Indiana University
CSCI-C212: Introduction to Software Systems (F25)
Joshua Crotts

Problem Set 7
October 15, 2025

Generics and Streams

Important Dates:

- Assigned: October 15, 2025
- Deadline: October 22, 2025 at 11:59 PM EST

Objectives:

- Students learn how generics allow them to write methods that work over any type.
- Students learn to use the Stream API and its methods to solve a problem.

What To Do:

For each of the following problems, create a class named `ProblemX`, where `X` is the problem number. E.g., the class for problem 1 should be `Problem1.java`. Write (JUnit) tests for each method that you design in corresponding test files named `ProblemXTest`, where `X` is the problem number. Additionally, write Javadoc comments explaining the purpose of the method, its parameters, and return value. **Do not round your solutions!**

What You Cannot Use:

You cannot use any content beyond Chapter 3. Anything in or before Chapter 3 is fair game for this problem set, **but** you must respect the rules enforced by the problem. Moreover, anything that trivializes the solution is disallowed. Please contact a staff member if you are unsure about something you should or should not use. Any use of anything in the above-listed forbidden categories will result in a **zero** (0) on the problem set.

Problem 1:

Design the `<T> List<T> take(List<T> l, int n)` method that returns a list with the first n elements of the given list l. If $|l| \leq n$, return all elements of the list, but don't return the list itself. (That is, don't just return a reference to l.)

Problem 2:

Design the `<T> boolean isPalinList(List<T> ls)` method that, when given a `List<T>`, returns whether the list is "palindromic." A list is palindromic if, when its elements are reversed, it is the same. Note the similarity to checking if a string is a palindrome.

Problem 3:

Design the `<T> List<List<T>> cartesianProduct(List<List<T>> ls)` method that, when given a list of lists, returns the cartesian product of each list. That is, if ls contains lists $l_1, l_2, ..., l_n$, you should construct and return $l_1 \times l_2 \times \cdots \times l_n$. The cartesian product of *two* lists $X \times Y$ is $(x_1, y_1), (x_1, y_2), ..., (x_1, y_n), ..., (x_n, y_n)$, but generalizing this to n lists is more difficult.

One way to solve this problem is to design a method that computes the cartesian product of only two lists. Then, write a method that computes the cartesian product of a list of lists plus a second list of values. This then generalizes to any number of lists via a case analysis of having 0, 1, 2, or greater than 2 lists to compute the cartesian product of.

Another way is to use a recursive approach:

(i) If the list is empty, return the empty list.

(ii) If the list contains only one list inside, return a list where the elements are singletons. For example, `[[1, 2]]` returns `[[1], [2]]`.

(iii) Otherwise, split the list into two parts: call the first element *xs* and the rest *yss*. Instantiate a "result list" R. Recursively call the method on *ys*. Then, for every element $x \in xs$, for every element $ys \in yss$, create a new list l. Add to it x and all of the elements from *ys*. Then, add l to R.

Problem 4:

Design the `<T> List<T> interleave(List<T> l1, List<T> l2, int m, int n)` method that, when given two lists l_1, l_2 and two integers m, n, returns a new list with the first m elements of l_1, then the first n elements of l_2, then the next m elements of l_1, and so forth. If there are less than m elements left to take from l_1 or there are less than n elements left to take from l_2, take the rest of the lists. You may assume that $m, n \geq 1$. We provide an example below.

```
interleave(List.of("a", "b", "c", "d", "e", "f", "g", "h", "i", "j"),
           List.of("10", "20", "30", "40", "50"),
           3,
           2)
   => List.of("a", "b", "c", "10", "20", "d", "e", "f", "30", "40",
              "g", "h", "i", "50", "j")
```

Problem 5:

Design the `boolean containsHigh(List<List<Integer>> lop)` method that, when given a list of two-element arrays representing x, y coordinate pairs, returns whether or not any of the y-coordinates are greater than 450. **You must use only the Stream API.**

Problem 6:

Design the `List<Integer> sqAddFiveOmit(List<Integer> lon)` that receives a list of numbers, returns a list of those numbers squared and adds five to the result, omitting any of the resulting numbers that end in 5 or 6. **You must use only the Stream API.**

Problem 7:

Design the `List<String> removeLonger(List<String> los, int n)` method that receives a list of strings, and removes all strings that contain more characters than a given integer n. Return this result as a list. **You must use only the Stream API.**

Problem 8:

Design the `int filterSumChars(String s)` method that, when given a string *s*, removes all non-alphanumeric characters, converts all letters to uppercase, and computes the sum of the ASCII values of the letters. Digits should also be added, but use the digit itself and not its ASCII value. **You must use only the Stream API.**

Problem 9:

Design the `int productOdds(int n)` method that returns the sum of all of the odd integers from 1 to n, inclusive. You should use an `IntStream` to generate the appropriate range. **You must use only the Stream API.**

Problem 10:

Design the `int keyLength(Map<String, Set<String>> M, int n)` that, when given a map M of strings to sets of strings and a length n, returns the total length of the keys that map to a set of size at most n. **You must use only the Stream API.**