Illustrations of Design Patterns

1. The Digital Clock Applet - Example 3.10 in Section 3.7

2. An Animation Applet (Example 5.1) applied to the digital clock. - Illustrates factorization using inheritance.

3. The Bouncing Ball Applet - Example 4.4 in Section 4.6

4. Example 5.2 - More complex factorization and the Template Design Pattern illustrated with the BouncingBall Applet.

5. Example 5.3 - Another application of the Template Design Pattern where the design pattern is a generic plotting algorithm.

The Digital Clock Applet

This applet periodically reads the current time using the Java Calendar class and writes it to the screen in a digital format. (hour:minute:second).

1. **This is an active applet which must keep running. Therefore it extends Applet and implements Runnable.**
2. **The JVM starts the program at the start() method of Applet which creates a thread and uses the start() method of Thread to execute the program (execute the run() method).**
3. **The run() method invokes repaint() which is a method inherited from Component. repaint() invokes paint().**
4. **Paint() paints the screen with a title, uses the methods of Calendar to get the time, and invokes paint(Graphics g) to write the current value of the time in digital format.**

Design Patterns - Examples

```
<!--ex3_10.html-->
<HTML>
 <HEAD>
   <TITLE> Digital Clock Applet </TITLE>
 </HEAD>
<BODY BGCOLOR=white>
 <CENTER>
  <H1> The Digital Clock Applet</H1>
   <P>
    <APPLET CODE=DigitalClock.class
         WIDTH=250 HEIGHT=80>
    </APPLET>
 </CENTER>
</BODY>
</HTML>
```

```
import java.awt.*;
import java.util.Calendar;
public class DigitalClock
  extends java.applet.Applet implements Runnable
{protected Thread clockThread = null;
  protected Font font = new Font("Monospaced", Font.BOLD, 48);
  protected Color color = Color.green;
  public void start()
/* The browser invokes the start() method that DigitalClock inherits from
              Applet. */
   { if (clockThread == null) {
      clockThread = new Thread(this);
      clockThread.start();
/The start() method above is the start() method of Thread. */
    }
  }
  public void stop()
    { clockThread = null;
  }
  /* The stop() method is invoked by the browser when the applet is flushed
                              from the cache. */
```

Design Patterns - Examples

```
public void run() {
   while (Thread.currentThread()  ==  clockThread)
  { /* currentThread is a static method of Thread which returns the instance of
            Thread which is currently active. */
     repaint();
/* repaint() is a method inherited from Component.  repaint() expects there to be a
            paint() method defined in any class in which it is invoked.  The entity
which is painted is the Panel in which the Applet is instantiated. */
       try {
         Thread.currentThread().sleep(1000);
/* sleep() puts the thread  to sleep for 1000 milliseconds.  This thread is awakened
by the JVM after 1000 milliseconds. */
       } catch (InterruptedException e) {}
   }
  }
```

Design Patterns - Examples

```
public void paint(Graphics g)
{ /* The g instance of Graphics is the context in which the Applet is executing.  */
   Calendar calendar = Calendar.getInstance();
/* Note that the instance of Calendar is obtained through getInstance(), not
                          through a constructor. */
   int hour = calendar.get(Calendar.HOUR_OF_DAY);
   int minute = calendar.get(Calendar.MINUTE);
   int second = calendar.get(Calendar.SECOND);
   g.setFont(font);
   g.setColor(color);
   g.drawString(hour + ":" + minute / 10 + minute % 10 +
                   ":" + second / 10 + second % 10, 10, 60);
 }
}
```

Re-Implementation of the DigitalClock class using generic animation capabilities which can be reused in other applets.

What is unique to the DigitalClock class?

Only the thing which it displays!

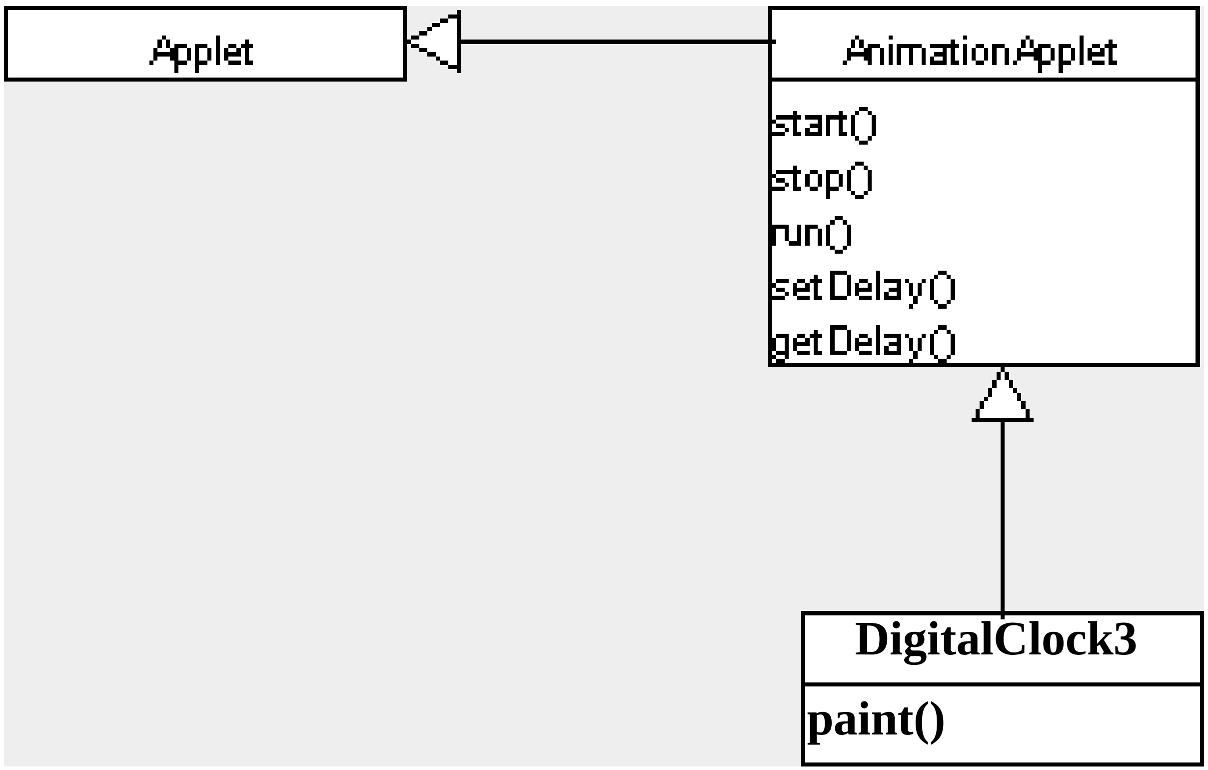
All of the other functionality are common to all animations.

start(), stop(), run(), and delay().

It will turn out to be useful to separate choosing a delay and setting the delay.

Therefore we can create an “AmimationApplet” class which factors out and implements these commonly used functions.

Design Patterns - Examples



Design Patterns - Examples

```
import java.awt.*;
public class AnimationApplet
          extends java.applet.Applet
          implements java.lang.Runnable {
 public void start() {
  animationThread = new Thread(this);
  animationThread.start();
 }
 public void stop() {
  animationThread = null;
 }
 public void run() {
  while (Thread.currentThread() == animationThread)
  {   try
    {  Thread.currentThread().sleep(delay);
    } catch (InterruptedException e) {}
    repaint();
  }
 }
```

Note the use of "this." Any instance of a class which inherits from AnimationApplet will inherit all of the methods directly. For example, as we will see, the start() method defined here is the start() method known to the browser when the applet is loaded.

Design Patterns - Examples

```
final public void setDelay(int delay) {
    this.delay = delay;
  }

  final public int getDelay() {
    return delay;
  }

  protected Thread animationThread;
  protected int delay = 100;
}
```

Design Patterns - Examples

```
import java.awt.*;
import java.util.Calendar;
public class DigitalClock3 extends AnimationApplet
{  public DigitalClock3()
{  setDelay(1000); }
  public void paint(Graphics g) {
    Calendar calendar = Calendar.getInstance();
    int hour = calendar.get(Calendar.HOUR_OF_DAY);
    int minute = calendar.get(Calendar.MINUTE);
    int second = calendar.get(Calendar.SECOND);
    g.setFont(font);
    g.setColor(color);
    g.drawString(hour + ":" + minute / 10 + minute % 10 +
                    ":" + second / 10 + second % 10, 10, 60);
  }
  protected Font font = new Font("Monospaced",
              Font.BOLD, 48);
  protected Color color = Color.green;
}
```

Note that since DigitalClock3 inherits from AnimationApplet that it inherits all of the the methods and instance variables of AnimationApplet. Therefore when this Applet is embedded in a context the start() method of AnimationApplet is invoked.

The Bouncing Ball Applet

A ball is bouncing randomly inside a box. The ball changes directions when it hits a wall of the box. The animation shows the ball moving and reversing its direction when it hits a wall.

We must specify:

The ball - size and color

The container

The position and rate of movement of the ball.

Miscellaneous other stuff.

```
import java.awt.*;
public class BouncingBall  extends java.applet.Applet implements Runnable
{protected Color color = Color.green;
 protected int radius = 20;
 protected int x, y;
 protected int dx = -2, dy = -4;
 protected Image image;
 protected Graphics offscreen;
 protected Dimension d;
 public void init() {
  String att = getParameter("delay");
  if (att != null) {
   delay = Integer.parseInt(att);
  }
  d = getSize();
  x = d.width * 2 / 3;
  y = d.height - radius;
 }
```

```
public void update(Graphics g) {
   // create the off-screen image buffer
   // if it is invoked the first time
   if (image == null) {
     image = createImage(d.width,
d.height);
     offscreen = image.getGraphics();
   }
   // draw the background
   offscreen.setColor(Color.white);
   offscreen.fillRect(0,0,d.width,d.height);
   // adjust the position of the ball
   // reverse the direction if it touches
   // any of the four sides
   if (x < radius || x > d.width - radius) {
     dx  =  -dx;
   }
   if (y < radius || y > d.height - radius) {
     dy  =  -dy;
   }
```

```
   x += dx;
   y += dy;
// draw the ball
   offscreen.setColor(color);
   offscreen.fillOval(x - radius, y -radius,
                      radius * 2, radius * 2);
   // copy the off-screen image to the
screen
   g.drawImage(image, 0, 0, this);
  }
```

Design Patterns - Examples

```
 public void paint(Graphics g) {
   update(g);
  }
  // The animation applet idiom
  protected Thread bouncingThread;
  protected int delay = 100;
  public void start() {
    bouncingThread = new Thread(this);
    bouncingThread.start();
  }
  public void stop() {
    bouncingThread = null;
  }
  public void run() {
    while (Thread.currentThread() == bouncingThread) {
      try {
        Thread.currentThread().sleep(delay);
      } catch  (InterruptedException  e) {}
        repaint();
      }
    }
}
```

Design Pattern - Template

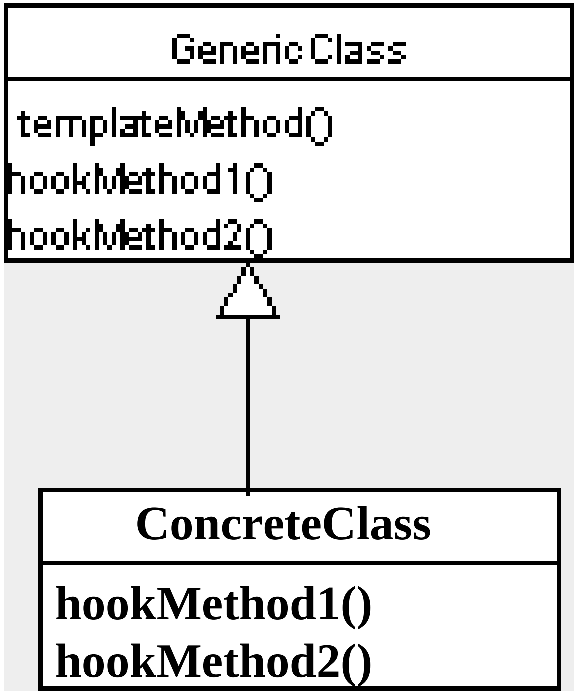
Category -Behavioral Design Pattern

Intent - Define the skeleton of an algorithm in a method but defer some steps of the algorithm to subclasses thus allowing for customization of the implementation of the algorithm.

Applicability -

- **Implement invariant parts of an algorithm once to minimize coding and maintenance efforts across a family of algorithms.**
- **Factorize common behaviors to avoid code duplication across subclasses.**

Design Patterns - Examples



GenericClass (e.g. DBAnimationApplet) defines abstract hook methods (e.g. paitFrame()) that concrete classes (e.g. BouncingBall2) override to implement steps of an algorithm and implements a template method (e.g. update())that defines the skeleton of an algorithm by calling hook methods

ConcreteClass - (e.g. BouncingBall2) implements the hok methods (e.g. paintFrame()) to call out subclass specific steps of the algorithm defined in the template method.

Let us extend the AnimationApplet to implement another solution for other commonly occurring problems.

1. **Screen Flicker and Double Buffering**
2. **Selection of Sizes for Display Area**
3. **Allow for variable parts in common code segments**

In order to accommodate these behaviors we need to redesign the factorization in AnimationApplet.

The redesigned Applet is call DBAnimationApplet where DB is the abbreviation for “Double Buffered.”

“Double Buffered” means that a new image is painted off-screen and moved to the screen when the applet thread is awakened. Therefore the delay caused by generation of the image is not seen by the viewer.

Image Size Management

1. Image size is defined in the HTML and is not known to the applet until it begins execution. (Until init() is executed.)

2. But Applet has a getSize() method inherited from Component.

```
final public void init() {
    d = getSize();
    im = createImage(d.width, d.height);
    offscreen = im.getGraphics();
    initAnimator();
  }

protected void initAnimator() {}
```

The solution has two parts, override init() from Applet and create a new abstract unimplemented method initAnimator() which provides for addition of special requirements of a given applet. The init() method is made final to force use of initAnimator().

Use of Double Buffering

An animation applet should be about to choose whether or not it needs double buffering.

```
final public void
update(Graphics g) {
   if (doubleBuffered) {
     paintFrame(offscreen);
     g.drawImage(im, 0, 0,
this);
   } else {
     // super.update();
   }
```

This is specified by adding a boolean instance variable, doubleBuffered, which is set when an animated applet is initialized.

If double buffering is choosen then the update() method must be implemented in the animated applet.

The part of the update() method which deals with double buffering is the same across all animated applets which use double buffering. Only the painting of the frames is different. This is an example of factorizing common code which is embedded in a method.

```
class ContextA
{   void method()
    {
      <common code segment 1>
      <context specific code A>
      <common code segment 2>
   }
……..
}
```

```
class ContextB
{   void method()
    {
      <common code segment 1>
      <context specific code B>
      <common code segment 2>
   }
……..
}
```

How to factorize this code??

Design Patterns - Examples

The obvious way is: ==>

```
class Common
{   void commonCode1()
  { <common code segment 1>
    void commonCode2()
    { <common code segment 2> }
……..
}
```

```
Class ContextA extends Common
{ void method()
   { commonCode1();
     <context specific code A>
     commonCode2()
    }
…….
}
```

```
Class ContextA extends Common
{ void method()
   { commonCode1();
     <context specific code A>
     commonCode2()
    }
…….
}
```

What are the problems with this code?

Design Patterns - Examples

Let us try again using an abstract method ====>

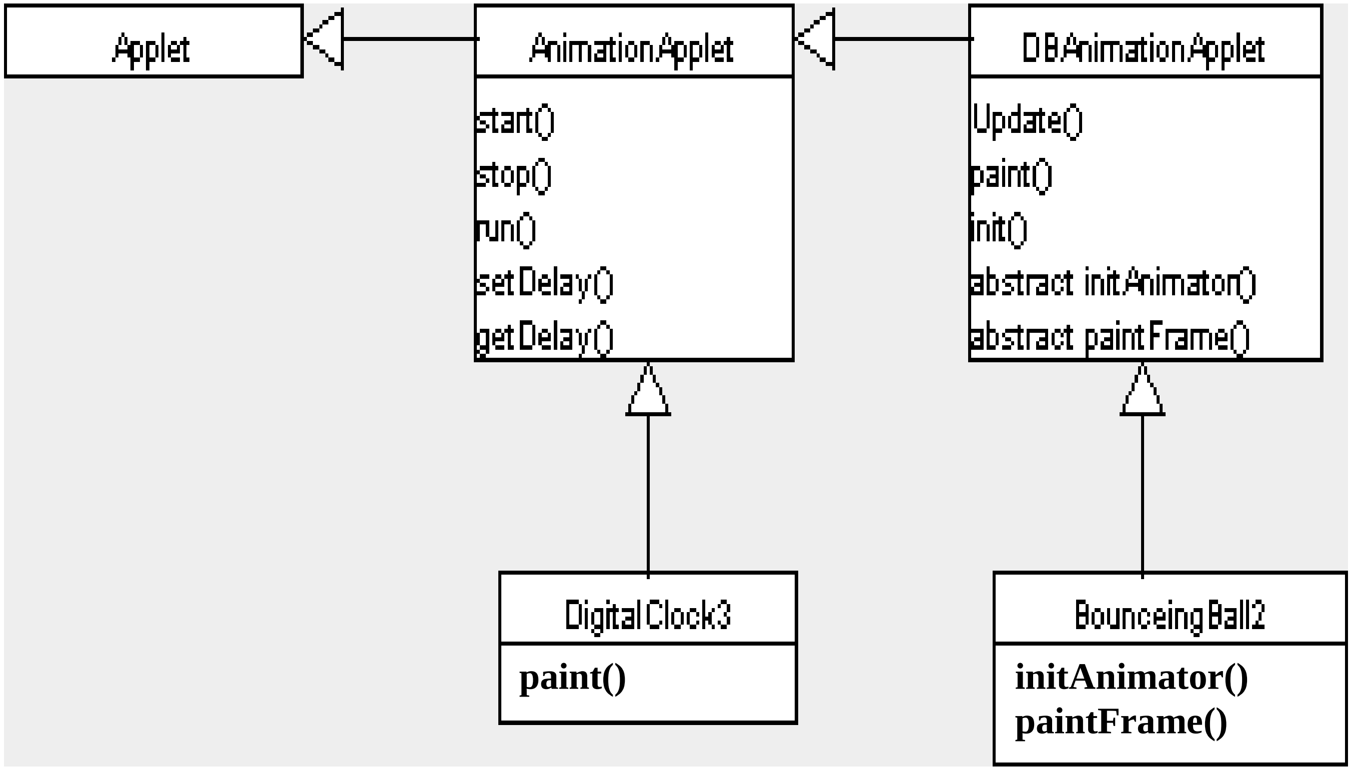
```
Class Common
{ void method()
    { <common code segment 1>
      contextSpecificCode()
      <common code segment 2>
     }
  abstract  void contextSpecificCode()
}
```

```
Class ContextA extends Common
{ void context SpecificCode()
   {  <context specific code A>  }
…….
}
```

```
Class ContextB extends Common
{ void context SpecificCode()
   {  <context specific code B>  }
…….
}
```

Each context contains only its unique code.

Design Patterns - Examples



The abstract method paintFrame() is the context specific part of the double buffered animated applet. paintFrame() and initAnimator() are so-called "hook" methods. They are placeholders for methods which are implemented differently in each implementation. The DBAnimationApplet is an example of a Template Design pattern. There are presumably many classes with the pattern it implements.

Design Patterns - Examples

```
import java.awt.*;
public abstract class DBAnimationApplet
    extends AnimationApplet {
  final public void update(Graphics g) {
    if (doubleBuffered) {
      paintFrame(offscreen);
      g.drawImage(im, 0, 0, this);
    } else {
      // super.update();
    }
  }
  final public void paint(Graphics g) {
    paintFrame(g);
  }
  final public void init() {
    d = getSize();
    im = createImage(d.width, d.height);
    offscreen = im.getGraphics();
    initAnimator();
  }
```

Applet inherits update(Graphics g) from Container.

The “this” in the g.drawImage(im, 0, 0, this); method invocation is the panel in which the applet is executing.

abstract <u>Graphics</u> <u>getGraphics</u>()
Creates a graphics context for drawing to an off-screen image.

<u>Image</u> <u>createImage</u>(int width, int height)
Creates an off-screen drawable image to be used for double buffering.

```
protected void initAnimator() {}
  abstract protected void paintFrame(Graphics g);
  protected DBAnimationApplet(boolean doubleBuffered) {
    this.doubleBuffered = doubleBuffered;
  }
  protected DBAnimationApplet() {
    this.doubleBuffered = true;
  }
  protected boolean doubleBuffered;
  protected Dimension d;
  protected Image im;
  protected Graphics offscreen;
}
```

"im" is an instance of the Image abstract class.
public abstract class Image
extends Object

The abstract class `Image` is the superclass of all classes that represent graphical images. The image must be obtained in a platform-specific manner.

public abstract class Graphics extends Object

The `Graphics` class is the abstract base class for all graphics contexts that allow an application to draw onto components that are realized on various devices, as well as onto off-screen images.

How do we use the DBAnimationApplet?

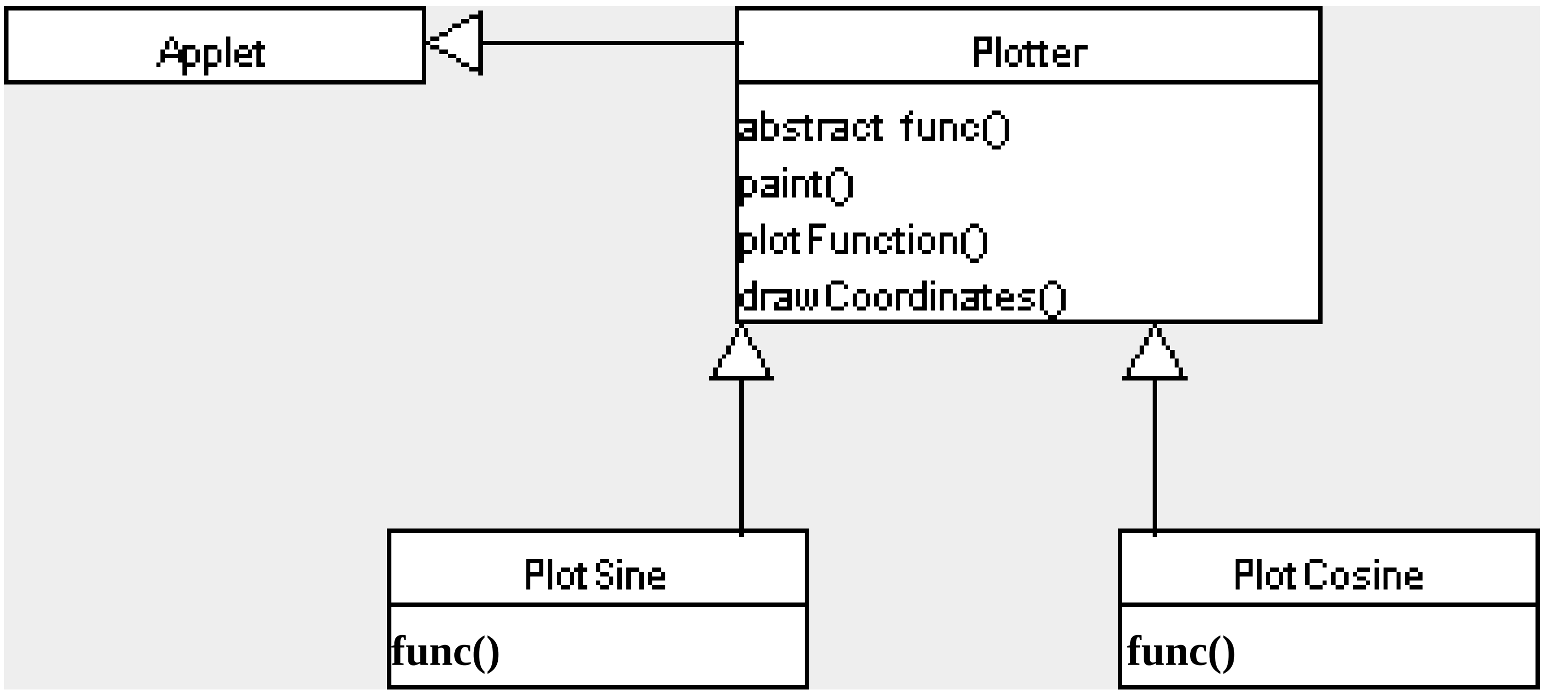
1. Extend DBAnimationApplet
2. Override paintFrame() - hook method
3. Override initAnimation() (if necessary) - hook method

```
import java.awt.*;
public class BouncingBall2 extends DBAnimationApplet {
 public BouncingBall2() {
  super(true);  // double buffering
 }
 protected void initAnimator() {
  String att = getParameter("delay");
  if (att != null)
   setDelay(Integer.parseInt(att));
  x = d.width * 2 / 3 ;
  y = d.height - radius;
 }
```

```
protected void paintFrame(Graphics g) {
    g.setColor(Color.white);
    g.fillRect(0,0,d.width,d.height);
    if (x < radius || x > d.width - radius) {
      dx = -dx;
    }
    if (y < radius || y > d.height - radius) {
      dy  =  -dy;
    }
    x += dx; y += dy;
    g.setColor(color);
    g.fillOval(x - radius, y - radius, radius * 2, radius * 2);
  }

  protected int x, y;
  protected int dx = -2, dy = -4;
  protected int radius = 20;
  protected Color color = Color.green;
}
```

The Generic Function Plotter - Another Instance of a Template Design Pattern



The structure of all simple plot classes will follow the plotter template.

Only the function to be plotted will vary. That is, the hook method is func().

Design Patterns - Examples

```
import java.awt.*;
public abstract class Plotter extends java.applet.Applet {
  public abstract double func(double x);
  public void init() {
    d = getSize();
    String att = getParameter("xratio");
    if (att != null)
      xratio = Integer.parseInt(att);
    att = getParameter("yratio");
    if (att != null)
      yratio = Integer.parseInt(att);
    att = getParameter("xorigin");
    if (att != null)
      xorigin = Integer.parseInt(att);
    else
      xorigin = d.width / 2;
    att = getParameter("yorigin");
    if (att != null)
      yorigin = Integer.parseInt(att);
    else
      yorigin = d.height / 2;
  }
```

```
public void paint(Graphics g) {
    drawCoordinates(g);
    plotFunction(g);
  }

  /** the dimension of the viewing area */
  protected Dimension d;

  /** The color used for plotting */
  protected Color color = Color.black;

  /** The position of the origin of the coordinate system */
  protected int xorigin, yorigin;

  /** The number of pixels between 0 and 1 in x and y direction */
  protected int xratio = 100, yratio = 100;
```

Design Patterns - Examples

```
protected void plotFunction(Graphics g) {
    for (int px = 0; px < d.width; px++) {
      try {
        double x = (double)(px - xorigin) / (double)xratio;
        double y = func(x);
        int py = yorigin - (int) (y * yratio);
        g.fillOval(px - 1, py - 1, 3, 3);
      } catch (Exception e) {}
    }
  }
  protected void drawCoordinates(Graphics g) {
    ……….
    ………..
  }
}
```

```
public class PlotSine extends Plotter {
  public double func(double x) {
    return Math.sin(x);
  }
}
```