Arrays

7.1 Introduction

- Arrays
  - Data structures
  - Related data items of same type
  - Remain same size once created
    - Fixed-length entries

7.2 Arrays

- Array
  - Group of variables
    - Have same type
  - Reference type
### 7.2 Arrays (Cont.)

- **Index**
  - Also called subscript
  - Position number in square brackets
  - Must be positive integer or integer expression
  - First element has index zero

```java
da = 5;
db = 6;
c[a + b] += 2;
```

- Adds 2 to `c[11]`

### 7.3 Declaring and Creating Arrays

- **Declaring and Creating arrays**
  - Arrays are objects that occupy memory
    - `c.length`
  - Created dynamically with keyword `new`
    
    ```java
    int c[] = new int[5];
    ```
    - Equivalent to
      ```java
      int c[]; // declare array variable
      // reference
      c = new int[5]; // create array
      // array of int primitive type
    ```
  - We can create arrays of objects too
    ```java
    String b[] = new String[100];
    // array of object
    ```
### Stack
- RAM (random-access memory)
- Faster than Heap
- Memory allocation is in compiled time
  - Memory size, lifetime
  - Primitive types: char, int, boolean ...
  - Reference

### Heap
- RAM
- More flexible than Stack, slower than stack
- Memory allocation can be determined in runtime
  - object

### 7.4 Examples Using Arrays

- Declaring arrays
- Creating arrays
- Initializing arrays
- Manipulating array elements
7.4 Examples Using Arrays

• Creating and initializing an array
  – Declare array
  – Create array
  – Initialize array elements

• Using an array initializer
  – Use initializer list
    • Items enclosed in braces {}
    • Items in list separated by commas
      int n[] = { 10, 20, 30, 40, 50 };  // Creates a five-element array
      int values of 0, 1, 2, 3, 4
    – Do not need keyword new

String[] names = {"caterpillar", "momor", "beckyday", "bush"};

names[0] => "caterpillar" 物件
names[1] => "momor" 物件
names[2] => "beckyday" 物件
names[3] => "bush" 物件


"caterpillar" "momor" "beckyday" "bush"
7.4 Examples Using Arrays (Cont.)

- Calculating a value to store in each array element
  - Initialize elements of 10-element array to even integers
7.4 Examples Using Arrays (Cont.)

- Summing the elements of an array
  - Array elements can represent a series of values
    - We can sum these values

- Using bar charts to display array data graphically
  - Present data in graphical manner
    - E.g., bar chart
  - Examine the distribution of grades
7.4 Examples Using Arrays (Cont.)

- Using the elements of an array as counters
  - Use a series of counter variables to summarize data

Outline

BarChart.java

Outline

RollDie.java

Program output

Grade distribution:

00-09: 1
10-19: 2
20-29: 3
30-39: 4
40-49: 5
50-59: 6
60-69: 7
70-79: 8
80-89: 9
90-99: 10
100: 

Face Frequency

1 988
2 963
3 1018
4 1041
5 978
6 1012

Outline

RollDie.java

Program output

Face Frequency

1 988
2 963
3 1018
4 1041
5 978
6 1012
7.4 Examples Using Arrays (Cont.)

- Using arrays to analyze survey results
  - 40 students rate the quality of food
  - 1–10 Rating scale: 1 mean awful, 10 means excellent
  - Place 40 responses in array of integers
  - Summarize results

<table>
<thead>
<tr>
<th>Rating</th>
<th>Frequency</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
<tr>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>11</td>
</tr>
<tr>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>8</td>
<td>7</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
</tr>
</tbody>
</table>

7.5 Case Study: Card Shuffling and Dealing Simulation

- Program simulates card shuffling and dealing
  - Use random number generation
  - Use an array of reference type elements to represent cards
  - Three classes
    - Card
      - Represent a playing card
    - DeckOfCards
      - Represent a deck of 52 playing cards
    - DeckOfCardsTest
      - Demonstrate card shuffling and dealing
public class Card
{
    private String face; // face of card ("Ace", "Deuce", ...) 
    private String suit; // suit of card ("Hearts", "Diamonds", ...) 
    // two-argument constructor initializes card's face and suit 
    public Card(String cardFace, String cardSuit)
    {
        face = cardFace; // initialize face of card 
        suit = cardSuit; // initialize suit of card 
    } // end two-argument Card constructor 
    // constructor fills deck of Cards 
    public DeckOfCards()
    {
        String faces[] = {"Ace", "Deuce", "Three", "Four", "Five", "Six", 
                          "Seven", "Eight", "Nine", "Ten", "Jack", "Queen", "King"}; 
        String suits[] = {"Hearts", "Diamonds", "Clubs", "Spades"}; 
        deck = new Card[NUMBER_OF_CARDS]; // create array of Card objects 
        currentCard = 0; // set currentCard so first Card dealt is deck[0] 
        randomNumbers = new Random(); // create random number generator 
        // populate deck with Card objects 
        for (int count = 0; count < deck.length; count++)
            deck[count] = new Card(faces[count % 13], suits[count / 13]); 
    } // end DeckOfCards constructor 
    // after shuffling, dealing should start at deck[0] again 
    currentCard = 0; // reinitialize currentCard 
    // for each Card, pick another random Card and swap them 
    for (int first = 0; first < deck.length; first++)
    {
        // select a random number between 0 and 51 
        int second = randomNumbers.nextInt(NUMBER_OF_CARDS); 
        // swap current Card with randomly selected Card 
        Card temp = deck[first]; 
        deck[first] = deck[second]; 
        deck[second] = temp; 
    } // end for 
    // deal one Card 
    public Card dealCard()
    {
        // determine whether Cards remain to be dealt 
        if (currentCard < deck.length)
            return deck[currentCard++]; // return current Card in array 
        else 
            return null; // return null to indicate that all Cards were dealt 
    } // end method dealCard 
    // return String representation of Card 
    public String toString()
    {
        String faceOfCard = face; 
        String suitOfCard = suit; 
        return faceOfCard + " of " + suitOfCard; 
    } // end method toString 
} // end class Card
7.6 Enhanced for Statement

- **Enhanced for statement**
  - New feature of J2SE 5.0
  - Allows iterates through elements of an array or a collection without using a counter
  - Syntax
    
    ```
    for ( parameter : arrayName )
    statement
    ```

- Usage
  - Can access array elements
  - Cannot modify array elements
  - Cannot access the counter indicating the index
7.7 Passing Arrays to Methods

- To pass array argument to a method
  - Specify array name without brackets
    - Array `hourlyTemperatures` is declared as
      ```java
      int hourlyTemperatures = new int[24];
      ```
  - The method call
    ```java
    modifyArray(hourlyTemperatures);
    ```
  - Passes array `hourlyTemperatures` to method `modifyArray`

Notes on passing arguments to methods
- Two ways to pass arguments to methods
  - Pass-by-value
    - Copy of argument’s value is passed to called method
    - In Java, every primitive is pass-by-value
  - Pass-by-reference
    - Caller gives called method direct access to caller’s data
    - Called method can manipulate this data
    - Improved performance over pass-by-value
    - In Java, every object is pass-by-reference
    - In Java, arrays are objects
    - Therefore, arrays are passed to methods by reference

Outline

PassArray.java

```java
// Fig. 7.13: PassArray.java
// Passing arrays and individual array elements to methods.
public class PassArray
{
    // main creates array and calls modifyArray and modifyElement
    public static void main(String args[])
    {
        int array[] = {1, 2, 3, 4, 5};

        System.out.println(
            "Effects of passing reference to entire array:
            The values of the original array are:");

        // output original array elements
        for (int value : array)
            System.out.printf("   %d", value);

        modifyArray(array); // pass array reference

        System.out.println(
            "The values of the modified array are:");

        // output modified array elements
        for (int value : array)
            System.out.printf("   %d", value);

        System.out.printf(
            "Effects of passing array element value:
            array[3] before modifyElement: %d
            Value of element in modifyElement: %d
            array[3] after modifyElement: %d",
            array[3], array[3], array[3]);
    }
}
```

```java
public static void modifyArray(int array2[])
{
    for (int counter = 0; counter < array2.length; counter++)
        array2[counter] *= 2;
}
```

```java
public static void modifyElement(int element)
{
    element *= 2;

    System.out.printf("Value of element in modifyElement: %d", element);
}
```

```
```

Pass array element `array[3]` to method `modifyElement`
Method `modifyArray` manipulates the array directly
Method `modifyElement` manipulates a primitive’s copy
Program output
7.8 Case Study: Class GradeBook Using an Array to Store Grades

- Further evolve class GradeBook
- Class GradeBook
  - Represent a grade book that stores and analyzes grades
  - Does not maintain individual grade values
  - Repeat calculations require reentering the same grades
    - Can be solved by storing grades in an array

```java
public class GradeBook
{
  private String courseName; // name of course this GradeBook represents
  private int grades[]; // array of student grades

  // two-argument constructor initializes courseName and grades array
  public GradeBook( String name, int gradesArray[] )
  {
    courseName = name; // initialize courseName
    grades = gradesArray; // store grades
  } // end two-argument GradeBook constructor

  // method to set the course name
  public void setCourseName( String name )
  {
    courseName = name; // store the course name
  } // end method setCourseName

  // method to retrieve the course name
  public String getCourseName()
  {
    return courseName;
  } // end method getCourseName

  // display a welcome message to the GradeBook user
  public void displayMessage()
  {
    System.out.printf( "Welcome to the grade book for
%s!

", getCourseName() );
  } // end method displayMessage

  // perform various operations on the data
  public void processGrades()
  {
    // output grades array
    outputGrades();

    // call method getAverage to calculate the average grade
    System.out.printf( "Class average is %.2f
", getAverage() );

    // call methods getMinimum and getMaximum
    System.out.printf( "Lowest grade is %d
Highest grade is %d

", getMinimum(), getMaximum() );

    // call outputBarChart to print grade distribution chart
    outputBarChart();
  } // end method processGrades

  // find minimum grade
  public int getMinimum()
  { // loop through grades array
    int lowGrade = grades[ 0 ]; // assume grades[ 0 ] is smallest
    for ( int grade : grades )
    {
      if ( grade < lowGrade )
        lowGrade = grade; // new lowest grade
    } // end for
    return lowGrade; // return lowest grade
  } // end method getMinimum

  // find maximum grade
  public int getMaximum()
  { // loop through grades array
    int highGrade = grades[ 0 ]; // assume grades[ 0 ] is largest
    for ( int grade : grades )
    {
      if ( grade > highGrade )
        highGrade = grade; // new highest grade
    } // end for
    return highGrade; // return highest grade
  } // end method getMaximum
```

Outline

GradeBook.java
(1 of 5) Lines 1-3

Declare array grades to store individual grades

Line 7
Assign the array’s reference to instance variable grades

Outline

GradeBook.java
(2 of 5) Lines 5-6

Loop through grades to find the lowest grade

Outline

GradeBook.java
(3 of 5) Lines 59-64

Loop through grades to find the highest grade

Outline

GradeBook.java
(4 of 5) Lines 75-80
public double getAverage()
{
    int total = 0; // initialize total

    // sum grades for one student
    for ( int grade : grades )
        total += grade;

    // return average of grades
    return (double) total / grades.length;
} // end method getAverage

public void outputBarChart()
{
    System.out.println( "Grade distribution:" );

    // stores frequency of grades in each range of 10 grades
    int frequency[] = new int[ 11 ];

    // for each grade, increment the appropriate frequency
    for ( int grade : grades )
        ++frequency[ grade / 10 ];

    // for each grade frequency, print bar in chart
    for ( int count = 0; count < frequency.length; count++ )
    {
        // output bar label ( "00-09: ", ..., "90-99: ", "100: " )
        if ( count == 10 )
            System.out.printf( "%5d: ", 100 );
        else
            System.out.printf( "%02d-%02d: ",
                               count * 10, count * 10 + 9 );

        // print bar of asterisks
        for ( int stars = 0; stars < frequency[ count ]; stars++ )
            System.out.print( "*" );

        System.out.println(); // start a new line of output
    } // end outer for
} // end method outputBarChart

public void outputGrades()
{
    System.out.println( "The grades are:
" );

    // output each student's grade
    for ( int student = 0; student < grades.length; student++ )
        System.out.printf( "Student %2d: %3d
", student + 1, grades[ student ] );
} // end method outputGrades

// ends class GradeBook
7.9 Multidimensional Arrays

- Multidimensional arrays
  - Tables with rows and columns
    - Two-dimensional array
    - m-by-n array

- Arrays of one-dimensional array
  - Declaring two-dimensional array `b[2][2]`
    ```
    int b[][] = { { 1, 2 }, { 3, 4 } };
    ```
  - 1 and 2 initialize `b[0][0]` and `b[0][1]`
  - 3 and 4 initialize `b[1][0]` and `b[1][1]`

- Two-dimensional arrays with rows of different lengths
  - Lengths of rows in array are not required to be the same
    - E.g., `int b[][] = { { 1, 2 }, { 3, 4, 5 } };`
7.9 Multidimensional Arrays (Cont.)

- Creating two-dimensional arrays with array-creation expressions
  - Can be created dynamically
    - 3-by-4 array
      ```java
      int b[][];
      b = new int[3][4];
      ```
    - Rows can have different number of columns
      ```java
      int b[][];
      b = new int[2][]; // create 2 rows
      b[0] = new int[5]; // create 5 columns for row 0
      b[1] = new int[3]; // create 3 columns for row 1
      ```

二維陣列宣告

- `type[][]` variable;
  - `int[][]` intArray;
- `type` variable[][];
  - `int` intArray[][];
- `type[]` variable[];
  - `int[]` intArray[];
取得陣列長度

一維陣列
- 陣列名稱.length
- int[] intArray = {3, 5, 2, 6, 4};
  intArray.length \to 5

二維陣列
- 無法取得整個二維陣列長度；只能取得各列長度。
- 陣列名稱[index].length
- int[][] intArray = {{1}, {2, 3, 5}, {4, 8}};
  intArray[0].length \to 1
  intArray[1].length \to 3
  intArray[2].length \to 2

```
// Fig. 7.17: InitArray.java
// Initializing two-dimensional arrays.

public class InitArray
{
  // create and output two-dimensional arrays
  public static void main( String args[] )
  {
    int array1[][] = {{ 1, 2, 3 }, { 4, 5, 6 }};
    int array2[][] = {{ 1, 2 }, { 3 }, { 4, 5, 6 }};

    System.out.println( "Values in array1 by row are" );
    outputArray( array1 ); // displays array1 by row

    System.out.println( "Values in array2 by row are" );
    outputArray( array2 ); // displays array2 by row
  } // end main

  // output rows and columns of a two-dimensional array
  public static void outputArray( int array[][] )
  {
    // loop through array's rows
    for ( int row = 0; row < array.length; row++ )
    {
      // loop through columns of current row
      for ( int column = 0; column < array[ row ].length; column++ )
      {
        System.out.printf( "%d  ", array[ row ][ column ] );
      }  
      System.out.println(); // start new line of output
    }  // end outer for
  } // end method outputArray
} // end class InitArray
```

7.9 Multidimensional Arrays (Cont.)

- Common multidimensional-array manipulations performed with for statements
  - Many common array manipulations use for statements
  E.g.,
  ```java
  for ( int column = 0; column < a[ 2 ].length; column++ )
    a[ 2 ][ column ] = 0;
  ```

```
Use nested array initializers to initialize array1
Use nested array initializers of different lengths to initialize array2
array[row].length returns number of columns associated with row subscript
Use double-bracket notation to access two-dimensional array values
Program output
```

Values in array1 by row are
1 2 3
4 5 6
Values in array2 by row are
1 2
3
4 5 6
7.10 Case Study: Class GradeBook Using a Two-Dimensional Array

- **Class GradeBook**
  - One-dimensional array
    - Store student grades on a single exam
  - Two-dimensional array
    - Store grades for a single student and for the class as a whole

---

```java
// Fig. 7.18: GradeBook.java
// Grade book using a two-dimensional array to store grades.
public class GradeBook {
  private String courseName; // Name of course this grade book represents
  private int grades[]; // Two-dimensional array of student grades

  // Two-argument constructor initializes courseName and grades array
  public GradeBook( String name, int gradesArray[][]) {
    courseName = name; // Initialize courseName
    grades = gradesArray; // Store grades
  } // end two-argument GradeBook constructor

  // Method to set the course name
  public void setCourseName( String name ) {
    courseName = name; // Store the course name
  } // end method setCourseName

  // Method to retrieve the course name
  public String getCourseName() {
    return courseName; // Return courseName.
  } // end method getCourseName

  // Display a welcome message to the GradeBook user
  public void displayMessage() {
    System.out.printf( "Welcome to the grade book for
%s!\n
", getCourseName() );
  } // end method displayMessage

  // Perform various operations on the data
  public void processGrades() {
    // Output grades array
    outputGrades();

    // Call methods getMinimum and getMaximum
    System.out.printf( "\nLowest grade in the grade book is", getMinimum(),
"Highest grade in the grade book is", getMaximum() );

    // Output grade distribution chart of all grades on all tests
    outputBarChart();
  } // end method processGrades

  // Find minimum grade
  public int getMinimum() {
    // Assume first element of grades array is smallest
    int lowGrade = grades[ 0 ][ 0 ];

    // Loop through rows of grades array
    for ( int studentGrades[] : grades ) {
      // Loop through columns of current row
      for ( int grade : studentGrades ) {
        // If grade less than lowGrade, assign it to lowGrade
        if ( grade < lowGrade ) {
          lowGrade = grade;
        }
      } // end inner for
    } // end outer for
    return lowGrade; // Return lowest grade
  } // end method getMinimum

  // Find maximum grade
  public int getMaximum() {
    // Assume first element of grades array is largest
    int highGrade = grades[ 0 ][ 0 ];

    // Loop through rows of grades array
    for ( int studentGrades[] : grades ) {
      // Loop through columns of current row
      for ( int grade : studentGrades ) {
        // If grade greater than highGrade, replace highGrade
        if ( grade > highGrade ) {
          highGrade = grade;
        }
      } // end inner for
    } // end outer for
    return highGrade; // Return highest grade.
  } // end method getMaximum
}
```

---

**Outline**

- GradeBook.java (1 of 7)
  - Line 7
    - Declare two-dimensional array grades
  - GradeBook.java (2 of 7)
    - Loop through rows of grades to find the lowest grade of any student
  - GradeBook.java (3 of 7)
    - Declare two-dimensional array grades
  - GradeBook.java (4 of 7)
    - GradeBook constructor accepts a String and a two-dimensional array
  - GradeBook.java (5 of 7)
    - GradeBook constructor accepts a String and a two-dimensional array
  - GradeBook.java (6 of 7)
    - GradeBook constructor accepts a String and a two-dimensional array
  - GradeBook.java (7 of 7)
    - GradeBook constructor accepts a String and a two-dimensional array
Loop through rows of grades to find the highest grade of any student

Calculate the distribution of all student grades

Calculate a particular student’s semester average

Loop through rows of grades array

Lines 79-88

Lines 94-104

Output bar chart displaying overall grade distribution

Lines 115-119

Calculate a particular student’s semester average

Output the contents of the grades array

Lines 134-138

Output average grades

Lines 154-158

Call method getAverage to calculate student’s average grade;

Lines 169-173

Pass row of grades as the argument to getAverage

Lines 184-188
```java
public class GradeBookTest {

  public static void main( String args[] ) {
    // two-dimensional array of student grades
    int gradesArray[][] = {
      { 87, 96, 70 },
      { 68, 87, 90 },
      { 94, 100, 90 },
      { 100, 81, 82 },
      { 83, 65, 85 },
      { 78, 87, 65 },
      { 85, 75, 83 },
      { 91, 94, 100 },
      { 76, 72, 84 },
      { 87, 93, 73 }
    };

    GradeBook myGradeBook = new GradeBook(
        "CS101 Introduction to Java Programming", gradesArray );
    myGradeBook.displayMessage();
    myGradeBook.processGrades();
  }
}
```

**Program output**

```
Welcome to the grade book for CS101 Introduction to Java Programming!
The grades are:
Test 1  Test 2  Test 3  Average
Student 1      87      96      70    84.33
Student 2      68      87      90    81.67
Student 3      94     100      90  94.67
Student 4     100      81      82    87.67
Student 5      83      65      85    77.67
Student 6      78      87      65    76.67
Student 7      85      75      83    83.00
Student 8      92      94     100    95.00
Student 9      76      72      84    77.33
Student 10     87      93      73    84.33
Lowest grade in the grade book is 65
Highest grade in the grade book is 100
Overall grade distribution:
30-39: 0
40-49: 0
50-59: 0
60-69: 3
70-79: 4
80-89: 4
90-99: 2
100: 0
```
Variable-Length Argument Lists

public class VarargsTest
{
    // calculate average
    public static double average( double... numbers )
    {
        double total = 0.0; // initialize total
        for ( double d : numbers )
            total += d;
        return total / numbers.length;
    } // end method average

    public static void main( String args[] )
    {
        double d1 = 10.0;
        double d2 = 20.0;
        double d3 = 30.0;
        double d4 = 40.0;

        System.out.printf( "d1 = %.1f
d2 = %.1f
d3 = %.1f
d4 = %.1f

        Average of d1 and d2 is %.1f
        Average of d1, d2 and d3 is %.1f
        Average of d1, d2, d3 and d4 is %.1f
        " ,
          d1, d2, d3, d4 );
        System.out.printf(  "Average of d1 and d2 is %.1f
", average( d1, d2 ) );
        System.out.printf( "Average of d1, d2 and d3 is %.1f
", average( d1, d2, d3 ) );
        System.out.printf( "Average of d1, d2, d3 and d4 is %.1f
", average( d1, d2, d3, d4 ) );
    } // end main
} // end class VarargsTest

• Variable-length argument lists
  – New feature in J2SE 5.0
  – Unspecified number of arguments
  – Use ellipsis (…) in method’s parameter list
    • Can occur only once in parameter list
    • Must be placed at the end of parameter list
  – Array whose elements are all of the same type
在方法上使用不定长度引數時，記得必須宣告的參數必須設定在參數列的最後一個，
例如下面的方式是合法的：
```
public void someMethod(int arg1, int arg2, int... varargs) {
    // ....
}
```
但下面的方式是不合法的：
```
public void someMethod(int... varargs, int arg1, int arg2) {
    // ....
}
```
您也沒辦法使用兩個以上的不定長度引數，例如下面的方式是不合法的：
```
public void someMethod(int... varargs1, int... varargs2) {
    // ....
}
```

### Variable-Length Argument Lists

（Support Different Data Types）

- In Java, all class extends from java.lang.Object i.e.,
  java.lang.Object serves as the root class for all Java
class hierarchies

```
double calculateAllPrices(Object... shapes) {
    // 可附加重複次數的版本
    double total = 0; // 計算加總
    int i = 0;
    while(i < shapes.length) {
        times = 1;
        if(shapes[i] instanceof Shape) {
            s = (Shape) shapes[i++]; // 取得形狀
            if(i < shapes.length) // 還有下一個元素
                && shapes[i] instanceof Integer // 而且是整數
                times = (Integer) shapes[i++]; // 取得重複次數
        } total += times * calculatePrice(s);
    } return total;
}
```

### 7.12 Using Command-Line Arguments

- **Command-line arguments**
  - Pass arguments from the command line
    - `String args[]`
  - Appear after the class name in the `java` command
    - `java MyClass a b`
  - Number of arguments passed in from command line
    - `args.length`
  - First command-line argument
    - `args[ 0 ]`
### 7.13 (Optional) GUI and Graphics Case Study: Drawing Arcs

- **Draw rainbow**
  - Use arrays
  - Use repetition statement
  - Use `Graphics` method `fillArc`

```java
public class DrawRainbow extends JPanel {
    // Define indigo and violet
    final Color VIOLET = new Color(128, 0, 128);
    final Color INDIGO = new Color(75, 0, 130);

    // colors to use in the rainbow, starting from the innermost
    private Color colors[] = {Color.WHITE, Color.WHITE, VIOLET, INDIGO, Color.BLUE, Color.GREEN, Color.YELLOW, Color.ORANGE, Color.RED};

    // constructor
    public DrawRainbow() {
        setBackground(Color.WHITE); // set the background to white
    }

    // draws a rainbow using
    public void paintComponent(Graphics g) {
        setbackground(Color.WHITE); // set the background to white
        // end DrawRainbow constructor

        int radius = 20; // radius of an arch
    }
}
```
```java
31 // draw the rainbow near the bottom-center
32 int centerX = getWidth() / 2;
33 int centerY = getHeight() - 10;
34
35 // draws filled arcs starting with the outermost
36 for ( int counter = colors.length; counter > 0; counter-- )
37 {
38     // set the color for the current arc
39     g.setColor(colors[counter - 1]);
40
41     // fill the arc from 0 to 180 degrees
42     g.fillArc(centerX - counter * radius, centerY - counter * radius, counter * radius * 2, counter * radius * 2, 0, 180);
43 }
44 // end for
45 // end method paintComponent
46 // end class DrawRainbow
```

```
1 // Fig. 7.23: DrawRainbowTest.java
2 // Test application to display a rainbow.
3 import javax.swing.JFrame;
4
5 public class DrawRainbowTest
6 {
7     public static void main(String args[]) {
8         DrawRainbow panel = new DrawRainbow();
9         JFrame application = new JFrame();
10         application.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11         application.add(panel);
12         application.setSize(400, 250);
13         application.setVisible(true);
14     }
15 // end main
16 } // end class DrawRainbowTest
```

### 7.14 (Optional) Software Engineering Case Study: Collaboration Among Objects

- **Collaborations**
  - When objects communicate to accomplish task
    - Accomplished by invoking operations (methods)
  - One object sends a message to another object

![Fig. 7.24](image) | Drawing a spiral using `drawLine` (left) and `drawArc` (right).
7.14 (Optional) Software Engineering Case Study (Cont.)

• Identifying the collaborations in a system
  – Read requirements document to find
    • What ATM should do to authenticate a user
    • What ATM should do to perform transactions
  – For each action, decide
    • Which objects must interact
      – Sending object
      – Receiving object

<table>
<thead>
<tr>
<th>An object of class...</th>
<th>sends the message...</th>
<th>to an object of class...</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATM</td>
<td>displayMessage</td>
<td>Screen</td>
</tr>
<tr>
<td></td>
<td>getInput</td>
<td>Keypad</td>
</tr>
<tr>
<td></td>
<td>authenticateUser</td>
<td>BankDatabase</td>
</tr>
<tr>
<td></td>
<td>execute</td>
<td>BalanceInquiry</td>
</tr>
<tr>
<td></td>
<td>execute</td>
<td>withdrawal</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Deposit</td>
</tr>
<tr>
<td>BalanceInquiry</td>
<td>getAvailableBalance</td>
<td>BankDatabase</td>
</tr>
<tr>
<td></td>
<td>getTotalBalance</td>
<td>BankDatabase</td>
</tr>
<tr>
<td></td>
<td>displayMessage</td>
<td>Screen</td>
</tr>
<tr>
<td>Withdrawal</td>
<td>displayMessage</td>
<td>Screen</td>
</tr>
<tr>
<td></td>
<td>getInput</td>
<td>Keypad</td>
</tr>
<tr>
<td></td>
<td>getAvailableBalance</td>
<td>BankDatabase</td>
</tr>
<tr>
<td></td>
<td>isSufficientCashAvail</td>
<td>BankDatabase</td>
</tr>
<tr>
<td></td>
<td>debit</td>
<td>CashDispenser</td>
</tr>
<tr>
<td></td>
<td>dispenseCash</td>
<td>CashDispenser</td>
</tr>
<tr>
<td>Deposit</td>
<td>displayMessage</td>
<td>Screen</td>
</tr>
<tr>
<td></td>
<td>getInput</td>
<td>Keypad</td>
</tr>
<tr>
<td></td>
<td>isEnvelopeReceived</td>
<td>DepositsSlot</td>
</tr>
<tr>
<td></td>
<td>Credit</td>
<td>BankDatabase</td>
</tr>
<tr>
<td>BankDatabase</td>
<td>validatePIN</td>
<td>Account</td>
</tr>
<tr>
<td></td>
<td>getAvailableBalance</td>
<td>Account</td>
</tr>
<tr>
<td></td>
<td>getTotalBalance</td>
<td>Account</td>
</tr>
<tr>
<td></td>
<td>debit</td>
<td>Account</td>
</tr>
<tr>
<td></td>
<td>Credit</td>
<td>Account</td>
</tr>
</tbody>
</table>

Fig. 7.25 | Collaborations in the ATM system.

7.14 (Optional) Software Engineering Case Study (Cont.)

• Interaction Diagrams
  – Model interactions use UML
  – Communication diagrams
    • Also called collaboration diagrams
    • Emphasize which objects participate in collaborations
  – Sequence diagrams
    • Emphasize when messages are sent between objects

7.14 (Optional) Software Engineering Case Study (Cont.)

• Communication diagrams
  – Objects
    • Modeled as rectangles
    • Contain names in the form objectName:className
  – Objects are connected with solid lines
  – Messages are passed alone these lines in the direction shown by arrows
  – Name of message appears next to the arrow
7.14 (Optional) Software Engineering Case Study (Cont.)

- Sequence of messages in a communication diagram
  - Appear to the left of a message name
  - Indicate the order in which the message is passed
  - Process in numerical order from least to greatest

Fig. 7.26 | Communication diagram of the ATM executing a balance inquiry.

7.14 (Optional) Software Engineering Case Study (Cont.)

- Sequence diagrams
  - Help model the timing of collaborations
  - Lifeline
    - Dotted line extending down from an object's rectangle
      - Represents the progression of time
  - Activation
    - Thin vertical rectangle
      - Indicates that an object is executing

Fig. 7.27 | Communication diagram for executing a balance inquiry.
Fig. 7.28 | Sequence diagram that models a withdrawal executing.

Fig. 7.29 | Sequence diagram that models a deposit executing.