2009 November 23

Design by Contract
Example Test Questions

1.

Explain when and why one can use Design by Contract.
Explain when and why one cannot use Design by Contract.

2.

A State and explain the correctness rule for a class with respect to its assertions for a creation routine.

B State and explain the formal correctness rule for a class with respect to its assertions for exported routines.

C State and explain the formal correctness rule for retry inducing rescue clauses (does a retry).

D State and explain the formal correctness rule for failure inducing rescue clauses (doesn't do a retry).

3.

Describe the benefits and obligations of the client and supplier when using design by contract.

4.

In the context of software environment define the term *exception.*
Does every exception lead to a routine failure? Justify your answer.
Discuss the two legitimate responses to an exception. Justify your answers.

5.

A What is defensive programming?

B Explain why defensive programming is a poor method.

C Defensive programming suggests the following programming style.

```
remove (object) is
require  size > 0
  if size < 1 then throw exception
  else ... rest of procedure ...
end remove
```

Using example pseudocode from the client side, show why defensive programming is futile, if the client is a good programmer.

6.

Given following two classes:

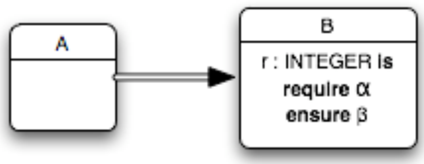
class B **feature** x : INTEGER; y : INTEGER do_work (a : INTEGER) **is** **require** alpha: a < 0 … **ensure** beta: x >= a + 20 **and** y < **old** y + x **end**	**class** C **feature** **inherit** B **redefine** do_work **end** do_work (a : INTEGER) **is** **require** gamma: a <= 0 … **ensure** delta: x >= 21 + a **and** y – **old** y - 20 < a **end**

end	**end**

Assume we have an Eiffel compiler that does not enforce the Assertion Redeclaration Rule. We therefore need to use our knowledge of mathematics and logic to verify the subcontracting by class C eliminates the possibility of cheating. State what needs to be proved and prove it.

7.

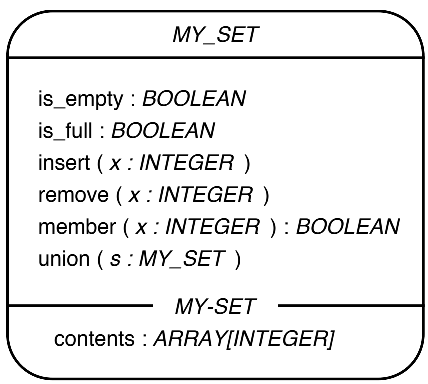
Let *A* and *B* be two classes related as shown by the following BON diagram.



In order to prevent cheating, the redeclaration of the assertions in an heir must follow the "Assertion Redeclaration Rule". Give a definition for this rule. Using references to feature *r* of class *B*, explain this rule.

8.

The following figure shows a simple class interface for a class MY_SET that holds only integers. The elements of the set are represented as an ARRAY of integers.



Give **require** and **ensure** clauses for each of the set features. Write your contracts as formally as possible. You can use any features of ARRAY that you like in your contracts; if you are unsure as to which features ARRAY possesses, clearly state your assumptions. You may use BON assertion language. Answers expressed in mathematics receive significantly higher grades than answers expressed in English.

9.

Suppose you are given the following classes.

```
class A
feature
  x : INTEGER  ;  y : INTEGER

  do_work ( a : INTEGER ) is
  require a > 0
```

```
class B
inherit A redefine do_work end
feature

  do_work ( a : INTEGER ) is
  require else a >= 0
```

```
  ensure x >= a + 10
         and y < old y + 5
  end
end
```

```
  ensure then x >= 14 + a - 3
              and  y - old y = 3
  end
end
```

Is the redefinition of feature `do_work` valid according to Eiffel's refinement rule? Carefully **prove** that your answer is correct.

10.

Suppose a class has an invariant that includes the clause (x >= 0 and y = 0). I want to inherit from this class and in doing so, add a new clause (x >= 0 implies y >= 0). Is this acceptable? Why or why not?

11.

State the three main criteria used to define what is meant by “reasonable preconditions” in design by contract.

12.

The exception handling principle deals with two ways of handling exceptions. Describe those two ways. Describe their pre and post conditions. Give an Eiffel template and show where the conditions are met.

13.

Answer the following questions with respect to the following program text..

A What would be the class invariant for class A? Will the class invariant be satisfied immediately after creating an instance of A – create a.make. Justify your answer completely and in detail.

B Assume that all assertions are changed to **true**, what would be the result of executing the program text in the ROOT_CLASS. Justify your answer completely and in detail.

C Is the redefinition of the feature print in class B correct. Justify your answer in detail by using the complete pre and post conditions to justify your conclusion.

```
class D  creation  make  feature
  x : INTEGER
  y : INTEGER

  make is do x := 2 ; y := 5 end

  printa( a : INTEGER) is
  require y > x
  do
  io.put_string("In D a is: ")
  io.put_integer(a)
  ensure y - x > 0
  end

  double : INTEGER is
  require y > 2
  do
  result := 2 * y + x
  ensure result > y + x
  end

  invariant x > 1
```

```
class C
inherit D
   rename printa as c_print
   redefine double end

creation make feature

  printa( a : INTEGER) is
  require true
  do
  io.put_string("In C a is: ")
  io.put_integer(a)
  ensure true
  end

  double : INTEGER is
  require else y > 0
  do
  result := x + y - x*y
  ensure then result > x + y
  end
```

```
class ROOT_CLASS
  creation make
  feature

  a : A  ;  b : B
  c : C  ;  d : D

make is do
  create  a.make
  create b.make
  create  c.make
  create d.make

  a.printa(a.double)
  d := a
  d.printa(a.double)

end
```

end	**invariant** y >= 5 **end**	
class B **inherit D** **redefine** printa, make **end** **creation** make **feature** z : INTEGER make **is** do precursor ; z := -4 **end** printa(a : INTEGER) **is** **require else** x > 0 **do** io.put_string("In B a is: ") io.put_integer(a) **ensure then true** **end** **invariant** z < x **or** z > y y – 5 > x **end**	**class** A **inherit** B **rename** printa **as** b_print **redefine** double, make **end** C **undefine make** **redefine** double **select** c_print **end** **creation** make **feature** make **is do** precursor double := z + x + y **end** double : INTEGER **invariant** ??? **end**	

14.

Describe when a class invariant must be true?

Describe when a class invariant may be false.

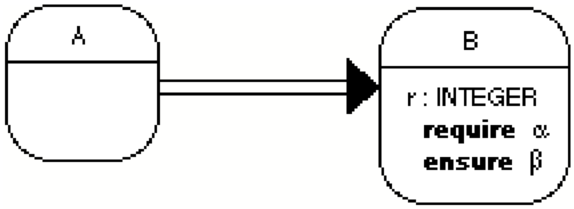
15.

Give **require** and **ensure** clauses for a class *COMPLEX*, representing complex numbers. Recall that a complex number is of the form $a + ib$, where a and b are real numbers, and $i^2 = -1$. Write your contracts as formally as possible. Answers expressed in mathematics receive significantly higher grades than answers expressed in English.

```
class COMPLEX
creation make
feature
  a : REAL  ;  b : REAL
  make(a : REAL; b : REAL)
  add(c : COMPLEX) -- add c to the object
  subtract(c : COMPLEX)  -- Subtract c from the object
  multiply(c : COMPLEX)  -- multiply the object by c
  length : REAL -- returns the length of the complex number
end -- COMPLEX
```

16.

Let *A* and *B* be two classes related as shown by the following BON diagram.



A Recall that through inheritance and dynamic binding there is a potential for cheating *A* with respect to its contract with *B*. Use references to feature *r* of class *B* to show exactly how inheritance and dynamic binding can be used to cheat *A*.

B In order to prevent cheating, the redeclaration of the assertions in an heir must follow the "Assertion Redeclaration Rule". Using references to feature *r* of class *B*, explain this rule.

C If feature *r* of class *B* is redefined into an attribute in a proper descendent of *B* discuss how α and β will be handled?

17.

Although both of the correctness formulae "{False} A {Q}" and "{P} A {True}" require minimum effort on the part of the supplier, there is a subtle distinction between them. Explain what it is.

18.

In the context of design by contract recall that, for a routine, it is desirable to have the weakest precondition and the strongest postcondition that make its task feasible. Explain why this is reasonable. Hint: think about starting a business and offering a set of services to clients.

19.

Explain why design by contract is suitable for software-to-software communication and not for software-to-human or software-to-outside-world communication.

20.

The routine *foo*, shown below, is a client of MATRIX and has been written according to contract by using the MATRIX feature *singular* which returns the solution, if the current matrix is singular.

```
a : MATRIX;  b, x : VECTOR
foo is do
        …       -- instructions to create  a, b, and x
        if not a.singular then
                x := a.solution(b)
        else
                io.put_string("solution not possible")
        end
end -- foo
```

Assuming that a pre-condition violation will occur if solution is called on a singular matrix, rewrite *foo* so that it does not check the pre-condition of solution but still behaves as though it had been programmed according to contract.

21.

Give **class invariant**, **require** and **ensure** clauses for a class *PRIORITY_QUEUE*. Do not forget to annotate your clauses with English statements.

Priority queues are ordered by priority in the sense that the item removed from the queue is the item with the highest priority (the larger the integer the higher the priority). Other than that, the standard queue discipline holds.

```
class QUEUE_ITEM[G]
creation make
```

```
feature
  priority : INTEGER  ;  time : INTEGER ; data : G
  make(thePriority : INTEGER; theTime : INTEGER; theData : G)
  theData : data -- returns the data
  theTime : INTEGER -- returns the time
  thePrioirity : INTEGER -- returns the priority
end -- QUEUE_ITEM

class PRIORITY_QUEUE[G]
creation make
feature { NONE }

  pq : LIST[QUEUE_ITEM]

  time : INTEGER -- Simulate with an increasing counter for each item
                 -- added to the queue.

feature

currentItem : QUEUE_ITEM  -- Last queue item to be enqueued or dequeued

make

  require ???

  ensure ???

enqueue(item : G ; priority : INTEGER)  -- Add to priority queue

  require ???

  ensure ???

dequeue  -- Remove highest priority item.

  require ???

  ensure ???

invariant ???

end -- PRIORITY_QUEUE
```

22.

You are given the following defined class ROOM with the only features you need for this problem.

```
class ROOM
feature {ANY}
  status : STATUS     -- One of reserved, unreserved, occupied, repair
  guest : GUEST       -- Only for HOTEL void unless occupied or reserved.
end
```

Write require, ensure and class invariant assertions for the following methods of a class HOTEL that represents rooms and guests at a hotel. The minimum size of a hotel is 100 rooms. Write your assertions in as formal a mathematical notation as possible. Your assertions do not have to be executable.

```
class HOTEL  creation make
feature {NONE}

avail_rooms   : LIST[ROOMS]     -- List of all the rooms available not under repair
repair_rooms  : LIST[ROOMS]     -- List of all the rooms under repair
capacity      : INTEGER         -- Number of rooms in the hotel

feature
make (size : INTEGER) is
 -- Build a new hotel with size rooms where all rooms are available.
```

```
    require  ???
    ensure  ???

  vacancy : BOOLEAN is
   -- Returns true if and only if there is an unreserved room.
    require ???
    ensure  ???

  unreserved_check_in (guest : GUEST) is
   -- The guest has not made a reservation.  Puts the guest into an unreserved room.
    require  ???
    ensure  ???
  remove_room_for_repair (room : ROOM) is
   -- Moves an unoccupied room from the available list to the repair list.
    require  ???
    ensure  ???

  invariant

end -- HOTEL
```

23.

The following Eiffel system compiles correctly (execution starts at `make` of `ROOT_CLASS`). However, when executed it creates a contract violation. Explain why that is, and describe how you would modify the two classes to fix the problem.

```
class ROOT_CLASS

create make

feature
  make is
  do
    create t
  end

  t: TEENAGER

end
```

```
class TEENAGER

feature
  age: INTEGER

invariant
  age > 12 and age < 20

end
```

24.

The class SET describes collection of objects where each element must be unique.

Here we have provided part of the code for this class. Your task is to complete the missing contracts for each routine. You DO NOT need to implement any of these routines.

```
indexing
      description: "Collection, where each element must be
unique."
deferred  class
      SET [G]
```

```
inherit
      COLLECTION [G]
            redefine
                  changeable_comparison_criterion
            end
feature  -- Measurement

      count: INTEGER is
                  -- Number of items
            deferred
            end

feature  -- Element change

      extend (v: G) is
                  -- Ensure that set includes `v'.
                  -- Was declared in SET as synonym of `put'.
            deferred
            ensure  then
                  in_set_already:
                  added_to_set:
            end

      put (v: G) is
                  -- Ensure that set includes `v'.
                  -- Was declared in SET as synonym of `extend'.
            deferred
            ensure  then
                  in_set_already:
                  added_to_set:
            end

feature  -- Removal

      prune (v: G) is
                  -- Remove `v' if present.
            deferred
            ensure  then
                  removed_count_change:
                  not_removed_no_count_change:
                  item_deleted:
            end

end  -- class SET
```

25.

You are given the following partially defined class TABLE with only the features you need for this problem.

```
class TABLE
feature {ANY}
  id : SET[LABEL]      -- Every table needs a unique identifier
  chairs : INTEGER     -- Number of people that can sit at the table
end
```

Write require, ensure and class invariant clauses for the following features in a RESTAURANT class. Combining two tables loses one seating position and has the labels from both tables. Splitting a table in two adds one additional seating position and labels are arbitrarily divided. Tables of size 1 are not permitted. Write your clauses in: (1) English, at least; and (2) in as formal a mathematical notation as possible. Your clauses do not have to be executable.

```
class RESTAURANT  creation make
```

```
feature
  place : LIST[TABLE]        -- Keep track of the tables
  init_chairs : INTEGER      -- Number of chairs in the initial restaurant
  capacity : INTEGER         -- Current number of people that can be seated at tables
  spare_chairs : INTEGER     -- Number of chairs not at tables
```

class invariants ???

```
make(initial_chairs : INTEGER) is
        -- Creates a restaurant that has an initial capacity for a minimum of 20 people.
        -- Initially all the chairs are at two seat tables.  Every table has one label in its id.
```

require ???
ensure ???

```
combine(table1 : TABLE; table2 : TABLE) is
        -- combines table1 and table2 into a single larger table.
```

require ???
ensure ???

```
split(big_table : TABLE; size1 : INTEGER; size2 : INTEGER) is
        -- split the big_table into two smaller tables with size1 chairs and size2 chairs.
```

require ???
ensure ???

You are given the following partially defined class BOOK with the only features you need for this question.

• Must use this information to distinguish cases

```
class BOOK
feature {ANY}
  state : STATUS            --  Books available for lending are either reserved,
                            -- or  unreserved. Books not available for lending
                            -- are  borrowed  or  under_repair
  borrower : BORROWER       -- Only for LIBRARY void unless borrowed or reserved.
End
```

26.

Complete the **require**, **ensure** and **class invariant** clauses for the following methods of a class LIBRARY that represents books and borrowers. Write your clauses in: (1) English, at least; and (2) in as formal a mathematical notation as possible; your clauses do not have to be executable.

```
class LIBRARY  creation make
feature

avail_books      : SET[BOOKS]   -- List of available books for lending
unavail_books    : SET[BOOKS]   -- List of unavailable books for lending
number_of_books  : INTEGER      -- Number of books owned by the library, minimum
                                -- is 10,000.

make (books : SET[BOOK]) is
  -- Build a library containing the books in the input set.

  require   books ≠ void
            ???
  ensure  ???

reserved_count : INTEGER is
  -- Returns the number of reserved books in the library

  require ???
```

```
    ensure ???

  borrow (book : BOOK; borrower : BORROWER) is
   -- The borrower checks the given book out of the library.

    require  book ≠ void and borrower ≠ void

    ensure  ???

  repair (book : BOOK) is
   -- Book needs repair.  It becomes unavailable for lending unless it is reserved.

    require  book ≠ void
              ???
    ensure  ???

  invariant  ???

end – LIBRARY
```

27.

Complete, in mathematical notation, the contract for the student association. **Do not** use agents.

Each study group has a name and a list of its members. Each student has a name and a list of the names of the study groups in which they want to be members.

```
class STUDY_GROUP                class STUDENT
  name    : STRING                 name     : STRING
  members : SET[STUDENT]           in_group : SET[STRING]
end                              end

class STUDENT_ASSOCIATION
  study_groups : SET[STUDY_GROUP]    -- All the study groups in the association
  members      : SET[STUDENT]        -- All the students in the association

make(students : SET[STUDENT] , avail_groups: SET[STRING])
   -- Creates an association with the study groups according to the preferences of the students.
   -- students – the students making the association
   -- avail_groups – the list of names of  all the study groups in the association

require ???
    -- Every student wants to be in 1 to 3 study groups.  # is "the number in the set"
-- Between 3 and 5 students want to be in each available study group.
-- The names of the study groups that students want to be in are in avail_groups.

Ensure ???

-- There are no members of the association other than the students making the association and all
-- the students are members of the association.
-- The study groups in the association are precisely those in avail_groups.

invariant

end – STUDENT_ASSOCIATION
```

28.

The members of "*The International Society of Bureaucrats*" are developing some programs to help them manage the innumerable clubs that they have. As a part of the interview process as a software designer to write the class contracts you are to complete the following sample contracts in as formal a mathematical notation as possible. **Do not** use agents.

Each committee has a name and a list of its members. Each person has a name and a list of the names of the committees on which they serve once the club has been made.

```
class COMMITTEE
  name    : STRING
  members : SET[PERSON]
end
```

```
class PERSON
  name      : STRING
  serves_on : SET[STRING]
end
```

Each committee in a club must have between three and seven members. No person can serve on more than five committees but each person must serve on at least one. Committees do not exist until the club is made.

```
class CLUB
  committees : SET[COMMITTEE]
  members    : SET[PERSON]

  make_club(people : SET[PERSON], first_committees : SET[STRING])
    -- Creates the committees according to the preferences of the starting members.
    -- people – the starting members of the club
    -- first_committees – the list of names of the initial committees

   require ???
  -- Everyone serves on 1 to 5 committees.
  -- Between 3 and 7 people serve on each committee.

  -- The names of the committees that people want to serve on are the ones in first_committees.
   ensure ???
-- The people forming the club are members and there are no other members.
-- The committees after making the club are precisely those in first_committees.
end

  change_committee(p : PERSON ; from, to : COMMITTEE)
    -- The person, p, stops serving on the from committee and starts serving on
    -- the to committee.

   require ???
 -- p serves on the from committee and is not a member of the to committee.
 -- Sufficient members will be left on the from committee after p leaves.
 -- There is space in the to committee to accept a new member.
  ensure ???
-- p has changed committees.

invariant ???
-- At all times every club member serves on 1 to 5 committees.

-- At all times every committee has 3 to 7 members.
-- The members of every committee are all the people who want to serve on that committee.

end
```

29.

A The exception handling principle deals with two ways of handling exceptions. Describe one of those two ways. Describe its pre and post conditions; using Hoare triples is most effective.

B Give an Eiffel template and show where the conditions are met.