Introduction to Computational Thinking

User-Defined Data Types
Object-oriented programming

Qiao Xiang, Qingyu Song

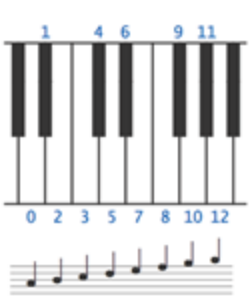
https://sngroup.org.cn/courses/ct-xmuf25/index.shtml

12/3/2025

This deck of slides are heavily based on cs112 at Yale University and cs101 at UCAS, respectively, by courtesy of Dr. Y. Richard Yang and Dr. Zhiwei Xu.

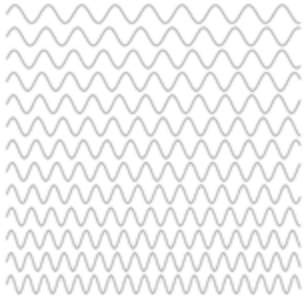
Recap: Digital Audio and Arrays

- Audio is a composition of sin waves



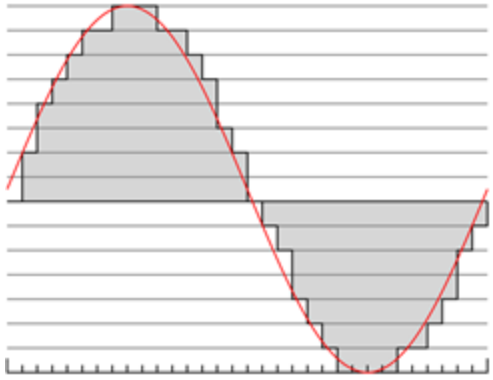
note	i	frequency
A	0	440.00
A♯ or B♭	1	466.16
B	2	493.88
C	3	523.25
C♯ or D♭	4	554.37
D	5	587.33
D♯ or E♭	6	622.25
E	7	659.26
F	8	698.46
F♯ or G♭	9	739.99
G	10	783.99
G♯ or A♭	11	830.61
A	12	880.00

$440 \times 2^{i/12}$



Notes, numbers, and waves

- Digital audio is samples of audio waves



$$y(i) = \sin\left(2\pi f \frac{i}{44{,}100}\right)$$

Recap: Digital Audio and Arrays

- Audio synthesis is to generate the sin waves and send to the speaker (StdAudio.play())

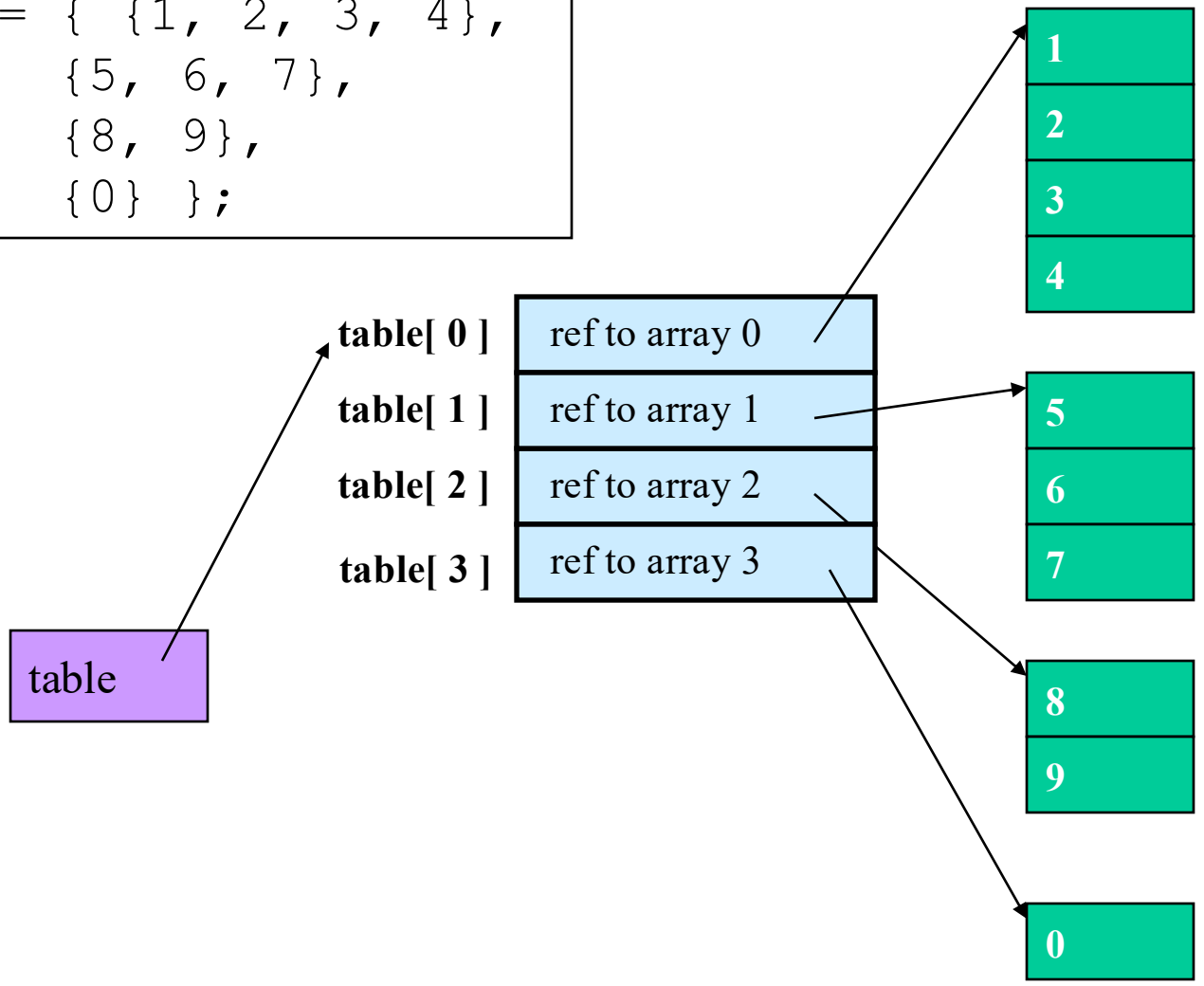
```
final double hz = 440.0;
final double seconds = 1.5;
final int SAMPLE_RATE = 44100;

int N = (int) (seconds * SAMPLE_RATE);
double[] a = new double[N];
for (int i = 0; i < N; i++) {
   a[i] = Math.sin(2 * Math.PI * hz * i / SAMPLE_RATE);
}
StdAudio.play(a);
```

- StdAudio plays/saves audio samples

public class StdAudio	
void play(String file)	*play the given .wav file*
void play(double[] a)	*play the given sound wave*
void play(double x)	*play sample for 1/44100 second*
void save(String file, double[] a)	*save to a .wav file*
double[] read(String file)	*read from a .wav file*

Recap: Multidimensional Arrays

```
int[][] table = { {1, 2, 3, 4},
                  {5, 6, 7},
                  {8, 9},
                  {0} };
```

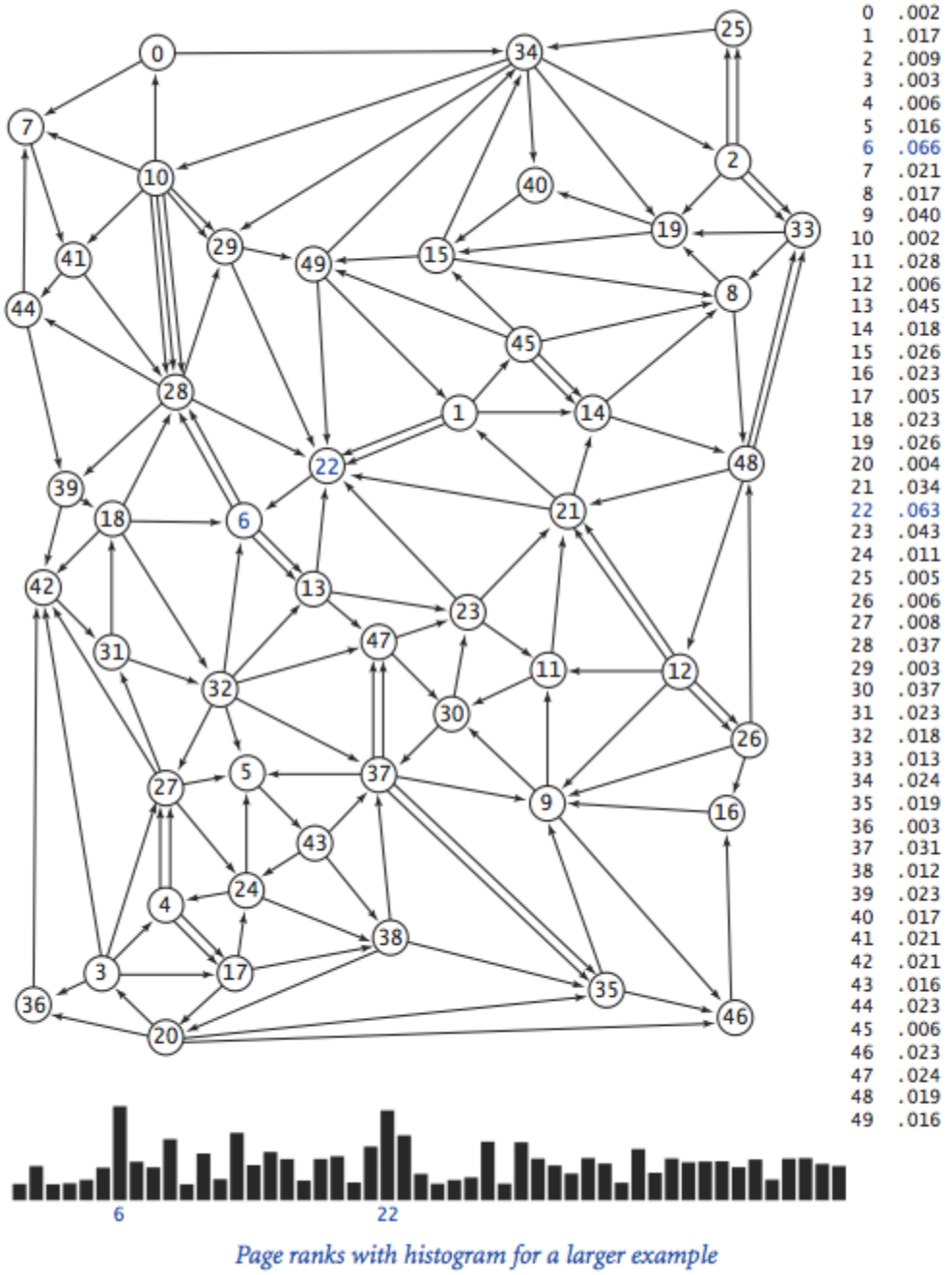


PageRank
[Sergey Brin and Larry Page, 1998]

- Problem: many Web pages may contain the searched key word (e.g., XMU), how to rank the pages when displaying search results?

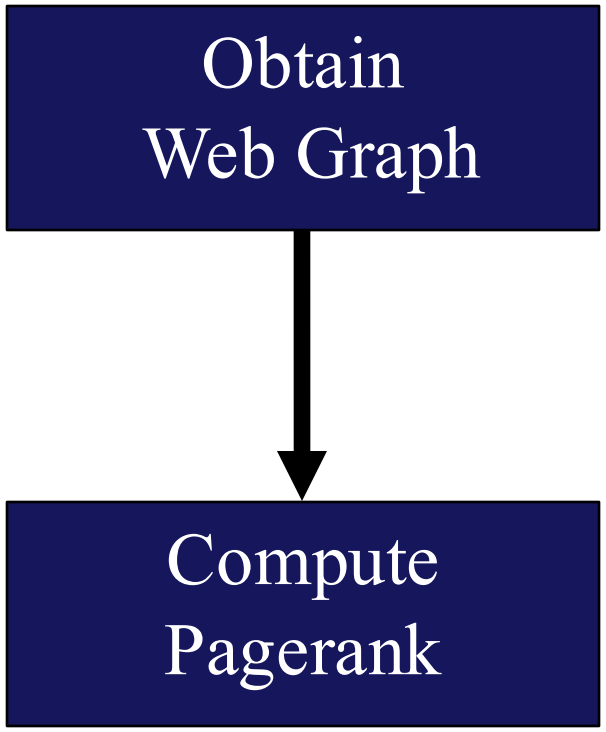
- Basic PageRank™ idea
  - The 10-90 rule
    - 10% of the time, user types a random page
    - 90% of the time, user clicks (follows) a random link on a given page
  - PageRank ranks pages according to frequencies (we call the pageranks) user visits the pages

PageRank



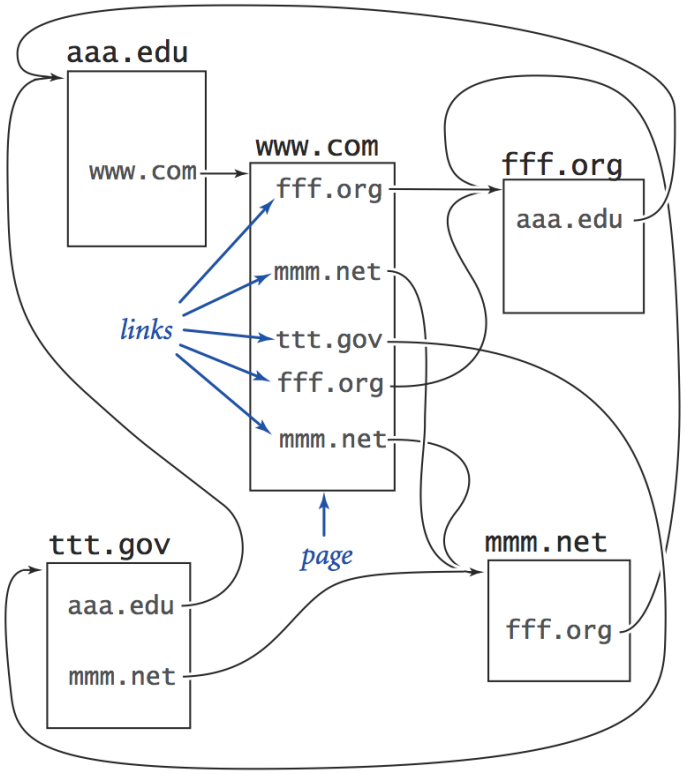
Page ranks with histogram for a larger example

High-Level Structure of Google PageRank System



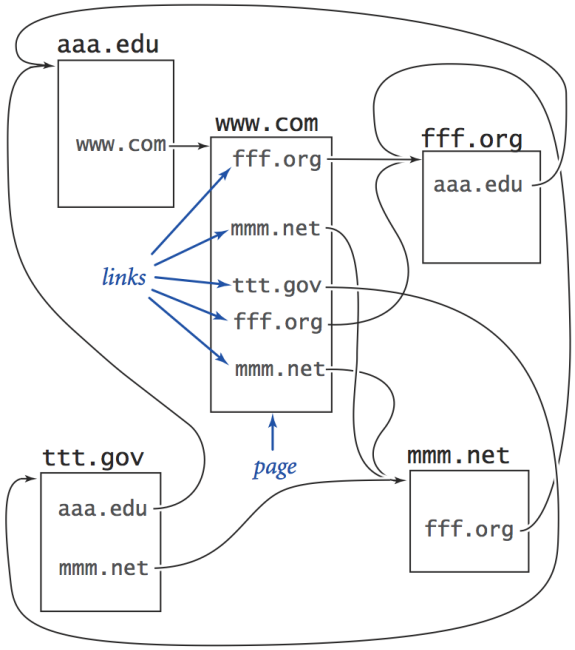
Obtain Web Graph

- Crawl the Web to obtain Web pages
- Parse pages to obtain links among pages



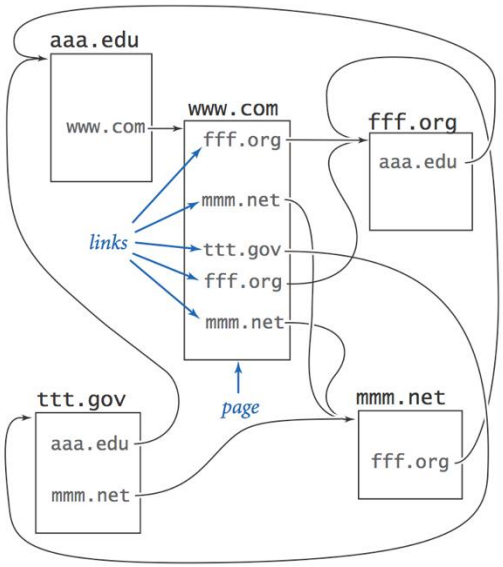
Storing Web Graph

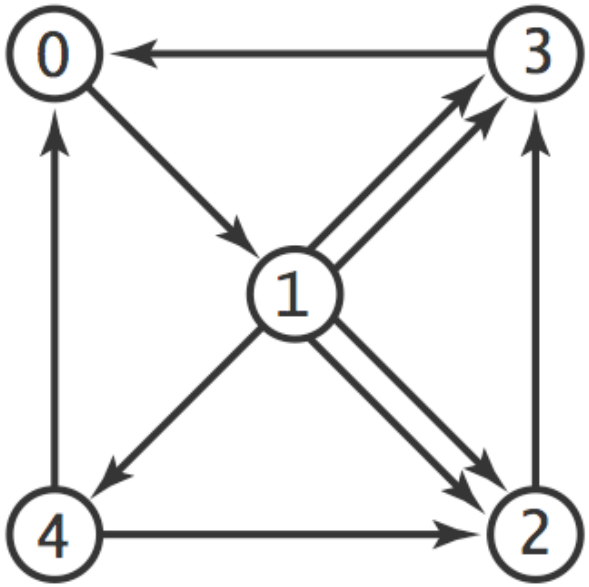
- Number pages 0 to N – 1
- Approach 1: a regular array storing connectivity matrix
  - conn[i][j] = 1 if page i links to page j; 0 otherwise
  - Q: How big is the array?
    - Most are 0



Storing Web Graph

- Number pages 0 to N – 1
- Approach 2: adjacent list
  - edges[i]: stores the pages that page i links to





Storing Web Graph: Adjacent List

#pages; allow 1-pass read

5

page 0 has 1 outgoing link to page 1

1

2 2 3 3 4

page 1 has four 5 outgoing links to pages 2, 2, 3, 3, 4

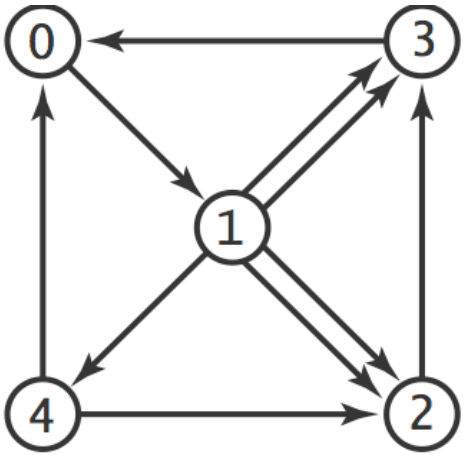
3

0

0 2

Outgoing adjacency List:

- Each line for one page
- Stores the links contained in this page



PageRank: Load Web Graph

```
5
1
2 2 3 3 4
3
0
0 2
```

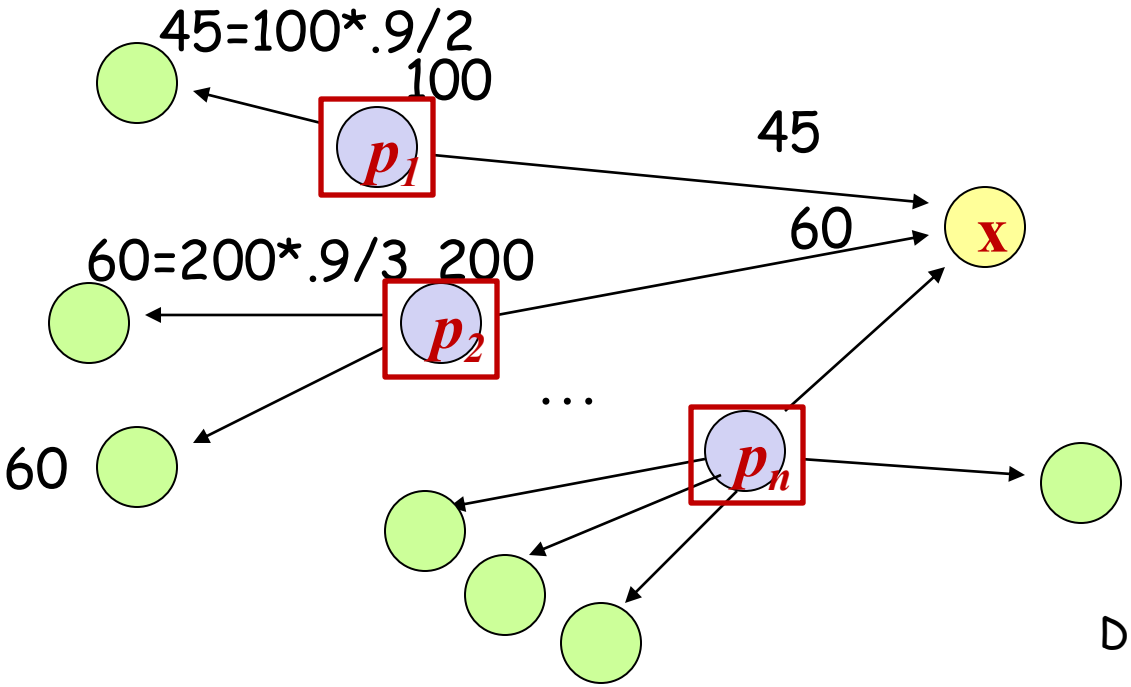
```
Scanner input = new Scanner(new File("tiny-web.txt"));

// First read N, the number of pages
int N = Integer.parseInt(input.nextLine());

// An irregular 2D array to keep track of outgoing links
int[][] outgoingLinks = new int[N][];

// read in graph one line at a time
for (int i = 0; i < N; i++) {
   String line = input.nextLine(); // read outgoing links of i
   String[] links = line.split(" ");
   outgoingLinks[i] = new int[links.length];
   for (int j = 0; j < links.length; j++) {
      outgoingLinks[i][j] = Integer.parseInt(links[j]);
   }
}
```

Approach 1: Simulating Random Walk among Web Pages

```
// assume N pages
node = init position of user at a node visited[node] ++;
Loop
  pick a random number to decide restart or follow links
  if random chooses restart (10% chances)
    node = choose random 0, N-1
  else // follow links
    nLinks = #outgoing links of node
    outIndex = choose random 0, nLinks-1;
    node = links[node][outIndex]
  visited[node] ++

Sort visited array
```

Approach 2: (Large-Popultion) Round-Based Visit Distribution

- Assume current round page rank (frequency, #visitors) of page p_i is $PR_c(p_i)$
- Next round page rank
  - Each page gets 1/N of the 0.1 of total visitors
  - Each page distributes its 90% of its current-round frequency (visitors) equally among its outgoing hyperlinks

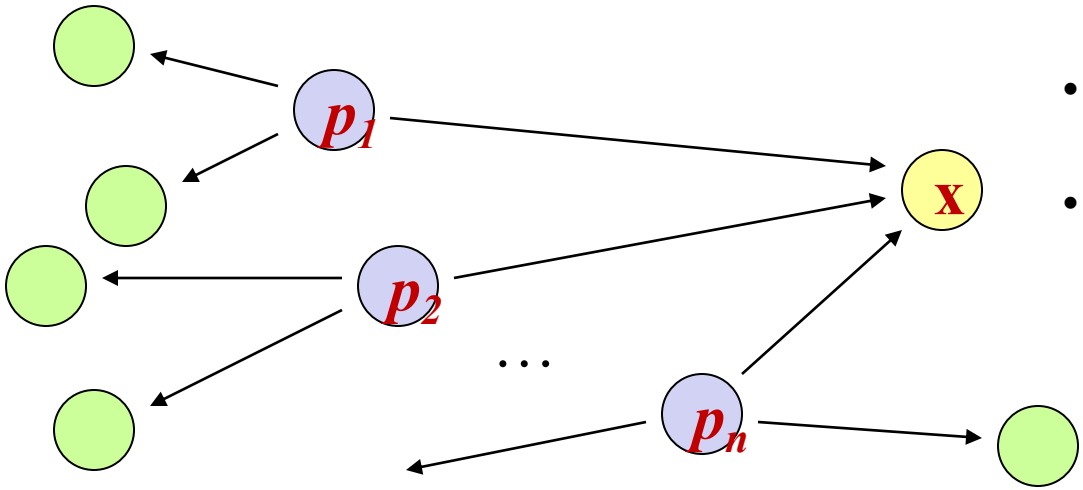


Distributes 散步

Round-Based PageRank

- Initialize arbitrary page ranks
- Iterative algorithm to simulate visit redistribution
  - Assume current round page rank of page p_i is $PR_c(p_i)$
  - Update next round

$$PR_{new}(x) = 0.1\left[\frac{1}{N}\right] + 0.9\sum_{i=1}^{n}\frac{PR_{pre}(p_i)}{C(p_i)}$$

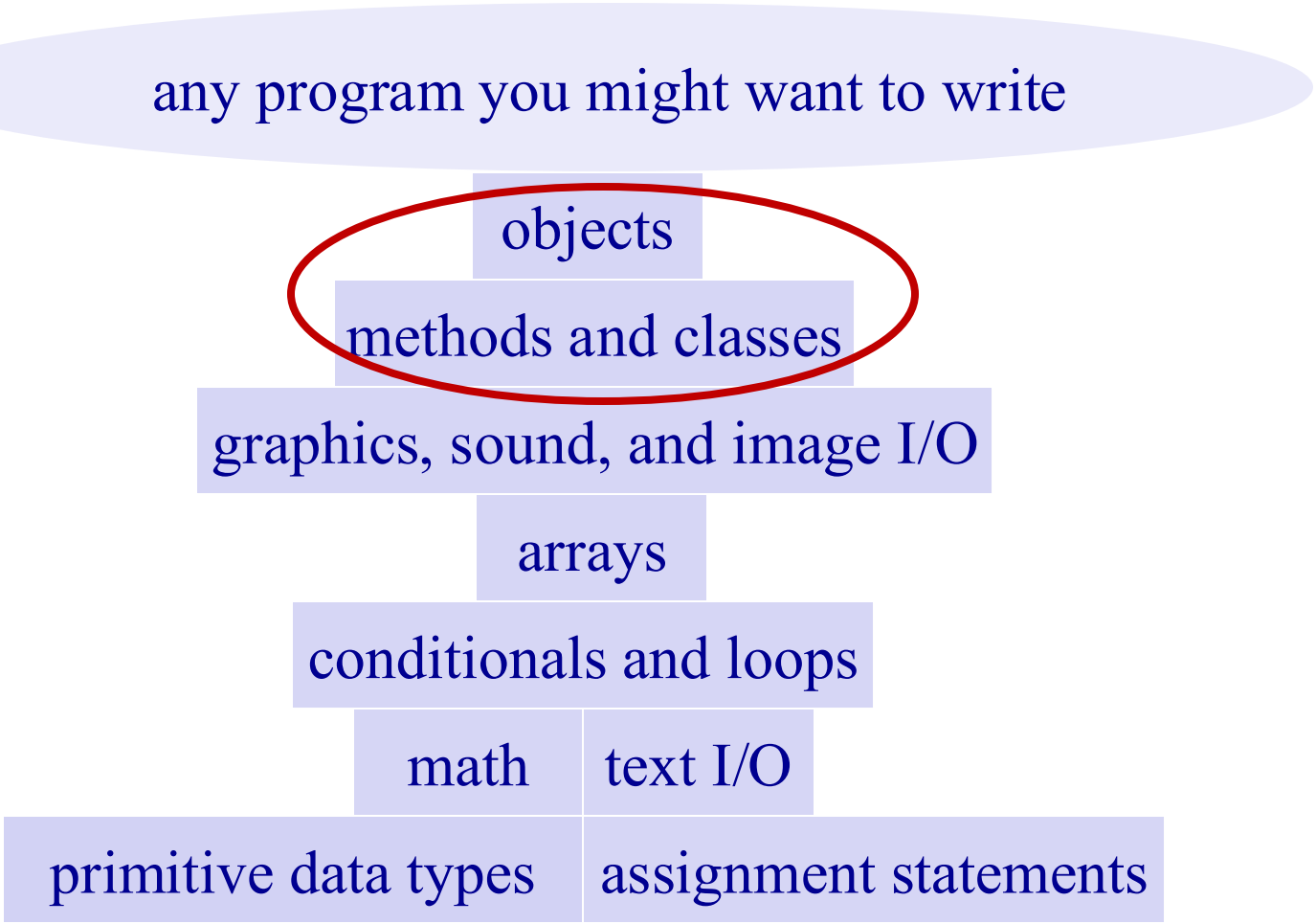


- $PR_{pre}()$: outbound links to p_i.
- $C()$: count of outbound links.

PageRank: Compute Rank

```
double[] pr = new double[N];
pr[0] = 1; // initialize to assume start at web page 0
// or Arrays.fill(pr, 1.0 / N);

for (int t = 0; t < 20; t++) {
   double[] newpr = new double[N]; // init newpr
   Arrays.fill(newpr, 0.1 / N);

   // loop over the node to redistribute the frequencies
   for (int i = 0; i < N; i++) { // redistribute node i
      for (int j = 0; j < outgoingLinks[i].length; j++) {
         int to = outgoingLinks[i][j];
         newpr[to] += 0.9 * pr[i] / outgoingLinks[i].length;
      }
   }
   pr = newpr; // swap newpr to be pr
   System.out.printf("pr[%2d] = %s\n", t, Arrays.toString(pr) );
}
```

Roadmap: Foundational Programming Concepts



Programming Organizing Structure

- Method and class are concepts for programming organization

  - Method provides an abstraction to group a sequence of statements
    - abstraction: we can use it without knowing its details

  - Class
    - So far our usage is to group similar methods (such as a folder) together, e.g., the Math class pre-defined by Java
    - A general, important functionality of class is to organize both data and behaviors

Road Ahead: Object Oriented Programming

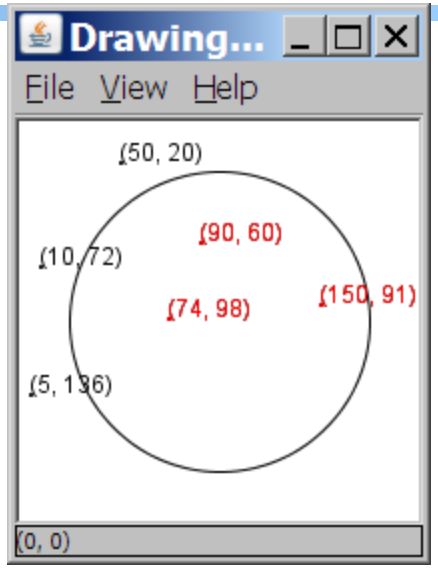
- Four steps
  - Struct data type (organize data together)
  - Object-oriented programming (organize data and code)
  - Class inheritance (reuse code)
  - Polymorphism (reuse code, allowing adaptation)

Inheritance 继承
Polymorphism 多态

A Programming Problem

- Given a file of cities' (x, y) coordinates, which begins with the number of cities:

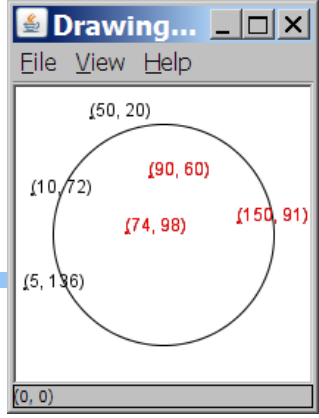
```
6
50 20
90 60
10 72
74 98
5 136
150 91
```



- Write a program to draw the cities and then drop a center point and turn all cities that are within a given radius red :

```
Center site x? 100
Center site y? 100
Radius? 75
```

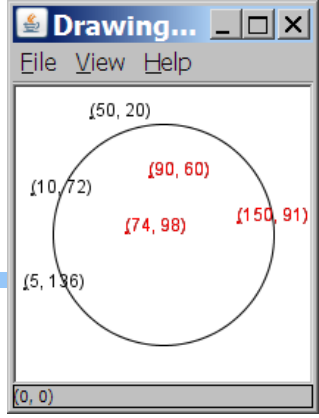
Solution We Can Do So Far




```
Scanner input = new Scanner(new File("cities.txt"));
int cityCount = input.nextInt();
int[] xCoords = new int[cityCount];
int[] yCoords = new int[cityCount];

Scanner console = new Scanner( System.in );
int xCenter = console.nextInt();
int yCenter = console.nextInt();

for (int i = 0; i < cityCount; i++) {

    xCoords[i] = input.nextInt();
    yCoords[i] = input.nextInt();
    if (distance(xCoords[i], yCoords[i],
                 xCenter,    yCenter      ) < THRESHOLD) {
        // draw red
    } else …
}
```

- **Parallel arrays**: 2 arrays with related data at same indexes.
- Conceptually, we should have just a single array of points.

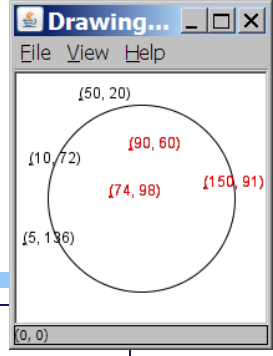
A Conceptual Design




```
Scanner input = new Scanner(new File("cities.txt"));
int cityCount = input.nextInt();
Point[] points = new Point[cityCount];
Point pointCenter = readPointFromUser();

for (int i = 0; i < cityCount; i++) {
    points[i] = readPoint(input);
    if ( distance(points[i], pointCenter) < THRESHOLD) {
        // draw red
    }
}
...
```

- By introducing the concept of Point, we abstract away the details of the coordinate system

Summary: Complex Data Type Motivation



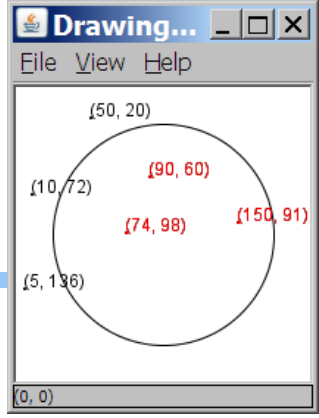
2 parallel arrays

```
Scanner input = new Scanner(new File( cities.txt"));
int cityCount = input.nextInt();
int[] xCoords = new int[cityCount];
int[] yCoords = new int[cityCount];
for (int i = 0; i < cityCount; i++) {
    xCoords[i] = input.nextInt();
    yCoords[i] = input.nextInt();
    if (distance(xCoords[i], yCoords[i],
                 xCenter,    yCenter      ) < THRESHOLD) {
        // draw red
    }
}
```

point abstraction

```
Scanner input = new Scanner(new File("cities.txt"));
int cityCount = input.nextInt();
Point[] points = new Point[cityCount];
for (int i = 0; i < cityCount; i++) {
    points[i] = readPoint(input);

    if ( distance(points[i], pointCenter) < THRESHOLD) {
        // draw red
    }
}
```

Language Support: Defining Point



- Need a language structure to specify that each point is defined by two data components:
  - The x coordinate
  - The y coordinate

```
public class Point {
    int x;
    int y;
}
```

  - Save this code into a file named `Point.java`.

- The above code creates a new user-defined type (class) named `Point`.
- The `Point` class definition serves as a template for creating `Point` objects.

A Class and its client

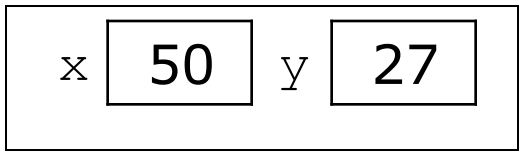
- `Point.java` is not, by itself, a runnable program.
  - A class can be used by multiple **client** programs.

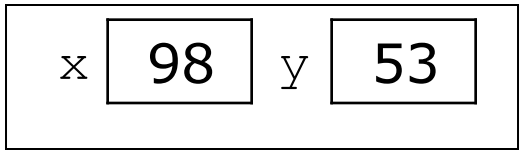
`PointMain1.java` (client program)

```
public class PointMain1 {
  main(String args) {
    Point p1 = new Point();
    p1.x = 50;
    p1.y = 27;

    Point p2 = new Point();
    p2.x = 98;
    p2.y = 53;

  }
}
```

`Point.java` (class of objects)

```
public class Point {
    int x;
    int y;
}
```





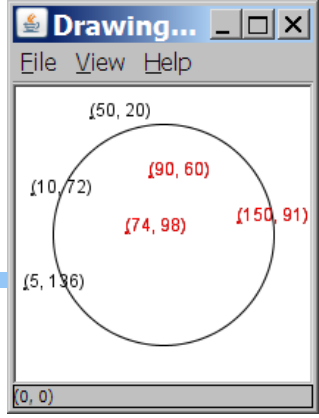
User-Defined Type Term: Fields

```
public class Point {
    int x;
    int y;
}
```

- **x or y is called a field (字段)**
  - A non-static, class scope variable defined inside a class (new type)
  - A field is also called an instance variable, or an attribute
  - A field is part of the descriptive characteristics (state) of an object
  - Each object has *its own copy* of each field
- More example:

```
public class BankAccount {
    String acctName;
    int    acctNumber;
    double balance;
}
```

Summary of Important Concepts: Class, Object, Variable, Field




```
public class Point {
    int x;
    int y;
}
```

- The `Point` class definition serves as a template for creating `Point` objects.
- A field (also called instance variable, attribute) is a class-scope, non-static variable describing characteristics of each object. Each object has its own copy of each field
- A Point variable stores a reference to a Point object

Historical Note

- Before object-oriented programming, traditional languages (e.g. `C`) already have a language structure called `struct` to allow user to define data types combining multiple data fields

- Object-oriented languages go one step further to organize both data and methods together

Outline

- Admin
- Defining classes
  - Data encapsulation (struct)
  - Data+behavior encapsulation

Encapsulation 封装

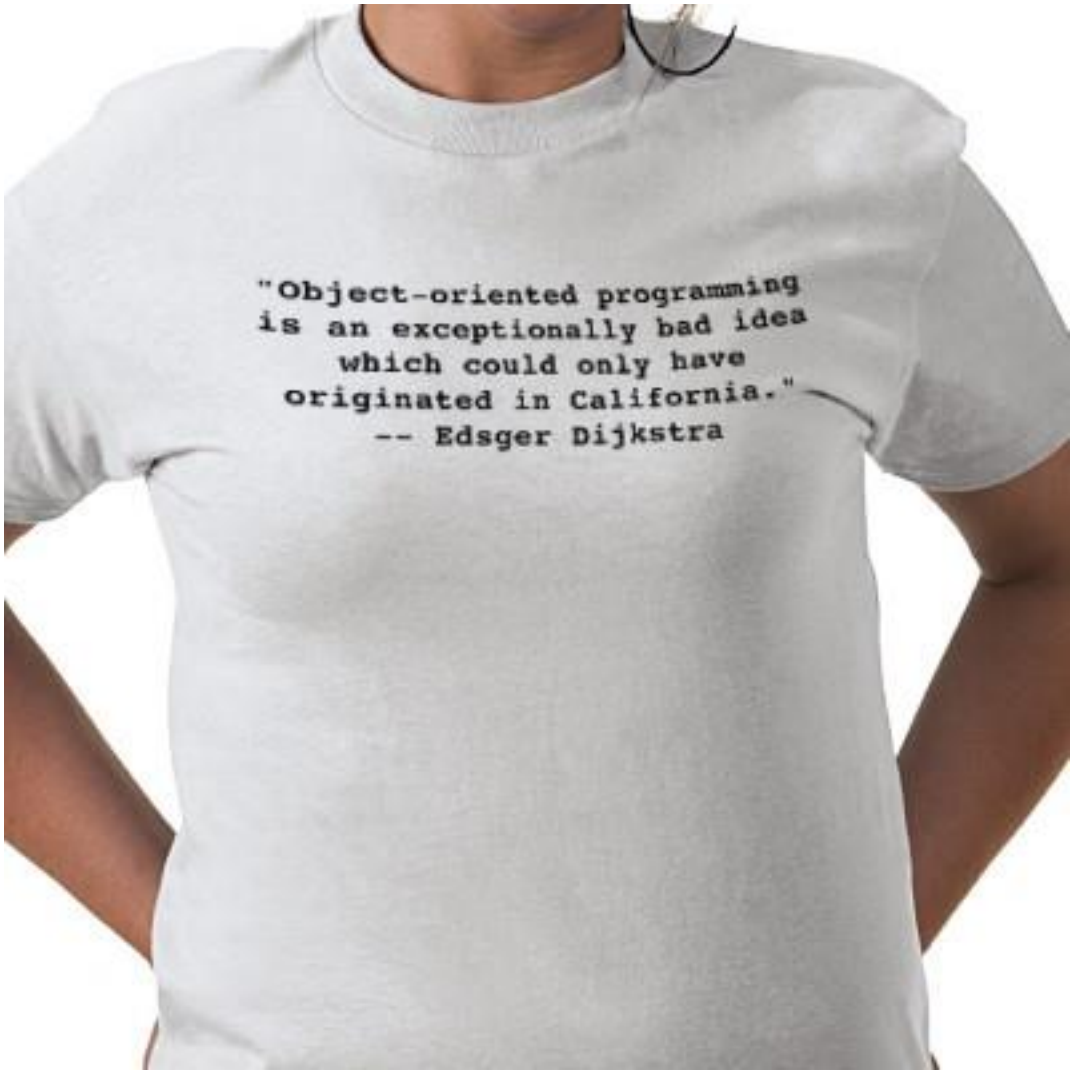
Object Oriented Programming Overview

- 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.

- Procedural(过程式) programming. [verb-oriented]
  - Tell the computer to do this or that.

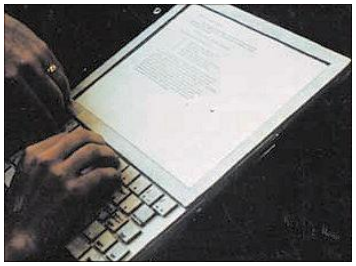
- Objected oriented programming (OOP). [noun-oriented]
  - Programming paradigm based on data types.
  - Identify objects that are part of the problem domain or solution.
  - Identity: objects are distinguished from other objects (references).
  - State: objects know things (instance variables).
  -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

Object Oriented Programming



Alan Kay: Father of OOP

- Alan Kay. [Xerox PARC 1970s]
  - Invented Smalltalk programming language.
  - Conceived (构思) 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

Alan Kay
2003 Turing Award

Xerox PARC is research and development center in the US.

Motivation: Drawing Points

- The client program PointMain1 draws `Point` objects:

```
// draw each city
Point p1 = new Point();
p1.x = 15;
p1.y = 37;
StdDraw.filledCircle(p1.x, p1.y, 3);
StdDraw.textLeft(p1.x, p1.y,
                 "(" + p1.x + ", " + p1.y + ")" );

Point p2 = new Point();
p2.x = 120;
p2.y = 140;
StdDraw.filledCircle(p2.x, p2.y, 3);
StdDraw.textLeft(p2.x, p2.y,
                 "(" + p2.x + ", " + p2.y + ")" );
```

Eliminating Redundancy, v1

- We can eliminate the redundancy with a static method:

```
// Draws the given point using StdDraw
public static void draw(Point p) {
    StdDraw.filledCircle(p.x, p.y, 3);
    StdDraw.textLeft(p.x, p.y,
                     "(" + p.x + ", " + p.y + ")" );
}
```

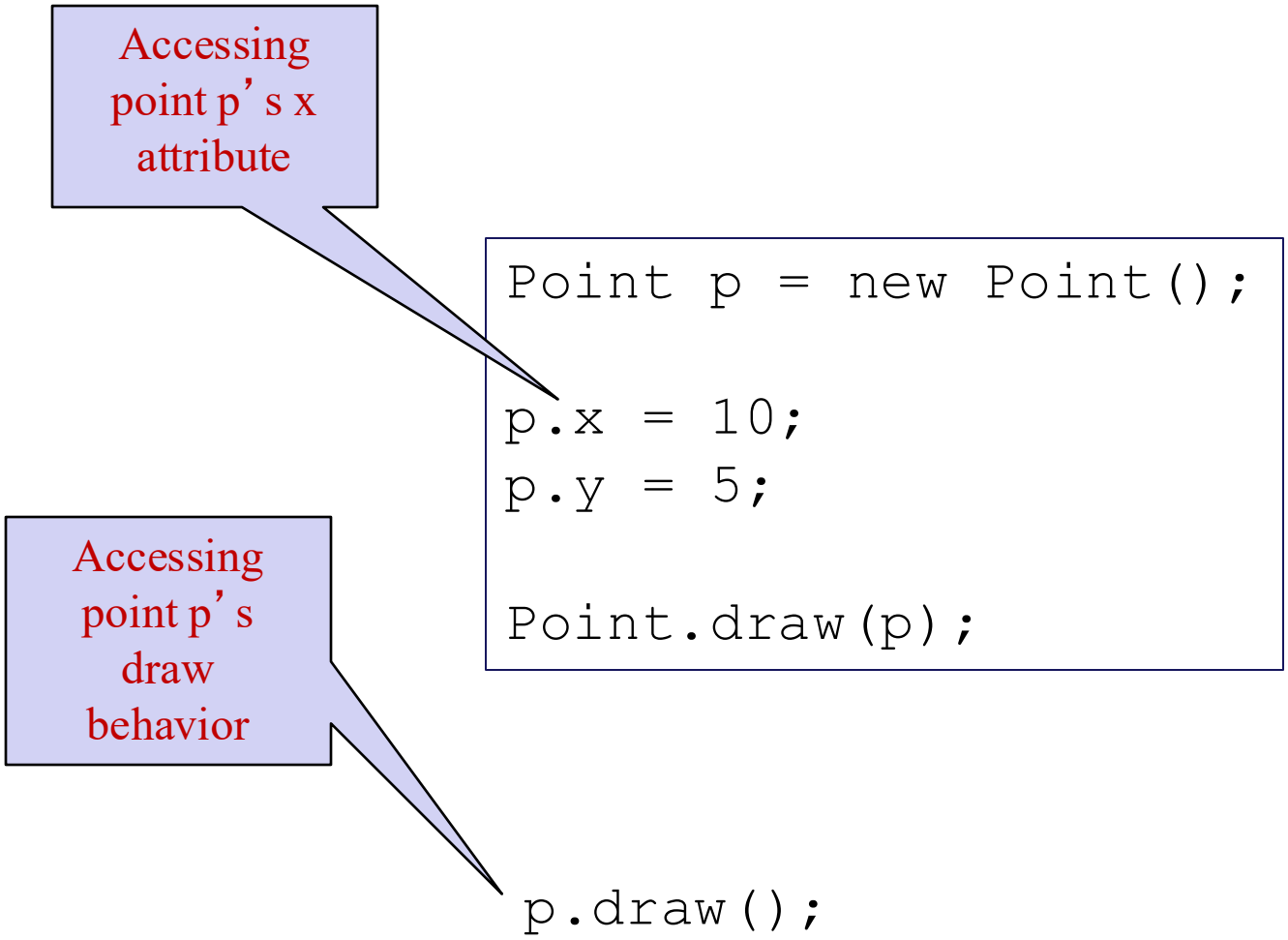
Question: where is a natural class (Point or PointMain1) to define the method **`draw(Point p)`**?

Where do we Define draw(p): Attempt (意图) 1: in Point as a Static Method

```
public class Point {
    int x;
    int y;

    public static void draw(Point p) {
        StdDraw.filledCircle(p.x, p.y, 3);
        StdDraw.textLeft(p.x, p.y,
                         "(" + p.x + ", " + p.y + ")" );
    }
}
```

```
Point p1 = new Point();
p1.x = 7; p1.y = 2;
Point.draw(p1);

Point p2 = new Point();
p2.x = 4; p2.y = 3;
Point.draw(p2);
```

Consistent Data Access and Method Access



Instance Methods

- **instance method** (or **object method**): Exists inside each object of a class and gives behavior to each object.

```
public type name(parameters) {
    statements;
}
```

  - same syntax as static methods, but without `static` keyword

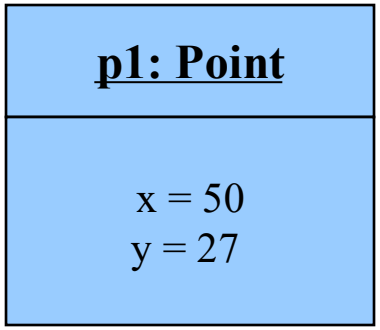
Example:

```
public void shout() {
    System.out.println("HELLO THERE!");
}
```

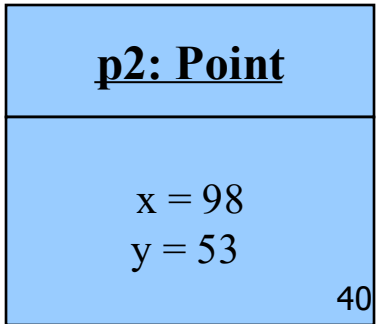
Instance Method example

Instance method (or object method) can access all class scope variables

```
public class Point {
    int x;
    int y;

    // Draws this Point object with the given pen.
    public void draw() {
      StdDraw.filledCircle(x, y, 3);
      StdDraw.textLeft(x, y,
                       "(" + x + ", " + y + ")");
    }
}
```



- The `draw` method no longer has a `Point p` parameter.
- How will the method know which point's x/y to draw?



Invoking Instance Method

- Format

  <obj>.<instance method>(…)

- The <obj> provides an implicit parameter

The Implicit Parameter

- During the call `p1.draw();`
  the object referred to by `p1` is the implicit parameter.

- During the call `p2.draw();`
  the object referred to by `p2` is the implicit parameter.

- In an instance method, when we refer to a field, we are referring to the field of the implicit parameter.
  - We say that it executes in the *context of* a particular object.

Summary: Defining Related Method and Data in the Same Class: Instance Method

```
public class Point {
    int x;
    int y;

    public static void draw(Point p) {
        StdDraw.filledCircle(p.x, p.y, 3);
        StdDraw.textLeft(p.x, p.y,
                         "(" + p.x + ", " + p.y + ")");
    }
}
```

```
Point p1 = new Point();
p1.x = 7; p1.y = 2;
p1.draw(); // Point.draw(p1);

Point p2 = new Point();
p2.x = 4; p2.y = 3;
p2.draw(); // Point.draw(p2);
```

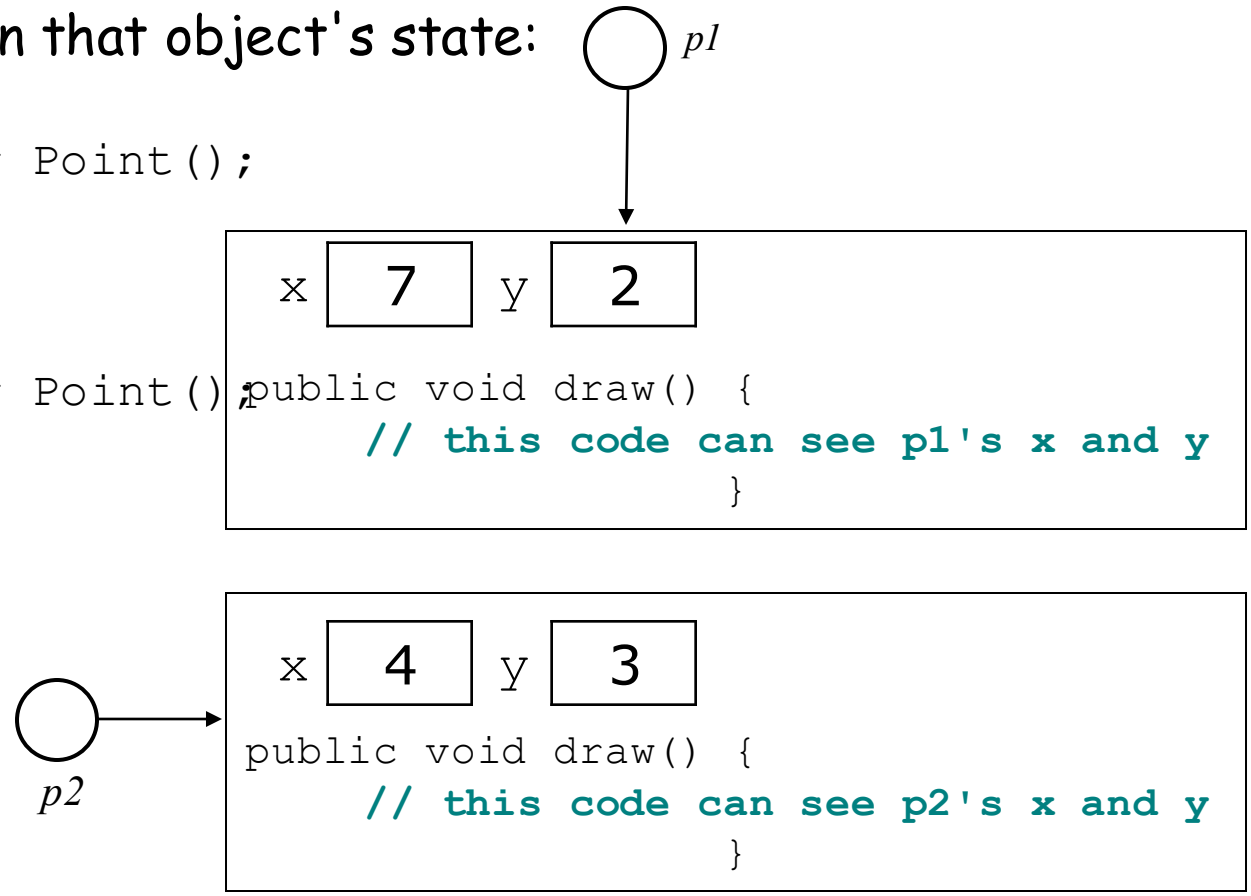
p1 provides the implicit parameter: The x and y in draw() are those of the object referenced by **p1**.

p2 provides the implicit parameter: The x and y in draw() are those of the object referenced by **p2**.

Point objects w/ method

- Each Point object has its own copy of the draw method, which operates on that object's state:

```
Point p1 = new Point();
p1.x = 7;
p1.y = 2;

Point p2 = new Point();
p2.x = 4;
p2.y = 3;

p1.draw();
p2.draw();
```



Static Method vs Instance Method

```
public class Point {
    int x;
    int y;
    public static void draw(Point p) {
        StdDraw.filledCircle(p.x, p.y, 3, 3);
        StdDraw.textLeft(p.x, p.y, "(" + p.x + ", " + p.y + ")" );
    }
}
```

```
public class Point {
    int x;
    int y;
    public ~~static~~ void draw(~~Point p~~) {
        StdDraw.filledCircle(~~p.~~x, ~~p.~~y, 3, 3);
        StdDraw.textLeft(~~p.~~x, ~~p.~~y, "(" + ~~p.~~x + ", " + ~~p.~~y + ")");
    }
}
```

Benefit of Instance Method

```
Point p = new Point();

p.x = 10;
p.y = 5;

p.draw();
```

Consistent (same) syntax(语法) accessing data and behaviors: a class is an abstraction for a collection of objects with common

- data fields (attributes)
- behaviors/services (i.e., what such objects can do or be done to them)

Instance Method questions

- Write an instance method `toString` to generate string (x, y)
- Write an instance method `translate` that changes a `Point`'s location by a given *dx*, *dy* amount.
- Write an instance method `distanceFrom` that returns the distance between the `Point` and another point

  Use the formula: $\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$