Supplier Classes

Consider a small program, with a Driver class to run it, using JFrame, Image & JButton classes

- Driver is a client
- The others are suppliers
- A client class uses what a supplier class gives it

So far, someone else has written the suppliers for your client code to use.

The Image Class

- Has multiple constructors:
  - First creates basic object
  - Second creates object with size and location set
  - Third creates object with size, location, and given image file
  - 3rd constructor and setImage() take name of image file as a String
  - File itself should be in the same folder as src and bin folders in the Eclipse project

An example from last week

Here is code for class Driver

DrawingGizmo is a supplier class — it provides code that helps us solve a problem

What are some other supplier classes we’ve used?

```java
private DrawingGizmo pen;
public static void main(String[] args)
{
    Driver d = new Driver();
    d.drawSquare();
}

// constructor for Driver objects
public Driver()
{
    pen = new DrawingGizmo();
}

private void drawSquare()
{
    pen.draw();
    pen.moveBy(25);
    pen.turnBy(90);
    pen.moveBy(25);
    pen.turnBy(90);
    pen.moveBy(25);
    pen.dontDraw();
}
```
The Driver Class

- The Driver code contains everything we need so far.
- Creates the window and adds image to it.
- To simplify things, we may decide to move things from Driver into a separate supplier.
- A supplier class is one we write to create objects and supply information for us to use in another class.

```java
public class Driver implements ActionListener {
    public static void main(String[] args) {
        Driver d = new Driver();
        d.makeWindow();
    }
    private void makeWindow() {
        // make window and 3 buttons
    }
    public void actionPerformed(ActionEvent e) {
        // call methods based on buttons pressed
    }
    private void actionA() {
        // 1st button: display whale image
    }
    private void actionB() {
        // 2nd button: display spouting whale image
    }
    private void actionC() {
        // 3rd button: remove whale image
    }
}
```

Writing a Whale (Supplier) Class

- Choose New > Class in Eclipse and choose option to make “Constructors from superclass”.
- This gives us a simple shell to fill in with more detail:
  1. Class name
  2. Constructor method (empty at first)
- Note: there is no main() here, as Driver will still handle that part.

```java
public class Whale {
    public Whale() {
        // TODO Auto-generated constructor stub
    }
    // Whale constructor: save Window, make Image
    public Whale(Window w) {
        window = w;
        img = new Image(150, 150, 215, 225);
    }
    // adds the whale image to the window
    public void addWhale() {
        window.remove(img);
        img.setImage("whale.jpg");
        window.add(img);
    }
    // picture of whale spouting
    public void spout() {
        window.remove(img);
        img.setImage("whaleSpout.jpg");
        window.add(img);
    }
    // remove picture of whale
    public void dive() {
        window.remove(img);
    }
}
```

Writing Supplier Classes

- We must decide which variables and methods should belong to which of our classes.
  - E.g., should window variable be part of Driver or Whale?
  - How do we make those decisions?
  - What about the image variable?
- Once we have made these sorts of decisions, we need to decide on access control.
  - Want to make as much private as possible.
  - Only make public what we absolutely need to be able to get access to when running code.
  - This goes double for instance variables.

The New Whale Supplier

- Suppose we decide that the Driver should still make the Window object, but the Whale class will use it, so it can add the Image properly.
- Since Window object is local to Driver, we must share it with supplier class.
- One solution: the Whale() constructor can take the Window as input, and make its own local variable reference it.
  - Thus both the client and the supplier class can work on the very same Window object.
The New Whale Supplier

- Now, we put code for manipulating Image into methods in Whale class
- Because we want to be able to call these methods in Driver, we must make them public.

```java
public class Driver {
    private JFrame window;
    private Image img = new Image();
    private JButton aButton, bButton, cButton;
    private void makeWindow() {
        window = new JFrame();
        ...
    }
}
```

The New Driver Client

- Once we have separated out our supplier code, we have much simpler Driver, with new use hierarchy (in terms of which class declares the variable).

```
public class Driver {
    private void makeWindow() {
        JFrame win = new JFrame();
        ...
    }
}
```

Modularity in Code

- Separating the graphics code for the Whale example into its own class helps organize things and keep the main Driver class simple.
- Another potential advantage is that we can easily re-use the Whale class in multiple programs, or in multiple places in a single program.
- Even within a single program, we could have multiple Whale instances, each with its own Window in which it appears.

```
JFrame win1 = new JFrame();
JFrame win2 = new JFrame();
Whale moby = new Whale( win1 );
Whale shamu = new Whale( win2 );
```

Note: The window is now a local method variable in Driver, since it is not shared with other methods in that class. It is shared with the Whale class via input parameter passing.

Now, whatever moby does appears in the first window, and whatever shamu does appears in the other one.
This Week & Next

- **Meetings this week:**
  - Monday/Wednesday: regular classroom
  - Tuesday/Friday: in the CS Lab (16 Wing)

- **Reading 07:** Ch. 8 due Monday, Nov 18 at 11:59 PM
  - Some participation exercises this time

- **Program 07:** due Thursday November 21 at 11:59 PM

- **Office Hours:** Wing 212
  - Monday/Friday: 2:15 PM–3:15 PM
  - Tuesday: 1:30 PM–2:30 PM
  - Wednesday: 12:05 PM–1:00 PM

- **Thursday tutor hours changed to 4:30-6:30 this week**