Functional Programming in Java

The goal of this recitation is to familiarize you with functional programming within the context of an object-oriented programming language, specifically focusing on the features provided by Java 8. In this recitation, you will use the `Stream` interface to complete a series of short exercises.

Java Shorthand

When a class implements an interface with a single method (such as the `Runnable` and `ActionListener` interfaces), it's often the case that defining the interface method is the only thing the class does—no other methods or fields are needed. Such a class is equivalent to a function, since the only defined behavior for the class is the implemented method. Java allows classes which implement a single method to be replaced by a lambda expression. For example: the code

```
button.addActionListener(new ActionListener() {
    public void actionPerformed(ActionEvent e) {
        System.out.println("Action!");
    }
});
```

sets the listener of the button by creating an anonymous subclass of `ActionListener`. In Java 8, the code may be re-written as:

```
button.addActionListener(event -> System.out.println("Action!"));
```

Streams

In Java, each Collection can yield an associated `Stream` - a sequence of elements which support aggregate parallel or sequential operations. Operations on a `Stream<E>` include:

- `Stream<F> map(Function<E, F> f)` - applies a function `f` : `E` → `F` (`f` takes objects of type `E` and returns an object of `F`) to each element of a stream and uses each element's resulting value in the returned stream.
- `Stream<E> filter(Predicate<E> f)` - returns a stream containing only the elements where `f` : `E` → `bool` evaluates to true.

- `E reduce(E identity, BinaryOperator<E> f)` - repeatedly combines adjacent elements of the stream with an associative binary operator `f` until the stream has been reduced to a single value. The identity must satisfy `f(identity, x) = x` for any value of `x`.

More examples of operations can be found in the official documentation here.

Exercise: A malfunctioning inventory

The inventory manager is buggy and broken. Your mission, if you choose to accept it, is to fix the inventory manager with the powers of functional programming!

In `edu.cmu.cs.cs214.rec14.exercises.inventory.InventoryExercise` complete the following:

1. Get a list of all product names.
2. Get a list of all *distinct* product names.
3. Calculate the total price.
4. Calculate the total price of all products with the category "Laptop"
5. Calculate the total price of all products in each category.

Exercise: Stock Oracle

In `edu.cmu.cs.cs214.rec14.exercises.stocks.StockExercise` complete the following:

1. For each `Stock`, output minimum and maximum price over its entire history.
2. Find the maximum gain for a single stock.

Exercise: One Way Ticket

In `edu.cmu.cs.cs214.rec14.exercises.flights.FlightExercise` complete the following:

1. Given an initial city and an unordered list of `Flights`, identify a circuit that starts and ends in the initial city, using each flight once (but not necessarily in the order given in the input list). Your function should express this circuit by returning the list of cities in the order they are visited. You may assume that each city is only visited once, and the number of cities visited is equal to the number of flights. The initial city should be at the beginning of the output list, but not at the end.
2. Given a list of flights, return all cities which have incoming flights.