Chapter 7 Arrays and Vectors

- Introducing Arrays
- Declaring Arrays, Creating Arrays, and Initializing Arrays
- Array of Objects
- Copying Arrays
- Multidimensional Arrays
- Numeric Wrapper Classes
- Command-Line Parameters
- Creating Generic Classes
- The `Vector` Class

Introducing Arrays

Array is introduced for storage of large amount of data

Array is a data structure that represents a collection of the same types of data. Java treats these arrays as objects.

An array of 10 Elements of type `double` can be declare as

```
double[] myList = new double[10]

or

double myList[]= new double[10]
```

Declaring Arrays

- `datatype[] arrayname;`

  Example:

  `int[] myList;`

Alternatively

- `datatype arrayname[];`

  Example:

  `int myList[];`

Creating Arrays

```
arrayName = new datatype[arraySize];
```

Example:

```
myList = new double[10];
```

Declaring and Creating in One Step

- `datatype[] arrayname = new datatype[arraySize];`

```
double[] myList = new double[10];
```

- `datatype arrayname[] = new datatype[arraySize];`

```
double myList[] = new double[10];
```

Initializing Arrays

- Using a loop:

```
for (int i = 0; i < myList.length; i++)
    myList[i] = (double)i;
```

- Declaring, creating, initializing in one step:

```
double[] myList = {1.9, 2.9, 3.4, 3.5};
```

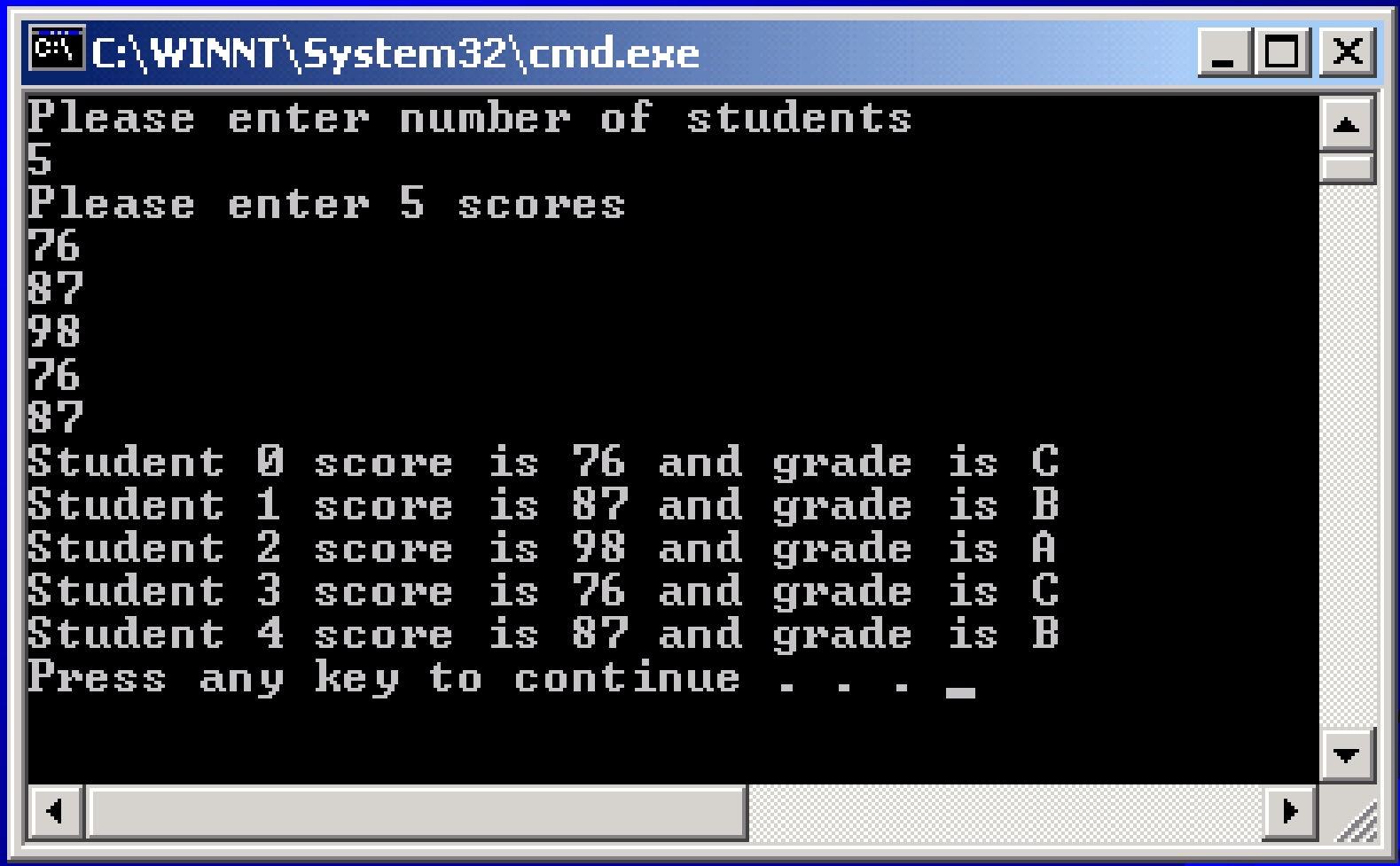
Example 7.1
Assigning Grades

- Objective: read student scores (int) from the keyboard, get the best score, and then assign grades based on the following scheme:
  - Grade is A if score is >= best–10;
  - Grade is B if score is >= best–20;
  - Grade is C if score is >= best–30;
  - Grade is D if score is >= best–40;
  - Grade is F otherwise.

```
// AssignGrade.java: Assign grade
public class AssignGrade
{
  // Main method
  public static void main(String[] args)
  {
    int numOfStudents = 0; // The number of students
    int[] scores; // Array scores
    int best = 0; // The best score
    char grade; // The grade

    // Get number of students
    System.out.println("Please enter number of students");
    numOfStudents = MyInput.readInt();
```

```
// Create array scores
   scores = new int[numOfStudents];

   // Read scores and find the best score
   System.out.println("Please enter " + numOfStudents + " scores");
   for (int i=0; i<scores.length; i++)
   {
     scores[i] = MyInput.readInt();
     if (scores[i] > best)
       best = scores[i];
   }

   // Assign and display grades
   for (int i=0; i<scores.length; i++)
   {
     if (scores[i] >= best - 10)
       grade = 'A';
```

```
else if (scores[i] >= best - 20)
        grade = 'B';
      else if (scores[i] >= best - 30)
        grade = 'C';
      else if (scores[i] >= best - 40)
        grade = 'D';
      else
        grade = 'F';

      System.out.println("Student " + i + " score is " + scores[i] +
        " and grade is " + grade);
    }
  }
}
```


C:\WINNT\System32\cmd.exe
Please enter number of students
5
Please enter 5 scores
76
87
98
76
87
Student 0 score is 76 and grade is C
Student 1 score is 87 and grade is B
Student 2 score is 98 and grade is A
Student 3 score is 76 and grade is C
Student 4 score is 87 and grade is B
Press any key to continue . . .

Example 7.2
Using Arrays in Sorting

- Sorting is a common task
- It has many sorting algorithms
- Here is a selection sorting
  - 2 9 5 4 8 1 6 original list set
  - 2 6 5 4 8 1 9
  - 2 6 5 4 1 8 9
  - 2 1 5 4 6 8 9
  - 2 1 4 5 6 8 9
  - 1 2 4 5 6 8 9

Example 7.2
Using Arrays in Sorting

- Objective: Use the `selectionSort` method to write a program that will sort a list of double floating-point numbers.

```
// SelectionSort.java: Sort numbers using selection sort
public class SelectionSort
{
  // Main method
  public static void main(String[] args)
  {
    // Initialize the list
    double[] myList = {5.0, 4.4, 1.9, 2.9, 3.4, 3.5};

    // Print the original list
    System.out.println("My list before sort is: ");
    printList(myList);

    // Sort the list
    selectionSort(myList);
```

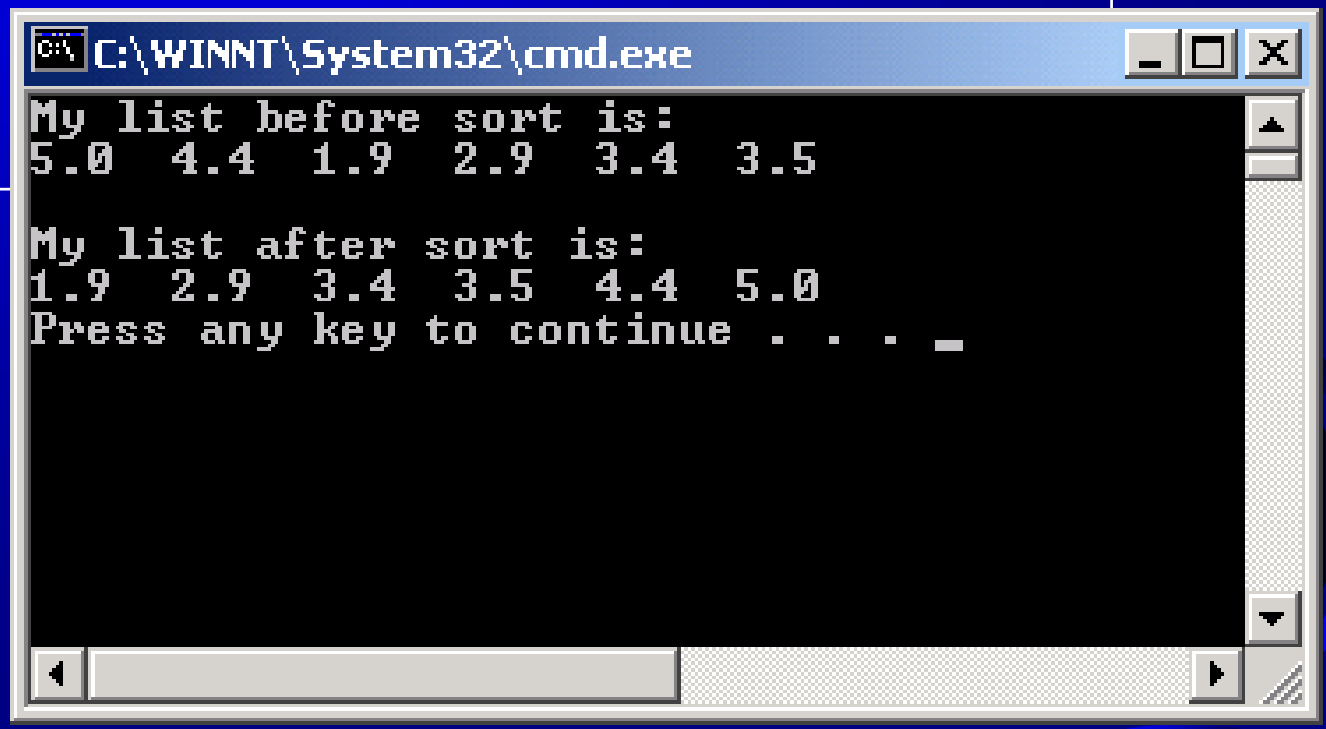
```
// Print the sorted list
  System.out.println();
  System.out.println("My list after sort is: ");
  printList(myList);
 }

 // The method for printing numbers
 static void printList(double[] list)
 {
  for (int i=0; i<list.length; i++)
    System.out.print(list[i] + "  ");
  System.out.println();
 }
```

```
// The method for sorting the numbers
 static void selectionSort(double[] list)
 {
   double currentMax;
   int currentMaxIndex;
   for (int i=list.length-1; i>=1; i--)
   {
     // Find the maximum in the list[0..i]
     currentMax = list[i];
     currentMaxIndex = i;
     for (int j=i-1; j>=0; j--)
     {
       if (currentMax < list[j])
       {
         currentMax = list[j];
         currentMaxIndex = j;
       }
     }
```

Outer loop

Inner loop

```
    // Swap list[i] with list[currentMaxIndex] if necessary;
    if (currentMaxIndex != i)
    {
      list[currentMaxIndex] = list[i];
      list[i] = currentMax;
    }
  }
}
```

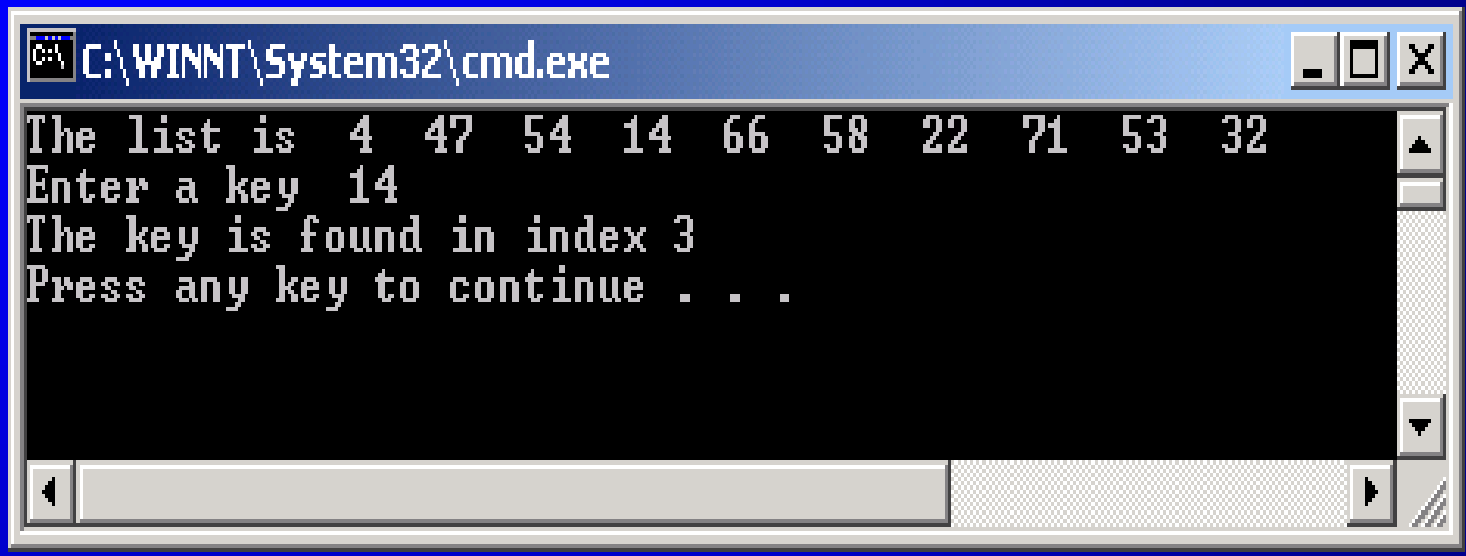


Example 7.3
Testing Linear Search

- Search one by one from the beginning to the end. If the "search keyword" or number is found, return one signal, otherwise return another signal.
- Search is the base point of quering
- Objective: Implement and test the linear search method by creating an array of 10 elements of `int` type randomly and then displaying this array. Prompt the user to enter a key for testing the linear search.

```
// LinearSearch.java: Search for a number in a list
public class LinearSearch
{
  // Main method
  public static void main(String[] args)
  {
    int[] list = new int[10];

    // Create the list randomly and display it
    System.out.print("The list is  ");
    for (int i=0; i<list.length; i++)
    {
      list[i] = (int)(Math.random()*100);
      System.out.print(list[i]+"  ");
    }
    System.out.println();
```

```
// Prompt the user to enter a key
  System.out.print("Enter a key  ");
  int key = MyInput.readInt();
  int index = linearSearch(key, list);
  if (index != -1)
    System.out.println("The key is found in index "+index);
  else
    System.out.println("The key is not found in the list");
 }
 // The method for finding a key in the list
 public static int linearSearch(int key, int[] list)
 {
  for (int i=0; i<list.length; i++)
    if (key == list[i])
      return i;
  return -1;
 }
}
```


C:\WINNT\System32\cmd.exe
The list is 4 47 54 14 66 58 22 71 53 32
Enter a key 14
The key is found in index 3
Press any key to continue . . .

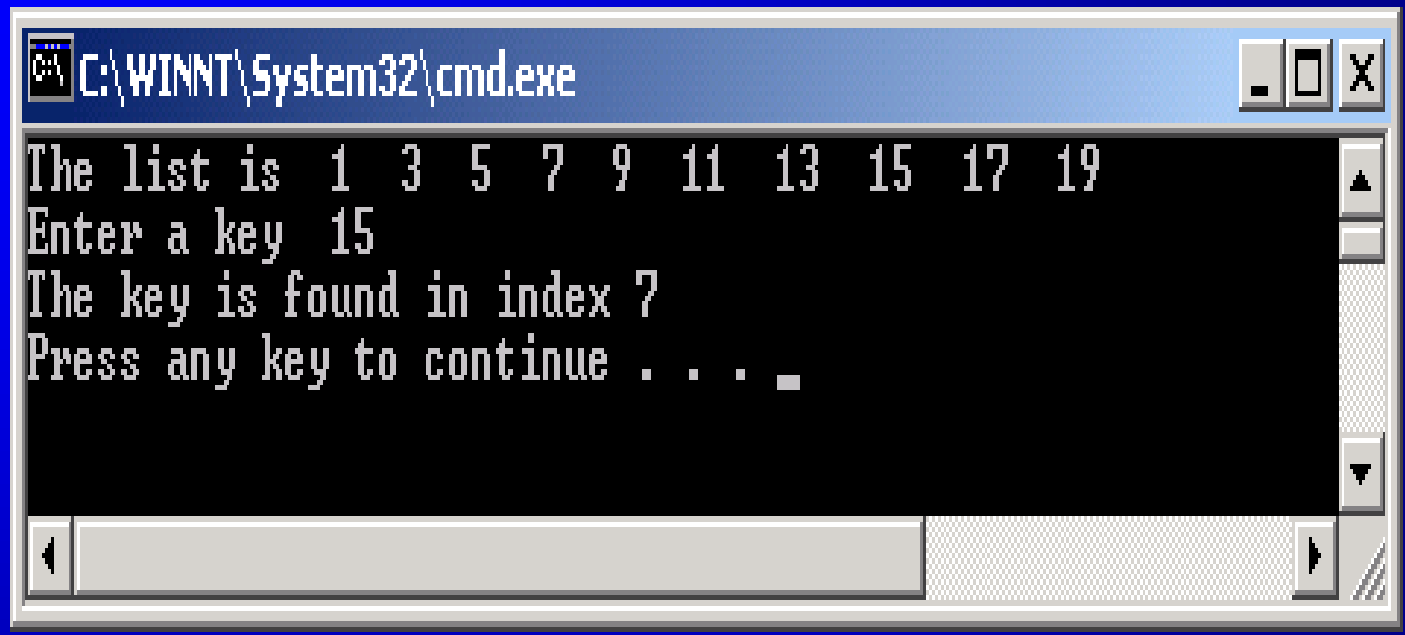
Example 7.4
Testing Binary Search

- First do sorting in either descending order or ascending order
- Compare between the keyword and the middle element of the list (array)
- Search times is less than log2(n+1), if you the search list is n; 1024 needs 11 step comparisons.
- Objective: Implement and test the binary search method by creating an array of 10 elements of `int` type in ascending order and then displaying this array. (assume the list is already sorted)
- Prompt the user to enter a key for testing the binary search.

```
// BinarySearch.java: Search a key in a sorted list
public class BinarySearch
{
  // Main method
  public static void main(String[] args)
  {
    int[] list = new int[10];

    // Create a sorted list and display it
    System.out.print("The list is  ");
    for (int i=0; i<list.length; i++)
    {
      list[i] = 2*i + 1;
      System.out.print(list[i] + "  ");
    }
    System.out.println();
```

```
// Prompt the user to enter a key
  System.out.print("Enter a key  ");
  int key = MyInput.readInt();
  int index = binarySearch(key, list);
  if (index != -1)
    System.out.println("The key is found in index " + index);
  else
    System.out.println("The key is not found in the list");
}

// Use binary search to find the key in the list
public static int binarySearch(int key, int[] list)
{
  int low = 0;
  int up = list.length - 1;
  return binarySearch(key, list, low, up);
}
```

```
// Use binary search to find the key in the list between
 // list[low] list[up]
 public static int binarySearch(int key, int[] list, int low, int up)
 {
   if (low > up)  // The list has been exhausted without a match
     return -1;

   int mid = (low + up)/2;
   if (key < list[mid])
     return binarySearch(key, list, low, mid-1);
   else if (key == list[mid])
     return mid;
   else if (key > list[mid])
     return binarySearch(key, list, mid+1, up);

   return -1;
  }
}
```


C:\WINNT\System32\cmd.exe
The list is 1 3 5 7 9 11 13 15 17 19
Enter a key 15
The key is found in index 7
Press any key to continue . . .

Array of Objects

- Declaring and creating:

```
Circle[] circleArray = new Circle[10];
```

- Initializing (creating objects):

```
for (int i=0; i<circleArray.length; i++)
{
  circleArray[i] = new Circle();
}
```

Example 7.5
Summarizing the Areas of Circles

- Objective: Summarize the areas of an array of circles with randomized radius

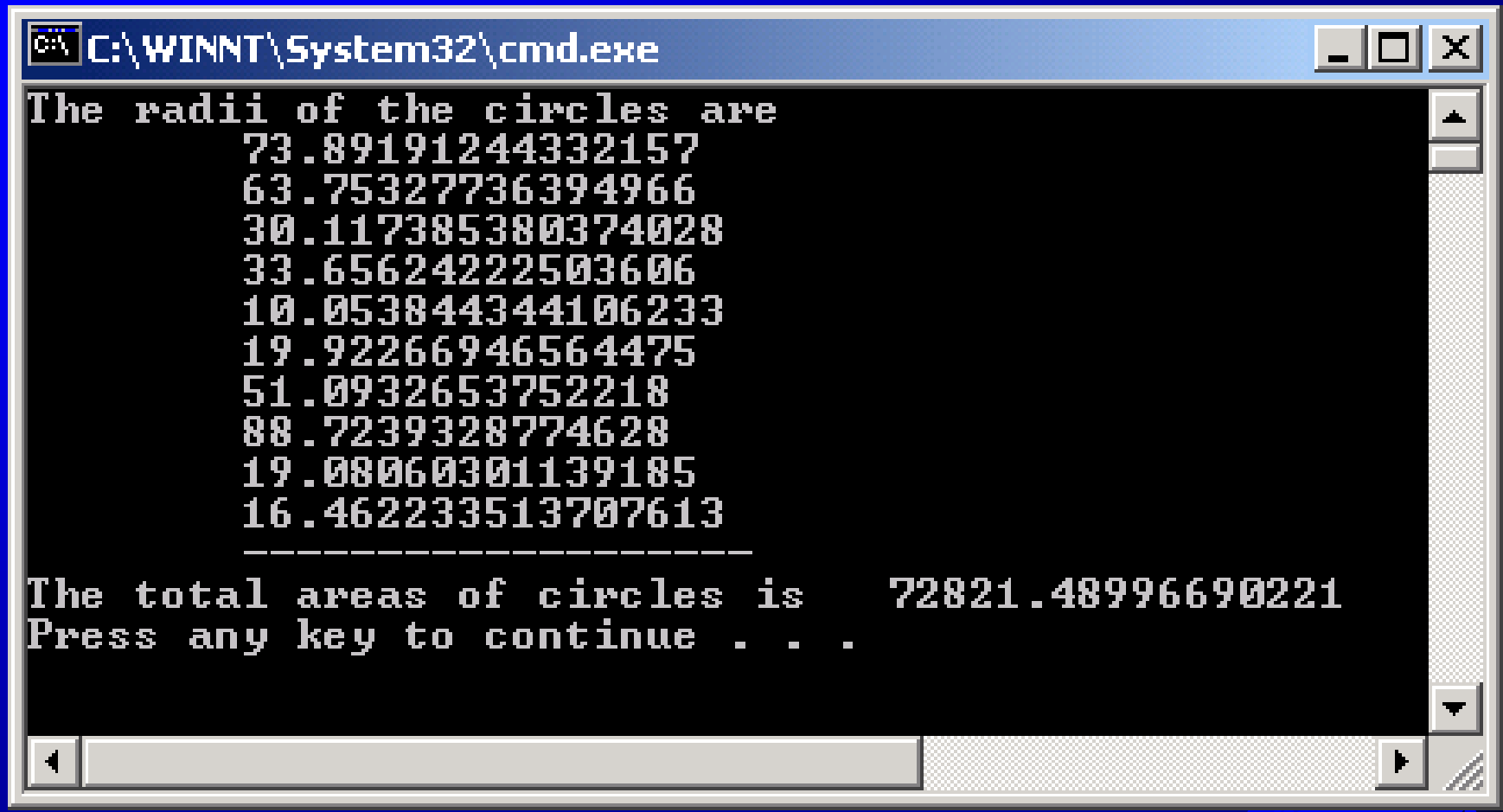
```
// TotalArea.java: Test passing an array of objects to the method
public class TotalArea
{
  // Main method
  public static void main(String[] args)
  {
    // Declare circleArray
    Circle[] circleArray=new Circle[10];
    for (int i=0; i<circleArray.length; i++)
    {
      circleArray[i] = new Circle(Math.random()*100);
    }
//    circleArray = createCircleArray();

// Print circleArray and total areas of the circles
    printCircleArray(circleArray);
  }
```

```
// TotalArea.java: Test passing an array of objects to the method
public class TotalArea
{
  // Main method
  public static void main(String[] args)
  {
    // Declare circleArray
    Circle[] circleArray=new Circle[10];
    for (int i=0; i<circleArray.length; i++)
    {
      circleArray[i] = new Circle(Math.random()*100);
    }
//    circleArray = createCircleArray();      // alternatively

// Print circleArray and total areas of the circles
    printCircleArray(circleArray);
  }
```

```
/*
  // Create an array of Circle objects
  public static Circle[] createCircleArray()
  {
    Circle[] circleArray = new Circle[10];

    for (int i=0; i<circleArray.length; i++)
    {
      circleArray[i] = new Circle(Math.random()*100);
    }

    // Return Circle array
    return circleArray;
  } */
```

```
public static void printCircleArray(Circle[] circleArray)
  {
    System.out.println("The radii of the circles are");
    for (int i=0; i<circleArray.length; i++)
    {
      System.out.print("\t" +
        circleArray[i].getRadius() + '\n');
    }

    System.out.println("\t------------------");

    // Compute and display the result
    System.out.println("The total areas of circles is \t" +
      sum(circleArray));
  }
```

```
// Add circle areas
public static double sum(Circle[] circleArray)
{
  // Initialize sum
  double sum = 0;

  // Add areas to sum
  for (int i = 0; i < circleArray.length; i++)
    sum += circleArray[i].findArea();

  return sum;
  }
}
```


C:\WINNT\System32\cmd.exe
The radii of the circles are
73.89191244332157
63.75327736394966
30.11738538037402 8
33.65624222503606
10.05384434410623 3
19.92266946564475
51.093265375221 8
88.72393287746 28
19.08060301139185
16.46223351370761 3
The total areas of circles is 72821.48996690221
Press any key to continue . . .

Copying Arrays

Individual primitive variable you can use assignment to copy value, such as

```
int a=2, b;
b=a;
```

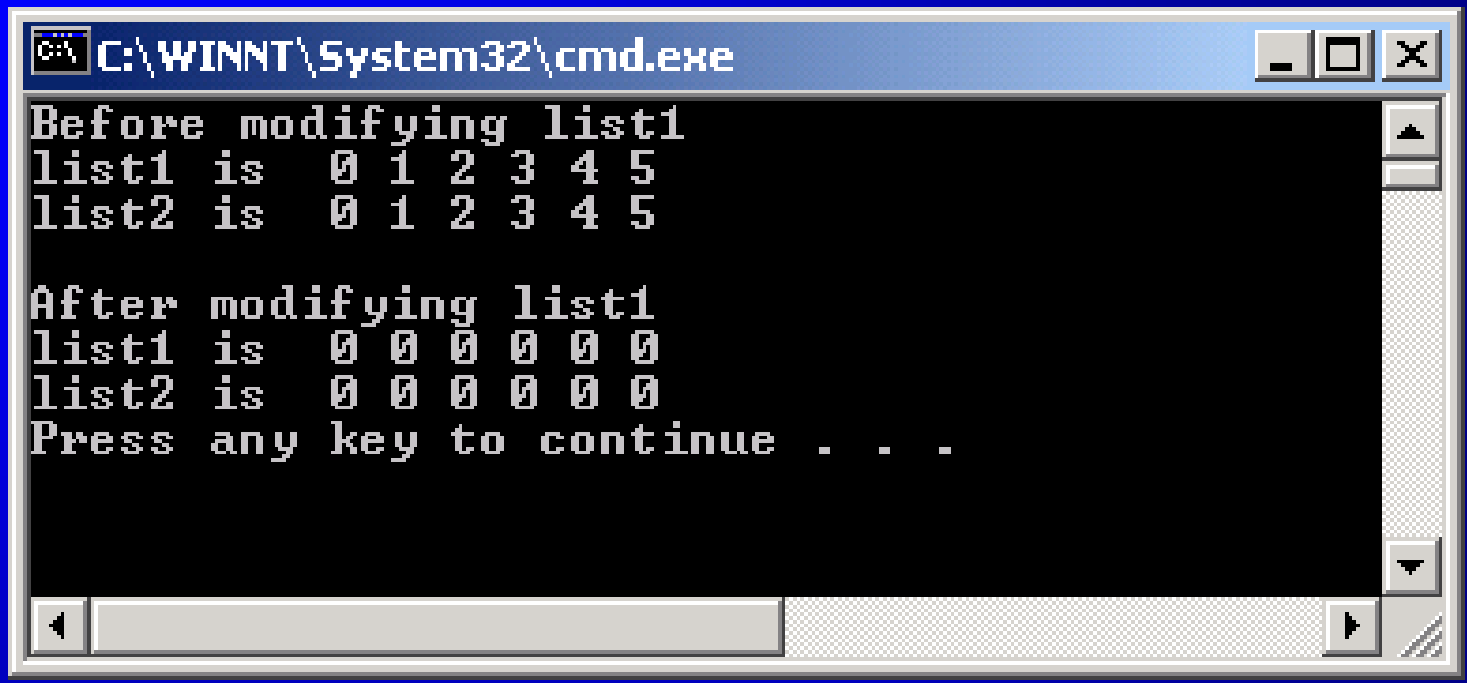
However, it does not apply to copying arrays

```
int [] array1={1,2,3,4};
Int [] array2=new int[array1.length];
Array2=array1;
```

Example 7.6 show the failure of copying arrays

```
// TestCopyArray.java: Demonstrate copying arrays
public class TestCopyArray
{
  // Main method
  public static void main(String[] args)
  {
    // Create an array and assign values
    int[] list1 = {0, 1, 2, 3, 4 ,5};
    // Create an array with default values
    int[] list2 = new int[list1.length];

    // Assign array list to array list2
    list2 = list1;
    // Display list1 and list2
    System.out.println("Before modifying list1");
    printList("list1 is ", list1);
    printList("list2 is ", list2);
```

```
    // Modify list1
    for (int i=0; i<list1.length; i++)
      list1[i] = 0;
    // Display list1 and list2 after modifying list1
    System.out.println("\nAfter modifying list1");
    printList("list1 is ", list1);
    printList("list2 is ", list2);
  }
  // The method for printing a list
  public static void printList(String s, int[] list)
  {
    System.out.print(s + " ");
    for (int i=0; i<list.length; i++)
      System.out.print(list[i] + " ");
    System.out.print('\n');
  }
}
```



Three ways to copy arrays

Use loop to copy individual elements one by one

Using a loop:

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray = new
  int[sourceArray.length];

for (int i = 0; i < sourceArrays.length; i++)
   targetArray[i] = sourceArray[i];
```

Use clone method copy, treating like object

Using a loop:

```
int[] sourceArray = {2, 3, 1, 5, 10};
int[] targetArray =
   (int[])sourceArray.clone();
```

The `arraycopy` method in java.lang.System class

```
arraycopy(sourceArray, src_pos, targetArray,
   tar_pos, length);
```

src_pos and tar_pos indicating the starting positions in sourceArray and targetArray, respectively

The number of elements copied from sourceArray to targetArray is indicated by length

Example:

```
int[] sourceArray={2,3,1,5,10};
int[] targetArray = new int[sourceArray.length];
System.arraycopy(sourceArray, 0, targetArray, 0,
   sourceArray.length);
```

Multidimensional Arrays

For table or matrix, you need a multidimensional array

```
int[][] matrix = new int[10][10];
```

or

```
int matrix[][] = new int[10][10];
```

```
for (int i=0; i<matrix.length; i++)
  for (int j=0; j<matrix[i].length; j++)
  {
    matrix[i][j] = (int)(Math.random()*1000);
  }
```

Multidimensional Arrays

Declaration and initialization are put together

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

```
matrix[2][0]=7;
```

Example 7.7
Adding and Multiplying Two Matrices

- Objective: Use two-dimensional arrays to create two matrices, and then add and multiple the two matrices.

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \\ a_{51} & a_{52} & a_{53} & a_{54} & a_{55} \end{pmatrix} + \begin{pmatrix} b_{11} & b_{12} & b_{13} & b_{14} & b_{15} \\ b_{21} & b_{22} & b_{23} & b_{24} & b_{25} \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} \\ b_{51} & b_{52} & b_{53} & b_{54} & b_{55} \end{pmatrix} = \begin{pmatrix} a_{11}+b_{11} & a_{12}+b_{12} & a_{12}+b_{13} & a_{12}+b_{14} & a_{15}+b_{15} \\ a_{21}+b_{21} & a_{22}+b_{22} & a_{23}+b_{23} & a_{24}+b_{24} & a_{25}+b_{25} \\ a_{31}+b_{31} & a_{32}+b_{32} & a_{33}+b_{33} & a_{34}+b_{34} & a_{35}+b_{35} \\ a_{41}+b_{41} & a_{42}+b_{42} & a_{43}+b_{43} & a_{44}+b_{44} & a_{45}+b_{45} \\ a_{51}+b_{51} & a_{52}+b_{52} & a_{53}+b_{53} & a_{54}+b_{54} & a_{55}+b_{55} \end{pmatrix}$$

```
// TestMatrixOperation.java: Add and multiply two matrices
public class TestMatrixOperation
{
  // Main method
  public static void main(String[] args)
  {
    // Create two matrices as two dimensional arrays
    int[][] matrix1 = new int[5][5];
    int[][] matrix2 = new int[5][5];

    // Assign random values to matrix1 and matrix2
    for (int i=0; i<matrix1.length; i++)
      for (int j=0; j<matrix1[i].length; j++)
      {
        matrix1[i][j] = (int)(Math.random()*10);
        matrix2[i][j] = (int)(Math.random()*10);
      }
```

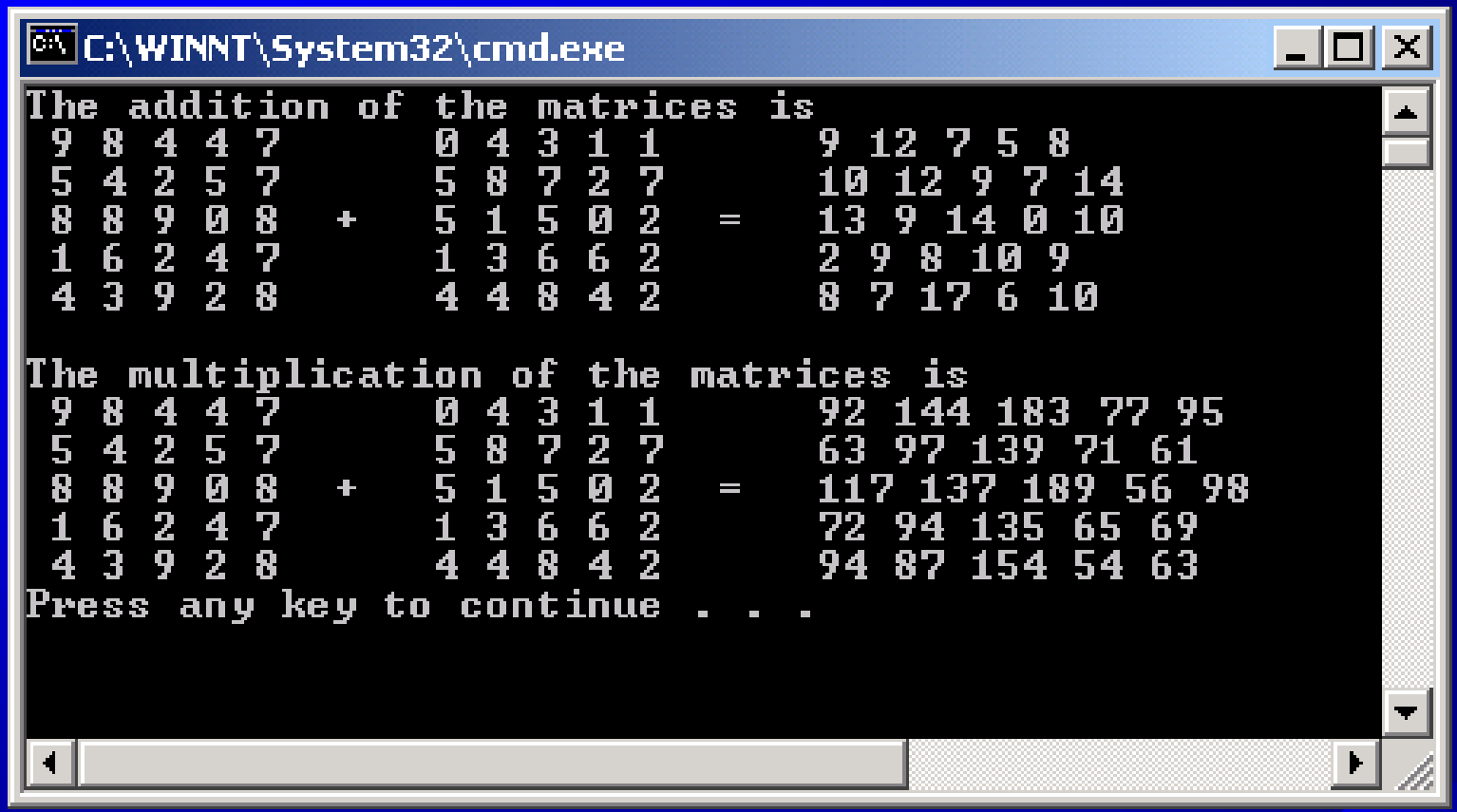
```
    // Add two matrics and print the result
    int[][] resultMatrix = new int[5][5];
    resultMatrix = addMatrix(matrix1, matrix2);
    System.out.println("The addition of the matrices is ");
    printResult(matrix1, matrix2, resultMatrix, '+');

    // Multiply two matrics and print the result
    resultMatrix = multiplyMatrix(matrix1, matrix2);
    System.out.println("\nThe multiplication of the matrices is ");
    printResult(matrix1, matrix2, resultMatrix, '+');
}
```

```
// The method for adding two matrices
  public static int[][] addMatrix(int[][] m1, int[][] m2)
  {
    int[][] result = new int[m1.length][m1[0].length];
    for (int i=0; i<m1.length; i++)
      for (int j=0; j<m1[0].length; j++)
        result[i][j] = m1[i][j] + m2[i][j];

    return result;
  }
```

```
// The method for multiplying two matrices
 public static int[][] multiplyMatrix(int[][] m1, int[][] m2)
 {
  int[][] result = new int[m1[0].length][m2.length];
  for (int i=0; i<m1.length; i++)
    for (int j=0; j<m2[0].length; j++)
      for (int k=0; k<m1[0].length; k++)
        result[i][j] += m1[i][k]*m2[k][j];

  return result;
 }
```

```
// Print result
 public static void printResult(
   int[][] m1, int[][] m2, int[][] m3, char op)
 {
   for (int i=0; i<m1.length; i++)
   {
     for (int j=0; j<m1[0].length; j++)
       System.out.print(" " + m1[i][j]);

     if (i == m1.length/2)
       System.out.print( "   " + op + "   " );
     else
       System.out.print( "       " );

     for (int j=0; j<m2[0].length; j++)
       System.out.print(" " + m2[i][j]);
```

```
if (i == m1.length/2)
      System.out.print( "  =  " );
    else
      System.out.print( "     " );

    for (int j=0; j<m3[0].length; j++)
      System.out.print(" " + m3[i][j]);

    System.out.println();
   }
  }
}
```


C:\WINNT\System32\cmd.exe
The addition of the matrices is
9 8 4 4 7 0 4 3 1 1 9 12 7 5 8
5 4 2 5 7 5 8 7 2 7 10 12 9 7 14
8 8 9 0 8 + 5 1 5 0 2 = 13 9 14 0 10
1 6 2 4 7 1 3 6 6 2 2 9 8 10 9
4 3 9 2 8 4 4 8 4 2 8 7 17 6 10
The multiplication of the matrices is
9 8 4 4 7 0 4 3 1 1 92 144 183 77 95
5 4 2 5 7 5 8 7 2 7 63 97 139 71 61
8 8 9 0 8 + 5 1 5 0 2 = 117 137 189 56 98
1 6 2 4 7 1 3 6 6 2 72 94 135 65 69
4 3 9 2 8 4 4 8 4 2 94 87 154 54 63
Press any key to continue . . .

Wrapper Classes

Primitive data types are not used as objects in Java.
Many Java methods require the use of objects as arguments

Question: how we can wrap a primitive data type into an object or how to wrap a data member of an object to a primitive data type?

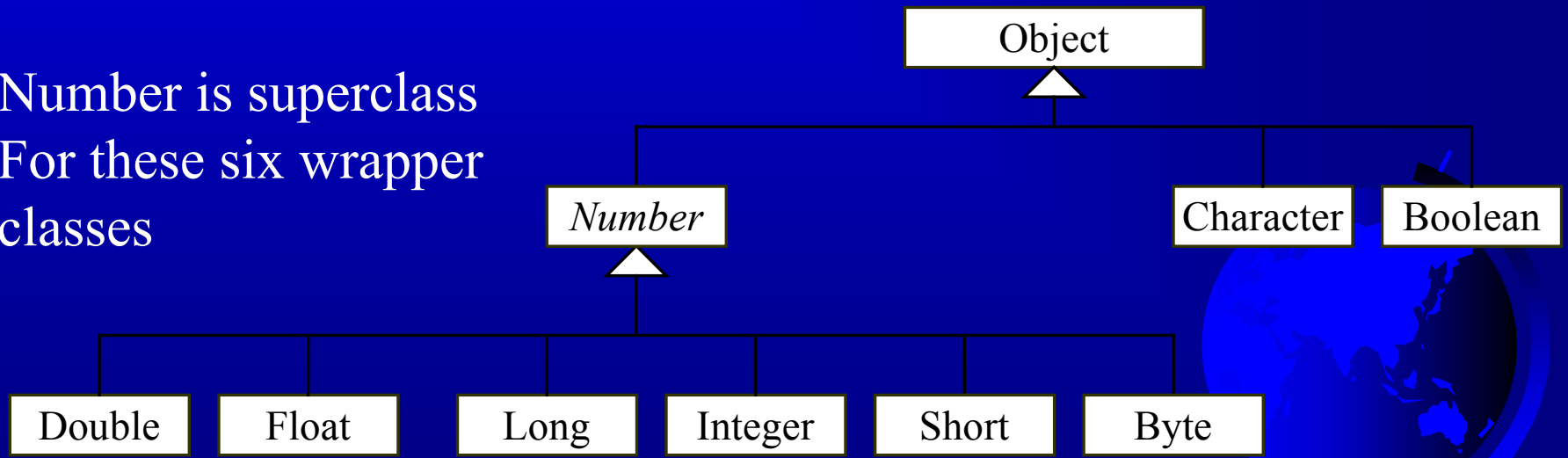
Java introduce a wrapper class which is very convenient for programmer to do conversion job.
Warp class help programmer to do generic programming

Java provides several wrapper classes

Wrapper Classes

- Boolean
- Character
- Short
- Byte
- Integer
- Long
- Float
- Double

Number is superclass
For these six wrapper classes



The `Number` Class

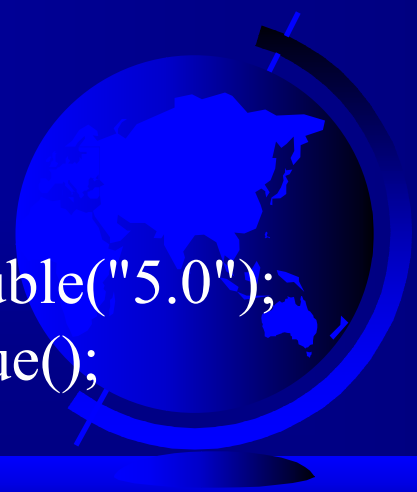
- Define abstract methods to convert the represented numerical value to byte, double, float, int, long, and short. They are implemented in each subclass
  - public byte byteValue() in Byte class
  - public short shortValue() in Short class
  - public int intValue() in Int class
  - public long longValue() in Long class
  - public float floatValue() in Float class
  - public double doubleValue() in Double class

These methods are kinds of getter accessors.

The constructors of Numeric wrap classes

- Constructors
  - Public Integer(int value)
  - Public Integer(String s)
  - Public Double(double d)
  - Public Double(String s)
- Similar in other numeric wrap classes
- Example:

```
Double doubleObject =new Double("5.0");
int a = doubleObject.doubleValue();
```

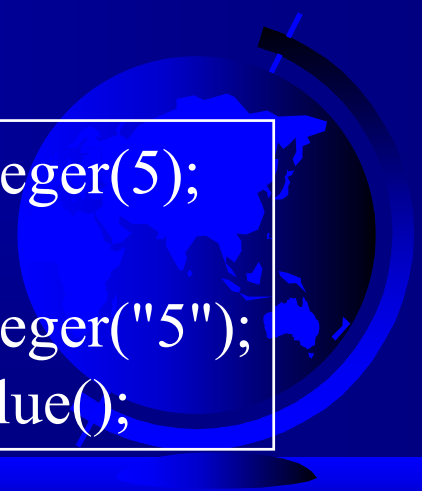


The constructors of Numeric wrap classes

- Example:

```
Double doubleObject =new Double(5.0);
Or
Double doubleObject =new Double("5.0");

double d = doubleObject.doubleValue();
```

```
Integer integerObject=new Integer(5);
Or
Integer integerObject=new Integer("5");
int i= integerObject.integerValue();
```



The `Numeric class constants`

- Each numeric wrapper class has constants: MAX_VALU and MIN_VALUE
- How to find out these values?
  - Integer.Max_VALUE
  - Double.Min_VALUE

The conversion methods between value and string

- For example: double to string

```
Double d=5.9;
Double doubleObject=new Double(d);
String s = doubleObject.toString();
```

The conversion methods between value and string

- For example: string to a double

```
String s1="12.4";
String s2="4";

Double doubleObject=Double.valueOf(s1);
double d=doubleObject.doubleValue();

Integer integerObject=Integer.valueOf(s2);
int j = integerObject.integerValue();
```

The conversion methods between value and string

- Alternatively, one can use parseInt() or parseDouble()

```
String s1="12.4";
String s2="4";

double d=Double.parseDouble(s1);

int j = Integer.parseInt(s2);
```

Command-Line Parameters

Answer what is "String[] args" in the main() method
Args String is used to pass parameters to the code

```
class TestMain
{
  public static void main(String[] args)
    { ... }
}
```


```
java  TestMain arg0 arg1 arg2 ... argn
```

Processing Command-Line Parameters

In the main method, get the arguments from `args[0]`, `args[1]`, `...`, `args[n]`, which corresponds to `arg0`, `arg1`, `...`, `argn` in the command line.

Example 7.8
Using Command-Line Parameters

- Objective: Write a program that will perform binary operations on integers. The program receives three parameters: an operator and two integers.

java Calculator + 2 3

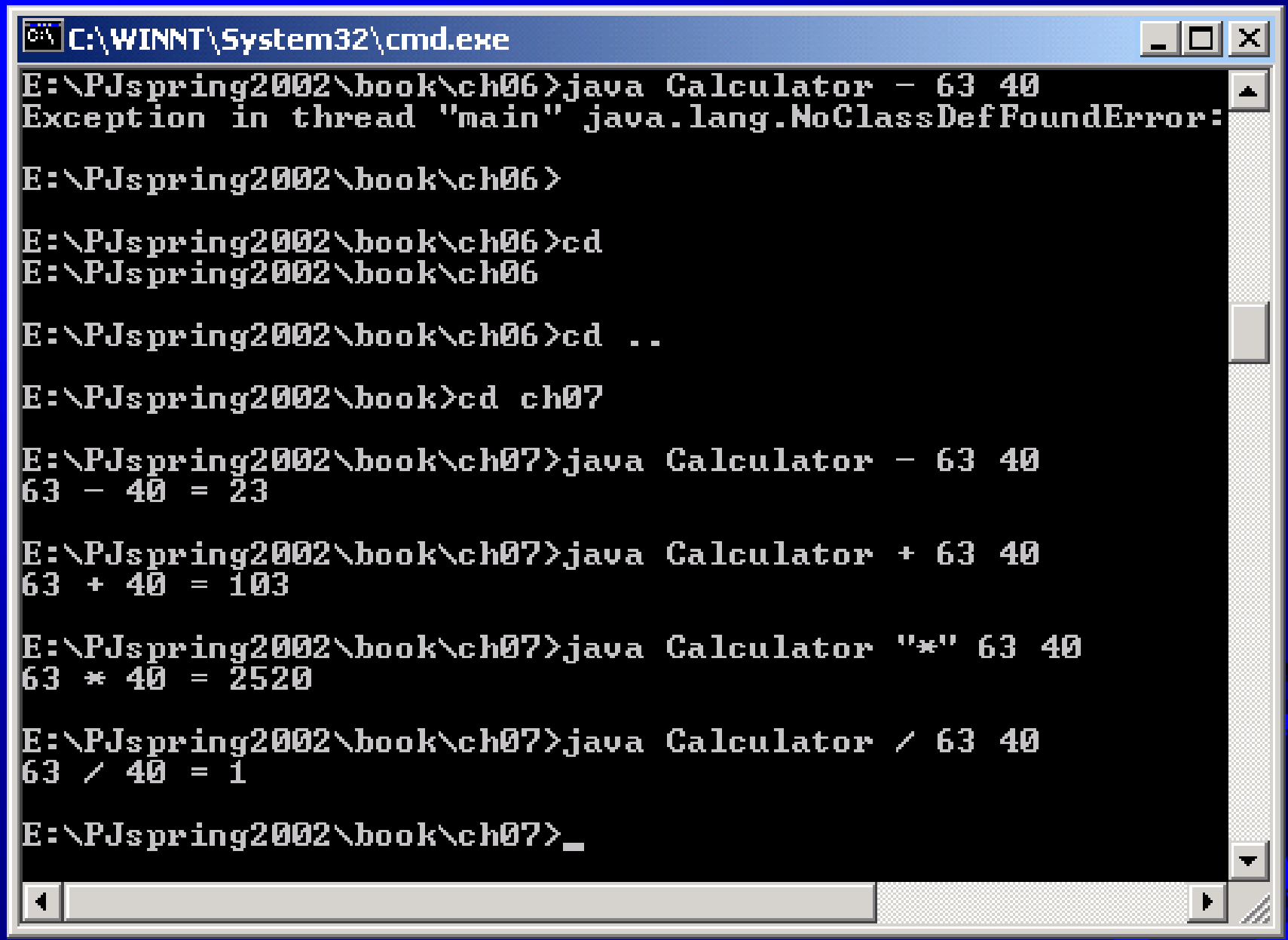
java Calculator - 2 3

java Calculator / 2 3

```
// Calculator.java: Pass parameters from the command line
public class Calculator
{
  // Main method
  public static void main(String[] args)
  {
    // The result of the operation
    int result = 0;

    if (args.length != 3)
    {
      System.out.println(
        "Usage: java Calculator operator operand1 operand2");
      System.exit(0);
    }
```

```
// Determine the operator
   switch (args[0].charAt(0))
   {
    case '+': result = Integer.parseInt(args[1]) +
                       Integer.parseInt(args[2]);
              break;
    case '-': result = Integer.parseInt(args[1]) -
                       Integer.parseInt(args[2]);
              break;
    case '*': result = Integer.parseInt(args[1]) *
                       Integer.parseInt(args[2]);
              break;
    case '/': result = Integer.parseInt(args[1]) /
                       Integer.parseInt(args[2]);
   }
```

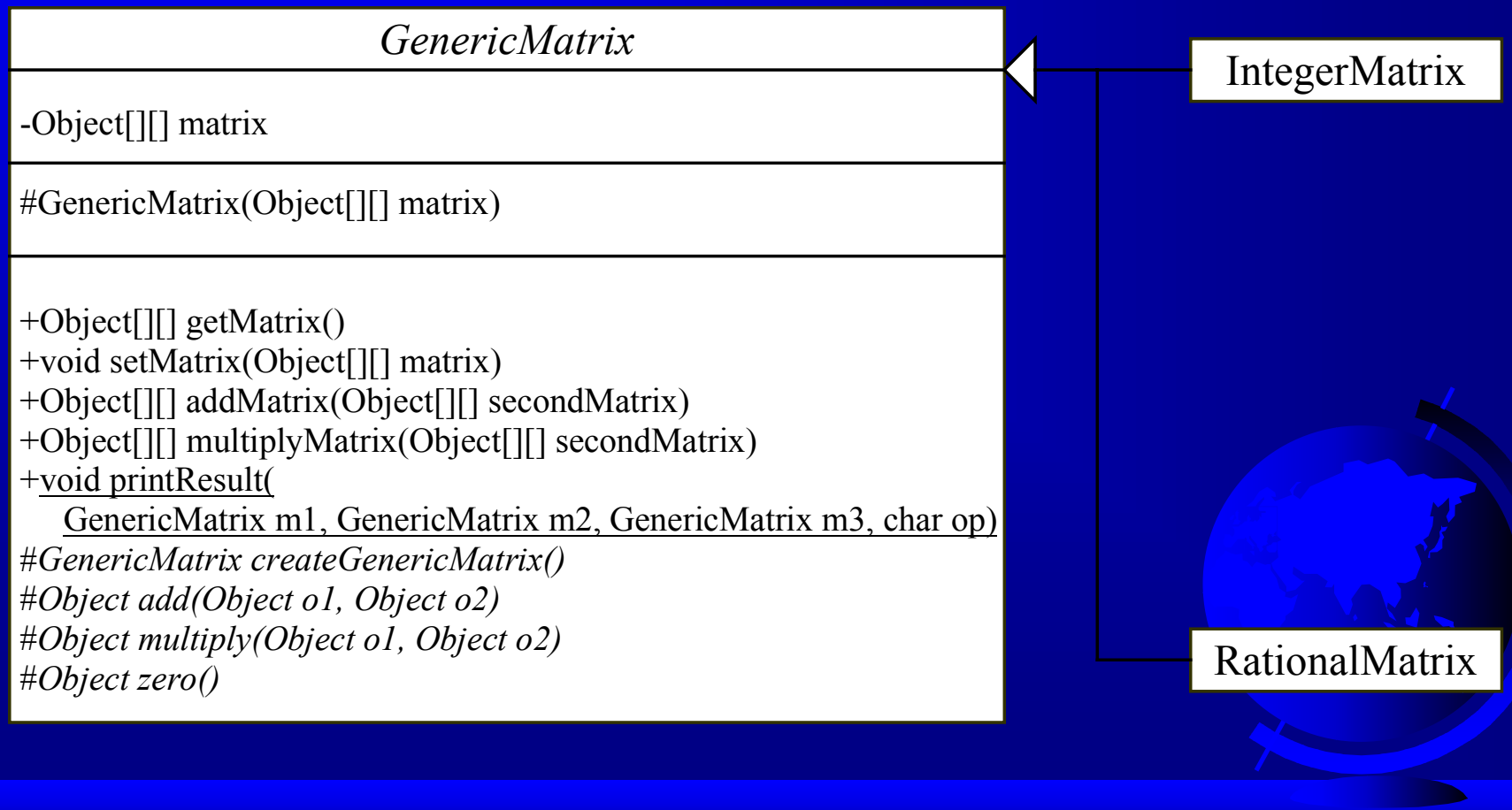
```
    // Display result
    System.out.println(args[1] + ' ' + args[0] + ' ' + args[2]
      + " = " + result);
  }
}
```


C:\WINNT\System32\cmd.exe
E:\PJspring2002\book\ch06>java Calculator - 63 40
Exception in thread "main" java.lang.NoClassDefFoundError:
E:\PJspring2002\book\ch06>
E:\PJspring2002\book\ch06>cd
E:\PJspring2002\book\ch06
E:\PJspring2002\book\ch06>cd ..
E:\PJspring2002\book>cd ch07
E:\PJspring2002\book\ch07>java Calculator - 63 40
63 - 40 = 23
E:\PJspring2002\book\ch07>java Calculator + 63 40
63 + 40 = 103
E:\PJspring2002\book\ch07>java Calculator "*" 63 40
63 * 40 = 2520
E:\PJspring2002\book\ch07>java Calculator / 63 40
63 / 40 = 1
E:\PJspring2002\book\ch07>

Example 7.9
Designing Generic Classes

- Objective: This example gives a generic class for matrix arithmetic. This class implements matrix addition and multiplication common for all types of matrices.

Example 7.10
Extending Abstract Classes



```
// GenericMatrix.java: Define a matrix and its associated
// operations such as add and multiply
public abstract class GenericMatrix
{
  // Representation of a matrix using a two-dimensional array
  private Object[][] matrix;

  // Construct a matrix
  protected GenericMatrix(Object[][] matrix)
  {
    this.matrix = matrix;
  }
  // Getter method for matrix
  public Object[][] getMatrix()
  {
    return matrix;
  }
```

```
// Setter method for matrix
 public void setMatrix(Object[][] matrix)
 {
   this.matrix = matrix;
 }

 // Add two matrices
 public GenericMatrix addMatrix
                 (GenericMatrix secondGenericMatrix)
 {
   // Create a result matrix
   Object[][] result =
     new Object[matrix.length][matrix[0].length];

   // Obtain the second matrix
   Object[][] secondMatrix =  secondGenericMatrix.getMatrix();
```

```
// Check bounds of the two matrices
  if ((matrix.length != secondMatrix.length) ||
      (matrix[0].length != secondMatrix.length))
  {
    System.out.println(
      "The matrices do not have the same size");
    System.exit(0);
  }

  // Perform addition
  for (int i=0; i<result.length; i++)
    for (int j=0; j<result[i].length; j++)
      result[i][j] = add(matrix[i][j], secondMatrix[i][j]);
  GenericMatrix genericResultMatrix = createGenericMatrix();
  genericResultMatrix.setMatrix(result);
  return genericResultMatrix;
}
```

```
// Multiply two matrices
 public GenericMatrix
   multiplyMatrix(GenericMatrix secondGenericMatrix)
 {
   // Obtain the second matrix
   Object[][] secondMatrix =  secondGenericMatrix.getMatrix();

   // Create result matrix
   Object[][] result =
     new Object[matrix.length][secondMatrix[0].length];

   // Check bounds
   if (matrix[0].length != secondMatrix.length)
   {
     System.out.println("Bounds error");
     System.exit(0);
   }
```

```
// Perform multiplication of two matrices
    for (int i=0; i<result.length; i++)
      for (int j=0; j<result[0].length; j++)
    {
      result[i][j] = zero();

      for (int k=0; k<matrix[0].length; k++)
      {
        result[i][j] = add(result[i][j],
          multiply(this.matrix[i][k], secondMatrix[k][j]));
      }
    }

    GenericMatrix genericResultMatrix = createGenericMatrix();
    genericResultMatrix.setMatrix(result);
    return genericResultMatrix;
  }
```

```
// Print matrice, the operator, and their operation result
public static void printResult(
  GenericMatrix m1, GenericMatrix m2, GenericMatrix m3, char op)
{
  for (int i=0; i<(m1.getMatrix()).length; i++)
  {
    for (int j=0; j<(m1.getMatrix())[0].length; j++)
      System.out.print(" " + (m1.getMatrix())[i][j]);

    if (i == (m1.getMatrix()).length/2)
      System.out.print( "   " + op + "   " );
    else
      System.out.print( "       " );

    for (int j=0; j<(m2.getMatrix()).length; j++)
      System.out.print(" " + (m2.getMatrix())[i][j]);
```

```
    if (i == (m1.getMatrix()).length/2)
      System.out.print( "  =  " );
    else
      System.out.print( "     " );

    for (int j=0; j<(m3.getMatrix()).length; j++)
      System.out.print(" " + (m3.getMatrix())[i][j]);

    System.out.println();
  }
}
```

```
// Abstract method for creating a GenericMatrix instance
public abstract GenericMatrix createGenericMatrix();

// Abstract method for adding two elements of the matrices
protected abstract Object add(Object o1, Object o2);

// Abstract method for multiplying two elements of the matrices
protected abstract Object multiply(Object o1, Object o2);

// Abstract method for defining zero for the matrix element
protected abstract Object zero();
}
```

```
// IntegerMatrix.java:
// Declare IntegerMatrix derived from GenericMatrix
public class IntegerMatrix extends GenericMatrix
{
  // Construct an IntegerMatrix
  public IntegerMatrix(Integer[][] m)
  {
    super(m);
  }

  // Implement the createGenericMatrix method
  public GenericMatrix createGenericMatrix()
  {
    Integer[][] matrix =
      new Integer[getMatrix().length][getMatrix().length];
    return new IntegerMatrix(matrix);
  }
```

```
// Implement the add method for adding two matrix elements
protected Object add(Object o1, Object o2)
{
  Integer i1 = (Integer)o1;
  Integer i2 = (Integer)o2;
  return new Integer(i1.intValue() + i2.intValue());
}

// Implement the multiply method for multiplying two
// matrix elements
protected Object multiply(Object o1, Object o2)
{
  Integer i1 = (Integer)o1;
  Integer i2 = (Integer)o2;
  return new Integer(i1.intValue() * i2.intValue());
}
```

```
  // Implement the zero method to specify zero for Integer
  protected Object zero()
  {
    return new Integer(0);
  }
}
```

```
// RationalMatrix.java:
// Declare RationalMatrix derived from GenericMatrix
public class RationalMatrix extends GenericMatrix
{
  // Construct a RationalMatrix for a given Ratonal array
  public RationalMatrix(Rational[][] m1)
  {
    super(m1);
  }

  // Implement the createGenericMatrix method
  public GenericMatrix createGenericMatrix()
  {
    Rational[][] matrix =
new Rational[getMatrix().length][getMatrix().length];
    return new RationalMatrix(matrix);
  }
```

```
// Implement the add method for adding two rational elements
 protected Object add(Object o1, Object o2)
 {
   Rational r1 = (Rational)o1;
   Rational r2 = (Rational)o2;
   return r1.add(r2);
 }

 // Implement the multiply method for multiplying
 // two rational elements
 protected Object multiply(Object o1, Object o2)
 {
   Rational r1 = (Rational)o1;
   Rational r2 = (Rational)o2;
   return r1.multiply(r2);
 }
```

```
  // Implement the zero method to specify zero for Rational
  protected Object zero()
  {
    return new Rational(0,1);
  }
}
```

Example 7.10
Extending Abstract Classes, cont.

- Objective: This example gives two programs that utilize the GenericMatrix class for integer matrix arithmetic and rational matrix arithmetic.

IntegerMatrix

TestIntegerMatrix

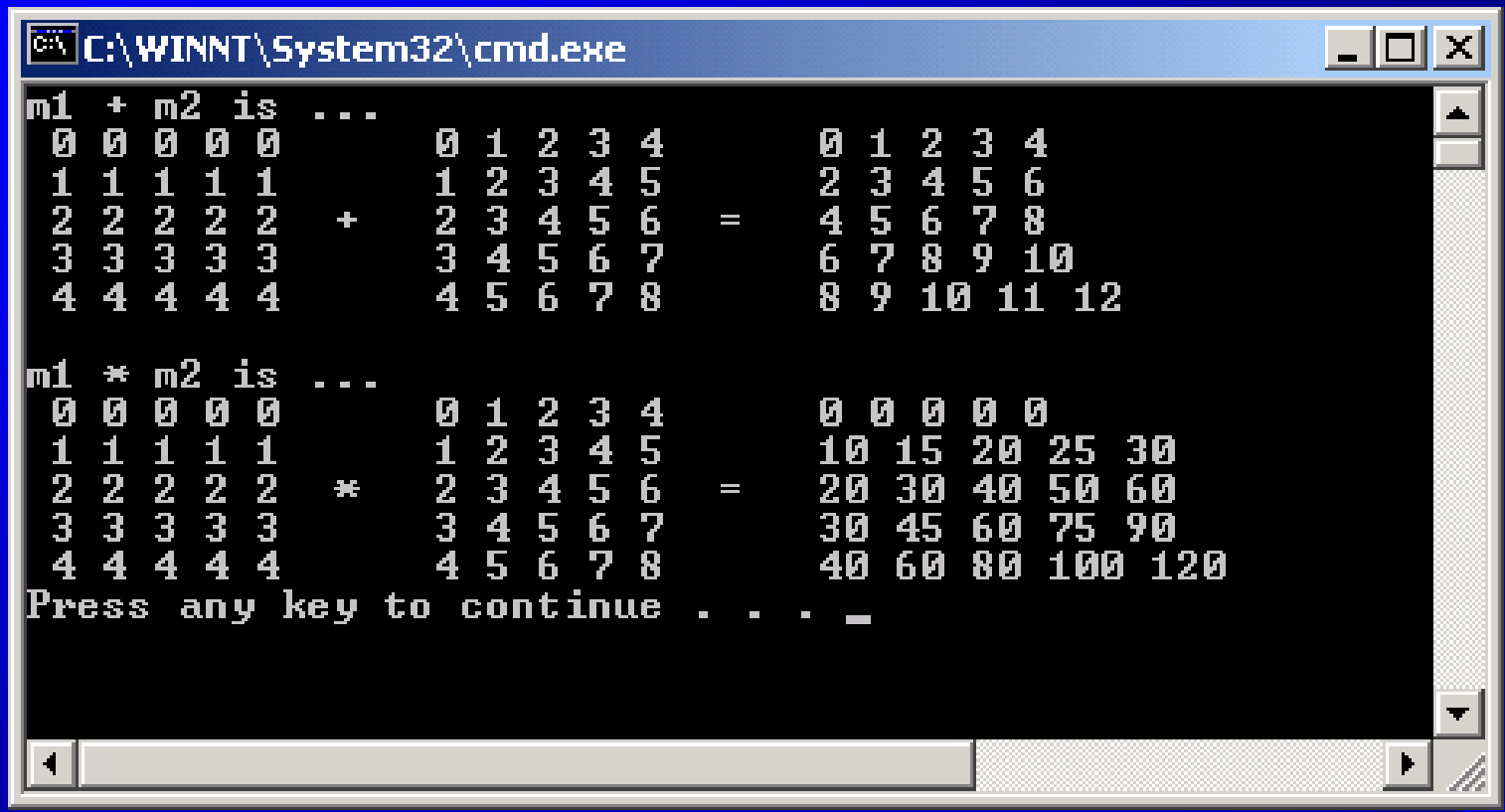
RationalMatrix

TestRationalMatrix

```
// TestIntegerMatrix.java: Test matrix operations involving
// integer values
public class TestIntegerMatrix
{
  // Main method
  public static void main(String[] args)
  {
    // Create Integer arrays m1, m2
    Integer[][] m1 = new Integer[5][5];
    Integer[][] m2 = new Integer[5][5];

    // Intialize Integer arrays m1 and m2
    for (int i=0; i<m1.length; i++)
      for (int j=0; j<m1[0].length; j++)
      {
        m1[i][j] = new Integer(i);
      }
```

```
for (int i=0; i<m2.length; i++)
    for (int j=0; j<m2[0].length; j++)
    {
      m2[i][j] = new Integer(i+j);
    }
  // Create instances of IntegerMatrix
  IntegerMatrix im1 = new IntegerMatrix(m1);
  IntegerMatrix im2 = new IntegerMatrix(m2);
  // Perform integer matrix addition, and multiplication
  IntegerMatrix im3 = (IntegerMatrix)im1.addMatrix(im2);
  IntegerMatrix im4 = (IntegerMatrix)im1.multiplyMatrix(im2);
  // Display im1, im2, im3, im4
  System.out.println("m1 + m2 is ...");
  GenericMatrix.printResult(im1, im2, im3, '+');
  System.out.println("\nm1 * m2 is ...");
  GenericMatrix.printResult(im1, im2, im4, '*');
 }
}
```

```
for (int i=0; i<m2.length; i++)
    for (int j=0; j<m2[0].length; j++)
    {
      m2[i][j] = new Integer(i+j);
    }
  // Create instances of IntegerMatrix
  IntegerMatrix im1 = new IntegerMatrix(m1);
  IntegerMatrix im2 = new IntegerMatrix(m2);
  // Perform integer matrix addition, and multiplication
  IntegerMatrix im3 = (IntegerMatrix)im1.addMatrix(im2);
  IntegerMatrix im4 = (IntegerMatrix)im1.multiplyMatrix(im2);
  // Display im1, im2, im3, im4
  System.out.println("m1 + m2 is ...");
  GenericMatrix.printResult(im1, im2, im3, '+');
  System.out.println("\nm1 * m2 is ...");
  GenericMatrix.printResult(im1, im2, im4, '*');
 }
}
```


C:\WINNT\System32\cmd.exe
m1 + m2 is ...
0 0 0 0 0
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
+
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
=
0 1 2 3 4
2 3 4 5 6
4 5 6 7 8
6 7 8 9 10
8 9 10 11 12
m1 * m2 is ...
0 0 0 0 0
1 1 1 1 1
2 2 2 2 2
3 3 3 3 3
4 4 4 4 4
*
0 1 2 3 4
1 2 3 4 5
2 3 4 5 6
3 4 5 6 7
4 5 6 7 8
=
0 0 0 0 0
10 15 20 25 30
20 30 40 50 60
30 45 60 75 90
40 60 80 100 120
Press any key to continue . . .

```
// Rational.java: Define a rational number and its associated
// operations such as add, subtract, multiply, and divide
public class Rational
{
  // Data fields for numerator and denominator
  private long numerator = 0;
  private long denominator = 1;

  // Default constructor
  public Rational()
  {
    this(0, 1);
  }
```

```
// Construct a rational with specified numerator and denominator
  public Rational(long numerator, long denominator)
  {
    long gcd = gcd(numerator, denominator);
    this.numerator = numerator/gcd;
    this.denominator = denominator/gcd;
  }

  // Find GCD of two numbers
  private long gcd(long n, long d)
  {
    long t1 = Math.abs(n);
    long t2 = Math.abs(d);
    long remainder = t1%t2;
```

```
while (remainder != 0)
   {
    t1 = t2;
    t2 = remainder;
    remainder = t1%t2;
   }

  return t2;
}

// Getter method for numerator
public long getNumerator()
{
  return numerator;
}
```

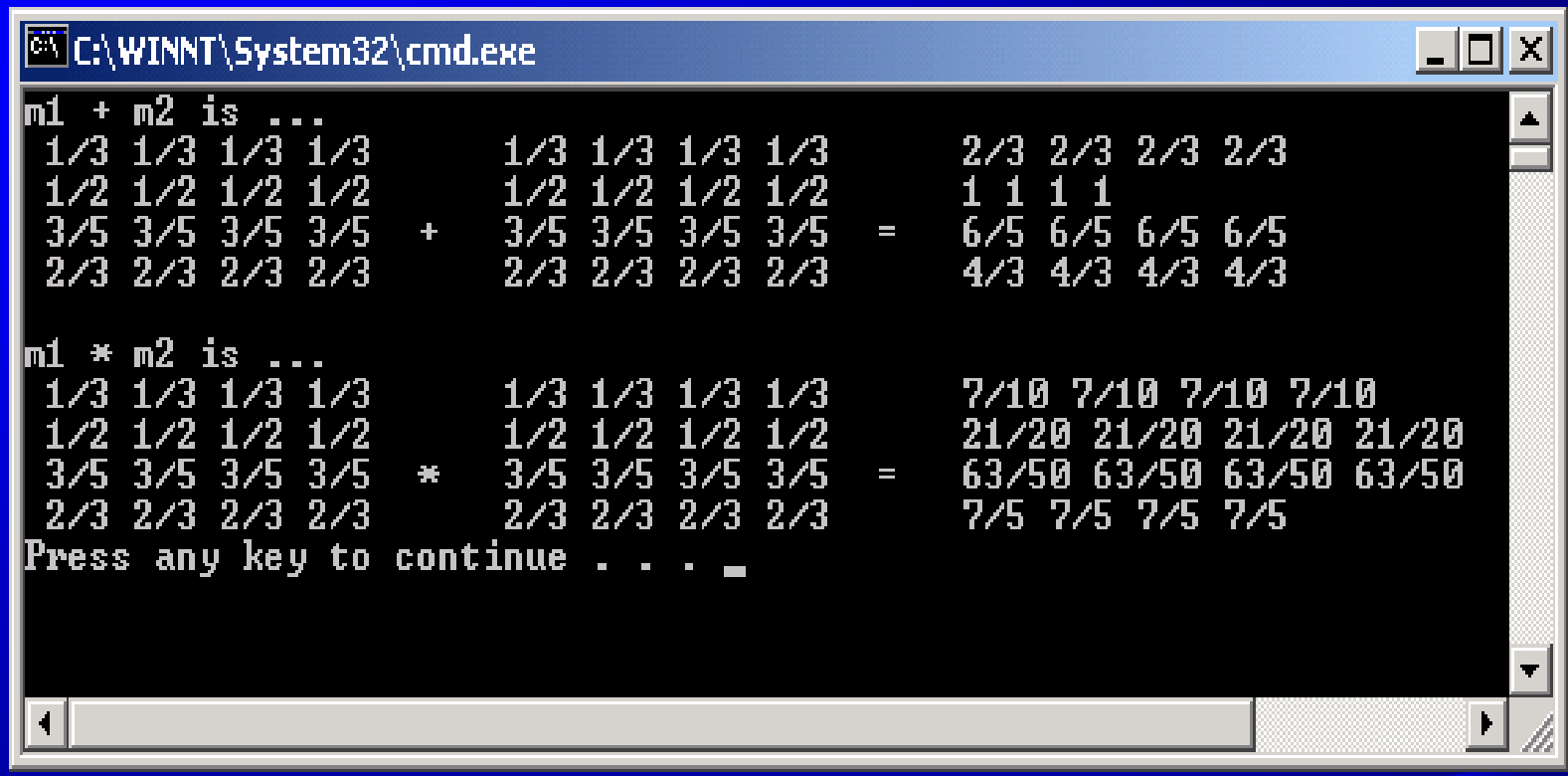
```
public long getDenominator()
 {
  return denominator;
 }

 // Add a rational number to this rational
 public Rational add(Rational secondRational)
 {
  long n = numerator*secondRational.getDenominator() +
    denominator*secondRational.getNumerator();
  long d = denominator*secondRational.getDenominator();
  return new Rational(n, d);
 }
```

```
// Subtract a rational number from this rational
 public Rational subtract(Rational secondRational)
 {
  long n = numerator*secondRational.getDenominator()
    - denominator*secondRational.getNumerator();
  long d = denominator*secondRational.getDenominator();
  return new Rational(n, d);
 }

 // Multiply a rational number to this rational
 public Rational multiply(Rational secondRational)
 {
  long n = numerator*secondRational.getNumerator();
  long d = denominator*secondRational.getDenominator();
  return new Rational(n, d);
 }
```

```
// Divide a rational number from this rational
 public Rational divide(Rational secondRational)
   throws RuntimeException
 {
   if (secondRational.getNumerator() == 0)
     throw new RuntimeException("Denominator cannot be zero");
   long n = numerator*secondRational.getDenominator();
   long d = denominator*secondRational.getNumerator();
   return new Rational(n, d);
 }
public String toString()     // Override the toString() method
 {
   if (denominator == 1)
     return numerator + "";
   else
     return numerator + "/" + denominator;
 }
}
```

```
// TestRationalMatrix.java: Test matrix operations involving
// Rational values
public class TestRationalMatrix
{
  // Main method
  public static void main(String[] args)
  {
    // Declare Rational arrays m1, m2
    Rational[][] m1 = new Rational[4][4];
    Rational[][] m2 = new Rational[4][4];

    // Initialize Rational arrays m1 and m2
    for (int i=0; i<m1.length; i++)
      for (int j=0; j<m1[0].length; j++)
      {
        m1[i][j] = new Rational(i+1,i+3);
        m2[i][j] = new Rational(i+1,i+3);
      }
```

```
    // Create RationalMatrix instances
    RationalMatrix rm1 = new RationalMatrix(m1);
    RationalMatrix rm2 = new RationalMatrix(m2);

    // Perform Rational matrix addition, and multiplication
    RationalMatrix rm3 = (RationalMatrix)rm1.addMatrix(rm2);
    RationalMatrix rm4 = (RationalMatrix)rm1.multiplyMatrix(rm2);

    // Display rm1, rm2, rm3, rm4
    System.out.println("m1 + m2 is ...");
    GenericMatrix.printResult(rm1, rm2, rm3, '+');

    System.out.println("\nm1 * m2 is ...");
    GenericMatrix.printResult(rm1, rm2, rm4, '*');
  }
}
```


C:\WINNT\System32\cmd.exe
m1 + m2 is ...
1/3 1/3 1/3 1/3 1/3 1/3 1/3 1/3 2/3 2/3 2/3 2/3
1/2 1/2 1/2 1/2 1/2 1/2 1/2 1/2 1 1 1 1
3/5 3/5 3/5 3/5 + 3/5 3/5 3/5 3/5 = 6/5 6/5 6/5 6/5
2/3 2/3 2/3 2/3 2/3 2/3 2/3 2/3 4/3 4/3 4/3 4/3
m1 * m2 is ...
1/3 1/3 1/3 1/3 1/3 1/3 1/3 1/3 7/10 7/10 7/10 7/10
1/2 1/2 1/2 1/2 1/2 1/2 1/2 1/2 21/20 21/20 21/20 21/20
3/5 3/5 3/5 3/5 * 3/5 3/5 3/5 3/5 = 63/50 63/50 63/50 63/50
2/3 2/3 2/3 2/3 2/3 2/3 2/3 2/3 7/5 7/5 7/5 7/5
Press any key to continue . . .

The `Vector` Class

Java provides the Vector class in the java.util package, which can be used to store an unspecified number of elements. A vector can grow or shrink dynamically as needed.

The `Vector` Class, cont.

To create a vector, you can use its default constructor like this:

```
Vector vector = new Vector();
```

To add an element to the vector, use the addElement method as follows:

```
vector.addElement(anObject);
```

The element in the vector must be an object.

The element is appended to the vector.

The `Vector` Class, cont.

To retrieve an element, use the `elementAt` method as follows:

```
Object anObject = vector.elementAt(0);
```

To find the number of elements in the vector, use the size method as follows:

```
int numOfElements = vector.size();
```

The `Vector` Class, cont.

To insert an element at a specific index, use the insertElement method

```
vector.insertElement(anObject,4);
```

To replace an element at a specific index, use the setElement() method

```
vector.setElement(anObject, 6);
```

To remove an element at a specific index, use the removeElement() method

```
vector.removeElement(4);
```

Example 7.10
Assigning Grades Using a Vector

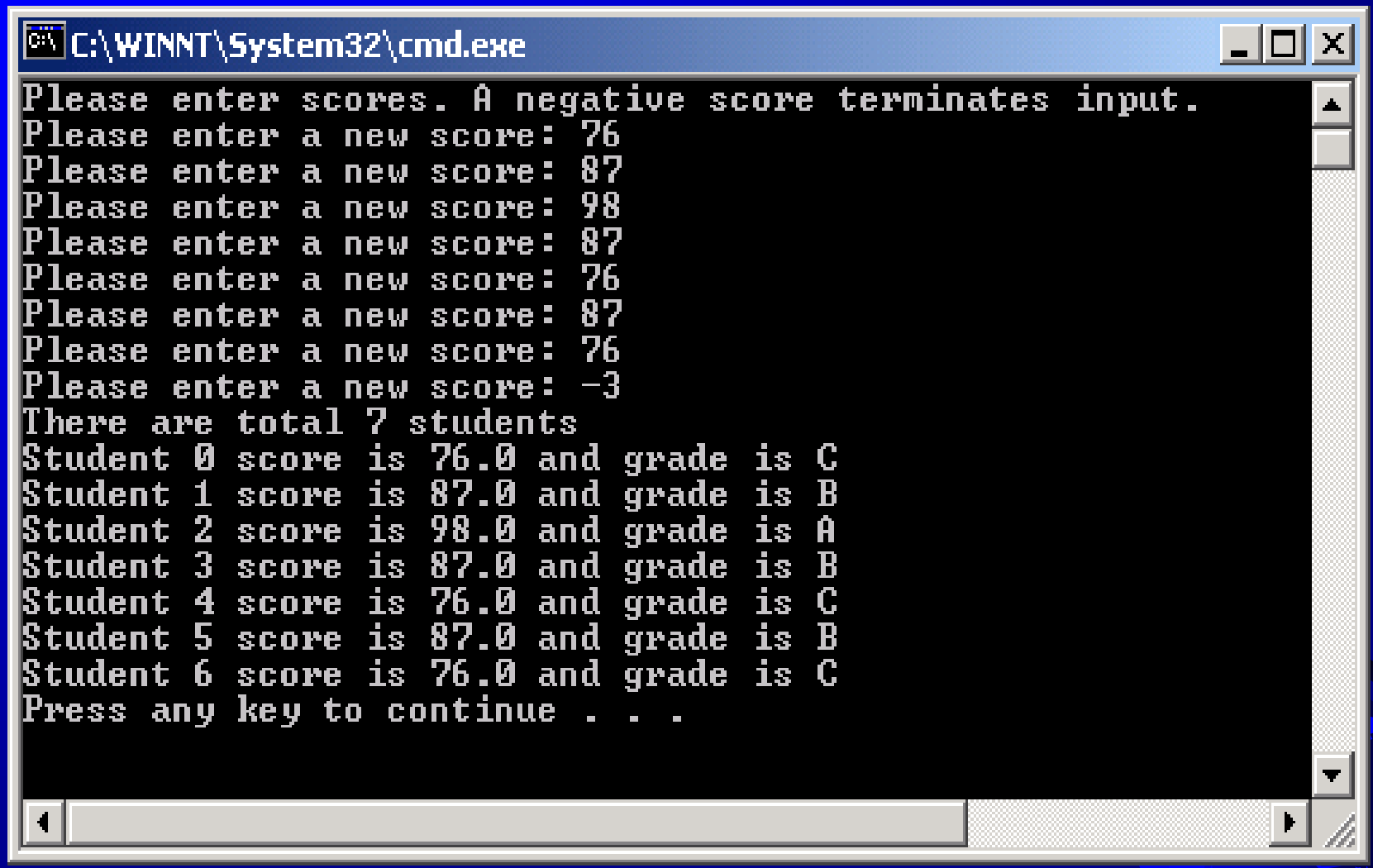
- Objective: Rewrites Example 7.1 using a vector, instead of using an array. The program reads student scores from the keyboard, stores the score in the vector, finds the best score, and then assigns grades for all the students.

```
// AssignGradeUsingVector.java: Assign grade
import java.util.*;

public class AssignGradeUsingVector
{
  // Main method
  public static void main(String[] args)
  {
    Vector scoreVector = new Vector(); // Vector to hold scores
    double best = 0; // The best score
    char grade; // The grade

    // Read scores and find the best score
    System.out.println("Please enter scores. " +
      "A negative score terminates input.");
```

```
do
    {
     System.out.print("Please enter a new score: ");
     double score = MyInput.readDouble();

     if (score < 0) break;

     // Add the score into the vector
     scoreVector.addElement(new Double(score));

     // Find the best score
     if (score > best)
       best = score;
    } while (true);
```

```
System.out.println("There are total " + scoreVector.size() +
    " students ");

  // Assign and display grades
  for (int i=0; i<scoreVector.size(); i++)
  {
    // Retrieve an element from the vector
    Double doubleObject = (Double)(scoreVector.elementAt(i));

    // Get the score
    double score = doubleObject.doubleValue();

    if ( score >= best - 10)
      grade = 'A';
    else if (score >= best - 20)
      grade = 'B';
```

```
else if (score >= best - 30)
        grade = 'C';
      else if (score >= best - 40)
        grade = 'D';
      else
        grade = 'F';

      System.out.println("Student " + i + " score is " + score +
        " and grade is " + grade);
    }
  }
}
```


C:\WINNT\System32\cmd.exe
Please enter scores. A negative score terminates input.
Please enter a new score: 76
Please enter a new score: 87
Please enter a new score: 98
Please enter a new score: 87
Please enter a new score: 76
Please enter a new score: 87
Please enter a new score: 76
Please enter a new score: -3
There are total 7 students
Student 0 score is 76.0 and grade is C
Student 1 score is 87.0 and grade is B
Student 2 score is 98.0 and grade is A
Student 3 score is 87.0 and grade is B
Student 4 score is 76.0 and grade is C
Student 5 score is 87.0 and grade is B
Student 6 score is 76.0 and grade is C
Press any key to continue . . .