Our goals for today are to experiment with Eclipse's facilities for code navigation and to develop some additional code for the vending package.

Setup

1. Launch Eclipse and use the Package Explorer to re-open the Vend project and the vending package (in the src directory). Load your TestCoinBox program into an editor.

Navigation

2. In the reading for today, the authors suggested several different ways to navigate from any occurrence of an identifier to exact position of its declaration in the file that contains that declaration. (My favorite is to select the identifier by double-left-clicking on it and then to press the F3 key.) Select any occurrence of the identifier depositQuarter in the TestCoinBox program and navigate to its declaration. Describe the visible effect of the navigation operation on the workspace.

3. The Package Explorer window partially displays a hierarchy of Java code in which the nodes nearest the root are packages and the leaf nodes are field and method declarations. You can expose the children of any visible internal node in this hierarchy by clicking on the rightwards-pointing triangle adjacent to its name, or hide an entire subtree by clicking on the downwards-pointing triangle adjacent to the name at the root of the subtree. Alternatively, you can open up an entire branch of the hierarchy by selecting the name of a leaf node — a field or method declaration — and then right-clicking in the editor to pop up a long menu and selecting Show In ⊲ Package Explorer from that menu. Use this technique to expose the Package Explorer entry for the dimes field of the CoinBox class.

4. Eclipse has access to the entire set of library classes in the Open Java Development Kit. The navigation facilities make it quite easy to examine the source code for any classes that you are interested in.

   You can search for any class by name by selecting Navigation ⊲ Go To ⊲ Type from the menu bar. This pops a window with the title Go to Type. In the text field at the top of that window, you can type the name of a class or a regular expression that might match a class, and Eclipse will display a range of possibilities in the text area below. When it finds the right one, you can select it from the list by double-left-clicking on it. The Package Explorer will open up the hierarchy to expose that class and pre-select it for you.

   Use this tool to find the InputStream class. Expand the entry for that class in the Package Explorer to expose its fields and methods and open an editor to examine the source code for the skip method.

More on Vending Machines

5. With the definitions of the CoinBox and Stockpile classes from the eclipse-lab.pdf, we’re ready to start work on the main class for the vending package, namely VendingMachine. A minimal VendingMachine object needs at least one Stockpile of items to sell (we’ll start with just one, to keep things simple at first) and two CoinBoxes, one to hold on to the customer’s money as she deposits it, one coin at a time, and the other to hold the receipts from previous sales. We’ll transfer money from one CoinBox to the other when a sale is completed.
In addition, we’ll need three `Switch` objects, to control the lights on warning panels on the exterior of the vending machine: “Sold Out” (to be lit if and only if the Stockpile is empty); “Exact Change Only” (to be lit when the number of coins in the second CoinBox is insufficient to make change for the next customer); and “Deposit More Money” (to be lit when the customer has pressed the “purchase item” button prematurely and turned off when another coin is deposited or the “coin return” lever is pressed).

Write the class definition for this minimal `VendingMachine` class. Provide methods that the hardware user interface (not implemented here) could call when the customer inserts a quarter, or a dime, or a nickel, depresses the coin-return lever or pushes the button to purchase an item.

The last method is the trickiest one, since you need to provide different behaviors depending on whether the amount that the customer has deposited so far is less than, equal to, or greater than the purchase price of the stockpiled item.