This Week's Topics

- We'll learn how to create objects and graphics
- Look at some more methods in the Graphics class
- How to declare variables and create objects
- How to use the java.awt.Color class
- How to create and use Java's primitive int type
- How to use the Font class and call "function" methods
- How to work with Java's "measuring" classes
- Reading: finish Chapter 2, read first section of Chapter 5

Graphics Arts

- The Graphics class contains methods to:
  - Draw simple lines
  - Draw rectangles, both outlined and filled
  - Draw fancy rounded and 3D rectangles
  - Draw ovals
- In later units you'll use the class to display images
- Right now, let's start with some simple drawing
Painting Lines

- Simplest of the various drawing methods:
  
  ```java
  g.drawLine(x1, y1, x2, y2);
  ```

- Takes 4 int arguments representing line endpoints
- Uses 1-pixel wide pen in default color
- Change color using `g.setColor()`
- Cannot change width of pen used to draw
- Inclusive algorithm: `x1,y1-x2,y2` are both drawn
- Example: `DrawStar.java`

Drawing Shapes I

- Most closed shapes use a “bounding box” concept

Drawing Shapes II

- Coordinates used for `x,y` represent “points”
  - Infinitely small location located between pixels
- Shape-drawing methods come in two varieties
  - Those that draw an outline
    - Use 1-pixel wide pen that hangs down/right
  - Those that draw a solid filled area
    - Fills an area inside coordinates
- Practical Effect:
  - Drawn shapes are 1-pixel bigger than filled shapes
Drawing Plain Rectangles

- Three methods:
  - `drawRect(int x, int y, int width, int height);`
  - `fillRect(int x, int y, int width, int height);`
  - `clearRect(int x, int y, int width, int height);`

  - First draws outline anchored at x,y in the foreground color
  - Will be one pixel higher and wider than requested
  - Second draws solid square in foreground color
  - Last works like `fillRect()`, using background color

  - Example: `SimpleRect.java [ run it ]`

Fancy Rectangles

- Two types:
  - Round-cornered rectangles
  - 3D (sunken or raised) rectangles

  - Here's the signature for the round-rect methods
    - `drawRoundRect(int x, int y, int w, int h, int xArc, int yArc);`
    - `fillRoundRect(int x, int y, int w, int h, int xArc, int yArc);`

  - Last two arguments represent corners.

Drawing Ovals

- Same methods as rectangles except no "clearOval"
  - `g.drawOval(int x, int y, int width, int height);`
  - `g.fillOval(int x, int y, int width, int height);`

  - Drawing thicker ovals?
    - Draw several and shrink the bounding box each time
    - Has a problem with "aliasing"
    - Fill larger then smaller oval [for background]

  - Complete exercise 4A
Objects and Variables

- When you call a method you pass information like this:
  ```java
g.drawRect( 0, 0, 100, 100 );
```
- The values 0 and 100 are called **literals** or **constants**
- Problems with literals
  - No meaning associated with number (magic number)
  - Making changes to code can be difficult
  - Literals only work for simple primitive types
- Solution? Learn to create **named values** called **variables**

Creating Variables

- A variable is a "named memory location that holds a value"
- Create (or define) a variable using this syntax:
  ```java
  [modifiers] type name [ = initial-value ];
  ```
- Parts in [ ] are optional. The brackets aren't part of syntax.
- Here are some example definitions:
  ```java
  int width;          // no modifiers or initializer
  int height = 95;   // uses initializer
  private Color background; // private modifier
  public static final int SIZE = 35; // the works
  ```

Variable Scope

- You can create variables in two places:
  - Inside a method: called a **local variable**
  - Outside a method: called a **field** or **instance variable**
- **Local variables**
  - Can only be used inside method where defined
  - Can only be used **after** definition
  - Not automatically given a value when defined
    - Must use initializer or assignment before you use variable
  - Do not use any modifiers with local variables
Variable Scope II

- **Instance variables** or fields
  - Defined outside of any method, but inside class
  - Can be used inside **any method** in the class
  - Do not have to be defined “before” method (like Pascal)
- Normally instance variables will be **private**
  - Don’t want variable used outside of class
  - In rare instances will use **public** or **protected**
- Instance variables are automatically **initialized**
  - Numeric variables are given value 0
  - boolean variables are given value false
  - Reference types (objects and strings) are given value null

Initializing Variables

- Initializing primitive types like int
  - Use literals or constants to store value

```java
public class Example extends Applet {
    private int leftCorner;
    private int rightCorner = 25;
    public void paint(Graphics g) {
        int size = 25;
    }
}
```

- Created almost the same way as creating a primitive
  - For every object variable you describe
    - What specific **kind** of thing that will be stored
    - What **name** you want to use for the variable
    - What **value** the attribute should have
- **Step 1**: Start with the “**kind**” of thing
Object Variables II

- Step 2: Give your variable a name to identify it
  - Color hairColor ...
  - Button actionBtn ...
  - Font headLineFnt ...
  - Frog kermit ...

- Step 3: Assign a value to your variable
  - Color hairColor = <make a Color object>;
  - Button actionBtn = <make a Button>;
  - Font headLineFnt = <make a Font>;
  - Frog kermit = <make a Frog>;

Object Variables III

- How do we fill in the <make a button> part?
  - Use the keyword new with a constructor
    - A constructor is a factory that makes objects
    - It always has the same name as the class
  - Color hairColor = new Color(128, 3, 15);
  - Button actionBtn = new Button("Hi");
  - Font headLineFnt = new Font("Serif", Font.BOLD, 36);
  - Frog kermit = new Frog("green", Frog.MALE);

- Constructors may take different arguments

Variable Recap

- You must define every variable:
  - Begin with the type (or class) name
  - Name your variable, subject to rules and conventions
  - Give it an initial value using assignment operator
  - Is it a primitive? Use literal value
  - Create a new object using new and a constructor
    - Constructor will have same name as class
  - Customize by passing arguments in parentheses
    - Look up constructor arguments in the JDK documentation
  - Complete Exercise 4B
Color Objects I
- Look up Color class, find constructor summary
  `Color(int red, int blue, int green);`
- Pass integer values between 0 (off) and 255 (on)
- Use your Color variable like this:
  ```java
  public void paint(Graphics g)
  {
    Color background = new Color(255, 128, 128);
    g.setColor(background);
  }
  ```
- Can also use setForeground(), setBackground()

Color Objects II
- Java 2 browsers can also display transparent colors
  `Color(int red, int green, int blue, int alpha);`
- The alpha channel is 0 (transparent) to 255 (opaque)
- Example: `AlphaColor.java` [run the applet]
- Color (class) constants
  - “Variables” that are part of the Color class
  - Probably should use lowercase version, rather than upper
  ```java
  g.setColor(Color.lightGray);
  ```

3D Rectangles
- draw3DRect(int x, int y, int w, int h, boolean raised);
- Takes 5 arguments, last is boolean (true or false)
  - Rectangle is raised if last arg is true
  - Rectangle is depressed if last arg is false
  - Only looks good on gray background
  - Need to repeat to make it look good
- Example: `ThreeDRect.java` [run the applet]
- Complete Exercise 4C
Java programs are composed of cooperating objects
- You've worked with Applet, Color, and Graphics, etc.
- Peer inside, and you’ll find simpler components
- At the bottom - the bit: the “atoms” of the computing world
- One step up from bits, are primitive or fundamental types
  - Similar to objects: have state and behavior
  - Value types rather than reference types
  - Unlike objects, can only hold a single, simple value
  - Manipulated by operators rather than methods
  - Behavior is built-in; it cannot be extended

### Meet the Integers I
- The first primitive types we’ll look at are integers
- Integers are whole numbers, including zero and negative
  - 27 and -345 are integers, 7.25 is not an integer
- Theoretically, integers are infinite
  - No matter how big, there’s always room for another digit
- Computer-implemented integer primitives are finite
  - Stored in fixed area of memory
  - Represents “integer concept”, limited by constraints
  - Similar to the way B/W photo represents reality

### Meet the Integers II
- Computer integers come in two varieties
  - Signed integers hold both positive and negative numbers
  - Unsigned integers hold only zero and the positive numbers
- For any given amount of storage:
  - Both types can store the same “number of numbers”
    - Eight bits can store 256 numbers whether signed or unsigned
    - The unsigned number represents a greater (2X) magnitude
      - With 16 bits, unsigned = 0 – 65,535 (65,536 values)
      - As an unsigned = -32,768 to 32,767 (65,536 values)
  - All Java integers are signed
The "Four Families" of Java Integers

- The byte uses 8 bits of storage
  - Represents 256 numbers from -128 to 127
- The short uses 16 bit of memory
  - Represents 65,536 numbers from -32,768 to 32,767
- The int (the "natural sized") uses 32 bits of memory
  - Represents slightly more than 4 billion numbers
    -2,147,483,648 to  2,147,483,647
- The long uses 64 bits of memory
  - Represents numbers from (roughly) +/- nine quintillion

Writing Integers

- To construct an integer variable, use integer type-name where you would have used a class name, like this:


```
byte myAge = <byte literal>;
short milesToGo = <short literal>;
ing populationUSA = <int literal>;
long populationWorld = <long literal>;
```

- Initializer not required for fields [defaults to 0]
- Initializer is required for local variables
- Initializer uses literals instead of the new operator

Writing Literal Integer Values

- Write numbers without commas or decimals


```
byte myAge = 49;
short milesToGo = 23455;
ing populationUSA = 263456732;
long populationWorld = 5374678328;
```

- Literals use int storage, even if small
- Literals assume number is decimal (base 10)
- You can change these assumptions if you like
Storage and Bases

- Changing the way literals are interpreted
- Use an L suffix for long storage
- 3 uses int storage, 3L uses long storage
- Literals that start with zero are considered octal
- Literals that start with zero-x are hexadecimal

```
int ten = 10;
int eight = 010;
int sixteen = 0x10;
int htmlColor = 0x3F703F
```

Making a Choice

- Which integer type should you use in your programs?
- Unless you have a reason not to, use the int
- Smaller sizes won’t necessarily save memory
- Calculations always use int or long
- Integers can overflow without warning
- Here are two examples:

```
(400000 * 400000 / 400000) == 2715?
2147483647 + 1 == -2147483648?
```

- Complete Exercise 4D

Font Objects

- Construct Font objects using this constructor

```
Font(String type, int style, int size);
```
- Type: "Serif", "SansSerif", "Monospaced", "Dialog"
- Style: Font.PLAIN, Font.BOLD, Font.ITALIC
- Size: points (really, pixels) for height of average character
- Use Font objects like this:

```
Font bigFont = new Font("Serif", Font.BOLD, 36);
g.setFont(bigFont);
```
Value Producing Methods

- All of the methods you've used perform an action
  - Signature starts with keyword `void` in documentation
  - Value-producing methods have return type instead
  - Font methods `getSize()`, `getName()`, `getStyle()`
- A function returns a value. What do you do with it?
  - Ignore it? OK, but not very common in Java
  - Store it in a variable?
    - Good, but variable must be of the correct type
  - Use the value directly like this:
    ```java
g.drawString(curFont.getName(), 10, 20);
```

The Three "Measuring" Classes

- Three simple classes with public instance variables
  - Rectangle: x, y, height, and width
  - Dimension: height, width
  - Point: x, y
- Access the fields directly or use methods
  ```java
  Rectangle r = this.getBounds();
r.grow(-8, -8);           // Shrink
r.width = r.width + 5; // expand width
Dimension d = this.getSize();
  ```
- Complete Exercise 4E

Week 4 Wrap Up

- Lab: Wormhole lablet
  - Working with variables and Graphics
  - Use class time on Wednesday/Thursday
  - Due before Tuesday, February 24
- Homework 4: Wormhole modifications
  - Due on Tuesday, March 2
  - Homework 3 (Nash) due by Tuesday, February 24
- Quiz 4: Deadline is Tuesday, February 24