-------- Answer ------------------------

The test case:

```
Date someDate = new Date(12, 16, 2015)
LL<Post> snaps = getSnapshot("kathi", someDate)

t.checkExpect(snaps.size() <= 10 &&
              DatesWithinDays(snaps, somedate, 3) &&
              uniquePosters(snaps).size() >= 5, true)
```

The methods:

```
// are all dates within numDays of the given date
boolean DatesWithinDays(LL<Post> posts, Date someDate, int numDays)
  -- goes into the SocialNetwork or Examples class

// list of unique members who have a post in the list, no duplicates
LL<Member> uniquePosters(LL<Post>)
  -- goes into the SocialNetwork or Examples class
```

------- Grading Comments -----------------

* The general strategy here is to have a method or a built-in operator that can check each of the criteria for your expected answer. Generate the answer, then check each of the criteria within the checkExpect.

* It is critical that someDate (in the test case part) appears in both the call used to build the snaps list and in the DatesWithinDays call used to check the list of snaps. This shows that you understand that the data that is used to generate the candidate answer is often needed to check that the answer is valid.

* The exact constructor for Date was not important. As long as you did something that showed you had a common Date object in both the call and the check, we wouldn't have cared about the constructor part.

* You could have broken this down differently, such as putting a method to compare dates in the Post class. Grading on this question did not care about encapsulation. It did care that you put methods in classes that had access to the other methods and data that you cared about.