Infinite List

Problem Description

An infinite list `InfiniteList` is a generic list that can store elements of type `T` in order where duplicates are allowed. Unlike the previous lab, intermediate operations of `InfiniteList` should be lazily evaluated.

The Task

You are to design your own `InfiniteList` **interface** with the following requirements below. As `InfiniteList` is similar to Java's `Stream` in Java, and so, you are **not allowed** to import packages from `java.util.stream`

- Create the `InfiniteList` interface and the `InfiniteListImpl` implementation
- Define a `get` method for each operation

This task is divided into several levels. Read through all the levels to see how the different levels are related. **You need to complete all levels**.

Just remember to:

- archive and save a copy of all source files into the appropriate level directory (see specific level for usage details).

Level 1

Start the stream pipeline via the following data sources:

- `InfiniteList.generate(Supplier<T> supplier)`
- `InfiniteList.iterate(T seed, Function<T,T> next)`

Implement a `T get()` method for each data source such that exactly one element is retrieved each time the method is called.

You will also need to define the abstract class `InfinitListImpl` which you can leave empty now. This method will be populated with intermediate and terminal operations in the later levels.

Test the methods by writing a suitable test class, or using jshell. You may even write a jshell script.

```
jshell> /open InfiniteListImpl.java

jshell> /open InfiniteList.java
jshell> InfiniteList<String> ifl = InfiniteList.generate(() -> "A")
ifl ==> InfiniteList$1@59fa1d9b

jshell> IntStream.range(1, 5).forEach(x -> System.out.println(ifl.get()))
A
A
A
A

jshell> InfiniteList<String> ifl = InfiniteList.iterate("A", x -> "A" + x)
ifl ==> InfiniteList$2@146ba0ac

jshell> ifl.get()
$.. ==> "A"

jshell> ifl.get()
$.. ==> "AA"

jshell> ifl.get()
$.. ==> "AAA"

jshell> InfiniteList<String> ifl = InfiniteList.iterate("A", x -> "A" + x).generate(() -> "A")
|  Error:
|  illegal static interface method call
|    the receiver expression should be replaced with the type qualifier
|    'InfiniteList<java.lang.String>'
|  InfiniteList<String> ifl = InfiniteList.iterate("A", x -> "A" + x).generate(() -> "A");
|                             ^---------------------------------------------------------^
```

Click here to submit to CodeCrunch.

Make a copy of your Java programs to the level directory by typing the Unix commands

```
$ jar cvf infinite1.jar *.java
$ mkdir infinite1
$ cp *.java infinite1
$ cp infinite1.jar infinite1
```

Verify your jar archive using

```
$ jar tf ~/infinite1/infinite1.jar
```

Level 2

Implement the following **intermediate** operations following the corresponding specifications of Java's `Stream` API:

- `InfiniteList<R> map(Function<T, R> mapper)`
- `InfiniteList<T> limit(int maxSize)`
- `InfiniteList<T> filter(Predicate<T> predicate)`
- `InfiniteList<T> takeWhile(Predicate<T> predicate)`

As some of the methods could possibly produce no elements, you will need to redefine `get` to have a return type of `Optional<T>`.

Test the methods by writing a suitable test class, or using jshell. You may even write a jshell script.

```
jshell> /open InfiniteListImpl.java

jshell> /open InfiniteList.java

jshell> InfiniteList<String> ifl = InfiniteList.generate(() -> "A").map(x -> x + 1)
ifl ==> InfiniteListImpl$2@28d25987

jshell> IntStream.range(1, 5).forEach(x -> System.out.println(ifl.get()))
Optional[A1]
Optional[A1]
Optional[A1]
Optional[A1]

jshell> InfiniteList<Integer> ifl = InfiniteList.iterate(1, x -> x + 1).filter(x -> x % 2 == 0)
ifl ==> InfiniteListImpl$3@7cd62f43

jshell> IntStream.range(1, 5).forEach(x -> System.out.println(ifl.get()))
Optional[2]
Optional[4]
Optional[6]
Optional[8]

jshell> InfiniteList<Integer> ifl = InfiniteList.iterate(1, x -> x + 1).limit(2)
ifl ==> InfiniteListImpl$1@39c0f4a

jshell> IntStream.range(1, 5).forEach(x -> System.out.println(ifl.get()))
Optional[1]
Optional[2]
Optional.empty
Optional.empty

jshell> InfiniteList<Integer> ifl = InfiniteList.iterate(1, x -> x + 1).limit(2).filter(x -> x % 2 == 0)
ifl ==> InfiniteListImpl$3@53b32d7

jshell> IntStream.range(1, 5).forEach(x -> System.out.println(ifl.get()))
Optional[2]
Optional.empty
Optional.empty
Optional.empty

jshell> InfiniteList<Integer> ifl = InfiniteList.iterate(1, x -> x + 1).takeWhile(x -> x < 3)
ifl ==> InfiniteListImpl$4@3abbfa04

jshell> IntStream.range(1, 5).forEach(x -> System.out.println(ifl.get()))
Optional[1]
Optional[2]
Optional.empty
Optional.empty
```

Make a copy of your Java programs to the level directory by typing the Unix commands

```
$ jar cvf infinite2.jar *.java
$ mkdir infinite2
$ cp *.java infinite2
$ cp infinite2.jar infinite2
```

Verify your jar archive using

```
$ jar tf ~/infinite2/infinite2.jar
```

Level 3

Now implement the following **terminal** operations by following the corresponding specifications of Java's `Stream` API:

- `long count()`
- `void forEach(Consumer<T> action)`
- `Optional<T> reduce(BiFunction<T,T,T> accumulator)`
- `T reduce(T identity, BiFunction<T,T,T> accumulator)`
- `Object[] toArray()`

You will also need to ensure that the `get` method can no longer be called from a client class.

Test the methods by writing a suitable test class, or using jshell. You may even write a jshell script.

```
jshell> /open InfiniteListImpl.java

jshell> /open InfiniteList.java

jshell> InfiniteList.iterate(1, x -> x + 1).filter(x -> x % 2 == 1).limit(10).count()
$.. ==> 10
```

```
jshell> InfiniteList.iterate(1, x -> x + 1).limit(10).filter(x -> x % 2 == 1).count()
$.. ==> 5

jshell> InfiniteList.iterate(1, x -> x + 1).limit(5).forEach(System.out::println)
1
2
3
4
5

jshell> InfiniteList.iterate(1, x -> x + 1).limit(5).reduce(0, (x, y) -> x + y)
$.. ==> 15

jshell> InfiniteList.iterate(1, x -> x + 1).limit(0).reduce(0, (x, y) -> x + y)
$.. ==> 0

jshell> InfiniteList.iterate(1, x -> x + 1).limit(5).reduce((x, y) -> x + y)
$.. ==> Optional[15]

jshell> InfiniteList.iterate(1, x -> x + 1).limit(0).reduce((x, y) -> x + y)
$.. ==> Optional.empty

jshell> InfiniteList.iterate(1, x -> x + 1).map(x -> x * 2).limit(10).toArray()
$20 ==> Object[10] { 2, 4, 6, 8, 10, 12, 14, 16, 18, 20 }

jshell> InfiniteList.generate(() -> 1).get()
|  Error:
|  cannot find symbol
|    symbol:   method get()
|  InfiniteList.generate(() -> 1).get()
|  ^--------------------------------^

jshell> InfiniteList.generate(() -> 1).map(x -> x * 2).get()
|  Error:
|  cannot find symbol
|    symbol:   method get()
|  InfiniteList.generate(() -> 1).map(x -> x * 2).get()
|  ^------------------------------------------------^
```

Make a copy of your Java programs to the level directory by typing the Unix commands

```
$ jar cvf infinite3.jar *.java
$ mkdir infinite3
$ cp *.java infinite3
$ cp infinite3.jar infinite3
```

Verify your jar archive using

```
$ jar tf ~/infinite3/infinite3.jar
```

Level 4

Finally, create the package `cs2030.mystream` for the `InfiniteList` interface and its implementation class.

Define a client class `Main` that imports `cs2030.mystream` to test your implementation and compile your program using

```
$ javac -d . *.java
```

Make a copy of your Java programs to the level directory by typing the Unix commands

```
$ jar cvf infinite4.jar *.java
$ mkdir infinite4
$ cp *.java infinite4
$ cp infinite4.jar infinite4
```

Verify your jar archive using

```
$ jar tf ~/infinite4/infinite4.jar
```