COP2805C Module 5 Practice Exercise

Message queues are a fundamental programming construct which solve fundamental issues with distributed applications. You can read about the basics of messaging on the O'Reilly system at this link:

https://learning.oreilly.com/library/view/enterprise-integration-patterns/0321200683/pref06.htm l

In this practice exercise we will create and implement a generic interface to dispatch birthday cards. We will also use the Queue interface from Java Collections to act as a primitive message queue.

```
Modify the BirthdayGreeter interface by changing the abstract method
parameters to the following (we are now using an actual BirthdayCard class);
```

```
public interface BirthdayGreeter {
    // build a birthday card for a with a greeting
    public BirthdayCard buildCard(User u, String msg);
    // send a birthday card
    public void sendCard(BirthdayCard bc);
}
```

Since our application is growing in size, I have started splitting some of the classes out into separate source code files. For instance, the User and BirthdayCard classes are separate files in the birthday package (User should probably be moved into its own package, I may do that later).

Create the Dispatcher interface. This is a generic interface that dispatches things that may not be birthday cards, so we create a separate package for it and use a generic parameter.

```
package edu.fscj.cop2805c.dispatch;

public interface Dispatcher<T> {
    public void dispatch(T t);
}
```

The HappyBirthdayApp class now also implements a dispatcher for birthday cards; the BirthdayCard class replaces the generic parameter. We are using the Queue interface from Java Collections to dispatch the cards. Notice that we have to use a concrete class to instantiate the Queue. Here I have chosen a linked list, a general purpose list structure which is optimized for insertion/deletion at both ends of the queue. There are other classes we could use for this depending on the desired behavior; a PriorityQueue, for instance, would remove elements from the queue based on a specified priority instead of first-in-first-out (FIFO) which is the default behavior.

```
public class HappyBirthdayApp implements BirthdayGreeter,
Dispatcher<BirthdayCard>  {
    //private User user;
    private ArrayList<User> birthdays = new ArrayList<>();

    // Use a Queue<LinkedList> to act as message queue for the
dispatcher
    private Queue<BirthdayCard> queue = new
              LinkedList<BirthdayCard>();
```

This implementation acts as a message queue to store the cards. When we dispatch a card, we will just add it to the queue for now and let them accumulate, we will figure out how the queue is accessed in order to send them out over the network later. I am printing the queue length as debugging output to monitor the queue size, this would be removed in a production version of the code.

```java
// implement a dispatcher for a BirthdayCard
class BirthdayCardDispatcher implements Dispatcher<BirthdayCard> {
    private Queue<BirthdayCard> queue;

    public BirthdayCardDispatcher() {
        queue = new LinkedList<BirthdayCard>();
    }

    public void dispatch(BirthdayCard bc) {
        this.queue.add(bc);
        System.out.println("current queue length is " + this.queue.size());
        System.out.println(bc);
        System.out.println("sending a birthday card to " +
                   bc.getUser().getEmail() + "\n");
    }
}
```

The sendCard method now simply dispatches the card. There are two ways we can do this, one is by calling the dispatch method:

```java
public void sendCard(BirthdayCard bc) {
    // dispatch the card
    dispatch(bc);
}
```

An alternative is to create a lambda expression. The logic for this implementation is a bit lengthy, we would normally use a method for this many statements (if we were chaining method calls it would be pretty ugly) but for demonstration purposes here is the lambda:

```java
public void sendCard(BirthdayCard bc) {
    // dispatch the card
    // dispatch(bc);
    // show an alternative dispatch using a lambda
    Dispatcher<BirthdayCard> d = (c)-> {
        this.queue.add(c);
        System.out.println("current queue length is " + this.queue.size());
        System.out.println(c);
        System.out.println("sending a birthday card to " +
                c.getUser().getEmail() + "\n");
    };
    d.dispatch(bc);
}
```