Flyweight Pattern

Example

- We want to draw a small folder icon with a name under it for each person in an organization.
- We want two types of icons, one for “is Selected” and one for “not Selected.”
- We can have an icon object for each person, with its own coordinates, name and selected state.
  - Each icon can then draw() itself.
- Let’s code it.
Efficiency Issues

• If this is a large organization, there could be a large number of such icons, but they are actually all the same graphical image.
• Even if we have two icons, one for “is Selected” and one for “not Selected” the number of different icons is small.
• In such a system, having an icon object for each person, with its own coordinates, name and selected state is a waste of resources.

A “better” Approach

• Instead, we’ll create a FolderFactory that returns either the selected or the unselected folder drawing class, but does not create additional instances once one of each has been created.
• Since this is such a simple case, we just create them both at the outset and then return one or the other.

```java
class FolderFactory
{
    Folder unSelected, Selected;
    public FolderFactory()
    {
        Color brown = new Color(0x5f5f1c);
        Selected = new Folder(brown);
        unSelected = new Folder(Color.yellow);
    }
    //-------------------------------
    public Folder getFolder(boolean isSelected)
    {
        if (isSelected)
            return Selected;
        else
            return unSelected;
    }
}
More Complex Cases

- For cases where more instances could exist, the factory could keep a table of the ones it had already created and only create new ones if they weren’t already in the table.

Flyweight Pattern

- The unique thing about using Flyweights in this example is that we pass the coordinates and the name to be drawn into the folder when we draw it.
- These coordinates are the *extrinsic* data that allow us to share the folder objects, and in this case create only two instances.
  - an instance’s *intrinsic* data makes the instance unique, and the *extrinsic* data is passed in as arguments.
The Folder class

- We’ll develop a folder class that simply creates a folder instance with one background color or the other and has a public Draw method that draws the folder at the point you specify.

class Folder extends JPanel
{
    private Color color;
    final int W = 50, H = 30;
    public Folder(Color c)
    {
        color = c;
    }
    //-----------------------------
    public void Draw(Graphics g, int tx, int ty, String name)
    {
        g.setColor(Color.black); //outline
        g.drawRect(tx, ty, W, H);
        g.drawString(name, tx, ty + H+5); //title
        g.setColor(color); //fill rectangle
        g.fillRect(tx+1, ty+1, W-1, H-1);
        g.setColor(Color.lightGray); //bend line
        g.drawLine(tx+1, ty+H-5, tx+W-1, ty+H-5);
        g.setColor(Color.black); //shadow lines
        g.drawLine(tx, ty+H+1, tx+W-1, ty+H+1);
        g.drawLine(tx+W+1, ty, tx+W+1, ty+H);
        g.setColor(Color.white); //highlight lines
        g.drawLine(tx+1, ty+1, tx+W-1, ty+1);
        g.drawLine(tx+1, ty+1, tx+1, ty+W-1);
    }
}
The paint() routine

• To use a Flyweight class like this, your main program must calculate the position of each folder as part of its paint routine and then pass the coordinates to the folder instance.

• This is actually rather common, since you need a different layout depending on the window’s dimensions, and you would not want to have to keep telling each instance where its new location is going to be.

• Hence, we compute the position dynamically during the paint routine.

```java
public void paint(Graphics g)
{
    Folder f;
    String name;
    int j = 0; //count number in row
    int row = Top; //start in upper left
    int x = Left;

    //go through all the names and folders
    for (int i = 0; i < names.size(); i++)
    {
        name = (String)names.elementAt(i);
        if (name.equals(selectedName))
            f = fact.getFolder(true);
        else
            f = fact.getFolder(false);
        //have that folder draw itself at this spot
        f.Draw(g, x, row, name);
        x = x + HSpace; //change to next posn
        j++;
        if (j >= HCount) //reset for next row
        {
            j = 0;
            row += VSpace;
            x = Left;
        }
    }
}
```
Selecting a Folder

• Since we have two folder instances, that we termed selected and unselected, we’d like to be able to select folders by moving the mouse over them.
• In the paint routine, we simply remember the name of the folder which was selected and ask the factory to return a “selected” folder for it.
• We’ll now check for mouse motion at the window level and if the mouse is found to be within a Rectangle, we make that corresponding name the selected name.
• This allows us to just check each name when we redraw and create a selected folder instance where it is needed.

Checking Mouse Coordinates

```java
public void mouseMoved(MouseEvent e) {
    int j = 0;  // count number in row
    int row = Top;  // start in upper left
    int x = Left;

    // go through all the names and folders
    for (int i = 0; i < names.size(); i++)
        if (r.contains(e.getX(), e.getY()))
            selectedName=(String)names.elementAt(i);
            repaint();
    x = x + HSpace;  // change to next posn
    j++;
    if (j >= HCount)  // reset for next row
        j = 0;
        row += VSpace;
        x = Left;
}
```
What is it?

- There are cases in programming where it seems that you need to generate a very large number of small class instances to represent data.
- Sometimes you can greatly reduce the number of different classes that you need to instantiate if you can recognize that the instances are fundamentally the same except for a few parameters.
- If you can move those variables outside the class instance and pass them in as part of a method call, the number of separate instances can be greatly reduced.
- The Flyweight design pattern provides an approach for handling such classes.
- It refers to the instance’s *intrinsic* data that makes the instance unique, and the *extrinsic* data which is passed in as arguments.
- The Flyweight is appropriate for small, fine-grained classes like individual characters or icons on the screen.