Introduction to Programming

ETH Zürich

Date: 6.2.2017

Family name, first name: ....

Student number: ....

I confirm with my signature, that I was able to take this exam under regular circumstances and that I have read and understood the directions below.

Signature: ....

Directions:

- Exam duration: 120 minutes.
- Except for a dictionary you are not allowed to use any supplementary material.
- Use a pen (**not** a pencil)!
- Please write your student number onto **each** sheet.
- All solutions can be written directly onto the exam sheets. If you need more space for your solution ask the supervisors for a sheet of official paper. You are **not** allowed to use other paper.
- Only one solution can be handed in per question. Invalid solutions need to be crossed out clearly.
- Please write legibly! We will only correct solutions that we can read.
- Manage your time carefully (take into account the number of points for each question).
- Please **immediately** tell the exam supervisors if you feel disturbed during the exam.
- All program elements should be given in the programming language used in the course: Eiffel. All programming-language-related questions refer, unless otherwise noted, to Eiffel.

Good luck!

Question	Number of possible points	Points
1	10	
2	16	
3	16	
4	10	
5	18	
Total	70	

1 Multiple Choice (10 points)

For each statement about Eiffel and its methodology, decide if it is true or false and put a checkmark in the corresponding box. Each correct answer is worth 0.5 points. An incorrect answer or no answer is worth 0 points.

Example:

		T	F	
a.	$2 \times 4 = 8$	☐	☐	0 points
b.	"Rösti" is a kind of sausage.	☒	☐	0 points
c.	C is an object-oriented programming language.	☐	☒	0.5 points

---

	T	F
1. A feature is a query in case it modifies an object.	☐	☐
2. A name clash is acceptable since Eiffel supports feature overloading.	☐	☐
3. For any object `o`, the feature call `o.is_equal(o)` always returns `True`.	☐	☐
4. Binary search trees provide log-time ($O(\log n)$) access in the worst case.	☐	☐
5. An object may be deferred or effective.	☐	☐
6. A postcondition must hold before and after the execution of any feature that is available to the clients of the class.	☐	☐
7. A creation procedure has to ensure that after its execution, the new instance will satisfy the invariant of the class.	☐	☐
8. A procedure that is exported to `NONE` can be used as creation procedure.	☐	☐
9. Polymorphism is the capability of objects to change their types at run time.	☐	☐
10. If a routine redefinition contains a new postcondition, this condition has to hold in addition to the inherited postcondition.	☐	☐
11. If `C` is a deferred class, then no entity with static type `C` can exist in a program.	☐	☐
12. A loop invariant is allowed to be violated between the execution of any two instructions in the loop body.	☐	☐
13. Calling routine `r` can result in infinite recursion only if the body of `r` contains a call to `r`.	☐	☐
14. An empty precondition is equivalent to the precondition `False`.	☐	☐
15. A deferred class can inherit from an effective class.	☐	☐
16. An expanded class `C` can have an attribute of type `C`.	☐	☐
17. A deferred class can have attributes.	☐	☐
18. Void references can be the target of a successful call.	☐	☐
19. A class can always call all features of its immediate parent classes.	☐	☐
20. Different generic derivations of the same generic class always conform to each other.	☐	☐

2 Inheritance and Polymorphism (16 Points)

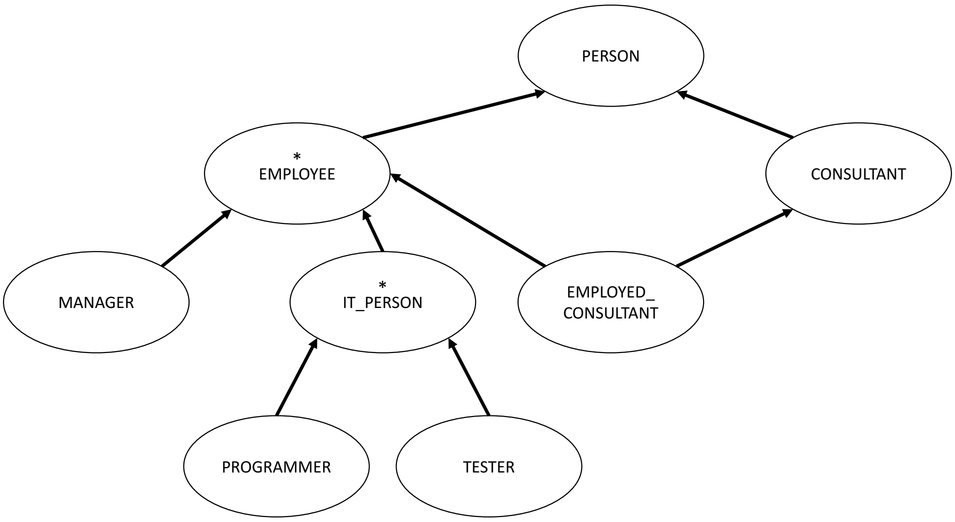


Figure 1: BON Diagram

This part is based on the following classes:

```
class
    PERSON

create
    make

feature
    name: STRING

feature
    make (a_name: STRING)
        require
            a_name /= Void and then not a_name.is_empty
        do
            name := a_name
        ensure
            name = a_name
        end
end
```

```
deferred class
    EMPLOYEE

inherit
    PERSON

feature
```

```
    work
        deferred
    end
end
```

```
class
    CONSULTANT

inherit
    PERSON
        rename
            make as make_person
        end

create
    make

feature
    consulting_topic: STRING

feature

    make (a_name: STRING; a_consulting_topic: STRING)
        do
            make_person (a_name)
            consulting_topic := a_consulting_topic
        end

    consult (a_person: PERSON)
        do
            print (name + " consults " + a_person.name + " on " + consulting_topic
                + ". ")
        end

    prepare
        do
            print (name + " prepares topic " + consulting_topic + ". ")
        end
end
```

```
class
    MANAGER

inherit
    EMPLOYEE
        rename
            work as manage,
            make as make_employee
        end
create
    make

feature
    subordinate: EMPLOYEE
```

```
feature
    make (a_name: STRING; a_subordinate: EMPLOYEE)
        do
            make_employee (a_name)
            subordinate := a_subordinate
            print ("Subordinate: " + subordinate.name + ". ")
        end

    manage
        do
            print (name + " issues a task for " + subordinate.name + ". ")
            subordinate.work
        end
end
```

```
deferred class
    IT_PERSON

inherit
    EMPLOYEE
        redefine
            make
        end

feature
    discuss_tech (a_colleague: IT_PERSON)
        do
            print (name + " discusses with " + a_colleague.name + ". ")
        end

    make (a_name : STRING)
        do
            precursor (a_name)
            print (a_name + " ")
        end
end
```

```
class
    EMPLOYED_CONSULTANT

inherit
    CONSULTANT
        rename
            prepare as study
        select
            make_person
        end

    EMPLOYEE
        rename
            make as make_employee,
            work as study
        end
```

```
create
    make
end
```

```
class
    PROGRAMMER
inherit
    IT_PERSON
        rename
            work as code
        redefine
            discuss_tech
        end
create
    make_c_programmer, make_eiffel_programmer

feature
    programming_language: STRING

feature
    make_c_programmer (a_name: STRING)
        do
            make (a_name)
            programming_language := "C"
            print ("programs in C. ")
        end

    make_eiffel_programmer (a_name: STRING)
        do
            make (a_name)
            programming_language := "Eiffel"
            print ("programs in Eiffel. ")
        end

    code
        do
            print (name + " writes some code. ")
        end

    discuss_tech (a_colleague: IT_PERSON)
        do
            precursor (a_colleague)
            print ("It is all about " + programming_language + ". ")
        end
end
```

```
class
    TESTER

inherit
    IT_PERSON
        rename
            work as test,
```

```
            make as make_it_guy
        end

create
    make

feature
    colleague: IT_PERSON

feature
    make (a_name: STRING; a_colleague: IT_PERSON)
        do
            make_it_guy(a_name)
            colleague := a_colleague
            print("has colleague " + colleague.name + ". ")
        end

    test
        local
            working_colleague : IT_PERSON
        do
            working_colleague := colleague
            if attached {TESTER} colleague as tester then
                working_colleague := tester.colleague
            end
            working_colleague.work
            print (name + " tests work from " + working_colleague.name + ". ")
        end
end
```

Tasks Indicate, for each of the code fragments below, if it compiles by checking the corresponding box. If the code fragment does not compile, explain why this is the case and clearly mark the line that does not compile. If the code fragment compiles, specify the text that is printed to the console when the code fragment is executed.
Given the following variable declarations:

```
person: PERSON
employee: EMPLOYEE
it_person: IT_PERSON
manager: MANAGER
programmer: PROGRAMMER
tester1: TESTER
tester2: TESTER
consultant: CONSULTANT
employed_consultant: EMPLOYED_CONSULTANT
```

Example 1:

```
create programmer.make_eiffel_programmer("Eric")
programmer.code
```

Does the code compile? ☒ Yes ☐ No
Output/error description
Eric programs in Eiffel. Eric writes some code.

Example 2:

```
create programmer.make_java_programmer("Jason")
```

Does the code compile? ☐ Yes ☒ No
Output/error description
Unknown identifier "make_java_programmer".

Task 1

```
create it_person.make("Igor")
create programmer.make_eiffel_programmer("Eric")
it_person.discuss_tech(programmer)
```

Does the code compile? ☐ Yes ☐ No
Output/error description

. . . .

. . . .

. . . .

Task 2

```
create programmer.make_eiffel_programmer("Eric")
employee := create {MANAGER}.make_manager("Max", programmer)
manager := employee
manager.manage
```

Does the code compile? ☐ Yes ☐ No
Output/error description

. . . .

. . . .

. . . .

Task 3

```
create programmer.make_c_programmer("Chris")
it_person := create {PROGRAMMER}.make_eiffel_programmer("Eric")
it_person.discuss_tech(programmer)
```

Does the code compile? ☐ Yes ☐ No
Output/error description

. . . .

. . . .

Task 4

```
create {EMPLOYED_CONSULTANT} employee.make("Steve","Data Model")
create person.make("Hector")
employee.work
```

Does the code compile? ☐ Yes ☐ No
Output/error description

. . . .

. . . .

Task 5

```
it_person := create {PROGRAMMER}.make_c_programmer("Chris")
it_person.code
```

Does the code compile? ☐ Yes ☐ No
Output/error description

. . . .

. . . .

Task 6

```
create programmer.make_eiffel_programmer("Eric")
create tester1.make("Ted", programmer)
employee := create {MANAGER}.make("Max", tester1)
employee.work
```

Does the code compile? ☐ Yes ☐ No
Output/error description

. . . .

. . . .

. . . .

. . . .

Task 7

```
create programmer.make_c_programmer("Arnold")
create tester1.make("Robert", programmer)
create tester2.make("Delores",tester1)
tester2.test
```

Does the code compile? ☐ Yes ☐ No
Output/error description

. . . .

. . . .

. . . .

. . . .

3 Specifying Software through Contracts (16 points)

Your start-up company is designing a new webshop framework and you are tasked with the back-end of the software. You decided to employ Design-by-Contract to achieve high correctness for your software.

Here are some facts:

- A fresh session starts with an empty basket with neither payment method nor a shipping address specified.
- Only positive quantities of items can be added to the basket.
- Only existing items in the basket can be removed.
- You can only check out if the payment method was selected and the shipping address has been set.
- Upon login, the payment method and shipping address are set to the preferred one if they are not yet set.
- Upon checkout, if there is at least one product in the basket which costs more than 100, mark in the session that the next order will be discounted. Also, upon checkout, empty the basket.

Task The following classes *ITEM* and *SESSION* are the heart of the webshop. Please fill in the missing contracts (preconditions, postconditions, and class invariants), so that all facts from the informal specification above as well as other necessary properties are reflected.

Please note:

- The number of dotted lines is not indicative of the number of missing contract clauses.
- You need to write **True** at places where you think no explicit contract is necessary: leaving a contract empty gives you 0 point for that section.
- Don't forget to express all consistency constraints in the invariants. For instance, you should describe the interrelation between the attribute *total* and the content of attribute *items*
- The system is not void safe. You need to handle **Void** references where it makes sense.
- Besides other features, you can assume the following features to be available in class *HASH_TABLE*:

```
class
    HASH_TABLE [G, K -> detachable HASHABLE]

feature

    for_all (test: FUNCTION [ANY, TUPLE [G,K], BOOLEAN]): BOOLEAN
            -- Is 'test' true for all key-value pairs?

    there_exists (test: FUNCTION [ANY, TUPLE [G,K], BOOLEAN]): BOOLEAN
            -- Is 'test' true for at least one key-value pair?
```

- The following features from class *CUSTOMER* may be useful:

```
class CUSTOMER

feature
    preferred_payment: PAYMENT_METHOD
            -- The preferred payment method

    address: ADDRESS
            -- The address of the customer

    -- Other features omitted.
end
```

Listing 1: Class *ITEM*

```
class
    ITEM

inherit
    HASHABLE

create
    make

feature

    make (a_name: STRING; a_price: INTEGER)
        require

            ..........................................................................

            ..........................................................................

            ..........................................................................
        do
            -- Implementation omitted.
        ensure

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................
        end

feature -- Access

    price: INTEGER
        -- Current price of the item.

    name: STRING
        -- Name of the item
```

```
feature -- Inherited

    hash_code: INTEGER
            -- Hash code value
        do
            -- Implementation omitted.
        end

    is_equal (a_other: like Current): BOOLEAN
        do
            -- Implementation omitted.
        end

invariant

    ..........................................................................

    ..........................................................................

    ..........................................................................
end
```

Listing 2: Class *SESSION*

```
class
    SESSION

create
    make

feature

    make
        require

            ..................................................................

            ..................................................................

            ..................................................................
        do
            -- Implementation omitted.
        ensure

            ..................................................................

            ..................................................................

            ..................................................................

            ..................................................................

            ..................................................................

            ..................................................................
        end
```

```
feature -- Access

    items: HASH_TABLE[INTEGER, ITEM]
        -- Items in the basket, table from {ITEM} to {INTEGER}

    total: INTEGER
        -- Current total price of all items in the basket.

    discount_next_order: BOOLEAN
        -- If true, make a discount on the next order (should be set to true
        -- upon checkout if there is at least one product costing more than 100)

    payment_method: PAYMENT_METHOD
        -- The payment method selected by the customer.

    shipping_address: ADDRESS
        -- Shipping addresses

feature -- Basic operations

    add_item (a_quantity: INTEGER; a_item: ITEM)
            -- Add 'a_quantity' 'a_item's to the basket.
        require

            ..........................................................................

            ..........................................................................

            ..........................................................................
        do
            -- Implementation omitted.
        ensure

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................
        end


    remove_item (a_quantity: INTEGER; a_item: ITEM)
            -- Remove 'a_quantity' 'a_item's from the basket.
        require

            ..........................................................................

            ..........................................................................

            ..........................................................................
        do
            -- Implementation omitted.
```

```
    ensure

        ..........................................................................

        ..........................................................................

        ..........................................................................

        ..........................................................................

        ..........................................................................
    end

check_out
        -- Check out the basket.
    require

        ..........................................................................

        ..........................................................................

        ..........................................................................

        ..........................................................................
    do
        -- Implementation omitted.
    ensure

        ..........................................................................

        ..........................................................................

        ..........................................................................

        ..........................................................................

        ..........................................................................

        ..........................................................................

        ..........................................................................

        ..........................................................................
    end

set_payment_method (a_payment_method: PAYMENT_METHOD)
        -- Set the payment method
    require

        ..........................................................................

        ..........................................................................

        ..........................................................................
    do
        -- Implementation omitted.
```

```
    ensure

        ...........................................................................

        ...........................................................................

        ...........................................................................
    end

set_shipping_address (a_address: ADDRESS)
        -- Set the shipment address
    require

        ...........................................................................

        ...........................................................................

        ...........................................................................
    do
        -- Implementation omitted.
    ensure

        ...........................................................................

        ...........................................................................

        ...........................................................................
    end

log_in (a_customer: CUSTOMER)
        -- 'a_customer' has logged in
    require

        ...........................................................................

        ...........................................................................

        ...........................................................................

    do
        -- Implementation omitted.
    ensure

        ...........................................................................

        ...........................................................................

        ...........................................................................

        ...........................................................................

        ...........................................................................

        ...........................................................................

    end
```

```
feature -- Extra space for features that you might need to create for writing the
    contracts

    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................

invariant

    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
    ..........................................................................
end
```

4 Recursion: Catalan Numbers (10 points)

A *Catalan number* is a natural number, which is part of a sequence that occurs in many interesting counting problems.
You can compute a Catalan number using the following formula:

$$C_0 = 1, \quad C_{n+1} = \sum_{i=0}^{i=n} C_i C_{n-i} \quad \text{for} \quad n \geq 0$$

By applying the formula above, you can see that the first few Catalan numbers for $n = 0, 1, 2, 3, 4, 5, 6, 7, ...$ are $1, 1, 2, 5, 14, 42, 132, 429, ...$

Task Your task is to implement the recursive function *catalan_number*, that computes Catalan number C_n.

```
class CATALAN

    catalan_number (n: INTEGER): INTEGER
            -- Compute Catalan's number Cn.
        require
            n_non_negative: n >= 0
        local

            ..........................................................................

            ..........................................................................

        do

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................
```

```
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
            ..........................................................................
        end
end
```

5 Data Structures: Bounded Priority Queue (18 points)

A mobile network operator wants to introduce a new near-realtime monitoring system to track the 100 most valuable customers. To this end, you are feeding a slightly modified *bounded priority queue* with a stream of revenue data such that the queue contains, at any time, the revenue data of the currently top 100 customers.

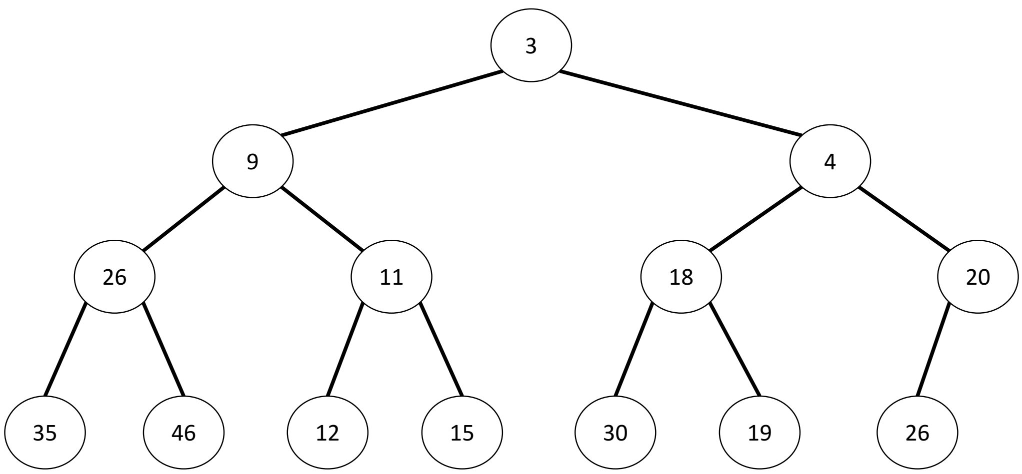
A typical way to implement a priority queue is by using a *min-heap* data-structure. A min-heap is a complete binary tree in which the value of a node is smaller than all the values in its subtrees.

Heaps may be represented in a very efficient way (as an implicit data structure) using an array alone, without requiring pointers between elements. The first element will contain the root. The next two elements of the array contain its children. The next four contain the four children of the two child nodes, etc. Thus the left child of a node at position n would be at position $2n$ and the right child is always right next to the left child. This allows moving up or down the tree by doing simple index computations. Figure 2 shows an example binary heap and its corresponding array representation.

After an element is inserted or replaced, the heap property needs to be reestablished by the internal operations *sift_down* and *sift_up*, respectively.

In the following implementation, the feature *process* is called for each revenue in the stream of customers' revenues. This feature conditionally inserts or replaces a customer's revenue in the queue, if:

- the queue is not full, OR
- the queue is full but the revenue under processing is larger than any of the revenues in the heap (and thereby evicting the smallest revenue)



3	**9**	**4**	**26**	**11**	**18**	**20**	**35**	**46**	**12**	**15**	**30**	**19**	**26**

Figure 2: Example of a priority queue implemented using a binary heap, and its respective array representation.

Task Your task is to complete the implementation of features *parent_index*, *left_child_index*, *right_child_index*, and an efficient, non-recursive version of *sift_down*.

Listing 3: Class *BOUNDED_PRIORITY_QUEUE*

```
class
    BOUNDED_PRIORITY_QUEUE [G −> COMPARABLE]

create
    make

feature {NONE}
    heap: ARRAY[G]
    max_size: INTEGER
    count: INTEGER

feature {NONE} −− Initialization

    make (n: INTEGER)
            −− Allocate heap space.
        do
            create heap.make(1, n)
            count := 0
            max_size := n
        end

feature −− Processing of new elements

    process (v: like item)
            −− Conditionally insert/replaces the item v in the queue.
        do
            if count < max_size then
                count := count + 1
                heap.force (v, count)
                sift_up
            elseif v > heap.item (1) then
                heap.put (v, 1)
                sift_down
            end
        end

feature {NONE} −− Private access parent/children indices

    parent_index (i: INTEGER): INTEGER
        −− Get index of parent of entry at position i
        do
            ..........................................................................

            ..........................................................................
        end

    left_child_index (i: INTEGER): INTEGER
        −− Get index of left child of entry at position i
```

```
        do
            ..........................................................................

            ..........................................................................
        end

    right_child_index (i: INTEGER): INTEGER
        -- Get index of right child of entry at position i
        do
            ..........................................................................

            ..........................................................................
        end

feature {NONE} -- Heap internal operations

    sift_down
            -- Sift top element down until the heap property holds again
            -- Efficient, non-recursive implementation required!
        local

            ..........................................................................

            ..........................................................................

            ..........................................................................

            ..........................................................................
        do
            from

                ......................................................................

                ......................................................................
            until

                ......................................................................
            loop

                ......................................................................

                ......................................................................

                ......................................................................

                ......................................................................

                ......................................................................

                ......................................................................

                ......................................................................

                ......................................................................

                ......................................................................
```

```
                ..................................................................
                ..................................................................
                ..................................................................
                ..................................................................
                ..................................................................
                ..................................................................
            end
            ......................................................................
            ......................................................................
            ......................................................................
            ......................................................................
            ......................................................................
            ......................................................................
            ......................................................................
        end

    sift_up
        -- Sift last element up until the heap property holds again
        local
            i, j: INTEGER
            up, down: like item
            stop: BOOLEAN
        do
            from
                i := count
                down := heap.item (i)
            until
                stop or i = 1
            loop
                j := parent_index (i)
                up := heap.item (j)
                if up > down then
                    heap.put (up, i)
                    i := j
                else
                    stop := true
                end
            end
            heap.put (down, i)
        end

end
```