

DD2455 Foundations of Object Orientation

OOTeori 08

2008-05-19

Take Home Exam (tentamen)

NOTES: (Please read these carefully!)

(i) Clearly mark at the top of **each** sheet you use: (a) your **name**, (b) the **page number**.

(ii) On the **front page** indicate (a) how many pages are contained in your work in **total**. Clearly mark: (b) your **name** (c) your **personnummer**, (d) your **e-mail** address (in case I need to contact you)

(iii) There are two ways to submit your work. (a) Your work may be handed in to NADA's studentexpedition no later than **Tuesday 20th May 2008 at 12.00 am**. After this time it will be marked as late, and marks will be subtracted. (b) If for any reason you are unable to reach the studentexpedition yourself (for example if you are at work) then you may post your manuscript to: *Studentexpeditionen, NADA, KTH, 100 44 Stockholm*. The date and time on the postmark will be taken as the date and time of your submission. The same deadline applies to manuscripts submitted by post.

(iv) If manuscripts are submitted in any other place or by any other means than those described in Part (iii) then the examiner and NADA cannot be held responsible in the case that manuscripts are lost.

(v) If you have any questions about the exam (for example, if you do not understand a question) you may call my mobile number 076 223 86 79. Please do not call before 10.00 am or after 6.00pm!

(vi) You may use your kursbunt, and any course books. You are allowed to use the internet including search engines. However, any work which you download and use must be **credited to its source**. You are **not** allowed to discuss or develop your answer with anyone else before all manuscripts have been handed in. You are **not** allowed to copy anyone else's work. By handing in your manuscript you are declaring that you have abided by these rules. In the case that cheating is suspected, actions will be taken against any students involved.

(vii) Write **clearly**. No marks will be awarded for work that I cannot read. You can write your answers in swedish or in english.

Marking Scheme

This paper is divided into two parts: part I and part II.

The grades D and E, can be achieved by answering part I questions only to the level of 80% and 70% of the available marks respectively.

The grade Fx (which allows komplementering) requires 60% of the available marks on part I.

The grades A, B and C are achieved by obtaining at least 80% on part I, and respectively 60%, 40% and 20% of the marks available on part II.

PART I

Question 1. (total 13 points) Recall from your course notes that the natural numbers can be represented as objects in the **object calculus** in the following way

```
zero = [ iszero = true, pred = sigma(x)x, succ =
        sigma(x)(x.iszero := false).pred := x ]
```

- (i) **(4 points)** Extend this zero object to a new object zero' which has one extra test isEven such that if x is an even number then x.isEven should evaluate to true, while if x is an odd number then x.isEven should evaluate to false. You can make use of a Boolean if_then_else operator, as used in the 2007 exam model answers. (This operator was also used in the last section of the course on “Logic of Objects”).

Hint: Consider the following recursive definition:

isEven(0)=true,

isEven(1)=false,

isEven(x+1)= if isEven(x) then false else true

- (ii) **(5 points)** You should now test your solution to part (i) by proving that the object term zero'.succ.isEven (where zero' is the object you defined in your answer to part(i)) reduces to the value false. (Of course one is odd!) You will need to carefully define each single reduction step, including all reductions used for any if_then_else operators. **Hint:** you may find it useful to first establish what the object term zero'.succ reduces to?

- (iii) **(iii) (4 points)** You will now extend the object zero, described in part (i), in a different way, by adding a recursive binary subtraction method sub so that x.sub(y) evaluates to x - y. Your method should satisfy the usual recursive laws:

sub(0, y) = y

$\text{sub}(x, 0) = 0$

$\text{sub}(x, y) = \text{sub}(x-1, y-1)$ for $x > 0, y > 0$

Hints: Recall how unary operators can be added to the object calculus by using the lambda calculus notation for functions. See for example the model answers to the 2007 take home exam.

Question 2. (total 12 points) Write **Java Modeling Language (JML)** specifications consisting of pre- and post-conditions to formally model the following requirements on the following Java methods:

- (i). (3 points) A method `Integer Max(int[] A)` which returns the largest element in an array A, if A is not null.
- (ii). (3 points) A method `int GCD(int x, int y)` must produce a greatest common divisor of its non-negative arguments x and y.
- (iii). (6 points) A method `float smallestRoot(float a, float b, float c)` must produce the smallest of the two roots of a quadratic polynomial $ax^2 + bx + c$, for floats a, b and c if these exist. (Can one root exist without another?)

PART II

Question 3 . (total 25 points) In this question you are going to add a simple *dynamic* (also known as *runtime checkable*) *subtype mechanism* to the **Seqool language**. Since this requires several changes to the language, the questions below will lead you through the required changes step by step.

Let $C = \{ c_1, c_2, \dots \}$ be a set of class names for a seqool program. A *subtype declaration* $<$ is a binary relation on class names, formally, $<$ is a subset of the Cartesian product $C \times C$. Intuitively, if $c_i < c_j$ then c_i is a subtype of c_j or c_j is a supertype of c_i . We will take $c_i < c_j$ to mean that both:

- (a) an object of type c_i can be used everywhere that an object of type c_j is used, and
- (b) an object of type c_i can invoke any method of an object of type c_j .

Let \ll denote the *reflexive transitive closure* of $<$, thus: (i) $c_i < c_j$ implies $c_i \ll c_j$, (ii) $c_i \ll c_i$ and, (iii) $c_i \ll c_j$ and $c_j \ll c_k$ implies $c_i \ll c_k$.

(i). (3 points) Redefine the set $SExp(c, d)$ for c in C and d in C^+ of all expressions with possible side effects so that a method call

$$e_d^c . m_{d_0, \dots, d_n}^{c_0} (f_{d_1}^{c_1}, \dots, f_{d_n}^{c_n})$$

by a c class expression e_d^c can be made without knowing (at compile time) whether the owner class c_0 of method m is a superclass of d' . Also the actual parameter types d'_1, \dots, d'_n are not known to be subtypes of the method m parameter types d_1, \dots, d_n (at compile time). It is also not necessary to know (at compile time) whether the return type d_0 of method m is a subtype of d .

(ii). (3 points) Redefine the set $Stat(c)$ for c in C of all statements for a class c , so that any assignment is legal regardless of the types of the left and right hand sides of the assignment statement. (This will be checked at runtime.)

By now you have an **extremely type unsafe language**!! You will now have to remedy this by redefining the execution rules for the operational semantics of this extended Seqool language.

(iii). (3 points) Extend the definition of the set $State$ of all semantic states for the abstract virtual machine (AVM) that executes the basic Seqool language so that at run time:

- (a) a c class attribute x_d^c in $Ivar(c)_d$ of type d (for c in C) can be bound to any value of type $d' \ll d$, and
- (b) a temporary variable u_d in $Tvar_d$ of type d can be bound to any value of type $d' \ll d$.

(iv). (3 points) Extend the definition of the state of the AVM for basic Seqool so that (a) *normal* and *runtime type exception* states are allowed. (The idea is that extended AVM will enter a *runtime type exception* state when a runtime type error occurs, and remain stuck in that state.)

(v). (3 points) Extend the state of your AVM with a class attribute type lookup function

$$\text{Type} : \bigcup_{c \in C} O^c \times Ivar(c)_d \rightarrow C$$

and a temporary variable type lookup function

$$\text{Type} : Tvar_d \rightarrow C .$$

With reference to the variable binding functions σ_2 and σ_3 for class attributes and temporary variables, define $\text{Type}(o, x_d^c)$ and $\text{Type}(u_d)$ so that:

- (a) $\text{Type}(o, x_d^c) = c'$ if the c class attribute x_d^c in the object o is currently bound to a value of type c' , and
- (b) $\text{Type}(u_d) = c'$ if the temporary variable u_d is currently bound to a value of type c' .

(vi). (4 points) Redefine the *execution rule* for the assignment statement

$$x_d^c = e_{d'}^c$$

(Rule 1.a in the course notes) so that if the value currently bound to $e_{d'}^c$ is of type $d' \ll d$ then that value can be assigned to x_d^c otherwise a *runtime type exception* occurs and execution stops.

(vii). (4 points) Redefine the execution rule for the assignment statement

$$x_d^c = e_{d'}^c . m_{d_0, \dots, d_n}^{c_0} (f_{1_{d_1}^c}, \dots, f_{n_{d_n}^c})$$

(Rule 3.a in your notes) so that if:

- (a) the type of the value currently bound to $e_{d'}^c$ is a subtype of c_0 (so that the method m can legitimately be called by object $e_{d'}^c$), and
- (b) the types of the values currently bound to the actual parameter expressions f_1, \dots, f_n are subtypes of d_1, \dots, d_n , and
- (c) the type of the return value produced by executing the body of method m is a subtype of d

then the return value produced by executing the body of method m is stored in x_d^c . Otherwise a *runtime type exception* occurs and execution stops.

(viii). (2 points) Carefully explain one advantage and one disadvantage of using dynamic/runtime typing as opposed to static/compile time typing.