1 **Task 1 - Terminology**

Place a check-mark in the box if the statement is true. There may be multiple true statements per question, or none at all.

1. An object...
   - □ a. only exists at compile time.
   - □ b. may contain references to other objects.
   - □ c. can be deferred.
   - □ d. may define a precondition.

2. Creation procedures...
   - □ a. can only be called on targets that are void.
   - □ b. may define a postcondition, but no precondition.
   - □ c. can not be listed in deferred classes.
   - □ d. are always commands.

3. If class A inherits from class B...
   - □ a. an object of static type A can be used whenever an object of static type B is expected.
   - □ b. an object of static type B can be used whenever an object of static type A is expected.
   - □ c. In class A we can use the redefine clause to redefine a feature of class B.
   - □ d. In class B we can use the rename clause to rename a feature of class A.

4. The postcondition of a feature...
   - □ a. may contain any commands defined in the class of the feature, but no commands of other classes.
   - □ b. must be defined before the *do* keyword, but after the *local* keyword.
   - □ c. must hold at the end of every execution of the feature.
   - □ d. can contain assertion tags to make the code more readable.

5. About loops:
   - □ a. A loop must always define an invariant, otherwise the programme will not compile.
   - □ b. The variant of the loop must increase with every loop iteration.
   - □ c. The variant of the loop must decrease with every loop iteration and must always be > 0.
   - □ d. It is possible that a loop will never terminate.
2 Inheritance and Polymorphism

Below you find a few classes modelling a part of the animal world.

Listing 1: Class ANIMAL

```ruby
deferred class
  ANIMAL

  feature

  move
    deferred
    end

  eat
    deferred
    end

end
```

Listing 2: Class MAMMAL

```ruby
deferred class
  MAMMAL
  inherit
    ANIMAL

end
```

Listing 3: Class REPTILE

```ruby
deferred class
  REPTILE
  inherit
    ANIMAL

end
```
Listing 4: Class *SNAKE*

class *SNAKE*
inherit *REPTILE*

feature

*move*
do
  io.put_string ("Crawling in the jungle \%N")
end

eat
do
  io.put_string ("Eating a mouse \%N")
end
end

Listing 5: Class *DOLPHIN*

class *DOLPHIN*
inherit *MAMMAL*

feature

*move*
do
  io.put_string ("Swimming in the ocean \%N")
end

eat
do
  io.put_string ("Eating some fish \%N")
end
end
Listing 6: Class \textit{TIGER}

class \textit{TIGER}
\begin{verbatim}
  inherit MAMMAL
  rename
    eat as devour
\end{verbatim}

\begin{verbatim}
feature

  move
do
    io.put_string(\texttt{"Running through the desert \%N"})
end

devour
do
    io.put_string(\texttt{"Devouring an innocent sheep \%N"})
end
\end{verbatim}

end

Listing 7: Class \textit{HUMAN}

class \textit{HUMAN}
\begin{verbatim}
  inherit MAMMAL

feature

  move
do
    io.put_string(\texttt{"Taking a taxi \%N"})
end

  work
do
    io.put_string(\texttt{"working hard to earn money \%N"})
end

  eat
do
    io.put_string(\texttt{"eating with knife and fork \%N"})
end
\end{verbatim}

end
Listing 8: Class **STUDENT**

class **STUDENT**
  inherit **HUMAN**
  rename work as chill
  redefine
    move,
    chill,
    eat
end

feature

  move
do
    Precursor
    io.put_string ("or taking the Science City Link %N")
end

  legi_nr: **STRING**

    chill
do
      io.put_string ("Just chillin’ %N")
end

    eat
do
      Precursor
      io.put_string ("at Polymensa! %N")
end

end
2.1 Task 2

For each code fragment below, state whether it compiles or not. If it does not compile, explain why. If it does compile, what will be printed? Assume that every code fragment is executed separately, so they do not interfere with each other.

```
s: STUDENT
h: HUMAN
a: ANIMAL
m: MAMMAL
r: REPTILE
sn: SNAKE
t: TIGER
d: DOLPHIN
...

create d
a := d
d.move
...
create sn
create r
r := sn
r.eat
```

...
create s
  h := s
  io.put_string (h.legi_nr)

create s
  a := s
  a.eat

create t
create sn
  t := sn
  t.devour
\text{create} \ t
\begin{align*}
m & := t \\
m.eat & 
\end{align*}

\text{create} \ \{\text{TIGER}\} \ a
\begin{align*}
a & .devour 
\end{align*}

\text{create} \ s
\begin{align*}
s & .work 
\end{align*}

\text{create} \ \{\text{ANIMAL}\} \ d
\begin{align*}
d & .move 
\end{align*}
create $s$
$h := s$
$h.work$

create $\{SNAKE\} \ r$
$r.move$

create $\{STUDENT\} \ h$
$h.work$
3 Anagrams

Two words or phrases are said to be anagrams of each other if they contain the exact same letters, but in a different order. Blanks do not count as a letter. Have a look at the examples below:

- *ipod lover* = *poor devil*
- *listen* = *silent*
- *funeral* = *real fun*
- *Actor Sylvester Stallone* = *Very cool talentless star*
- *Justin Timberlake* = *Im a jerk but listen*

Your task is to write a feature that, given two strings, decides whether the two strings are anagrams or not. Fill in your code below. You can assume that the arguments are never void. The number of lines does not necessarily represent the number of lines of code you have to write. You can also assume that the strings passed as argument only contain the letters a-z, A-Z or blanks, so they do not contain numbers or other special characters. The feature should not modify the arguments!

Hints:

- The command `tolower` of class STRING transforms a string to all lowercase. e.g. "CaPiTaL" will become "capital"
- The command `prune_all(a_char : CHARACTER)` of class STRING removes all occurrences of `a_char`. e.g. with argument 'a', "Banana" will become "Bnn".
- The command `put(c : CHARACTER; i : INTEGER)` of class STRING replaces the character at position `i` with `c`
- Note that each letter from word 1 may only be used once in word 2 and vice versa. I.e. "abba" and "baba" are not anagrams.

10
is_anagrams(s1, s2: STRING): BOOLEAN

do