A print-out of the problems set by the schema *Dm03_CodeMaster*

1. Basic method construction

```
Create a 'compute' method in which you ...

define a character-string constant called  Title  with the value "Dame"
and a local var called  friend  of the String type.

Note that constants are given names starting with capitals, whereas
vars are given names starting with lowercase.

The method must ...

1) Accept (typed) into textbox  tbA  a female name like 'Kiri Te Kanawa'.

2) Copy it into   friend.

3) Output the original input into  tbR1.text, and

4) Output into  tbR2.text  the original text with 'Dame' tacked onto the front.
```

---

2. Assignment to simple variables (primitive types)

```
Create a 'compute' method in which you define two String variables (vars)
called  doNow  and  doLater.

In the method . . .

(1) Assign the string  "Mow the lawns"  to  doNow.

(2) Then also assign the variable  doNow  to  the variable doLater .

(3) Display the value of  doNow  in  tbR1.text, and the value of  doLater  in  tbR2.text.

(4) Then call on a delaying method we supply, using the command  waiter(10);

(5) Then have the method alter  doLater  to "Trim edges" and

(6) re-display as in step 3.

    >>> You should find that only  doLater  has been changed.
```

---

3. Assignment to references

```
This DemoCode schema has pre-defined the   Duty   class.
A Duty has only one attribute, a string called taskName

Create a 'compute' method in which you define two vars
which are references called  rDuty & rChore,
both references to objects of this Duty class.

In the method . . .
(1)  [Wrap your whole code between
         beginTransaction ; and
         abortTransaction ;   statements. ]

(2)  _Create_ the object referenced by rDuty and by rChore

(3) Assign the string  "Mow the lawns"  to  rDuty.taskName.

(4) Assign  the variable  rDuty  to  rChore.

(5) Display the value of   rDuty.taskName  in  tbR1.text, and
      display the value of   rChore.taskName in tbR2.text.
```

```
(6) Then call on a delaying method which DemoCode supplies,  using the command
waiter(10);

(7) Then have the method alter  rDuty.taskName  to "Lawns all finished" and

(8) re-display as in step 4.

                  >>> You should find that BOTH have been changed.
```

---

4. Strong type checking

```
Edit the last method.

Try to assign the String "Fred Smith" to the
reference called  rChore. That is to say, assign "Fred Smith"
not to rChore.taskName  but to rChore itself.

Try compiling the method with F8.

The method cannot be compiled while that error remains!

>>> That illustrates Strong Type Checking
    rChore.taskName and 'Fred Smith' are Strings whereas
    rChore is a reference of type Task.
    You cannot store a String in a variable declared to be for a reference.

                                    Comment out the faulty code and
                                    Move on to the next exercise.
```

---

5. Expressions

```
Create a 'compute' method which accepts a typed string into  A and another into B.

 Declare in the vars statement two variables called leng1 and leng2, both of type
Integer.
When the Compute button is pressed . . .

(1) put into the Integer variable  leng1  the length of  tbA.text
(2) put into the Integer variable  leng2  the length of  tbB.text

(3a) compute the sum of  leng1  and  leng2
(3b) (use   .String   to convert the result to a String) and display it in  tbR1.text
(3c) use   waiter(10); for a delay.

(4a) compute a true/false value (true if A is longer than B), and
(4b) set the option button Alpha (obAlpha) to that value
      You can do that in one line with obAlpha.value := (leng1>leng2);
(4c) And use waiter(10); for a delay.

(5a) Join the two strings  A  and B  end to end  in that order  and  display in
tbR2.text,
(5b) Join the three strings B, "  connect  " and A in that order  and  display in
tbR3.text
```

---

6. Methods illustrated with String methods

```
Enter some text - a string - into tbA.

Then compute in turn each of the following:

      (Place the result into tbR1, and place into tbR1
       the name of the demonstration e.g. LENGTH.)

After each, call on   waiter(10);   to make a pause.

Tasks
```

```
a) LENGTH: The length of the string tbA.
b) JOIN:   tbA joined onto string tbB JOINed onto the length
            of the whole.
c) POS:    the position in tbA where the SUBSTRING 'fred' occurs.
d) TRIM:   tbA with any blanks at left or right TRIMMED off.
e) PAD:    tbA PADded out with blanks to the length of 60 chars
f) UPPER:  tbA converted to UPPER case.

Can you devise one string that all six operations apply to?
```

7. Input from four input sources

```
Within  this DemoCode schema we have already . . .
a)  predefined an array of String data called   taskDescr.  The individual
      items are called  taskDescr[1],   taskDescr[2]   and so on.
     Then we have assigned String values to all the items in the array,
     values like "Input from different input sources".
    This is one source of data.

b)  Predefined a constant called   Buster,   which also holds a string value .
      This is another source of data.

c)  In addition, the TextBox input controls,  tbA and tbB, and the
     OptionButton controls obAlpha and obBeta constitute sources of data.

d)  And the Form has a predefined Collection of Task class objects.
      This is yet another source of data.

The compute method's task is to use these sources of data . . .

(1) To determine the length of the string data in  taskDescr[1]
      and output that length into  tbR1.text,  then

(2) To fetch a value by reading it from  tbB.text   and output it into   tbR2.text,

(3)  and to output into  tbR3.text the value of the string constant  Buster
       plus the number of items in the form's  allTasks  collection.

      (The expression  allTasks.size is an integer so adding .String  will return
       a String version of the size of the allTasks collection,
        which you can  append onto   Buster  with  & ).
```

7. More input from several sources

```
Amend the previous method.
a)  the built-in   read  method will ask the user for an input value.
      with a line like  . . .
          read result ;    and
                 where the variable  result   may be of any primitive type,
    and
b)  the built-in   random method of the  app (Application class)  will
      generate a  random number between 0 and 32768 when called
      with a line like  . . .
          tbR2.text  :=  app.random(maxNo).String ;.

      for example
          tbR2.text  :=  app.random(1500).String ;.

      will generate a random number between 0 and 1500.
Amend your method to do this.
```

8: The simple IF—ENDIF instruction

```
Type a string into A. Type another into B.
```

```
>>> The compute method's task is . . .
If the string you entered into A is longer than the
     string you entered into B
then put 'A longer than B' into tbR1.

Be sure to test both possibilities
```

9: The IF--ELSE instruction

```
Builds on previous: Type a string into A. Type another into B.

>>> The compute method's task is . . .
If the string you entered into A is longer than
   the string entered into B
then put "A longer than B"     into tbR1
else put "A not longer than B" into tbR1.
```

10: The IF--ELSEIF . . .ELSEIF instruction

```
Builds on previous: Type a string into A. Type another into B.

>>> The compute method's task is . . .
If      the string you entered into A is longer than
        the string entered into B
        then put 'A longer than B' into tbR1
elseif  A is equal in length to B
        then put 'EQUAL' into tbR1
else    put 'A shorter than B' into R1'
```

11: The WHILE DO instruction

```
Declare an integer variable called counter.
 Set counter to 1.  Start a

while counter<20 do
   ...
endwhile;

 loop.   Inside the loop place three instructions:

 1) to advance counter by 1,
 2) to display in tbR1 "Counter is now " plus the value of Counter (convert to String),
 3) to call on waiter(8);

                    Important: Read the Hint BEFORE you write your method!
```

13: The REPEAT—UNTIL instruction

```
There is not a REPEAT . . .UNTIL instruction in JADE.
However we may attain the same effect with a WHILE  instruction.

Edit the previous method so that the sense obeys the pseudo-code . . .

repeat . . .
     advance counter by 1,
     display in tbR1 "Counter is now " plus the value of Counter (convert to String),
     waiter(8);
until counter > 20;

                    Read the Hint BEFORE you write your method!
```

14: The FOREACH--ENDFOREACH instruction

```
In this Demo we have pre-defined a collection of Task class objects.
          (Each such task object has a String attribute called taskName.)

The Collection of Tasks is called allTasks.

The goal of the method is to do something 'foreach' task in the collection
         (namely, display the taskName.)

You will need to declare a local variable (say tk ) which is a reference to a Task.

The main loop is ...

foreach tk in allTasks do
     . . .
endforeach;

Inside that loop

(a) display tk.taskName in tbR2.text and
(b) call on waiter(8) (to slow the flow down).
```

15: Simulating a FOR instruction

```
Write a method which uses a while-do loop to
simulate the following instruction sequence, which is not itself
implemented in JADE prior to JADE 5:-

Most languages have a construct like ...

         for j = 1 to 100 step 2
               write j.String;
         endfor;
```

16: The FOREACH FOR instruction (actual in JADE 5)

```
If you have JADE 5 or higher write a method
to implement the sense of  ...

         for j = 1 to 100 step 2
               write j.String;
         endfor;

using the foreach j in s to e step do ...
```

17: The BREAK instruction (to jump out of a while loop)

```
Builds on previous problem using foreach to step through
a Collection: we want the method to break out of the foreach loop
as soon as it finds a task whose taskName="Feed cat again"

Use the labelled version of the break.
```

18: Illustrations of Recursion

```
Strings:  Enter a string into tbA.  (We suggest not longer than
12 characters.)  In the  compute  method
```

```
--- put tbA.text into str
--- call revList(str).

as a separate new method define revList , a method which returns a String.
If the incoming parameter is only one character long, return it.

Otherwise separate f, the first character, and the rest strRest.

Return the string    revList(strRest) & f;

Additional work: If you want to make use of waiter(10) inside revList
You could show the value of f and strRest at each stage.
```

18: GUARD expressions, to convert data types

```
Our form defines a variable called   today   of the Date class.

(It has automatically been initialised to the current date.)

Your method must

1) Take a string which you will type into   tbA.text
    in the form of a date (e.g. 13/10/1997)

2)  Convert it to a Date  (to do this append the .Date guard expression),

3)  Compare the two dates (with the > operator) and then

4)  Output into   R3.text  whichever date proves to be the later.
     (Use .String to convert the later Date into a String
      to display in tbR3.)

Any invalid dates entered will be treated by JADE as   null.
```

20: More GUARD expressions, to convert types

```
We build on the previous method.

In addition your method must. . .


Place into tbR2.text a message of the following form:
    "It is TRUE that DATE1 is later than DATE2."

. . . where DATE1 is the value of   today   (converted with .String),
            DATE2 is the value you typed in,  and
            TRUE   is (today > tbA.text.Date)
                      converted from a true/false value to a string, thus . . .
            (today > tbA.text.Date).String;

Prior to JADE 5 any invalid dates entered will be treated by JADE as   null.
With JADE 5 & later use .isInValid to test if the date was invalid.
```

.