Student Name:

Student Number:

Faculty of Science
Final Examination

Computer Science COMP 304B
Object-oriented Software Design

Examiner: Prof. Hans Vangheluwe — Wednesday, April 23rd, 2003

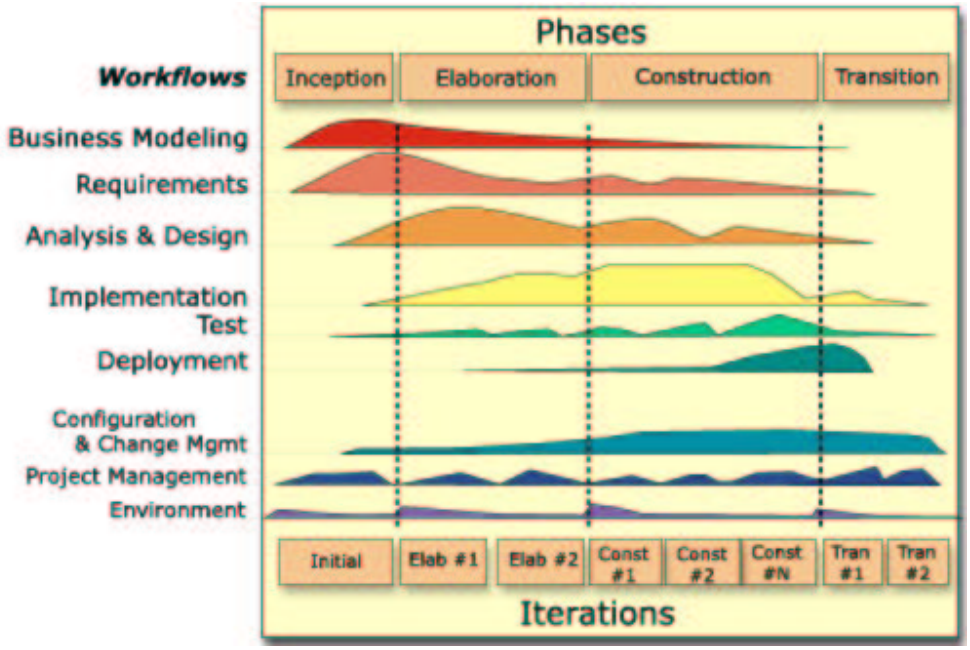
Associate Examiner: Prof. Karel Driesen — 14:00 – 17:00

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 14 questions on 12 pages.
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 45 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) [2]



The above figure depicts a software process. Which characteristics do you observe in this process ?

(2) [1]

Name the 9 generally accepted features of Object-Oriented systems (no explanation required).

1.
2.
3.
4.
5.
6.

7.

8.

9.

(3) [4]

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

- How would one check, during design, whether a relationship is an inheritance relationship or not ?

- When should one use aggregation ?

- When should one use composition ?

(4) [3]

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

(5) [4]

Draw the *class diagram* for Petri Nets (a formalism used as a rigourous basis for UML Activity Diagrams). Unless explicitly mentioned otherwise, all class properties have public visibility.

`PetriNet` has public methods `addPlace`, verb—addTransition— and `addArc`. `addArc` takes one integer argument `weight`.

A `PetriNet` is always composed of *zero or more* `Place`s and *zero or more* `Transition`s.

The `Place`s are referred to by the role name `places`. The `Transition`s are referred to by the role name `transitions`. Both `Place` and `Transition` have a `name` attribute of type `String`.

Zero or more `Place`s are connected via `pl2tr` to zero or more `Transition`s. Navigation is only possible in this direction (`Place` to `Transition`). Zero or more `Transition`s are connected via `tr2pl` to zero or more `Place`s. Navigation is only possible in this direction (`Transition` to `Place`). Both associations `pl2tr` and `tr2pl` have attributes encapsulated in the class `Arc`. `Arc` has an non-negative integer attribute `weight` and a boolean attribute `weightVisible`.

Exactly one `Place` contains zero or more `Token`s. The `Token`s are referred to (from a `Place`) by the role name `tokens`. A `Token` has one attribute `colour` of class `Colour`.

All classes have a `draw` method to render them on a canvas. All classes, with the exception of the `PetriNet` class have a private `position` attribute of class `Position`.

All `Place`s and `Transition`s must have unique names. This is a global constraint.

(6) [1]

Explain "cascading delete" of an object.

(7) [2]

Draw a State Automaton which recognizes floating point numbers without exponent (*e.g.,* 123, 123.23, 12., .123). Note how the automaton should *not* accept "." as a valid number. Add *actions* (outputs) to the automaton

updating a variable `value` such that when a valid input is recognized, `value` contains the number's value. You may use a help variable if required.

(8) [8]

The following describes a "chat" application.

A `Client` Class accepts user requests through private methods `userJoin(String)` and `userTalk(String)`. With `userJoin(crname:String)`, the user expresses the wish to join a chatroom with name `crname`. With `userTalk(msg:String)`, the user wants the string `msg` to be conveyed to the chatroom the `Client` is connected to at that time. The private method `display(String)` displays a string to the user.

Through the public method `ack(cr:ChatRoom)`, a `Client` receives a reference to a `ChatRoom` (after having requested the `Manager` to `join` the `ChatRoom`). Through the public method `nack(String)`, a `Client` receives a message (from the `Manager`) informing it that an attempt to `join` failed. A `nack` is sent when a `Client` tries to join *more than one* `ChatRoom`. Through the public method `echo(String)`, a `Client` receives a broadcast from a `Chatroom`. A `ChatRoom` sends `echo` messages to *all* `Client`s which have joined the room whenever it receives a `talk` message from one of the subscribers.

The `Manager` class is only instantiated once. All `Client`s have a reference to this unique object. `Manager` has a public method `join(Client, crn:String)`. Through it, a `Client` expresses the wish to join a `ChatRoom` with name `crn`. A `Client` may join a maximum of *one* `ChatRoom` at a time. The `Manager`'s `join` method will check this. Subsequently, the `Manager` will check whether a `ChatRoom` with the name given by `join`'s argument `crn` exists. If it doesn't, the `Manager` will instantiate a new `ChatRoom` and keep track of its name and reference. The constructor of `ChatRoom` takes as argument, the name of the `ChatRoom`. Once the `ChatRoom` instance exists, the `ChatRoom`'s `join(Client)` method is called, adding the `Client` to the `ChatRoom`'s clients List (which all need to be notified by means of an `echo(String)` message whenever a `talk(String)` message arrives).

The `ChatRoom` class has the aforementioned public constructor, `join(Client)`, and `talk(String)` methods. Furthermore, it has the public method `remove(Client)` called by a `Client` when it wants to remove itself from the `ChatRoom`.

A `Client` refers to its *unique* `Manager` through a reference in which the manager is known under the role name `mgr`. Any number of `Client`s can refer to the `Manager`. A `Client` can refer to its `Manager`, but not the other way around. The single `Manager` refers to an arbitrary number of `ChatRoom`s (but not the other way around). The rooms are referred to under the name `rooms`. An arbitrary number of `Client`s can be connected to zero or one `ChatRoom`s at any point in time. Navigation is possible in both directions. The `ChatRoom` is referred to by

the name `thechatroom`. The `Client`s are referred to by the name `clients`.

- Draw a UML Class Diagram for the design described above.

- Draw a UML Sequence Diagram for the following scenario. All messages are sent *asynchronously*.
  1. Two `Client` instances `c1` and `c2` as well as a `Manager` instance `m` have been instantiated.
  2. Client `c1` wants to `join` the chatroom with name "cs304". It contacts `m` which creates the `ChatRoom` `cs304cr`. `m` passes on the `join` request to `cs304cr` which in turn sends an appropriate `ack` message to `c1`.
  3. Client `c1` `talk`s to the chatroom it is now connected to, sending the string "hello". The chatroom reacts appropriately.
  4. Client `c2` now also wants to `join` the chatroom with name "cs304". It contacts `m` which passes on the `join` request to `cs304cr` which in turn sends an appropriate `ack` message to `c2`.
  5. Client `c2` now `talk`s to the chatroom it is now connected to, sending the string "I joined too". The chatroom reacts appropriately.
  6. Client `c2` now tries to `join` another chatroom with name "music". The manager reacts appropriately.
  7. Client `c2` now `remove`s itself from `cs304cr`.
  8. Client `c1` `talk`s to the chatroom it is still connected to, sending the string "Anyone listening ?". The chatroom reacts appropriately.

(9) [2]

Draw a Deployment Diagram for an `AccountingComponent` with interfaces `UserServices` and `ManagerServices` implemented on a `LinuxServer`, a `UserApps` component accessing `AccountingComponent`'s `UserServices`,

running an a `PCWindows2000` machine. Communication takes place over a 100Mbps TCP/IP LAN.

(10) [5]

1. Briefly describe the Statechart formalism. In particular, what do Statecharts add to State Automata ? Hint: there are three core features added to State Automata.

Draw a simple example for each of the “features”.

(11) [2]

- In good OO design, what is the relationship between the state-space of a class C and the state-space of a subclass S of C ?

- Illustrate by means of an example.

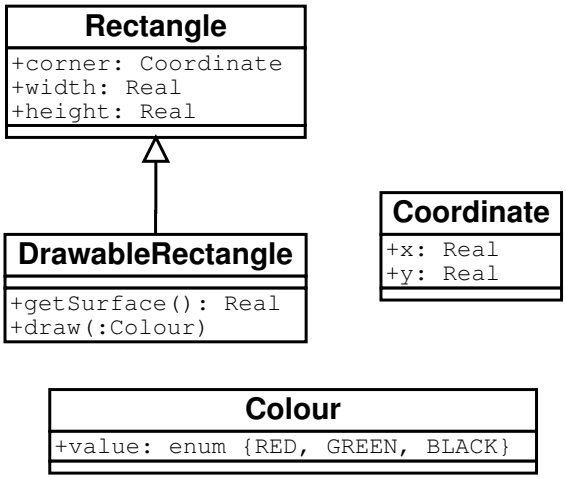
(12) [3]

Explain covariance and contravariance by means of an example.

(13) [4]

1. Explain encumbrance in your own words.

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



3. Give the indirect encumbrance for `DrawableRectangle`.

4. What does a class in a high domain (the application domain for example) but with a low indirect encumbrance indicate ?

(14) [4]

Describe the Command Pattern by means of a Class Diagram and a Sequence Diagram. Explain how to support Undo and Redo when using the Command Pattern.