# Introduction to Computational Thinking

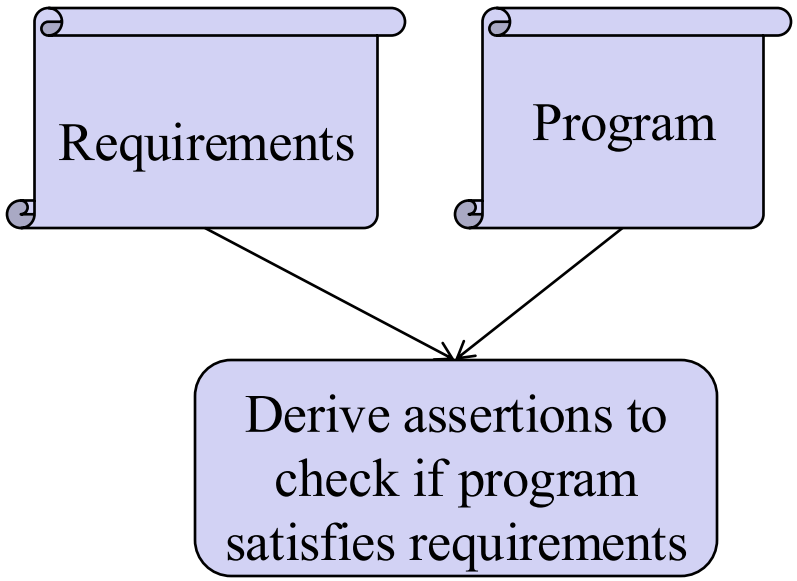
*Indefinite Loops: Examples*
*Arrays*

**Qiao Xiang**, Qingyu Song
https://sngroup.org.cn/courses/ct-xmuf25/index.shtml
11/26/2025

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: Program Analysis

- Program analysis is a useful tool to understand the flow control structure of a program
- Key idea is to
  - derive assertions on program statements
  - compare assertions with requirements



# Recap: Control Statements and Assertions

- Control statement (e.g., `if`, `while`, `break`) establishes assertions
  - At the start of an `if` or loop's body, the <test> and what can be implied by the <test> must be `true`, e.g.,

```
while (y < 10) {
   // (y < 10) must be true
   ...
}
```

  - In the `else` or after a loop w/o break, the <test> must be `false`, e.g.,

```
while (y < 10) {
   ...
}
// (y < 10) must be false
```

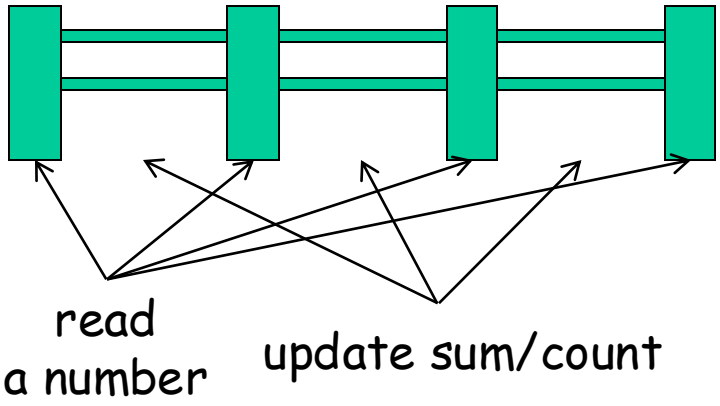
# Recap: Using Assertion to View and Fix Program Bug

Grade Analyzer, sentinel input

```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade;
do {
  System.out.print("Type a grade (-1 to exit): ");
  grade = console.nextInt();
  if (grade != -1) {
  sum += grade; count ++;

  }
} while (grade != -1);

System.out.println("The average is " +
                   (count > 0? sum /count : 0) );
```

# Recap: Another View of the Sentinel Bug: The Fencepost Problem

The sentinel input loop pattern is a fencepost problem: Must read *N* grades, but sum/count only the first *N*-1. The buggy version did not handle it correct.

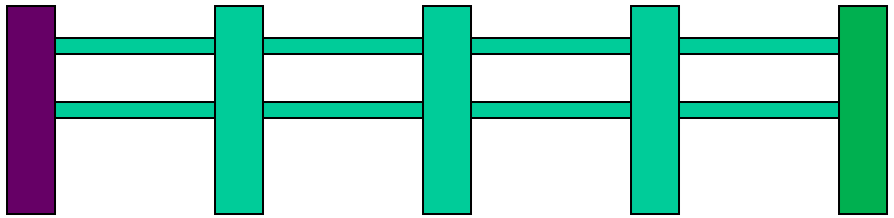



```
Scanner console = new Scanner(System.in);
int sum = 0, count = 0;
int grade;
do {
  System.out.print("Type a grade (-1 to
   exit): ");
  grade = console.nextInt();


  sum += grade; count ++;

} while (grade != -1);

System.out.println("The average is " +
                   (count > 0?
                       sum /count : 0) );
```

Fencepost 栅栏安置

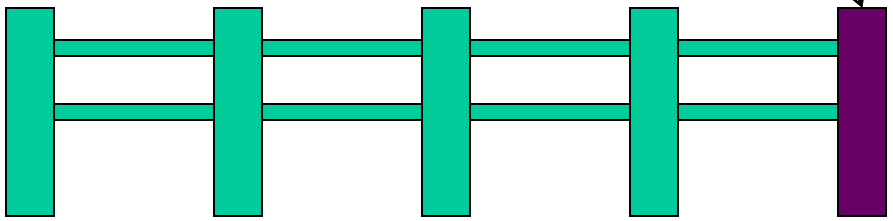
# Recap: Fencepost Loop Solutions

```
place a post
loop (length of post-1) {
    place a wire.
    place a post
}
```



```
loop (length of post) {
     place a post.
     if (! last post)
          place a wire.
}
```

Detect if it is the last post in the loop, if it is, do not place the wire



# Comparison

```
int sum = 0;
System.out.print("Enter a number (-1 to quit): ");
int grade = console.nextInt();

while ( grade != -1 ) {
    sum = sum + grade;

    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();
}
```

**place a post**
loop (length of post-**1**)
{
  place a wire.
  place a post
}

```
int sum = 0;
int grade = 0;
do {
    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();

    if ( grade != -1 ) {      // detect the last post
        sum = sum + grade;
    }
} while ( grade != -1 );
```

loop (length of post)
{
  place a post.
  **if (! last post)**
    place a wire.
}

# Comparison

```
int sum = 0;
System.out.print("Enter a number (-1 to quit): ");
int grade = console.nextInt();

while ( grade != -1 ) {
    sum = sum + grade;

    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();
}
```

**place a post**
loop (length of post-**1**)
{
  place a wire.
  place a post
}

```
int sum = 0;
int grade = 0;
while ( grade != -1 ) {
    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();

    if ( grade != -1 ) {      // detect the last post
       sum = sum + grade;
    }
};
```

loop (length of post)
{
  place a post.
  **if (! last post)**
    place a wire.
}

# Exercise: Remove a Test

So that we can exit the loop

So that we don't have an extra wire

```
int sum = 0;
int grade = 0;
while ( grade != -1 ) {
    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();

    if ( grade != -1 ) {      // detect the last post
       sum = sum + grade;
    }
}
```

Q: Can we remove one of the tests?

# Remove a Test

So that we can exit the loop

So that we don't have an extra wire

```
int sum = 0;
int grade = 0;
while ( true ) {
    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();

    if ( grade != -1 ) {      // detect the last post
       sum = sum + grade;
    }
}
```

# Remove a Test

```
int sum = 0;
int grade = 0;
while ( true ) {
    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();

    if ( grade == -1 ) {      // detect the last post
       break;
    }
    sum = sum + grade;
}
```

Two functions: (1) stop loop; (2) avoid last wire

This pattern is called infinite loop with sentinel break.

# Remove a Test

```
int sum = 0;
int grade = 0;
do {
    System.out.print("Enter a number (-1 to quit): ");
    grade = console.nextInt();

    if ( grade == -1 ) {      // detect the last post
       break;
    }
    sum = sum + grade;
} while ( true );
```

Two functions: (1) stop loop; (2) avoid last wire

For such infinite pattern, either while or do/while works.

# Outline

- Admin and recap
- Indefinite loops
  - Motivation
  - Program statements (for/while/do/while)
  - Common indefinite loop pattern: search loop
  - Program analysis
  - Common indefinite loop pattern: sentinel user input processing
  - Examples
    - Robust grade analyzer

# Robust Grade Analyzer

- Q: Is the program robust (i.e., against bad user input)?

```
int sum = 0;
System.out.print( PROMPT );
int grade = console.nextInt();

while ( grade != -1 ) {
    sum = sum + grade;

    System.out.print( PROMPT );

    // Cannot establish assertion: console.hasNextInt

    grade = console.nextInt();
}
```

A: No. The program will crash if user input is not an integer.

# Exercise

- Convert

```
System.out.print( PROMPT );
int grade = console.nextInt();
```

to a robust method

```
public static int getInt(Scanner console,
                         String prompt);
```

Loop
  show user the prompt
  show error msg skip for bad input,
until user gives an integer

# Design Questions

- Is the loop better controlled by

  `for` or `while/do/while`?

  - Indeterminate logical condition (console.hasNextInt) -> more likely a while or do/while loop

- How many times may the loop execute, at least once?
  - at least once => we may try a do/while loop

Indeterminate不确定

# getInt() as a Fencsepost

```
// Progress: If there is a valid int, the loop finishes
public static int getInt(Scanner console, String prompt) {
    System.out.print(prompt);                  // post

    while ( !console.hasNextInt() ) {    // check last post
       // skip/print error only if next token is not int
       console.next();                         // wire
       System.out.println("That is not an integer.   " +
                          "Please try again.");

       System.out.print(prompt);               // post
    }
    // safety: next token should be an int before nextInt
    return console.nextInt();
}
```

# Outline

- Admin and recap
- Indefinite loops
  - Motivation
  - Program statements (for/while/do/while)
  - Common indefinite loop pattern: search loop
  - Program analysis
  - Common indefinite loop pattern: sentinel user input processing
  - Examples
    - Robust grade analyzer
    - MathAdding game

# Exercise: MathAdding Game

- Write a program that plays an adding game.
  - Ask user to solve random adding problems with n (2-5) numbers.
  - The user gets n-1 points for an addition with n numbers, 0 for incorrect.
  - A user has 3 lives (3 chances) to make mistakes.

```
4 + 10 + 3 + 10 = 27
9 + 2 = 11
8 + 6 + 7 + 9 = 25
Wrong! The answer was 30
5 + 9 = 13
Wrong! The answer was 14
4 + 9 + 9 = 22
3 + 1 + 7 + 2 = 13
4 + 2 + 10 + 9 + 7 = 42
Wrong! The answer was 32
You earned 9 total points.
```

# Design Questions

- Is the loop better controlled by `for` or `while/do/while`?
  - Indeterminate logical condition (cumulate 3 errors) -> more likely a `while` or `do/while` loop
- How many times may the loop execute, at least once?
  - at least once => we may try a `do/while` loop

# MathAddingGame Structure

```
// Asks the user to do adding problems and scores them.
import java.util.*;

public class MathAddingGame {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        // init. state
        int totalPoints = 0;
        int nChances = 3;
        do {

            // play a round;
            // update state: nChances, totalPoints


        } while (nChances > 0);

        System.out.println("You earned " + totalPoints
                           + " total points.");
    }
```

Design question: we design a method playOneRound, what should the return be?

The user gets n-1 points for an addition with n numbers, 0 for incorrect.

# `playOneRound` Requirement

```
// play a round:
// return the # of points earned;
// 0 means user made a mistake
static int playOneRound()
```

# playOneRound Structure

```
1 + 2 + 3 + 4 + 5 = 15
4 + 10 + 3 + 10 = 27
4 + 9 + 9 = 22
9 + 2 = 11
```

structure:

- pick n = # of operands to be added (2-5)
- build problem string
- output problem to user
- compute points to return

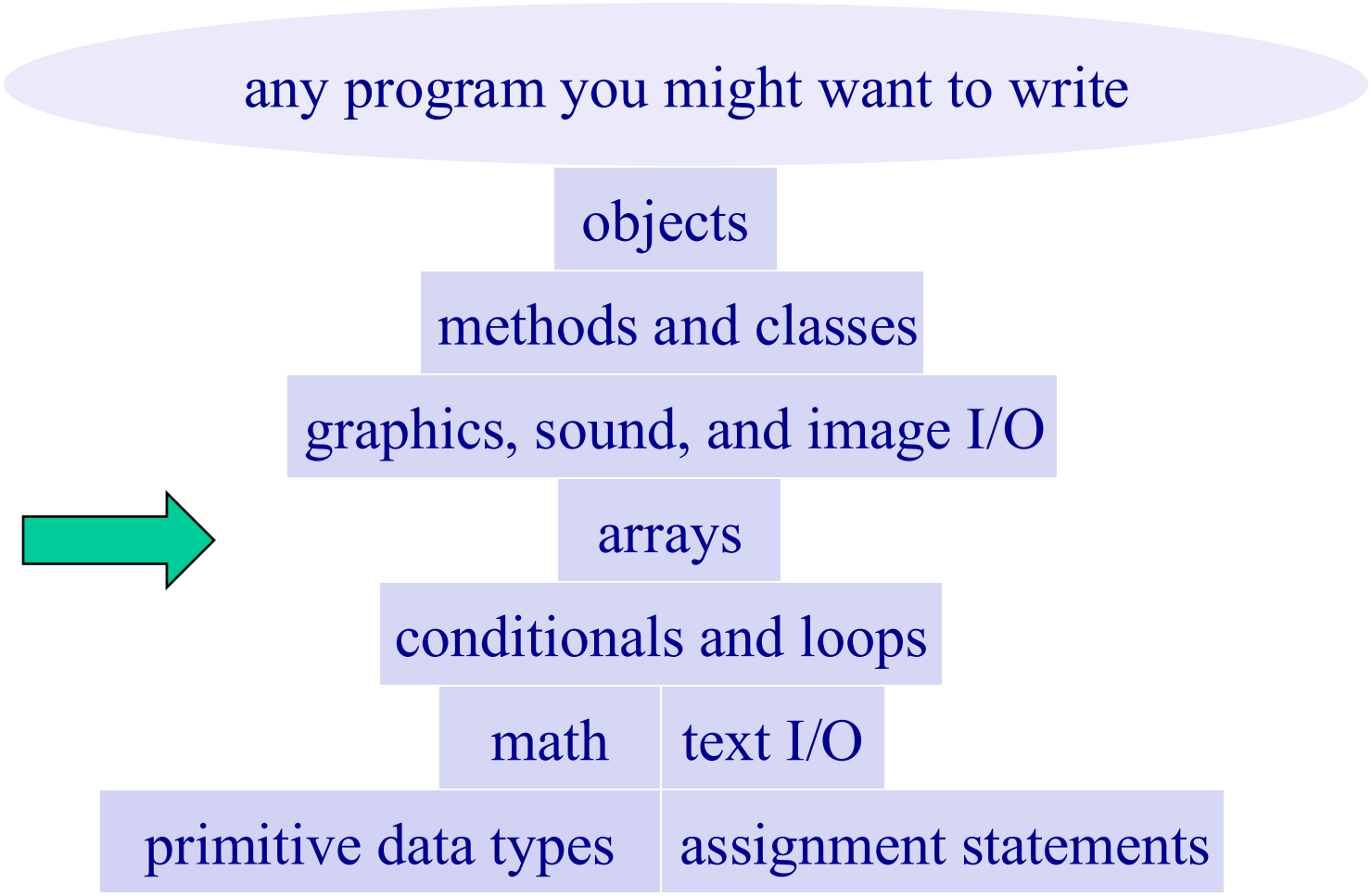
# playOneRound

```
...
// Builds one addition problem and presents it to the user.
// Returns # of points earned.
public static int playOneRound(Scanner console) {
    // print the operands being added, and sum them
    int nOperands = randInt(2, 5);
    int sum = 0;

    int n = randInt(1, 10);
    String question = "" + n; // post

    for (int i = 2; i <= nOperands; i++) {
        n = randInt(1, 10);
        question += " + " + n; // wire + post
        sum += n;
    }
    question += " = ";

    // read user's guess
    int guess = getInt( console, question );
    if (guess == sum)
        return nOperands - 1;
    else
        return 0;
} // end of playOneRound
```

# MathAddingGame Program

```
// Asks the user to do adding problems and scores them.
import java.util.*;

public class MathAddingGame {
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);

        // play until user gets 3 wrong
        int totalPoints = 0;
        int nChances = 3;
        do {
             int roundPoints = playOneRound(console);
             totalPoints += roundPoints;
             if ( roundPoints == 0) {
                nChances--;
             }
        } while (nChances > 0);

        System.out.println("You earned " + totalPoints
                           + " total points.");
    }
```

# Roadmap: Foundational Programming Concepts



# Outline

- Arrays
  - Array reviews
  - Arrays and reference semantics

# Recap: Arrays

- Arrays are similar to primitives in most ways, e.g.,
  - Pass arrays as parameters, e.g.,

```
public static double average(int[] numbers)
```

- Differences between arrays and primitives e.g.,
  - Declare and initialize

```
int[] numbers = new int[10];
char[] letterGrades = {'A', 'B', 'C', 'D', 'F'};
```

  - Syntax to access array elements, e.g.,

```
numbers[1] = 3;
numbers[2] = 2;

numbers[ numbers[2] ] = 42;
```

  - Reference vs value semantics

# Outline

- Arrays
  - Array reviews

# Coupon-Collector Problem

- Given N different types of chocolates, and you get one random type each day. Simulate how many days you need in order to have (at least) one of each type.

Coupon优惠券

# Coupon-Collector Problem: Pseudo Code

```
// assume items are numbered 0 to N - 1

repeat when not collected all distinct items
  pick a random item
  if item is new
    record new item
```

How?

# Coupon-Collector Problem: Pseudo Code

```
// assume items are numbered 0 to N - 1
int distinct = 0;

while distinct < N

  pick a random item
  if item is new
    record new item
    distinct ++;
```

How?

# Coupon-Collector Problem: Pseudo Code

```
// assume items are numbered 0 to N - 1
int distinct = 0;
boolean[] has = new boolean[N];

while distinct < N

  pick a random item

  if item is new

    has[item] = true; // record new item
    distinct ++;
```

# Coupon-Collector Problem

```
public class CouponCollector {
   public static void main(String[] args) {
      int N = Integer.parseInt(args[0]);
      int cnt = 0;    // number of received collected
      int distinctCnt = 0;     // number of distinct

      boolean[] has = new boolean[N]; // array keeping state

      while (distinctCnt < N) {
         int val = rand(0, N-1);
         cnt++;
         if (!has[val]) {                // find state var
            distinctCnt++;
            has[val] = true;             // update state var
         }
      }

      // all N distinct cards found
      System.out.println(cnt);
}}
```

# Array Merge Question

- Write a method `merge` that accepts two arrays of integers and returns a new array containing all elements of the first array followed by all elements of the second.

```
int[] a1 = {12, 34, 56};
int[] a2 = {7, 8, 9, 10};

int[] a3 = merge(a1, a2);
System.out.println( Arrays.toString(a3) );
// [12, 34, 56, 7, 8, 9, 10]
```

# Array Merge

```
// Returns a new array containing all elements of a1
// followed by all elements of a2.
public static int[] merge(int[] a1, int[] a2) {
    int[] result = new int[a1.length + a2.length];

    for (int i = 0; i < a1.length; i++) {
        result[i] = a1[i];
    }
    for (int i = 0; i < a2.length; i++) {
        result[a1.length + i] = a2[i];
    }

    return result;
}
```

# Array Merge 3 Question

- Write a method `merge3` that merges 3 arrays similarly.

```
int[] a1 = {12, 34, 56};
int[] a2 = {7, 8, 9, 10};
int[] a3 = {444, 222, -1};

int[] a4 = merge3(a1, a2, a3);
System.out.println( Arrays.toString(a4) );
// [12, 34, 56, 7, 8, 9, 10, 444, 222, -1]
```

# Array Merge 3

```
// Returns a new array containing all elements of a1,a2,a3.
public static int[] merge3(int[] a1, int[] a2, int[] a3) {
    int[] a4 = new int[a1.length + a2.length + a3.length];

    for (int i = 0; i < a1.length; i++) {
        a4[i] = a1[i];
    }
    for (int i = 0; i < a2.length; i++) {
        a4[a1.length + i] = a2[i];
    }
    for (int i = 0; i < a3.length; i++) {
        a4[a1.length + a2.length + i] = a3[i];
    }

    return a4;
}


// Shorter version that calls merge.
public static int[] merge3(int[] a1, int[] a2, int[] a3) {
    return merge(merge(a1, a2), a3);
}
```

Discussion: Which version do you use?

# Complexity Analysis

- V1
  - Creation of array (static complexity)
    - One array of size N1 + N2 + N3
  - Copy values (dynamic)
    - N1 + N2 + N3 values

- V2
  - Creation of array
    - First size N1 + N2; second size (N1 + N2) + N3
  - Copy values
    - First N1 + N2
    - Then N1 + N2 + N3

# Outline

- Arrays
  - Array reviews
  - Arrays and reference semantics

# Motivation: Primitive swap

```
public static void main(String[] args) {
    int a = 1;
    int b = 2;
    System.out.println(a + " " + b); // ?
    swap(a, b);
    System.out.println(a + " " + b); // ?
}

public static void swap(int a, int b) {
    // swap a and b's values
    int temp = a;
    a = b;
    b = temp;
    System.out.println(a + " " + b); // ?
}
```

# Motivation: Array Reversal

```
public static void main(String[] args) {
    int[] a = {1, 2};
    System.out.println( Arrays.toString(a) );
    reverse(a);
    System.out.println( Arrays.toString(a) );
}

public static void reverse(int[] a) {
    for (int i = 0; i < a.length / 2; i++) {
        int temp = a[i];
        a[i] = a[a.length-1-i];
        a[a.length-1-i] = temp;
    }
}
```

# Value Semantics and Reference Semantics

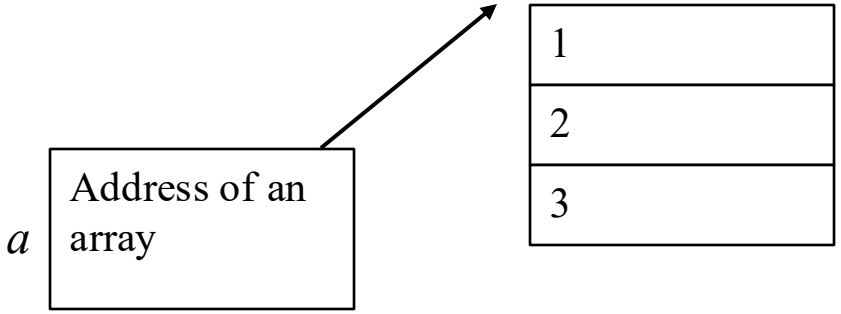
- Primitive data types use value semantics: variable stores value

```
int x = 5;
```


- Non primitive data types (e.g., arrays and objects) use reference semantics: variable stores reference address

Reference 引用

```
int[] a = {1, 2, 3};
```




- See printVariable() of ArrayAsRef.java

# References

- Why reference variables
  - Efficiency. Copying large objects/arrays slows down a program
  - Sharing. It's useful to share an object/array's data among methods

# Value/Reference Semantics and Assignment

- When variable `a` is assigned to variable `b`:

  `b = a`

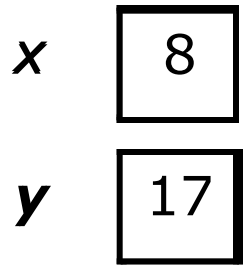
  it is always that the content of `a` is copied to `b`

  - if a value type, then it is the **value** that is copied
  - if a reference type, then it is the **reference** that is copied, `b` becomes an alias of `a`

# Example: Value Assignment

- Modifying the value of one value variable does not affect others.

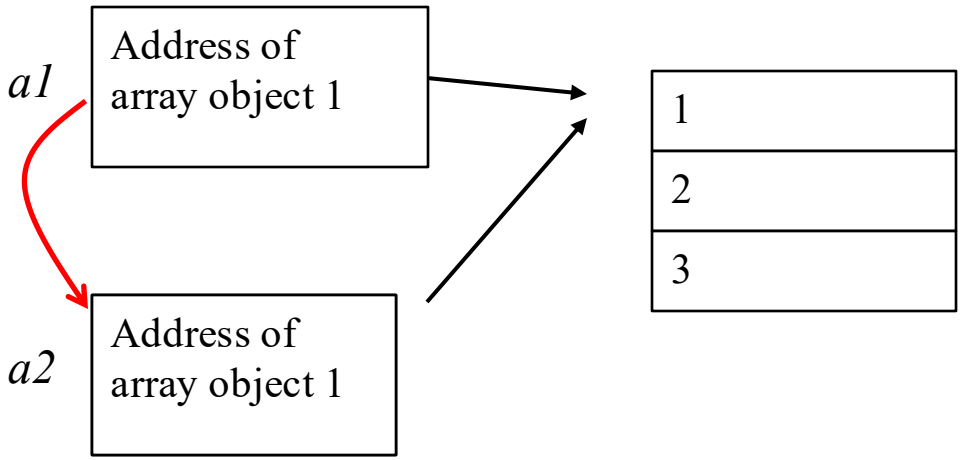
```
int x = 5;
int y = x;        // x = 5, y = 5
y = 17;           // x = 5, y = 17
x = 8;            // x = 8, y = 17
```



# Example: Reference Assignment

- Modifying an object/array through one reference variable changes the object/array and hence all references to the same object/array see the changes.

```
int[] a1 = {1, 2, 3};
int[] a2 = a1;   // same array
```



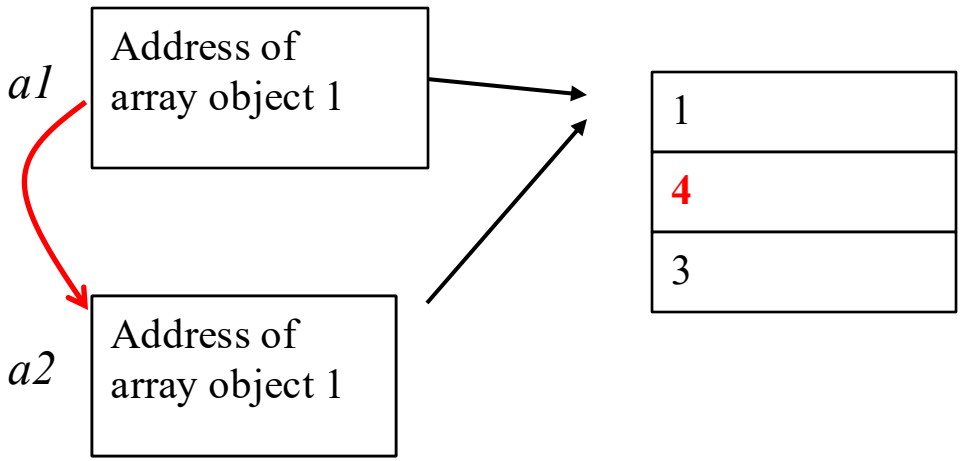

Content copied, but content is **address**

```
a2[1] = 4;
```

# Example: Reference Assignment

- Modifying an object/array through one reference variable changes the object/array and hence all references to the same object/array see the changes.

```
int[] a1 = {1, 2, 3};
int[] a2 = a1;   // same array
```




Content copied, but content is **address**

```
a2[1] = 4;
```

# Special Case: Value/Reference Semantics and Parameter Passing

- Each time a method is called, the *actual argument* in the invocation is copied into the corresponding *formal argument*
  => if a parameter is a reference type, then the actual argument and the formal argument now refer to the same object

# Example

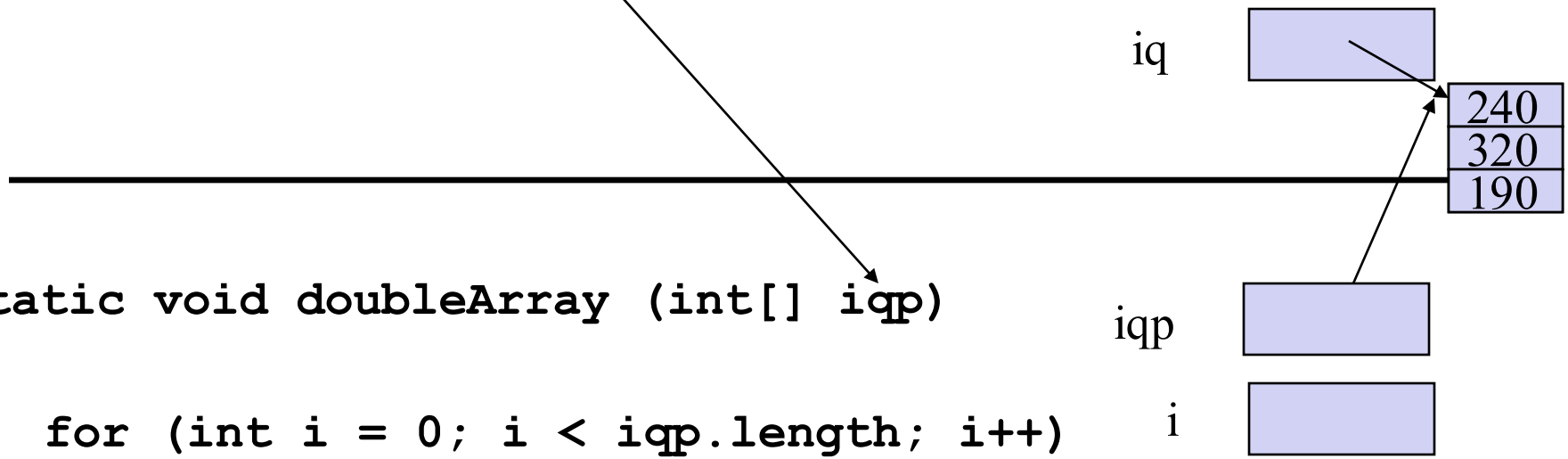
```
public static void main(String[] args) {
      int[] iq = {120, 160, 95};
      doubleArray(iq);
      System.out.println(Arrays.toString(iq));
}
```

---

```
static void doubleArray (int[] iqp)
{
   for (int i = 0; i < iqp.length; i++)
       iqp[i] *= 2;

}
```

# Example: Illustration

```
public static void main(String[] args) {
      int[] iq = {120, 160, 95};
      doubleArray(iq);
      System.out.println(Arrays.toString(iq));
}
```




```
static void doubleArray (int[] iqp)
{
   for (int i = 0; i < iqp.length; i++)
       iqp[i] *= 2;

}
```

# Exercise

- Write a method `swapTwo` that accepts an array of integers and two indexes and swaps the elements at those indexes.

```
int[] a1 = {12, 34, 56};
swapTwo(a1, 1, 2);
System.out.println(Arrays.toString(a1)); // [12, 56, 34]
```

```
// Swaps the values at the given two indexes.
public static void swapTwo(int[] a, int i, int j) {
    int temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}
```

# Exercise

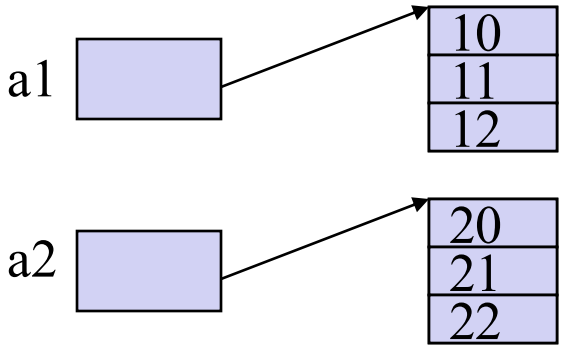
- Write a method `swapAll` that accepts two same-size arrays of integers as parameters and swaps their entire contents.

```
int[] a1 = {10, 11, 12};
int[] a2 = {20, 21, 22};
swapAll(a1, a2);
System.out.println(Arrays.toString(a1));  // [20, 21, 22]
System.out.println(Arrays.toString(a2));  // [10, 11, 12]
```

```
// Does this method swap the entire contents of
// a1 with those of a2?
public static void swapAll1(int[] a1, int[] a2) {
    int[] temp = a1;
    a1 = a2;
    a2 = temp;
}
```

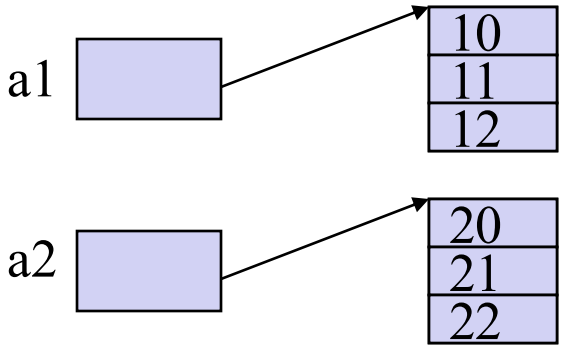
# Why it does not work? Init.

```
public static void main(String[] args) {
      int[] a1 = {10, 11, 12};
      int[] a2 = {20, 21, 22};

}
```



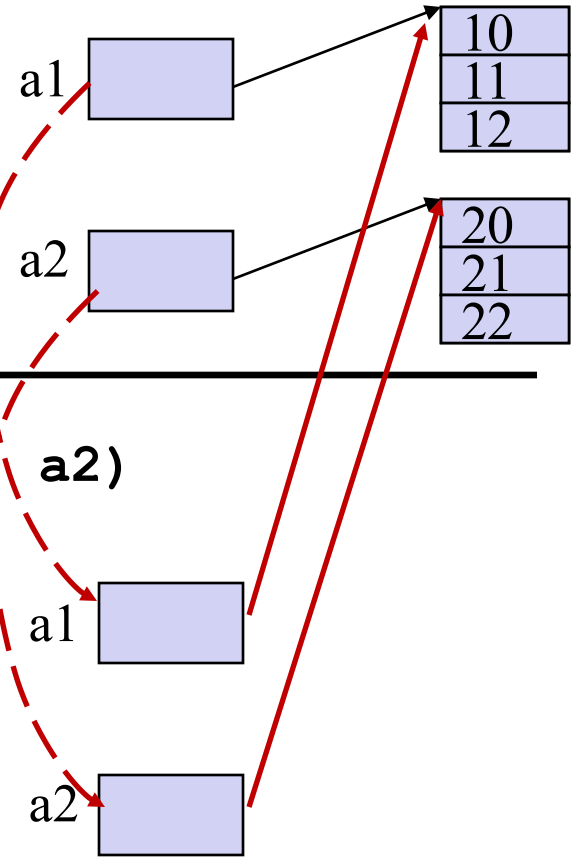
# Why it does not work? Invoke

```
public static void main(String[] args) {
      int[] a1 = {10, 11, 12};
      int[] a2 = {20, 21, 22};
      swapAll (a1, a2);
}
```



# Why it does not work? Invoke

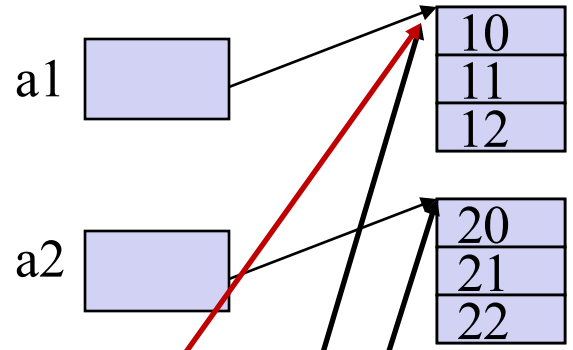
```
public static void main(String[] args) {
      int[] a1 = {10, 11, 12};
      int[] a2 = {20, 21, 22};
      swapAll (a1, a2);
}
```



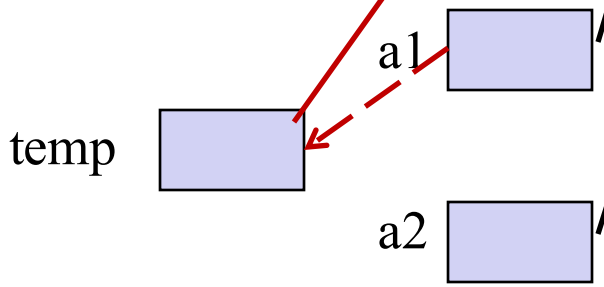

```
static void swapAll(int[] a1, int[] a2)
{



}
```

# Why it does not work? Swap (temp = a1)

```
public static void main(String[] args) {
      int[] a1 = {10, 11, 12};
      int[] a2 = {20, 21, 22};
      swapAll (a1, a2);
}
```



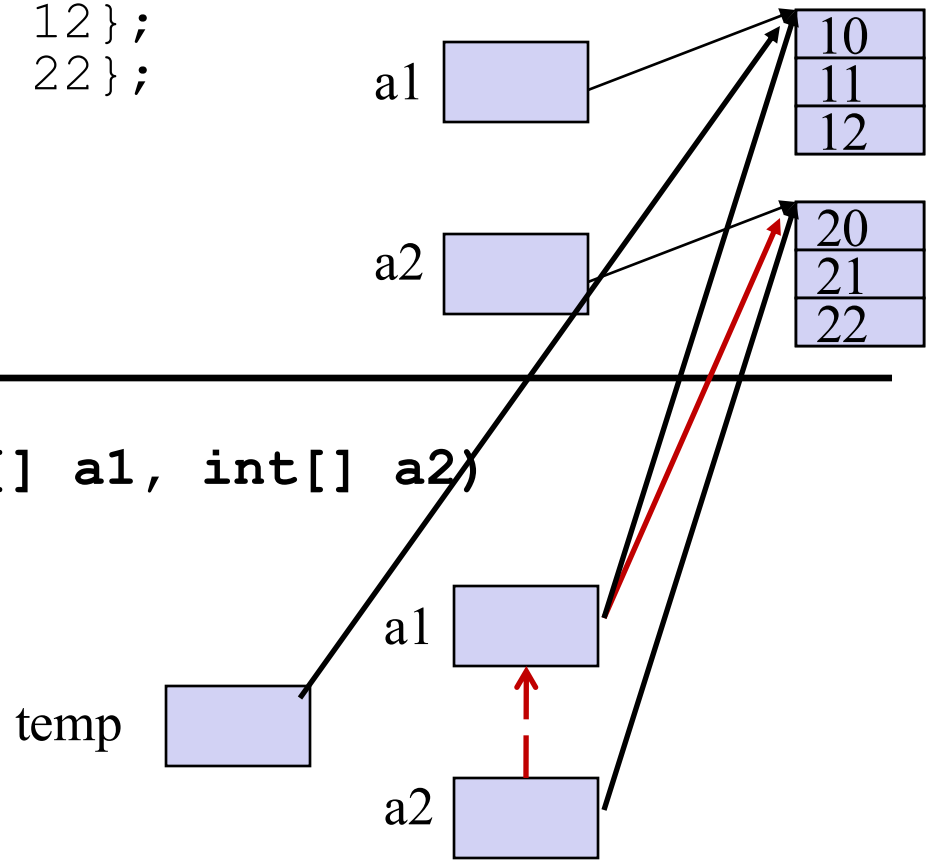

```
static void swapAll(int[] a1, int[] a2)
{
    int[] temp = a1;


}
```



# Why it does not work? Swap (a1 = a2)

```
public static void main(String[] args) {
      int[] a1 = {10, 11, 12};
      int[] a2 = {20, 21, 22};
      swapAll (a1, a2);
}
```

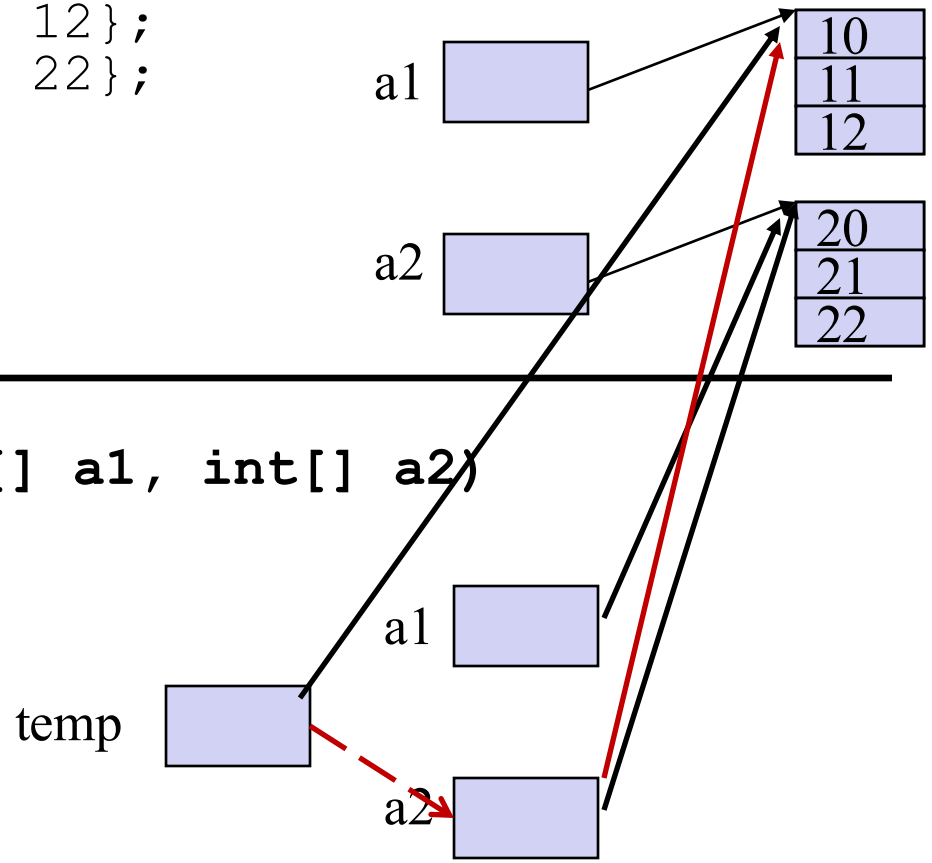
```
static void swapAll(int[] a1, int[] a2)
{
    int[] temp = a1;
    a1 = a2;

}
```



# Why it does not work? Swap (a2 = temp)

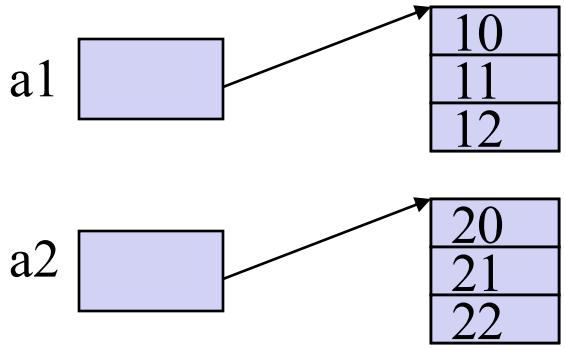
```
public static void main(String[] args) {
      int[] a1 = {10, 11, 12};
      int[] a2 = {20, 21, 22};
      swapAll (a1, a2);
}
```

```
static void swapAll(int[] a1, int[] a2)
{
    int[] temp = a1;
    a1 = a2;
    a2 = temp;
}
```



# Why it does not work? After swapAll

```
public static void main(String[] args) {
      int[] a1 = {10, 11, 12};
      int[] a2 = {20, 21, 22};
      swapAll (a1, a2);
}
```



# Solution

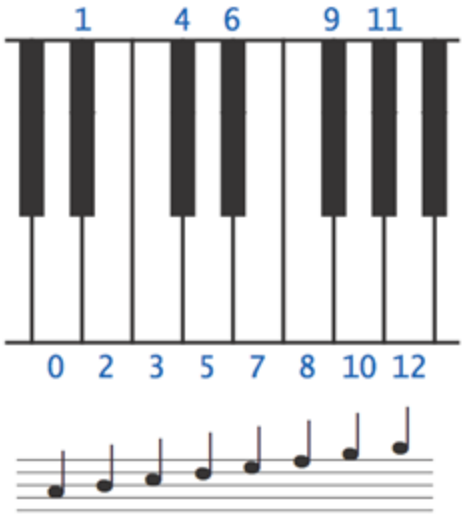
```
// Swaps the entire contents of a1 with those of a2.
public static void swapAll(int[] a1, int[] a2) {
    for (int i = 0; i < a1.length; i++) {
        int temp = a1[i];
        a1[i] = a2[i];
        a2[i] = temp;
    }
}
```

# Outline

- Arrays
  - Array reviews
  - Arrays and reference semantics
  - Arrays and digital audio

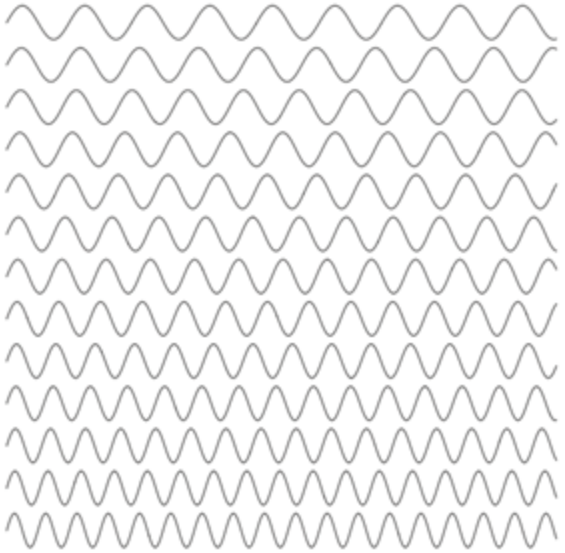
# Sound

- Sound is the perception of vibrations in our eardrums.




| 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 |

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



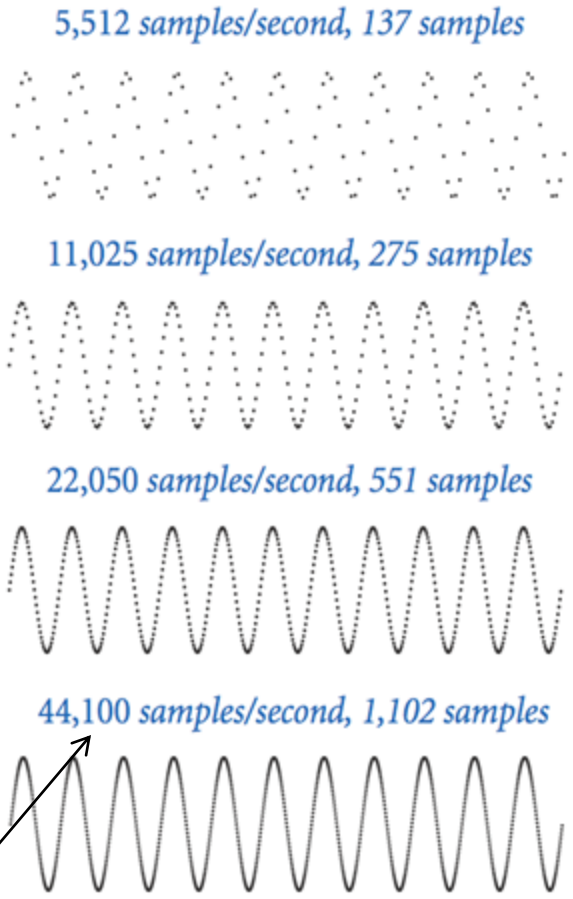
Perception 感知
Vibrations震动
Eardrums 耳膜

*Notes, numbers, and waves*

$y(t) = sin(2\pi ft)$

# Digital Audio

- Computer represents audio as a sequence of samples (how?)
- When these samples are played in succession, it approximates the continuous sound




Audio CD

in succession 陆续

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

# Use Digital Audio

- Java has a complex digital audio system
- We use the StdAudio library for playing and saving sound

Typical use: generate a sequence of samples, each for 1/44100 sec.

| 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* |

library developed from book:

# Example: Generate Musical Tone

- Play Concert A (440hz) for 1.5 seconds using StdAudio.

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

```
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);
```

# Exercise: Generate Musical Tone from Array

- Gen a tone from multiple frequencies

# Example: PlayThatTune

- Read in pitches and durations from standard input and sonify with StdAudio.

```
% more elise.txt
7 .125
6 .125
7 .125
6 .125
7 .125
2 .125
5 .125
3 .125
0 .25
```

```
% java PlayThatTune < elise.txt
```

# PlayThatTune

```
public class PlayThatTune {
   public static void main(String[] args) {
      Scanner console = new Scanner( System.in );
      while (console.hasNextInt()) {
         int pitch = console.nextInt();
         double seconds = console.nextDouble();
         double hz = 440.0 * Math.pow(2, pitch / 12.0);

         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);
      }
   }
}
```

Same as before

# Outline

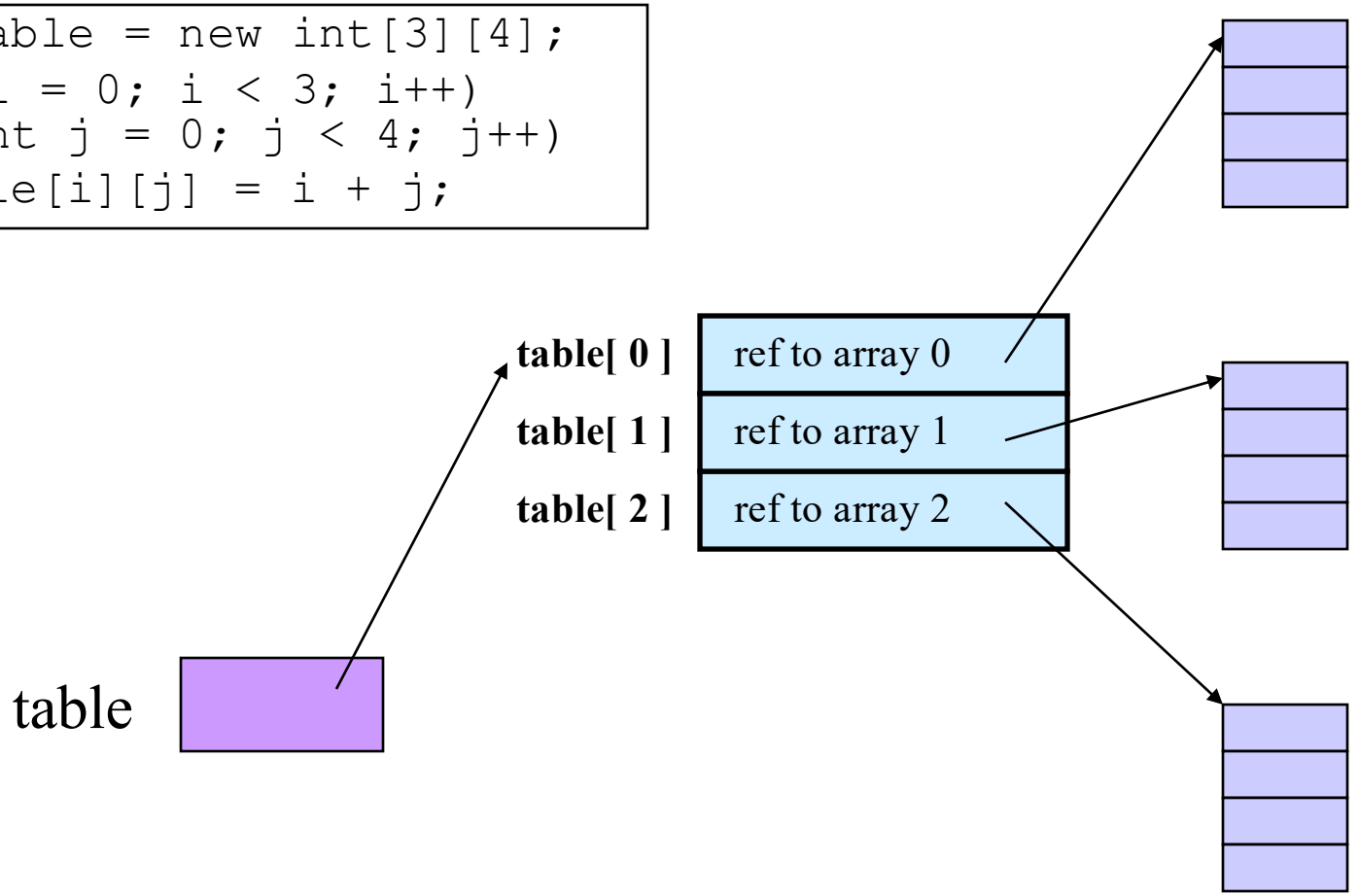
- Arrays
  - Array reviews
  - Arrays and reference semantics
  - Arrays and digital audio
  - Multidimensional arrays

# Motivation

- Arrays are used in many settings, e.g.,
  - 1d array in sound
  - 2d in image processing, graph processing

# Understanding Two Dimensional Array: An Array of Arrays

```
int[][] table = new int[3][4];
for (int i = 0; i < 3; i++)
   for (int j = 0; j < 4; j++)
      table[i][j] = i + j;
```



# Outline

- Admin
- Arrays
  - Array exercises
  - Arrays and reference semantics
  - Arrays and digital audio
  - Multidimensional arrays
    - Regular 2D arrays

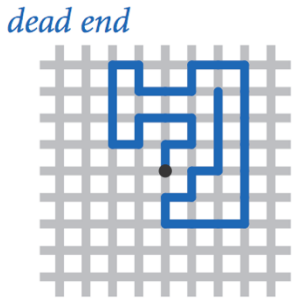
# Using 2-Dimensional Arrays

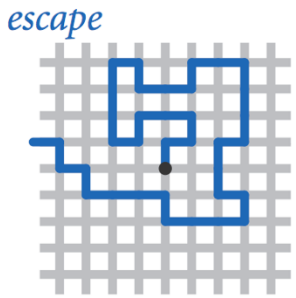
- In most cases, you deal with regular 2D arrays, and hence no need to worry about the internal storage structure of 2-D arrays

```
final int NROWS = 3;
final int NCOLUMNS = 4;

int[][] table = new int[NROWS][NCOLUMNS];
for (int i = 0; i < NROWS; i++)
   for (int j = 0; j < NCOLUMNS; j++)
      table[i][j] = i + j;
```

# Example: Simulate Self-Avoiding Walk

- Model.
  - N-by-N lattice(格子).
  - Start in the middle.
  - Each step randomly moves to a neighboring intersection, if a previously moved intersections, no move.
  - Two possible outcomes:
    - dead end and escape.



# Key Design Points

- How to represent the state of the visited status of the N-by-N lattice?

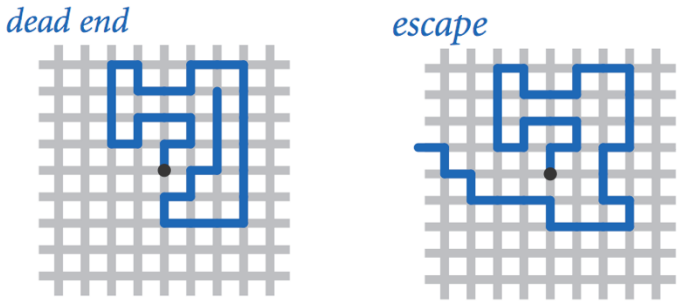
```
boolean[][] visited = new boolean[N][N];
```

- How to detect if (x,y) is a terminating state:
  - dead end (trap):

```
visited[x+1][y] && visited[x-1][y] &&
visited[x][y-1] && visited[x][y+1]
```

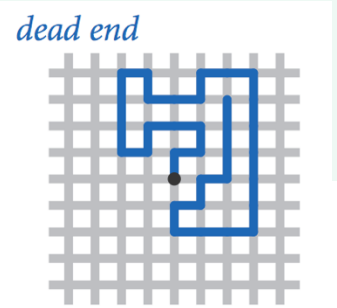
  - escape

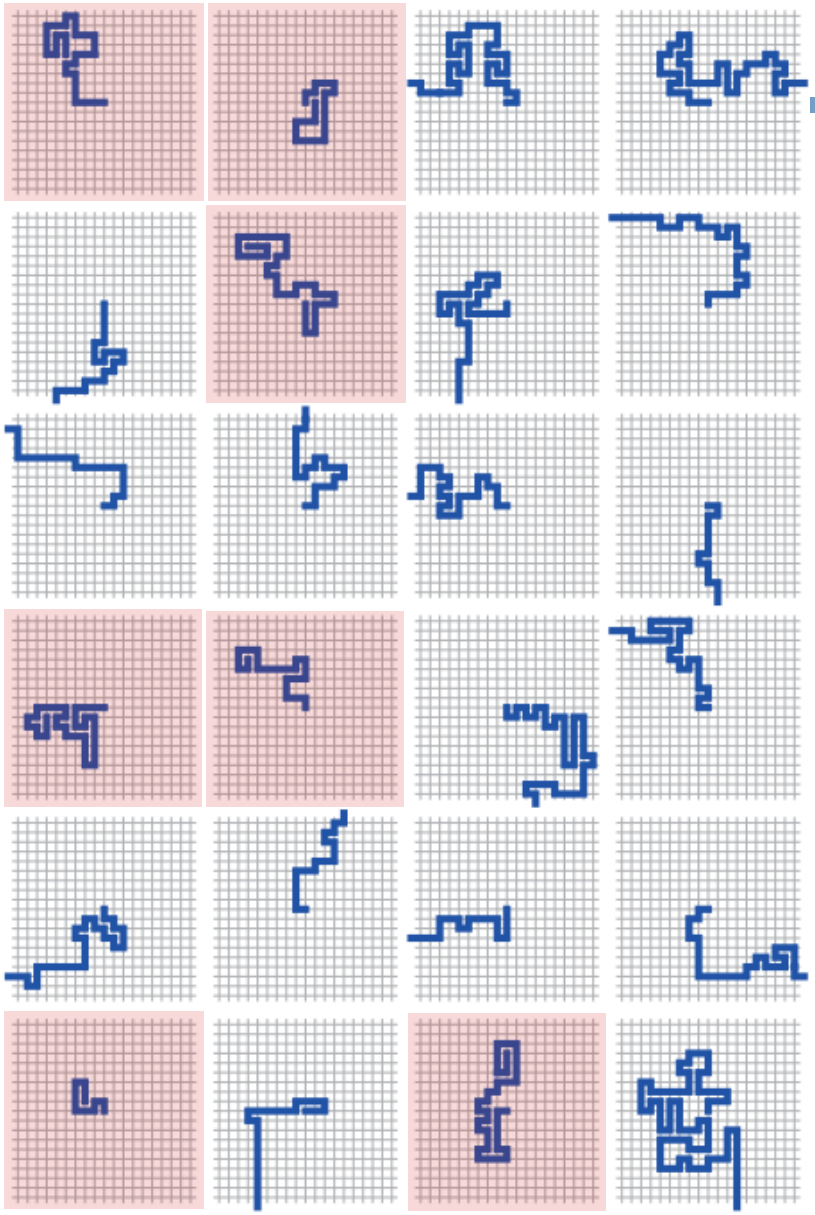
```
x == 0 || x == N-1 || y == 0 || y == N-1
```



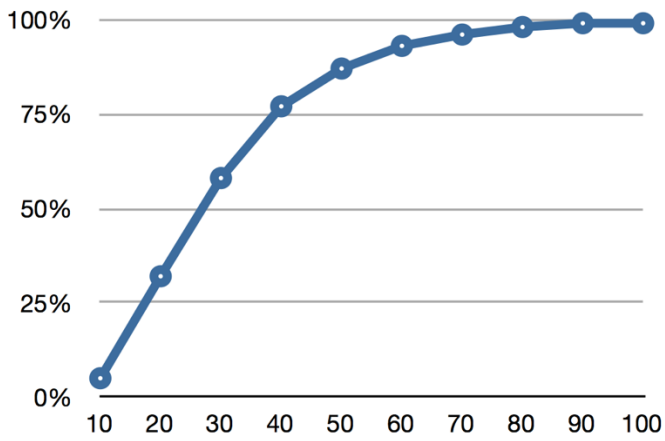
# Self-Avoiding Random Walk

```
// read in lattice size N as command-line argument.
// read in number of trials T as command-line argument.

// repeat T times:

    // initialize (x, y) to center of N-by-N grid.

    // repeat as long as (x, y) is not escape or trap

       // mark (x, y) as visited.
       // take a random step, updating (x, y).

    // if not escape
         increase #deadEnds
// print fraction of dead ends.
```





```
% java SelfAvoidingWalks 4 100000
0% dead ends

% java SelfAvoidingWalks 8 100000
1% dead ends

% java SelfAvoidingWalks 16 100000
20% dead ends
...

% java SelfAvoidingWalks 64 100000
94% dead ends
```

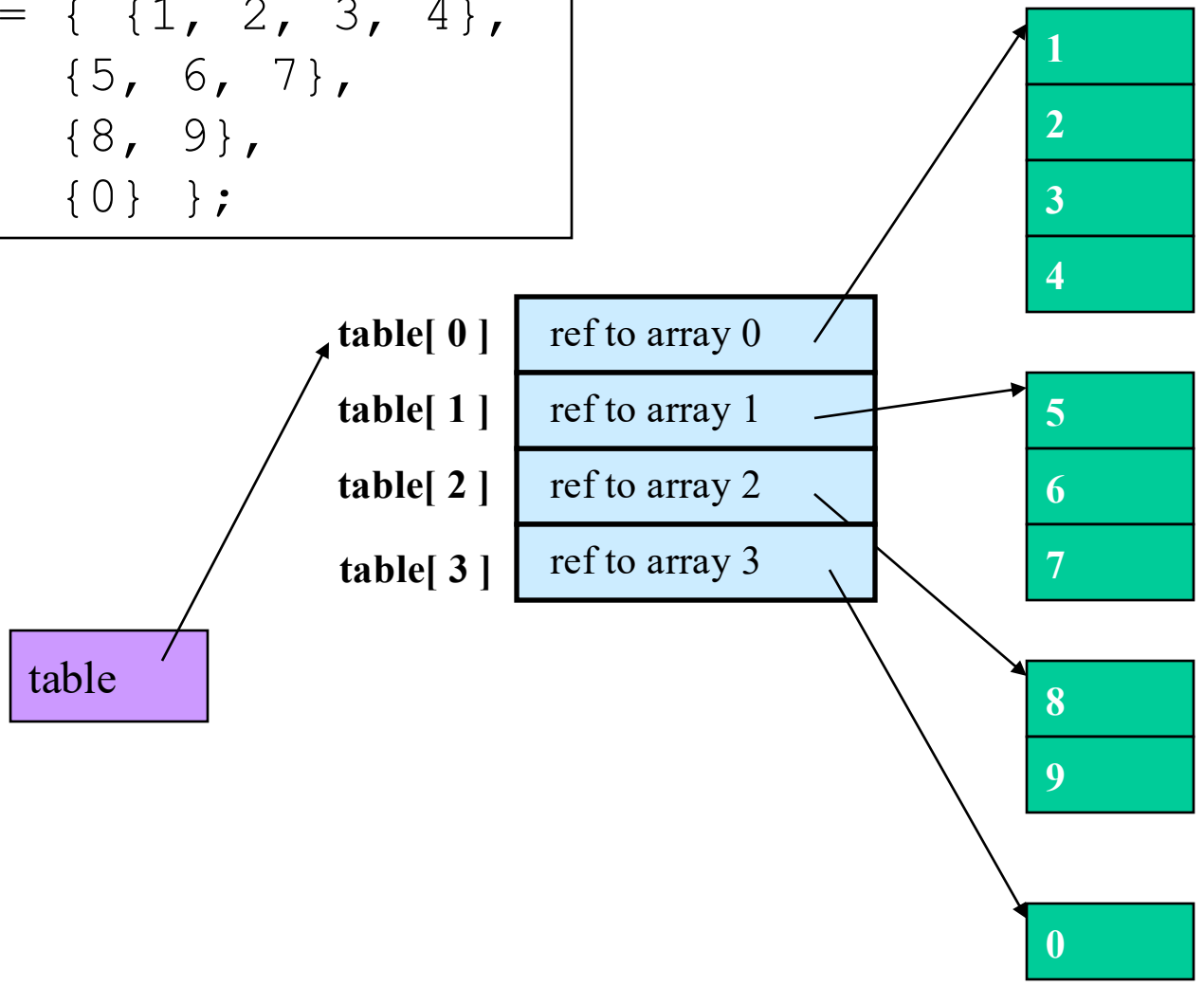


# Outline

- Admin
- Arrays
  - Array exercises
  - Arrays and reference semantics
  - Arrays and digital audio
  - Multidimensional arrays
    - Regular 2D arrays
    - Irregular 2D arrays

# Irregular Two-Dimensional Array

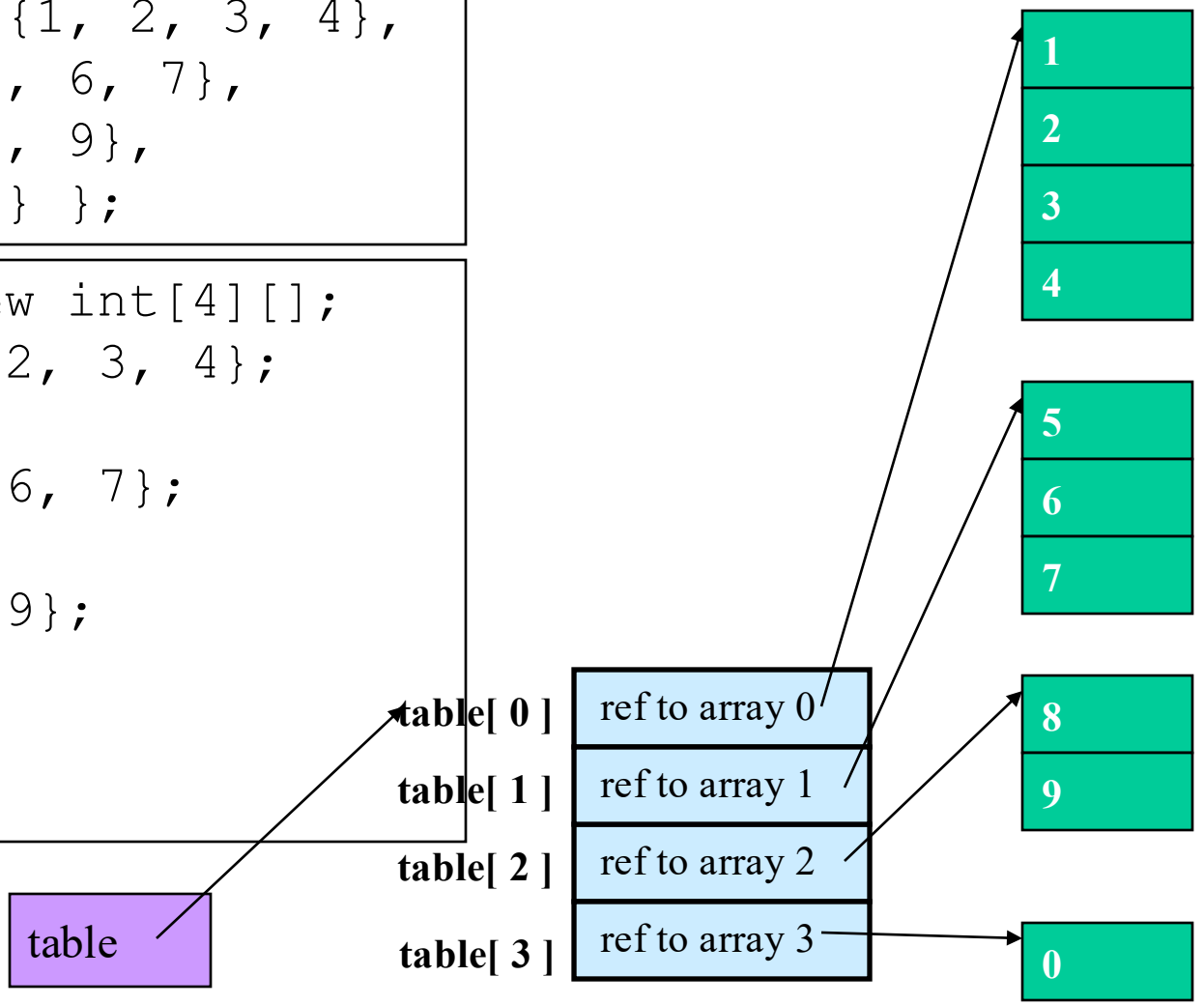
```
int[][] table = { {1, 2, 3, 4},
                  {5, 6, 7},
                  {8, 9},
                  {0} };
```



# Irregular Two-Dimensional Array

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

```
int[][] table = new int[4][];
int[] temp0 = {1, 2, 3, 4};
table[0] = temp0;
int[] temp1 = {5, 6, 7};
table[1] = temp1;
int[] temp2 = {8, 9};
table[2] = temp2;
int[] temp3 = {0};
table[3] = temp3;
```



# Access Irregular 2D Arrays

```
public class Test2DArray
{
   public static void main(String[] args)
   {
      int[][] days = { {1, 2, 3, 4},
                       {5, 6, 7},
                       {8, 9},
                       {0} };

      for (int i = 0; i < days.length; i++)
      {
         for (int j = 0; j < days[i].length; j++)
            System.out.print( days[i][j] );
         System.out.println ();
      }
   } // end of main
} // end of class
```

# PageRank
## [Sergey Brin and Larry Page, 1998]

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

# PageRank
## [Sergey Brin and Larry Page, 1998]


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

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

# Offline Exercise

- Think about how to implement pagerank.