Principles of Computer Science I

Prof. Nadeem Abdul Hamid

CSC 120 – Fall 2006

Lecture Unit 8 - Arrays

Lecture Outline

- Become familiar with arrays and array lists
- Using wrapper classes, auto-boxing
- Enhanced for loop
- Array algorithms
- Two-dimensional arrays

Arrays

- Many programs need to manipulate (large) collections of related data values
  - Would be very inefficient to use a bunch of variables: `data1`, `data2`, `data3`, …
- Array: sequence of values of the same type
  - To construct an array of 10 f.p. numbers:

    `new double[ 10 ]`

Constructing Arrays

- `new` operator constructs an array
- Array reference can be stored in a variable
  - Type of the array is the element type, followed by `[]`

    `double[] data = new double[ 10 ];`
- Can create arrays of any type (even other arrays)

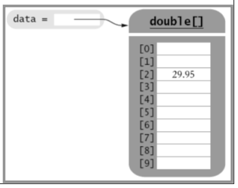
  `BankAccount[] accounts = new BankAccount[ 25 ];`
- Upon array construction, values initialized depending on type
  - Numbers: `0`
  - Boolean: `false`
  - Object references: `null`

Accessing Array Elements

- Specify array element by integer within square brackets `[ ]`

  `data[ 4 ]`
- Store values using an assignment statement

  `data[2] = 29.95;`
- Notice numbering starts at `0`
  - Array of length 10 has indices 0 to 9



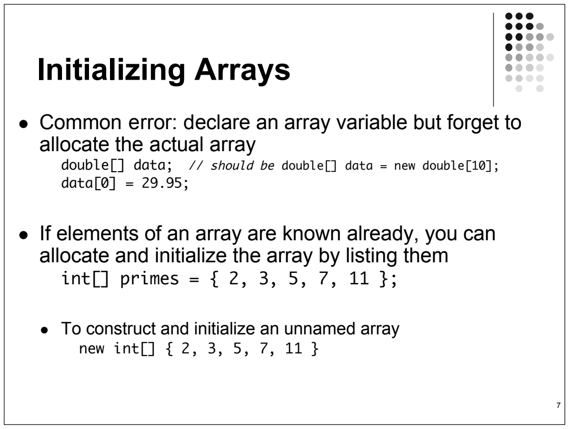
Array Bounds

- Index values start at 0 and go up to one less than the array length

  `data[10] = 29.95; // BOUNDS ERROR`
- To find the number of elements in an array use the length field

  `data.length`
  - Unlike most other properties of objects, length of arrays is an instance field, not a method (so no parentheses)
- Common for loop pattern for processing arrays

```
for ( int i = 0; i < data.length; i++ )
   . . .
```



Initializing Arrays

- Common error: declare an array variable but forget to allocate the actual array

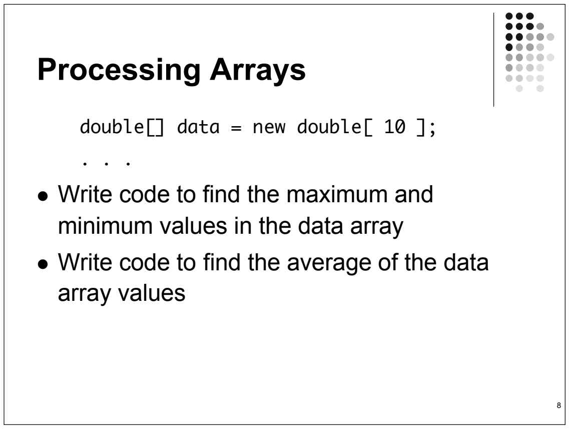
```
double[] data;  // should be double[] data = new double[10];
data[0] = 29.95;
```

- If elements of an array are known already, you can allocate and initialize the array by listing them

```
int[] primes = { 2, 3, 5, 7, 11 };
```

  - To construct and initialize an unnamed array

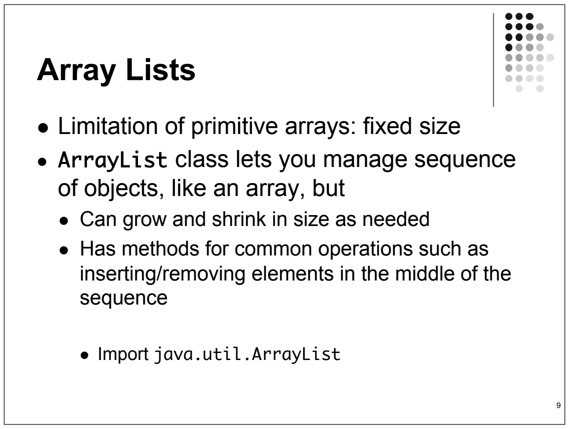
```
new int[] { 2, 3, 5, 7, 11 }
```



Processing Arrays

```
double[] data = new double[ 10 ];
. . .
```

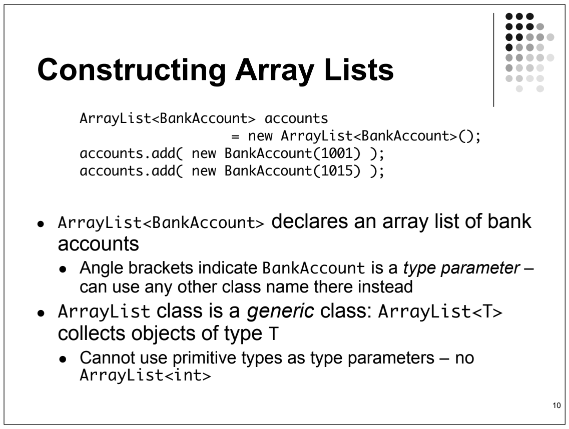
- Write code to find the maximum and minimum values in the data array
- Write code to find the average of the data array values



Array Lists

- Limitation of primitive arrays: fixed size
- `ArrayList` class lets you manage sequence of objects, like an array, but
  - Can grow and shrink in size as needed
  - Has methods for common operations such as inserting/removing elements in the middle of the sequence

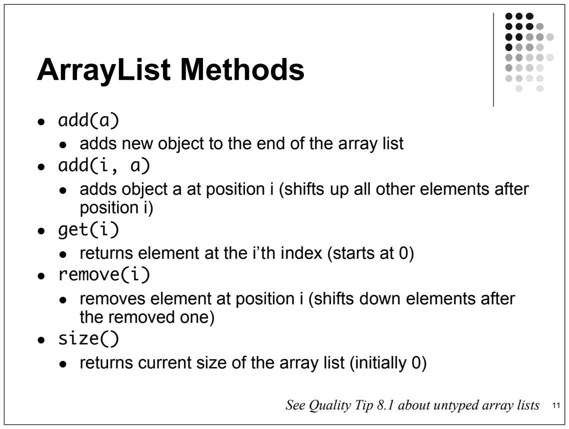
  - Import `java.util.ArrayList`



Constructing Array Lists

```
ArrayList<BankAccount> accounts
                    = new ArrayList<BankAccount>();
accounts.add( new BankAccount(1001) );
accounts.add( new BankAccount(1015) );
```

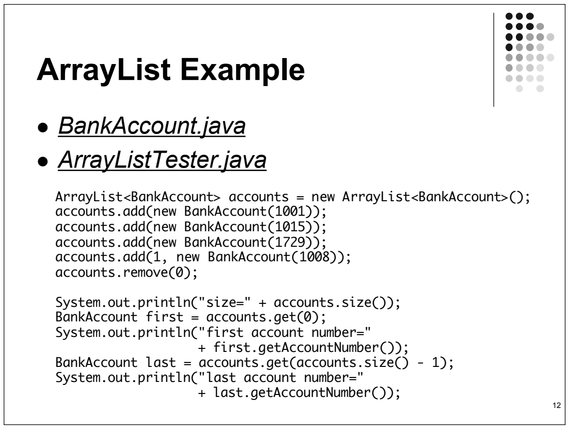
- `ArrayList<BankAccount>` declares an array list of bank accounts
  - Angle brackets indicate `BankAccount` is a *type parameter* – can use any other class name there instead
- `ArrayList` class is a *generic* class: `ArrayList<T>` collects objects of type `T`
  - Cannot use primitive types as type parameters – no `ArrayList<int>`



ArrayList Methods

- `add(a)`
  - adds new object to the end of the array list
- `add(i, a)`
  - adds object a at position i (shifts up all other elements after position i)
- `get(i)`
  - returns element at the i'th index (starts at 0)
- `remove(i)`
  - removes element at position i (shifts down elements after the removed one)
- `size()`
  - returns current size of the array list (initially 0)

See Quality Tip 8.1 about untyped array lists



ArrayList Example

- *BankAccount.java*
- *ArrayListTester.java*

```
ArrayList<BankAccount> accounts = new ArrayList<BankAccount>();
accounts.add(new BankAccount(1001));
accounts.add(new BankAccount(1015));
accounts.add(new BankAccount(1729));
accounts.add(1, new BankAccount(1008));
accounts.remove(0);

System.out.println("size=" + accounts.size());
BankAccount first = accounts.get(0);
System.out.println("first account number="
                  + first.getAccountNumber());
BankAccount last = accounts.get(accounts.size() - 1);
System.out.println("last account number="
                  + last.getAccountNumber());
```

Length and Size

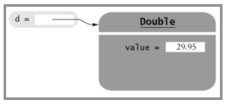
- Java has inconsistent syntax for determining length/size of strings, arrays, array lists:

Array	–	`a.length`
Array list	–	`a.size()`
String	–	`a.length()`

Primitive Types and Objects

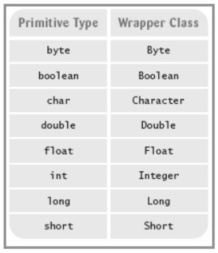
- Cannot directly store primitive types (int, double, char) in array lists
  - Must first 'wrap' them up into objects

```
ArrayList<Double> data = new ArrayList<Double>();
data.add( new Double(29.95) );
double x = data.get(0).doubleValue();
```



`Double d = new Double( 29.95 );`

Wrapper Classes



Primitive Type	Wrapper Class
byte	Byte
boolean	Boolean
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

- Notice differences in names
- Wrapper objects can be used anywhere objects are required instead of primitive values

Auto-boxing

- Better name: 'auto-wrapping' – only in Java 5.0
- Automatic conversion between primitive types and corresponding wrapper classes

```
Double d = 29.95; // same as Double d = new Double(29.95);
double x = d;     // same as double x = d.doubleValue();
Double e = d + 1;
```

- Last statement means:
  - auto-unbox `d` into a `double`
  - add `1`
  - auto-box the result into a new `Double`
  - store a reference to the newly created wrapper object in `e`

Arrays vs Array Lists

- Arrays
  - Pros: Efficient (less space, faster access), built-in Java construct - supports primitive types and objects, multi-dimensional arrays
  - Cons: Fixed size, no operations besides index access
- Array Lists
  - Pros: Resizable (automatically), provides add/insert/remove operations
  - Cons: Only stores objects, less efficient (especially when using wrapper objects for primitive types), syntax little more cluttered

Enhanced `for` Loop

  - Only available from Java version 5.0 onward
- Shortcut for iterating through sequence of elements from beginning to end

Enhanced version

```
double[] data = . . .;
double sum = 0;
for (double e : data) {
   sum = sum + e;
}
```

Read as '*for each e in data*'

Traditional version

```
double[] data = . . .;
double sum = 0;
for (int i = 0; i < data.length; i++) {
   double e = data[i];
   sum = sum + e;
}
```

Enhanced `for` with Array Lists

```
ArrayList<BankAccount> accounts = . . . ;

double sum = 0;
for (BankAccount a : accounts) {
   sum = sum + a.getBalance();
}



double sum = 0;
for (int i = 0; i < accounts.size(); i++) {
   BankAccount a = accounts.get(i);
   sum = sum + a.getBalance();
}
```

19

Syntax: ‘for each’ loop

```
for ( Type variable : collection )
   statement
```

Purpose:
To execute a loop for each element in the collection. In each iteration, the variable is assigned the next element of the collection. Then the statement is executed.

- The ‘for each’ construct has very specific purpose. If you don’t want to start at the beginning of the collection, or need to traverse the collection in reverse order, use a regular for loop

20

Simple Array Algorithms: Counting Matches

- Check all elements and count the matches until you reach the end of the array list.

```
public class Bank
 {
   . . .
   public int count(double atLeast) {





   }
   . . .
   private ArrayList<BankAccount> accounts;
}
```

21

Simple Array Algorithms: Finding a Value

- Check all elements until you have found a match (return `null` if no match found)

```
public class Bank
 {
   . . .
   public BankAccount find(int accountNumber) {





   }
   . . .
   private ArrayList<BankAccount> accounts;
}
```

22

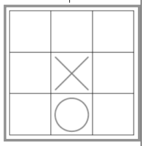
Simple Array Algorithms: Finding a Maximum/Minimum

- Initialize a candidate with the starting element
- Compare candidate with remaining elements
- Update it if you find a larger or smaller value
- (Return null if the collection is empty)

- *Bank.java*
- *BankTester.java*

23

Two-Dimensional Arrays



- Example: Tic-Tac-Toe board
  - Rows and columns of values make up a 2D array or *matrix*
- Access elements with index pair `a[i][j]`
- Construct by specify dimensions

```
final int ROWS = 3;
final int COLUMNS = 3;
String[][] board = new String[ROWS][COLUMNS];
```

- Results in a 2D array with 9 elements:

```
board[0][0]   board[0][1]   board[0][2]
board[1][0]   board[1][1]   board[1][2]
board[2][0]   board[2][1]   board[2][2]
```

24

Tic-Tac-Toe Program

- *TicTacToe.java*
- *TicTacToeTester.java*

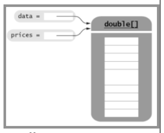
```
/** Sets a field in the board. The field must be unoccupied.
   @param i the row index
   @param j the column index
   @param player the player ("x" or "o")
*/
public void set(int i, int j, String player)

/** Creates a string representation of the board, such as
   |x  o|
   |  x |
   |   o|
   @return the string representation
*/
public String toString()
```

Copying Arrays

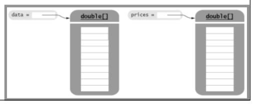
- Copying an array variable results in a second reference to the *same array*

```
double[] data = new double[10];
// fill array . . .
double[] prices = data;
```



- Use `clone` to make a copy of the *elements*

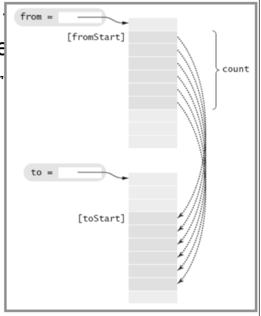
```
double[] prices = (double[]) data.clone();
```



`System.arraycopy` Method

- Use `System.arraycopy` t
  from one array to anothe

```
System.arraycopy( from, fromStar
```



Using `System.arraycopy`

- Insert element in an array

```
System.arraycopy(data, i, data, i + 1, data.length - i - 1);
data[i] = x;
```

- Remove element from an array

```
System.arraycopy(data, i + 1, data, i, data.length - i - 1);
```

- Grow an array that is out of space

```
   // 1. create a new, larger array
double[] newData = new double[ 2 * data.length ];
   // 2. copy all elements into the new array
System.arraycopy( data, 0, newData, 0, data.length );
   // 3. store reference to the new array in the array variable
data = newData;
```

Partially Filled Arrays

- Suppose you need to input a set of numbers from the user – may be between 10 and 100 numbers
  - Allocate an array of the maximum size
  - Keep a companion variable to tell how many elements of the array are actually being used

    ```
    final int DATA_LENGTH = 100;
    double[] data = new double[DATA_LENGTH];
    int dataSize = 0;
    ```

  - Update size variable as elements are added

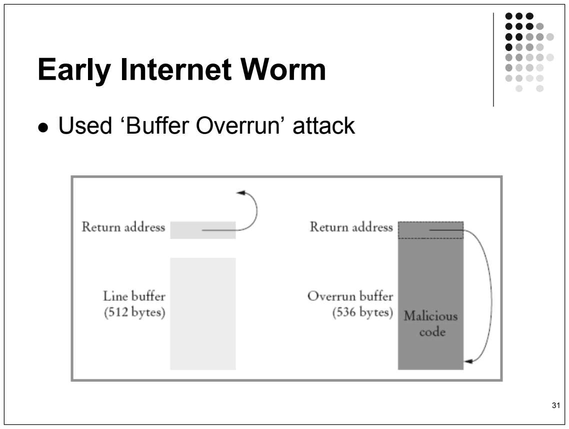
    ```
    data[ dataSize++ ] = x;
    ```

- *Note:* The array list class uses techniques on this slide and previous slide behind the scenes

Methods with Variable Number of Parameters

- Feature added in Java 5.0
- Parameters passed as an array of values

- See Advanced Topic 8.5 (page 309)



Early Internet Worm

- Used ‘Buffer Overrun’ attack