CS1567

Intermediate Programming and System Design Using a Mobile Robot

Lab 0: Java Warm-up

In this course, we will be programming in Java. As an object-oriented programming language, Java is similar in many ways to C++, so you will be able to start using it very quickly, and then learn as you go. Your programs will be developed using the Java Developers Kit (JDK) from Sun Microsystems and an Integrated Development Environment (IDE) called Jbuilder from Borland. The Jbuilder IDE facilitates the construction of GUI's for your systems, and shares many similarities with other Visual IDE's you may have used, such as Visual Basic, Visual C, etc.

> Probably the most important feature of Java is that it is designed for portability: you can design and build a single program that will run on a wide variety of platforms. This is achieved in a straightforward fashion: when you write Java code, it is compiled into an intermediate language called Java Byte Code. Each operating system then provides its own interpreter for byte code, called a Java Virtual Machine.
>
> Even better, there are versions of the Jbuilder IDE and the JDK tools available ***for free*** on the web for Linux, Windows, Solaris platforms. To get a head start on this assignment, and to help you out throughout the semester, visit the Borland Web site and download a personal copy of Jbuilder for your home machine. You will find a link to Borland on the class web page.

This lab consists of two parts. Part one is simply to run through the JBuilder tutorial that is incorporated into the *welcome* project in J**B**uilder. This will give you a basic understanding of how to get around in JBuilder. Then in Part two, you will work your way through a step-by-step illustration of building your own, tiny and frankly pretty useless, Java project. By going through it carefully, however, you will be introduced to many aspects of the language and the programming environment, and this will make it easier for you to do the later, more interesting labs on the robots.

PART 1:

1. Start JBuilder. This may take a while, especially if this is the first time Visual Cafe' has run on your machine. When Jbuilder has opened, you should find yourself in the *Welcome* project. You will know this because the IDE shows the current project name in the upper left, just over the class hierarchy window. If you are in the welcome project, skip to step three.
2. If you are not in the welcome project, select File/Close Project from the menu bar, close whatever project you are in, if any, and then open the Welcome project. There are two ways to open the welcome project. One way is to find it in the File/Recent Projects/ menu. If it's not ther ask the lab assistant to help you locate the path and open it will the File/Open Project dialog.

3. The quickest way to learn your around is to start running the tour. Click on the arrow and go to the first screen. This diagram is a little busy but it explains the various panes that will appear from time to time within the Juilder IDE window. For now, you are looking at a window with two panes. The pane to the left is the project pane. It tracks the file hierarchy within the project. The panel to the right is generically called the *content* pane. It will show various views of applications within the project with different views being selected by the file view tabs at the bottom. Click *home*. This will return you to a page of html associated with the project. From this page you can choose various introductory options such as the tour that you just started, a tutorial, or a view of segments of sample core. Continue taking the tour for now. Don't worry if not everything makes sense at this point. Try to pick up what you can. Finally, click on the tutorial and run though it. This will walk you though a simple "hello world" style application just to get your feet wet.

PART 2: Now we are ready to start up an original project of our own and to pay a little more attention to the details as we put together a simple calculator application.

1. Close the welcome project and any others that you created in the tutorial. Use File/Close Projects, select ALL in the dialog box, then OK. The project pane should grey out the content pane will also go blank.

> Files within your project directories can have several different extensions. The most important ones to know about for now are:
> - .jpr files contain various kinds of information about your project, and are used by. If you double-click a .jpr file from your desktop, the project with which it is associated will be opened.
> - .java files contain Java source-code
> - .class files contain Java Byte Code (compiled Java)
>
> Java has special requirements concerning the naming of files. These requirements are intended to make directory and file naming more rational, by associating the names given to files and directories in the operating system (and even the entire network) with the names given to objects in a program. The basic rule is that each class must be stored in its own file, which has the same name as the class. There's a similar rule for packages: every package must be stored in its own directory (folder), which has the same name as the package.

2. Select **File/New** (don't use New Project for now, we will get to the same place, but I want you to see something first). The dialog box will give you three options, ***Project, Application, or Applet***. Chose project to create a new project. This will take you though a two panel wizard (you already did this in the tutorial) that creates the directory structure and path for your project. Note that this only creates the directory. Source files will come next when we create an application within the project.

> There are several main types of Java projects, of which the most important are Applets, Applications, and Beans. Applets are programs that run within a browser, typically on a client machine. To ensure security when they are run on clients, there are restrictions on what Applets can do. Applications are stand-alone programs that run on a local machine, and thus don't have these security restrictions. A Bean is a reusable component that complies with a set of standards ("the JavaBean standards") to facilitate interoperability across systems and platforms.

3. After you finish with the new project wizard. Select File/New again. This time select a new ***application.*** In the next dialog boxes, You will be ask to name the package and classname for the class that will invoke your application. A package is a collection of class files that make up an application. As you add new classes they will be associated by this package name. The classname is the name of the class that will invoke you r application. Name it calculator and click *next*.

4. Your application will consist of one or more frames. A frame is a single window on the desktop and can be created in various default forms within the IDE. Your calculator application will start with a single frame. Name it calculator Frame. The title field is printed in the top border. Enter something appropriate here. We won't need all the menu bars etc for this application (you saw those in the tutorial) so leave all but the "center" selection unchecked. Click Finish.

5. Note that two new .java files have been added to your project pane. These are source files for the calculator and calculator frame class that have been automatically generated for your application. It is essentially a blank application dialog template. Just for fun select Project/Run and see what you have so far.

6. Returning now to Jbuilder, we will customize this application in two steps, first we will build the interface and second we will add the specific java code for each function. To design the interface, click on the ***Design*** tab at the bottom of the content pane. A blank grey square corresponding to the area of your dialog box is displayed. Also, a properties pane appears to the right. You will create the interface by dragging components on to this area and customizing the properties of each as you go.

> The windowing toolkit that you will use is part of the Java Foundation Classes (JFC). The JFC provides what are called "Swing" components. Swing components are not yet fully supported by some browsers, That's ok for us, but in some applet based projects you may wish to alternatively use the AWT windowing toolkit instead.

7. Below the Project Pane, you'll see the structure pane. This pane shows the objects currently instantiated in your application and the hierarchy of objects that create them. For now, click on the object labeled ***content_pane,*** this is the grey area that we are currently manipulating.

8. Now look over to the right at the properties pane. Lets change a few of the properties of our content pane. First, find the border property. Change it from ***none*** to ***line*** note the change in the display. Experiment with some other border styles. Try changing the color. The most important property that you must change at this point is the ***Layout.*** This controls whether the objects that you place in the pane will be arranged automatically or according to a specific pattern. In this case we want to use fully manual layout, so change the Layout property to null.

> A note on terminology: We'll use the terms *object* and *instance* interchangeably, and we'll also use the terms *method* and *function* interchangeably. You should know these terms from C++. Another fundamental term we'll use is *package*. If you've programmed in Lisp or some other languages, you'll know about packages already. Otherwise, you can think of a package as very similar to a C++ library: it's a collection of related files that are grouped together.

9. You're finally ready to design a GUI for your calculator. In the design window, locate the component palette directly above the content pane. We will use the first two tabs on this pallete to select and add swing components. Although we will not use this in this tutorial, look briefly at the ***swing containers*** tab. You would use these components to define sub-regions (individual panes) within the content pane. This would group components both for layout control and to associating functionality in the software. Go back to the ***swing*** tab. Locate and add a button and three textfields by first clicking on the palette and then clicking the location in the pane to deposit the component. Once layed out, the components can be then be moved, resized, cut, or pasted.

10. Now select each of the components in turn and use the properties pane to give them reasonable names: name the button AddButton, and name the text fields Addend1, Addend2, and Sum. Also change the label on AddButton to +. Finally, adjust the size, shape and color of the objects as you see fit.

11. Again open the source code, and see what's been added by Visual Cafe'. Look in particular at the line

    JButton addbutton = new JButton();

    You can learn how to create a new object in Java from this statement.

> Java separates variable declaration from memory allocation for objects. If you specify
>
> JButton AddButton;
>
> Java will reserve a variable named AddButton of the type java.awt.Button, but it will not yet allocate space for on object. Memory will be allocated only when you further specify
>
> AddButton = new JButton( );
>
> You can combine declaration and allocation into a single instruction, e.g.,
>
> JButton AddButton = JButton( );
>
> As you've probably already figured out, JButton( ) is the default constructor function for the JButton class. You can have alternative constructor functions that take parameters, but each constructor function for a class must have a distinct signature.

12. Select Project/Run to execute the project you've built so far---but remember it's just a GUI that doesn't do anything yet. If there were any syntactic errors, they'd show up in the message window, but since all you've got so far is automatically generated code, you shouldn't see any errors. Click the close button (X) to close the application.

. . .

13. Now you'll finally write the code to make addition occur when the AddButton is clicked. Jbuilder will help you again, this time by writing a stub for the new method you want to create. Go to the properties pane and click on the events tab at the bottom. Now select the "+" button. At the top of the events list. The properties pane now has a list of all the possible events for this button. The fields to the right can be filled in with the names of methods to be executed in case of this event. In our case, we only care about the first, ***action performed*** . Click on the right hand field and a default method for handling mouse clicks will be inserted. Now look at your source code: JBuilder has added an empty AddButton_ActionPerformed method, as well as a new line in the JBInit code to set up a listener for this event.

> As an object-oriented language, Java allows the user to assign various access levels to variables, methods, and classes. The simplest ones, which you should know about now are these:
> - public: visible everywhere.
> - default (no modifier preceding class): visible only in the variable, method, or class's own package.
> - private: visible only in the variable, method, or class's own class.

14. It's pretty obvious what our new method, AddButton_ActionPerformed, should do. It should take the numbers that are entered into each Addend text field, add them together, and show the sum in the Sum text field. Before you write the code for this, there's a little about types in Java that you should know.

> There are only eight primitive (i.e., built-in) types in Java. These are:
> - boolean
> - int, long, byte, short
> - double, float
> - char
>
> Java specifies the exact length in bytes for each primitive type; you can find out more about this by looking at any Java text or reference book.
>
> Java handles primitive-type variables differently than it handles real, first-class objects. Remember how declaration and allocation are separated for objects? This separation does not occur for primitive variables. The instruction
>
> int i;
>
> will both declare a new variable i and allocate memory for it.
>
> Do not confuse ints with the members of the Integer class. The Integer class, with a capital "I", is not primitive. It contains a number of useful methods.
>
> Strings in Java are also non-primitive objects (of the class String). However, because they are so frequently used, Java will also perform automatic memory allocation when you declare an object of type String.

15. Add the variable declarations that you'll need for the method AddButton_ActionPerformed You'll need three int variables: one for each of the addends, and one of the sum; and you'll need two String variables, because the text that gets input to the text fields will be String objects. Your code should look like this:

```
void AddButton_ ActionPerformed(ActionEvent e)
{
int sumval, add1, add2;
String input1, input2;
}
```

16. The methods for inputting and outputting text to a text field are getText and setText. You might have guessed this, but how can you find out more about the definitions of these methods? JBuilder will help you. Type

Addend1.

in your source code, somewhere inside the AddButton_ActionPerformed method. You'll see a list of all the methods for the JtextField class (of which Addend1 is an object) Scroll down and click on the getText method. Don't worry too much if this step seems a little mysterious. Visual Café' provides you with lots of different tools for navigating through the Java class hierarchy, and this step is just meant to give you an illustration of one such tool. In two more steps, you'll see another such tool.

17. Now you know how to write the instructions to read the input from your calculator's GUI . Your code should now read:

```
void jButton1_actionPerformed(ActionEvent e) {
   int sumval, add1, add2;
   String  input1,input2;
   input1 = Addend1.getText();
   input1 = Addend2.getText();

}
```

18. To add input1 and input2 together, you're going to need to convert them to integers. To try and see how you might do the conversion, select Help/Java Reference. Scroll down the table of packages to ***java.lang*** and select it. Now scroll down the class summary table until you see String and select it. This will provide you with lots of documentation about the String class, including a complete list of its public members and methods. You may have to scroll down to see these. For more details on a specific method, select it.

    Take a moment to look through the members of the String class. There's one that converts ints to Strings---valueOf(int I)---but unfortunately, you won't find one that converts a String to an integer. Try a different tack. Look at the Integer class (not int) in the API Reference. There you'll find a method called parseInt(String s), which takes String s and parses in into a int value, which it returns. That's just what you need.

19. You can now add parseInt to your code. But first, little review of some object-oriented programming fundamentals is in order.

You'll see in the definition of parseInt that it is not only a public method, but also a static one. A method is static when it relates to the whole class, rather than an individual object in the class. (It may be easier to think first about static data members. A member is static if there's only one for the whole class. So, for example, if you have a Student class for a database containing university records, you may want to have a static member NumberEnrolled.)

When you invoke a non-static method, you invoke it using the name of the object to which it is related , e.g.

```
Addend1.getText( );
```

But when you invoke a static member, you instead use the name of the class to which it is related, e.g.,

```
Integer.parseInt(s);
```

Now modify your code should look like this:

```
void AddButton_ActionPerformed(Actionevent e)
{
   int sumval, add1, add2;
  String input1, input2;

   input1 = Addend1.getText( );
   input2 = Addend2.getText( );
   add1 = Integer.parseInt(input1);
  add2 = Integer.parseInt(input2);

}
```

20. Finally, add the code to actually sum the two integers, and then to display the result. For the latter, use the setText(s String) method:

```
void AddButton_ ActionPerformed(Actionevent e)

{
   int sumval, add1, add2;
  String input1, input2;

   input1 = Addend1.getText( );
   input2 = Addend2.getText( );
   add1 = Integer.parseInt(input1);
  add2 = Integer.parseInt(input2);

  sumval = add1 + add2;
  Sum.setText(String.valueOf(sumval));

}
```

21. Execute your project. (Project/Run) Congratulations, you've built an adding calculator! More importantly, you've reviewed some of the fundamentals of object-oriented programming, learned about how those fundamentals are implemented in the Java, and learned the basics of using Visual Café' to build Java systems. Save your work, by selecting File/Save All. (Be careful: there's a tendency to want to select Save, but that's not sufficient in general.) Now extend your calculator in whatever ways you'd like, to gain additional familiarity with Java and Visual Café'.

After this, it's on to the robots!