Chapter outline

- drawing 2D graphics
  - DrawingPanel and Graphics objects
  - drawing and filling shapes
  - coordinate system
  - colors
  - drawing with loops
  - drawing with parameterized methods
  - basic animation

Graphical objects

- We will draw graphics using these classes of objects:
  - DrawingPanel: A window on the screen.
    - This is not part of Java; it is provided by the authors.
  - Graphics: A "pen" that can draw shapes/lines onto a window.
  - Color: The colors that indicate what color to draw our shapes.

DrawingPanel

- To create a window, construct a DrawingPanel object:
  ```java
drawingPanel <name> = new DrawingPanel(<width>, <height>);
```

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

- The window has nothing on it.
  - But we can draw shapes and lines on it using another object of a class named Graphics.
Graphics

Shapes are drawn using an object of class Graphics.
- You must place an import declaration in your program:
  ```java
  import java.awt.*;
  ```
- Access it by calling `getGraphics` on your `DrawingPanel`.
  ```java
  public class DrawingExample1 {
    public static void main(String[] args) {
      DrawingPanel panel = new DrawingPanel(300, 200);
      Graphics g = panel.getGraphics();
      g.fillRect(10, 30, 60, 35); g.fillOval(80, 40, 50, 70);
    }
  }
  ```

Once you have the Graphics object, draw shapes by calling its methods.
- Example:
  ```java
  g.fillRect(10, 30, 60, 35);
  g.fillOval(80, 40, 50, 70);
  ```

Coordinate system

- Each (x, y) position on the DrawingPanel is represented by a pixel (short for "picture element").
- The origin (0, 0) is at the window's top-left corner.
  - x increases rightward and the y increases downward
  - The y is reversed from what you may expect.
- The rectangle from (0, 0) to (200, 100) looks like this:

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

A complete program

```java
import java.awt.*;
public class DrawingExample1 {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(300, 200);
        Graphics g = panel.getGraphics();
        g.fillRect(10, 30, 60, 35);
        g.fillOval(80, 40, 50, 70);
    }
}
```
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 these to the `Graphics` object's `setColor` method.

  Example:
  ```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`:

  Example:
  ```java
  panel.setBackground(Color.YELLOW);
  ```

Custom colors

- It is also legal to construct a `Color` object of your own.
- Colors are specified by three numbers (ints from 0 to 255) representing the amount of red, green, and blue.
- Computers use red-green-blue or "RGB" as primary colors.

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

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

Superimposing shapes

- Drawing one shape on top of another causes the last shape to appear on top of the previous one(s).

  ```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

- We can draw many repetitions of the same item at different x/y positions with `for` loops.

  The x or y expression contains the loop counter, `i`, so that in each pass of the loop, when `i` changes, so does x or y.

  ```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);
  }
  g.setColor(Color.BLUE);
  for (int i = 1; i <= 10; i++) {
      g.drawString("Hello, world!", 150 - 10 * i, 200 + 10 * i);
  }
  ```
Loops to change shape's size

A for loop can also vary a shape's size:

```java
import java.awt.*;
public class DrawCircles {
    public static void main(String[] args) {
        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);
        }
    }
}
```

A loop that varies both

The loop in this program affects both the size and shape of the figures being drawn.

- Each pass of the loop, the square drawn becomes 20 pixels smaller in size, and shifts 10 pixels to the right.

```java
import java.awt.*;
public class DrawingExample2 {
    public static final int NUM_CIRCLES = 10;
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(220, 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);
        }
        g.setColor(Color.MAGENTA);
        for (int i = 1; i <= NUM_CIRCLES; i++) {
            g.fillOval(15 * (NUM_CIRCLES + 1 - i), 15 * i, 30, 30);
        }
    }
}
```

Drawing example 2

What sort of figure does the following code draw?

```java
import java.awt.*;
public class DrawingExample2 {
    public static final int NUM_CIRCLES = 10;
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(220, 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);
        }
        g.setColor(Color.MAGENTA);
        for (int i = 1; i <= NUM_CIRCLES; i++) {
            g.fillOval(15 * (NUM_CIRCLES + 1 - i), 15 * i, 30, 30);
        }
    }
}
```

Loops that begin at 0

Often with graphics (and loops in general), we begin our count at 0 and use \( < \) instead of \( \leq \).

- A loop that repeats from 0 to \( < 10 \) still repeats 10 times, just like a loop that repeats from 1 to \( \leq 10 \).
- But when the loop counter variable \( i \) is used to set the figure's coordinates, often starting at 0 gives the coordinates we want.

Example: Draw ten stacked rectangles starting at \((20, 20)\), height 10, with widths that start at 100 and decrease by 10 each time:

```java
import java.awt.*;
public class DrawingExample2 {
    public static final int NUM_CIRCLES = 10;
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(160, 160);
        Graphics g = panel.getGraphics();
        for (int i = 0; i < 10; i++) {
            g.drawRect(20 + 10 * i, 20 - 10 * i, 100 - 10 * i, 10);
        }
    }
}
```
Check below for the code snippets and examples:

### Drawing w/ loops questions
- Write variations of the preceding 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);
  }
  ```

### Drawing with methods
- It is possible to draw graphics in multiple methods.
  - Since you'll need to send commands to the `Graphics g` to draw the figure, you should pass `Graphics g` as a parameter.

```java
import java.awt.*;
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();
    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
- If you want to draw the same figure many times, write a method to draw that figure and accept the x/y position as parameters.
  - Adjust the x/y coordinates of your drawing commands to take into account the parameters.

- **Exercise:**
  Modify the previous car-drawing method to work at any location, so that it can produce an image such as the following:
  - One car's top-left corner is at (10, 30).
  - The other car's top-left corner is at (150, 10).
import java.awt.*; public class DrawingWithParameters {
  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);
  }
}

Methods can accept any number of parameters to adjust the figure's appearance.

Exercise:
Write a new version of the drawCar method that allows the cars to be drawn at any size, such as the following:

import java.awt.*; public class DrawingWithParameters2 {
  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);
  }

  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);
  }
}

Write a program that will display the following figures on a drawing panel of size 300x400:

- top-left figure:
  - overall size = 100
  - top-left corner = (10, 10)
  - inner rectangle and oval size = 50
  - inner top-left corner = (35, 35)

- top-right figure:
  - overall size = 60
  - top-left corner = (150, 10)
  - inner rectangle and oval size = 30
  - inner top-left corner = (165, 25)

- bottom figure:
  - overall size = 140
  - top-left corner = (60, 120)
  - inner rectangle and oval size = 70
  - inner top-left corner = (95, 155)
Parameterized figure answer

```java
import java.awt.
public class DrawFigures {
    public static void main(String[] args) {
        DrawingPanel panel = new DrawingPanel(400, 400);
        panel.setBackground(Color.CYAN);
        Graphics g = panel.getGraphics();
        drawFigure(g, 10, 10, 100);
        drawFigure(g, 150, 10, 60);
        drawFigure(g, 60, 120, 140);
    }
    public static void drawFigure(Graphics g, int x, int y, int size) {
        g.setColor(Color.RED);
        g.fillRect(x, y, size, size);
        g.setColor(Color.YELLOW);
        g.fillOval(x + size / 4, y + size / 4, size / 2, size / 2);
        g.setColor(Color.BLACK);
        g.drawLine(x + size / 4, y + size / 2,
                    x + size * 3 / 4, y + size / 2);
        g.drawLine(x + size / 2, y + size / 4,
                    x + size / 2, y + size * 3 / 4);
    }
}
```

Animation with sleep

- **DrawingPanel** has a method named `sleep` that pauses your program for a given number of milliseconds.
- You can use `sleep` to produce 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.

Drawing polygons

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

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