Chapter 2: Entering the jBoxes World

This chapter will take you into the jBoxes universe and show you around a little, covering most editing details. By the end of the chapter, you will be comfortable operating in this rather strange new world. But, note that you still will not have learned anything about programming.

In a sense this whole booklet is about concepts of jBoxes, but you need to learn some of them right away in order to get started. If this book were teaching you to write a novel, the material in this chapter would be like the part of the book that teaches you how to use your word processing program to read a novel already written by someone else. None of the following material really has anything to do with creating a program—it is all about the mechanics of using the jBoxes software to view and execute a program that someone else has already created. Of course, as you go on to later chapters, you will start learning how to create your own programs.

Note that many of the details covered in this chapter are summarized in the Summary Sheets.

Fundamental Constructs in jBoxes

The jBoxes system was designed to use a minimal number of basic building blocks, or constructs. This section will detail those constructs, and how they impact editing a jBoxes universe.

Now that you have seen some jBoxes diagrams, it is time to talk about the components that make up any jBoxes universe. Fortunately, this is quite simple and elegant—a jBoxes universe is made up entirely of *boxes* and *strings*, together with some extra *decorations*. The basic organizational ideas are *sequencing*—where things are arranged one after the other, and *containment*—where things are located inside other things.

A box consists of its rectangular border together with its contents. A box typically contains some other boxes and some symbols. The collection of boxes that are contained in a box are organized in a sequence, though they are physically arranged in various ways.

Each symbol in a diagram belongs to a sequence of 1 or more symbols which is known as a *string*.

Certain kinds of boxes also contain decorations. These are extra images such as arrows that are used to visually remind you what kind of box you are seeing.

Certain special symbols, known as *marks*, appear in a jBoxes diagram. Marks simply indicate where a string or a list of boxes begins, providing an indication that a sequence of symbols or boxes is expected at that location, even if none are present. Marks can not be removed.

Boxes Have Parts

Every box has its own collection of *parts*, depending on what kind of box it is. One type of part is simply a string. For example, many boxes have a *name* part, which typically is displayed in the upper left corner of the box. Other string parts will be discussed later.

In addition to having string parts, all but a few kinds of boxes have an *inner list* of other boxes. These boxes are always displayed inside the box that owns them. They are arranged in a variety of ways, depending on the kind of owner they have. But, no matter how the inner boxes are arranged for display, they are always stored internally in a sequence, with a first box, a second box, and so on, up to a last box. For some kinds of boxes the first box is at the top, with the second one below it. For other kinds of boxes, the first box is at the left, with the second one to its right. Other kinds of boxes do other specialized arrangements.

Marks indicate where parts begin, even if a string part has no symbols or an inner list has no boxes.

Kinds of Boxes

A jBoxes diagram is made up entirely of boxes of different *kinds*. The kind of a box determines how it looks and how it behaves. You can tell what kind a box is by looking at it—different kinds have different appearances. In a way, your study of programming in the jBoxes universe will be a matter of learning about all the different kinds of boxes, how they behave, and how to build diagrams with them.

The different kinds of boxes are like different kinds of blocks in a set of toy building blocks. As a child, you learned how different pieces fit together and how to build simple structures with the various pieces. Here, throughout this booklet, you will learn how each kind of box looks and behaves as you learn the corresponding concepts of programming.

Navigation

In a word processing program the "cursor" is a crucial concept. When you press certain keys, such as the $\boxed{a}$ key, the symbol for that key is inserted at the cursor position. When you hit the backspace key, the symbol to the left of the cursor is removed. When you hit certain other keys, like the arrow keys, the cursor moves in the document. And, you can click the mouse anywhere in the document and move the cursor to that position.

jBoxes has a similar concept known as the *focus* (short for "focus of attention") which indicates the point in a diagram where interactive requests are targeted. The focus is a box or symbol that is displayed with a special yellow background. Like with the cursor in a word processor, newly created entities are inserted near the focus, the focus box or symbol is removed when certain keys are hit, and the focus can be changed by hitting certain keys or by using the mouse.

The term "navigation" is used to refer to the process of moving the focus from one entity to another.

The key to navigation by keystrokes is to understand that a typical box belongs to the inner list of some other box, and itself has an inner list. Moving the focus from a box to its inner list or a string part is known as moving "in." Similarly, to move the focus from a part in a box to the box itself is known as moving "out." Further, if the focus is on a box or symbol, the focus can be moved to the next or previous box or symbol, because symbols in strings and boxes in an inner list are arranged in a sequential order. Finally, to move the focus from one part of a box to another is known as "jumping" the focus.

These ideas are quite natural and can be accomplished by various keystrokes. To move in, move out, move previous, move next, or to jump, you can hit, respectively, $\boxed{\text{ctrl}}\boxed{\text{i}}$, $\boxed{\text{ctrl}}\boxed{\text{o}}$, $\boxed{\text{ctrl}}\boxed{\text{p}}$, $\boxed{\text{ctrl}}\boxed{\text{n}}$, or $\boxed{\text{ctrl}}\boxed{\text{j}}$. These keys are very easy to remember because they are simply the first letter in the name of the desired motion.

These keys are easy to remember, but they require holding down the $\boxed{\text{ctrl}}$ key while hitting another key, which is somewhat irritating. So, jBoxes has alternative keys, namely the $\boxed{\Longleftarrow}$ key to move previous, the $\boxed{\Longrightarrow}$ key to move next, the $\boxed{\Downarrow}$ to move in, and the $\boxed{\Uparrow}$ key to move out.

Note that moving "in" is not well-defined, because a box can have several string parts in addition to an inner list of boxes and, even worse, the inner list might have a number of boxes in it. So, when you move in, the focus will simply go to the mark, symbol, or box which was most recently the focus in the box. Often the box will not have ever had the focus inside it, in which case jBoxes decides on a reasonable place to put the focus.

Keystrokes are often the most efficient way to move the focus, especially when you need to move it to a nearby location. The mouse can also be used, as you know, to move the focus, by simply putting the mouse cursor over the desired entity and left-clicking.

Exercise:

Start jBoxes on the universe `counting.box` from Chapter 1. Observe how various boxes have parts, which are strings, inner lists of boxes, and decorations. Experiment with navigation, moving in, out, previous, and next using both `ctrl` keys and arrow keys (including $\boxed{\text{del}}$ and $\boxed{\text{ins}}$.

Aspects

jBoxes allows you to control how much detail about itself a box displays. The particular parts that a box is displaying at a given time is known as the *aspect* the box is showing.

So far all you have seen is boxes showing their *full* aspects, with black used as the drawing color. A box showing its full aspect displays its inner list of boxes and most of its string parts. When a box is showing its *abstract* aspect, it only shows its name part, using blue as the drawing color. Many kinds of boxes have a documentation part, which is a string written by the programmer to describe the box. Some kinds of boxes can also show a *value* aspect, which shows the current value string of the box, using red for the drawing color.

Aspects are useful for hiding information. You might think it strange that you would want to hide things from yourself, but this is actually an important idea in computer science. The full aspect of a box is used when you want to see the details of a box. But, when you only want to see that the box exists, the abstract aspect is useful, and when you only want to see that the box exists and be able to read a brief description of it, the documentation aspect is appropriate.

In the jBoxes context, using the abstract or documentation aspects helps to make the diagram you are viewing appear to be simpler by hiding a lot of details. Also, it turns out that using different aspects can control the amount of detail that is seen during execution.

You can change the aspect a particular box is showing by right-clicking on the box, which also makes it the focus. Or, you can make the box the focus by any desired method, and then press the $\boxed{\text{ctrl}}\boxed{\text{c}}$ key to “change” the aspect that is shown. Also, you can use the misleadingly labeled $\boxed{\text{delete}}$ key to do the same thing (this key is used as an easy-to-hit replacement, just as the $\boxed{\text{insert}}$ key is used as a replacement for $\boxed{\text{ctrl}}\boxed{\text{j}}$, simply because it is near the arrow keys on the typical keyboard).

Note that each time you right-click on a box or hit either of the “change aspect” keys, the aspect being shown cycles from the current one to the next one. Most kinds of boxes have several aspects which are cycled through one after the other.

Exercise:

Working in the `counting.box` universe, experiment with changing aspects using both possible keys and the mouse. Use split-screen viewing and step through the program with $\boxed{\text{F1}}$ with the second inner box in the fourth inner box in the box named `counting` changed to show its documentation aspect. Try changing the box named `counting` to show its abstract aspect and hit $\boxed{\text{F1}}$ again. Try changing back to the original aspects and step through execution once again, this time with the `stack` box changed to show its abstract aspect.

Viewers

Note that some of this material has already been mentioned.

As you have seen, a jBoxes universe is a diagram of nested boxes, each box having some strings and other boxes inside it. When the software is running, it displays one or more rectangular portions of the full diagram in a rectangular region of the window known as a *viewer*. A viewer is like a camera, aimed at some part of the jBoxes diagram and showing it in the window. Usually one viewer owns the entire window, but you can use a split-screen mode where three viewers share the window.

The main viewer, whether in full-screen or split-screen mode, is the one that has the focus in it and responds to interactive requests from the keyboard or mouse. The border of the main viewer indicates the current operating state of the system. A red border means that no program is running and that the system is waiting for interactive commands. When a program is running without waiting for interactive commands, the border is green. When program execution has paused and is waiting for an interactive command, the border is yellow. Finally, when a program is executing and is waiting for input from the keyboard, the border is gray.

The border color for any auxiliary viewers is blue. These only show up in split-screen mode, where the main viewer uses the left part of the window and the right part is split, with the upper portion dedicated to the console viewer that is fixed over the console box, and the lower portion permanently aimed at the ports box.

Various function keys can be pressed to change the viewer setup. If you press $\boxed{\text{F7}}$, the system switches to using the whole window for the so-called user viewer, which just shows the console box. If you press $\boxed{\text{F7}}$ again, the system switches back to normal viewing. Similarly, the $\boxed{\text{F8}}$ key toggles between using full-screen and split-screen viewing.

You can press $\boxed{\text{F9}}$ and $\boxed{\text{F10}}$ to make the size of things in the diagram, respectively, smaller and larger. This may be useful depending on the keenness of your eyesight, the size of your display device, and the universe you are viewing. Using a smaller scale puts more of the universe on display in a viewer, but it makes everything smaller and thus harder to see.

The keys $\boxed{\text{PageUp}}$, $\boxed{\text{PageDown}}$, $\boxed{\text{Home}}$, and $\boxed{\text{End}}$ shift the main viewer, respectively, up, down, left, and right over the jBoxes diagram, allowing you to see different areas of the universe. When you hit a key that affects the focus, the viewer is always shifted so that the focus is visible in the view region, so you can also shift the viewer by moving the focus. You can also drag the universe with the mouse.

Controlling Program Execution

In the previous chapter you saw two ways to execute a program. Those two ways lie on opposite ends of the "detail spectrum." The first way was to press $\boxed{\text{F1}}$ once for every step of execution. The second way was to use $\boxed{\text{F7}}$ to switch to "user viewing," hit $\boxed{\text{F1}}$ once to start things up, and have the program execute fully with no further controlling keys.

Now that you know about aspects, you can be told the full truth, and learn some further techniques for controlling the amount of detail that you see when a program executes. The rule is simple: jBoxes pauses during execution and shows the way the universe currently looks whenever a *visible* change has been made, except for changes to the console box. So, when all the boxes are showing their full aspects, you have to press $\boxed{\text{F1}}$ for every step of execution, because every step produces some visible result. But, when you are using the user viewer, only the console box is visible, so you only have to press $\boxed{\text{F1}}$ once to start things off, and after that jBoxes runs without pausing because none of the changes to the various boxes during execution are visible.

Following this rule, you can control how much detail you see during execution by changing some boxes away from their full aspects so that changes to them are not visible.

This technique of controlling the level of detail seen during program execution by using aspects is useful, but sometimes, typically during debugging, you want execution to run to some point in your instructions without showing any details and then stop and let you see all the details from that point on. jBoxes lets you accomplish this as follows. First, you can specify a box as a "breakpoint" by hitting $\boxed{\text{F3}}$ when the focus is on the box. The box will gain an extra purple border indicating that it is a breakpoint. Then you can start execution as usual, but at some point, you can press $\boxed{\text{F2}}$ to start a special execution mode that executes without showing any details until the next breakpoint is reached. Once the execution process reaches the next breakpoint, you can press $\boxed{\text{F1}}$ to see all detail for a while, and then later press $\boxed{\text{F2}}$ to again execute "silently" until the next breakpoint is hit.

The key $\boxed{\text{F3}}$ is a "toggle" in the sense that each time you hit it, the focus box switches state, either from not being a breakpoint to being one, or from being a breakpoint to not being one.

Exercise:
Start jBoxes on the universe `ex1.box` and explore the use of $\boxed{\text{F2}}$ and $\boxed{\text{F3}}$.

Creation and Destruction

The most important editing task, of course, is creation of new material and removal of undesired material.

Creation of new material is in principle very simple to do, but requires some understanding of the context in which the focus is located.

Just like in a "normal" word processor, if the focus is on a symbol in a string, or on a mark at the beginning of a string, then pressing any of the 95 standard keys (letters, digits, special symbols, but not control keys, function keys, or any keys not on the main keyboard region) causes that symbol to be added to the sequence of symbols, immediately after the symbol with the focus, and the focus moves onto the new symbol. Sometimes only certain symbols are allowed, in which case an inappropriate key press will be ignored.

If the focus is on a box, or a mark at the beginning of an inner list of boxes, then pressing certain keys will cause a corresponding new box to be added to the list of boxes immediately after the box with the focus, and the focus moves to the newly added box.

The keys, if any, that will create a new box after the focus box depend on the kind of the focus box. The details of which keys work for each kind of focus box are given in the summary sheets, and will be explained in the following chapters as needed. For now, note that the $\boxed{\text{space}}$ key works to create a new box in lots of different contexts.

One very important kind of box is the *empty box*. When the focus is on an empty box and certain keys are hit, instead of creating a new box to be added after the focus, the focus itself is "filled in" with the desired kind of box.

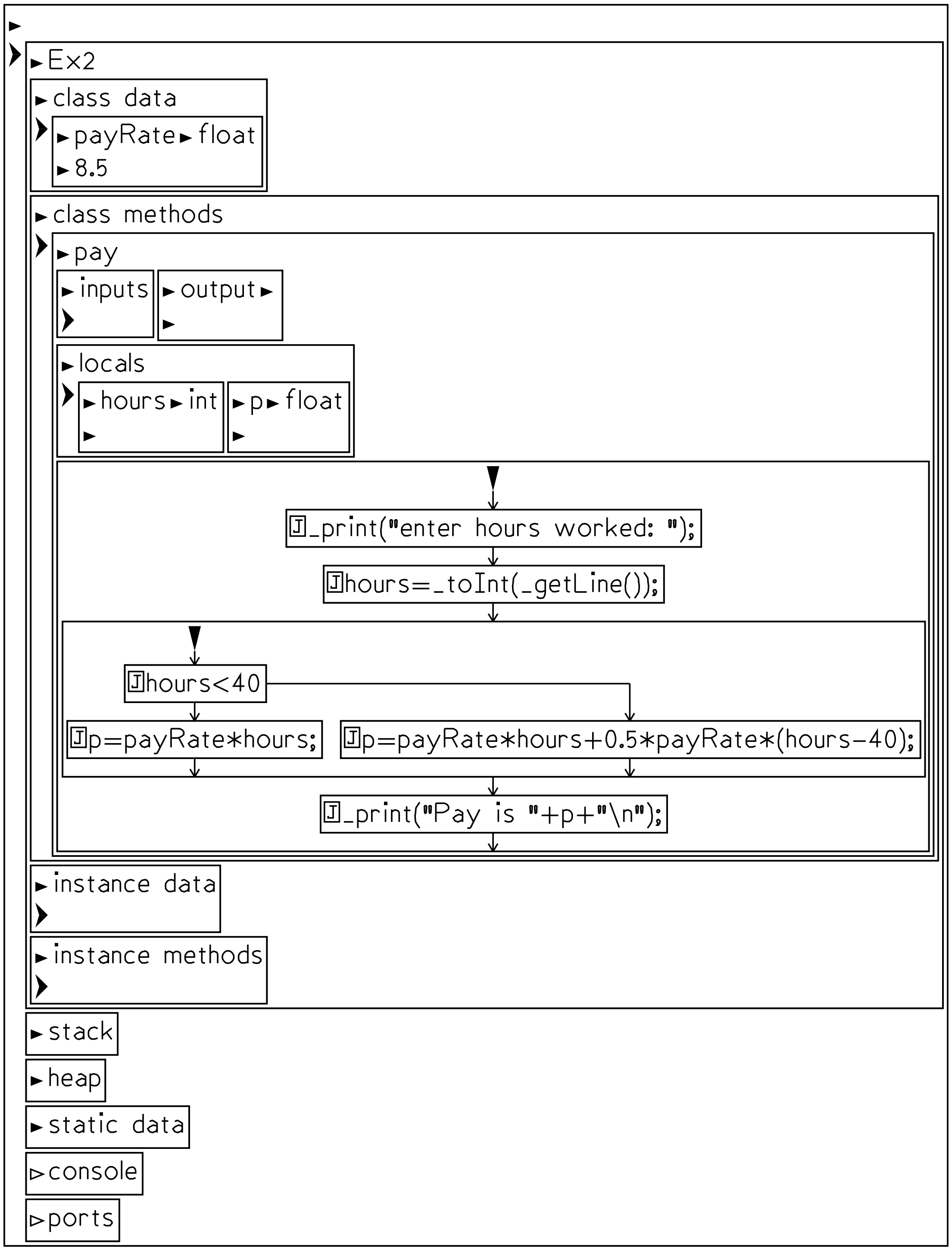
Removing a symbol or box is even simpler. If you hit $\boxed{\text{ctrl}}\boxed{\text{k}}$ or $\boxed{\text{backspace}}$, the focus box or symbol is removed, and the focus moves to the next box or symbol, or to the previous one if the box or symbol that was removed was on the end of its sequence of boxes or symbols.

Also, note that sometimes jBoxes refuses to let you remove a certain box or symbol, because it is a permanent part of the machinery of jBoxes. For example, you can't remove the box named `stack`, and you can't change its name.

Exercise 2 Officially due by the start of the next class period (but won't be penalized if turned in up to one week later). Turn in your answer by emailing the instructor at `shultzj@mscd.edu`.

Start jBoxes and enter `ex2` as the name of the universe to work on. Since that universe doesn't exist, a new universe will be started.

Using the chart in the Summary Sheets entitled "Creation Using Java Box Approach" and the information on editing in this chapter, create the diagram shown below in your universe:



```
_print("enter hours worked: ");
hours=_toInt(_getLine());
hours<40
p=payRate*hours;
p=payRate*hours+0.5*payRate*(hours-40);
_print("Pay is "+p+"\n");
```

When you think you have the correct diagram created, run the method named `pay` several times and verify that it works correctly. It is supposed to ask the user for the number of hours worked and report the pay, given that the person makes $8.50 per hour with time and a half for overtime. Test it on a person working 30 hours, and again for a person working 50 hours.

Email me with your file `ex2.box` as an *attachment*.