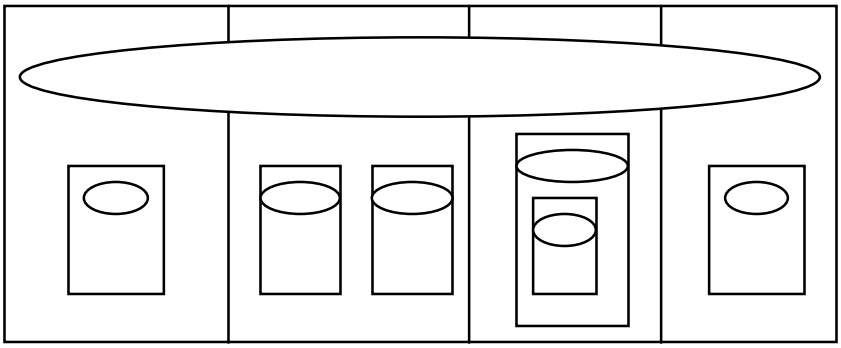
# *Object-Oriented Programming*
# *Midterm In-Class Exam*
# *9 May 1992*

### *25 out of 40 Points*

Name: _____*Solution*_____

1. [5 Points, Pp 26-32] Programming Language Topology. The various generations of programming languages have differed in many ways, but one key aspect of their evolution entails the topologies of the languages. In class and in the book, the topology of late third-generation programming languages was represented by a diagram like the following:



Briefly describe this picture. Be sure to include the following information:

- The meaning of the ovals
- The meaning of the smaller rectangles with ovals within them
- The meaning of the four larger rectangles which are covered by the largest oval
- The meaning of the encompasing rectangle

*The ovals represent data, the smaller rectangles represent subprograms, the four larger rectangles represent modules or subsystems, and the entire rectangle represents the entire software system.*

2. [3 Points, Pp. 36-37] Match the following terms with their definitions. Also, in short essay form, how are OOA, OOD, and OOP related?

| | |
|---|---|
| A. Object-Oriented Design | 1. A method that examines requirements from the perspective of the classes and objects found in the vocabulary of the problem domain. |
| B. Object-Oriented Programming | 2. A method encompassing the process of object-oriented decomposition and a notation for depicting both logical and physical as well as static and dynamic models of the system. |
| C. Object-Oriented Analysis | 3. A method in which software systems are organized as cooperative collections of objects, each of which represents an instance of some class, and whose classes are all members of a hierarchy of classes united via inheritance relationships. |

Matches:

A. _____*2*_____

B. _____*3*_____

C. _____*1*_____

*The products of OOA can serve as models from which we may start an OOD. The products of OOD can be used as blueprints for completely implementing a system using OOP methods.*

3. [2 Points, Pp.38-44] The conceptual framework of the *Object Model* includes four major elements. Match these elements to their definitions.

| | |
|---|---|
| A. Abstraction | 1. The property of a system that enables it to be decomposed into a set of cohesive and loosely-coupled entities. |
| B. Encapsulation | 2. The process of hiding all of the details of an object that do not contribute to its essential characteristics. |
| C. Modularity | 3. Denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries relative to the perspective of the user. |
| D. Hierarchy | 4. A ranking or ordering of the essential characteristics of an object that distinguish it from other objects. |

Matches:

A _____*3*_____

B _____*2*_____

C _____*1*_____

D _____*4*_____

4. [5 Points, P 40] Name and define the four kinds of abstractions. Order them from the most to the least useful.

| *Abstration* | *Meaning* |
|---|---|
| *Entity* | *an object that represents a useful model of a problem domain element* |
| *Action* | *an object that provides a generalized set of operations, all of which perform the same kind of function* |
| *Virtual Machine* | *an object that groups together operations that are all used by some superior level of control, or operations that all use some junior-level set of operations* |
| *Coincidental* | *a packaging of a set of operations that have no relation to each other* |

5. [5 Points, Pp 45-49] What features of Ada and C++ support encapsulation? How do these features support encapsulation - be specific? How are these features used during Object-Oriented Design?

*Encapsulation is supported by the ability to separate interface to a class or object from implementation. This is manifested by the Ada facility for the separation of specification and body for each of its four program units (subprogram, package, task, and generic). The corresponding facility in C++ supports class <u>declarations</u> as separate from class <u>definitions</u>.*

*These features are used during OOD by generating compilable specifications as a part of the design process. These are very precise declarations of the interfaces to objects and classes.*

6. [5 Points, Pp 59-70] The following two definitions are offered:

> *Typing* is the enforcement of the class of an object, such that objects of different types may not be interchanged, or at the most, they may be interchanged only in very restricted ways.
>
> *Persistence* is the property of an object through which its existence transcends time (i.e., the object continues to exist after its creator ceases to exist) and/or space (i.e., the object's location moves from the address space in which it was created).

How do the concepts of typing and persistence differ in Ada and C++? What similarities do the concepts of typing and persistence share in Ada and C++?

*Differences - Ada allows subtyping, such as*

***subtype COUNTER is INTEGER range 1..10;***

*and Ada automatically checks for violations of this at each assignment, raising the exception CONSTRAINT_ERROR if it fails the check. C++ does not.*

*Similarities - Ada and C++ both support strong typing, specification of member data for the life of an object, and specification of global data for the life of a system. Selection of subprograms is subject to the types of the parameters and the context of use.*