Program 3: Dynamic Programming

This program hopes to expose you to dynamic programming, a technique that facilitates a completely generalized and optimal version of the “make change” algorithm. Of note: the “programming” in “dynamic programming” does not refer to the code that you write, but to the optimal solution (i.e., the “plan” or “program”) that is sought by the problem.

Program to Write

Write a class called change.GoMakeChange that solves, in a general, optimal manner (thus giving “Go” a double meaning), the problem of making change for a given currency amount using a given set of coin denominations. The program’s output consists of the optimal way for making that amount using the given denominations. When no such way exists, the program prints a message to that effect.

Invoking change.GoMakeChange looks like this:

```
java change.GoMakeChange denominations amount
```

denominations is a comma-separated list of positive integers, while amount is the non-negative amount of change to be made. Arguments that do not conform to these constraints must be rejected with an appropriate error message, including but not limited to:

- Missing arguments
- Excess arguments
- Non-numeric arguments
- Any denomination \( \leq 0 \)
- Duplicate denominations
- Amount < 0
- Non-integral denominations or amount

The output of change.GoMakeChange may be either:

- “To make amount cents with denominations, use: \( n_1 d_1 \)-cent coins, \( n_2 d_2 \)-cent coins, \( n_3 d_3 \)-cent coins, …, and \( n_k d_k \)-cent coins.”
- “Sorry, but it is impossible to make amount cents with denominations.”

…where amount, denominations, and \( d_1, …, d_k \) are based on the user’s arguments, and \( n_1, …, n_k \) are the answers computed by the program.

Design Notes

The util.IntTuple class, which represents an ordered list of integers of some fixed cardinality, is crucial to this program. A JavaDoc description of this class can be found on the course Web site:

```
http://myweb.lmu.edu/dondi/spring2008/cmsi186/program3-api
```

You must complete this class — as well as unit tests, invokable from this class’s main() method — before writing a single line of change.GoMakeChange.

Note that util.IntTuple must be a general-purpose integer tuple class — it should not “know” that it is used to make generalized, optimal change.

Gotchas

- The denominations do not have to be sorted (and your code should not have to sort them, either).
- A one-cent denomination is not required; thus, the “no answer” case is certainly possible.
- There may be more than one optimal solution (i.e., a tie); in this case, the program may display any optimal solution.

Examples

- `java change.GoMakeChange 2` — error message (missing arguments)
- `java change.GoMakeChange 9,10 4` — error message (amount < 0)
- `java change.GoMakeChange 0,5,9 32` — error message (denomination \( \leq 0 \))
- `java change.GoMakeChange 10,1,10,14 28` — error message (duplicate denominations)
- `java change.GoMakeChange 2,16,8 5` — “Sorry, but it is impossible to make 5 cents with 2,16,8.”
- `java change.GoMakeChange 4,1,9 12` — “To make 12 cents with 4,1,9, use: 3 4-cent coins, 0 1-cent coins, and 0 9-cent coins.”