Glasgow Calculator Framework Design Document

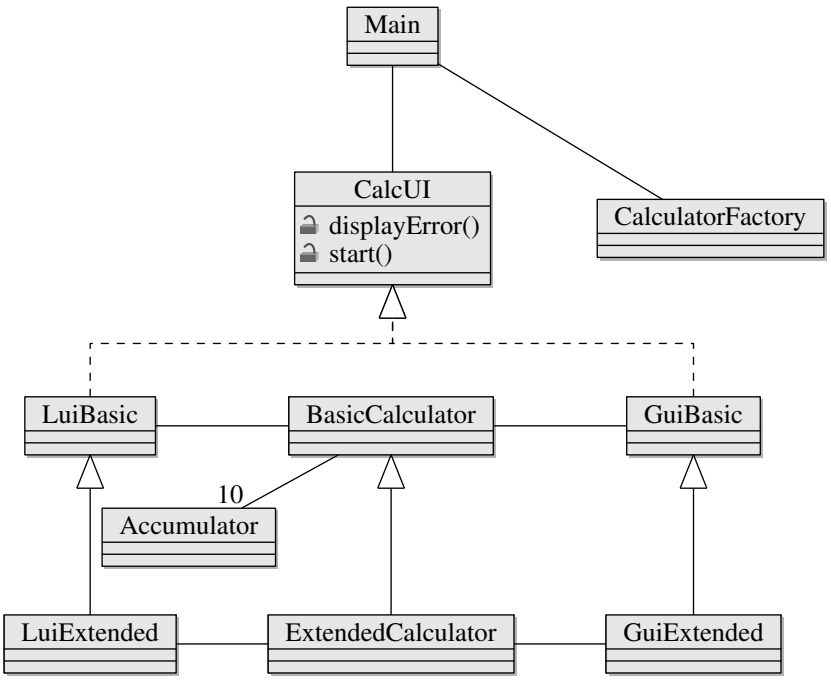
Tim Storer
February 24, 2010

This document describes the design of the *Glasgow Calculator Framework*.

1 Design Overview

Calculators built using the Glasgow Calculator Framework expect the full functionality of version 6 of the Java Standard Edition platform (J6SE). They should work on all platforms that claim J6SE compliance.

An overall structure for the design of the Glasgow Calculator Framework is shown below.



1.1 Slots

Since all calculators must support the functionality of the BasicCalculator (R1) and support both line-oriented and graphical user interfaces (R2), three base classes are defined in the framework:

- `oose2.ex2.model.BasicCalculator` supports all operations and commands specified in the appendix of the Requirements document, except the fix operation.
- `oose2.ex2.view.lui.LuiBasic` supports console stdin/stdout interaction with the `BasicCalculator`, and implements the fix operation.
- `oose2.ex2.view.gui.GuiBasic` supports Gui interaction with the `BasicCalculator`, and implements the fix operation.

- `oose2.ex2.model.tests.BasicCalculatorTest` provides a test harness for the operations provided by `BasicCalculator`

Any new calculator must provide three classes, each an extension of one of these base classes. If the name of the new calculator is "Extended", then the following naming convention must be followed:

- `oose2.ex2.model.ExtendedCalculator` **extends** `BasicCalculator`
- `oose2.ex2.view.LuiExtended` **extends** `LuiBasic`
- `oose2.ex2.view.GuiExtended` **extends** `GuiBasic`
- `oose2.ex2.model.tests.ExtendedCalculatorTest`

1.2 Calculator Factory

In order to enable support for multiple user interfaces by a particular calculator, the framework adopts the view and controller portion of the model-view-controller pattern - i.e. the presentation and interaction with the user is separated from the domain logic (in this case, from the RPN calculator logic).

Since we anticipate supporting a number of extended calculators in our product range, we have adopted the use of an abstract factory `oose2.ex2.view.CalculatorFactory` for manufacturing matching view and model components for each particular calculator, thus permitting the use of a single `main()` method entry point, found in the `oose2.ex2.Main` class, for all calculators.

The factory consists of a two static methods:

- `create(CalculatorView,CalculatorModel)`
- `create(String,String)`

The first create method takes two enumeration values as arguments, `CalculatorView` (available values are `LINE` and `GRAPHICAL`) and CalculatorModel (available values are `BASIC` and `EXTENDED`). Any combinations off these values will produce a GCF calculator view conforming to the `oose2.ex2.CalcUI` interface.

The second create method takes String equivalent values of the two enumeration types' values (legal arguments are currently "Line" or "Graphical" and"Basic" or "Extended") . This is a convenience for passing arguments directly from the command line. Note that the abstract factory uses the constructors in the framework classes directly, rather than delegating its work to concrete factories, a slight variation from the normal abstract factory pattern.

2 Adding a new Calculator

As indicated above, one must provide three class definitions for a new extended calculator, and modify the factory class to enable instantiation of an instance of this calculator. An example calculator, named "Extended", is provided with the framework to show an example of how this is done. The factory class already supports this extended calculator. The following describes how to go about creating these extended classes, using the `ExtendedCalculator` from the distribution as examples.

- `ExtendedCalculator` **extends** `BasicCalculator`
  - **public** `ExtendedCalculator()`
    This constructor simply delegates to the super class.
  - **public void** `celc()`
  - **public void** `fahr()`
  - **public void** `swap()`
    A new public method for each of these new supported commands must be provided.
- `LuiExtended` **extends** `LuiBasic`
  - **public** `LuiExtended(ExtendedCalculator)`
    Passes an extend calculator to the super class's constructor and assigns the extended calculator to a field in the sub class for access by new operations.
  - **public void** `displayError(String)`
    Overrides the same method in the super class to indicate that an error occured in an instance of the sub class.
  - **public void** `attemptCommand(String)`
    Overrides the same method in the super class to process extended commands. If the command is not processed by the sub class then the command should be delegated to the super class.
- `oose2.ex2.view.GuiExtended` **extends** `GuiBasic`
  - **public** `GuiExtended(ExtendedCalculator)`
    Passes an extend calculator to the super class's constructor and assigns the extended calculator to a field in the sub class for access by new operations.
  - **public void** `initialiseUI()`
    Overrides the same method in the super class to create additional buttons and provide those buttons to the super class via the `setExtra(JPanel)` method.
  - `processOperation(String cmd)`
    Overrides the same method in the super class to process extended operations. If the operation is not processed by the sub class then it should be delegated to the super class.
- `oose2.ex2.model.tests.ExtendedCalculatorTest` **extends** `BasicCalculatorTest`
  - **public void** `setUp()`
    Overrides the `BasicCalculator.setUp()` method so that an extended calculator is tested by the test harness. All the test methods of both `BasicCalculator` and `ExtendedCalculator` will be run against the target calculator.

3 Modifying `CalculatorFactory`

The general logic for the static `create(CalculatorView,CalculatorModel)` method in this class is as follows (in pseudo code):

```
if (model == BASIC){
  create BasicCalculator;
  if (view == LINE)
    create LuiBasic(BasicCalculator);
  else if (view == GRAPHICAL)
    create GuiBasic(BasicCalculator);
}else if (model == EXTENDED)
  create ExtendedCalculator;
  if (view == LINE)
    create LuiExtended(ExtendedCalculator);
  else if (view == GRAPHICAL)
    create GuiExtended(ExtendedCalculator);

  initialise and return view;
}
```

The enumerations in the factories and the logic should be extended to accomodate the new extended calculator model and view.

The `create(String,String)` method parses the `String` arguments as elements in the appropriate enumerations, before delegation creation the `create(CalculatorView,CalculatorModel)`. The method will throw an exception if the arguments cannot be mappped to enumeration elements. The logic should be extended to accept appropriate string representations of the new extended calculator.