Programming Languages: Introduction to Smalltalk

Jan Daciuk

Department of Intelligent Interactive Systems, ETI Faculty, GUT

January 4, 2022

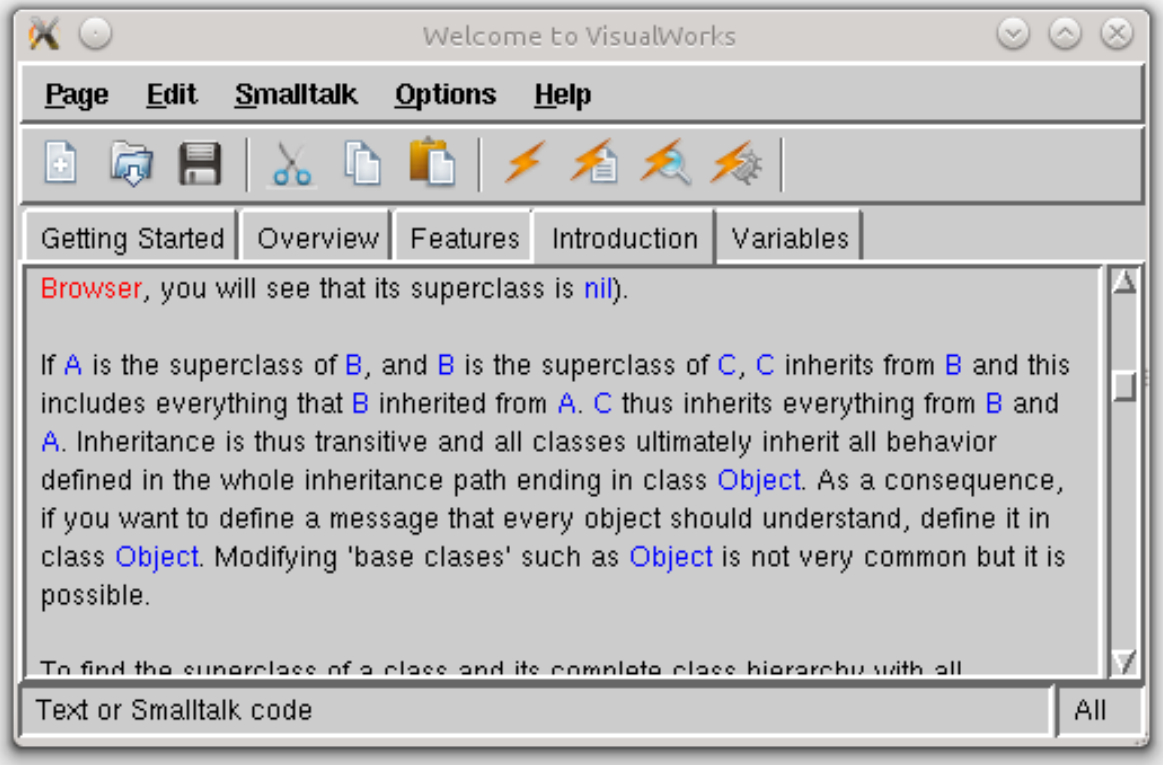
Manuals, Tutorials

Contrary to Ada, Smalltalk has many free manuals and tutorials. Some books, including up-to-date documentation from Cincom, do not record the newest changes, especially to the interface, but most material is up-to-date.

- Manuals: `http://stephane.ducasse.free.fr/FreeBooks/`
- Videos: `http://www.cincomsmalltalk.com/main/products/visualworks/visualworks-tutorials/`
- Documentation copied to disk during installation: subdirectory `doc` of the Smalltalk installation directory. One should start from `WalkTrough.pdf`.

Manuals, Tutorials

The most convenient way is to start with the tutorial included in the system. When the system is invoked, an additional window with the title ”Welcome to Visual Works” appears. One should click ”Introduction” tab.



Installation and Synopsis

- The software can be downloaded from `http://www.cincomsmalltalk.com/main/products/visualworks/`. It is available for free after registration. An ISO image contains software for Windows, Mac, and Linux.
- Linux can invoke the installation program rigth from the ISO image. If it is not possible on Windows, one should burn a CD and launch the installer from the CD.
- The installation program is trivial; one should choose the typical configuration. Additional software can be installed if needed. On Linux, one should set up some variables as shown in a message.
- On Windows an appropriate icon should appear, On Linux one should give the path to the virtual machine (depends on the architecture). E.g. for 64 bit: `bin/linuxx86_64/visual image/visualnc64.im`

Example Program

On eNauczanie platform, the example program is located in `kwadrat.st` file. One should download it and put into the working directory. In the upper window of VisualWorks, one should choose the system browser (`Browse/System` or press `F5` key, or) icon, then click `Package/New Package...` and enter the name of the new package: `Programming Languages`. In the `Comment` tab below, one should enter: `Programs for the Programming Languages course`. In the pop-up menu available by pressing the right mouse button, one should choose `Accept`. The example program `kwadrat.st` can be read in the system browser window in menu `Package/File into...`. Another possibility is to copy the file to `Workspace` (`Tools/Workspace` or), by highlighting it and selecting `Smalltalk/File it in`, but the file contents will then be in a package called `(none)`.

Example Program

The start of the file:

```
Object subclass: #Wielokat
        instanceVariableNames: 'wierzcholki nazwa '
        classVariableNames: ''
        poolDictionaries: ''
        category: 'JezykiProgramowania'!
```

It defines `Wielokat` (polygon) class as a subclass of `Object` class. Objects in `Wielokat` class have instance variables `wierzcholki` (vertices) and `nazwa` (name). The class belongs to `Jezyki Programowania` category. The comment is missing — it should be entered in double quotes.

Example Program

```
!Wielokat methodsFor: 'initialize-release'!

initialize: liczbaWierzcholkow name: nowaNazwa
        "konstruktor obiektu - wielokata"

        nazwa:=nowaNazwa.
        wierzcholki:=Array new: liczbaWierzcholkow.
        wierzcholki at: 1 put: 0@0.! !
```

It defines `initialize: name:` method. It assigns a value `nowaNazwa` (new name) to the variable `nazwa`, it creates `wierzcholki` (vertices) table for all vertices, and it fixes the coordinates of the first vertex as (0,0). The notation `x@y` denotes a point (an object of class `Point` with coordinates (x,y).

Example Program

```
!Wielokat methodsFor: 'accessing'!

nazwa
        "podaje nazwe wielokata"

        ^nazwa!

nazwa: nowa_nazwa
        "ustawia nowa nazwe wielokata"

        nazwa:=nowa_nazwa! !
```

It defines a message `nazwa` (name) that returns the name of the polygon, and a message `nazwa:` (with ":" at the end — this is a keyword message) setting the variable `nazwa`. The character ˆ denotes returning the given value. The characters `:=` denote assigning a value to a variable.

Example Program

```
Wielokat subclass: #Kwadrat
        instanceVariableNames: ''
        classVariableNames: ''
        poolDictionaries: ''
        category: 'JezykiProgramowania'!
```

It defines `Kwadrat` (square) class as a subclass of the polygon class. The new class belongs to the same category. Missing comment again.

Example Program

```
!Kwadrat methodsFor: 'arithmetic'!

+ figura
        "dodaj 2 figury w sensie pola"

        | p |

        p:=self pole + figura pole.
        ^(Kwadrat new) initialize: p sqrt! !
```

It defines the result of adding a figure `figura` to the square. The result is a square with the area equal to the sum of the area (`pole`) of the square that is the recipient of the message and the area of the other figure. A local variable `p` is declared inside vertical bars. Pseudo-variable `self` denotes the same object, and `new` message order the class to produce a new object.

Example Program

```
!Kwadrat methodsFor: 'actions'!

pole
        "licz pole kwadratu"

        ^(wierzcholki at: 2) x squared! !
```

It defines `pole` (area) method that returns the area of the square by squaring (`squared`) the `x` coordinate of (obtained by sending `x` message to) the second vertex.

Example Program

```
!Kwadrat methodsFor: 'initialize-release'!

initialize: bok
        "tworz kwadrat o podanym boku"

        super initialize: 4 name: 'Kwadrat'.
        wierzcholki at: 2 put: bok@0.
        wierzcholki at: 3 put: bok@bok.
        wierzcholki at: 4 put: 0@bok.! !
```

It creates a square with the given side (`bok`). In the first line, it sends `initialize: name:` message to the superior class (`super`), i.e. to the polygon class with a demand to create an object with 4 sides and the name `Kwadrat` (square). Then it sets up the coordinates of the other vertices accordingly (the first one was set to (0,0) in `Wielokat` class).