Student Name:

Student Number:

Faculty of Science
Final Examination

Computer Science COMP 304B
Object-oriented Software Design

Examiner: Prof. Hans Vangheluwe — Friday, April 15^{th}, 2005

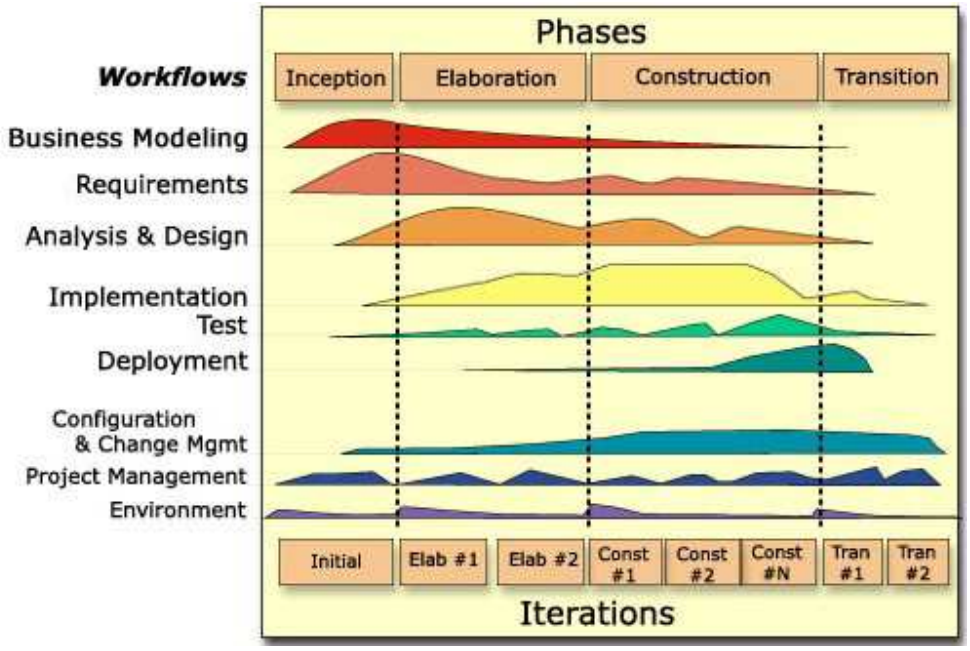
Associate Examiner: Jörg Kienzle — 14:00 – 17:00

INSTRUCTIONS:

1. Answer all questions directly on the examination paper.
2. No notes, books, calculators, computers or other aids of any type are permitted.
3. Translation dictionaries may be used.
4. The exam has 13 questions on 13 pages (not counting this front page).
5. Attempt all questions: partial marks are given for incomplete but correct answers.
6. Numbers between brackets [] denote the weight of each question. The exam is out of a total of 60 points.
7. This exam carries a weight of 35% of the total marks for CS304.
8. Use the back of the last page as scrap (it will be ignored during grading). The rear of the other pages may be used as extra space to answer questions.

Good luck !

(1) [5]



- **[3]** What can we infer from the above figure about the software process (RUP) used ? For each of your observations, indicate which part of the figure it pertains to. BTW, note that the time-scale of this figure is in the order of months.

- **[1]** Which of the features you identified in the figure are also typical for eXtreme Programming ?

- **[1]** Name two characteristics of XP which are *not* part of RUP.

(2) [5]

- **[2]** Draw in UML notation, examples of *inheritance*, *aggregation* and *composition*.

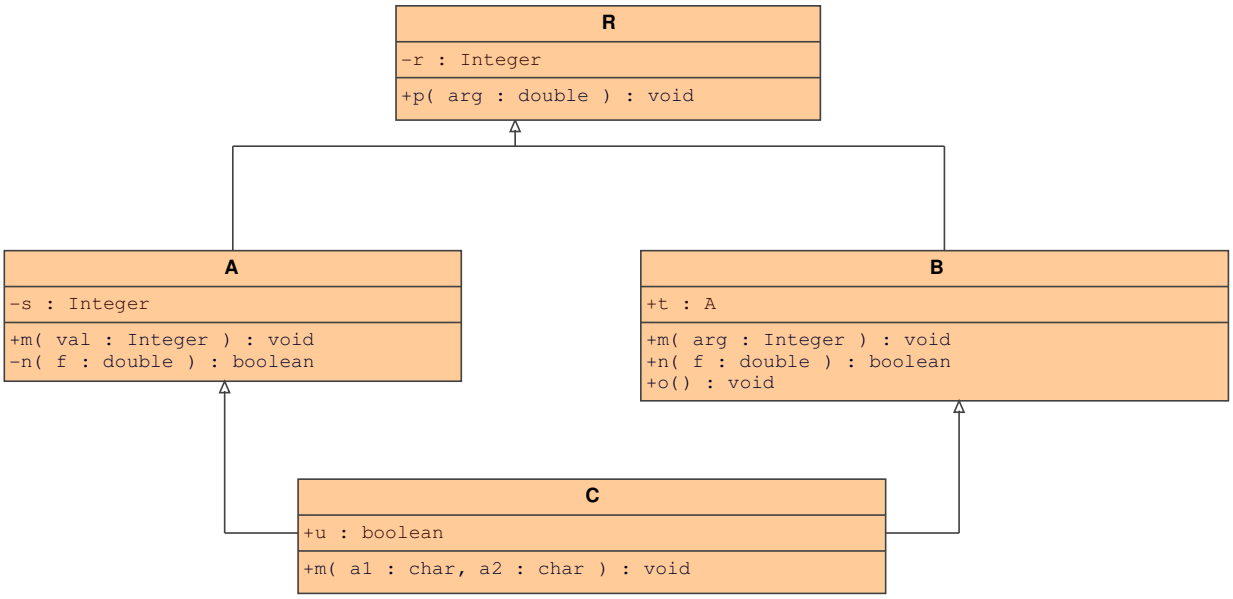
- **[0.5]** How would one check, during good Object-oriented design, whether to use an inheritance relationship between two classes ?

- **[1]** When should one use/what are the characteristics of aggregation ?

- **[1]** When should one use/what are the characteristics of composition ?

- **[0.5]** Explain "cascading delete".

(3) [6]



1. **[2]** Referring to the above figure, explain the concepts of *multiple* and *repeated* inheritance.

2. **[2]** `a, b, c` are instances of `A, B, C` respectively. If the multiple inheritance semantics of Python is used for `class C(A,B)`, which class' method (circle the appropriate one) will be invoked with

(a) `a.m(5)`	A	B	C	none
(b) `a.n(3.5)`	A	B	C	none
(c) `a.o()`	A	B	C	none
(d) `b.m(5)`	A	B	C	none
(e) `b.n(5.66)`	A	B	C	none
(f) `c.m(4)`	A	B	C	none
(g) `c.n(5.3)`	A	B	C	none
(h) `c.o()`	A	B	C	none

3. **[1]** Is the slot with name `r` corresponding to the attribute `r` in `R` a part of (the state of) instances of `C` ? YES NO.

4. **[1]** What is the *dimension* of the state space of instances of `C` ?

(4) [3]

Explain *polymorphism* and *overloading* of class methods. How are they different ? Illustrate by means of a UML diagram.

(5) [9]

- Class `F` has two public `float` attributes named `v1` and `v2`. Class `A` has a public `int` attribute `i`, a protected `String` attribute `s` and a private `F` attribute `f`. Class `A` also has a public method `m()` with a pre-condition `mApre` and a post-condition `mApost` (details not specified). The class also has a class invariant specifying that `str(i) == s`.
- Class `B` inherits from `A` and has a public `int` attribute `j`. Class `B` also has a public method `m()` with a pre-condition `mBpre` and a post-condition `mBpost` (details not specified). The class also has a class invariant specifying that `j >= 0 and i >= 0`.
- **[1]** Draw the class diagram for the above (including all information such as invariants and

pre/post-conditions).

- **[0.5]** What is the *state-space* of `F` ?

- **[0.5]** What is the *state-space* of `A` ?

- **[0.5]** What is the *state-space* of `B` ?

- **[0.5]** What is the name given to the relationship between the operation `m` in classes `A` and `B` ?

- In good OO design, what should be the relationship between
  1. **[1]** the state space of class `B` and that of class `A`;

  2. **[3]** the pre/post-conditions of `m` of class `B` and of class `A`. Under which name are the rela-

tionships also known (in type theory);

3. **[1]** the invariants of class `B` and of class `A`.

- **[1]** What is the principle of closed behaviour in good OO design ? Illustrate by means of the classes `A` and `B`.

(6) [3]

Draw a *UML Class Diagram* for a part of the design of a visual modelling environment (for graph-like structures such as UML diagrams) described below. At the heart of the design is the `DiagramElement`. A `DiagramElement` has a public `isVisible` attribute which is a `Boolean`. A `DiagramElement` *con-*

tains an arbitrary number of `Property` instances. These instances are known under the name `property` to `DiagramElement`. A `Property` has public `key` and `value` attributes which are both `String`s. A `DiagramElement` is either a `GraphElement` or a `LeafElement` (both are mutually exclusive). A `LeafElement` (which cannot be instantiated) is either a `TextElement` (with a single public `String` attribute named `text`), an `Image` (with two public `String` attributes named `url` and `mimeType`), or an *abstract* `GraphicPrimitive`. `GraphicPrimitive` is either a `Polyline` or an `Ellipse`. We will not look at the details of the two last classes. Each `GraphElement` has a `position` which is a `Point`. A `GraphElement` can either be a `GraphEdge` or a `GraphNode` (but not both at the same time; also, there are no other kinds of `GraphElement`). We don't go into the details of `GraphEdge` and `GraphNode`. A `GraphElement` *is composed of* an arbitrary number of `DiagramElement`s. These elements are known to the `GraphElement` under the name `contained`. The contained `DiagramElement`s form an *ordered* collection. In the containment relationship, the `GraphElement` is known as the `container`.

(7) [6]

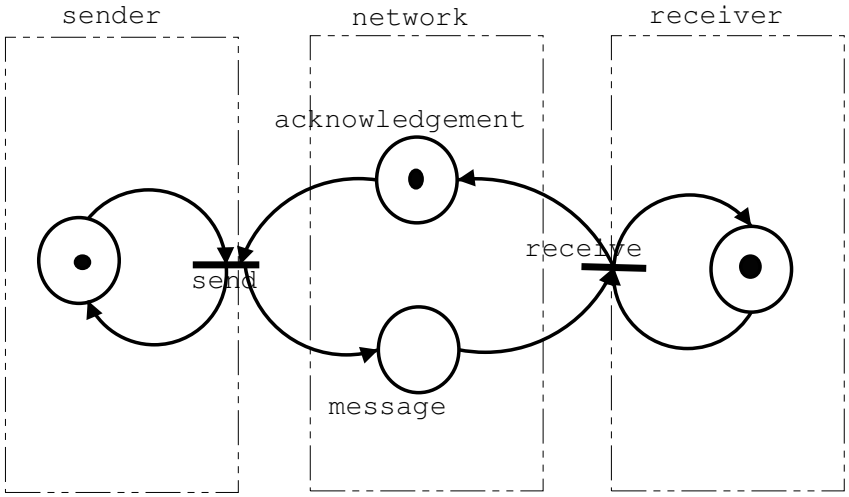
1. **[5]** Briefly describe the Statechart formalism. In particular, what do Statecharts add to State Automata ?

Draw a simple example for each of the "features".

2. **[1]** Discuss how the problem of non-determinism arises when state automata are nested. How is

the non-determinism resolved in UML Statecharts ?

(8) [2]



The above Petri Net (Activity Diagram) models a simple communication between a sender and a receiver. It is assumed messages are not lost nor damaged. A token in the sender place means the sender is ready to send. A token in the receiver place means the receiver is ready to receive. A token in the acknowledgement place means there is an acknowledgement (of receipt) message on the network. A token in the message place means there is a data message on the network.

Prove that the simple protocol modelled in this network can never (*i.e.,* for any possible behaviour of the modelled system) have two messages (a data message and an acknowledgement) simultaneously on the network.

(9) [3]

- **[2]** Draw a *UML Deployment Diagram* for an `AccountingComponent` software component with interfaces `UserServices` and `ManagerServices` implemented on a `LinuxServer`, a `UserApps`

component accessing `AccountingComponent`'s `UserServices`, running on a `Windows XP` machine. Communication between the two devices takes place over a 100Mbps TCP/IP LAN.

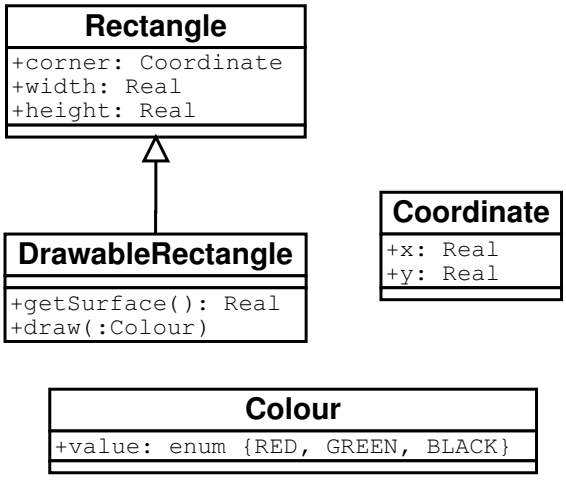
- **[0.5]** Show a minimal collection of classes which *need* to be present in the `AccountingComponent` component.

- **[0.5]** What are the *similarities* and *differences* between *components* and *packages* ?

(10) [3]

1. **[1]** Explain encumbrance in your own words. What is it ? What is it used for ?

2. **[0.5]** Give the indirect class-reference set of `DrawableRectangle` in the design given below.



3. **[0.5]** Give the (indirect) encumbrance for `DrawableRectangle`.

4. **[1]** What does a class in a low domain (the foundation domain for example) but with a high indirect encumbrance indicate ?

(11) [2]

State and explain the Law of Demeter.

(12) [6]

Describe (intent, when used, structure and behaviour) the Visitor Pattern by means of a

1. **[3]** Class Diagram and

2. **[3]** a Sequence Diagram.

(13) [7]

Two `Client` objects `client1` (first) and `client2` (later) register themselves with an appropriate message with a `Server` object `server`. Some time later, the `server`, prompted by an appropriate `make_change` method will do some local changes and *call back* both clients and invoke their appropriate methods updating the clients' view of the `Server` information. No information (about `make_change` and its effect on the `server`) may be passed to the clients by the server !

1. **[4]** Draw a *UML Class Diagram* for the above based on the *Observer Pattern* (choose names of

classes, attributes, and methods appropriately). Comment briefly.

2. **[3]** Draw a *UML Sequence Diagram* depicting the behaviour described above. Comment briefly.