Reading

VW Smalltalk Application Developer's Guide
Chapter 12 Working with Graphics & Colors
Don't worry about pixmaps & masks

Chapter 17 Process Control

VW Smalltalk GUI Developer's Guide
Chapter 1 Building an Application's GUI (basics about UIPainter)

Chapter 5 Custom Views (for drawing)

Chapter 9 Configuring Widgets (just for the widgets you use)

Chapter 4 Adapting Domain Models to Widgets (not clear if you need this)
Processes (Threads)

fork

Transcript clear.
Transcript show: 'start'; cr.
[ 10 timesRepeat:[Transcript show: 'In loop';cr]] fork.
Transcript show: 'after'; cr
Process Priorities

forkAt: 1 - 100

Transcript clear.
Transcript show: 'start'; cr.
[10 timesRepeat:[Transcript show: 'In loop'; cr]] forkAt: 100.
Transcript show: 'after'; cr
## Priorities

<table>
<thead>
<tr>
<th>Value</th>
<th>Priority Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>timingPriority</td>
<td>real time</td>
</tr>
<tr>
<td>98</td>
<td>highIOPriority</td>
<td>Critical I/O processes, such as network input handling</td>
</tr>
<tr>
<td>95</td>
<td>lowSpacePriority</td>
<td>Priority at which the low space action process runs.</td>
</tr>
<tr>
<td>90</td>
<td>lowIOPriority</td>
<td>Normal input/output activity, such as keyboard input</td>
</tr>
<tr>
<td>70</td>
<td>userInterruptPriority</td>
<td>High-priority user interaction; such a process pre-empts window management, so it should be of limited duration</td>
</tr>
<tr>
<td>50</td>
<td>userSchedulingPriority</td>
<td>Normal user interaction</td>
</tr>
<tr>
<td>30</td>
<td>userBackgroundPriority</td>
<td>Background user processes</td>
</tr>
<tr>
<td>10</td>
<td>systemBackgroundPriority</td>
<td>Background system processes</td>
</tr>
<tr>
<td>1</td>
<td>systemRockBottomPriority</td>
<td>The lowest possible priority</td>
</tr>
</tbody>
</table>
Other Ways of creating Processes

newProcess

Transcript clear.
Transcript show: 'start'; cr.
loop := [ 10 timesRepeat:[Transcript show: 'In loop';cr]] newProcess.
loop proceed.
Transcript show: 'after'; cr
Process Methods

pause
proceed
resume
suspend
terminate
yield
Delay

delay := Delay forSeconds: 1.
delay wait

(Delay forMilliseconds: 100) wait

(Delay forMicroseconds: 100) wait
queue := SharedQueue new.
producer :=
    [1 to: 10
do:
        [:each |
        Transcript show: 'Produced ', each printString; cr.
        queue nextPut: each.
        (Delay forMilliseconds: 500) wait]]
fork.

consumer :=
    [10 timesRepeat:]
        [product := queue next.
        Transcript show: 'Consumed ', product printString; cr.
        (Delay forMilliseconds:1000) wait]] fork
SharedQueue

queue := SharedQueue new.
producer :=
    [1 to: 10
do:
        [:each |
            Transcript show: 'Produced ', each printString; cr.
            queue nextPut: each.
        ]
    ] fork.

consumer :=
    [10 timesRepeat:
        [product := queue next.
            Transcript show: 'Consumed ', product printString; cr.
            (Delay forMilliseconds: 1000) wait]]
fork.
GUI, Processes & Event Queues

GUI has own process (thread)

Handles user events

When user event (button click) triggers your code

Your code is run on GUI thread

While your code runs GUI can not respond to user
Your Code & GUI Process

If your code takes time
   Use separate thread

Your code does something

Usually need to update view

Don't generate GUI events directly from your threads
   Meaning don't draw from your thread

So what to do?
Your Code & GUI Process

If your code takes time
   Use separate thread

Your code does something

Usually need to update view

Don't generate GUI events directly from your threads
   Meaning don't draw from your thread

So what to do?
uiEventFor:

Used to add event to GUI event queue from your thread

move

[300 timesRepeat:
    [
        antColony move.
        [view invalidate]
        uiEventFor: self mainWindow.
        (Delay forMilliseconds: 500) wait]
fork