7. Arrays

[ITP20003] Java Programming

Agenda

- Array Basics
- Arrays in Classes and Methods
- Programming with Arrays and Classes
- Sorting and Searching Arrays
- Multidimensional Arrays

Creating and Accessing Arrays

- An array is a special kind of object
  - Think of as collection of variables of same type

- Syntax for declaring an array with *new* operator

  Base_Type[ ] *Array_Name* = new *Base_Type*[*Length*];

  (*Base_Type Array_Name*[ ] = new *Base_Type*[*Length*]; is also OK.)

  Ex) double [ ] temperature = new double [7];

- To access an element use
  - The name of the array + an index number enclosed in braces

    Ex) temperature[5];

Array Indices

- Index of first array element is 0
- Last valid Index is arrayName.length – 1
- Array indices must be within bounds to be valid
  - When program tries to access outside bounds, run time error occurs

Array Basics

- Let's assume we want to compute the average temperature for the seven days. We may write the code as follows,

```
Scanner keyboard = new Scanner(System.in);
System.out.println("Enter 7 temperatures:");
double sum = 0;

for (int count = 0; count < 7; count++){
    double next = keyboard.nextDouble();
    sum = sum + next;
}
double average = sum / 7;
```

Array Basics

- But if we want to compare each temperature with the average?
- One possible way is to declare the seven variables of double type.

```
double temperature1;
double temperature2;
double temperature3;
double temperature4;
double temperature5;
double temperature6;
double temperature7;
```

- What if we should compute the average for a year?

Creating and Accessing Arrays

- We may use array for the previous problem,

```
double [] temperature = new double[7];
temperature[0], temperature[1], …
temperature[5], temperature[6]
```

‘Index’ or ‘subscript’
=> Integer number
and start with 0.

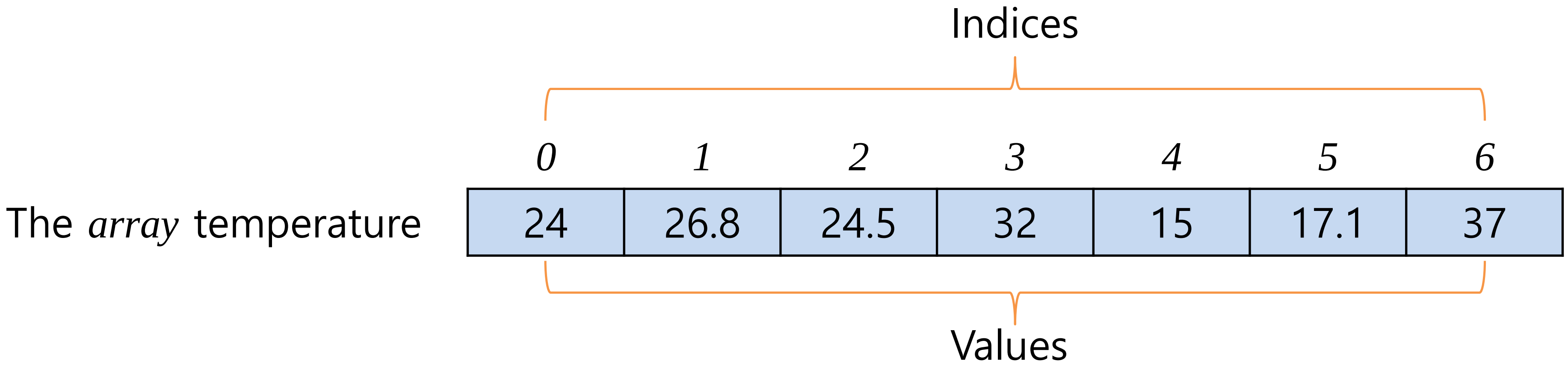
```
temperature[5]
```

Called	‘Indexed variable’
or	‘Subscripted variable’
or	‘Array elements’

Creating and Accessing Arrays

- The indexed variables can be used like an other variables.

```
temperature[3] = 32;
temperature[6] = temperature[3] + 5;

int index = 6;

System.out.println(temperature[index]);
```



Temperature example using arrays

```
import java.util.Scanner;
public class ArrayOfTemperatures{
public static void main(String[] args){
     double[] temperature = new double[7];
     // Read temperatures and compute their average:
     Scanner keyboard = new Scanner(System.in);
     System.out.println("Enter 7 temperatures:");
     double sum = 0;
     for (int index = 0; index < 7; index++){
          temperature[index] = keyboard.nextDouble();
          sum = sum + temperature[index];
     }
     double average = sum / 7;
     System.out.println("The average temperature is " + average);
     // Display each temperature and its relation to the average:
     System.out.println("The temperatures are");
     for (int index = 0; index < 7; index++){
     if (temperature[index] < average)
          System.out.println(temperature[index] + " below average");
     else if (temperature[index] > average)
          System.out.println(temperature[index] + " above average");
     else //temperature[index] == average
          System.out.println(temperature[index] + " the average");
     }
     System.out.println("Have a nice week.");
     }
}
```

Temperature example using arrays

```
Enter 7 temperatures:
32
30
25.7
26
34
31.5
29
The average temperature is
29.742857142857142857
The temperatures are
32.0 above average
30.0 above average
25.7 below average
26.0 below average
34.0 above average
31.5 above average
29.0 below average
Have a nice week.
```

Square Brackets with Arrays

- With a data type when declaring an array

```
int [ ] pressure;
```

- To enclose an integer expression to declare the length of the array

```
pressure = new int [100];
```

- To name an indexed value of the array

```
pressure[3] = keyboard.nextInt();
```

The Instance Variable length

- As an object,
  an array has only one public instance variable
  - Variable *length*
    - Contains number of elements in the array.
    - It is final, value cannot be changed.

Initializing Arrays

- Possible to initialize at declaration time

```java
double[] reading = {3.3, 15.8, 9.7};
```

- Also may use normal assignment statements
  - One at a time
  - In a loop

```java
int[] count = new int[100];
for (int i = 0; i < 100; i++)
    count[i] = 0;
```

Agenda

- Array Basics
- **<u>Arrays in Classes and Methods</u>**
- Programming with Arrays and Classes
- Sorting and Searching Arrays
- Multidimensional Arrays

Indexed Variables as Method Arguments

- Indexed variable of an array, such as a[i], can be used anywhere variable of array base type can be used.

```
public class ArgumentDemo {// Listing 7.5
    public static void main (String [] args)    {
        Scanner keyboard = new Scanner (System.in);
        System.out.println ("Enter your score on exam 1:");
        int firstScore = keyboard.nextInt ();
        int [] nextScore = new int [3];
        for (int i = 0 ; i < nextScore.length ; i++)
            nextScore [i] = firstScore + 5 * i;
        for (int i = 0 ; i < nextScore.length ; i++) {
            double possibleAverage = getAverage (firstScore, nextScore [i]);
            System.out.println ("If your score on exam 2 is " + nextScore [i]);
            System.out.println ("your average will be " + possibleAverage);
        }
    }

    public static double getAverage (int n1, int n2)  {
        return (n1 + n2) / 2.0;
    }
}
```

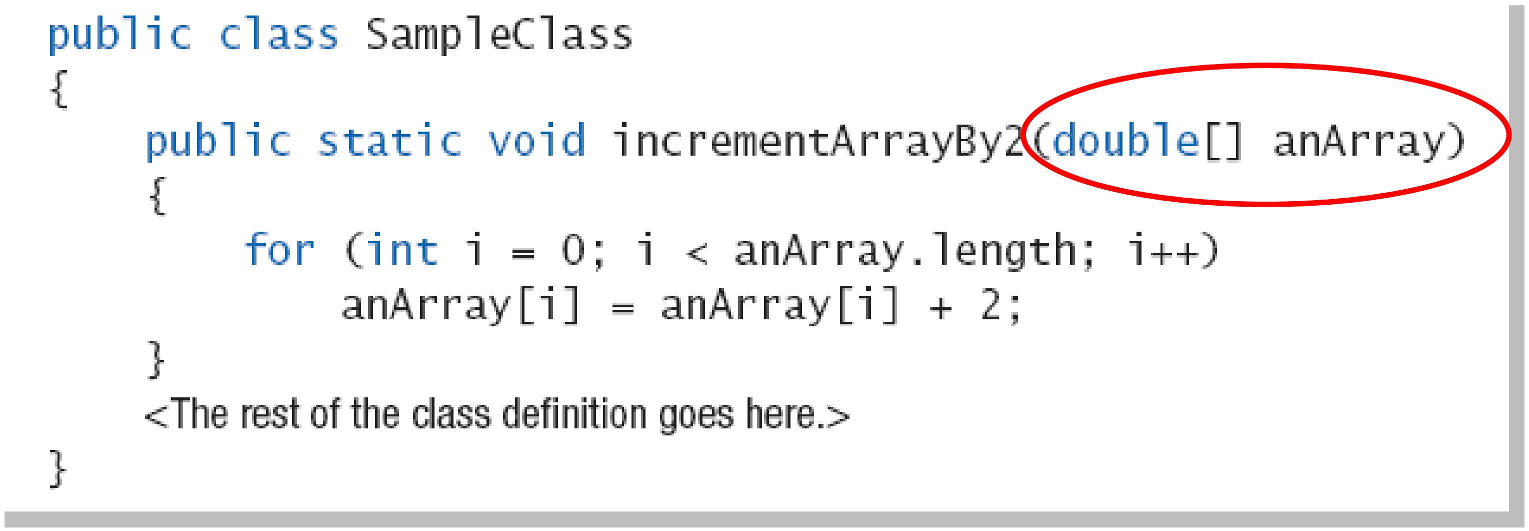
Indexed Variables as Method Arguments

```
Enter your score on exam 1:
89
If your score on exam 2 is 89
your average will be 89.0
If your score on exam 2 is 94
your average will be 91.5
If your score on exam 2 is 99
your average will be 94.0
```

Entire Arrays as Arguments

- Declaration of array parameter similar to how an array is declared

  Ex)



```
public class SampleClass
{
    public static void incrementArrayBy2(double[] anArray)
    {
        for (int i = 0; i < anArray.length; i++)
            anArray[i] = anArray[i] + 2;
    }
    <The rest of the class definition goes here.>
}
```

Entire Arrays as Arguments

- Array parameter in a method heading does not specify the length
  - An array of any length can be passed to the method.
  - Inside the method, elements of the array can be changed.
- When you pass the entire array, do not use square brackets in the actual parameter

Array Assignment and Equality

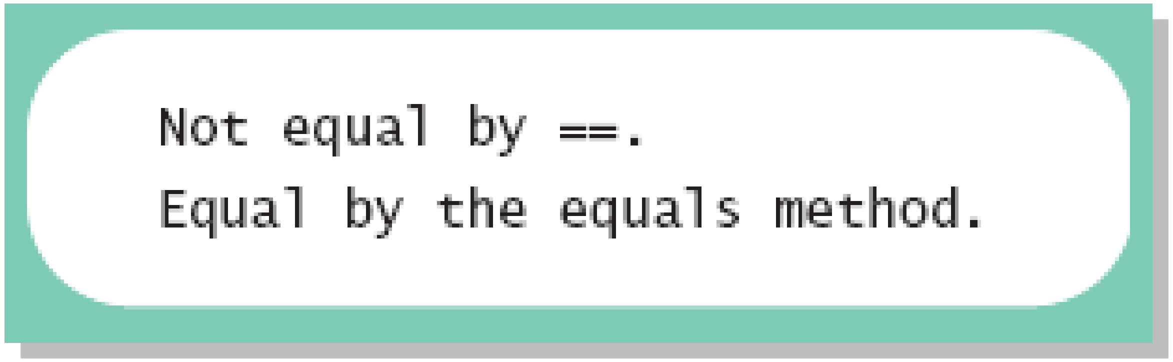
- Arrays are objects
  - Assignment and equality operators behave (misbehave) as specified in previous chapter
- Variable for the array object contains memory address of the object
  - Assignment operator = copies this address
  - Equality operator == tests whether two arrays are stored in same place in memory

Array Assignment and Equality

```java
public class TestEquals{
     public static void main(String[] args){
          int[] a = new int[3];          int[] b = new int[3];
          setArray(a);                   setArray(b);
          if (b == a)
                    System.out.println("Equal by ==.");
          else
                    System.out.println("Not equal by ==.");
          if (equals(b, a))
                    System.out.println("Equal by the equals method.");
          else
                    System.out.println("Not equal by the equals method.");
     }
     public static boolean equals(int[] a, int[] b){
          boolean elementsMatch = true;//tentatively
          if (a.length != b.length)      elementsMatch = false;
          else{int i = 0;
               while (elementsMatch && (i < a.length)){
                    if (a[i] != b[i])    elementsMatch = false;
                    i++;
               }
          }
          return elementsMatch;
     }
     public static void setArray(int[] array){
          for (int i = 0; i < array.length; i++)   array[i] = i;
     }
}
```

Array Assignment and Equality

- Two kinds of equality
  - View example program, listing 7.6 class TestEquals



```
Not equal by ==.
Equal by the equals method.
```

Array Assignment and Equality

- In Listing 7.6,
  - Note results of ==

  - Note definition and use of method *equals*
    - Receives two array parameters
    - Checks length and each individual pair of array elements
    - Note! *equals*() a method of *TestEquals*.

  - Remember array types are reference types

Methods that Return Arrays

- A Java method may return an array
- View example program, listing 7.7
  class ReturnArrayDemo
  - Note definition of return type as an array
- To return the array value
  - Declare a local array
  - Use that identifier in the return statement

Methods that Return Arrays (listing 7.7)

```java
import java.util.Scanner;
public class ReturnArrayDemo{
     public static void main(String[] args){
          Scanner kb = new Scanner(System.in);
          System.out.println("Enter your score on exam 1:");
          int firstScore = kb.nextInt();
          int[] nextScore = new int[3];
          for (int i = 0; i < nextScore.length; i++)
                    nextScore[i] = firstScore + 5 * i;

          double[] averageScore =getArrayOfAverages(firstScore, nextScore);
          for (int i = 0; i < nextScore.length; i++){
               System.out.println("If your score on exam 2 is " + nextScore[i]);
               System.out.println("your average will be " + averageScore[i]);
          }
     }

     public static double[] getArrayOfAverages(int firstScore, int[] nextScore){
          double[] temp = new double[nextScore.length];
          for (int i = 0; i < temp.length; i++)
                    temp[i] = getAverage(firstScore, nextScore[i]);
          return temp;
     }

     public static double getAverage(int n1, int n2){   return (n1 + n2) / 2.0; }
}
```

Methods that Return Arrays (listing 7.7)

```
Enter your score on exam 1:
89
If your score on exam 2 is 89
your average will be 89.0
If your score on exam 2 is 94
your average will be 91.5
If your score on exam 2 is 99
your average will be 94.0
```

Array as arguments

- An argument to a method may be an entire array.

SYNTAX

Return_Type Method_Name(*Base_Type*[ ] *Param_Name*)

EXAMPLES

public static int getOneElement(char[ ] anArray, int index)

public void readArray(int[ ] anotherArray)

Returning an array

SYNTAX

```
Base_Type[] Method_Name(Parameter_List){
    Base_Type[] temp = new Base_Type[Array_Size];
    …
    return temp;
}
```

EXAMPLE

```
public static char[] getVowels(){
    char[] newArray = {'a', 'e', 'i', 'o', 'u'};
    return newArray;
}
```

Arguments for Method *main*

- Recall heading of method main

  public static void main (**String[ ] args**)

- This declares an array
  - Formal parameter named *args*
  - Its base type is String

- Thus possible to pass to the run of a program multiple strings
  - These can then be used by the program.

Arguments for Method *main*

- We know every main method has the parameter part.
  `public static void main(String[] args)`

- What is it? Actually we have never called ‘main’ method.

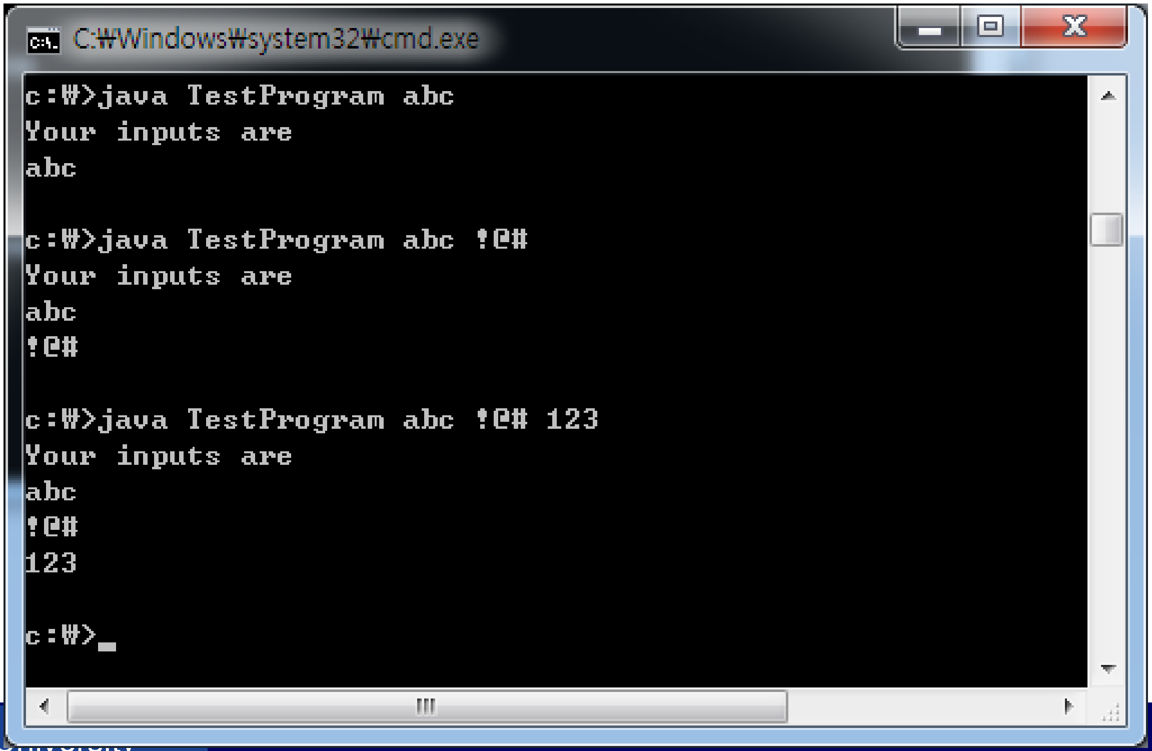
```
public class TestProgram {
public static void main(String[] args) {
System.out.println("Your inputs are");
for (int i=0; i<args.length;i++)
       System.out.println(args[i]);
}
}
```

```
Your inputs are
```

Arguments for Method *main*

- When you run the class, the main method is called by Java and the arguments are passed.

- Try run your bytecode in command window.



```
c:₩>java TestProgram abc
Your inputs are
abc

c:₩>java TestProgram abc !@#
Your inputs are
abc
!@#

c:₩>java TestProgram abc !@# 123
Your inputs are
abc
!@#
123

c:₩>_
```

Agenda

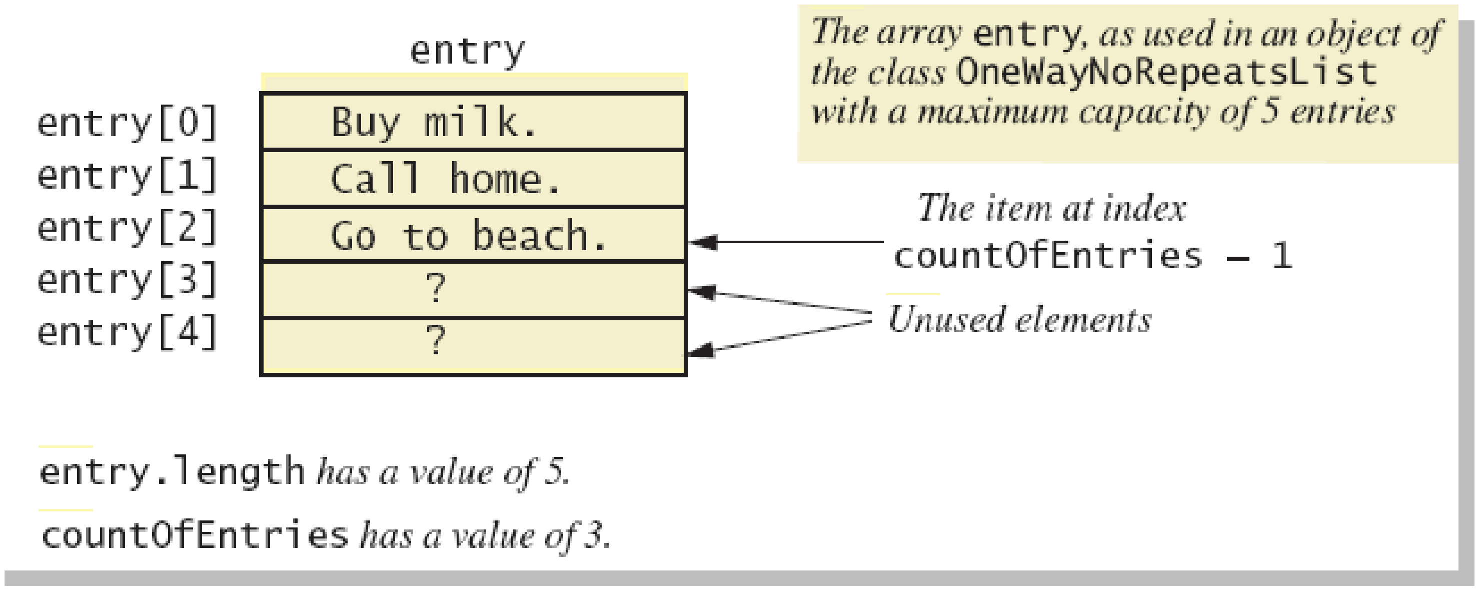
- Array Basics
- Arrays in Classes and Methods
- ~~**Programming with Arrays and Classes**~~ (skip)
- Sorting and Searching Arrays
- Multidimensional Arrays

Programming Example

- Goal: a specialized *List* class
  - Objects can be used for keeping lists of items
  - Implemented using an array.
- *List* as a special purpose array
  - The array is accessed only through class methods
  - You can add any checks and automatic processing
- Methods include
  - Capability to add items to the list
  - Also delete entire list, start with blank list
  - But no method to modify or delete list item

Partially Filled Arrays

- Array size specified at definition
- Not all elements of the array might receive values
  - This is termed a partially filled array
  - Programmer must keep track of how much of array is used



The array entry, as used in an object of the class OneWayNoRepeatsList with a maximum capacity of 5 entries
entry
entry[0]
entry[1]
entry[2]
entry[3]
entry[4]
Buy milk.
Call home.
Go to beach.
?
?
The item at index countOfEntries – 1
Unused elements
entry.length has a value of 5.
countOfEntries has a value of 3.

P08_List_main.java

```
import java.util.Scanner;
public class P08_List_main {
public static void main(String[] args) {
     int MAX_SIZE = 3;

     P08_List toDoList = new P08_List(MAX_SIZE);
     Scanner kb = new Scanner(System.in);
     String next = null;

     System.out.println("Enter items------");  // Reading items
     while (!toDoList.isFull()){      System.out.print("Item? ");
                                      next = kb.nextLine(); toDoList.addItem(next);}
     kb.close();

     printlist(toDoList);                             // Print the list

     System.out.print("Erasing the list...");  // Erase and Print
     toDoList.eraseList();  printlist(toDoList);
}
public static void printlist(P08_List list){
     if (list.isEmpty()==true)       System.out.println("\n\nThe List is empty.");
     else{String next = null;
          System.out.println("\n\nThe list contains:");
          int position = list.START_POSITION;
          next = list.getEntryAt(position);
          while (next != null) //null indicates end of list
          {        System.out.println(next);
                   next = list.getEntryAt(++position);
          }
     }
}}
```

P08_List_main.java Results

Case 1

```
Enter items------
Item? Go Home
Item? Do Homework
Item? Submit to Hisnet


The list contains:
Go Home
Do Homework
Submit to Hisnet


The List is empty.
```

Case 2

```
Enter items------
Item? Go Home
Item? Do Homework
Item? Do Homework
Already on the list. Not added.
Item? Submit to Hisnet


The list contains:
Go Home
Do Homework
Submit to Hisnet
Erasing the list...

The List is empty.
```

P08_List_main.java

```java
import java.util.Scanner;
public class P08_List_main {
public static void main(String[] args) {
     int MAX_SIZE = 3;

     P08_List toDoList = new P08_List(MAX_SIZE);
     Scanner kb = new Scanner(System.in);
     String next = null;

     System.out.println("Enter items------");  // Reading items
     while (!toDoList.isFull()){      System.out.print("Item? ");
                                      next = kb.nextLine(); toDoList.addItem(next);}
     kb.close();

     printlist(toDoList);                             // Print the list

     System.out.print("Erasing the list...");  // Erase and Print
     toDoList.eraseList();  printlist(toDoList);
}
public static void printlist(P08_List list){
     if (list.isEmpty()==true)        System.out.println("\n\nThe List is empty.");
     else{String next = null;
          System.out.println("\n\nThe list contains:");
          int position = list.START_POSITION;
          next = list.getEntryAt(position);
          while (next != null) //null indicates end of list
          {         System.out.println(next);
                    next = list.getEntryAt(++position);
          }
     }
}}
```

P08_List.java

```
public class P08_List
{
    public static int START_POSITION = 1;
    public static int DEFAULT_SIZE = 50; // used for default constructor.
    private int countOfEntries; //can be less than entry.length.
    private String[] entry;

    // Initialize the list (= array entry)
    public P08_List(int maximumNumberOfEntries){…}
    public P08_List() {…}

    public boolean isFull()      {…} // Check if the list is full.
    public boolean isEmpty()     {…} // Check if the list is empty.

    public void addItem(String item){…} // Add item to the list.

    public String getEntryAt(int position){…}//Returns the list at the pos.
    public boolean isOnList(String item) {…} //Check if the item is on the list.
    public int getMaximumNumberOfEntries(){…}//Returns the max. number of items.
    public int getNumberOfEntries()        {…} //Returns the number of items.
    public void eraseList()                {…} //Erase the list.
}
```

P08_List.java (implementation)

```
public P08_List(int maximumNumberOfEntries){
    entry = new String[maximumNumberOfEntries];
    countOfEntries = 0;
}
public P08_List(){
    entry = new String[DEFAULT_SIZE];
    countOfEntries = 0;
}
public boolean isFull()  {return countOfEntries == entry.length;}
public boolean isEmpty() {return countOfEntries == 0;}
public void addItem(String item){
    if (!isOnList(item)){
        if (countOfEntries == entry.length){
            System.out.println("Adding to a full list!");
            System.exit(0);
        }
        else{
            entry[countOfEntries] = item;
            countOfEntries++;
        }
    }
    else    System.out.println("Already on the list. Not added.");
}
```

P08_List.java (implementation)

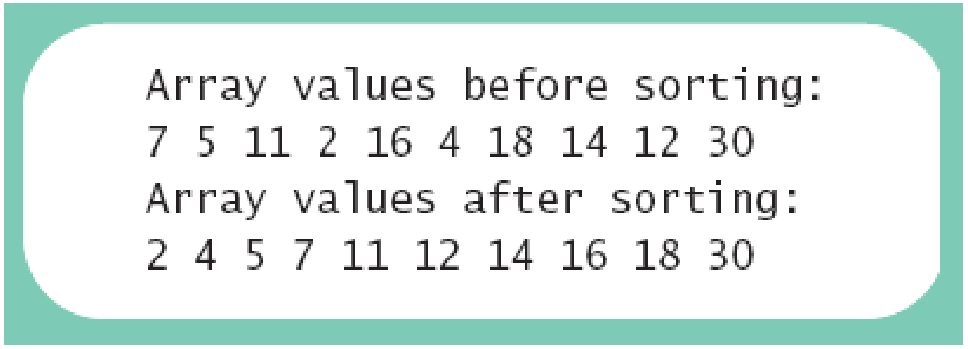
```java
public String getEntryAt(int position){
    String result = null;
    if ((1 <= position) && (position <= countOfEntries))
        result = entry[position - 1];
    return result;
}

public boolean atLastEntry(int position) {return position == countOfEntries;}

public boolean isOnList(String item){
    boolean found = false;
    int i = 0;
    while (!found && (i < countOfEntries)){
        if (item.equalsIgnoreCase(entry[i]))
                found = true;
        else
                i++;
    }
    return found;
}
public int getMaximumNumberOfEntries()   {return entry.length;}
public int getNumberOfEntries()          {return countOfEntries;}
public void eraseList()                  {countOfEntries = 0;}
```

Agenda

- Array Basics
- Arrays in Classes and Methods
- Programming with Arrays and Classes
- **Sorting and Searching Arrays**
- Multidimensional Arrays

Selection Sort

- Sorting: arranging all elements of an array so they are ascending (or descending) order

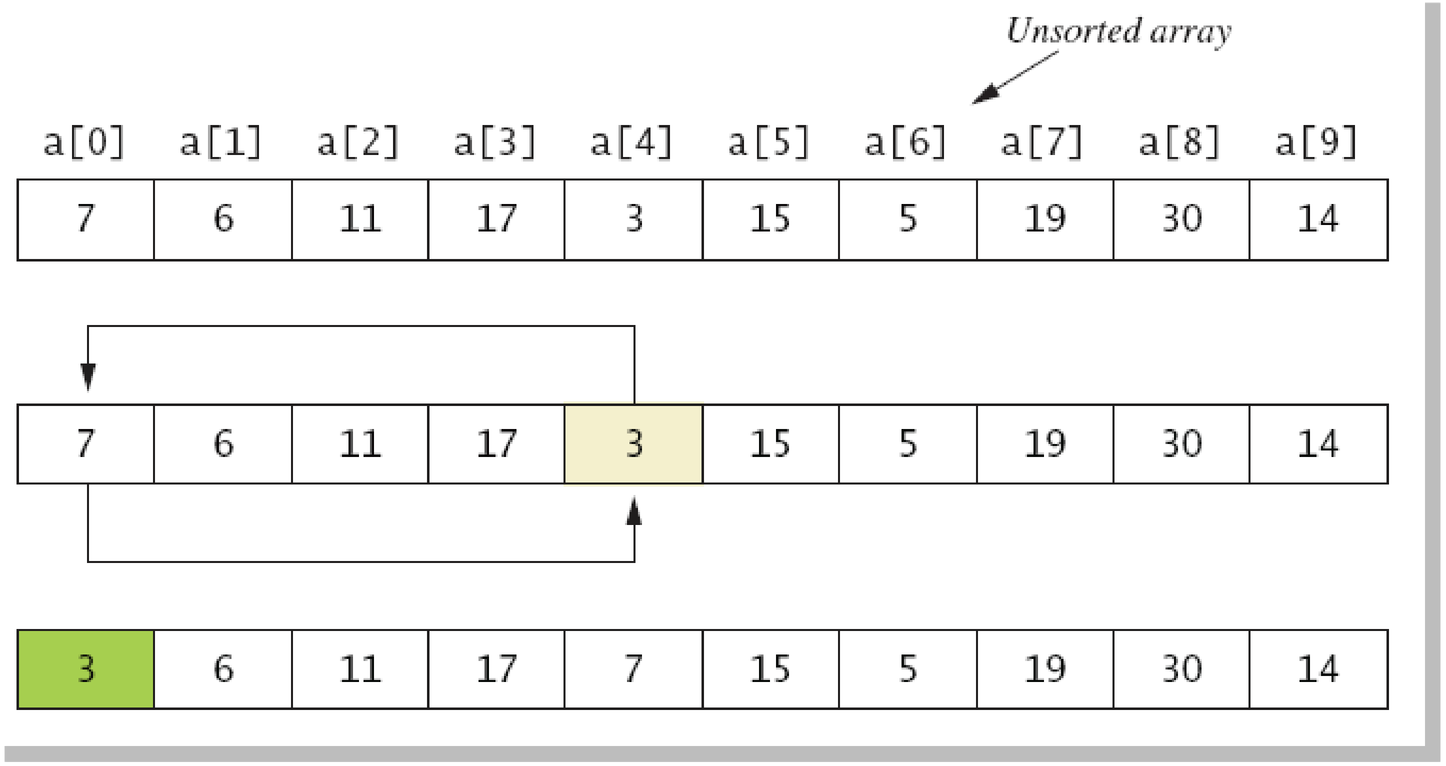


```
Array values before sorting:
7 5 11 2 16 4 18 14 12 30
Array values after sorting:
2 4 5 7 11 12 14 16 18 30
```

- Selection sort
  1. Algorithm is to step through the array
  2. Place smallest element in index 0
  3. Swap elements as needed to accomplish this

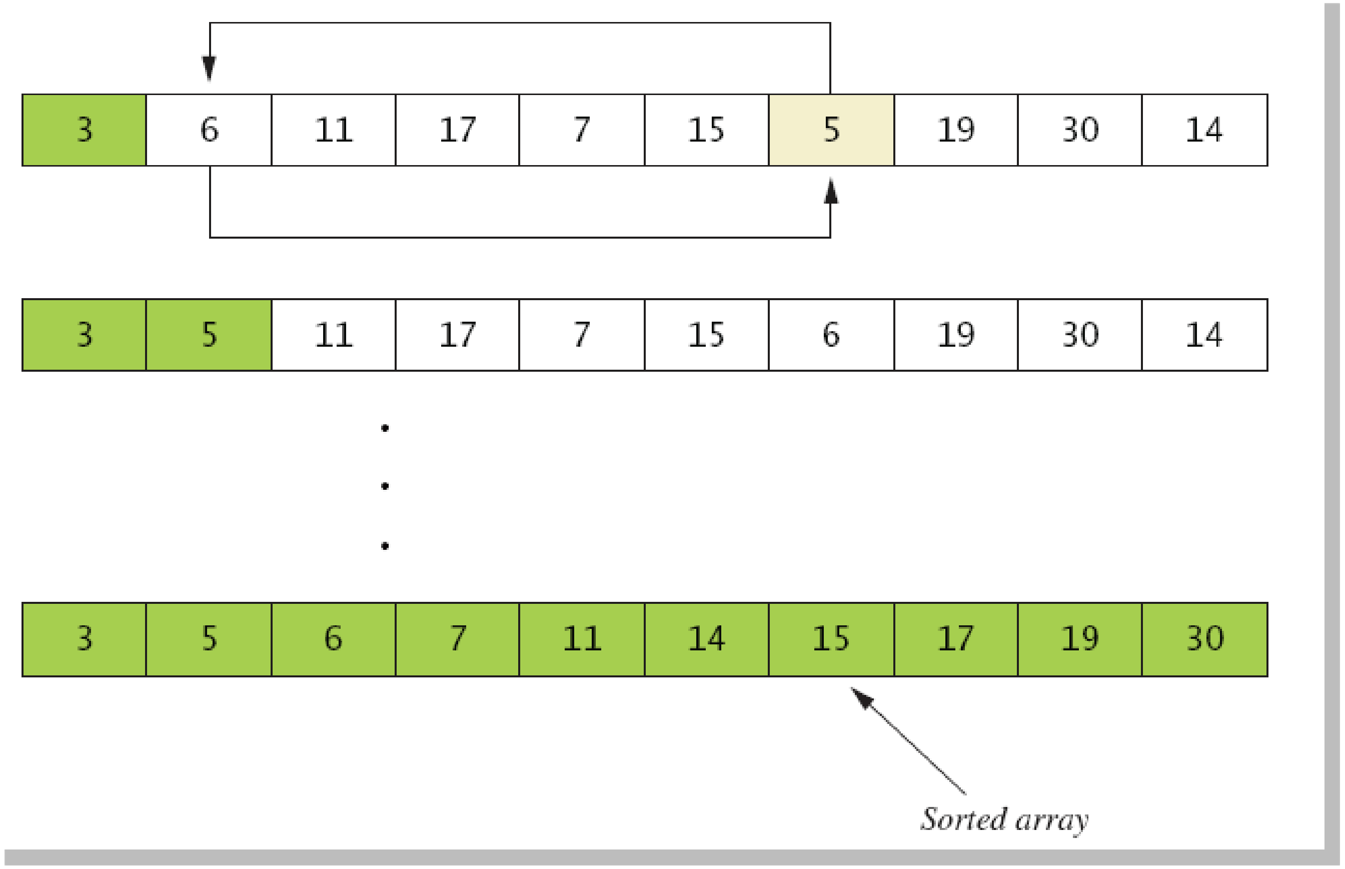
Selection Sort

- Figure 7.5a



Selection Sort

- Figure 7.5b



Selection Sort

- Algorithm for selection sort of an array

```
for (index = 0; index <a.length – 1; index++)
{ // Place the correct value in a[index]:
indexOfNextSmallest = the index of the smallest value among
                             a[index], a[index+1],...,
                             a[a.length - 1]

Interchange the values of a[index] and a[indexOfNextSmallest].
//Assertion: a[0] <= a[1] <= ... <= a[index] and these
//are the smallest of the original array elements.
//The remaining positions contain the rest of the
//original array elements.
}
```

Selection Sort

SelectionSortDemo
+ main(String[] args): static void + display(int[] array, String when): static void

ArraySorter
+ selectionSort(int[] anArray): static void - getIndexOfSmallest(int startIndex, int[] a): static int - interchange(int i, int j, int[] a): static void

Selection Sort

```
public class SelectionSortDemo
{
    public static void main(String[] args)
    {
    int[] b = {7, 5, 1, 2, 15, 4, 18, 14, 12, 30};
    display(b, "before");
    ArraySorter.selectionSort(b);
    display(b, "after");
    }

public static void display(int[] array, String when)
{
    System.out.println("Array values " + when + " sorting:");
    for (int i = 0; i < array.length; i++)
        System.out.print(array[i] + " ");
    System.out.println( );
}
}
```

Selection sort: ArraySorter class

```java
public class ArraySorter {
     /**  Precondition: Every element in anArray has a value.
     Action: Sorts the array into ascending order.*/
     public static void selectionSort(int[] anArray)
     {
          for (int index = 0; index < anArray.length - 1; index++){
               int indexOfNextSmallest = getIndexOfSmallest(index, anArray);
               interchange(index, indexOfNextSmallest, anArray);
          }
     }

     /**  Returns the index of the smallest value in the portion of the
     array that begins at the element whose index is startIndex and
     ends at the last element.        */
     private static int getIndexOfSmallest(int startIndex, int[] a)
     {
          …
          return indexOfMin;
     }

     /** Precondition: i and j are valid indices for the array a.
     Postcondition: Values of a[i] and a[j] have been interchanged.*/
     private static void interchange(int i, int j, int[] a)
     {
          int temp = a[i];   a[i] = a[j];      a[j] = temp; //original value of a[i]
     }
}
```

Selection sort: ArraySorter class

```
/** Returns the index of the smallest value in the portion of the
array that begins at the element whose index is startIndex and
ends at the last element. */
private static int getIndexOfSmallest(int startIndex, int[] a)
{
    int min = a[startIndex];
    int indexOfMin = startIndex;
    for (int index = startIndex + 1; index < a.length; index++){
    if (a[index] < min){
        min = a[index];
        indexOfMin = index;
        //min is smallest of a[startIndex] through a[index]
        }
    }
    return indexOfMin;
}
```

Selection sort

- Screen Output

```
Array values before sorting:
7 5 1 2 15 4 18 14 12 30
Array values after sorting:
1 2 4 5 7 12 14 15 18 30
```

Other Sorting Algorithms

- Selection sort is simplest
  - But it is very inefficient for large arrays

- Java Class Library provides for efficient sorting
  - Has a class called *Arrays*
  - Class has multiple versions of a sort method (static methods)
    Ex) Arrays.sort(int[] a), Arrays.sort(double[] a), …
  - See http://java.oracle.com or
    http://docs.oracle.com/javase/7/docs/api/java/util/Arrays.html

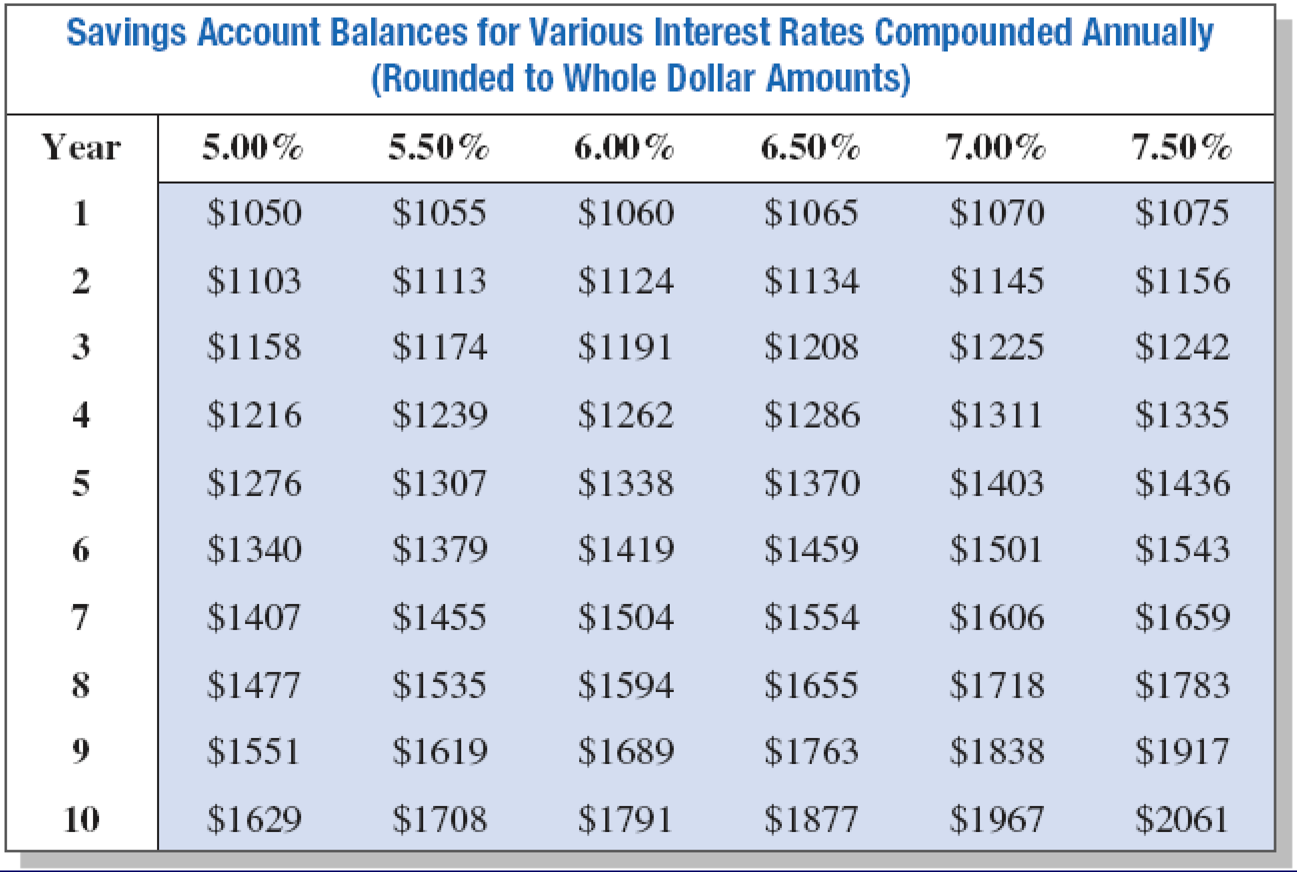
Agenda

- Array Basics
- Arrays in Classes and Methods
- Programming with Arrays and Classes
- Sorting and Searching Arrays
- **Multidimensional Arrays**

Multidimensional-Array Basics

- A table of values

  Ex) int [ ][ ] table = new int [10][6];



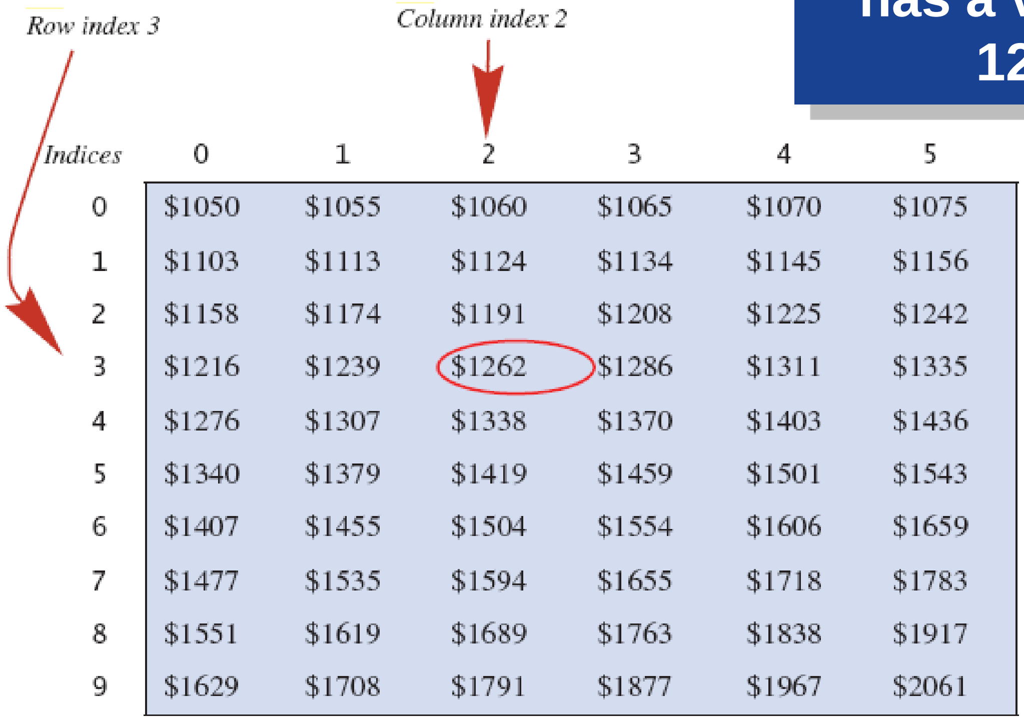
Savings Account Balances for Various Interest Rates Compounded Annually
(Rounded to Whole Dollar Amounts)

Year	5.00%	5.50%	6.00%	6.50%	7.00%	7.50%
1	$1050	$1055	$1060	$1065	$1070	$1075
2	$1103	$1113	$1124	$1134	$1145	$1156
3	$1158	$1174	$1191	$1208	$1225	$1242
4	$1216	$1239	$1262	$1286	$1311	$1335
5	$1276	$1307	$1338	$1370	$1403	$1436
6	$1340	$1379	$1419	$1459	$1501	$1543
7	$1407	$1455	$1504	$1554	$1606	$1659
8	$1477	$1535	$1594	$1655	$1718	$1783
9	$1551	$1619	$1689	$1763	$1838	$1917
10	$1629	$1708	$1791	$1877	$1967	$2061

Multidimensional-Array Basics

- Row and column indices for an array named table

`table[3][2]`
has a value of 1262

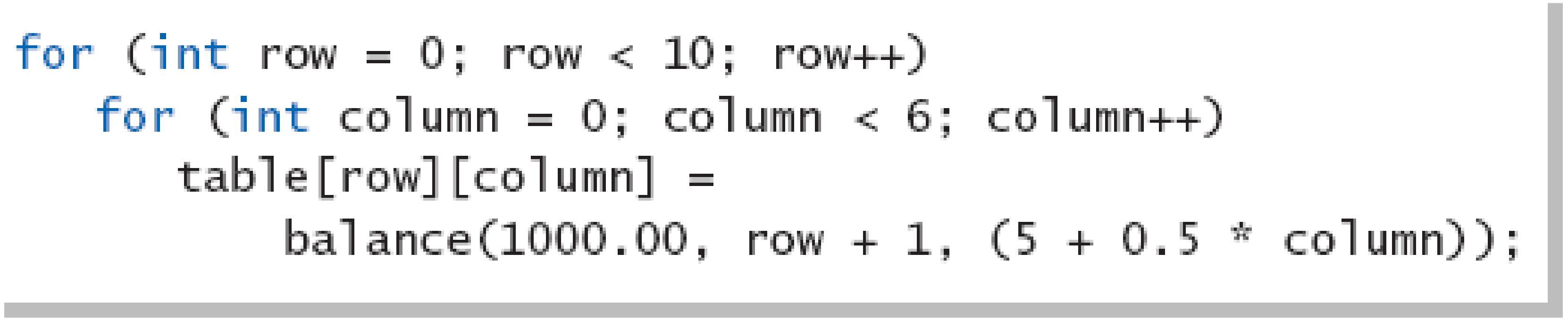

Row index 3
Column index 2

Indices	0	1	2	3	4	5
0	$1050	$1055	$1060	$1065	$1070	$1075
1	$1103	$1113	$1124	$1134	$1145	$1156
2	$1158	$1174	$1191	$1208	$1225	$1242
3	$1216	$1239	$1262	$1286	$1311	$1335
4	$1276	$1307	$1338	$1370	$1403	$1436
5	$1340	$1379	$1419	$1459	$1501	$1543
6	$1407	$1455	$1504	$1554	$1606	$1659
7	$1477	$1535	$1594	$1655	$1718	$1783
8	$1551	$1619	$1689	$1763	$1838	$1917
9	$1629	$1708	$1791	$1877	$1967	$2061

Multidimensional-Array Basics

- We can access elements of the table with a nested for loop

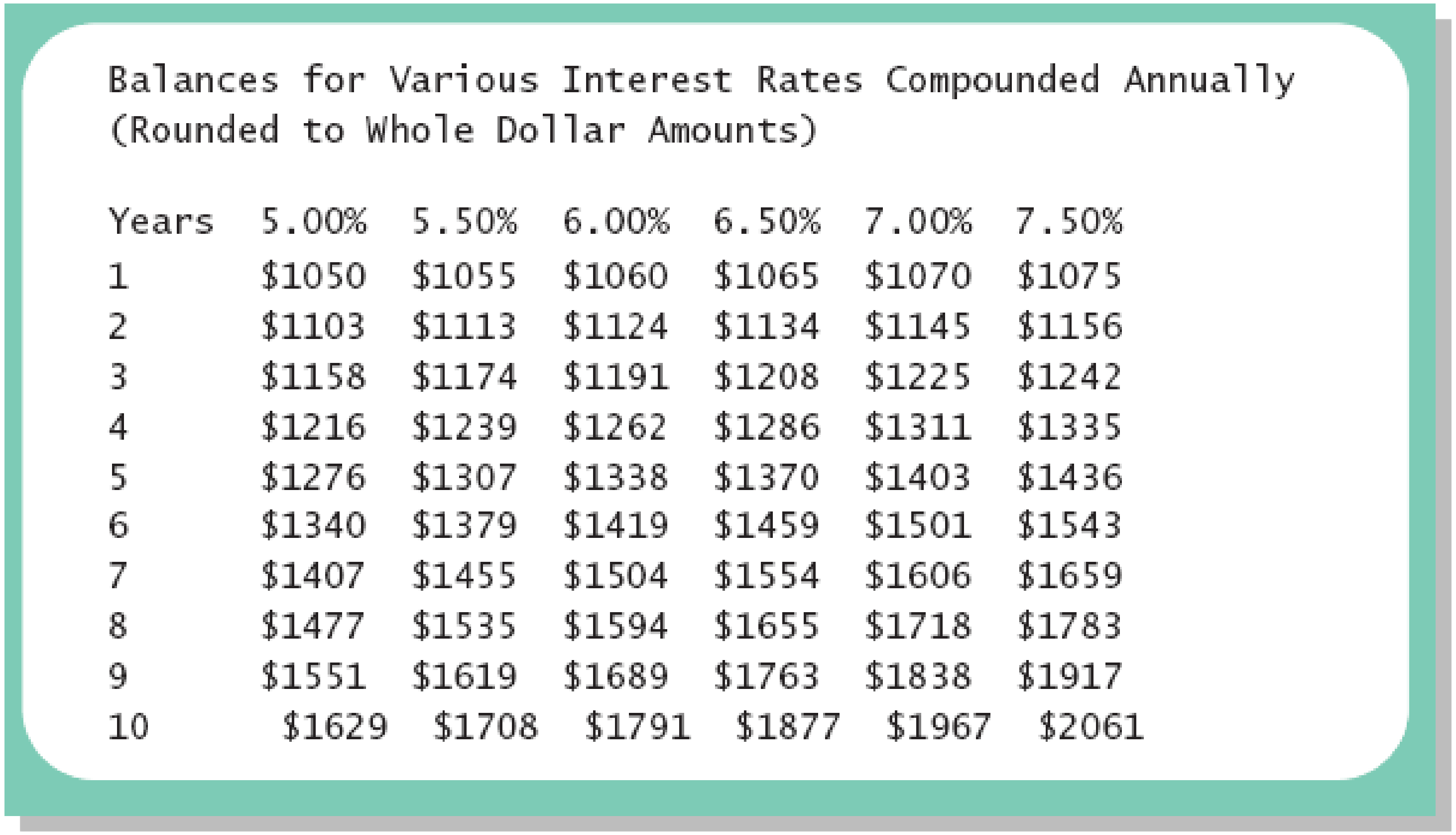
  Ex)



```
for (int row = 0; row < 10; row++)
    for (int column = 0; column < 6; column++)
        table[row][column] =
             balance(1000.00, row + 1, (5 + 0.5 * column));
```

- View sample program, listing 7.12
  class InterestTable

Multidimensional-Array Basics



```
Balances for Various Interest Rates Compounded Annually
(Rounded to Whole Dollar Amounts)

Years  5.00%  5.50%  6.00%  6.50%  7.00%  7.50%
1      $1050  $1055  $1060  $1065  $1070  $1075
2      $1103  $1113  $1124  $1134  $1145  $1156
3      $1158  $1174  $1191  $1208  $1225  $1242
4      $1216  $1239  $1262  $1286  $1311  $1335
5      $1276  $1307  $1338  $1370  $1403  $1436
6      $1340  $1379  $1419  $1459  $1501  $1543
7      $1407  $1455  $1504  $1554  $1606  $1659
8      $1477  $1535  $1594  $1655  $1718  $1783
9      $1551  $1619  $1689  $1763  $1838  $1917
10      $1629  $1708  $1791  $1877  $1967  $2061
```

Multidimensional-Array Parameters and Returned Values

- Methods can have
  - Parameters that are multidimensional-arrays
  - Return values that are multidimensional-arrays

- View sample code, listing 7.13
  class InterestTable2

Java's Representation of Multidimensional Arrays

- Multidimensional array represented as several one-dimensional arrays
- Given
  int [ ][ ] table = new int[10][6];
  - Array *table* is actually 1 dimensional of type int[ ]
  - It is an array of arrays
- Important when sequencing through multidimensional array

Initializing Multidimensional-Array

```
int[][] a = new int[3][5];
```

```
int[][] a = new int[3][];
a[0] = new int[5];
a[1] = new int[5];
a[2] = new int[5];
```

Ragged Arrays

- Not necessary for all rows to be of the same length

  Ex)

```
int[][] b;
b = new int[3][];
b[0] = new int[5]; //First row,  5 elements
b[1] = new int[7]; //Second row, 7 elements
b[2] = new int[4]; //Third row,  4 elements
```