Eiffel Cheat-Sheet

September 29, 2015

1 New project

- File – New Project... – Basic application – Create
- Enter location and names (default is OK)
- Root feature of the root class is a routine that is called when the program starts its execution.

2 Hello World

```
class HELLO
create make
feature
  make
    do
      print ("Hello World!%N)
    end
end
```

3 Compiling and Running

- Compile (F7)
- Run (F5)

4 Turning Off Void Safety

In case you get errors regarding unset variables (VEVI):
Project – Project Settings... – Target – Void safety – No

5 Assignment

- `:=` is used for assignment, e.g.:

  ```
  x := y + 1
  ```

- You can only assign to local variables or attributes in `Current`.
- Numbers and characters are stored by value, class types (including strings) are stored by reference.
- Null reference: `Void`.
- Default value: 0 for a value type, `Void` for a reference type.

6 Arithmetics

- Types: `INTEGER_32`, `INTEGER_64`, `DOUBLE`.
- Constants: `1000`, `-100`, `1.E3`, `3.2`.
- Operations: `+`, `-`, `*`, `/` (exact division), `//` (integer division), `\\` (modulo).
- Comparisons: `<`, `>`, `<=`, `>=`, `=`, `/=` (not equal).

7 Comparisons of Objects

For reference types:

- Shallow comparison: `=`, `/=`: checks whether the references point to the same object (memory location).
- Deep comparison: `~`, `/~`: checks whether objects have the same values. Internally calls `{ANY}.is_equal`.

8 Logic

- Type: `BOOLEAN`
- Constants: `True`, `False`.
- Operations: `and`, `or`, `not`.
- Short-circuit versions: `and then`, `or else`. For example:

  ```
  if x /= Void and then x.f = 10 then
    <instruction>
  end
  ```

is equivalent to:

```
if x /= Void then
  if x.f = 10 then
    <instruction>
  end
end
```

9 Characters and Strings

- Types: `CHARACTER`, `STRING`
- Constants: `'A'`, `'a'`, `'%N'` (newline), `'%%'` (percent), `"This is a string.%N"`
- String concatenation: `+`.
- Conversion of an object to a string: `out`. For example:

```
x := 10
print ("x is equal to " + x.out)
```

10 Attributes and Features

A class consists of a set of features: attributes (variables and constants) and routines (functions and procedures):

```
class HELLO
feature
  x: STRING      -- String attribute, initialized to Void.
  y, z: INTEGER  -- Integer attribute, initialized to 0.

  proc1
      -- Procedure, no arguments, no return value.
    do
      <instructions>
    end

  proc2 (a: INTEGER; b: DOUBLE)
      -- Procedure with arguments, no return value.
    do
      <instructions>
    end

  func1: STRING
      -- Function, no arguments.
    do
```

```
    <instructions>
  end

  func2 (a: CHARACTER; b: ARRAY[STRING]): STRING
      -- Function with arguments and local variables
    local
      c, d: STRING
    do
      <instructions>
      Result := b[1]
      <instructions>
    end
end
```

- Functions should not change the object state (command-query separation principle).
- Arguments to procedures and functions cannot be re-assigned (like final in Java).
- A function returns its value via `Result`. The last assignment to `Result` counts.
- Local variables have to be declared in advance.

11 If and else

```
if <boolean expression> then
  <instructions>
elseif <boolean expression> then
  <instructions>
else
  <instructions>
end
```

- There can be more (or zero) `elseif` branches.
- `else` branch is optional.
- At most one branch will be executed.

For example:

```
if x >= 0 then
  Result := x / y
else
  Result := -x / y
end
```

12 Loops

```
from <instructions>
until <boolean expression>
loop
  <instructions>  -- This is the body.
end
```

- `from` clause is executed exactly once in the beginning.
- The boolean expression is tested: if it is true, the loop body is skipped. Otherwise, the loop body is executed and the boolean expression is tested again. And so on.

For example:

```
from i := 0
until i = n
loop
  print (T[i])
  i := i+1
end
```

13 Feature calls

- Without arguments: `proc` (unqualified call), `x.proc` (qualified call on target `x`).
- With arguments: `proc(y, 'a')` (unqualified call), `x.proc(20, z)` (qualified call on target `x`).

14 Console IO

- Reading a whole line from the input:

  ```
  io.read_line
  print (io.last_string)
  ```

- Reading a single word from the input:

  ```
  io.read_word
  print (io.last_string)
  ```

- To read a basic type (e.g. an integer) from command line, use `io.read_word` followed by `io.last_string.to_integer`.

- Note: Each input operation reads the whole line. `read_word` also reads a whole line, but does not consume all of the input.
- Note: The `io.last_string` attribute is reused internally. If you need to store the string somewhere, always use `io.last_string.twin`.
- To output any object you can use a generic function `print`.
- Do not use the operations `io.read_integer`, `io.read_boolean`, etc.

15 Arrays

- Types: `ARRAYED_LIST [INTEGER]`, `ARRAYED_LIST [ARRAYED_LIST [DOUBLE]]`.
- It is a reference type, so it is `Void` initially. To initialize:

```
ints: ARRAYED_LIST [INTEGER]
...
create ints.make_filled (20)  -- Array with 20 elements indexed from 1 to 20.
```

- The elements are initialized to the default value (0 or `Void`).
- Assignment and access: `ints[i] := ints[i]+1`.
- Query `count`: Number of entries and maximum valid index.
- Query `capacity`: Allocated space (automatically increased when full).
- Note: On void safe systems you cannot use `make_filled` for attached types. Instead you can create an empty (i.e. `count=0`) array with creation procedure `make` and then fill it with the feature `extend`.