. The message for the exception should indicate which of the lists (the first one or the second one) is shorter. The exception class must have an integer variable `length` that contains the length of the shorter list.

Write `main` function to test the `addLists` function, include the code for catching the exception in `main`. Print out the message and the value of `length`. `main` should include several tests.

Does the function `addLists` return a list if the exception gets thrown? Check this in your program, include the testing code in your submission and explain the results.

Submit all the java files you have written for this problem.

**Problem 2.** Suppose you want to make sure that the `LinkedList` class (from Problem 1) doesn't cause the computer to crash when lists are used incorrectly. You would like to add exceptions to handle the following error conditions:

1. `EmptyListCarException` – an attempt to take a car of an empty list,
2. `EmptyListCdrException` – an attempt to take a cdr of an empty list,
3. `ListOverflowException` – an attempt to exceed the maximum number of elements in a list. This condition requires that you define a variable in the class `LinkedList` that indicates the maximum number of elements in a list. For the testing purposes you may set the variable to be some small number.

Consider the following 4 possibilities for defining the 3 kinds of exceptions:

1. Make each of the exceptions a direct extension of the class `Exception`.
2. Make each of the exceptions a direct extension of the class `RuntimeException`.
3. Make all 3 exceptions to be extensions of a class `LinkedListException`. Make `LinkedListException` to be an extension of `Exception`.

4. Make all 3 exceptions to be extensions of a class `LinkedListException`. Make `LinkedListException` to be an extension of `RuntimeException`.

**Question 1.** Compare these solutions based on ease of use and ease of understanding the code (for programmers who are using the LinkedList class). Which solution(s) would you recommend and why?

**Question 2.** Implement the exceptions using option 4 above. Include test examples for each exception in your submission.

**Problem 3.** Exercise 8.2 p. 229.

**Problem 4.** Write an implementation of linked lists in ML which will raise an exception `Head` at an attempt to return the head of an empty list and an exception `Tail` at an attempt to return the tail of an empty list (as the previous problem points out, the predefined list implementation raises exception Empty in both cases, so you can't use the predefined lists). Use the following datatype for your solution:

```
datatype 'a LinkedList = NIL | LIST of 'a * 'a list;
```

To get you started, below is the function `cons` operating on the type LinkedList. Note that the type of cons is `'a * 'a LinkedList -> 'a LinkedList`.

```
fun cons (x, NIL) = LIST (x, nil) | cons (x, LIST(y,z)) = LIST (x, y::z);
```

In this implementation `cons (8, cons (7, NIL))` will create a list of two elements: 8, 7.

Define the two exceptions `Head` and `Tail` and write the code for functions `head` and `tail` (or `car` and `cdr` if you prefer the scheme names) that raise the appropriate exceptions.

**Important:** the function `tail` must return an element of type `LinkedList`. It needs 3 cases (to see what the third case is, consider taking the tail of a 1-element list).

Test your functions, submit the test code, explain the results of the tests.