Exercise 4
Hand-out: 26 April 2005
Due: 11 May 2005

Master Solution

2. Classes vs. Objects

It is important to make a clear distinction between classes and objects. A class is the representation of an abstract data type; it is static. An object is an instance of a class; it is dynamic (exists only at run time).

Below is extracted from the textbook (page 167), which is an example that messes up between classes and objects. Here is the extract: (omitted)

For each use of the word “object”, “thing” or “user” in the above text, underline the word if you think that the authors really meant “object”; double-underline the word if you think that they really meant “class”.

Solution:

We might identify a “User” Object in a problem space where the system does not need to keep any information about the user. In this case, the system does not need the usual identification number, name, access privilege, and the like. However, the system does need to monitor the user, responding to requests and providing timely information. And so, because of required Services on behalf of the real world thing (in this case, User), we need to add a corresponding Object to the model of the problem space.

3. Design by Contract

Below is the ADT specification of an unbounded queue (FIFO, First-In First-Out):
WRITE THE INTERFACE OF THE EIFFEL CLASS QUEUE [G] CORRESPONDING TO THE ABOVE ADT.

**Solution:**

**INDEXING**

description: "Representation of unbounded queues"

class interface

QUEUE [G]

**FEATURE -- INITIALIZATION**

make

-- Create an empty queue.

ensure

is_empty: is_empty

**FEATURE -- ACCESS**
item: G
  -- Element at the front of the queue

require
  not_empty: not is_empty

count: INTEGER
  -- Number of elements in the queue

feature -- Status report

is_empty: BOOLEAN
  -- Is queue empty?

ensure
definition: Result = (count = 0)

has (an_element: G): BOOLEAN
  -- Does this queue contain `an_element'?

feature -- Element change

put (an_element: G)
  -- Put `an_element' to the queue.

ensure
  not_empty: not is_empty
  count_increased: count = old count + 1
  has_element: has (an_element)
  item_is_an_element_if_empty_before:
    old is_empty implies item = an_element

remove
  -- Remove first entered element (`item') from the queue.

require
  not_empty: not is_empty

ensure
  count_decreased: count = old count - 1

invariant
  count_non_negative: count >= 0

end