



Examiner: Carsten Butz, IT-University of Copenhagen
Censor: Fritz Henglein, DIKU, Copenhagen University
Exam Form: Written Exam
Date: January 3, 2003
Time: 9:00 – 13:00

WRITTEN EXAM, JANUARY 3, 2003

Please read the following instructions carefully before starting answering the questions.

- The exam is a written exam. You have four (4) hours to answer the questions.
- The exam is graded on the Danish 13 scale.
- The question set is available in English only. You are allowed to answer the questions in either *English* or *Danish*.
- The exam is an open book exam, which for this exam means that the following aids may be used:
 - The text book *Java by Dissection*.
 - Additional course notes, like copies of slides, the notes by Peter Sestoft on searching and sorting, and on software testing.
 - Handwritten notes.
 - A language dictionary.
- No electronic devices are permitted.
- The exam consists of four main questions. The weight of each question is given (in percentage points) next to the question. All four questions should be answered.
- This question set consists of nine (9) pages.
- When using classes or methods from the book or from the notes by Peter Sestoft then it is not necessary to copy them. However, you have to give the precise location of those (for example, source, edition, section number, page number). It is your responsibility that we can identify that location.

(Weight 25%) QUESTION 1

For this question consider the following fragment of code:

```
1 class Class1{
2     int a=0, b=0;
3
4     int getFirst(){return a;}
5     int getSecond(){return b;}
6 }
7
8 class Class2 extends Class1{
9     int c;
10
11     Class2(int c){a = c;this.c = c;}
12     int getThird(){return c;}
13 }
14
15 class Class3 extends Class1{
16     int c = 2;
17
18     Class3(int c){this.c = c;}
19     int getSecond(){
20         int c = 2;
21
22         return c*b;}
23     int getThird(){return c;}
24 }
25
26 class Class4 extends Class3{
27     int d;
28
29     Class4(int d){super(0);this.d = d;}
30     int getFourth(){return d;}
31 }
32
33 class Scope{
34
35     public static void main(String[] args){
36
37         for(int c = 0; c < 5; c++){
38             System.out.println(c);
39
40         }
41
42         static int a = 0, b = 1, c = 2, d = 4;
43     }
```

QUESTION 1.1

Draw the UML class diagram for the above classes.

QUESTION 1.2

State what the scope of each of the following identifiers is:

1. the variable `c` in method `getSecond()` of class `Class3` (line 20).
2. the variable `c` in `main()` (line 37);
3. the static variable `c` (line 42).

QUESTION 1.3

If the statement

```
c = 4;
```

is inserted into each method (constructors and instance methods) below, state, with reasons, whether or not it would be valid. Where you think it is valid, state where the identifier it refers to is declared.

1. `getSecond()`, line 5;
2. `getSecond()`, line 19;
3. `getSecond()`, line 21;
4. `Class4()`, line 29;
5. `main()`, line 36;
6. `main()`, line 39.

QUESTION 1.4

For each of the following `main()` methods, if executed state how many objects are created and of what type the created objects are:

1.

```
public static void main(String[] args){
    Class1 c1, c2;
    c1 = new Class1();
    c2 = c1;
}
```
2.

```
public static void main(String[] args){
    Class2 c1, c2;
    c1 = new Class2(1);
    c2 = new Class2(4);
}
```
3.

```
public static void main(String[] args){
    Class1 c;
    Class3 d;
    d = new Class3(0);
    c = d;
}
```
4.

```
public static void main(String[] args){
    Class4[] c = new Class4[5];
}
```
5.

```
public static void main(String[] args){
    Class4 c1 = new Class4(3);
    Class4 c2 = new Class4(5);
    c1 = c2;
}
```

QUESTION 1.5

How should the code above be modified so that one could check (by running the program) how many different objects of class `Class1` are created?

(Weight 30%) **QUESTION 3**

You are given a class `Tuple` which is a generalization of both an array and a (linked) list. An object of class `Tuple` can hold a variable number of other objects (similar to a linked list), and the individual components can be accessed by giving the component number (similar to arrays, though here we access those 'cells' through methods). Also, it is possible to remove components with a specified index, and to insert a component with a specified index. Note that since we can access components directly through their component number this means that when component `k` is removed, the 'old' component `k+1` now has index `k`. Similarly, if a component is added at position `k`, then the old component with index `k` has now become the component with index `k+1`. A UML class diagram for some of the methods is shown below, and documentation for those methods are attached to this exam paper.

Tuple
+ Tuple()
+ add(int, Object): void
+ addElement(Object): void
+ clear(): void
+ elementAt(int): Object
+ indexOf(Object): int
+ remove(int): Object
+ size(): int

In this question you are supposed to implement a (wrapper) class `DoubleTuple` which can be used to hold just `double` values. Thus, the only attribute of class `DoubleTuple` should be a `Tuple`.

QUESTION 3.1

Write the code for the class definition for class `DoubleTuple`, excluding the code of the constructor and the methods.

QUESTION 3.2

Write the code for the constructor `DoubleTuple()` and the five methods `add(int, double)`, `clear()`, `elementAt(int)`, `indexOf(double)`, `size()`.

- The constructor method `DoubleTuple()` should initialize an empty object of class `DoubleTuple`.
- The method `add(int, double)` adds the second argument at the position specified by the first argument.
- The method `clear()` clears the tuple, i.e., sets the number of components (occupied cells) back to zero.
- The method `elementAt(int)` returns the double number stored in the tuple cell determined by the argument.
- The method `size()` returns the number of components, i.e., the number of occupied cells.

Note that the those methods double on those of class `Tuple`, so that they have short and simple implementations.

QUESTION 3.3

Implement the method

`boolean contains(double)`

which will return `true` if the value of the passed parameter occurs in one of the tuple cells, and `false` otherwise.

In the following questions we will optimize the search algorithm used in method `contains(double)`. When looking at the search algorithm for this method then you can see that cells with low indices are good places to store frequently requested items. Thus, after each request we will modify the order of the items stored in the tuple so that items which are requested often are stored in cells with a low index and are thus found quicker the next time method `contains(double)` is called.

QUESTION 3.4

Change your implementation of method `contains(double)` so that in case you find the value in tuple cell `k` the following happens: Cell `k` is deleted, but a new cell at index 0 is inserted and the value found is stored there. Return `true` or `false` depending on whether the passed parameter is found in one of the tuple cells or not.

[Note: The idea is that we swap two cells, namely those of index 0 and `k`, so that requested elements are moved to the front of the tuple.]

QUESTION 3.5

Moving a requested element to the head of the tuple is often over-reacting. Change your implementation of method `contains(double)` so that in case you find the value in tuple cell `k` then you swap that cell with its neighbouring cell.

Return `true` or `false` depending on whether the passed parameter is found in one of the tuple cells or not.

QUESTION 3.6

Briefly describe in plain language (no implementation) one other way of optimising the method `contains()`. Briefly discuss which of the improvements is best, and when to use them.

QUESTION 3.7

[Note: This question is more difficult than the others, and should only be approached once you have finished working on the other questions in this exam.]

With a particular implementation of the class `Tuple` one can observe that the above two 'improvements' of the method `contains()` through re-arrangement of the data in the tuple ('swapping of cells') lead to a worse performance of the method than without re-arrangement. Give reasons why this might be the case.

(Weight 25%) QUESTION 4

In this question you are supposed to implement a class `Tuple` similar to the one used in the previous question. *To simplify matters we will implement tuples of strings.*

An object of class `Tuple` can hold a variable number of strings (similar to a linked list), and the individual components can be accessed by giving the component number (similar to arrays, though here we access those components through methods). Also, it is possible to remove components with a specified index (renaming the cells with a higher index), and to insert a component with a specified index (again renaming those cells with a higher index).

The UML class diagram for our implementation of class `Tuple` is shown below:

Tuple
-dataTuple: String[] -numocc: int
+Tuple()
+add(int,String): void +addElement(String): void +clear(): void +contains(String): boolean +elementAt(int): String +indexOf(String): int +remove(int): String +size(): int

Here `dataTuple` will be an array of strings, and the integer variable `numocc` (standing for number of occupied cells) will maintain the number of currently occupied cells.

QUESTION 4.1

Implement the following methods:

1. `Tuple()`
The constructor should instantiate the variables so that an array of strings with 10 cells is created, but none of the cells is occupied.
2. `int size()`
The method should return the number of currently occupied cells.
3. `void clear()`
The method should reset `numocc` to zero.
4. `String elementAt(int)`
The method returns the string stored in the named cell.
[Note: Do not worry about a potential array-out-of-bound error.]

QUESTION 4.2

Implement the method `indexOf(String)`, which should return the index of a cell containing the string, and `-1` if the string does not match the content of any occupied cell.

QUESTION 4.3

Implement the method `remove(int)`, which should remove the cell whose integer parameter is passed to the method. [Note: Think about what removing here means. If value `k` was passed as parameter to the method then the new cell `k` is the old cell `k+1`, the new cell `k+1` is the old cell `k+2`, etc.]

QUESTION 4.4

Write the code for the method `add(int,String)`. Note that before you can add a string to the tuple you first have to check whether there is still an empty cell that can hold the string. If not, then you should increase the storage space by creating an array of strings `tmp` twice as large as the original array. Copy the cells of `dataTuple` into the first half of `tmp`, and redirect the pointer `dataTuple` so that it now points to the larger array. Then insert your data.

DOCUMENTATION OF JAVA CLASS `TUPLE`

The `Tuple` class implements a growable array of objects. Like an array, it contains components that can be accessed using an integer index. However, the size of a `Tuple` can grow or shrink as needed to accommodate adding and removing items after the `Tuple` has been created.

CONSTRUCTOR DETAILS

```
public Tuple()
```

Constructs an empty tuple so that its internal data array has size 10.

METHOD DETAILS

```
public void add(int index, Object element)
```

Inserts the specified element at the specified position in this `Tuple`. Shifts the element currently at that position (if any) and any subsequent elements to the right (adds one to their indices).

```
public void addElement(Object obj)
```

Adds the specified component to the end of this tuple, increasing its size by one. The capacity of this tuple is increased if its size becomes greater than its capacity.

```
public void clear()
```

Removes all of the elements from this `Tuple`. The `Tuple` will be empty after this call returns (unless it throws an exception).

```
public Object elementAt(int index)
```

Returns the component at the specified index.

```
public int indexOf(Object elem)
```

Searches for the first occurrence of the given argument, testing for equality using the `equals` method.

```
public Object remove(int index)
```

Removes the element at the specified position in this `Tuple`. Shifts any subsequent elements to the left (subtracts one from their indices). Returns the element that was removed from the `Tuple`.

```
public int size()
```

Returns the number of components in this tuple.