**Mouse Events**

***Introduction***

Interactive input in graphical applications is often accomplished with a pointing device such as a mouse or trackball. Such applications react to mouse button clicks, mouse movement, and mouse dragging (i.e., moving the mouse when a button is held down). BreezyGUI includes several methods that respond to mouse events. Each method has two parameters:

1. The x coordinate of the mouse when the event occurs
2. The y coordinate of the mouse when the event occurs.

BreezyGUI's mouse handling methods are listed in the following table:

| **Method** | **What it does** |
| --- | --- |
| public void mouseClicked  (int x, int y) | This method is called when a mouse button is clicked. |
| public void mousePressed  (int x, int y) | This method is called when a mouse button is pressed down. |
| public void mouseReleased  (int x, int y) | This method is called when a mouse button is released. |
| public void mouseMoved  (int x, int y) | This method is called when the mouse is moved. |
| public void mouseDragged  (int x, int y) | This method is called when the mouse is moved with a button pressed. |

To detect and handle mouse events, an application must implement one or more of the above methods. A mouse handling method typically begins by transferring the values of the mouse coordinates to application instance variables. The method then usually invokes the repaint method, which in turn performs some actions and updates the display. For example, consider the following mousePressed method:

public void mousePressed (int x, int y){

mouseX = x;

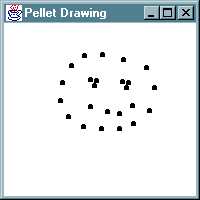
mouseY = y;

repaint();

}

***Example 1: A Primitive Sketching Program***

As a first example, we present a program that allows a user to make a very primitive type of drawing. Each time the user depresses a mouse button, the program draws a pellet-sized dot at the mouse's current location, with results similar to the following:



The drawing is supposed to look like a happy face. Here is the code:

import java.awt.\*;

import BreezyGUI.\*;

public class Sketchpad1 extends GBFrame{

public Sketchpad1(){

setTitle ("Pellet Drawing");

}

public void mousePressed (int x, int y){

Graphics g = getGraphics();

g.fillOval (x, y, 5, 5);

}

public static void main (String[] args){

Frame frm = new Sketchpad1();

frm.setSize (200, 200);

frm.setVisible (true);

}

}

***The Transient Image Problem and the Sketching Program***

In addition to being primitive, the sketching program has a major problem. When the application window is resized or covered and then uncovered, all the dots disappear. The reason is that the application draws a transient image. When the window is resized, Java invokes repaint, which invokes update, which erases any images drawn by the program.

To draw a permanent or refreshable image, one that reappears when the window is resized, the application must maintain a record of the image in its instance variables and redraw the image when necessary. We now modify the application so that it maintains a list of pellet positions in two parallel arrays. In addition:

1. When the mousePressed method is invoked, the x and y coordinates are added to the parallel arrays, one array for the x's and the other for the y's.
2. The paint method loops through the two parallel arrays and draws pellets at the indicated positions.

The application handles the details of saving and displaying points in two new methods, savePoint(int x, int y) and displayPoints(Graphics g). Here is the listing:

import java.awt.\*;

import BreezyGUI.\*;

public class Sketchpad2 extends GBFrame{

private static int MAX\_POINTS = 500;

private int numPoints;

private int xArray[];

private int yArray[];

public Sketchpad2(){

setTitle ("Pellet Drawing");

numPoints = 0;

xArray = new int[MAX\_POINTS];

yArray = new int[MAX\_POINTS];

}

public void paint (Graphics g){

displayPoints (g);

}

public void mousePressed (int x, int y){

if (numPoints < xArray.length){

Graphics g = getGraphics();

g.fillOval (x, y, 5, 5);

savePoint (x, y);

}

else

messageBox ("Sorry: cannot draw another pellet.");

}

private void displayPoints (Graphics g){

int i;

for (i = 0; i < numPoints; i++){

g.fillOval (xArray[i], yArray[i], 5, 5);

}

}

private void savePoint (int x, int y){

xArray[numPoints] = x;

yArray[numPoints] = y;

numPoints++;

}

public static void main (String[] args){

Frame frm = new Sketchpad2();

frm.setSize (200, 200);

frm.setVisible (true);

}

}

***Improving the Sketching Program***

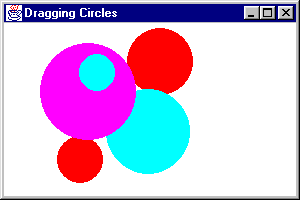
There are several simple but worthwhile modifications that can be made to our still very primitive drawing program.

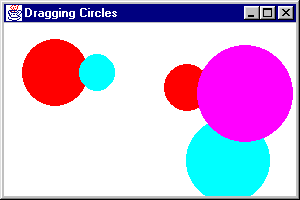
1. When the arrays become full, the program can no longer accept mouse clicks, so it displays a message box. There are several ways to deal with this problem. One way is to resize the arrays when they become full, perhaps by adding 50 new cells each time this happens. Another approach is to use one of Java's list classes to hold the coordinates. These classes implement collections that grow as needed to accommodate the data being added.

1. The program can be modified to draw "strokes" rather than distinct pellets. A stroke consists of a sequence of line segments connecting successive mouse positions. A stroke begins when the user depresses a button and continues until the user releases the button. In between, while the mouse is being dragged, the mouseDragged method adds mouse positions to the parallel arrays. An entry of (-999, -999) in the parallel arrays is used to mark the break between strokes. The modifications needed in the displayPoints methods are fairly straightforward.

***Example 2: Dragging Circle Objects***

As a second example of a mouse enabled program, we consider an application that begins by drawing several circles of random color and size in a window. The user can then use the mouse to drag these circles around the window. Here is what the interface looks like before and after some circles have been dragged around:





This application uses two classes, one to define a circle and the other to define the user interface. Here is the code:

import java.awt.\*;

public class Circle{

private int centerX, centerY, radius;

private Color color;

public Circle(int x, int y, int r, Color c){

centerX = x;

centerY = y;

radius = r;

color = c;

}

public int getCenterX(){

return centerX;

}

public int getCenterY(){

return centerY;

}

public int getRadius(){

return radius;

}

public Color getColor(){

return color;

}

public void setCenterX(int x){

centerX = x;

}

public void setCenterY(int y){

centerY = y;

}

public void setRadius(int r){

radius = r;

}

public void setColor(Color c){

color = c;

}

public void draw(Graphics g){

Color oldColor = g.getColor();

g.setColor(color);

// Translates circle's center to rectangle's origin for drawing.

g.fillOval(centerX - radius, centerY - radius, radius \* 2, radius \* 2);

g.setColor(oldColor);

}

public boolean containsPoint(int x, int y){

int xSquared = (x - centerX) \* (x - centerX);

int ySquared = (y - centerY) \* (y - centerY);

int radiusSquared = radius \* radius;

return xSquared + ySquared - radiusSquared <= 0;

}

public void move(int xAmount, int yAmount){

centerX = centerX + xAmount;

centerY = centerY + yAmount;

}

}

import java.awt.\*;

import BreezyGUI.\*;

public class DragCircles1 extends GBFrame{

private final static int MAX\_CIRCLES = 5;

private final static int MIN\_CENTER\_X = 50;

private final static int MAX\_CENTER\_X = 150;

private final static int MIN\_CENTER\_Y = 50;

private final static int MAX\_CENTER\_Y = 150;

private final static int MIN\_RADIUS = 10;

private final static int MAX\_RADIUS = 50;

private int mouseX = 0, mouseY = 0;

private boolean circleHasBeenSelected = true;

private int currentX = 0;

private int currentY = 0;

private Circle selectedCircle;

private Circle circles[] = new Circle[MAX\_CIRCLES];

public DragCircles1(){

int i;

for (i = 0; i < MAX\_CIRCLES; i++){

int centerX = randomInt(MIN\_CENTER\_X, MAX\_CENTER\_X);

int centerY = randomInt(MIN\_CENTER\_Y, MAX\_CENTER\_Y);

int radius = randomInt(MIN\_RADIUS, MAX\_RADIUS);

Color color = randomColor();

Circle circle = new Circle(centerX, centerY, radius, color);

circles[i] = circle;

}

setTitle("Dragging Circles");

}

public void paint (Graphics g){

int i;

for (i = 0; i < MAX\_CIRCLES; i++)

circles[i].draw(g);

}

public void mousePressed(int x, int y){

currentX = x;

currentY = y;

circleHasBeenSelected = findCircle();

}

public void mouseReleased(int x, int y){

circleHasBeenSelected = false;

}

public void mouseDragged(int x, int y){

if (circleHasBeenSelected){

selectedCircle.setCenterX(

selectedCircle.getCenterX() - (currentX - x));

selectedCircle.setCenterY(

selectedCircle.getCenterY() - (currentY - y));

repaint();

currentX = x;

currentY = y;

}

}

private boolean findCircle(){

int i;

for (i = 0; i < MAX\_CIRCLES; i++)

if (circles[i].containsPoint(currentX, currentY)){

selectedCircle = circles[i];

return true;

}

return false;

}

private int randomInt(int low, int high){

return (int) (low + Math.random() \* (high - low + 1));

}

private Color randomColor(){

Color color;

int number = randomInt(1, 5);

switch (number){

case 1:

color = Color.red;

break;

case 2:

color = Color.blue;

break;

case 3:

color = Color.green;

break;

case 4:

color = Color.magenta;

break;

case 5:

color = Color.cyan;

break;

default: color = Color.orange;

}

return color;

}

public static void main (String[] args){

Frame frm = new DragCircles1();

frm.setSize (300, 200);

frm.setVisible(true);

}

}

***Getting Rid of Flicker***

The above program has a problem. As we drag a circle around, the window seems to flicker. The cause lies in the mouseDragged method. Every time we move a circle, the program calls the repaint method. This method clears the window and then calls paint to redraw the circles. Unless the computer is very fast, the human eye experiences the cycles of clear and redraw as flicker. Fortunately, there is a way to overcome flicker. Java allows drawing to be done in two different modes. In the default mode, which we have been using exclusively so far, images (lines, text, shapes) overwrite whatever happens to be underneath them. This mode is called ***paint***. The second mode is called ***XOR***. The result of drawing an image in XOR mode depends on two colors -- the current color and the XOR color. Wherever the image overlays the XOR color it is drawn in the current color, and vice-versa. A consequence of this strange convention is that if an image is redrawn on top of itself, it disappears, and the window returns to its original appearance before the image was first drawn. Pixels of a different color (neither the current color nor the XOR color) underneath the image are changed in a manner too difficult to explain here; however, redrawing over these pixels returns them to their original color too.

In the code that follows, we use XOR to solve the flicker problem. When we select a circle, we draw an outline of it in XOR mode. Then, as the user drags the mouse, we redraw the outline using XOR (restoring the pixels covered by the outline) and draw another outline in XOR mode at the mouse's new location. Drawing an outline involves so few pixels that the user sees no flicker and believes the outline is smoothly following the mouse around the window. When the user finally releases the mouse, we repaint the window.

Here are the required modifications to the program:

public void mousePressed(int x, int y){

currentX = x;

currentY = y;

circleHasBeenSelected = findCircle();

if (circleHasBeenSelected)

selectedCircle.drawOutline (getGraphics());

}

public void mouseReleased(int x, int y){

circleHasBeenSelected = false;

repaint();

}

public void mouseDragged(int x, int y){

if (circleHasBeenSelected){

Graphics g = getGraphics();

selectedCircle.drawOutline (g); // Old location

selectedCircle.move(x - currentX, y - currentY);

selectedCircle.drawOutline (g); // New location

currentX = x;

currentY = y;

}

}

The drawOutline method in class Circle looks like this:

public void drawOutline (Graphics g){

Color oldColor = g.getColor();

g.setColor (Color.black);

g.setXORMode (Color.white);

g.drawOval (centerX - radius, centerY - radius, radius \* 2, radius \* 2);

g.setColor (oldColor);

g.setPaintMode();

}