Mock exam 2
1 Terminology

• 1. A class...
  ✓ is the description of a set of possible run-time objects to which the same features are applicable.
  ○ can only exist at runtime.
  ○ cannot be declared as expanded; only objects can be expanded.
  ✓ may have more than one creation procedure.
1 Terminology

• 2. Procedures, functions and attributes.
  o A query needs to be a function.
  o A function cannot modify any objects.
  ✓ An attribute is stored directly in memory.
  o A procedure can return values that are computed.
1 Terminology

• 3. What are the possible changes in a function redefinition?
  ✓ To change the implementation.
  ✓ To change the list of argument types.
  ✓ To change the contract.
  ✓ To change the result type.
1 Terminology

• 4. Clients and suppliers.
  o A supplier of a software mechanism is a system that uses the mechanism.
  o A client of a software mechanism cannot be a human.
  ✓ A client of a software mechanism is a system of any kind, software or not, that uses the mechanism. For its clients, the mechanism is a supplier.
  ✓ A supplier of a set of software mechanisms provides an interface to its clients.
1 Terminology

• 5. Polymorphism.

✓ A data structure is polymorphic if it may contain references to objects of different types.
✓ An assignment of argument passing is polymorphic if the target variable and source expression have different types.

○ Polymorphism is the capability of objects to change their types at runtime.
✓ An entity or expression is polymorphic if, as a result of polymorphic attachments, it may at run time become attached to objects of different types.
2 Design by Contract
2 Design by Contract

```
class CAR

create make

feature {NONE} -- Creation
    make
        -- Creates a default car.
    require

feature {ANY} -- Access

    is_convertible : BOOLEAN
        -- Is the car a convertible (cabriolet)? Default: no.

    doors: LIST [CAR_DOOR]
        -- The doors of the car. Number of doors must be 0, 2 or 4. Default: 0.

    color: COLOR
        -- The color of the car. ‘Void’ if not specified. Default: ‘Void’.

-- Lösung: keine Precondition
```

do
    create {LINKED_LIST [CAR_DOOR]} doors.make
ensure
    not is_convertible
        doors /= Void and then doors.count = 0
    color = Void
end
2 Design by Contract

feature {ANY} -- Element change

set_convertible ( a_is_convertible : BOOLEAN)
   require
     a_is_convertible_exists: a_is_convertible /= Void

   -- Lösung: keine Precondition

   do
     is_convertible := a_is_convertible
   ensure
     convertible_set: is_convertible = a_is_convertible

end
2 Design by Contract

```haskell
set_doors (a_doors: ARRAY [CAR_DOOR])
  require
    a_doors /= Void implies (a_doors.count = 0 or a_doors.count = 2 or a_doors.count = 4)

-- a_doors /= Void and then (a_doors.count = 0 or a_doors.count = 2 or a_doors.count = 4)
```
local
door_index: INTEGER

do
  doors.wipe_out
  if a_doors /= Void then
    from
    door_index := 1
  invariant
    doors.count + 1 = door_index
    door_index >= 1 and door_index <= a_doors.count + 1
  until
    door_index > a_doors.count
  loop
    doors.extend (a_doors [door_index])
    door_index := door_index + 1
  variant
    a_doors.count - door_index + 1
  end
end
2 Design by Contract

```plaintext
ensure
(a_doors = Void and doors.count = 0) or (a_doors /= Void and then a_doors.count = doors.count)

-- a_doors.count = doors.count -- nur, falls a_doors /= Void in Precondition!

end
```
2 Design by Contract

```
set_color (a_color: COLOR)
require

-- Lösung: keine Precondition

do
  color := a_color
ensure

  color_set: color = a_color

end
```

color: COLOR
-- The color of the car. ‘Void’ if not specified. Default: ‘Void’.

color_exists: color /= Void

-- Lösung: keine Precondition
class CAR
create
    make
feature {NONE} -- Creation
    make
        -- Creates a default car.
        require
            -- nothing
        do
            create \{LINKED_LIST [CAR_DOOR]\} doors.make
        ensure
            not is convertible
            doors \(!=\) Void and then doors.count = 0
            color = Void
        end
end

feature {ANY} -- Access
    is_convertible : BOOLEAN
        -- Is the car a convertible (cabriolet)? Default: no.
    doors: LIST [CAR_DOOR]
        -- The doors of the car. Number of doors must be 0, 2 or 4. Default: 0.
    color: COLOR
        -- The color of the car. 'Void' if not specified. Default: 'Void'.
end

invariant
    doors \(!=\) Void
    doors.count = 0 or doors.count = 2 or doors.count = 4
3 Inheritance and polymorphism

- **PRODUCT**: set_price, price, description
- **COFFEE**: make
- **ESPRESSO**: description+
- **CAPPUCINO**: description+
- **CAKE**: description+
- **PRODUCT** inheritance:
  - **COFFEE**: set_price, make
  - **ESPRESSO**, **CAPPUCINO**

The diagram illustrates the relationships and methods associated with each class.
3 Inheritance and polymorphism

1. \text{create product}

\text{io.put_string(product.description)}
3 Inheritance and polymorphism

2.

```python
create {ESPRESSO} product.set_price(5.20)
io.put_string(product.description)
```

Kompiliert!
Output: „A small strong coffee.“
3 Inheritance and polymorphism

3.

```ruby
create cappucino
make
io.put_string(cappucino.description)
```
3 Inheritance and polymorphism

4.

\texttt{create} \{\texttt{ESPRESSO}\} \texttt{cappucino.set\_price}(5.20)
\texttt{io.put\_string(cappucino.description)}
3 Inheritance and polymorphism

5.

```plaintext
create cake.make(6.50)
product := cake
io.put_string(product.description)
```

Kompiliert!
Output: „A sweet dessert.“
3 Inheritance and polymorphism

6.

```java
create {ESPRESSO} product.set_price(5.20)
espresso := product
io.put_string(espresso.description)
```
Inheritance and polymorphism

7. 

```markdown
create \{CAPPUCINO\} coffee.set_price(5.50)  
coffee.make
```

Kompiliert!  
Output: „I am making you a coffee.“

```markdown
defered class COFFEE
  inherit PRODUCT

  feature -- Main operations
    make
      -- Prepare the coffee.
      do
        print ("I am making you a coffee.")
      end
  end
end
```

```markdown
class CAPPUCINO
  inherit COFFEE

  create
    set_price

  feature -- Access
    description: STRING
      do
        Result := "A coffee with milk and milk foam"
      end
end
```
4 Recursion: Deleting directories

delete_all (directory: FILE)
   require
      directory /= Void and then (directory.exists and directory.is_directory)
   local
      directories: ARRAYED_LIST [FILE]
      cur_directory: FILE
   do
      -- delete all files
      from
         create directories.make
         directories.extend (directory)
         directories.start
      until
         directories.after
      loop
         cur_directory := directories.item
         from
            cur_directory.start
         until
            cur_directory.after
         loop
            if cur_directory.item.is_directory then
               directories.extend (cur_directory.item)
            else -- normal file
               cur_directory.item.delete
            end
         cur_directory.forth
      end
   directories.forth
end
directories

delete_all (directory: FILE)
do
  cur_directory := directories.item
  delete_all (directory: FILE)
  loop
    if cur_directory.item.is_directory then
      directories.extend (cur_directory.item)
    else -- normal file
      cur_directory.item.delete
    end
  end
  cur_directory.forth
  until
end
directories.forth

delete_all (directory: FILE)
require
  directory /= Void and then (directory.exists and directory.is_directory)
local
directories : ARRAYED_LIST [FILE]
cur_directory : FILE
do
  -- delete all files
  cur_directory := directories.item
  from
    create directories.make
    directories.extend (directory)
    directories.start
  until
directories.after
loop
directories

1     2

delete_all (directory: FILE)
define
    require
        directory /= Void and then (directory.exists and directory.is_directory)
    local
        directories: ARRAYED_LIST [FILE]
        cur_directory: FILE
do
    -- delete all files
    from
        create directories.make
        directories.extend (directory)
        directories.start
    until
        directories.after
loop
    cur_directory := directories.item
    from
        cur_directory.start
    until
        cur_directory.after
loop
    if cur_directory.item.is_directory then
        directories.extend (cur_directory.item)
    else
        cur_directory.item.delete
    end
    cur_directory.forth
end
directories.forth
directories

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

delete_all (directory: FILE)

directories

```
delete_all (directory: FILE)
require
    directory /= Void and then (directory.exists and directory.is_directory)
local
directories: ARRAYED_LIST [FILE]
cur_directory: FILE
do
    cur_directory := directories.item
    from
        create directories.make
        directories.extend (directory)
        directories.start
    until
directories.after
loop
    if cur_directory.item.is_directory then
        directories.extend (cur_directory.item)
    else -- normal file
        cur_directory.item.delete
    end
    cur_directory.forth
end
directories.forth
```
```plaintext
directories

1 2 4

delete_all (directory: FILE)
delete_all (directory: FILE)

require
directory /= Void and then (directory.exists and directory.is_directory)

local
directories : ARRAYED_LIST [FILE]
cur_directory : FILE

do
   -- delete all files
from
cur_directory := directories.item
from
directories.extend (directory)
directories.start
until
directories.after
loop
   if cur_directory.item.is_directory then
      directories.extend (cur_directory.item)
   else -- normal file
      cur_directory.item.delete
   end
   cur_directory.forth
end
directories.forth
```

1. C:\Temp\to_del\another_file.txt
directories

```
delete_all (directory: FILE)
require
directory /= Void and then (directory.exists and directory.is_directory)
local
directories: ARRAYED_LIST [FILE]
cur_directory: FILE
do
  -- delete all files
from
create directories.make
  directories.extend (directory)
directories.start
until
  directories.after
loop
cur_directory := directories.item
from
cur_directory.start
until
cur_directory.after
loop
  if cur_directory.item.is_directory then
    directories.extend (cur_directory.item)
  else -- normal file
    cur_directory.item.delete
  end
end
cur_directory.forth
end
directories.forth
```
directories

```
delete_all (directory: FILE)

require
    directory /= Void and then (directory.exists and directory.is_directory)

local
    directories: ARRAYED_LIST [FILE]
    cur_directory: FILE

do
    -- delete all files
    from
        create directories.make
        directories.extend (directory)
        directories.start
    until
        directories.after
    loop

    cur_directory := directories.item
    from
        cur_directory.start
    until
        cur_directory.after
    loop
        if cur_directory.item.is_directory then
            directories.extend (cur_directory.item)
        else
            -- normal file
            cur_directory.item.delete
        end
    end
    cur_directory.forth
end
end
```
**directories**

<table>
<thead>
<tr>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
</table>

```plaintext
delete_all (directory: FILE)

local
directories: ARRAYED_LIST [FILE]
cur_directory: FILE

do
    -- delete all files
from
cur_directory := directories.item
from
directories.extend (directory)
directories.start
until
directories.after

until
directories.after

loop
    if cur_directory.item.is_directory then
        directories.extend (cur_directory.item)
    else -- normal file
        cur_directory.item.delete
    end
    cur_directory.forth
end

end
```
directories

1 2 4 5

1. C:\Temp\to_del
2. C:\Temp\to_del\1
3. C:\Temp\to_del\1foo.txt
4. C:\Temp\to_del\2
5. C:\Temp\to_del\2\3
6. C:\Temp\to_del\2\3foobar.txt
7. C:\Temp\to_del\2bar.txt
8. C:\Temp\to_del\another_file.txt
9. C:\Temp\to_del\file.txt

delete_all (directory: FILE)

delete_all (directory: FILE)

require
directory /= Void and then (directory.exists and directory.is_directory)

local
directories : ARRAYED_LIST [FILE]
cur_directory: FILE

do

-- delete all files
from
create directories.make
directories.extend (directory)
directories.start
until
directories.after

loop

cur_directory := directories.item
from
cur_directory.start
until
cur_directory.after
loop

if cur_directory.item.is_directory then
directories.extend (cur_directory.item)
else -- normal file
cur_directory.item.delete
end

cur_directory.forth
end

directories.forth

directories
directories

1. C:\Temp\to_del
2. C:\Temp\to_del\1
3. C:\Temp\to_del\1\foo.txt
4. C:\Temp\to_del\2
5. C:\Temp\to_del\2\3
6. C:\Temp\to_del\2\3\foobar.txt
7. C:\Temp\to_del\2\bar.txt
8. C:\Temp\to_del\another_file.txt
9. C:\Temp\to_del\file.txt

```
delete_all (directory: FILE)
delete_all (directory: FILE)
require
    directory /= Void and then (directory.exists and directory.is_directory)
local
    directories : ARRAYED_LIST [FILE]
cur_directory: FILE
do
    -- delete all files
from
    create directories.make
    directories.extend (directory)
directories.start
until
    directories.after
loop
    cur_directory := directories.item
from
    cur_directory.start
until
    cur_directory.after
loop
    if cur_directory.item.is_directory then
        directories.extend (cur_directory.item)
    else -- normal file
        cur_directory.item.delete
    end
    cur_directory.forth
end
directories.forth
```
```plaintext
--- delete all directories
from directories . finish
until directories . before
loop directories . item . delete
   directories . back
end
ensure
   not directory . exists
end

1. C:\Temp\to_del
2. C:\Temp\to_del\1
3. C:\Temp\to_del\1\foo.txt
4. C:\Temp\to_del\2
5. C:\Temp\to_del\2\3
6. C:\Temp\to_del\2\3\foobart.txt
7. C:\Temp\to_del\2\bar.txt
8. C:\Temp\to_del\another_file.txt
9. C:\Temp\to_del\file.txt
```
directories

---
define delete all directories
from
directories . finish
until
directories . before
loop
directories . item . delete
directories . back
end
ensure
not directory . exists
end
directories

```plaintext
-- delete all directories from
    directories . finish
until
    directories . before
loop
    directories . item . delete
    directories . back
end
ensure
    not directory . exists
end
```
directories

```
-- delete all directories
from directories.finish
until directories.before
loop directories.item.delete directories.back
end
ensure not directory.exists
end
```