We number elements of a larger collection

the pages of a book
   (chapters and sections too)
days of a month
years

small collections—often use distinct names
days of the week are Mon, Tues, etc.
large collections—numbering is handy
pages of a book
Arrays

A collection of primitive values or objects

• Useful in programs that manipulate relatively large collections of similar data
• Number items in the data collection instead of using distinct names
• Each item in the collection is associated with a number called its index

Declaring Array Names

• The collection needs a name
  Ex: say that page is the name of an array and its items are Strings corresponding to pages of a book.
  We say page[12] to access 12th element of the collection
• To declare page as the name of the collection
  private String[ ] page
Types of Array Elements

• Array elements may be any type
  private FilledOval[ ] piece;
  (the ovals are pieces on a checkerboard)
• But all elements of a given array must have same type.

  No mixing of types!
  (We’ll figure out how later)

Creating an Array

• Declaration of a collection’s name does not create the collection or the items in it
  ex: private FilledOval[ ] piece;
• Need to
  – construct array
  – construct individual Filled Ovals
Constructing an Array

- Need to provide
  - Types of items
  - Total size

```
piece = new FilledOval[24];
```

Constructs the collection - not the individual elements

---

Array Constructions in Declarations

- Common to use array constructions as initial values in array name declarations

Ex 1: our array of checkers
```
private FilledOval[] piece = new FilledOval[24];
```

Ex 2: Array of Strings to hold text of pages of a book
```
private String[] page = new String[723];
```
assuming the book has 723 pages.
Array Elements

After

private FilledOval[] piece=new FilledOval[24];
we have an array but no FilledOvals.

piece[3].setColor(Color.RED);
will result in a null pointer exception

Use an assignment statement to associate members of
an array with index values

piece[3]=new FilledOval(checkerLeft, checkerTop,
SIZE, SIZE, canvas);

Indexed Variables

• An array name followed by an index value is
called an indexed variable
• Can be used in any context where a variable
of the array’s element type can be used
• Java arrays use 0 as the first index value
• Last usable index is 1 less than size of array.

piece[3] refers to fourth array element
Array Initializers

These combine creation of an array and association of values with its elements into a single step
- List values to be associated with array’s elements
- Separate values by commas
- Surround with curly braces

```java
private int[] monthLength = {31, 28, 31, 30, 31, 30, 31, 30, 31};
```

Using Arrays: A Triangle Class

Say you want to define a Triangle class
- need 3 instance variables to refer to the three Lines that make up a triangle- or can use an array

```java
private Line[] edge = new Line[3];
initially, edge[0], edge[1], edge[2] all have null as their values
```
A Simple Triangle Class

- Simplest way to describe a triangle is to provide coordinates of vertices
- Define constructor to expect 4 parameters: 3 Locations and the canvas

```java
public class Triangle {
    private Line[] edge = new Line[3];
    public Triangle(Location vert1, Location vert2, Location vert3, DrawingCanvas canvas) {
        edge[0] = new Line(vert1, vert2, canvas);
        edge[1] = new Line(vert2, vert3, canvas);
        edge[2] = new Line(vert3, vert1, canvas);
    }
    // additional method declarations
    ...
}
```

If Triangle constructor invoked with Locations (100, 50), (50, 150), (250, 50):

Note that elements of the edge array refer to components of a Triangle
Additional Triangle Methods

Might want to include implementations of methods like move, setColor, or hide

Could write

```java
public void hide() {
    edge[0].hide();
    edge[1].hide();
    edge[2].hide();
}
```

Or even better

```java
public void hide() {
    for ( int edgeNum = 0; edgeNum < edge.length; edgeNum++ ) {
        edge[edgeNum].hide();
    }
}
```

- The desired element of an array can be specified by a variable or any other expression that describes an int
- Name of an array variable followed by .length produces the number of elements in the array
Array-processing Loops

General Form:
for ( int elementPosition = 0; elementPosition < array.length; 
    elementPosition++ ) {
    // perform desired operation on array[ elementPosition ]
    ... 
}

Why loop?
- Flexibility- triangles, hexagons, etc can all be handled the same way
- Short, simple, and descriptive

Additional examples

```java
public void move( double dx, double dy ) {
    for ( int edgeNum = 0; edgeNum < edge.length; edgeNum++ ) {
        edge[edgeNum].move( dx, dy );
    }
}
```
```
public void show() {
    for ( int edgeNum = 0; edgeNum < edge.length; edgeNum++ ) {
        edge[edgeNum].show();
    }
}
```
Arrays are Objects

- can pass entire arrays as parameters
- can write methods that return arrays

```java
public Location[] getVertices() {
    Location[] result = new Location[edge.length];

    for (int edgeNum = 0; edgeNum < edge.length; edgeNum++) {
        result[edgeNum] = edge[edgeNum].getStart();
    }

    return result;
}
```

Enhanced for loop (Java 1.5)

Makes it easier to iterate through arrays

AKA: “foreach” or “forAllInOrder” loop

```java
public void hide() {
    for (Line nextLine: edge) {
        nextLine.hide();
    }
}
```
Gathering Information

• Often useful to gather info about a collection rather than process its elements independently.
  
  Ex 1. Determining the perimeter of a Triangle
  
  Ex 2. Computing traffic statistics

Computing stats in a traffic radar trailer
Speeding Violation Patterns

• Say we want to determine the number of speeders passing the trailer during each of 24 hrs.
• 24 numbers to count speeders can be kept in array
  private int [ ] speedersAt = new int[24];
• speedersAt[hour] accesses number of speeders at “hour” (using 24-hr clock)

Program Organization

• RadarController
  – acts as “controller”
  – event-handling method to be invoked when vehicle detected
• RadarStats
  – responsible for recording stats
  – update speedersAt when speeder detected
  – provide access to collected statistics
Counting Speeders

- Method in RadarStats class
- Invoked by RadarController when vehicle detected

```java
public void vehicleReport( double speed, int hour, int minute ) {
    if ( speed > speedLimit ) {
        speedersAt[hour]++;
    }
}
```

Remember that hour is based on a 24-hr clock

Summing Values in an Array

```java
private int speedersSeen() {
    int total = 0;
    for ( int hour = 0; hour < speedersAt.length; hour++ ) {
        total = total + speedersAt[hour];
    }
    return total;
}
```

Note the use of total to accumulate the sum
Reporting Stats

- number of speeders detected at different times of the day
- percent speeders in each hour

A Simpler Version
A Simple Histogram

- Loop is similar to loop of Triangle
- Operation: to draw bars corresponding to hours

```java
public void drawHistogram() {
    double barHeight;
    double totalSpeeders = speedersSeen();
    for (int hour = 0; hour < speedersAt.length; hour++) {
        barHeight = (speedersAt[hour]/totalSpeeders)*graphHeight;
        new FilledRect( graphLeft + hour*barWidth,
                        graphBottom - barHeight,
                        barWidth-1,
                        barHeight,
                        canvas
                       );
    }
}
```

Assume

- `graphHeight` is height of area in which graph is to be drawn
- `graphLeft` is x coordinate of bottom edge of graph
- `graphBottom` is y coordinate of bottom edge of graph
- `barWidth` is width of a single bar
Simple Histogram Output
Output of drawHistogram likely to look like this

```
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
</tr>
<tr>
<td>---------------</td>
</tr>
<tr>
<td></td>
</tr>
<tr>
<td>---------------</td>
</tr>
</tbody>
</table>
\```

- At any hour, the number of speeders on average is $\frac{1}{24}$ of the total number of speeders
- Bars on average are $\frac{1}{24}$ of the available vertical space

Finding the Largest Value in an Array

- Begin with the first value and then look hour by hour for a new maximum
- Variable max is equal to the largest number in the array so far. If a new, larger number is in the array, then max changes

```java
private int maxSpeeders() {
    int max = speedersAt[0];
    for ( int hour = 1; hour < speedersAt.length; hour++ ) {
        if ( speedersAt[hour] > max ) {
            max = speedersAt[hour];
        }
    }
    return max;
}
```
Review

• Arrays are collections of primitive values or objects
• Learned how to
  – Declare them
  – Create them
  – Refer to items in them
  – Process all items in them in some way (move, hide)
  – Gather information from them (sum, max)

Collections With Variable Sizes

• A new application: timing and scoring of a cross-country race
Team Score

- Add placements of a team’s four fastest racers.
- Last digit of runner’s bib indicates team
- Team 1’s score = 1+2+4+6 = 13

Program Organization

- RaceController
  - Extension of WindowController
  - User interface to enable officials to enter timing data
- RaceStatistics
  - Uses array to keep track of data entered
  - Methods to compute team score, etc.
Parallel Arrays vs. Arrays of Objects

Need to keep track of pairs of bib numbers and times
- Two separate arrays
  - Arrays are “parallel arrays,” one number from one associated with one from other
    
    ```java
    private int[] bibNumber;
    private String[] elapsedTime;
    ```
- Single array of racer information
  - Assumes definition of a RacerInfo class
    ```java
    private RacerInfo[] racer;
    ```

RacerInfo Class

```java
public class RacerInfo {
    private int bibNumber;
    private String time;
    public RacerInfo(int number, String finishingTime) {
        bibNumber = number;
        time = finishingTime;
    }
    public int getBib() {
        return bibNumber;
    }
    public String getTime() {
        return time;
    }
    public int getTeam() {
        return bibNumber % 10;
    }
}
```
Keeping Track of Size

- must specify size to construct racer array.
- often perfect size info unknown, but can give upper limit
- use upper limit as size
- separately keep track of actual number of items in array

```java
private static final int TEAMSIZE = 100;
private static final int TEAMSINMEET = 3;

private RacerInfo[ ] racer = new RacerInfo[TEAMSIZE*TEAMSINMEET];

private int racerCount;
```
Adding Array Entries

- Check that there’s room left
- Add new item to the end
- Update count of items

```java
public void addRacer( int bib, String time ) {
    if ( racerCount < racer.length ) {
        racer[racerCount] = new RacerInfo( bib, time );
        racerCount++;
    }
}
```

Iterating Through Collection of Variable Size

- Similar to earlier for loops
- Use array size variable to determine when to stop
- Ex. To create a string of race results for printing

```java
public String individualResults() {
    String results = "";
    for ( int place = 0; place < racerCount; place++ ) {
        results = results +
        (place+1) + ". " +
        "Racer" + racer[place].getBib() + " " +
        "/n";
    }
    return results;
}
```
Finding an Element

- Use a for loop
- Keep going as long as not found and items left to consider

```java
public int getPlacement( int bib ) {
    int result = -1
    for ( int place = 0;
         place < racerCount && result == -1
         place++
    ) {
        if ( racer[place].getBib() == bib ) {
            result = place+1;
        }
    }
    return result;
}
```

Alternate Version

```java
public int getPlacement( int bib ) {
    for ( int place = 0; place < racerCount; place++ ) {
        if ( racer[place].getBib() == bib ) {
            return place+1;
        }
    }
    return -1;
}
```
Computing a Team’s Score

A combination of
– Finding an element
– Adding to a running total

```java
public int teamScore( int teamNo ) {
    int racersCounted = 0;
    int score = 0;
    for ( int place = 0;
            place < racerCount && racersCounted < 4;
            place++ ) {
        if ( racer[place].getTeam() == teamNo ) {
            racersCounted++;
            score = score + (place + 1);
        }
    }
    if ( racersCounted < 4 ) {
        score = -1;
    }
    return score;
}
```

Ordered Arrays

- Previous examples assumed
  – Array items ordered by elapsed time
  – Items supplied to array in correct order

*What if we want to add or delete items and guarantee that correct order is maintained?*
Adding to an Ordered Array

The racer that should be associated with index 4 is missing

Runner with bib 200 was omitted from the array
Adding the New Item

Need to
• Find appropriate index for the new item
• Shift existing items out of the way
• Insert new item
• Update the count

Shifting Array Entries

raceCount = 10

race =

Bib = 81,
Time = 20:16

Bib = 170,
Time = 22:34

Bib = 41,
Time = 23:10

Bib = 83,
Time = 23:29

Bib = 20,
Time = 23:54

Bib = 71,
Time = 21:32

Bib = 32,
Time = 23:06

Bib = 73,
Time = 23:16

Bib = 189,
Time = 23:53

Bib = 9,
Time = 23:56
Shifting Racer Entries

To make room for runner 200 at index 4

```java
racer[10] = racer[9];
racer[9] = racer[8];
racer[8] = racer[7];
racer[7] = racer[6];
racer[6] = racer[5];
racer[5] = racer[4];
```

Note that each line is of the form

```java
racer[position] = racer[position-1]
```

Loop to Shift Array Entries

```java
for ( int position = racerCount; position > insertionPos; position-- ) {
    racer[position] = racer[position-1];
}
```

*Why does the loop go backward?*
Putting It Together

To insert at a specific index

```java
public void addRacerAtPosition( int bib, String time,
                                int insertionPos ) {
    if ( racerCount < racer.length && insertionPos <= racerCount) {
        for ( int position = racerCount;
             position > insertionPos;
             position-- ) {    
            racer[position] = racer[position-1];
        }
        racer[insertionPos] = new RacerInfo( bib, time );
        racerCount++;
    }
}
```

Removing from an Array

```
Need to shift again!
```
To shift entries 7, 8, 9 left (and delete 6)

racer[6] = racer[7];
racer[7] = racer[8];
racer[8] = racer[9];
racer[9] = null;

Putting it all Together

public void removeRacerAtPosition( int position ) {
    if ( position < racerCount ) {
        racerCount--;  
        for ( int place = position; place < racerCount; place++ ) {
            racer[place] = racer[place + 1];
        }
        racer[racerCount] = null
    }
}
Arrays of Arrays

An array can represent a collection of any type of object - including other arrays!

The world is filled with examples

- Monthly magazine: we number
  - the monthly editions
  - pages with in each

- Calendars: we number
  - the months
  - days in each month

General Two-Dimensional Arrays

Say we want to develop an annual calendar manager

Representing the Data

- A month is an array of strings that represent daily events
- A year is a 12- element array of months.
Declaring an Array of Arrays

- A month is an array of daily event Strings
- A year is an array of months
So a year is an array of String arrays

private String[ ][ ] dailyEvent;

Creating an Array of Arrays

Array declaration introduces a name, *but does not create an array*

Proceed in two steps
1. Construct 12- element year
2. Construct each individual month array
1. Construct 12- element year

```
11/16/2012
```

```
dailyEvent = new String[12] []
```

2. Construct months

```
for (int month = 0; month < 12; month++) {
    int numDays = getDays( month+1 );
    dailyEvent[month] = new String[numDays];
}
```

Assume `getDays` is a private method that returns the number of days in a month.
Indexing an Array of Arrays

Say a user enters the information

1/28- Spring semester starts

The month is 1
The day is 28
The event is “Spring semester starts”
Since array indexing begins at 0,

dailyEvent[0][27] = “Spring semester starts”;
Setting and Getting Array Values

// Set the event description for a given month and day
public void setEvent( int month, int day, String description ) {
    dailyEvent[month-1][day-1] = description;
}

// Returns the event associated with a given date
public String getEvent( int month, int day ) {
    return dailyEvent[month-1][day-1];
}
Arrays of Arrays are two dimensional

When you think of an array of arrays in this way, it is natural to think of
indices as specifying row and column
someArray[rowNum][colNum]

Traversing a 2-D Array

Often want to do something with every element in an array—Use for loops!

- Ex. Initialize all calendar entries to "No event today"
  - to initialize all calendar entries for a single month:
    for (int day = 0; day < dailyEvent[month].length; day++) {
      dailyEvent[month][day] = "No event today";
    }
  - to initialize all 12 months
    // Fill all entries in each month with "No event today"
    for (int month = 0; month < 12; month++) {
      // Fill all entries for one month with "No event today"
      ...
    }
Putting it all Together

// Fill all entries in each month with "No event today"
for (int month = 0; month < 12; month++) {
    // Fill all entries for one month with "No event today"
    for (int day = 0; day < dailyEvent[month].length; day++) {
        dailyEvent[month][day] = "No event today";
    }
}

General Structure of Nested for Loops for 2-D Arrays

for (row = 0; row < myArray.length; row++) {
    for (col = 0; col < myArray[row].length; col++) {
        // Do something with array element myArray[row][col]
        ...
    }
}
Matrices

- two dimensional arrays with rows of same length
  Ex. magnified region of pixels from an image

Each pixel can be described by row and column position, as well as color value

More examples
- chessboards
- sliding block puzzles
Magic Square

• a matrix in which the sums of rows, columns, and diagonals are all equal

\[
\begin{array}{ccc}
4 & 9 & 2 \\
3 & 5 & 7 \\
8 & 1 & 6 \\
\end{array}
\]  
\[
\begin{array}{ccc}
11 & 18 & 25 & 2 & 9 \\
10 & 12 & 19 & 21 & 3 \\
4 & 6 & 13 & 20 & 22 \\
23 & 5 & 7 & 14 & 16 \\
17 & 24 & 1 & 8 & 15 \\
\end{array}
\]

Declaring and Constructing a Matrix

• Matrices are simply 2-D arrays, so a matrix is declared in the same way

  private int[ ][ ] magicSquare;

• Matrix must be constructed before it is filled

  magicSquare = new int[SIZE][SIZE];

• \(n\)-row, \(m\)-column matrix constructed as follows

  rectangularArray = new type[n][m];
Traversing a Matrix

Ex. Determine whether a square matrix is a magic square
• Row, column, and diagonal sums must be equal.
• Start by finding target sum

// Compute sum of elements in row 0
int targetSum = 0;
for (int col = 0; col < SIZE; col++) {
    targetSum = targetSum + magicSquare[0][col];
}

Row by Row Traversal

• check sum of each row
• use nested for loops!

// Assume we have a magic square unless a sum is incorrect
boolean isMagicSquare = true;
for (int row = 1; row < SIZE; row++) {
    // Check sum of each row
    int sum = 0;
    for (col = 0; col < SIZE; col++) {
        sum = sum + magicSquare[row][col];
    }
    if (sum != targetSum) {
        isMagicSquare = false;
    }
}
A More Efficient Version

• If any row’s sum does not match target, can stop right away

    // Assume we have a magic square unless a sum is incorrect
    boolean isMagicSquare = true;
    for (int row = 1; row < SIZE && isMagicSquare; row++) {
        // Check sum of each row
        int sum = 0;
        for (col = 0; col < SIZE; col++) {
            sum = sum + magicSquare[row][col];
        }
        if (sum != targetSum) {
            isMagicSquare = false;
        }
    }

Column by Column

• nested loops again
• reverse order of nesting
  – outer loop through columns
  – inner loop through rows

    // Assume we have a magic square unless a sum is incorrect
    boolean isMagicSquare = true;
    for (int col = 0; col < SIZE && isMagicSquare; col++) {
        // Check sum of each column
        int sum = 0;
        for (row = 0; row < SIZE; row++) {
            sum = sum + magicSquare[row][col];
        }
        isMagicSquare = (sum == targetSum);
    }
Diagonal Traversal

- two diagonals - two loops
- no nested loops this time

```java
//Check sum of major diagonal
int sum = 0;
for (int element = 0; element < SIZE; element++) {
    sum = sum + magicSquare[element][element];
}
isMagicSquare = (sum == targetSum);
```

Minor Diagonal

- a bit more tricky to get indices right
- for a 4x4 matrix:
  - [0][3], [1][2], [2][1], [3][0]
  - if loop var is row (over 0,1,2,3), associated column is (SIZE-1)-row

```java
// Check sum of minor diagonal
int sum = 0;
for (int row = 0; row < SIZE; row++) {
    sum = sum + magicSquare[row][SIZE-1-row];
}
isMagicSquare = (sum == targetSum);
```
Student To Do’s

• HW09: Four problems:
  – 7.11.1, 7.11.2, 7.11.4, 7.11.5.
  – All but one of them you should do twice: once with a while loop, and once with a for loop.
  – One of them cannot easily be done with a for loop, you have to figure out which one.
  – In total you should have 7 programs!

• Practice examples on your own!

• Read Java: An Eventful Approach
  – Ch. 14 and 15 (Today)