A Appendix: Queue Implementation

```
type 'a qnode = { v: 'a;
                  mutable next: 'a qnode option }

type 'a queue = { mutable head: 'a qnode option;
                  mutable tail: 'a qnode option }

(* INVARIANT:
 - q.head and q.tail are either both None, or
 - q.head and q.tail both point to Some nodes, and
   - q.tail is reachable by following 'next' pointers
     from q.head
   - q.tail's next pointer is None
*)

let create () : 'a queue =
  { head = None; tail = None }

(* Add an element to the tail of a queue *)
let enq (elt: 'a) (q: 'a queue) : unit =
  let newnode = { v = elt; next = None } in
  begin match q.tail with
  | None ->
     (* Note that the invariant tells us that q.head is also None *)
     q.head <- Some newnode;
     q.tail <- Some newnode
  | Some n ->
     n.next <- Some newnode;
     q.tail <- Some newnode
  end

(* Remove an element from the head of the queue *)
let deq (q: 'a queue) : 'a =
  begin match q.head with
  | None ->
     failwith "deq called on empty queue"
  | Some n ->
     q.head <- n.next;
     if n.next = None then q.tail <- None;
     n.v
  end
```

B Appendix: Java Interfaces and Classes

```
1  public interface I {
2      void method1();
3  }
```

```
1  public interface J {
2      void method2();
3  }
```

```
1  public class C implements I {
2
3      @Override
4      public void method1() {
5          System.out.println("Class C's method1!");
6      }
7
8  }
```

```
1  public class D extends C implements J {
2
3      @Override
4      public void method2() {
5          System.out.println("Class D's method2!");
6      }
7
8  }
```

```
1  public class E implements I, J {
2
3      @Override
4      public void method1() {
5          System.out.println("Class E's method1!");
6      }
7
8      @Override
9      public void method2() {
10         System.out.println("Class E's method2!");
11     }
12
13 }
```

```
1  public class F {
2
3      void invokeMethod(J obj) {
4          obj.method2();
5      }
6
7  }
```