# Module 3: Introduction to Application Development

This module introduces the System Browser, illustrating the way in which it may be used to examine and modify existing code within the image, and create new methods and classes. This will be illustrated by a number of worked examples, together with some exercises.

The File Browser will also be introduced at this point, in order to allow existing applications to be “filed–in” to VisualWorks.

This module additionally illustrates some of the other types of Browsers that can be “spawned” to view the image in different ways and also explores some of the functions available from the pop–up menus.

Additionally, we conclude the description of the options available from the Launcher that are not covered elsewhere.

## 3.1. The System Browser

The System Browser is the primary user interface for entering code in VisualWorks. It allows the user to:

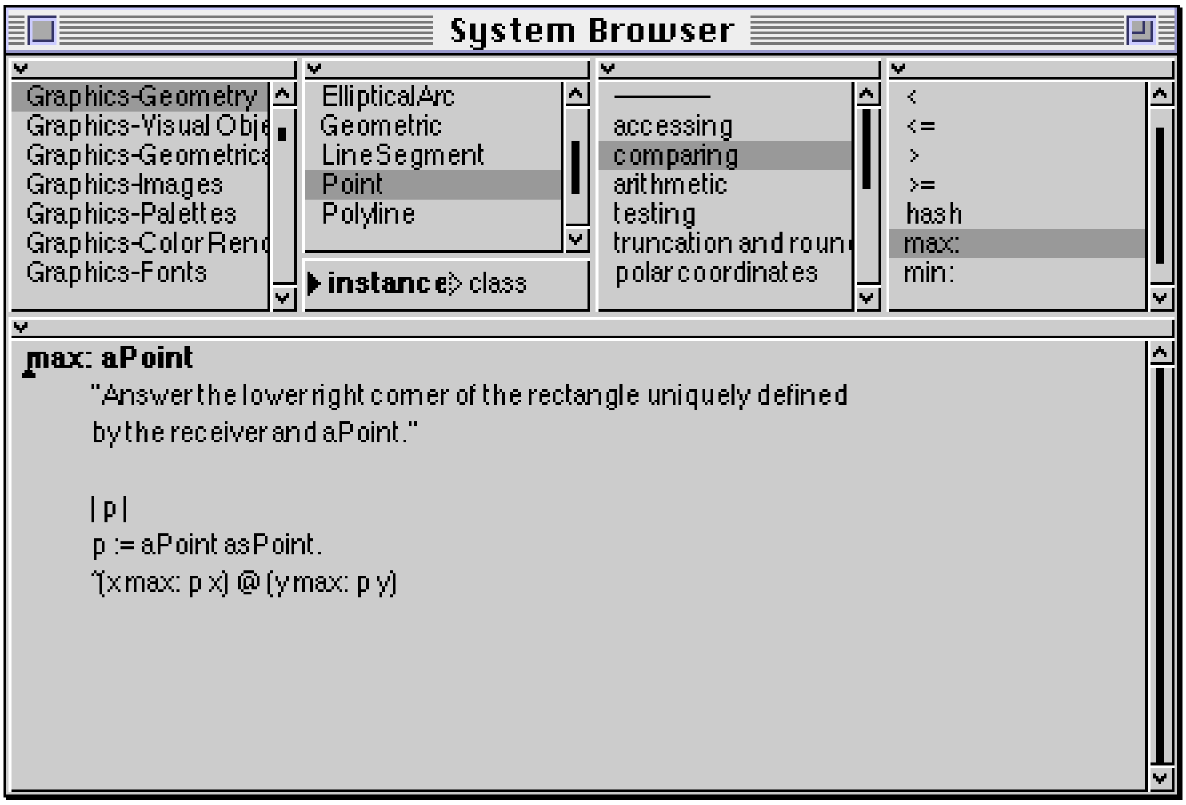
- create and edit source code;
- perform in–line testing;
- format (pretty print) method source code;
- explain code;
- save (“file out”) source code;
- organise classes and methods;
- find the senders of, implementors of, and messages sent by a method;
- create⁄alter⁄remove classes;
- display class hierarchy;
- spawn special purpose Browsers,

many of these will be described in later modules.

The System Browser allows the user to inspect the message interface to any object in the image, and to modify it if required. To open a System Browser, select the **All Classes** item from the **Browse** menu on the Launcher[1]. More than one System Browser can be open simultaneously.

The System Browser (figure 3.1) is made up of five panes and two buttons, marked **instance** and **class**. From left to right, the top four panes are:

[1]Note that the menu item is preceded by an icon, a copy of which is present as one of the buttons below the menu bar.



```
max: aPoint
	"Answer the lower right corner of the rectangle uniquely defined
	by the receiver and aPoint."

	| p |
	p := aPoint asPoint.
	^(x max: p x) @ (y max: p y)
```

**Figure 3.1: The System Browser**

### 3.1.1. Class Categories

These are groups of classes that are categorised for the convenience of the user. The order of the categories and the classes they contain is arbitrary and bears little relationship to the VisualWorks class hierarchy. One of these categories may be selected (by clicking the <select> mouse button). New categories may be added using the **add...** item from the <operate> button menu. Categories may also be *removed* or *renamed* (see later). The classes in the selected category are presented in the next pane:

### 3.1.2. Class Names

Classes in the selected category are presented in this pane. One class may be selected, causing categories of messages to be presented in the next pane. Selecting a class causes the *class definition* to be displayed in the lower (text) pane. Alternatively, a display of the part of the *class hierarchy* containing this class, or a *comment* about the functions of this class can be selected from the <operate> button menu. Classes may also be *removed* or *renamed.*

### 3.1.3. Message Categories

These are the categories of messages which can be sent either to instances of the selected class (**instance** button pressed) or to the class itself (**class** button pressed). By default, the **instance** button is selected. The message categories are also known as *protocols*. One of these protocols may be selected, causing all the message selectors in this protocol to be presented in the right–most pane. The <operate> button menu includes options to allow the user to *add* a new protocol, or *remove* or *rename* an existing protocol.

### 3.1.4. Message Selectors

All the message selectors in the selected protocol are presented in this pane. One of the selectors may be selected, causing its method (the code evaluated when this message is received) to be shown in the lower (text) pane. The code can be modified and re–inserted into the image if desired (see later). The source code for the method displayed in the text pane is held on an external file (the "sources" file — see module 2).

## 3.2. Example: Browsing the Implementation of an Existing Method

Try the following: Select the class category Magnitude–Numbers in the left–most top pane, using the <select> mouse button. Select Number in the Class Names pane. Select testing in the Protocol pane. Select the selector even in the Message Selector pane.

The code displayed in the lower pane is evaluated when an instance of a Number (or one of its subclasses) receives a message with the selector even. This code returns true if the receiver is even, otherwise false. Note that it sends a message to itself (self), using the selector \\ (modulo) with 2 as the argument.

Ex 3.1: Verify that the even method evaluates correctly by typing, selecting and evaluating (using print it) the following expression in a Workspace:

```
42 even
```

Repeat the above test with other numbers.

Ex 3.2: Try the effect of the following options from the <operate> button menu in the Class Names pane. First select Magnitude–Numbers (left–most pane). Select Number in the Class Names pane. Select **hierarchy** from the Class Names pane <operate> button menu. This will display a textual representation of the part of the hierarchical structure of the classes in the image which includes Number in the lower (text) pane.

You can see, for example, that classes Float, Fraction and Integer are all subclasses of Number. Thus, the message even should be understood by instances of all of these classes. Try:

```
4 even
17.91 even
(3/7) even
```

Also examine the **definition** and **comment** menu items from the <operate> button menu.

## 3.3. Defining Methods

A new method can be defined from the Browser whenever a protocol has been selected. If there are no protocols to select, one must be created by using the **add...** command in the Message Categories pane <operate> menu. The following template is provided by the Browser:

```
message selector and argument names
    "comment stating purpose of message"

    | temporary variable names |
    statements
```

This is relatively straightforward to fill in. Any number of statements can be placed in the statements section. There can be no more than 255 temporary variables and arguments.

## 3.4. Example: Creating a new Method

A function not currently implemented by instances of class Number is the “absolute difference” function. Here, we will add this functionality to Number. Select Magnitude–Numbers, Number and arithmetic in the left–most three panes in the System Browser. Do not select anything from the top right–most pane. The lower (text) pane should display a “template” for new methods. Edit (using the normal text editing conventions) the template, so that it appears as below.

```
diff: aNumber
    "return the absolute difference between me and aNumber"
    | temp |
    temp := self - aNumber.
    temp < 0
        ifTrue: [^temp negated]
        ifFalse: [^temp]
```

This method first calculates temp, which is the difference between the parameter aNumber and the receiver (self). It then answers with either temp or a negated version of temp, depending whether temp is negative. The new method can now be compiled and added to the VisualWorks environment by selecting **accept** from the <operate> button menu in the lower pane. Do this. Correct any errors that have inadvertently crept in!

(The **cancel** option discards all the text just added to the text pane and restores it to its earlier state.)

Ex 3.3: Test the functioning of the new method by typing and evaluating (**print it**) suitable expressions in a Workspace. For example, try:

```
42 diff: 17.
17 diff: 42.
-17 diff: -19.
10.15 diff: (3/7).
237846527938465 diff: 3456
```

Note that the addition you have made to your VisualWorks image will be there permanently, until you choose to remove it. (This assumes that you save the image eventually, of course!).

Ex 3.4 In fact, the implementation of diff: used here is not very good (although it works). A better version is shown below.

```
diff: aNumber
    "return the absolute difference between me and aNumber"
    ^ (self - aNumber) abs
```

This version eliminates the temporary variable, and uses the abs method, which is already implemented for Number. This minimises the amount of code added to the image to support this new functionality.

Ex 3.5: Modify your implementation of diff: by editing and accepting the text in the System Browser. Verify that this has not changed the functionality of the method by repeating the tests above. Note that there is no way in which the implementation can be determined by the sender of the diff: message. You may like to look at the implementation of the abs method in class ArithmeticValue.

## 3.5. Defining Classes

In much the same fashion as methods, class definitions can be added using the Browser when a class category is selected. As with methods, if no appropriate category exists then the **add...** option from the Class Categories pane <operate> menu can be used to create one. The Browser provides the following template:

```
NameOfSuperclass subclass: #NameOfClass
    instanceVariableNames: 'instVarName1 instVarName2'
    classVariableNames: 'ClassVarName1 ClassVarName2'
    poolDictionaries: ''
    category: 'Category–Name'
```

Once again, this is easy to fill out. Remember to keep the ‘#’ symbol in front of the class name, and also that class names should always begin with an uppercase character.

Example:

```
Number subclass: #Fraction
    instanceVariableNames: 'numerator denominator'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Magnitude–Numbers'
```

## 3.6. Example: Adding a new “Application”

This example is an exercise in adding a (small) new “application”, based on classes already available within the image. The example itself is adapted from the “Orange Book”, chapter 17.

Here, we will construct a class corresponding to an individual’s “spending history”. We will not be too concerned about the design, or the other classes used. This is an exercise in effectively using the System Browser and compiler. We will describe a new class SpendingHistory with several methods and instance variables. We will also try out this class in a simple manner.

Ex 3.6: Use the **add...** item from the <operate> button menu in the Class Categories pane (top left) of the System Browser. When prompted, choose a suitable name for the category, such as ‘Spending’. This new class category will have no classes in it, as yet. A template for the creation of new classes will be displayed in the lower pane of the System Browser.

Ex 3.7: Edit the template (using the normal text editor conventions) so that it appears as below.

```
Object subclass: #SpendingHistory
    instanceVariableNames: 'cashOnHand expenditures'
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Spending'
```

This declares the new class SpendingHistory to be a subclass of Object. The new class has two instance variables called cashOnHand and expenditures, and no class variables. Select **accept** from the <operate> button menu. This creates the new class SpendingHistory and permanently installs it in the image.

We now need to add some additional functionality to SpendingHistory, since at the moment all instances of SpendingHistory will have exactly the same functionality as instances of Object. First, we will add a method to the class protocol to create new initialised instances of SpendingHistory.

Ex 3.8: Select the **class** button in the System Browser and select the **add...** item from the <operate> button menu in the Message Categories pane. You will be prompted for the name of a protocol; respond with ‘instance creation’. The method template will appear in the lower window. Use the editing functions to create the initialBalance: method as shown.

```
initialBalance: anAmount
    ^super new setInitialBalance: anAmount
```

Note that this method uses a method defined further up the class hierarchy (new). Add the new method to the image using the **accept** item from the <operate> button menu. This method uses a message selector (setInitialBalance:) which is not yet defined, so you should select **proceed** when prompted with a Confirmer.

We will now add instance protocol to class SpendingHistory.

Ex 3.9: Select the **instance** button in the System Browser. Create a new protocol (as before) called ‘private’. Edit the template to add the setInitialBalance: method. Accept this using the <operate> button menu item.

```
setInitialBalance: anAmount
    "Initialize the instance variables;
    cashOnHand is set to amount"

    cashOnHand := anAmount.
    expenditures := Dictionary new.
```

Ex 3.10: Repeat the above for method totalSpentOn: in protocol ‘inquiries’, for method spend:on: in protocol ‘transactions’, and for method printOn: in protocol ‘printing’ (see below).

```
totalSpentOn: reason
    "return the amount spent on reason. Answer
    0 if reason is not used for expenditures"

    (expenditures includesKey: reason)
        ifTrue: [^expenditures at: reason]
        ifFalse: [^0]

spend: anAmount on: reason
    "Spend anAmount on reason, reducing the available cashOnHand"

    expenditures
        at: reason
        put: (self totalSpentOn: reason) + anAmount.
    cashOnHand := cashOnHand - anAmount.
```

```
printOn: aStream
    "print a suitable representation of myself on aStream"

    super printOn: aStream.
    aStream space.
    aStream nextPutAll: 'balance: '.
    cashOnHand printOn: aStream.
    expenditures keysAndValuesDo: [:reason :amount |
        aStream cr.
        reason printOn: aStream.
        aStream tab.
        amount printOn: aStream]
```

You can now create initialised instances of class SpendingHistory by evaluating the expression:

```
SpendingHistory initialBalance: 600
```

Ex 3.11: To test the new class, type the expressions shown below in a Workspace. Select and evaluate the code using **print it**.

```
| spendingHistory |
spendingHistory := SpendingHistory initialBalance: 800.
spendingHistory spend: 220 on: 'rent'.
spendingHistory spend: 30 on: 'food'.
spendingHistory spend: 45 on: 'drink'.
spendingHistory spend: 20 on: 'petrol'
```

## 3.7. Saving Your Code

Each of the upper panes in the System Browser has a **file out as…** option on the <operate> button menu. This option allows the user to produce a file containing the source code for the selected category, class, protocol or method.

Ex 3.12 Select the category Spending in the Class Categories pane, and choose the **file out as…** option from the <operate> button menu in that pane. You will be prompted to complete a filename specification (see figure 3.2) ending in '.st' (if necessary the filename will be truncated to the constraints of your platform's filing system). Simply press the <CR> key. The cursor will indicate that a file is being written to the disk.

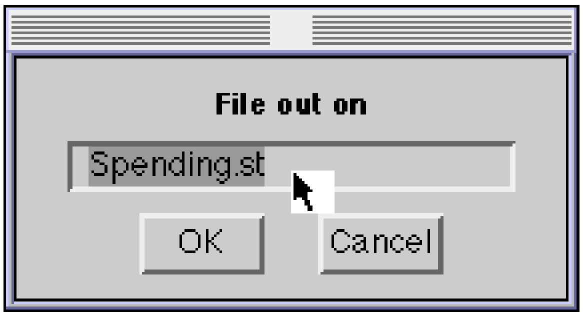


**Figure 3.2: Completing a Filename Specification**

Ex 3.13 File out the diff: method you created earlier.

## 3.8. The File Browser

The File Browser provides browsing access to the operating system’s file management facilities. It allows the user to:

- enumerate files by pattern;
- access information about files (e.g. creation date);
- access the contents of files;
- “file–in” existing source code.
- create, remove or rename files and directories;
- edit files;

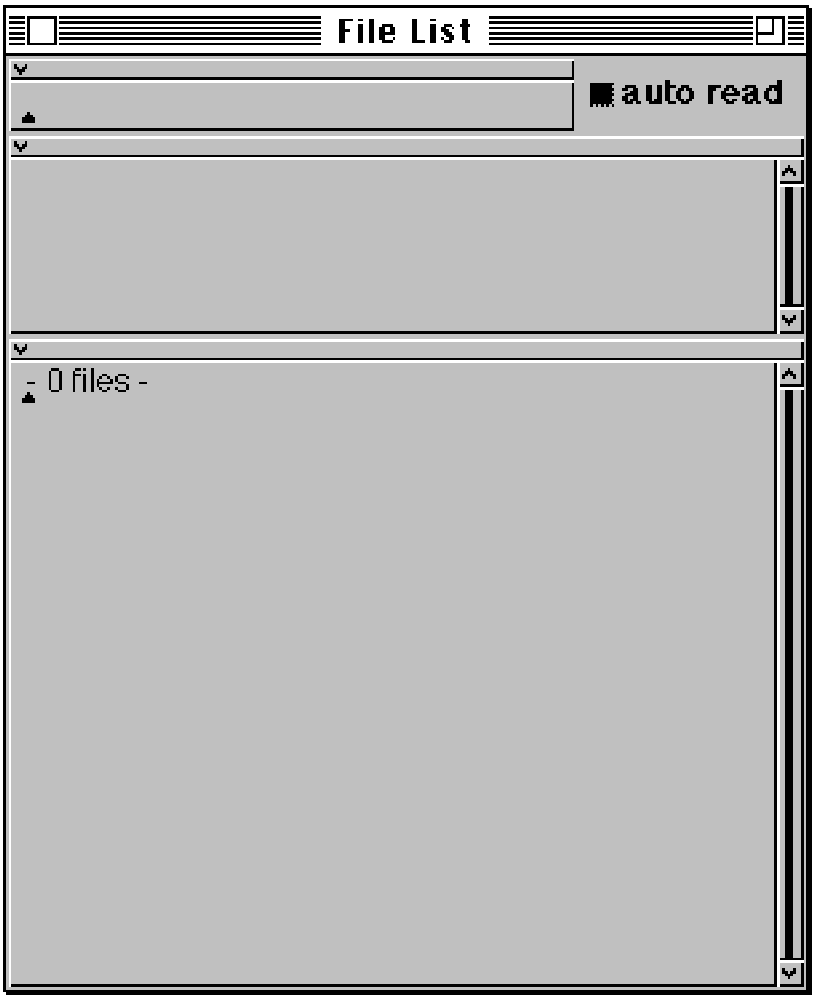


**Figure 3.3: The File Browser**

The File Browser consists of three panes and one button (figure 3.3).

### 3.8.1. The Upper Pane

The upper pane permits a file or directory name to be specified. Parts of the directory structure[1] are separated by appropriate characters (e.g. '/' for UNIX, '\' for PC machines, and ':' for the Macintosh). This pane is used as the initial access point into the file system. Wildcards may be used in the specification of the file or directory. An asterisk ('*') symbol may be used to substitute for any number of characters, and a hash ('#') for an individual character. Note that the label of the window reflects the currently selected file/directory.

The <operate> menu in the pane (figure 3.4) is similar to the standard text–editing menu, with the additional option **volumes...** which displays a menu of sub–directories in the root directory (UNIX); or currently available disk drives (Macintosh and MS–Windows).

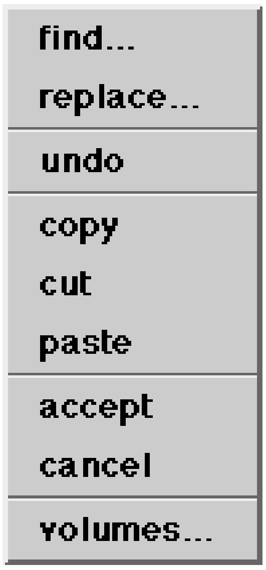


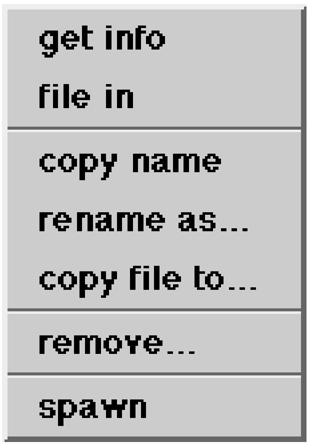
**Figure 3.4: The <operate> menu available in the upper pane of the File Browser**

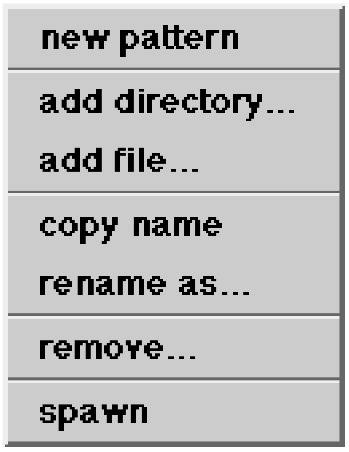
### 3.8.2. The Middle Pane

The middle pane normally contains an alphabetically sorted list of one or more files or directories (e.g. the contents of a directory). One of the items from the list may be selected using the <select> mouse button. Note that the contents of the <operate> menu in this pane depends on whether a file, a directory, or no item is selected.

[1]On the Macintosh the directory structure is replaced by folders. Broadly speaking, each folder on the Macintosh is equivalent to a directory on the other platforms.

If a file or directory *is* selected then the <operate> menu appears as in figures 3.5 and 3.6 respectively.





**Figures 3.5 & 3.6: The <operate> menu options available in the middle pane of the File Browser when a file or directory is selected (respectively)**

There are a number of options common to both menus that apply to the selected file⁄directory:

| | |
|---|---|
| **copy name** | Copies the path name of the selected file or directory so that it may later be pasted. |
| **rename as...** | Changes the name of the selected file⁄directory[1]. This may cause the position of the file to change in the list of files. A Prompter will appear requesting the new name, with the old name as default. |
| **remove...** | Deletes the selected file⁄directory, after prompting for user confirmation. |
| **spawn** | Opens a new File Browser in the selected directory, or opens a File Editor (see later) on the selected file. |

Those options only applicable to a selected *file* are as follows:

| | |
|---|---|
| **get info** | Displays information about the selected file in the lower pane (e.g. creation date, modification date). You should note that this option is replaced by **get contents**, when the lower pane contains file information. |
| **get contents** | Displays the contents of the selected file in the lower pane. |

[1]On some platforms this may produce an error if you do not have permission to change the name of the file.

| | |
|---|---|
| **file in** | Assumes that the file contains Smalltalk code (e.g. that has previously been filed out), retrieves the file contents, reading and evaluating the text. |
| **copy file to…** | Creates a new file after prompting the user for its name. The original file remains selected. If the destination file already exists then the user is prompted to try again (with a different file name) or abort. |

The options available when a *directory* is selected are as follows:

| | |
|---|---|
| **new pattern** | Copies and accepts the currently selected directory into the upper pane and displays its contents in the middle pane. |
| **add directory…** | Prompts the user for the name of a new directory, and creates a new directory with that name as a sub–directory of the selected directory. |
| **add file…** | Prompts the user for the name of a new file, and creates a new empty file with that name *within* the selected directory. |

### 3.8.3. The Lower Pane

The lower pane is where the contents of the selected file (or information about it) may be displayed using the **get contents** (**get info**) option from the <operate> button menu in the middle pane. The contents of a file may be edited using the normal VisualWorks editing conventions.

The options available on the <operate> menu in this pane are dependent on the selection in the middle pane. If a directory is selected then the pane will display the contents of the directory (if any) and the menu will be similar to the usual text editing menu.

If a file is selected in the middle pane and its contents are displayed in the lower pane, then the <operate> menu (figure 3.7) contains the following extra options:

| | |
|---|---|
| **file it in** | Evaluates the text selection as if it were reading it from the selected file. |
| **save** | Writes the contents of the file to disk (e.g. after editing the file). |
| **save as…** | Prompts the user for the name of a new file and then writes the contents of the existing file to a file with that name. |
| **cancel** | Ignores any edits made to the file since it was last saved, and resets its contents. |

**spawn** Opens a File Editor on the selected file with any changes that have been made to it. Cancels any edits that have been made in the original File List Browser.

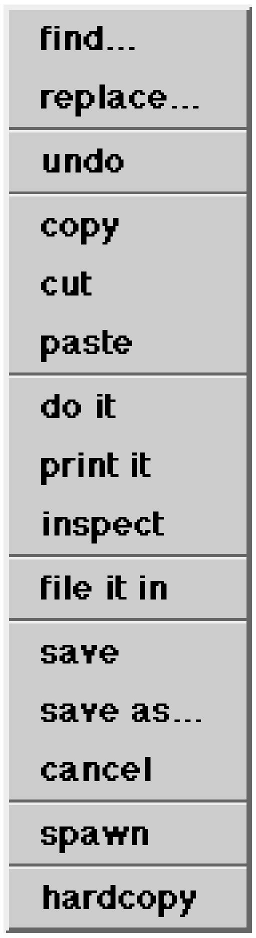


**Figure 3.7: The <operate> menu available from the lower pane of the File Browser when a file is selected**

### 3.8.4. The **auto read** button

The button is used in combination with the lower two panes, and indicates whether or not the contents of selected file should be automatically displayed (without recourse to the **get contents** option), rather than information about the file.

Ex 3.14 Open a File Browser by selecting the **File List** option from the **Tools** menu of the Launcher[1]. View all files in the current directory by typing a '*' in the top pane and using the **accept** option from the <operate> button menu. As a shortcut to using the menu, you may just press the <CR> key here.

Select the file visual.cha in the middle pane, and (if necessary) use the **get contents** option from the <operate> button menu. The contents of the file will be displayed in the lower window.

[1]Note that the menu item is preceded by an icon, a copy of which is present as one of the buttons below the menu bar.

Note that all changes you have made to the image, as well as all code evaluated in a Workspace, have been recorded in this file.

## 3.9. Other Kinds of Browsers

As we have already seen, the System Browser permits access to all classes and methods in the image. Using the System Browser, we can view and modify any method in the image, add and remove methods, as well as adding and removing classes. The System Browser is the most generally useful way of exploring the functionality associated with classes within the VisualWorks image.

During application development, however, it is frequently necessary to view methods in several different (and possibly unrelated) classes, and it is often convenient to be able to browse only a part of a class hierarchy. It is always possible to open two (or more) System Browsers on the screen simultaneously for this purpose; however, the System Browsers take up a lot of screen space and the screen can rapidly become very cluttered and crowded.

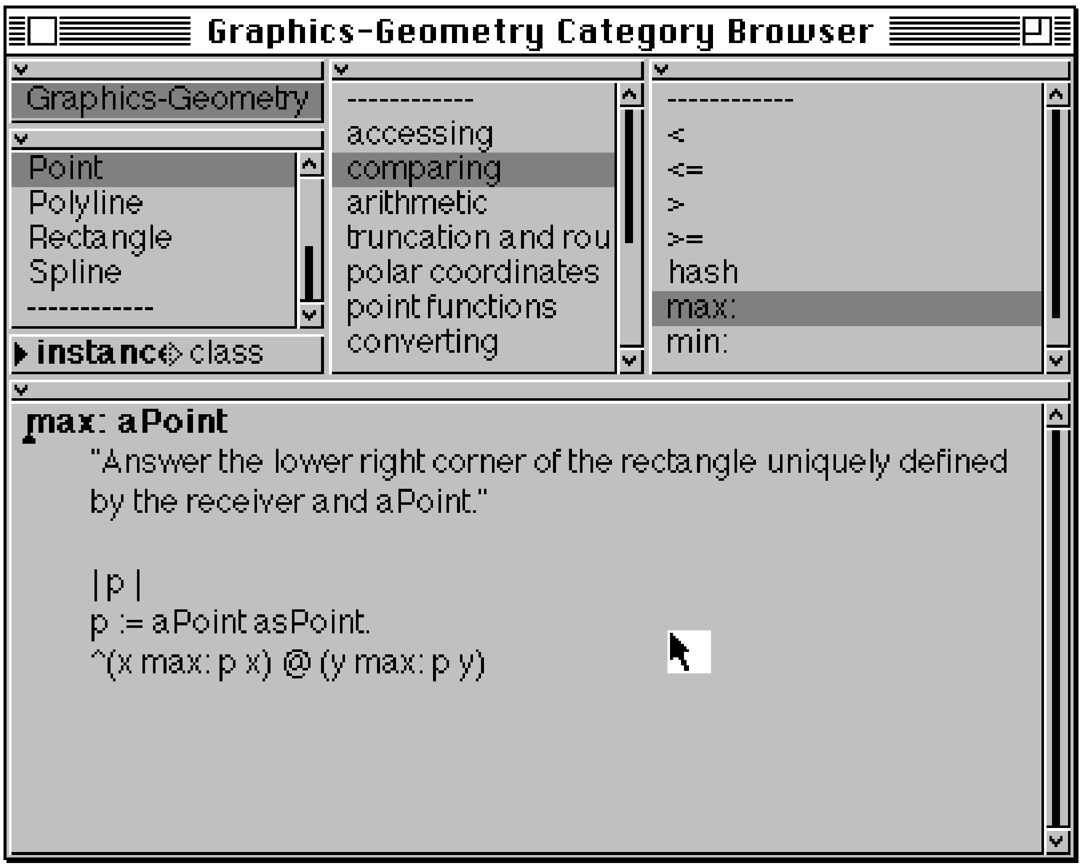
To attempt to alleviate this problem, VisualWorks provides several other kinds of Browsers, each of which permit access to only a limited amount of the image, such as just one class, or even just one method. Although these kinds of Browsers are limited in their access, they occupy less screen space, and are sometimes useful for this purpose.

## 3.10. Spawning other Browsers

Each of the panes in the System Browser has a **spawn** option on the <operate> menu. The <operate> menu available in the Class Names pane also has a **spawn hierarchy** option. Each of these options causes a different kind of Browser to be created, on a limited part of the class hierarchy.

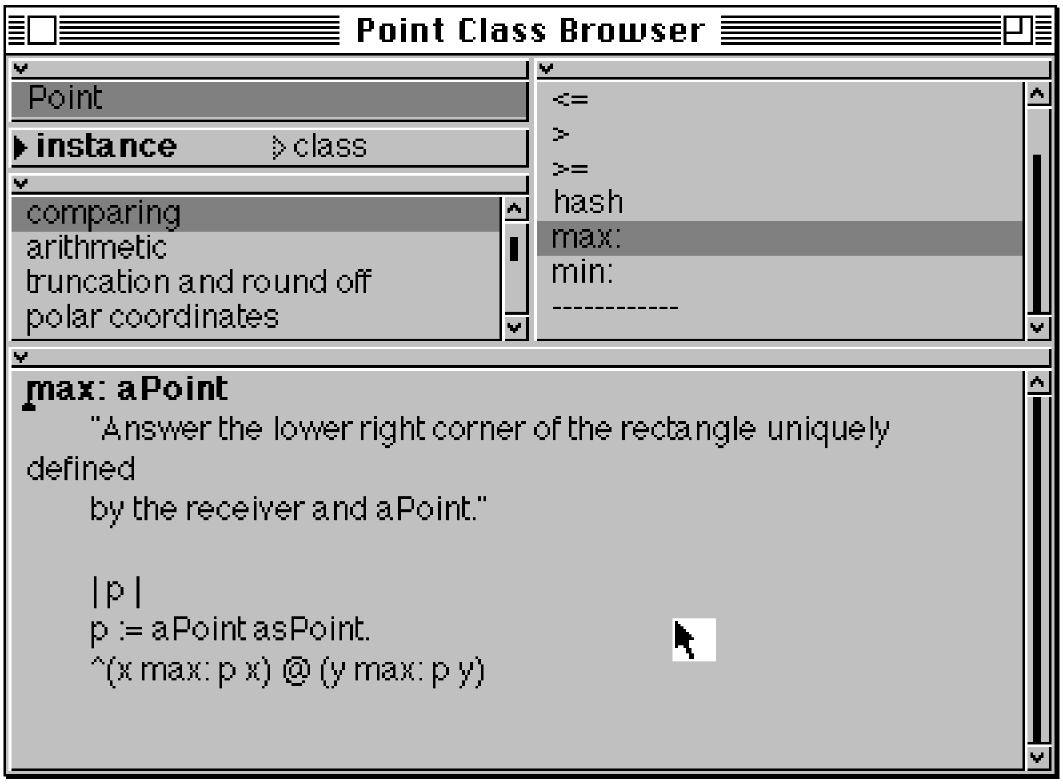
Working from left to right across the System Browser, the **spawn** option on the left–most pane (Class Categories) <operate> menu opens a Browser on only those classes in the selected category — a *Category Browser*. Other classes are not accessible (see figure 3.8).

Two **spawn** options are available from the <operate> menu of the Class Names pane: **spawn** creates a Browser on only the selected class — a *Class Browser*, see figure 3.9. (Other classes are not available.) Alternatively, **spawn hierarchy** creates a *Hierarchy Browser* on all classes in the hierarchy of the selected class (see module 5).



```
max: aPoint
	"Answer the lower right corner of the rectangle uniquely defined
	by the receiver and aPoint."

	| p |
	p := aPoint asPoint.
	^(x max: p x) @ (y max: p y)
```

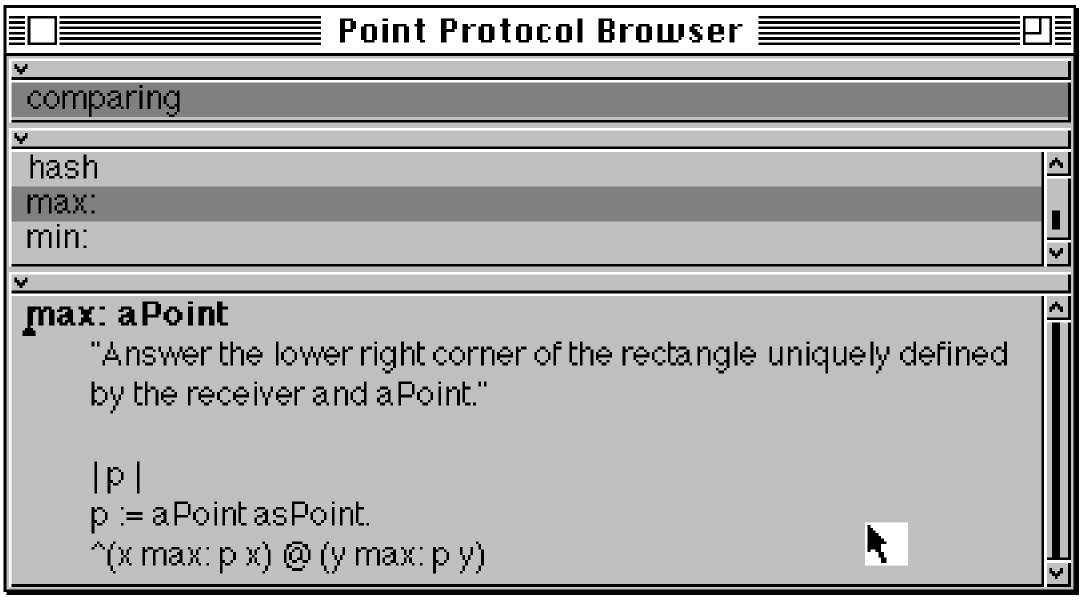
**Figure 3.8: A Class Category Browser**



```
max: aPoint
	"Answer the lower right corner of the rectangle uniquely
defined
	by the receiver and aPoint."

	| p |
	p := aPoint asPoint.
	^(x max: p x) @ (y max: p y)
```

**Figure 3.9: A Class Browser**

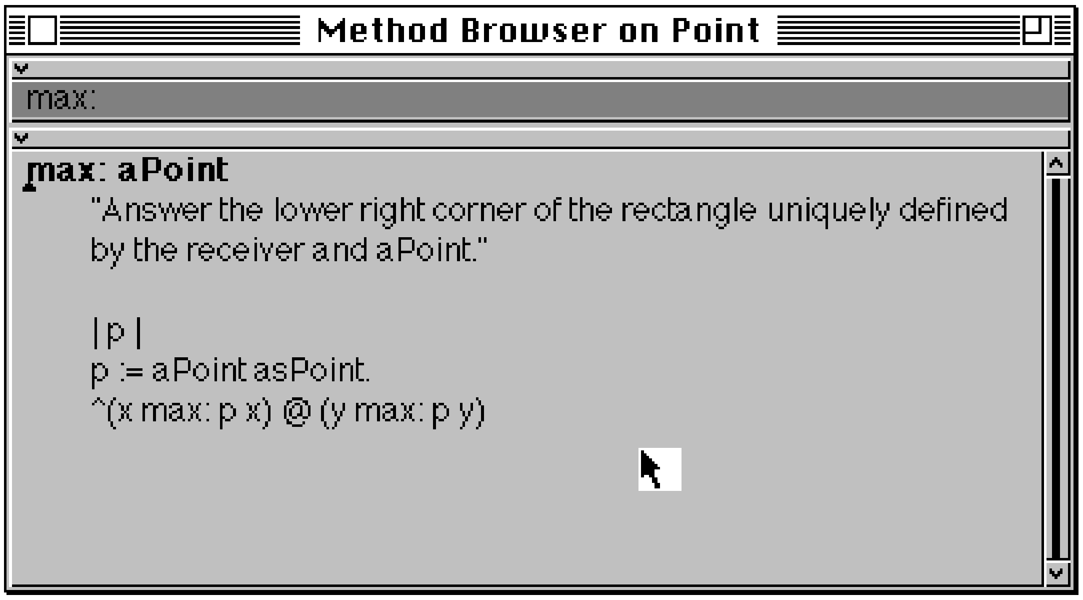
The **spawn** option from the <operate> menu of the Message Categories (Protocols) pane creates a Browser on only the methods in the selected protocol — a *Protocol Browser* (figure 3.10).



```
max: aPoint
	"Answer the lower right corner of the rectangle uniquely defined
	by the receiver and aPoint."

	| p |
	p := aPoint asPoint.
	^(x max: p x) @ (y max: p y)
```

**Figure 3.10: A Protocol Browser**

In the Message Selectors <operate> menu, **spawn** creates a Browser on only the selected method — a *Method Browser* (figure 3.11).



```
max: aPoint
	"Answer the lower right corner of the rectangle uniquely defined
	by the receiver and aPoint."

	| p |
	p := aPoint asPoint.
	^(x max: p x) @ (y max: p y)
```

**Figure 3.11: A Method Browser**

Note that the underlying class structure is equally accessible and modifiable through any of these Browsers (subject to the limitations of what can be located with the particular Browser used). Also, the panes in each of these Browsers have exactly the same <operate> button menu as the corresponding pane in the System Browser.

Finally, it is important to be aware of two consequences of spawning Browsers. If a new Browser is spawned from an existing Browser in which the method source code has been modified but not "accepted", then the original Browser reverts to the original source code (as if the **cancel** option had been selected) and the new Browser contains the modified code. Secondly, note that the consistency of the representation presented to the user when multiple windows are in operation is not automatically managed. Changes made to a class or category in one Browser are not reflected in other Browsers until another selection is made, or the **update** option from the Class Categories <operate> button menu is used.

## 3.11. Alternative Ways of Opening Browsers

Frequently, we will wish to browse on a particular class. Of course, we can always do this by finding the class in the System Browser, but there are two alternatives. The first is to use the **Browse** menu from the Launcher (figure 3.12). (You have used this option before, to open a System Browser.) The menu also contains other options, and the one that is of interest here is the **Class Named…** option. When this option is selected the user is presented with a request for a class. You may simply type the class name in full here, or, if you are unsure, use the "wildcards" described earlier (figure 3.13). If the later approach is taken you will be presented with a list of matching classes (figure 3.14), from which you may only select one.

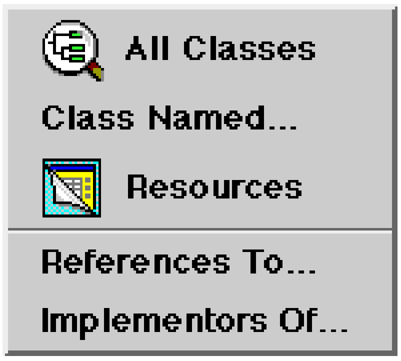


**Figure 3.12: The Browse menu of the Launcher**

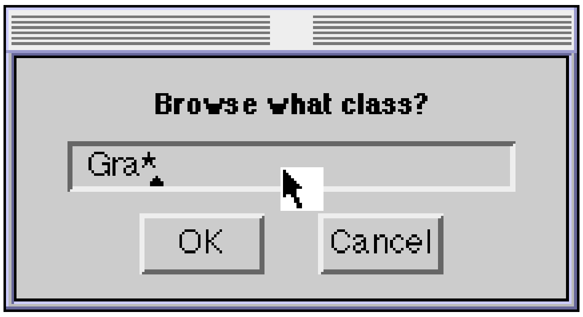


**Figure 3.13: Entering a class name**

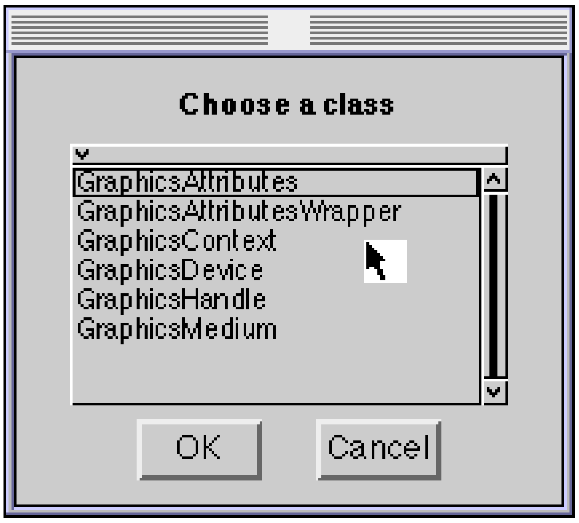


**Figure 3.14: A Confirmer containing a list of classes matching the above**

The other alternative is to open a Class Browser directly from a Workspace. This can be done by sending the newOnClass: message to class Browser; for example, a Browser on class Point can be created by typing and evaluating the following expression in a Workspace:

```
Browser newOnClass: Point
```

This is so useful that a shorthand way of opening a Class Browser is to send the message browse to that class, or to send the message browse to any instance of that class. So, alternative ways of creating a Browser on class Point would be:

```
Point browse.
(3@4) browse.
```

The Class Browsers created in this way are identical to those created when the **spawn** option is used from the Class Names <operate> menu.

A complete System Browser can be opened using the expression:

```
Browser open
```

Ex 3.15: Experiment with creating various Category, Class, Protocol, and Method Browsers from existing Browsers.

Ex 3.16: Experiment with creating various types of Browsers, by typing and evaluating expressions as suggested above.

Ex 3.17: Browse the instance creation class methods of class Browser and try creating some other kinds of Browsers using the messages found there.

Ex 3.18: Note how Browsers may be spawned from Browsers other than System Browsers.

Ex 3.19: Modify a method (non–destructively or reversibly!) in one Browser.

Ex 3.20: Verify that the changes are visible in other Browsers only after re–selection or the use of the **update** menu option.

## 3.12. Browser Menus

Using the System Browser as an example we will consider each menu option from the available <operate> menus. The Browsers that may be spawned from the System Browser (or produced as a result of evaluating a message expression, from a Workspace for example) all share these menus. Some of the options have already been described, the remainder will be described here.

### 3.12.1. Hardcopy

Before we examine each of the <operate> menus in turn, let us first describe the **hardcopy** option, since it is present in all the <operate> menus in the Browsers, and many of the other text–based windows. This option produces a file containing the source code for the selected method, protocol, class or category and sends it to the printer. (The format of the file is similar to that produced by the **file out as…** option, but doesn't contain any special characters recognised by the compiler — i.e. it may not be later filed–in.) After the operation has been successfully completed a notification is written to the System Transcript.

### 3.12.2. The Class Categories Menu

The Class Categories menu comes in two guises, according to whether or not a category is selected. Figure 3.15 shows the full menu, with those options always available indicated by a '†'.

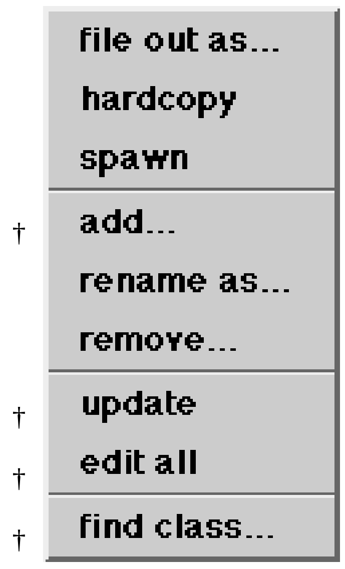


**Figure 3.15: The Class Categories <operate> menu**

| | |
|---|---|
| **rename as...** | The user is prompted to provide a new name for the selected category. This option is often used when there is a conflict between category names or when correcting a typing mistake. All classes in the selected category are modified accordingly. The operation fails if the user provides a category name that already exists (unfortunately, no feedback is given to the user in these circumstances)! |
| **remove...** | This option removes the selected category *and the classes it contains* from the VisualWorks image. Not recommended, unless you are sure you know what you are doing — it is possible to remove classes which will prevent any further work being carried out in this image! The user is prompted to confirm the operation if the category contains any classes. |
| **edit all** | Prints the structure of all categories and their classes in the lower text pane. The order of the categories is the same as that in the class categories pane. By editing this list the user may change category names, category contents, and the order in which the categories will appear. Although it is also possible to modify class names, all such changes are ignored. The user must select **accept** from the text pane <operate> menu to see the result of any changes that have been made. This option is seldom used, since other options exist by which the same modifications may be (more safely) carried out. Not recommended. |

**find class...** The user is prompted to specify a class name (possibly including the wildcards '*' and '#'). The category containing the specified class is selected in the category pane and the class is selected in the class pane. This mechanism is similar to that described earlier for opening a Browser on a specified class, i.e. if more than one class matches the specified name, then the user is prompted to select the required class.

Ex 3.21: Rename the class category Spending as Finance using the **rename as...** menu option

Ex 3.22: Undo the modification in exercise 3.21 using the **edit all** option.

Ex 3.23: File out the class categories Spending and Financial Tools. Examine the files using the File Editor .

Ex 3.24: Remove the class category Financial Tools.

Ex 3.25: File–in the category Financial Tools from the file–out you created in exercise 3.23.

Ex 3.26: Experiment with the **find class...** option.

### 3.12.3. The Class Names Menu

The Class Names menu (or "Class Menu") is only available if a class is selected (figure 3.16).

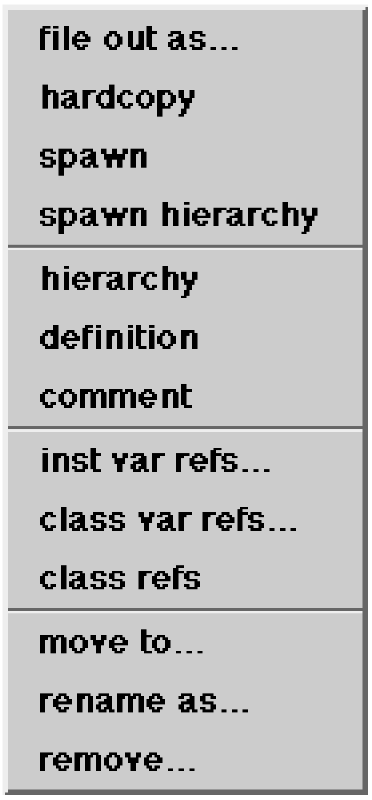


**Figure 3.16: The Class menu**

| | |
|---|---|
| **inst var refs…** | A menu is displayed listing the instance variables of the class and its superclasses. (The variables in each class are identified appropriately.) When the user selects a variable name, a Message–Set Browser (see later) is opened on the methods in which the selected variable is referenced. For example, if the class FinancialHistory is selected[1], the **inst var refs…** option produces a list of its instance variable names (figure 3.17). |

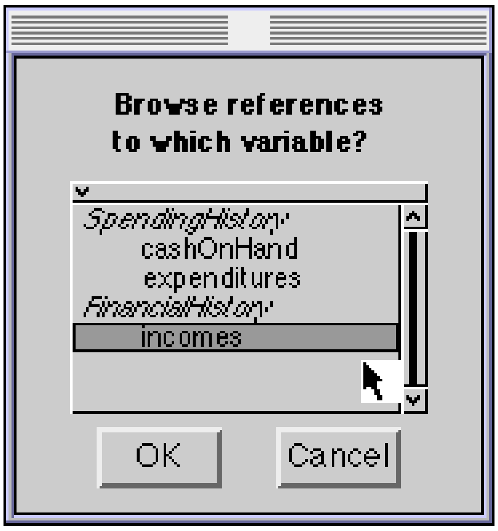

Browse references to which variable?

**Figure 3.17: A list of instance variable names for the class FinancialHistory**

| | |
|---|---|
| **class var refs…** | Similar to above, except that the resulting menu contains a list of the *class* variable names for the class and its superclasses. |
| **class refs** | This option opens a Message–Set Browser on all those methods in which the *class* is referenced. |
| **move to…** | The user is prompted for a category name (which may be new or existing) into which to move the class. The System Browser is updated accordingly. |
| **rename as…** | The user is prompted to provide a new class name for the selected class. If appropriate, a Message–Set Browser is opened containing the methods in which the class is currently referenced (using its old name); if necessary these methods should be amended. We recommend that you first check for references to this class (using the **class refs** option above) and make any necessary modifications *before* renaming the class. |

[1]Class FinancialHistory is introduced in module 5.

**remove...** The user is prompted to confirm this operation. It's easier to remove references to this class before you remove the class itself!

Ex 3.27: Open a Browser which contains all methods referencing the class variable DefaultForWindows in class LookPreferences.

Ex 3.28: Open a Browser which contains all methods which reference the class LookPreferences.

Ex 3.29: Move the class SpendingHistory into the category called 'Financial Tools'.

Ex 3.30: What will happen if you try to rename the class SpendingHistory as 'ExpendituresHistory'? Go ahead and see if you are correct.

Ex 3.31: What will happen if you try to remove the class SpendingHistory? Go ahead and see if you are correct. (This assumes that you didn't rename the class in exercise 3.30!)

### 3.12.4. The Message Categories Menu

The Message Categories menu (or "Protocols Menu") is similar to the Class Category menu since it has two guises, according to whether or not a protocol is selected. Figure 3.18 shows the full menu, with those options always available indicated with a '†'.

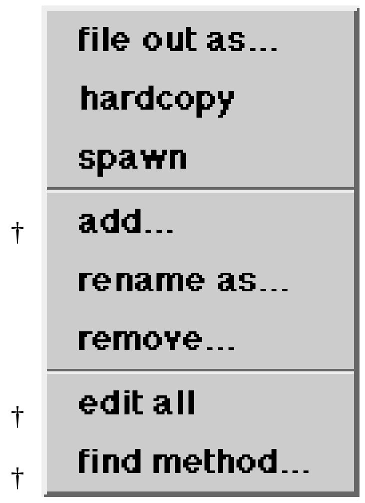


**Figure 3.18: The Protocols Menu**

**rename as...** Prompts the user for a new name for the selected protocol, then updates the protocol list in the Browser.

**remove...** After prompting the user for confirmation, this option removes the protocol *and the methods it contains*. However, there is no attempt to discover if any other methods send messages corresponding to the methods contained in the selected protocol you are about to remove. (To determine all senders of a given message, see the **senders** option from the Message Selectors menu below.)

**edit all** Prints the structure of the protocols and the methods they contain in the lower text pane. The order of the protocols is the same as that in the protocol pane. By editing this list the user may change protocol names, protocol contents, and the order in which protocols will appear. Although it is possible to modify message selector names, all such changes are ignored. The user must select **accept** from the text pane <operate> menu to see the result of any changes that have been made. This option is seldom used, since other options exist by which the same modifications may be (more safely) carried out. Not recommended.

**find method...** This option displays a list of the message selectors provided by the selected class. The instance⁄class switch on the Browser determines which group of selectors is displayed in the menu. Selecting a selector from the list causes its protocol to be selected in the protocol pane, itself to be selected in the message selectors pane, and its method to be displayed in the lower text pane. For example, figure 3.19 shows the list of selectors in class SpendingHistory.

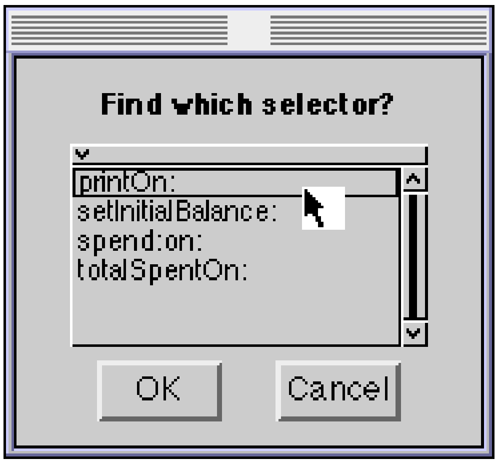


**Figure 3.19: The list of message selectors defined for class SpendingHistory**

Ex 3.32: In class SpendingHistory, rename the protocol private as initialize-release.

Ex 3.33: Reverse the modification made in exercise 3.32, by using the **edit all** menu option.

Ex 3.34: File–out the class protocol instance creation in class SpendingHistory. Examine the file using the File Editor

Ex 3.35: Remove the class protocol instance creation from class SpendingHistory. File–in the protocol from the file–out you created in exercise 3.34.

### 3.12.5. The Message Selectors Menu

The Message Selectors menu is only available if a message selector is selected (figure 3.20).

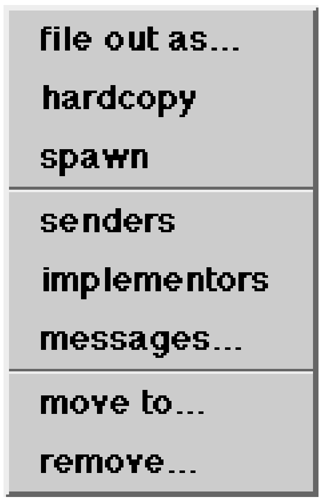


**Figure 3.20: The Message Selectors menu**

The first three menu options are very useful in tracing message–sends[1]:

| | |
|---|---|
| **senders** | This option searches the VisualWorks image for all methods in which the message selector is sent. A Message–Set Browser is opened. If the selected message selector is not sent by any method in the image, then the user is presented with a “warning” Prompter. |
| **implementors** | This option opens a Message–Set Browser on all classes that implement a method corresponding to the selected message selector. |
| **messages...** | This option displays a menu of the messages sent in the selected method. Selecting one of these messages opens a Message–Set Browser on the implementors of that message. |
| **move to...** | The user is prompted for the name of the destination protocol into which the selected method will be moved. If a protocol of that name does not exist then it is created. To *copy* the method to another class, the user must include both the class name and the message protocol, in the form *ClassName>selector*. |

[1]See also later, for alternative ways of browsing the senders and implementors of a message.

**remove...** The method is deleted after confirmation from the user. It's advisable to ensure that there are no senders of the message (using the **senders** option above) before removing a method!

Ex 3.36: Select the message selector printOn: in class SpendingHistory. Open a Browser on all senders of this message.

Ex 3.37: Whilst having the same message selected, open a Browser on all implementors of printOn:.

Ex 3.38: Again from the same message, browse all implementors of keysAndValuesDo:.

### 3.12.6. The Text Pane Menu

The text pane menu (or "Code" menu) is always available, since the pane can also be used as a workspace for experimentation (figure 3.21) .

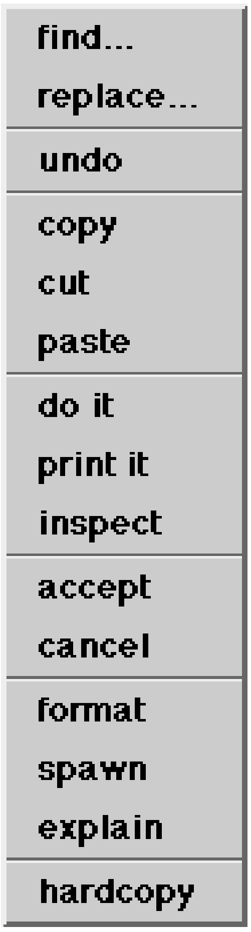
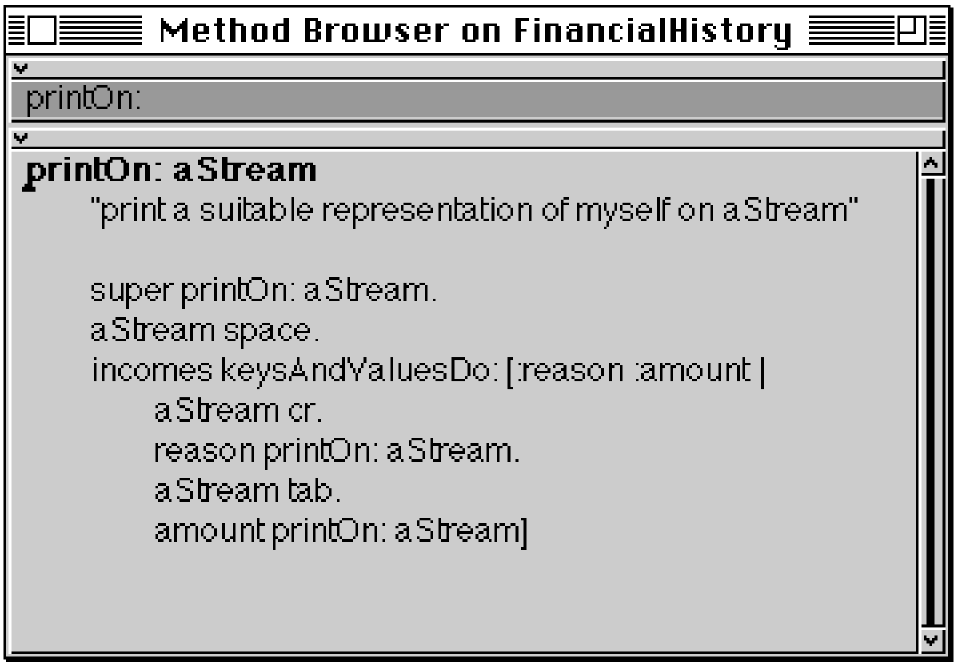


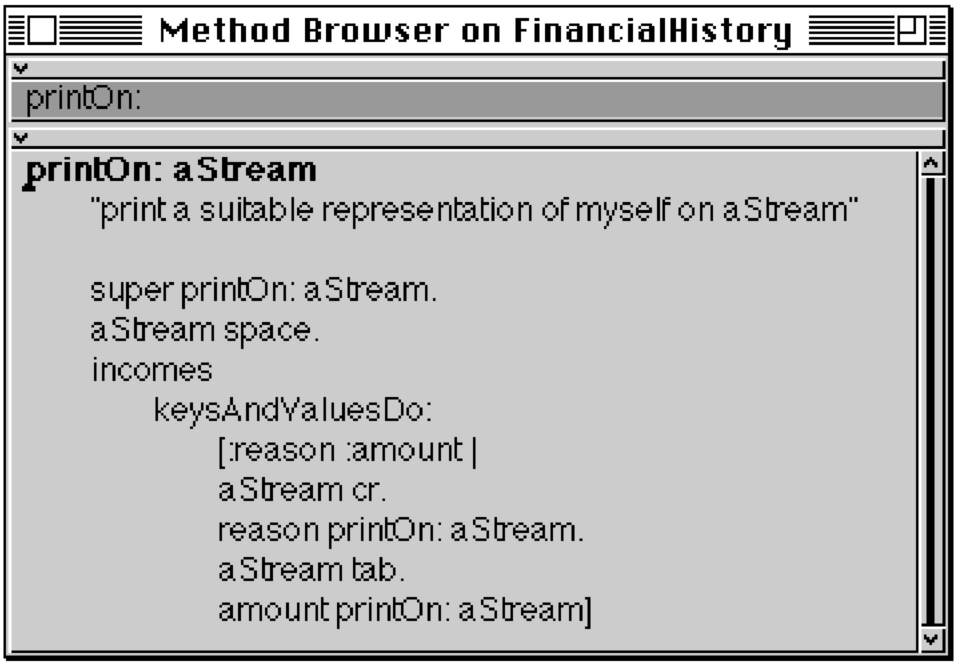
**Figure 3.21: The Code menu**

**format** This option modifies the layout of the method so that it adheres to the code indentation conventions. For example, figures 3.22 and 3.23 display the original printOn: method in class FinancialHistory before and after the use of the **format** option respectively.



```
printOn: aStream
	"print a suitable representation of myself on aStream"

	super printOn: aStream.
	aStream space.
	incomes keysAndValuesDo: [:reason :amount |
		aStream cr.
		reason printOn: aStream.
		aStream tab.
		amount printOn: aStream]
```

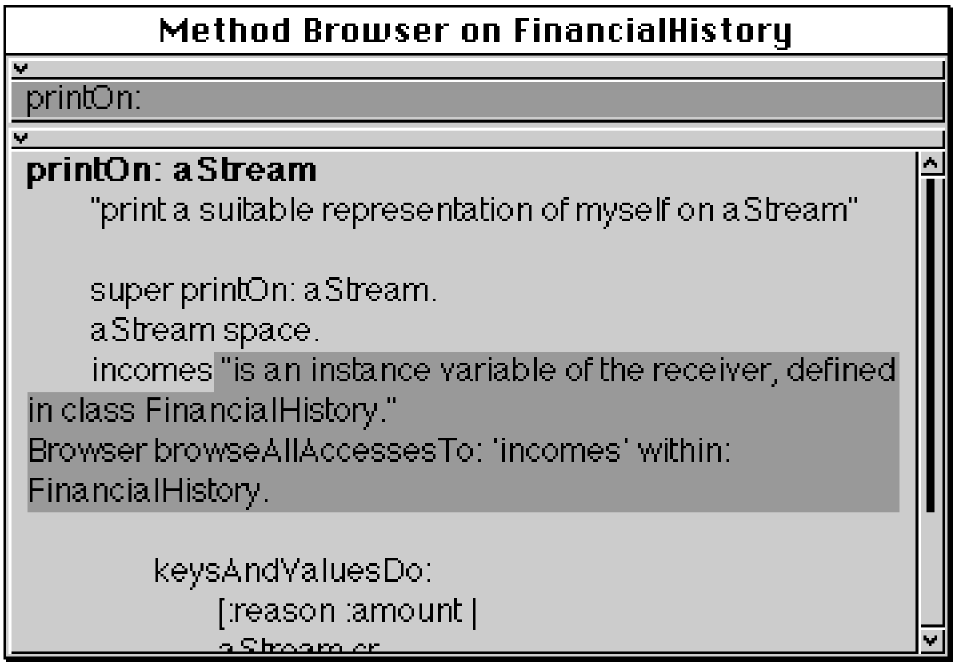
**Figure 3.22: The printOn: method *before* formatting**



```
printOn: aStream
	"print a suitable representation of myself on aStream"

	super printOn: aStream.
	aStream space.
	incomes
		keysAndValuesDo:
			[:reason :amount |
			aStream cr.
			reason printOn: aStream.
			aStream tab.
			amount printOn: aStream]
```

**Figure 3.23: The printOn: method *after* formatting**

**explain** Used when a variable, literal or message selector is selected, this option appends an “explanation” of the selection. The explanation usually includes some code which, when evaluated, opens a Browser on references to the selection. For example, in figure 3.24 the Browser contains an explanation of the instance variable incomes.



```
printOn: aStream
	"print a suitable representation of myself on aStream"

	super printOn: aStream.
	aStream space.
	incomes "is an instance variable of the receiver, defined
in class FinancialHistory."
Browser browseAllAccessesTo: 'incomes' within:
FinancialHistory.

		keysAndValuesDo:
			[:reason :amount |
```

**Figure 3.24: An "explanation" of the variable incomes**

Ex 3.39: Ensure that the methods in classes SpendingHistory adhere to the code formatting conventions.

Ex 3.40: "Explain" the message space in the method pictured above.

You can also find all senders and implementors of a message (respectively) by evaluating expressions of the form:

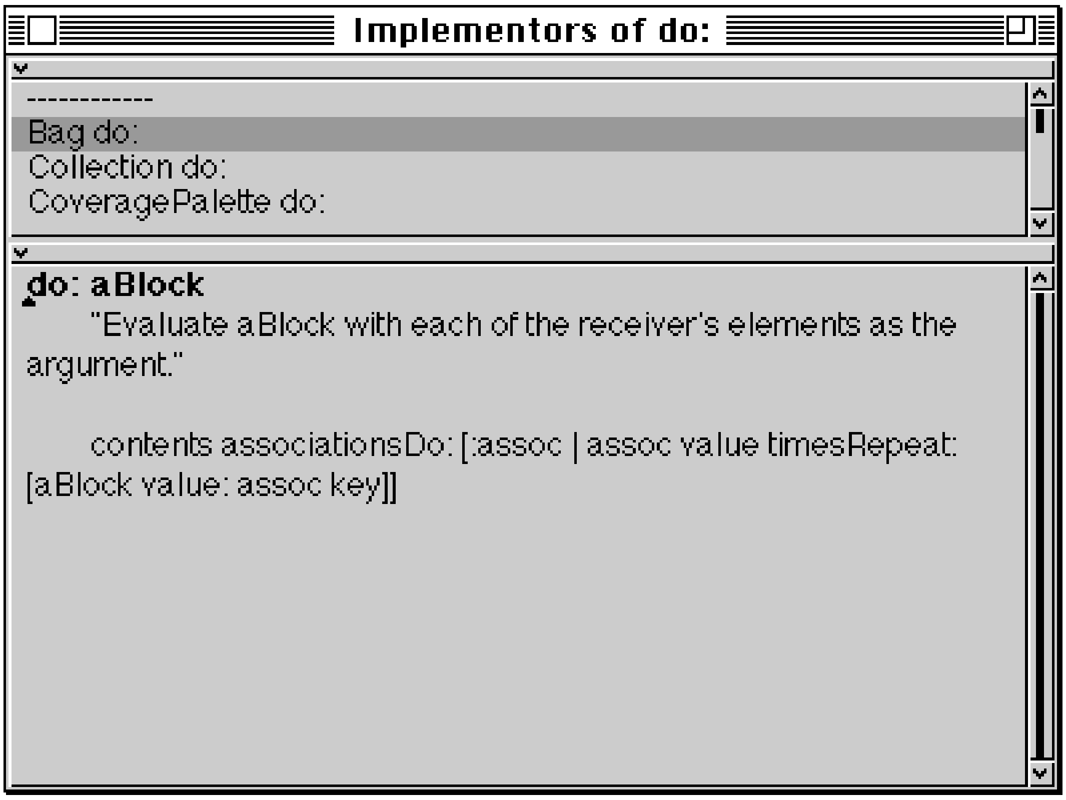
```
Browser browseAllCallsOn: #printOn:

Browser browseAllImplementorsOf: #do:
```

You can find all references to a class by:

```
Browser browseAllCallsOn: (Smalltalk associationAt: #Array)
```

## 3.13. Message–Set Browser

A Message–Set Browser is a special kind of Browser that gives access to a collection of methods with specific characteristics. For example, figure 3.25 shows a Message–Set Browser containing all implementors of do:. You can see that the upper pane contains a list of class–selector pairs corresponding to the label of the Browser.



```
------------
Bag do:
Collection do:
CoveragePalette do:
```

```
do: aBlock
	"Evaluate aBlock with each of the receiver's elements as the argument."

	contents associationsDo: [:assoc | assoc value timesRepeat: [aBlock value: assoc key]]
```

**Figure 3.25: A Message–Set Browser**

## 3.14. The Launcher

We have already described most of the operations available from the Launcher. The following sections describe the remainder, and identify those options already described.

### 3.14.1. File

The options available from the **File** menu (figure 3.26) are described below:

| | |
|---|---|
| **Collect Garbage** | collects objects in the image that are no longer required (called *garbage*) and discards them, thus removing them from memory. Although this process occurs automatically at regular intervals, you may want to use this option to discard objects for a specific reason. The operation also writes a message to the Transcript indicating how much space remains. |
| **Collect All Garbage** | performs a similar operation to above. In addition, this operation searches for garbage in a memory zone called *PermSpace*. Consult your User Guide for more information on both options. |
| **Settings** | opens a window in which various options can be set. See module 2. |

We have already described the **Save As…** option in module 2. There are two other options for saving, **Perm Save As…** and **Perm Undo As….** These more advanced options provide a means of using PermSpace; consult your User Guide for more information.

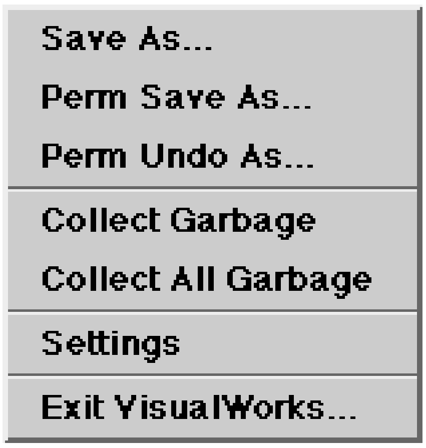


**Figure 3.26: The File menu of the Launcher**

You should also be familiar with the dialogue box that is produced when you select the **Exit VisualWorks…** option from the Launcher (figure 3.27). This was described in module 2.

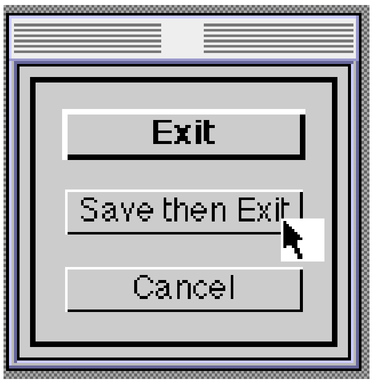


**Figure 3.27: The Exit VisualWorks… dialogue box**

### 3.14.2. Browse

The **Browsers** menu of the Launcher (figure 3.28) contains five options. By now you should be very familiar with the **All Classes** option. The **Class Named…** option was described earlier The **References To…** and **Implementors Of…** options provide the same functionality as the **senders** and **implementors** options described earlier in this module. The remaining option, **Resources**, is described later in the course.

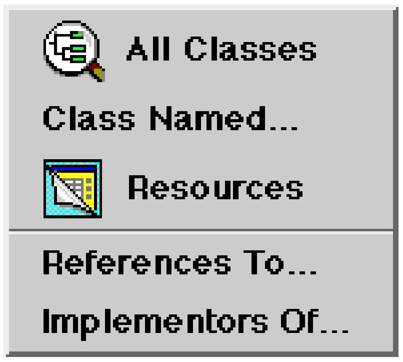


**Figure 3.28: The Browse menu of the Launcher**

### 3.14.3. Tools

Four of the options available from the **Tools** menu (figure 3.29) have all been described elsewhere, others are described later in the course. The remainder are beyond the scope of this course.

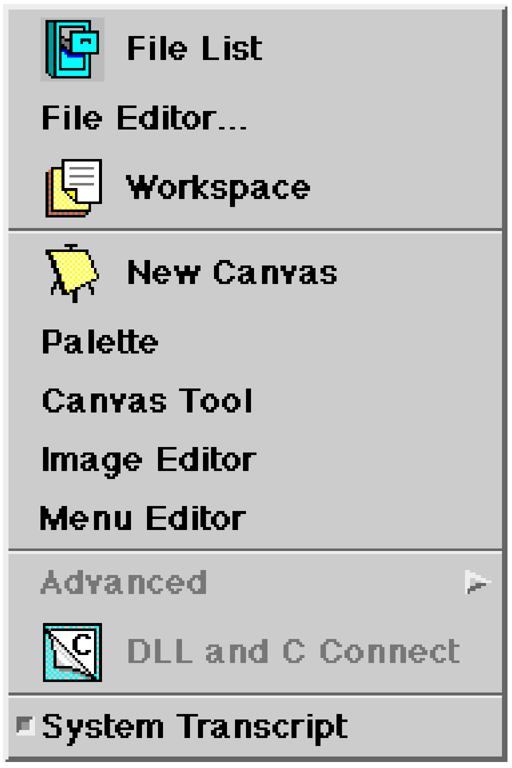


**Figure 3.29: The Tools menu of the Launcher**

### 3.14.4. Changes

The options available from **Changes** menu of the Launcher are not described in this course.