Classes and Objects
A Foundation for Programming

any program you might want to write

objects

functions and modules

graphics, sound, and image I/O

arrays

conditionals and loops

Math text I/O

primitive data types assignment statements

create your own data types
Data Types

Data type. Set of values and operations on those values.

Primitive types. Values directly map to machine representation; ops directly map to machine instructions.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Set of Values</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>boolean</td>
<td>true, false</td>
<td>not, and, or, xor</td>
</tr>
<tr>
<td>int</td>
<td>$-2^{31}$ to $2^{31} - 1$</td>
<td>add, subtract, multiply</td>
</tr>
<tr>
<td>double</td>
<td>any of $2^{64}$ possible reals</td>
<td>add, subtract, multiply</td>
</tr>
</tbody>
</table>

We want to write programs that process other types of data.
- Colors, pictures, strings, input streams, ...
- Complex numbers, vectors, matrices, polynomials, ...
- Points, polygons, charged particles, celestial bodies, ...
Objects

**Object.** Holds a data type value; variable name refers to object.

**Object-oriented programming.**
- Create your own data types (set of values and ops on them).
- Use them in your programs (manipulate objects that hold values).

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Set of Values</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Color</td>
<td>24 bits</td>
<td>get red component, brighten</td>
</tr>
<tr>
<td>Picture</td>
<td>2D array of colors</td>
<td>get/set color of pixel (i, j)</td>
</tr>
<tr>
<td>String</td>
<td>sequence of characters</td>
<td>length, substring, compare</td>
</tr>
</tbody>
</table>

**This lecture.** Use existing data types.

**Next lecture.** Create your own data types.
Constructors and Methods

To construct a new object:
- Use keyword `new` (to invoke constructor).
- Use name of data type (to specify which type of object).

To apply an operation:
- Use name of object (to specify which object).
- Use the dot operator (to invoke method).
- Use the name of the method (to specify which operation).

```
String s;
s = new String("Hello, World");
System.out.println(s.substring(0, 5));
```
Image Processing
Color Data Type

**Color.** A sensation in the eye from electromagnetic radiation.

**Set of values.** [RGB representation] $256^3$ possible values, which quantify the amount of red, green, and blue, each on a scale of 0 to 255.

<table>
<thead>
<tr>
<th>R</th>
<th>G</th>
<th>B</th>
<th>Color</th>
</tr>
</thead>
<tbody>
<tr>
<td>255</td>
<td>0</td>
<td>0</td>
<td>Red</td>
</tr>
<tr>
<td>0</td>
<td>255</td>
<td>0</td>
<td>Green</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>255</td>
<td>Blue</td>
</tr>
<tr>
<td>255</td>
<td>255</td>
<td>255</td>
<td>White</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>Black</td>
</tr>
<tr>
<td>255</td>
<td>0</td>
<td>255</td>
<td>Pink</td>
</tr>
<tr>
<td>105</td>
<td>105</td>
<td>105</td>
<td>Gray</td>
</tr>
</tbody>
</table>
Color Data Type

**Color.** A sensation in the eye from electromagnetic radiation.

**Set of values.** [RGB representation] $256^3$ possible values, which quantify the amount of red, green, and blue, each on a scale of 0 to 255.

**API.** Application Programming Interface.

```java
public class java.awt.Color

    Color(int r, int g, int b)
    int getRed()       red intensity
    int getGreen()     green intensity
    int getBlue()      blue intensity
    Color brighter()   brighter version of this color
    Color darker()     darker version of this color
    String toString()  string representation of this color
    boolean equals(Color c)  is this color's value the same as c's?
```

http://download.oracle.com/javase/6/docs/api/java/awt/Color.html
Josef Albers. Revolutionized the way people think about color.

Homage to the Square by Josef Albers (1949-1975)
Josef Albers. Revolutionized the way people think about color.
Using Colors in Java

```java
import java.awt.Color;

class AlbersSquares {
    public static void main(String[] args) {
        int r1 = Integer.parseInt(args[0]);
        int g1 = Integer.parseInt(args[1]);
        int b1 = Integer.parseInt(args[2]);
        Color c1 = new Color(r1, g1, b1);

        int r2 = Integer.parseInt(args[3]);
        int g2 = Integer.parseInt(args[4]);
        int b2 = Integer.parseInt(args[5]);
        Color c2 = new Color(r2, g2, b2);

        StdDraw.setPenColor(c1);
        StdDraw.filledSquare(.25, .5, .2);
        StdDraw.setPenColor(c2);
        StdDraw.filledSquare(.25, .5, .1);

        StdDraw.setPenColor(c2);
        StdDraw.filledSquare(.75, .5, .2);
        StdDraw.setPenColor(c1);
        StdDraw.filledSquare(.75, .5, .1);
    }
}
```
Monochrome Luminance

Monochrome luminance. Effective brightness of a color.

NTSC formula. \( Y = 0.299r + 0.587g + 0.114b \).

```java
import java.awt.Color;

public class Luminance {
    public static double lum(Color c) {
        int r = c.getRed();
        int g = c.getGreen();
        int b = c.getBlue();
        return .299*r + .587*g + .114*b;
    }
}
```
**Color Compatibility**

**Q.** Which font colors will be most readable with which background colors on computer and cell phone screens?

**A.** Rule of thumb: difference in luminance should be $\geq 128$.

```java
public static boolean compatible(Color a, Color b) {
    return Math.abs(lum(a) - lum(b)) >= 128.0;
}
```
**Grayscale**

*Grayscale.* When all three R, G, and B values are the same, resulting color is on grayscale from 0 (black) to 255 (white).

**Convert to grayscale.** Use luminance to determine value.

```java
public static Color toGray(Color c) {
    int y = (int) Math.round(lum(c));
    Color gray = new Color(y, y, y);
    return gray;
}
```

**Bottom line.** We are writing programs that manipulate color.
Picture Data Type

**Raster graphics.** Basis for image processing.

**Set of values.** 2D array of color objects (pixels).

**API.**

```java
public class Picture {
    Picture(String filename) // create a picture from a file
    Picture(int w, int h) // create a blank w-by-h picture
    int width() // return the width of the picture
    int height() // return the height of the picture
    Color get(int x, int y) // return the color of pixel (x, y)
    void set(int x, int y, Color c) // set the color of pixel (x, y) to c
    void show() // display the image in a window
    void save(String filename) // save the image to a file
}
```
Image Processing: Grayscale Filter

**Goal.** Convert color image to grayscale according to luminance formula.

```java
import java.awt.Color;

public class Grayscale {
    public static void main(String[] args) {
        Picture pic = new Picture(args[0]);
        for (int x = 0; x < pic.width(); x++) {
            for (int y = 0; y < pic.height(); y++) {
                Color color = pic.get(x, y);
                Color gray = Luminance.toGray(color);
                pic.set(x, y, gray);
            }
        }
        pic.show();
    }
}
```
Image Processing: Grayscale Filter

Goal. Convert color image to grayscale according to luminance formula.

% java Grayscale mandrill.jpg

mandrill.jpg

% java Grayscale mandrill.jpg

mandrill.jpg
Image Processing: Scaling Filter

**Goal.** Shrink or enlarge an image to desired size.

**Downscaling.** To shrink, delete half the rows and columns.

**Upscaling.** To enlarge, replace each pixel by 4 copies.
Image Processing: Scaling Filter

**Goal.** Shrink or enlarge an image to desired size.

**Uniform strategy.** To convert from $w_s$-by-$h_s$ to $w_t$-by-$h_t$:

- Scale column index by $w_s / w_t$.
- Scale row index by $h_s / h_t$.
- Set color of pixel $(x, y)$ in target image to color of pixel $(x \times w_s / w_t, y \times h_s / h_t)$ in source image.
import java.awt.Color;

public class Scale {
    public static void main(String[] args) {
        String filename = args[0];
        int w = Integer.parseInt(args[1]);
        int h = Integer.parseInt(args[2]);
        Picture source = new Picture(filename);
        Picture target = new Picture(w, h);
        for (int tx = 0; tx < target.width(); tx++) {
            for (int ty = 0; ty < target.height(); ty++) {
                int sx = tx * source.width() / target.width();
                int sy = ty * source.height() / target.height();
                Color color = source.get(sx, sy);
                target.set(tx, ty, color);
            }
        }
        source.show();
        target.show();
    }
}
Image Processing: Scaling Filter

Scaling filter. Creates two Picture objects and two windows.

mandrill.jpg
(298-by-298)

% java Scale mandrill.jpg 400 200
More Image Processing Effects

RGB color separation

swirl filter
double filter
glass filter
Sobel edge detection
Text Processing
String Data Type

String data type. Basis for text processing.
Set of values. Sequence of Unicode characters.

API.

```java
public class String (Java string data type)

    String(String s)  // create a string with the same value as s
    int length()      // string length
    char charAt(int i) // i'th character
    String substring(int i, int j) // i'th through (j-1)st characters
    boolean contains(String sub) // does string contain sub as a substring?
    boolean startsWith(String pre) // does string start with pre?
    boolean endsWith(String post) // does string end with post?
    int indexOf(String p) // index of first occurrence of p
    int indexOf(String p, int i) // index of first occurrence of p after i
    String concat(String t) // this string with t appended
    int compareTo(String t) // string comparison
    String replaceAll(String a, String b) // result of changing as to bs
    String[] split(String delim) // strings between occurrences of delim
    boolean equals(String t) // is this string's value the same as t's?
```

http://download.oracle.com/javase/6/docs/api/java/lang/String.html
## Typical String Processing Code

**is the string a palindrome?**

| public static boolean isPalindrome(String s) {   |
| int N = s.length(); |
| for (int i = 0; i < N/2; i++) |
| if (s.charAt(i) != s.charAt(N-1-i)) |
| return false; |
| return true; |
|
**extract file name and extension from a command-line argument**

| String s = args[0]; |
| int dot = s.indexOf("."); |
| String base = s.substring(0, dot); |
| String extension = s.substring(dot + 1, s.length()); |
|
**print all lines in standard input that contain a string specified on the command line**

| String query = args[0]; |
| while (!StdIn.isEmpty()) |
| { |
| String s = StdIn.readLine(); |
| if (s.contains(query)) StdOut.println(s); |
| } |
|
**print all the hyperlinks (to educational institutions) in the text file on standard input**

| while (!StdIn.isEmpty()) |
| { |
| String s = StdIn.readString(); |
| if (s.startsWith("http://") && s.endsWith(".edu")) |
| StdOut.println(s); |
| } |
Gene Finding

Pre-genomics era. Sequence a human genome.
Post-genomics era. Analyze the data and understand structure.

Genomics. Represent genome as a string over \( \{ A, C, T, G \} \) alphabet.

Gene. A substring of genome that represents a functional unit.
- Preceded by \texttt{ATG}. \([\text{start codon}]\)
- Multiple of 3 nucleotides. \([\text{codons other than start/stop}]\)
- Succeeded by \texttt{TAG}, \texttt{TAA}, or \texttt{TGA}. \([\text{stop codons}]\)

Goal. Find all genes.
Gene Finding: Algorithm

**Algorithm.** Scan left-to-right through genome.

- If start codon, then set \( \text{beg} \) to index \( i \).
- If stop codon and substring is a multiple of 3
  - output gene
  - reset \( \text{beg} \) to -1

<table>
<thead>
<tr>
<th>( i )</th>
<th>codon</th>
<th>( \text{beg} )</th>
<th>gene</th>
<th>remaining portion of input string</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td></td>
<td>-1</td>
<td>ATAGATGCATAGCGCATAGCTAGATGTGCTAGC</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>TAG</td>
<td>-1</td>
<td>ATAGATGCATAGCGCATAGCTAGATGTGCTAGC</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>ATG</td>
<td>4</td>
<td>ATAGATGCATAGCGCATAGCTAGATGTGCTAGC</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>TAG</td>
<td>4</td>
<td>ATAGATGCATAGCGCATAGCTAGATGTGCTAGC</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>TAG</td>
<td>4</td>
<td>CATAGCGCA</td>
<td>ATAGATGCATAGCGCATAGCTAGATGTGCTAGC</td>
</tr>
<tr>
<td>20</td>
<td>TAG</td>
<td>-1</td>
<td>ATAGATGCATAGCGCATAGCTAGATGTGCTAGC</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>ATG</td>
<td>23</td>
<td>ATAGATGCATAGCGCATAGCTAGATGTGCTAGC</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>TAG</td>
<td>23</td>
<td>TGC</td>
<td>ATAGATGCATAGCGCATAGCTAGATGTGCTAGC</td>
</tr>
</tbody>
</table>
public class GeneFind {
    public static void main(String[] args) {
        String start = args[0];
        String stop = args[1];
        String genome = StdIn.readAll();

        int beg = -1;
        for (int i = 0; i < genome.length() - 2; i++) {
            String codon = genome.substring(i, i+3);
            if (codon.equals(start)) beg = i;
            if (codon.equals(stop) && beg != -1 && beg+3 < i) {
                String gene = genome.substring(beg+3, i);
                if (gene.length() % 3 == 0) {
                    StdOut.println(gene);
                    beg = -1;
                }
            }
        }
    }
}

% more genomeTiny.txt
ATAGATGCA|TAGCCCAA|TAGCTAGTGCTAGC

% java GeneFind ATG TAG < genomeTiny.txt
CATAGCGCA
TGC
Possible memory representation of a string.

- `genome = "aacaagttttacaagc";

| D0 | D1 | D2 | D3 | D4 | D5 | D6 | D7 | D8 | D9 | DA | DB | DC | DD | DE |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| a | a | c | a | a | g | t | t | t | a | c | a | a | g | c |

- `s = genome.substring(1, 5);`
- `t = genome.substring(9, 13);`

- `(s == t)` is false, but `(s.equals(t))` is true.

`s` and `t` refer to different strings that have the same value "acaa".

compares pointers

compares character sequences
In and Out
Bird's Eye View (Revisited)

input streams

standard input

pictures

drawings

command-line arguments

output streams

standard output
Non-Standard Input

Standard input. Read from terminal window.

Goal. Read from several different input streams.

In data type. Read text from stdin, a file, a web site, or network.

Ex: Are two text files identical?

```java
public class Diff {
    public static void main(String[] args) {
        In in0 = new In(args[0]);    // read from one file
        In in1 = new In(args[1]);    // read from another file
        String s = in0.ReadAll();
        String t = in1.ReadAll();
        StdOut.println(s.equals(t));
    }
}
```
Screen Scraping

Goal. Find current stock price of Google.

http://finance.yahoo.com/q?s=goog
Screen Scraping

**Goal.** Find current stock price of Google.
- `s.indexOf(t, i)`: index of first occurrence of pattern `t` in string `s`, starting at offset `i`.
- Find first string delimited by `<b>` and `</b>` after Last Trade.

```java
public class StockQuote {
    public static void main(String[] args) {
        String name = "http://finance.yahoo.com/q?s=";
        In in = new In(name + args[0]);
        String input = in.readAll();
        int start = input.indexOf("Last Trade:", 0);
        int from = input.indexOf("<b>", start);
        int to = input.indexOf("</b>", from);
        String price = input.substring(from + 3, to);
        StdOut.println(price);
    }
}
```

% java StockQuote goog
576.50
Object. Holds a data type value; variable name refers to object.

In Java, programs manipulate references to objects.
  - Exception: primitive types, e.g., boolean, int, double.
  - Reference types: String, Picture, Color, arrays, everything else.
  - OOP purist: language should not have separate primitive types.

Bottom line. We wrote programs that manipulate colors, pictures, and strings.

Next time. We'll write programs that manipulate our own abstractions.
Extra Slides
import java.awt.Color;
public class ColorSeparation {
    public static void main(String args[]) {
        Picture pic = new Picture(args[0]);
        int width = pic.width();
        int height = pic.height();

        Picture R = new Picture(width, height);
        Picture G = new Picture(width, height);
        Picture B = new Picture(width, height);

        for (int x = 0; x < width; x++) {
            for (int y = 0; y < height; y++) {
                Color c = pic.get(x, y);
                int r = c.getRed();
                int g = c.getGreen();
                int b = c.getBlue();
                R.set(x, y, new Color(r, 0, 0));
                G.set(x, y, new Color(0, g, 0));
                B.set(x, y, new Color(0, 0, b));
            }
        }
        R.show();
        G.show();
        B.show();
    }
}
Color Separation

ColorSeparation.java. Creates three Picture objects and windows.
Memory Management

Value types.
- Allocate memory when variable is declared.
- Can reclaim memory when variable goes out of scope.

Reference types.
- Allocate memory when object is created with new.
- Can reclaim memory when last reference goes out of scope.
- Significantly more challenging if several references to same object.

Garbage collector. System automatically reclaims memory; programmer relieved of tedious and error-prone activity.
Defining Data Types in Java

To define a data type, specify:
- Set of values.
- Operations defined on those values.

Java class. Defines a data type by specifying:
- **Instance variables.** (set of values)
- **Methods.** (operations defined on those values)
- **Constructors.** (create and initialize new objects)
Point Charge Data Type

Goal. Create a data type to manipulate point charges.

Set of values. Three real numbers. [position and electrical charge]

Operations.
- Create a new point charge at \((r_x, r_y)\) with electric charge \(q\).
- Determine electric potential \(V\) at \((x, y)\) due to point charge.
- Convert to string.

\[ V = k \frac{q}{r} \]

\(r =\) distance between \((x, y)\) and \((r_x, r_y)\)
\(k =\) electrostatic constant = \(8.99 \times 10^9\) N \(\cdot\) m² / C²
Point Charge Data Type

**Goal.** Create a data type to manipulate point charges.

**Set of values.** Three real numbers. [position and electrical charge]

**API.**

```java
public class Charge {
    Charge(double x0, double y0, double q0)
    double potentialAt(double x, double y)  // electric potential at (x, y) due to charge
    String toString()  // string representation
}
```
**Charge Data Type: A Simple Client**

**Client program.** Uses data type operations to calculate something.

```java
public static void main(String[] args) {
    double x = Double.parseDouble(args[0]);
    double y = Double.parseDouble(args[1]);
    Charge c1 = new Charge(.51, .63, 21.3);
    Charge c2 = new Charge(.13, .94, 81.9);
    double v1 = c1.potentialAt(x, y);
    double v2 = c2.potentialAt(x, y);
    StdOut.println(c1);
    StdOut.println(c2);
    StdOut.println(v1 + v2);
}
```

% java Charge .50 .50
21.3 at (0.51, 0.63)
81.9 at (0.13, 0.94)
2.74936907085912e12

Automagically invokes the `toString()` method
Anatomy of Instance Variables

Instance variables. Specifies the set of values.
- Declare outside any method.
- Always use access modifier `private`.
- Use modifier `final` with instance variables that never change.

```java
public class Charge {
    private final double rx, ry;
    private final double q;
}
```
Anatomy of a Constructor

**Constructor.** Specifies what happens when you create a new object.

```java
public Charge(double x0, double y0, double q0)
{
  rX = x0;
  ry = y0;
  q = q0;
}
```

**Calling a constructor.** Use `new` operator to create a new object.

```java
Charge c1 = new Charge(.51, .63, 21.3);
Charge c2 = new Charge(.13, .94, 81.9);
```
Anatomy of an Instance Method

**Instance method.** Define operations on instance variables.

```java
public double potentialAt(double x, double y) {
    double k = 8.99e09;
    double dx = x - rx;
    double dy = y - ry;
    return k * q / Math.sqrt(dx*dx + dy*dy);
}
```

**Invoking an instance method.** Use dot operator to invoke a method.

```java
double v1 = c1.potentialAt(x, y);
double v2 = c2.potentialAt(x, y);
```
Anatomy of a Class

```java
public class Charge {

    private final double rx, ry;
    private final double q;

    public Charge(double x0, double y0, double q0) {
        rx = x0; ry = y0; q = q0;
    }

    public double potentialAt(double x, double y) {
        double k = 8.99e09;
        double dx = x - rx;
        double dy = y - ry;
        return k * q / Math.sqrt(dx*dx + dy*dy);
    }

    public String toString() {
        return q + " at " + "(" + rx + ", " + ry + ")";
    }

    public static void main(String[] args) {
        double x = Double.parseDouble(args[0]);
        double y = Double.parseDouble(args[1]);
        Charge c1 = new Charge(.51, .63, 21.3);
        Charge c2 = new Charge(.13, .94, 81.9);
        double v1 = c1.potentialAt(x, y);
        double v2 = c2.potentialAt(x, y);
        System.out.printf("%.1e\n", (v1 + v2));
    }
}
```
Potential visualization. Read in N point charges from standard input; compute total potential at each point in unit square.

% more charges.txt
9
.51 .63  -100
.50 .50   40
.50 .72   10
.33 .33    5
.20 .20  -10
.70 .70   10
.82 .72   20
.85 .23   30
.90 .12  -50

% java Potential < charges.txt
Potential Visualization

Arrays of objects. Allocate memory for the array with `new`; then allocate memory for each individual object with `new`.

```java
// read in the data
int N = StdIn.readInt();
Charge[] a = new Charge[N];
for (int i = 0; i < N; i++) {
    double x0 = StdIn.readDouble();
    double y0 = StdIn.readDouble();
    double q0 = StdIn.readDouble();
    a[i] = new Charge(x0, y0, q0);
}
```
Potential Visualization

```java
// plot the data
int SIZE = 512;
Picture pic = new Picture(SIZE, SIZE);
for (int i = 0; i < SIZE; i++) {
    for (int j = 0; j < SIZE; j++) {
        double V = 0.0;
        for (int k = 0; k < N; k++) {
            double x = 1.0 * i / SIZE;
            double y = 1.0 * j / SIZE;
            V += a[k].potentialAt(x, y);
        }
        Color color = getColor(V);
        pic.set(i, SIZE-1-j, color);
    }
}
pic.show();
```

\[ V = \sum_{k} \left( k \frac{q_k}{r_k} \right) \]

- Compute color as a function of potential \( V \)
- \((0, 0)\) is upper left
Turtle Graphics
Turtle Graphics

Goal. Create a data type to manipulate a turtle moving in the plane.

Set of values. Location and orientation of turtle.

API. public class Turtle

Turtle(double x0, double y0, double a0) create a new turtle at \((x_0, y_0)\) facing \(a_0\) degrees counterclockwise from the x-axis

void turnLeft(double delta) rotate \(\delta\) degrees counterclockwise

void goForward(double step) move distance \(\text{step}\), drawing a line

// draw a square
Turtle turtle = new Turtle(0.0, 0.0, 0.0);
turtle.goForward(1.0);
turtle.turnLeft(90.0);
turtle.goForward(1.0);
turtle.turnLeft(90.0);
turtle.goForward(1.0);
turtle.turnLeft(90.0);
turtle.goForward(1.0);
turtle.turnLeft(90.0);
public class Turtle {
    private double x, y; // turtle is at (x, y)
    private double angle; // facing this direction

    public Turtle(double x0, double y0, double a0) {
        x = x0;
        y = y0;
        angle = a0;
    }

    public void turnLeft(double delta) {
        angle += delta;
    }

    public void goForward(double d) {
        double oldx = x;
        double oldy = y;
        x = x + d * Math.cos(Math.toRadians(angle));
        y = y + d * Math.sin(Math.toRadians(angle));
        StdDraw.line(oldx, oldy, x, y);
    }
}

Turtle trigonometry
public class Ngon {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        double angle = 360.0 / N;
        double step = Math.sin(Math.toRadians(angle/2.0));
        Turtle turtle = new Turtle(0.5, 0, angle/2.0);
        for (int i = 0; i < N; i++) {
            turtle.goForward(step);
            turtle.turnLeft(angle);
        }
    }
}
public class Spiral {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        double decay = Double.parseDouble(args[1]);
        double angle = 360.0 / N;
        double step = Math.sin(Math.toRadians(angle / 2.0));
        Turtle turtle = new Turtle(0.5, 0, angle / 2.0);
        for (int i = 0; i < 10 * N; i++) {
            step /= decay;
            turtle.goForward(step);
            turtle.turnLeft(angle);
        }
    }
}
Spira Mirabilis in Nature
Complex Numbers
Complex Number Data Type

**Goal.** Create a data type to manipulate complex numbers.

**Set of values.** Two real numbers: real and imaginary parts.

**API.**

```java
public class Complex
{
    Complex(double real, double imag)
    Complex plus(Complex b) // sum of this number and b
    Complex times(Complex b) // product of this number and b
    double abs() // magnitude
    String toString() // string representation
}
```

\[
a = 3 + 4i, \quad b = -2 + 3i\\
a + b = 1 + 7i\\
a \times b = -18 + i\\
|a| = 5
\]
Relevance. A quintessential mathematical abstraction.

Applications.
- Fractals.
- Impedance in RLC circuits.
- Signal processing and Fourier analysis.
- Control theory and Laplace transforms.
- Quantum mechanics and Hilbert spaces.
- ...

Applications of Complex Numbers
Complex Number Data Type: A Simple Client

Client program. Uses data type operations to calculate something.

```java
public static void main(String[] args) {
    Complex a = new Complex(3.0, 4.0);
    Complex b = new Complex(-2.0, 3.0);
    Complex c = a.times(b);
    StdOut.println("a = " + a);
    StdOut.println("b = " + b);
    StdOut.println("c = " + c);
}
```

% java TestClient
a = 3.0 + 4.0i
b = -2.0 + 3.0i
c = -18.0 + 1.0i

Remark. Can't write c = a * b since no operator overloading in Java.
public class Complex {

private final double re;
private final double im;  

public Complex(double real, double imag) {
    re = real;
    im = imag;
}

public String toString() { return re + " + " + im + "i"; }

double abs() { return Math.sqrt(re*re + im*im); }

public Complex plus(Complex b) {
    double real = re + b.re;
    double imag = im + b.im;
    return new Complex(real, imag);
}

public Complex times(Complex b) {
    double real = re * b.re - im * b.im;
    double imag = re * b.im + im * b.re;
    return new Complex(real, imag);
}
}

Complex Number Data Type: Implementation
Mandelbrot set. A set of complex numbers.

Plot. Plot \((x, y)\) black if \(z = x + y i\) is in the set, and white otherwise.

- No simple formula describes which complex numbers are in set.
- Instead, describe using an algorithm.
**Mandelbrot Set**

**Mandelbrot set.** Is complex number \( z_0 \) in the set?

- Iterate \( z_{t+1} = (z_t)^2 + z_0 \).
- If \( |z_t| \) diverges to infinity, then \( z_0 \) is not in set; otherwise \( z_0 \) is in set.

\[
\begin{array}{|c|c|}
\hline
\tau & Z_\tau \\
\hline
0 & -1/2 + 0i \\
1 & -1/4 + 0i \\
2 & -7/16 + 0i \\
3 & -79/256 + 0i \\
4 & -26527/65536 + 0i \\
5 & -1443801919/4294967296 + 0i \\
\hline
\end{array}
\]

\( z = -1/2 \) is in Mandelbrot set

\[
\begin{array}{|c|c|}
\hline
\tau & Z_\tau \\
\hline
0 & 1 + i \\
1 & 1 + 3i \\
2 & -7 + 7i \\
3 & 1 - 97i \\
4 & -9407 - 193i \\
5 & 88454401 + 3631103i \\
\hline
\end{array}
\]

\( z = 1 + i \) not in Mandelbrot set
Practical issues.
- Cannot plot infinitely many points.
- Cannot iterate infinitely many times.

Approximate solution.
- Sample from an $N$-by-$N$ grid of points in the plane.
- Fact: if $|z_t| > 2$ for any $t$, then $z$ not in Mandelbrot set.
- Pseudo-fact: if $|z_{255}| \leq 2$ then $z$ "likely" in Mandelbrot set.
Mandelbrot function with complex numbers.

- Is $z_0$ in the Mandelbrot set?
- Returns white (definitely no) or black (probably yes).

```java
public static Color mand(Complex z0) {
    Complex z = z0;
    for (int t = 0; t < 255; t++) {
        if (z.abs() > 2.0) return StdDraw.WHITE;
        z = z.times(z);
        z = z.plus(z0);
    }
    return StdDraw.BLACK;
}
```

More dramatic picture: replace `StdDraw.WHITE` with grayscale or color.

```
new Color(255-t, 255-t, 255-t)
```
Plot the Mandelbrot set in gray scale.

```java
public static void main(String[] args) {
    double xc = Double.parseDouble(args[0]);
    double yc = Double.parseDouble(args[1]);
    double size = Double.parseDouble(args[2]);
    int N = 512;
    Picture pic = new Picture(N, N);

    for (int i = 0; i < N; i++) {
        for (int j = 0; j < N; j++) {
            double x0 = xc - size/2 + size*i/N;
            double y0 = yc - size/2 + size*j/N;
            Complex z0 = new Complex(x0, y0);
            Color color = mand(z0);
            pic.set(i, N-1-j, color);
        }
    }
    pic.show();
}
```
Mandelbrot Set

% java Mandelbrot -.5 0 2

% java Mandelbrot .1045 -.637 .01
Mandelbrot Set

% java ColorMandelbrot -.5 0 2 < mandel.txt
Mandelbrot Set Music Video

http://www.jonathancoulton.com/songdetails/Mandelbrot_Set
Applications of Data Types

Data type. Set of values and collection of operations on those values.

Simulating the physical world.
- Java objects model real-world objects.
- Not always easy to make model reflect reality.
  - Ex: charged particle, molecule, COS 126 student, ....

Extending the Java language.
- Java doesn't have a data type for every possible application.
- Data types enable us to add our own abstractions.
  - Ex: complex, vector, polynomial, matrix, ....
3.2 Extra Slides
Example: Bouncing Ball in Unit Square

Bouncing ball. Model a bouncing ball moving in the unit square with constant velocity.
public class Ball {

    private double rx, ry; ← instance variables
    private double vx, vy;
    private double radius;

    public Ball() {
        rx = ry = 0.5;
        vx = 0.015 - Math.random() * 0.03;
        vy = 0.015 - Math.random() * 0.03;
        radius = 0.01 + Math.random() * 0.01;
    }

    public void move() {
        if ((rx + vx > 1.0) || (rx + vx < 0.0)) vx = -vx;
        if ((ry + vy > 1.0) || (ry + vy < 0.0)) vy = -vy;
        rx = rx + vx;
        ry = ry + vy;
    }

    public void draw() {
        StdDraw.filledCircle(rx, ry, radius);
    }
}

Example: Bouncing Ball in Unit Square
Object References

Object reference.
- Allow client to manipulate an object as a single entity.
- Essentially a machine address (pointer).

```
Ball b1 = new Ball();
b1.move();
b1.move();

Ball b2 = new Ball();
b2.move();

b2 = b1;
b2.move();
```
Object References

Object reference.
- Allow client to manipulate an object as a single entity.
- Essentially a machine address (pointer).

```
Ball b1 = new Ball();
b1.move();
b1.move();

Ball b2 = new Ball();
b2.move();

b2 = b1;
b2.move();
```
Object References

Object reference.
- Allow client to manipulate an object as a single entity.
- Essentially a machine address (pointer).

```java
Ball b1 = new Ball();
b1.move();
b1.move();

Ball b2 = new Ball();
b2.move();

b2 = b1;
b2.move();
```
Object References

Object reference.
- Allow client to manipulate an object as a single entity.
- Essentially a machine address (pointer).

```
Ball b1 = new Ball();
b1.move();
b1.move();

Ball b2 = new Ball();
b2.move();
b2 = b1;
b2.move();
```
Object References

Object reference.
- Allow client to manipulate an object as a single entity.
- Essentially a machine address (pointer).

```java
Ball b1 = new Ball();
b1.move();
b1.move();

Ball b2 = new Ball();
b2.move();
b2 = b1;
b2.move();
```

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>0.60</td>
</tr>
<tr>
<td>C1</td>
<td>0.52</td>
</tr>
<tr>
<td>C2</td>
<td>0.05</td>
</tr>
<tr>
<td>C3</td>
<td>0.01</td>
</tr>
<tr>
<td>C4</td>
<td>0.03</td>
</tr>
<tr>
<td>C5</td>
<td>0</td>
</tr>
<tr>
<td>C6</td>
<td>0</td>
</tr>
<tr>
<td>C7</td>
<td>0.50</td>
</tr>
<tr>
<td>C8</td>
<td>0.50</td>
</tr>
<tr>
<td>C9</td>
<td>0.07</td>
</tr>
<tr>
<td>CA</td>
<td>0.04</td>
</tr>
<tr>
<td>CB</td>
<td>0.04</td>
</tr>
<tr>
<td>CC</td>
<td>0</td>
</tr>
</tbody>
</table>

Registers main memory (64-bit machine)
Object References

Object reference.
- Allow client to manipulate an object as a single entity.
- Essentially a machine address (pointer).

```java
Ball b1 = new Ball();
b1.move();
b1.move();

Ball b2 = new Ball();
b2.move();

b2 = b1;
b2.move();
```

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>0.60</td>
</tr>
<tr>
<td>C1</td>
<td>0.52</td>
</tr>
<tr>
<td>C2</td>
<td>0.05</td>
</tr>
<tr>
<td>C3</td>
<td>0.01</td>
</tr>
<tr>
<td>C4</td>
<td>0.03</td>
</tr>
<tr>
<td>C5</td>
<td>0</td>
</tr>
<tr>
<td>C6</td>
<td>0</td>
</tr>
<tr>
<td>C7</td>
<td>0.57</td>
</tr>
<tr>
<td>C8</td>
<td>0.54</td>
</tr>
<tr>
<td>C9</td>
<td>0.07</td>
</tr>
<tr>
<td>CA</td>
<td>0.04</td>
</tr>
<tr>
<td>CB</td>
<td>0.04</td>
</tr>
<tr>
<td>CC</td>
<td>0</td>
</tr>
</tbody>
</table>

Registers | Main memory (64-bit machine)
Object References

Object reference.
- Allow client to manipulate an object as a single entity.
- Essentially a machine address (pointer).

```java
Ball b1 = new Ball();
b1.move();
b1.move();

Ball b2 = new Ball();
b2.move();

b2 = b1;
b2.move();
```

Data stored in C7 - CB for abstract bit recycler.

<table>
<thead>
<tr>
<th>addr</th>
<th>value</th>
</tr>
</thead>
<tbody>
<tr>
<td>C0</td>
<td>0.60</td>
</tr>
<tr>
<td>C1</td>
<td>0.52</td>
</tr>
<tr>
<td>C2</td>
<td>0.05</td>
</tr>
<tr>
<td>C3</td>
<td>0.01</td>
</tr>
<tr>
<td>C4</td>
<td>0.03</td>
</tr>
<tr>
<td>C5</td>
<td>0</td>
</tr>
<tr>
<td>C6</td>
<td>0</td>
</tr>
<tr>
<td>C7</td>
<td>0.57</td>
</tr>
<tr>
<td>C8</td>
<td>0.54</td>
</tr>
<tr>
<td>C9</td>
<td>0.07</td>
</tr>
<tr>
<td>CA</td>
<td>0.04</td>
</tr>
<tr>
<td>CB</td>
<td>0.04</td>
</tr>
<tr>
<td>CC</td>
<td>0</td>
</tr>
</tbody>
</table>

Registers | Main Memory (64-bit machine)
Object References

Object reference.
- Allow client to manipulate an object as a single entity.
- Essentially a machine address (pointer).

```java
Ball b1 = new Ball();
b1.move();
b1.move();

Ball b2 = new Ball();
b2.move();

b2 = b1;
b2.move();
```

Moving `b2` also moves `b1` since they are aliases that reference the same object.
Creating Many Objects

Each object is a data type value.
- Use `new` to invoke constructor and create each one.
- Ex: create N bouncing balls and animate them.

```java
public class BouncingBalls {
    public static void main(String[] args) {
        int N = Integer.parseInt(args[0]);
        Ball balls[] = new Ball[N];
        for (int i = 0; i < N; i++)
            balls[i] = new Ball();

        while(true) {
            StdDraw.clear();
            for (int i = 0; i < N; i++) {
                balls[i].move();
                balls[i].draw();
            }
            StdDraw.show(20);
        }
    }
}
```
50 Bouncing Balls

**Color.** Associate a color with each ball; paint background black.

```
% java BouncingBalls 50
```

**Scientific variations.** Account for gravity, spin, collisions, drag, ...
Reference. Variable that stores the name of a thing.

<table>
<thead>
<tr>
<th>Thing</th>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Web page</td>
<td><a href="http://www.princeton.edu">www.princeton.edu</a></td>
</tr>
<tr>
<td>Bank account</td>
<td>45-234-23310076</td>
</tr>
<tr>
<td>Word of TOY memory</td>
<td>1C</td>
</tr>
<tr>
<td>Byte of computer memory</td>
<td>00FACADE</td>
</tr>
<tr>
<td>Home</td>
<td>35 Olden Street</td>
</tr>
</tbody>
</table>

Some consequences.
- Assignment statements copy references (not objects).
- The == operator tests if two references refer to same object.
- Pass copies of references (not objects) to functions.
  - efficient since no copying of data
  - function can change the object
Using a Data Type in Java

**Client.** A sample client program that uses the `Point` data type.

```java
public class PointTest {
    public static void main(String[] args) {
        Point a = new Point();
        Point b = new Point();
        double distance = a.distanceTo(b);
        StdOut.println("a = " + a);
        StdOut.println("b = " + b);
        StdOut.println("distance = " + distance);
    }
}
```

% java PointTest
```
a = (0.716810971264761, 0.0753539063358446)
b = (0.4052136795358151, 0.033848435224524076)
distance = 0.31434944941098036
```
Points in the Plane

Data type. Points in the plane.

```java
public class Point {
    private double x;
    private double y;

    public Point() {
        x = Math.random();
        y = Math.random();
    }

    public String toString() {
        return "(" + x + ", " + y + ")";
    }

    public double distanceTo(Point p) {
        double dx = x - p.x;
        double dy = y - p.y;
        return Math.sqrt(dx*dx + dy*dy);
    }
}
```
A Compound Data Type: Circles

**Goal.** Data type for circles in the plane.

```java
public class Circle {
    private Point center;
    private double radius;

    public Circle(Point center, double radius) {
        this.center = center;
        this.radius = radius;
    }

    public boolean contains(Point p) {
        return p.dist(center) <= radius;
    }

    public double area() {
        return Math.PI * radius * radius;
    }

    public boolean intersects(Circle c) {
        return center.dist(c.center) <= radius + c.radius;
    }
}
```
Arguments to methods are always passed by value.
- Primitive types: passes copy of value of actual parameter.
- Objects: passes copy of reference to actual parameter.

```
public class PassByValue {
    static void update(int a, int[] b, String c) {
        a = 7;
        b[3] = 7;
        c = "seven";
        StdOut.println(a + " " + b[3] + " " + c);
    }
    public static void main(String[] args) {
        int a = 3;
        int[] b = { 0, 1, 2, 3, 4, 5 };
        String c = "three";
        StdOut.println(a + " " + b[3] + " " + c);
        update(a, b, c);
        StdOut.println(a + " " + b[3] + " " + c);
    }
}
```

% java PassByValue
3 3 three
7 7 seven
3 7 three
**Object Oriented Programming**

**Procedural programming.** [verb-oriented]
- Tell the computer to do this.
- Tell the computer to do that.

**OOP philosophy.** Software is a *simulation* of the real world.
- We know (approximately) how the real world works.
- Design software to model the real world.

**Objected oriented programming (OOP).** [noun-oriented]
- Programming paradigm based on data types.
- Identify *objects* that are part of the problem domain or solution.
  - **Identity:** objects are distinguished from other objects (references).
  - **State:** objects know things (instance variables).
  - **Behavior:** objects do things (methods).
Alan Kay

**Alan Kay.** [Xerox PARC 1970s]

- Invented Smalltalk programming language.
- Conceived Dynabook portable computer.
- Ideas led to: laptop, modern GUI, OOP.

“*The computer revolution hasn't started yet.*”

“The best way to predict the future is to invent it.”

“If you don't fail at least 90 per cent of the time, you're not aiming high enough.”

— Alan Kay
Encapsulation
Encapsulation

Data type. Set of values and operations on those values.
Ex. int, String, Complex, Vector, Document, GuitarString, ...

Encapsulated data type. Hide internal representation of data type.

Separate implementation from design specification.
- Class provides data representation and code for operations.
- Client uses data type as black box.
- API specifies contract between client and class.

Bottom line. You don't need to know how a data type is implemented in order to use it.
Intuition

Client

API
- volume
- change channel
- adjust picture
- decode NTSC signal

Implementation
- cathode ray tube
- electron gun
- Sony Wega 36XBR250
- 241 pounds

client needs to know how to use API

implementation needs to know what API to implement

Implementation and client need to agree on API ahead of time.
Intuition

Client

API
- volume
- change channel
- adjust picture
- decode NTSC signal

Implementation
- gas plasma monitor
- Samsung FPT-6374
- wall mountable
- 4 inches deep

client needs to know how to use API
implementation needs to know what API to implement

Can substitute better implementation without changing the client.
Counter Data Type

Counter. Data type to count electronic votes.

```java
public class Counter {
    public int count;
    public final String name;

    public Counter(String id) {
        name = id;
    }

    public void increment() {
        count++;
    }

    public int value() {
        return count;
    }
}
```

Legal Java client.

Counter c = new Counter("Volusia County");
c.count = -16022;

Oops. Al Gore receives -16,022 votes in Volusia County, Florida.
Counter Data Type

**Counter. Encapsulated data type to count electronic votes.**

```java
public class Counter {
    private int count;
    private final String name;

    public Counter(String id) { name = id; }
    public void increment() { count++; }
    public int value() { return count; }
}
```

Does not compile.

```java
Counter c = new Counter("Volusia County");
c.count = -16022;
```

**Benefit.** Can guarantee that each data type value remains in a consistent state.
Changing Internal Representation

**Encapsulation.**
- Keep data representation hidden with `private` access modifier.
- Expose API to clients using `public` access modifier.

```java
public class Complex {
    private final double re, im;

    public Complex(double re, double im) { ... }
    public double abs() { ... }
    public Complex plus(Complex b) { ... }
    public Complex times(Complex b) { ... }
    public String toString() { ... }
}
```

**Advantage.** Can switch internal representation without changing client.

**Note.** All our data types are already encapsulated!

E.g., to polar coordinates
Time Bombs

Internal representation changes.

- [VIN numbers] We'll run out by 2010.

Lesson. By exposing data representation to client, might need to sift through millions of lines of code in client to update.
Encapsulated data types.
- Don't touch data and do whatever you want.
- Instead, ask object to manipulate its data.

"Ask, don't touch."

Adele Goldberg
Former president of ACM
Co-developed Smalltalk

Lesson. Limiting scope makes programs easier to maintain and understand.

"principle of least privilege"
Immutability
Immutability

**Immutable data type.** Object's value cannot change once constructed.

<table>
<thead>
<tr>
<th>mutable</th>
<th>immutable</th>
</tr>
</thead>
<tbody>
<tr>
<td>Picture</td>
<td>Charge</td>
</tr>
<tr>
<td>Histogram</td>
<td>Color</td>
</tr>
<tr>
<td>Turtle</td>
<td>Stopwatch</td>
</tr>
<tr>
<td>StockAccount</td>
<td>Complex</td>
</tr>
<tr>
<td>Counter</td>
<td>String</td>
</tr>
<tr>
<td>Java arrays</td>
<td>primitive types</td>
</tr>
</tbody>
</table>
Immutability: Advantages and Disadvantages

Immutable data type. Object's value cannot change once constructed.

Advantages.
- Avoid aliasing bugs.
- Makes program easier to debug.
- Limits scope of code that can change values.
- Pass objects around without worrying about modification.

Disadvantage. New object must be created for every value.
Final Access Modifier

**Final.** Declaring an instance variable to be *final* means that you can assign it a value only once, in initializer or constructor.

```java
public class Counter {
    private final String name;
    private int count;
    ...
}
```

**Advantages.**
- Helps enforce immutability.
- Prevents accidental changes.
- Makes program easier to debug.
- Documents that the value cannot change.
Spatial Vectors
## Vector Data Type

**Set of values.** Sequence of real numbers. [Cartesian coordinates]

### API.

```java
public class Vector {
    // Create a vector with the given Cartesian coordinates
    public Vector(double[] a) {
        // Implementation
    }

    // Sum of this vector and b
    public Vector plus(Vector b) {
        // Implementation
    }

    // Difference of this vector and b
    public Vector minus(Vector b) {
        // Implementation
    }

    // Scalar product of this vector and t
    public Vector times(double t) {
        // Implementation
    }

    // Dot product of this vector and b
    public double dot(Vector b) {
        // Implementation
    }

    // Magnitude of this vector
    public double magnitude() {
        // Implementation
    }

    // Unit vector with same direction as this vector
    public Vector direction() {
        // Implementation
    }
}
```

Example:

- \( x = (0, 3, 4, 0), \ y = (0, -3, 1, -4) \)
- \( x + y = (0, 0, 5, -4) \)
- \( 3x = (0, 9, 12, 0) \)
- \( x \cdot y = (0 \times 0) + (3 \times -3) + (4 \times 1) + (0 \times -4) = -5 \)
- \( |x| = \sqrt{0^2 + 3^2 + 4^2 + 0^2} = 5 \)
- \( x = x / |x| = (0, 0.6, 0.8, 0) \)
Vector Data Type Applications

Relevance. A quintessential mathematical abstraction.

Applications.
- Statistics.
- Linear algebra.
- Clustering and similarity search.
- Force, velocity, acceleration, momentum, torque.
- ...

...
Vector Data Type: Implementation

```java
public class Vector {
    private int N;
    private double[] coords;

    // constructor
    public Vector(double[] a) {
        N = a.length;
        coords = new double[N];
        for (int i = 0; i < N; i++)
            coords[i] = a[i];
    }

    // dot product method
    public double dot(Vector b) {
        double sum = 0.0;
        for (int i = 0; i < N; i++)
            sum += (coords[i] * b.coords[i]);
        return sum;
    }

    // addition method
    public Vector plus(Vector b) {
        double[] c = new double[N];
        for (int i = 0; i < N; i++)
            c[i] = coords[i] + b.coords[i];
        return new Vector(c);
    }
}
```
public Vector times(double t) {
    double[] c = new double[N];
    for (int i = 0; i < N; i++)
        c[i] = t * coords[i];
    return new Vector(c);
}

public double magnitude() {
    return Math.sqrt(this.dot(this));
}

public Vector direction() {
    return this.times(1.0 / this.magnitude());
}
...

This. The keyword this is a reference to the invoking object.
Ex. When you invoke a.magnitude(), this is an alias for a.
N-body Simulation
**Goal.** Determine the motion of $N$ particles, moving under their mutual Newtonian gravitational forces.

**Ex.** Planets orbit the sun.
N-Body: Applications

Applications to astrophysics.

- Orbits of solar system bodies.
- Stellar dynamics at the galactic center.
- Stellar dynamics in a globular cluster.
- Stellar dynamics during the collision of two galaxies.
- Formation of structure in the universe.
- Dynamics of galaxies during cluster formation.
N-Body Problem

**Goal.** Determine the motion of $N$ particles, moving under their mutual Newtonian gravitational forces.

**Context.** Newton formulated the physical principles in *Principia*.

\[ F = m a \quad F = \frac{G m_1 m_2}{r^2} \]

*Newton's second law of motion*  
*Newton's law of universal gravitation*

Kepler  Bernoulli  Newton  Euler  Lagrange  Delaunay  Poincaré
2-Body Problem

2 body problem.
- Can be solved analytically via Kepler's 3\textsuperscript{rd} law.
- Bodies move around a common barycenter (center-of-mass) with elliptical orbits.
3-body problem. No solution possible in terms of elementary functions; moreover, orbits may not be stable or periodic!

Consequence. Must resort to computational methods.
N-Body Simulation

N-body simulation. The ultimate object-oriented program: simulate the universe.
Body Data Type

**Body data type.** Represent a particle.

```java
public class Body {
    private Vector r; // position
    private Vector v; // velocity
    private double mass; // mass

    public Body(Vector r, Vector v, double mass) {
    }

    public void move(Vector f, double dt) {
        // apply force f, move body for dt seconds
    }

    public void draw() {
        // draw the ball
    }

    public Vector forceFrom(Body b) {
        // force vector between this body and b
    }
}
```

**Vector notation.** Represent position, velocity, and force using `Vector`.

```java
public class Body {
    private Vector r; // position
    private Vector v; // velocity
    private double mass; // mass

    // instance variables
```
Moving a body. Assuming no other forces, body moves in straight line.

\[ r = r + \text{v} \times \text{dt}; \]

\[
\begin{align*}
    r_x &= r_x + dt \cdot v_x \\
    r_y &= r_y + dt \cdot v_y
\end{align*}
\]
Moving a Body

Moving a body.
- Given external force \( F \), acceleration \( a = F/m \).
- Use acceleration (assume fixed) to compute change in velocity.
- Use velocity to compute change in position.

\[
\begin{align*}
\text{time } t & & \text{time } t+1 \\
\text{new position is vector sum of } & \text{new velocity is vector sum of} \\
\text{old position and velocity} & \text{old velocity and acceleration} \\
\end{align*}
\]

Vector \( a = f \times (1/\text{mass}); \)
\( v = v + a \times (\text{dt}); \)
\( r = r + v \times (\text{dt}); \)
Force Between Two Bodies

Newton’s law of universal gravitation.

- $F = G \frac{m_1 m_2}{r^2}$.
- Direction of force is line between two particles.

```java
double G = 6.67e-11;
Vector delta = a.r.minus(b.r);
double dist = delta.magnitude();
double F = (G * a.mass * b.mass) / (dist * dist);
Vector force = delta.direction().times(F);
```
public class Body {
    private Vector r;       // position
    private Vector v;       // velocity
    private double mass;    // mass

    public Body(Vector r, Vector v, double mass) {
        this.r = r;
        this.v = v;
        this.mass = mass;
    }

    public void move(Vector f, double dt) {
        Vector a = f.times(1/mass);
        v = v.plus(a.times(dt));
        r = r.plus(v.times(dt));
    }

    public Vector forceFrom(Body that) {
        double G = 6.67e-11;
        Vector delta = that.r.minus(this.r);
        double dist = delta.magnitude();
        double F = (G * this.mass * that.mass) / (dist * dist);
        return delta.direction().times(F);
    }

    public void draw() {
        StdDraw.setPenRadius(0.025);
        StdDraw.point(r.cartesian(0), r.cartesian(1));
    }
}
Universe Data Type

**Universe data type.** Represent a universe of N particles.

```java
public class Universe {

    public Universe() {
    }

    void increaseTime(double dt) {
        // simulate the passing of dt seconds
    }

    void draw() {
        // draw the universe
    }

    public static void main(String[] args) {
        Universe newton = new Universe();
        double dt = Double.parseDouble(args[0]);
        while (true) {
            StdDraw.clear();
            newton.increaseTime(dt);
            newton.draw();
            StdDraw.show(10);
        }
    }
}
```

**main simulation loop**
Universe Data Type

**Universe data type.** Represent a universe of N particles.

```java
public class Universe {
    private double radius; // radius of universe
    private int N; // number of particles
    private Body[] orbs; // the bodies
    public Universe() {
    }
    void increaseTime(double dt) {
        // simulate the passing of dt seconds
    }
    void draw() {
        // draw the universe
    }
}
```

*instance variables*
Data-Driven Design

File format.

```
% more 4body.txt

4 ← N

5.0e10 ← radius

-3.5e10 0.0e00 0.0e00 1.4e03 3.0e28
-1.0e10 0.0e00 0.0e00 1.4e04 3.0e28
1.0e10 0.0e00 0.0e00 -1.4e04 3.0e28
3.5e10 0.0e00 0.0e00 -1.4e03 3.0e28
```

Constructor.

```
public Universe() {
    N = StdIn.readInt();
    radius = StdIn.readDouble();
    StdDraw.setXscale(-radius, +radius);
    StdDraw.setYscale(-radius, +radius);

    // read in the N bodies
    orbs = new Body[N];
    for (int i = 0; i < N; i++) {
        double rx   = StdIn.readDouble();
        double ry   = StdIn.readDouble();
        double vx   = StdIn.readDouble();
        double vy   = StdIn.readDouble();
        double mass = StdIn.readDouble();
        double[] position = { rx, ry };
        double[] velocity = { vx, vy };
        Vector r = new Vector(position);
        Vector v = new Vector(velocity);
        orbs[i] = new Body(r, v, mass);
    }
}
```
Principle of Superposition

**Principle of superposition.** Net gravitational force acting on a body is the sum of the individual forces.

\[
F_i = \sum_{i \neq j} \frac{G m_i m_j}{|r_i - r_j|^2}
\]
public class Universe {
    private final double radius; // radius of universe
    private final int N; // number of bodies
    private final Body[] orbs; // array of N bodies

    public Universe() { /* see previous slide */ }

    public void increaseTime(double dt) {
        Vector[] f = new Vector[N];
        for (int i = 0; i < N; i++)
            f[i] = new Vector(new double[2]);
        for (int i = 0; i < N; i++)
            for (int j = 0; j < N; j++)
                if (i != j)
                    f[i] = f[i].plus(orbs[j].forceTo(orbs[i]));
        for (int i = 0; i < N; i++)
            orbs[i].move(f[i], dt);
    }

    public void draw() {
        for (int i = 0; i < N; i++)
            orbs[i].draw();
    }

    public static void main(String[] args) { /* see previous slide */ }
}
Odds and Ends

Accuracy. How small to make $dt$? How to avoid floating-point inaccuracies from accumulating?

Efficiency.
- Direct sum: takes time proportional to $N^2$\;\Rightarrow\;not\;usable\;for\;large\;N.
- Appel / Barnes-Hut: takes time proportional to $N \log N$ time\;\Rightarrow\;can\;simulate\;large\;universes.

3D universe. Use a 3D vector (only drawing code changes!).

Collisions.
- Model inelastic collisions.
- Use a softening parameter to avoid collisions.

\[
F_i = \sum_{i \neq j} \frac{G m_i m_j}{|r_i - r_j|^2 + \epsilon^2}
\]
Extra Slides
N-Body Simulation

1. Setup initial distribution of particles.
   - Need accurate data and model of mass distribution.

2. Compute forces between particles.
   - Direct sum: $N^2$.
   - Appel / Barnes-Hut: $N \log N$.

\[ F_i = \sum_{i \neq j} \frac{Gm_i m_j}{|r_i - r_j|^2 + \epsilon^2} \]

- $\epsilon$ = softening parameter eliminates binary stars with $r < \epsilon$.
- Hard binaries can be important source of energy.

3. Evolve particles using ODE solver.
   - Leapfrog method balances efficiency and accuracy.
   - Truncation error = $O(dt^2)$.
   - Symplectic.

\[ \frac{dX_i}{dt} = V_i \]
\[ m_i \frac{dV_i}{dt} = F_i \]

4. Display and analyze results.
Solving the force problem with hardware.

**GRAPE-6.** Special purpose hardware to compute force.

---

*Jun Makino, U. Tokyo*
Do we really need to compute force from every star for distant objects?

Andromeda – 2 million light years away
Solving the force problem with software – tree codes

Distance = 25 times size

Viewing the Andromeda Galaxy from Earth

D

Earth

r = distance to center of mass

x = location of center of mass

Andromeda
Organize particles into a tree. In Barnes-Hut algorithm, use a quadtree in 2D

A Complete Quadtree with 4 Levels