Goal

This change tries to simplify the model and improve the separation between content and presentation regarding the class list in the browsers.

In particular, the `#classList` message (implemented in `Browser`, `HierarchyBrowser`, `CodeFileBrowser`, and `ChangeSorter`) returns a collection of strings representing class names, sometimes with leading-blank indentation for hierarchy display. Senders that need actual class objects must resolve those names themselves (via code like `Smalltalk classNamed: classItem withoutLeadingBlanks asSymbol`, etc.), and so they end up being coupled with the display format. If we wanted to change the way the class items in the browser look like, we'd have to change all of the senders of `#classList` to take that formatting into account.

Following are some details that can be used while reviewing (and looking at) the code.

Description of the changes

1. `make_basicClassList_return_classes`

Makes the `#basicClassList` method on `Browser` return class objects instead of a symbol array.

Changes:

- ➔ `Browser>>basicClassList` now returns an array of classes.
- ➔ `Browser>>classList` is updated so that the non-hierarchical branch collects the class names (to keep returning strings).
- ➔ `Browser>>hierarchicalClassList` now receives classes directly from `#basicClassList` instead of resolving the names itself.
- ➔ `SinglePackageBrowser>>basicClassList` also resolves class names at the end before returning, to be consistent with `Browser>>basicClassList`.

2. `make_CodeFileBrowser_inherit_classList`

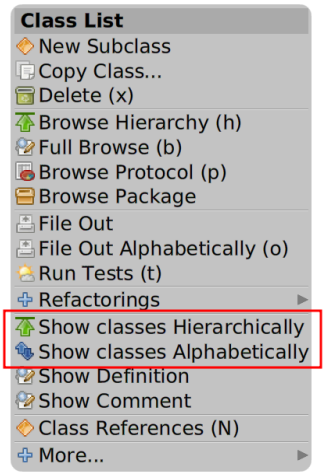
The old `CodeFileBrowser>>classList` duplicated the hierarchical/alphabetical branching that `Browser>>classList` already does. Since `CodeFileBrowser` is a subclass of `Browser`, this step:

- ➔ Overrides `#basicClassList` returning the sorted pseudo-class objects.
- ➔ Adds the relevant if condition from the current `CodeFileBrowser>>classList` to `#hierarchicalClassList`.
- ➔ Removes `CodeFileBrowser>>classList`, since now inheriting the one from `Browser` is enough.

3. `make_HierarchyBrowser_inherit_classList_and_fix_bug`

The current `#classList` method in `HierarchyBrowser` has a side-effect: it mutates the `classList` instance variable by filtering out removed classes every time it's run. `classList` stores the class names with spaces, to build the hierarchical class list. The formatting is done in `HierarchyBrowser>>initHierarchyForClass:` (and it ignores the value of the inherited `listClassesHierarchically` instance variable). It also does not use the logic from `Browser>>classList`.

One consequence of this is that the contextual menu option "Show classes Alphabetically" (of the class list on the hierarchy browser) does not work.



That bug will be solved in this step, which:

- ➔ Overrides `#basicClassList` to resolve class names from the `classList` instance variable, select the valid ones, sort them, and return class objects.
- ➔ Changes `#classListIndex:` to work with the result of `#classList`.
- ➔ Updates `#initAlphabeticListing` and `initHierarchyForClass:` to initialize `listClassesHierarchically`, so that we can reuse the logic from `Browser>>classList`.
- ➔ Lastly, it removes `HierarchyBrowser>>classList`, so that now the logic from `Browser` is inherited.

4. `make_classList_in_HierarchyBrowser_store_just_class_names`

We now clean up the `classList` instance variable in `HierarchyBrowser` so that it stores just the class names, without formatting.

This step:

- ➔ Changes `#initHierarchyForClass:` to store just the class names, simplifying its logic (and making it a bit more declarative).
- ➔ Simplifies `#classListIndex:` because now it doesn't need to send `#withoutLeadingBlanks`.
- ➔ Similarly simplifies `#basicClassList` and `#potentialClassNames`.

- ➔ Adds `PseudoClass>>allSubclasses`, which is needed because `HierarchyBrowser` can be opened on a `PseudoClass` by using the "hierarchy" button (it delegates to the real class, which preserves the current behavior).
- ➔ This changeset also includes an expression to migrate existing instances of `HierarchyBrowser` (stripping leading blanks from their `classList` instance variable), so that they continue working if some are opened during the code review.

5. `make_Browser_classList_return_a_list_of_classes`

After this change, `Browser>>classList` returns classes instead of strings. This implies separating the formatted strings (which will be used as the list items in the UI) from the class names. I chose to call the formatted display strings "class items".

This step:

- ➔ Introduces `#classesToClassItems`, which returns `Association`s of `class -> classItem`. For hierarchical mode, the class item includes indentation. For alphabetical mode it's just the class name.
- ➔ Updates `#flattenHierarchyTree:on:indent:by:` to build the association.
- ➔ Introduces `#classItems`, which is the UI-facing list; and redefines `classList`, now the model-facing list. Both use `#classesToClassItems`.
- ➔ Updates `classListIndex:`, it uses `at:ifPresent:ifAbsent:` on the classList (which now contains class objects) to extract the name.
- ➔ Updates `classListIndexOf:` to directly compare by class name.
- ➔ Updates `BrowserWindow>>buildMorphicClassList` to switch the list getter from `#classList` to `#classItems`.
- ➔ Adds `Browser>>changed:` override, so that when `#classList` fires, also fires `#classItems` so the view updates.
- ➔ Simplifies `CodeFileBrowserWindow>>findClass`.
- ➔ Uses `#classList` from `HierarchyBrowser>>classListIndex:`.
- ➔ This changeset also includes a migration for existing `BrowserWindow` subinstances.

6. `use_basicClassList_in_moveAllToOtherSystemCategory`

`Browser>>moveAllToOtherSystemCategory` previously used `SystemOrganization classesAt:` to get actual classes for the move operation, with a comment noting that `#classList` wasn't safe for this.

Now that `#basicClassList` returns class objects, it can simply send `self basicClassList` for both the emptiness check and the iteration, removing the old workaround.

7. `make_ChangeSorter_classList_return_a_list_of_classes`

We now apply the same class-list/class-items split as step 5, this time to `ChangeSorter`.

This step:

- ➔ Introduces `#classesToClassItems`, which resolves each class name from `myChangeSet changedClassNames` to a class (or a `PseudoClass` stand-in if the class was removed) and returns `classOrPseudoClass -> classItem` associations. I wasn't 100% sure if we should use a `PseudoClass` here, but it works, and it seems it's our best existing option to represent this[1].
- ➔ Introduces `#classItems`, the UI-facing list; and redefines `#classList` to return class objects.
- ➔ Adds a `ChangeSorter>>changed:` method to propagate `#classList` to `#classItems`.
- ➔ Defines `#classListIndex` and `#classListIndex:`.
- ➔ Updates `ChangeSorterWindow>>buildMorphicWindow` to switch the class list widget from `PluggableListMorphByItem` to a plain `PluggableListMorph` reading `#classItems` / `#classListIndex` / `#classListIndex:`, to have a similar logic as in `BrowserWindow`.
- ➔ This changeset also includes a migration for existing `ChangeSorterWindow` subinstances (replacing the `PluggableListMorphByItem` with a `PluggableListMorph` wired to the new selectors).

8. `remove_now_unused_methods_from_ChangeSorter`

This is a cleanup change, which:

- ➔ Inlines `#currentClassName:` in `ChangeSorter>>classListIndex:`.
- ➔ Removes `ChangeSorter>>currentClassName` and `ChangeSorter>>currentClassName:`.

9. `extract_labelForClass_method`

This step consolidates the creation of the label for classes in a single method. It:

- ➔ Introduces `CodeProvider>>labelForClass:` (returns `aClass name`) as a “hook” method on the common superclass between `Browser` and `ChangeSorter`.
- ➔ Updates `Browser>>classesToClassItems` and `Browser>>flattenHierarchyTree:on:indent:by:` to send `#labelForClass:`.
- ➔ Updates `ChangeSorter>>classesToClassItems` to send `#labelForClass:`.

This sets up a single extension point: we can override `labelForClass:` to customize how class names appear on the class lists without touching the list-building logic.

---

[1] I think having `PseudoClass`es is a first step towards the principle of "Temporal Correspondence" as defined by Gilad Bracha and David Ungar in the Mirrors paper. However, I also know that we aren't there yet. In the case of the `ChangeSorter`, where we're (mostly) browsing existing classes *but with the aim of reviewing the changes for filing-out*, so it can be a bit confusing...

10. haveFun

This last change is not a real proposal for integration, but it serves to test the separation that was done during the previous steps, and it might give us some ideas for a potential change. It adds icons to the class items in the browser.

To do that, I had to define a new `Text class>>withDecorativeForm:` method, which is similar to `#withForm:` but without adding an asterisk character to the text representation (so that going to a class by typing a letter while on the browser still works). For the icons themselves, I'm using the IntelliJ Platform Icons (https://intellij-icons.jetbrains.design/), which are under an Apache 2 license.

Of course, to properly implement this we should avoid having everything in one method (e.g. have those icons defined on the current theme), scale the icons when the size of GUI elements change, and so on. But I think this is a nice test of what the design starts to enable after the previous refactorings, especially considering we only needed to change a single method.