2009 November 23

Example Test Questions for Eiffel and ADTs

1. Eiffel

1. Java distinguishes between the primitive types -- int, char, real, etc. -- and real objects. Eiffel does not make this distinction. Explain how Eiffel can treat the primitive types as first class objects.

2. Eiffel has four mechanisms for adaptation. Describe and give an example of each one.

3. Describe, in execution order, the steps Eiffel follows in creating an Object.

4. Explain what is an expanded type and why expanded types are needed.

5. In order to implement a vector of similar objects in Eiffel, one could create a generic class VECTOR[G]. In Java, one would have to use the class Vector (stores arbitrary objects). Explain the benefits of the former approach, and the dangers of the latter.

6. Consider the following two classes:

```
class CLIENT

feature
      s : SUPPLIER

      foo is
            do
                  …
            end

end
```

```
class SUPPLIER

feature

      bar: INTEGER

      -- Other features…

end
```

It is illegal in Eiffel to do the following in the body of feature `foo`: `s.bar := 0`
Explain what is the rationale behind this restriction.

2. Agents & Tuples

1. Consider a priority queue `PQ` as a sequence consisting of items, each called `item`, each containing the following fields.

   `< priority : INTEGER , time : INTEGER , data : ANY >`

   Larger integers indicate higher priority and later arrival time.

   Mathematically, the weakest class invariant that describes such a priority queue is the following.

   ∀ j, k : 1.. #PQ | j < k • item[j].priority > item[k].priority (# means length of).
   or (item[j].priority = item[k].priority
   and item[j].time < item[k].time)

   You are given the following agent.

```
valid_pair ( item_j : Q_ITEM[STRING]
           ; item_k : Q_ITEM[STRING]) : BOOLEAN is
  -- Returns true if and only if it is valid that item_j be closer to the front of the queue
```

```
-- than item_k.
do
  Result :=    (item_j.priority > item_k.priority)
            or (item_j.priority = item_k.priority
                 and
               item_j.time < item_k.time)
end
```

A Define and explain the signature, in Eiffel syntax, for the function `forall` that would enable a client to pass the agent `valid_pair` to verify the correctness of the preceding invariant.

B Give the calling sequence a client of the priority queue would use to invoke the function `forall`; assume it is a feature in the priority class queue that is exported to all.

C Now give the implementation body for the function forall that matches your signature.

2. For the following question you need to write assertions and then implements them using agents in Eiffel. You need to complete the code for *for_all*. Here is exactly what needs to be done:
   a) write, in English, what the sort feature need to ensure.
   b) Implement what contract, written in part a, using Eiffel agents.
   c) Complete the body of for_all.
   d) Implement agents that are needed for part b.

```
class
      SORTED_ARRAY
create
      make
feature

      a: ARRAY [INTEGER]

      make is
                  -- example only
            do
                  create  a.make (1, 6)
                  a := <<2, 4, 6, 6, 8, 5>>
            end

      sort  is
                  -- Sort array  'a' in non-decreasing order
            require
                  v /= void
            do
            deferred
            ensure
                  ???
            end

      for_all (low, up: INTEGER;
                test: FUNCTION [ANY, TUPLE [INTEGER], BOOLEAN])
              : BOOLEAN is
            -- Is "test(a @ i)" true for all i, low <= i <= up
            local
                  i: INTEGER
            do
                  from
                        ???
                  until
                        ???
                  loop
                        ???
                  end
```

```
                ???
            end

        -- Write the auxiliary routines needed to implement the
        -- contracts in the space below:
            ???

end  -- class SORTED_ARRAY
```

3. ADTs and Classes

1. Describe the relationship between an abstract data type and a class.

2. Describe the basic steps in getting a class from an abstract data type.

3. Consider a priority queue `PQ` as a sequence consisting of items, each called `item`, each containing the following fields.

   ```
   < priority : INTEGER , time : INTEGER , data : ANY >
   ```

   Larger integers indicate higher priority and later arrival time. Time increases indefinitely

   A. Mathematically, give the weakest class invariant that describes such a priority queue.

   B. Mathematically, give a class invariant that describes such a priority queue **and** also captures the notion of a priority queue as consisting of a sequence of sub-sequences of the same priority.

4. Describe the relationship between an abstract data type and a class.

5. Describe various ways that are used to find classes.

6. Describe various issues pertaining to designing classes.