Student Name:

Student Number:

Midterm Examination
COMP 304B 2005: Object-oriented Design

Examiner: Prof. Hans Vangheluwe — Friday February 18^{th}, 2005

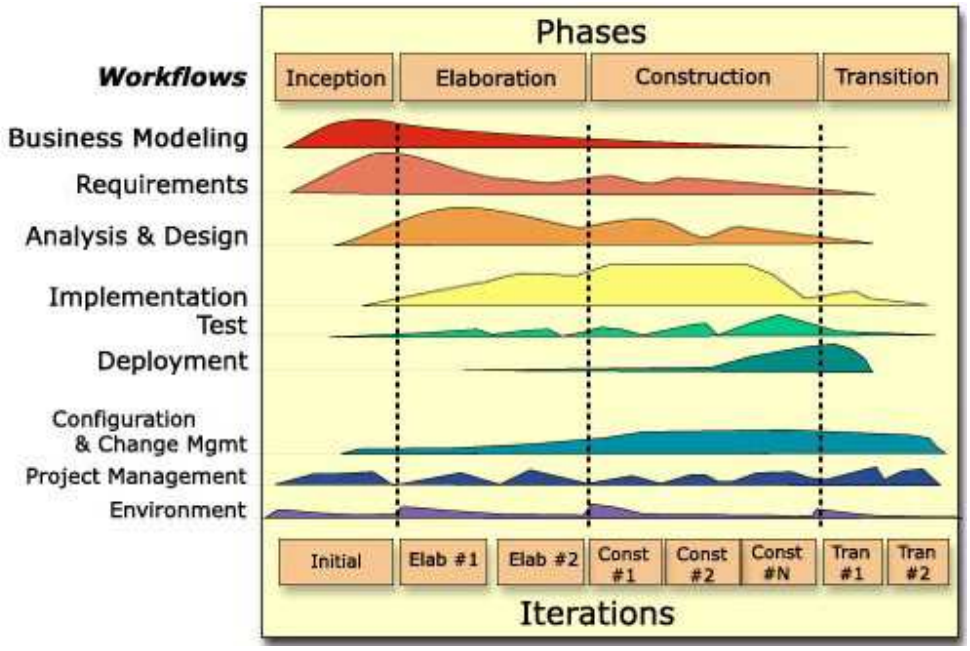
Invigilators: Sadaf Mustafiz, Jean-Sébastien Bolduc — 14:30 – 15:30

INSTRUCTIONS:

1. Answer all questions directly on the examination paper.
2. No aids of whatever type are permitted.
3. The exam has 7 questions on 8 pages (not including the cover page).
4. Attempt all questions: partial marks are given for incomplete but correct answers. If you make assumptions, mention them explicitly.
5. Numbers between brackets [] denote the weight of each question. The exam is out of a total of 40 points.
6. The midterm counts for 15 % of the total COMP 304 grade.
7. 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.

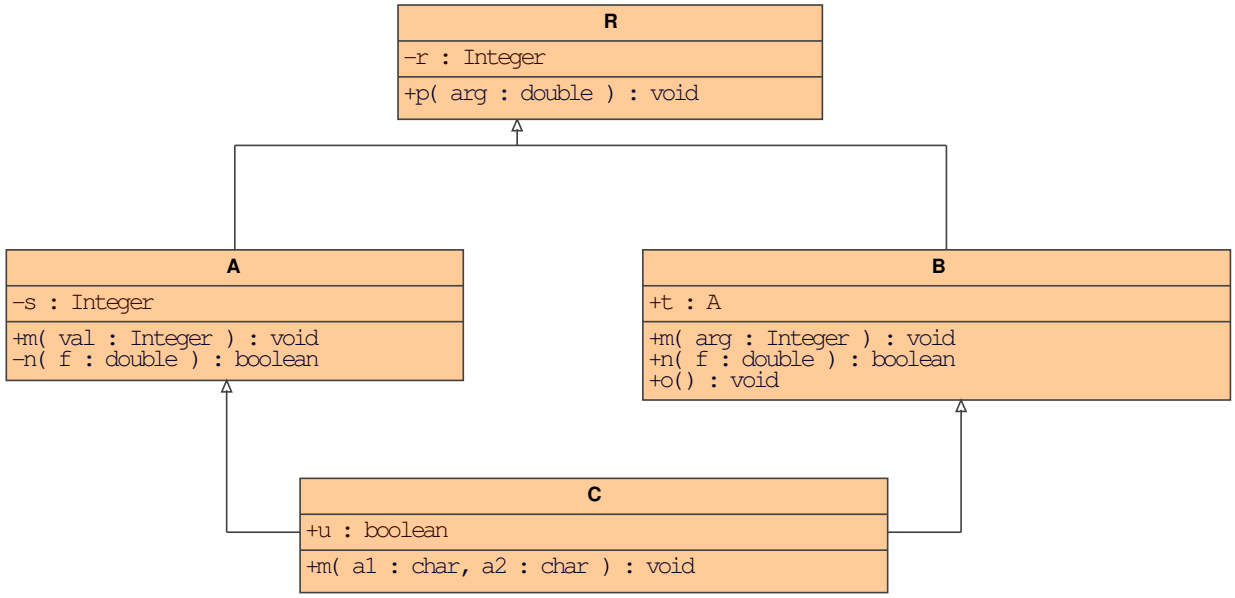
(2) [3]

- **[2]** What are the fundamental characteristics of good unit tests ? Note: the answer is *not* the different types of tests.

- **[1]** Does the following snippet of a unit test satisfy the above requirements ? Why (not) ?

```
class CaseCheck(unittest.TestCase):
    def testToRoman(self):
        """toRoman test"""
        for integer in range(1, 4000):
            numeral = roman.toRoman(integer)
            print numeral, numeral.upper()
            roman.fromRoman(numeral.upper())
            self.assertRaises(roman.InvalidRomanNumeralError,
                              roman.fromRoman, numeral.lower())
```

(3) [6]



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

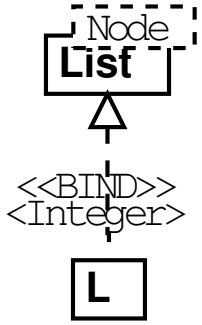
2. **[2]** a, b, c are instances of A, B, C respectively. If the multiple inheritance semantics of Python is used for class C(A,B), which class' method (circle the appropriate one) will be invoked with

(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.

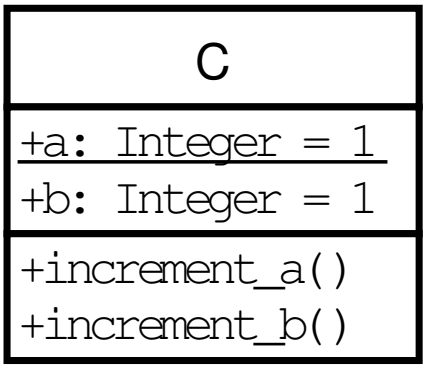
4. **[1]** What is the *dimension* of the state space of instances of `C`?

(4) [1]



Explain the above figure.

(5) [1]



For the class definition above, and after sequential execution of (using Python syntax)

```
c1=C()
c2=C()
c1.increment_a()
c1.increment_b()
c2.increment_a()
c2.increment_b()
```

give the values of

- c1.a
- c1.b
- c2.a
- c2.b

Note how a is underlined !

(6) [18]

A class PositiveInteger has two public attributes: value of type int and isZero of type enum{True, False}. The class invariant of PositiveInteger states that value >= 0.

A class A has two public attributes: a1 and a2, both instances of class PositiveInteger. The class invariant of A states that a1.value + a2.value > 10. A also has a public operation update which takes two arguments arg1 and arg2, both of type int. update has not return value but it does change A's attributes a1 and a2. update has as pre-condition which states that arg1 and arg2 must both be larger than 2. update has as post-condition which states that a1.value > 10 and a2.value > 100.

A class B inherits from A. In B, a private attribute b of type int is declared. B also defines a public operation update with *exactly* the same signature as the update declared in class A.

- **[2]** Draw the class diagram for the above (including *all* information such as pre/post-conditions).

- **[1]** What is the relationship between class invariants on the one hand and the state space of a class on the other hand ?

- **[2]** What is the state-space of (instances of) each of the classes in the class diagram ?

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

- **[2]** In good OO design, the LSP is often used. What is the LSP (give name and description) ?

- More concretely, in good OO design, what should be the relationship between (include concrete examples which satisfy the required relationships):

    1. **[2]** the invariants of class `B` and of class `A`.

2. **[2]** the state space of class `B` and that of class `A`;

3. **[4]** the pre/post-conditions of the `update` method of class `B` and of class `A`. Under which name are the relationships also known (in type theory);

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

(7) [6]

Draw a *class diagram* for the following:

- A class `Control` with as a single attribute `points`, a list of `Point` objects and as single method `moveall(delta_x, delta_y)`. `delta_x` and `delta_y` are strictly positive `float`ing point numbers. Moving can only happen either in the horizontal *or* in the vertical direction (not both at the same time). `moveall` will move all objects referred to in `points`.
- A class `Point` with attributes `x`, `y`, `R`, `theta` and appropriate methods for manipulating them. Also a `move(delta_x, delta_y)` method with `delta_x` and `delta_y` non-negative `float`ing point numbers.

Draw an *object interaction diagram* for a `Control` instance `mycontrol` which has two `Point`s `p1` and `p2` in its `points` list and which sends appropriate messages to `p1` and `p2` after receiving a `moveall(3.5, 0)` message.