Assignment 1 (solution)

Exercise 1 (Requirements Elicitation)

There is no authoritative solution to this exercise since it depends on the discussion in the exercise session. The following should be mainly seen as hints:

- Actors:
  - Customer
  - Flower Shop Manager
  - Messenger
- Some open issues:
  - How does the messenger communicate with the web browser?
  - How are undelivered flower orders handled?
  - Can he use the system to "push" flowers that will expire soon?
  - Who is going to host the system?
- Scenarios:
  - Scenario 1 (normal)
    1. Jill wishes to purchase some flowers.
    2. She logs into the internet browser with her user name and password.
    3. She selects the flowers of her choosing and presses check-out.
    4. For the address, she selects her home address.
    5. Jill pays with her credit card.
    6. The system offers her a receipt for the delivery and her credit card is charged.
  - Scenario 2 (exceptional)
    1. Bob wants to become a frequent customer for the web shop.
    2. He enters the URL of the shop and selects new customer.
    3. Bob gives his preferred username.
    4. The system finds that there is already a username existing, and notifies Bob that he cannot have this username.
  - Scenario 3 (unspecified)
    - Sarah has already checked out and printed her receipt.

    * She realizes that the address she selected was incorrect.
    * She immediately logs back in and selects her last order.
    * The system tells her that it hasn't been prepared yet and that she is able to modify the order.
    * She changes the address to the correct one and prints out the new receipt.

- Non-functional requirements:
  - Client should be able to use standard web browsers.
  - Respond time of the system should be within 3 seconds.
  - The system should support at least 400 clients.
  - The system should use the existing point of sales system.

Exercise 2 (Design and Documentation)

One could cache the shortest path by recomputing it lazily or eagerly (question 4):

- Lazily:

```
class Graph
{
  List<Edge> edges;
  Node source;
  Node destination;

  private List<Node> sp;

  void addEdge(Edge e)
  {
    edges.Add(e);
    sp = null;
  }

  List<Node> shortestPath()
  {
    if (sp != null)
    {
      return sp;
    }
    else
    {
      ...
    }
  }
}
```

  - This should not influence the client-visible documentation.

- One could add the following documentation:
  1. postcondition for `addEdge`: `sp == null`
  2. postcondition for `shortestPath`: `old(sp == null) || result == sp`

- Eagerly:

```
class Graph
{
  List<Edge> edges;
  Node source;
  Node destination;

  private List<Node> sp;

  void addEdge(Edge e)
  {
    edges.Add(e);
    sp = computeShortestPath();
  }

  private computeShortestPath()
  {
    ...
  }

  List<Node> shortestPath()
  {
    return sp;
  }
}
```

  - This should not influence the client-visible documentation.
  - One could add the following documentation:
    1. class invariant for class `Graph`: `sp != null`
    2. postcondition for `shortestPath`: `result == sp`

Exercise 3 (Design)

1. This is one possible scenario:
   (a) A new list 'a' is created.
   (b) This list is cloned to create a list 'b'.
   (c) List 'b' is modified by calling method `set`.
   (d) List 'a' is modified by calling method `set` and a new `ListRep` object is created even though the list is technically not shared anymore.
2. You could use actual reference counting instead of using the boolean field `shared`.

3. Yes, for instance:

   (a) A new list 'a' is created.

   (b) This list is cloned to create a list 'b'.

   (c) List 'b' is not used anymore and is removed from the heap by the garbage collector.

   (d) List 'a' is modified by calling method `set` and a new `ListRep` object is created even though the list is technically not shared anymore.

4. You could implement a finalizer that decreases the `shared` counter before the object is eventually removed from the heap.