Part 1: Key notions introduced in the course

Feature classification (8 pts)

A class is an implementation of an Abstract Data Type (ADT). ADT have “functions”; ADT “functions” yield class “features”.

Give two taxonomies of class features and explain the difference between these different sorts of features. (Use a picture if needed.)

Genericity (7 pts)

4.1 Is $STACK [G]$ a type? Explain. (1 pt)
4.2 How is $G$ called in $STACK [G]$? (1 pt)
4.3 How is $STACK [ACCOUNT]$ called? Is $STACK [ACCOUNT]$ a type? (1 pt)
4.4 How is $ACCOUNT$ called in $STACK [ACCOUNT]$? (1 pt)
4.5 What is constrained genericity? Give an example. (3 pts)

Design by Contract (6 pts)

List the different kinds of assertions, and explain the role of each.

Part 2: Advanced topics

Abstract Data Types (8 pts)

A class is an implementation of an Abstract Data Type (ADT). An ADT specification is a formal, mathematical description rather than a software text. It is defined by Types, Functions, Axioms, and Preconditions. Here is an ADT specification of unbounded stacks:
TYPES
•  \textit{STACK} [G]

FUNCTIONS
•  \textit{put}: \textit{STACK} [G] \times G \rightarrow \textit{STACK} [G]
•  \textit{remove}: \textit{STACK} [G] \rightarrow \textit{STACK} [G]
•  \textit{item}: \textit{STACK} [G] \rightarrow G
•  \textit{empty}: \textit{STACK} [G] \rightarrow \text{BOOLEAN}
•  \textit{new}: \textit{STACK} [G]

AXIOMS
For any \( x, s: \textit{STACK} [G] \)
•  \textit{item} (\textit{put} (s, x)) = x
•  \textit{remove} (\textit{put} (s, x)) = s
•  \textit{empty} (\textit{new})
•  \textbf{not} \textit{empty} (\textit{put} (s, x))

PRECONDITIONS
•  \textit{remove} (s: \textit{STACK} [G]) \textbf{require} \textbf{not} \textit{empty} (s)
•  \textit{item} (s: \textit{STACK} [G]) \textbf{require} \textbf{not} \textit{empty} (s)

Indicate the precise changes needed to describe bounded-size stacks. (3 pts)

---

ETH Zurich – Computer Science

Course: Advanced Topics in Object Technology, by Bertrand Meyer

Summer semester 2003 — Exam

\textbf{No notes, computers, calculators, etc.} Please write the answers on one or more separate sheets of paper and be sure to write your name at the top of each sheet.

\textit{Abstract Data Types}

\textbf{Naming style (4 points)}
The class \textit{STACK} of EiffelBase uses the feature names \textit{put}, \textit{remove}, and \textit{item} to denote the services that are usually called \textit{push} (adding an element at the top of the stack), \textit{pop} (removing the top-most element of the stack) and \textit{top} (returning the top-most element of the stack). Explain why.
Mathematics vs. Software (4 points)
Explain the core difference between an ADT and a class.

Sufficient completeness (2 points)
Define the concept of sufficient completeness for an ADT.

**Objects**

Return vs. Result (3 points)
Eiffel uses the keyword `Result` to denote the result of a function, whereas other O-O languages like C# or Java use a `return` instruction. Explain the differences between these two approaches.

Cloning (5 points)
Define shallow cloning and deep cloning. (2 points)
Suppose you want to clone an instance of `LINKED_LIST`, what kind of cloning facility would you like to use? Explain why. (3 points)

Export status of attributes in Eiffel (12 points)
Let’s consider a class `DEVICE` defined as follows:

```eiffel
class DEVICE
feature -- Access
  temperature: REAL
    -- Temperature captured by the device
end
```

Explain what isn’t valid in: (2 points)

```eiffel
class DEVICE_CLIENT
feature -- Basic operations
  print_temperature is
    local
      a_device: DEVICE
    do
      create a_device
      print (a_device.temperature)
    end
  change_temperature (t: REAL) is
    local
      a_device: DEVICE
    do
      create a_device
      a_device.temperature := t
    end
end
```

3
Why? Explain the export status policy for attributes in Eiffel. (2 points)
The export status of attributes in Eiffel is related to a core O-O principle: which one? (2 points)
Give a valid version of these classes achieving the desired intent. (4 points)
Explain how C# achieves the same result. (2 points)

**Genericity**

**Constrained genericity (5 points)**
What does the following notation mean? (2.5 points)

```plaintext
class C [G -> COMPARABLE]
```
Give an example using constrained genericity. (2.5 points)

**Design by Contract**

**Assertions vs. assert (4 points)**
Explain the differences between the assert instruction of C/C++ and assertions as defined in Design by Contract.

**Hoare triples (6 points)**
Explain what the following notation means. (2 points)

```plaintext
{P} A {Q}
```
What is the weakest possible precondition? The strongest? (2 points)
What is the weakest possible postcondition? The strongest? (2 points)

**Imperative vs. Applicative (4 points)**
Explain the difference(s) between

```plaintext
do
    balance := balance - sum
end
```
and

```plaintext
ensure
    balance = old balance - sum
```