Building Java Programs

Graphics

Reading: Supplement 3G
Objects (usage)

- **object**: An entity that contains data and behavior.
  - **data**: variables inside the object
  - **behavior**: methods inside the object
    - You interact with the methods; the data is hidden in the object.
    - A **class** is a type of objects.

- Constructing (creating) an object:
  
  ```
  Type objectName = new Type(parameters);
  ```

- Calling an object's method:
  
  ```
  objectName. methodName(parameters);
  ```
Graphical objects

We will draw graphics in Java using 3 kinds of objects:

- **DrawingPanel**: A window on the screen.
  - Not part of Java; provided by the authors. See class web site.

- **Graphics**: A "pen" to draw shapes and lines on a window.

- **Color**: Colors in which to draw shapes.
DrawingPanel

• To create a window:
  DrawingPanel <name> = new DrawingPanel(<width>, <height>);

  Example:
  DrawingPanel panel = new DrawingPanel(300, 200);

• The window has nothing on it.
  • We can draw shapes and lines on it using another object of type Graphics.
Shapes are drawn using an object of class `Graphics`.

You must place an import declaration in your program:
```
import java.awt.*;
```

Access it by calling `getGraphics` on your `DrawingPanel`.
```
Graphics g = panel.getGraphics();
```

Draw shapes by calling methods on the `Graphics` object.
```
g.fillRect(10, 30, 60, 35);
g.fillOval(80, 40, 50, 70);
```
## Graphics methods

<table>
<thead>
<tr>
<th>Method name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>g.drawLine(x1, y1, x2, y2);</code></td>
<td>line between points (x1, y1), (x2, y2)</td>
</tr>
<tr>
<td><code>g.drawOval(x, y, width, height);</code></td>
<td>outline largest oval that fits in a box of size width * height with top-left at (x, y)</td>
</tr>
<tr>
<td><code>g.drawRect(x, y, width, height);</code></td>
<td>outline of rectangle of size width * height with top-left at (x, y)</td>
</tr>
<tr>
<td><code>g.drawString(text, x, y);</code></td>
<td>text with bottom-left at (x, y)</td>
</tr>
<tr>
<td><code>g.fillOval(x, y, width, height);</code></td>
<td>fill largest oval that fits in a box of size width * height with top-left at (x, y)</td>
</tr>
<tr>
<td><code>g.fillRect(x, y, width, height);</code></td>
<td>fill rectangle of size width * height with top-left at (x, y)</td>
</tr>
<tr>
<td><code>g.setColor(Color);</code></td>
<td>set Graphics to paint any following shapes in the given color</td>
</tr>
</tbody>
</table>
Coordinate system

- Each \((x, y)\) position is a *pixel* ("picture element").

- \((0, 0)\) is at the window's top-left corner.
  - \(x\) increases rightward and the \(y\) increases downward.

- The rectangle from \((0, 0)\) to \((200, 100)\) looks like this:
Colors

- Colors are specified by `Color` class constants named: BLACK, BLUE, CYAN, DARK_GRAY, GRAY, GREEN, LIGHT_GRAY, MAGENTA, ORANGE, PINK, RED, WHITE, YELLOW

- Pass to Graphics object's `setColor` method:
  ```java
  g.setColor(Color.BLACK);
  g.fillRect(10, 30, 100, 50);
  g.setColor(Color.RED);
  g.fillOval(60, 40, 40, 70);
  ```

- The background color can be set by calling `setBackground` on the DrawingPanel:
  ```java
  panel.setBackground(Color.YELLOW);
  ```
Outlined shapes

- To draw a shape with a fill and outline, first fill it in the fill color and then draw the same shape in the outline color.

```java
import java.awt.*; // so I can use Graphics
public class DrawOutline {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(150, 70);
        Graphics g = panel.getGraphics();

        // inner red fill
        g.setColor(Color.RED);
        g.fillRect(20, 10, 100, 50);

        // black outline
        g.setColor(Color.BLACK);
        g.drawRect(20, 10, 100, 50);
    }
}
```
Superimposing shapes

- When two shapes occupy the same pixels, the last one drawn is seen.

```java
import java.awt.*;

public class DrawCar {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(200, 100);
        panel.setBackground(Color.LIGHT_GRAY);
        Graphics g = panel.getGraphics();
        g.setColor(Color.BLACK);
        g.fillRect(10, 30, 100, 50);
        g.setColor(Color.RED);
        g.fillOval(20, 70, 20, 20);
        g.fillOval(80, 70, 20, 20);
        g.setColor(Color.CYAN);
        g.fillRect(80, 40, 30, 20);
    }
}
```
Drawing with loops

- The $x$, $y$, $w$, $h$ expression can contain the loop counter, $i$.

```java
DrawingPanel panel = new DrawingPanel(400, 300);  
panel.setBackground(Color.YELLOW);  
Graphics g = panel.getGraphics();  

g.setColor(Color.RED);  
for (int i = 1; i <= 10; i++) {  
    g.fillOval(100 + 20 * i, 5 + 20 * i, 50, 50);  
}
```

```java
DrawingPanel panel = new DrawingPanel(250, 220);  
Graphics g = panel.getGraphics();  
g.setColor(Color.MAGENTA);  
for (int i = 1; i <= 10; i++) {  
    g.drawOval(30, 5, 20 * i, 20 * i);  
}
```
Loops that begin at 0

- Beginning a loop at 0 and using < can make coordinates easier to compute.

- Example:
  - Draw ten stacked rectangles starting at (20, 20), height 10, width starting at 100 and decreasing by 10 each time:

```java
DrawingPanel panel = new DrawingPanel(160, 160);
Graphics g = panel.getGraphics();

for (int i = 0; i < 10; i++) {
    g.drawRect(20, 20 + 10 * i,
               100 - 10 * i, 10);
}
```
Drawing w/ loops questions

• Code from previous slide:

```java
DrawingPanel panel = new DrawingPanel(160, 160);
Graphics g = panel.getGraphics();

for (int i = 0; i < 10; i++) {
    g.drawRect(20, 20 + 10 * i,
               100 - 10 * i, 10);
}
```

• Write variations of the above program that draw the figures at right as output.
Drawing w/ loops answers

- **Solution #1:**
  
  ```java
  Graphics g = panel.getGraphics();
  for (int i = 0; i < 10; i++) {
    g.drawRect(20 + 10 * i, 20 + 10 * i, 100 - 10 * i, 10);
  }
  ```

- **Solution #2:**
  
  ```java
  Graphics g = panel.getGraphics();
  for (int i = 0; i < 10; i++) {
    g.drawRect(110 - 10 * i, 20 + 10 * i, 10 + 10 * i, 10);
  }
  ```
To draw in multiple methods, you must pass Graphics g.

```java
import java.awt.*;

public class DrawCar1 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(200, 100);
        panel.setBackground(Color.LIGHT_GRAY);
        Graphics g = panel.getGraphics();
        drawCar(g);
    }

    public static void drawCar(Graphics g) {
        g.setColor(Color.BLACK);
        g.fillRect(10, 30, 100, 50);
        g.setColor(Color.RED);
        g.fillOval(20, 70, 20, 20);
        g.fillOval(80, 70, 20, 20);
        g.setColor(Color.CYAN);
        g.fillRect(80, 40, 30, 20);
    }
}
```
Parameterized figures

- Modify the car-drawing method so that it can draw many cars, such as in the following image.
  - Top-left corners: (10, 30), (150, 10)
  - Hint: We must modify our `drawCar` method to accept x/y coordinates as parameters.
import java.awt.*;

public class DrawCar2 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(260, 100);
        panel.setBackground(Color.LIGHT_GRAY);
        Graphics g = panel.getGraphics();
        drawCar(g, 10, 30);
        drawCar(g, 150, 10);
    }

    public static void drawCar(Graphics g, int x, int y) {
        g.setColor(Color.BLACK);
        g.fillRect(x, y, 100, 50);
        g.setColor(Color.RED);
        g.fillOval(x + 10, y + 40, 20, 20);
        g.fillOval(x + 70, y + 40, 20, 20);
        g.setColor(Color.CYAN);
        g.fillRect(x + 70, y + 10, 30, 20);
    }
}
Modify `drawCar` to allow the car to be drawn at any size.

- Existing car: size 100. Second car: (150, 10), size 50.

Once you have this working, use a `for` loop with your method to draw a line of cars, like the picture at right.

- Start at (10, 130), each size 40, separated by 50px.
import java.awt.*;

public class DrawCar3 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(210, 100);
        panel.setBackground(Color.LIGHT_GRAY);

        Graphics g = panel.getGraphics();
        drawCar(g, 10, 30, 100);
        drawCar(g, 150, 10, 50);
        for (int i = 0; i < 5; i++) {
            drawCar(g, 10 + i * 50, 130, 40);
        }
    }

    public static void drawCar(Graphics g, int x, int y, int size) {
        g.setColor(Color.BLACK);
        g.fillRect(x, y, size, size / 2);

        g.setColor(Color.RED);
        g.fillOval(x + size / 10, y + 2 * size / 5, size / 5, size / 5);
        g.fillOval(x + 7 * size / 10, y + 2 * size / 5, size / 5, size / 5);

        g.setColor(Color.CYAN);
        g.fillRect(x + 7 * size / 10, y + size / 10, 3 * size / 10, size / 5);
    }
}

Drawing parameter answer

import java.awt.*;

public class DrawCar3 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(210, 100);
        panel.setBackground(Color.LIGHT_GRAY);

        Graphics g = panel.getGraphics();
        drawCar(g, 10, 30, 100);
        drawCar(g, 150, 10, 50);
        for (int i = 0; i < 5; i++) {
            drawCar(g, 10 + i * 50, 130, 40);
        }
    }

    public static void drawCar(Graphics g, int x, int y, int size) {
        g.setColor(Color.BLACK);
        g.fillRect(x, y, size, size / 2);

        g.setColor(Color.RED);
        g.fillOval(x + size / 10, y + 2 * size / 5, size / 5, size / 5);
        g.fillOval(x + 7 * size / 10, y + 2 * size / 5, size / 5, size / 5);

        g.setColor(Color.CYAN);
        g.fillRect(x + 7 * size / 10, y + size / 10, 3 * size / 10, size / 5);
    }
}
Custom colors

• You can construct custom Color objects.
  • Pass 3 numbers from 0-255 for red, green, and blue.

    DrawingPanel panel = new DrawingPanel(80, 50);
    Color brown = new Color(192, 128, 64);
    panel.setBackground(brown);

    or:

    DrawingPanel panel = new DrawingPanel(80, 50);
    panel.setBackground(new Color(192, 128, 64));
Drawing polygons

- **Polygon** objects represent arbitrary shapes.
  - Add points to a Polygon using its `addPoint(x, y)` method.

**Example:**
```
DrawingPanel p = new DrawingPanel(100, 100);
Graphics g = p.getGraphics();
g.setColor(Color.GREEN);
Polygon poly = new Polygon();
poly.addPoint(10, 90);
poly.addPoint(50, 10);
poly.addPoint(90, 90);
g.fillPolygon(poly);
```
Animation with **sleep**

- DrawingPanel's **sleep** method pauses your program for a given number of milliseconds.

- You can use **sleep** to create simple animations.

  ```java
  DrawingPanel panel = new DrawingPanel(250, 200);
  Graphics g = panel.getGraphics();

  g.setColor(Color.BLUE);
  for (int i = 1; i <= NUM_CIRCLES; i++) {
    g.fillOval(15 * i, 15 * i, 30, 30);
    panel.sleep(500);
  }
  ```

- Try adding **sleep** commands to loops in past exercises in this chapter and watch the panel draw itself piece by piece.