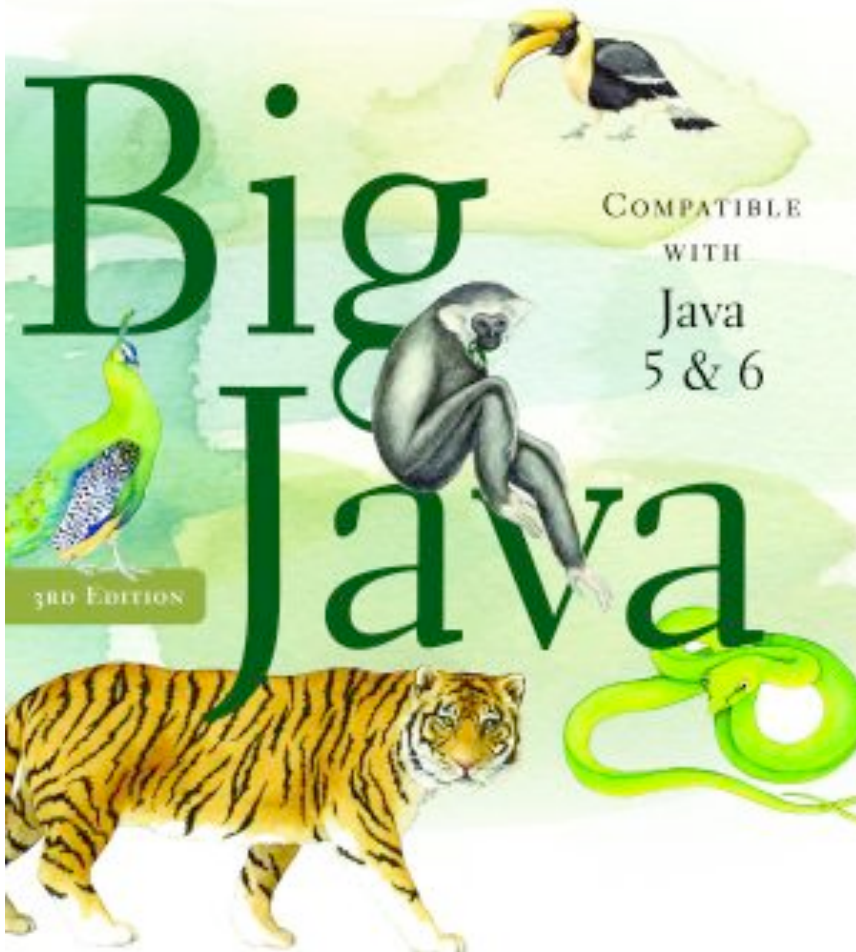


ICOM 4015: Advanced Programming

Lecture 7

Chapter Seven: Arrays and Array Lists

CAY HORSTMANN



Chapter Seven: Arrays and Array Lists

Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Chapter Goals

- To become familiar with using arrays and array lists
- To learn about wrapper classes, auto-boxing and the generalized for loop
- To study common array algorithms
- To learn how to use two-dimensional arrays
- To understand when to choose array lists and arrays in your programs
- To implement partially filled arrays
- To understand the concept of regression testing

Arrays

- Array: Sequence of values of the same type
- Construct array:
`new double[10]`
- Store in variable of type `double[]`
`double[] data = new double[10];`
- When array is created, all values are initialized depending on array type:
 - Numbers: `0`
 - Boolean: `false`
 - Object References: `null`

Arrays

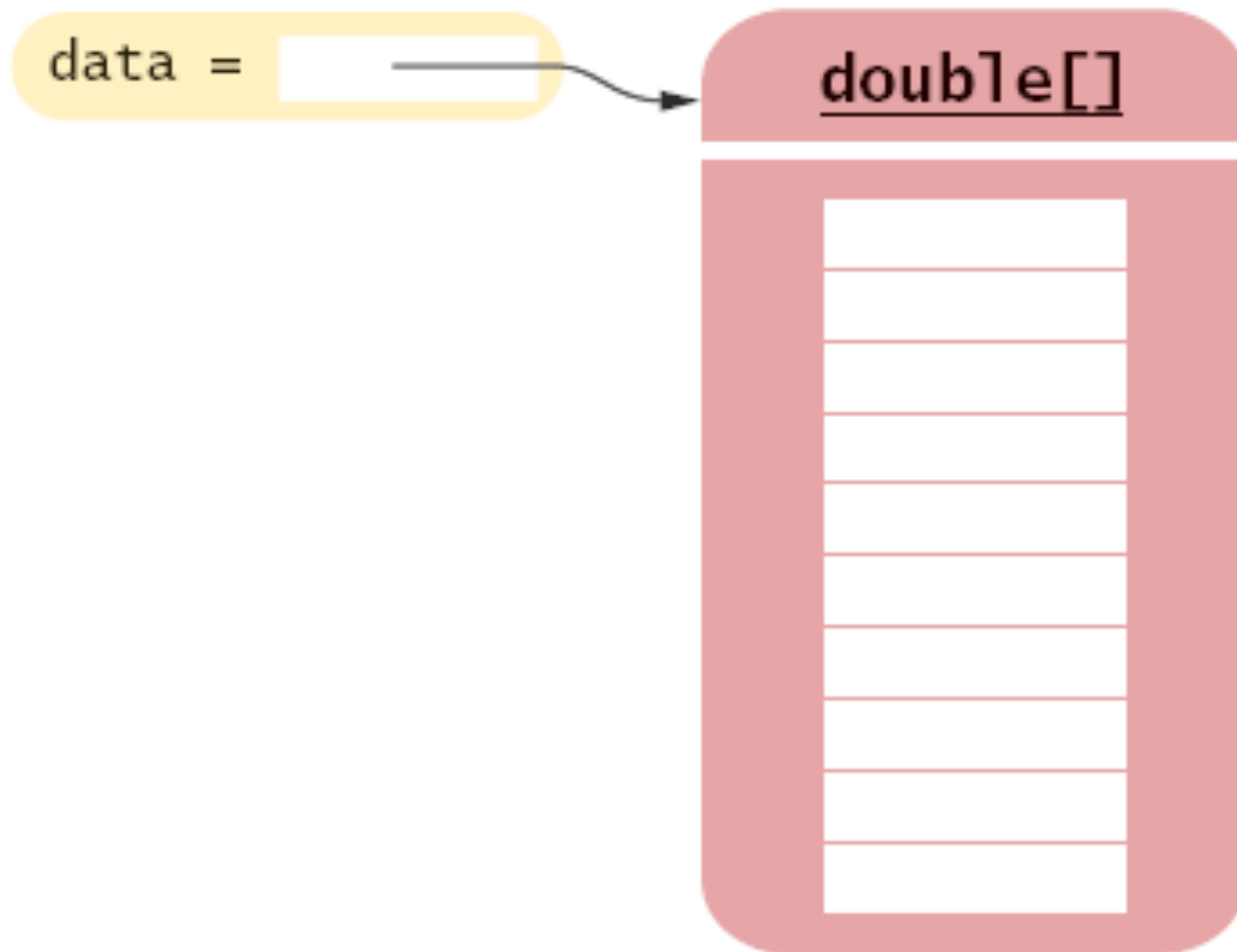


Figure 1 An Array Reference and an Array

Arrays

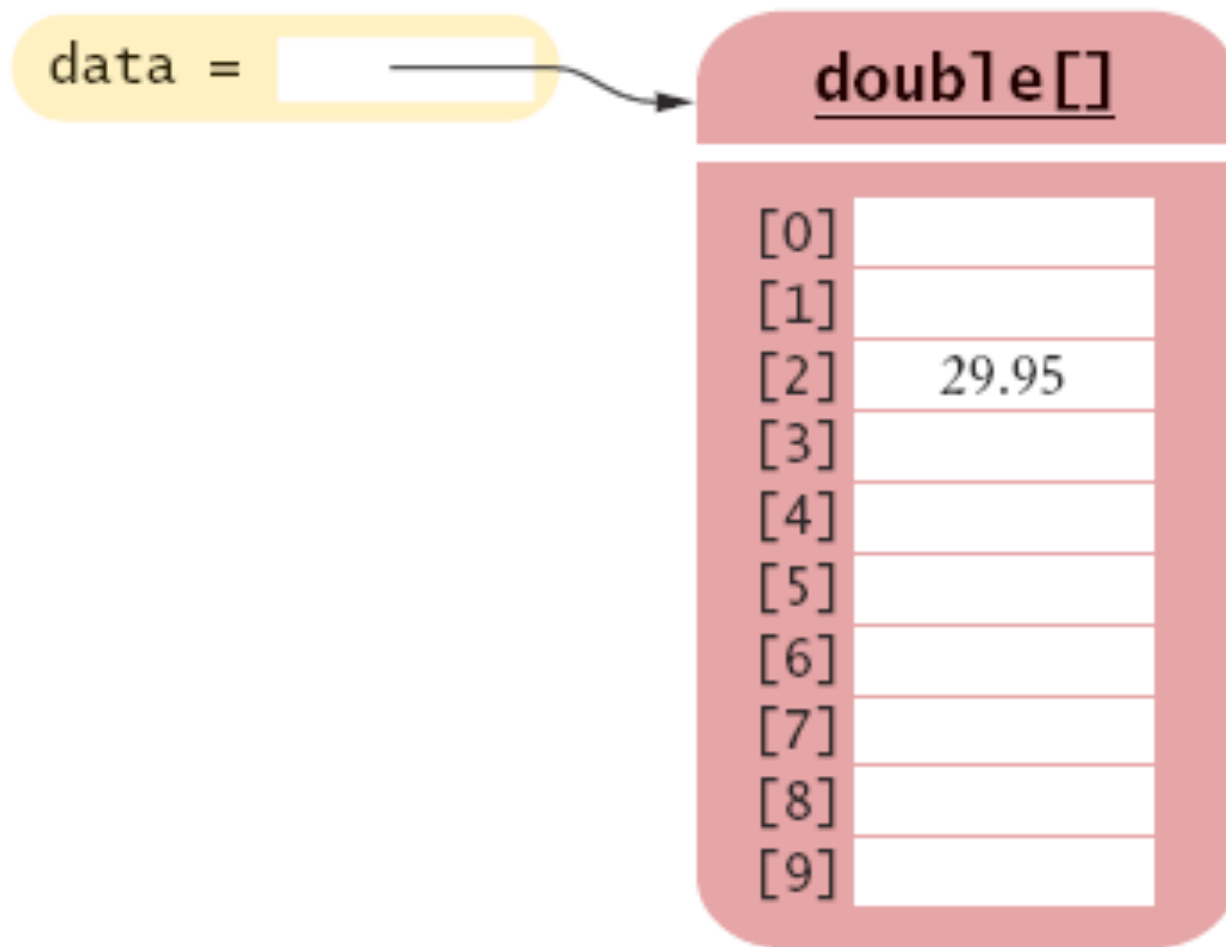


Figure 2 Storing a Value in an Array

Arrays

- Using the value stored:

```
System.out.println("The value of this data item is "  
    + data[4]);
```

- Get array length as `data.length` (Not a method!)
- Index values range from 0 to `length - 1`
- Accessing a nonexistent element results in a bounds error

```
double[] data = new double[10];  
data[10] = 29.95; // ERROR
```
- Limitation: Arrays have fixed length

Syntax 7.1 Array Construction

```
new typeName[length]
```

Example:

```
new double[10]
```

Purpose:

To construct an array with a given number of elements.

Syntax 7.2 Array Element Access

arrayReference[*index*]

Example:

`data[2]`

Purpose:

To access an element in an array.

Self Check 7.1

What elements does the data array contain after the following statements?

```
double[] data = new double[10];  
for (int i = 0; i < data.length; i++) data[i] = i * i;
```

Answer: 0, 1, 4, 9, 16, 25, 36, 49, 64, 81, but not 100

Self Check 7.2

What do the following program segments print? Or, if there is an error, describe the error and specify whether it is detected at compile-time or at run-time.

a) `double[] a = new double[10];`
`System.out.println(a[0]);`

b) `double[] b = new double[10];`
`System.out.println(b[10]);`

c) `double[] c;`
`System.out.println(c[0]);`

Answer:

a) *0*

b) *a run-time error: array index out of bounds*

c) *a compile-time error: c is not initialized*

Array Lists

- The `ArrayList` class manages a sequence of objects
- Can grow and shrink as needed
- `ArrayList` class supplies methods for many common tasks, such as inserting and removing elements
- The `ArrayList` class is a generic class: `ArrayList<T>` collects objects of type `T`:

```
ArrayList<BankAccount> accounts = new
    ArrayList<BankAccount> ();
accounts.add(new BankAccount (1001));
accounts.add(new BankAccount (1015));
accounts.add(new BankAccount (1022));
```

- `size` method yields number of elements

Retrieving Array List Elements

- Use `get` method
- Index starts at 0
- `BankAccount anAccount = accounts.get(2); // gets the third element of the array list`
- Bounds error if index is out of range
- Most common bounds error:

```
int i = accounts.size();
anAccount = accounts.get(i); // Error
//legal index values are 0. . .i-1
```

Adding Elements

- **set overwrites an existing value**

```
BankAccount anAccount = new BankAccount(1729);  
accounts.set(2, anAccount);
```

- **add adds a new value before the index**

```
accounts.add(i, a)
```

Continued

Adding Elements (cont.)

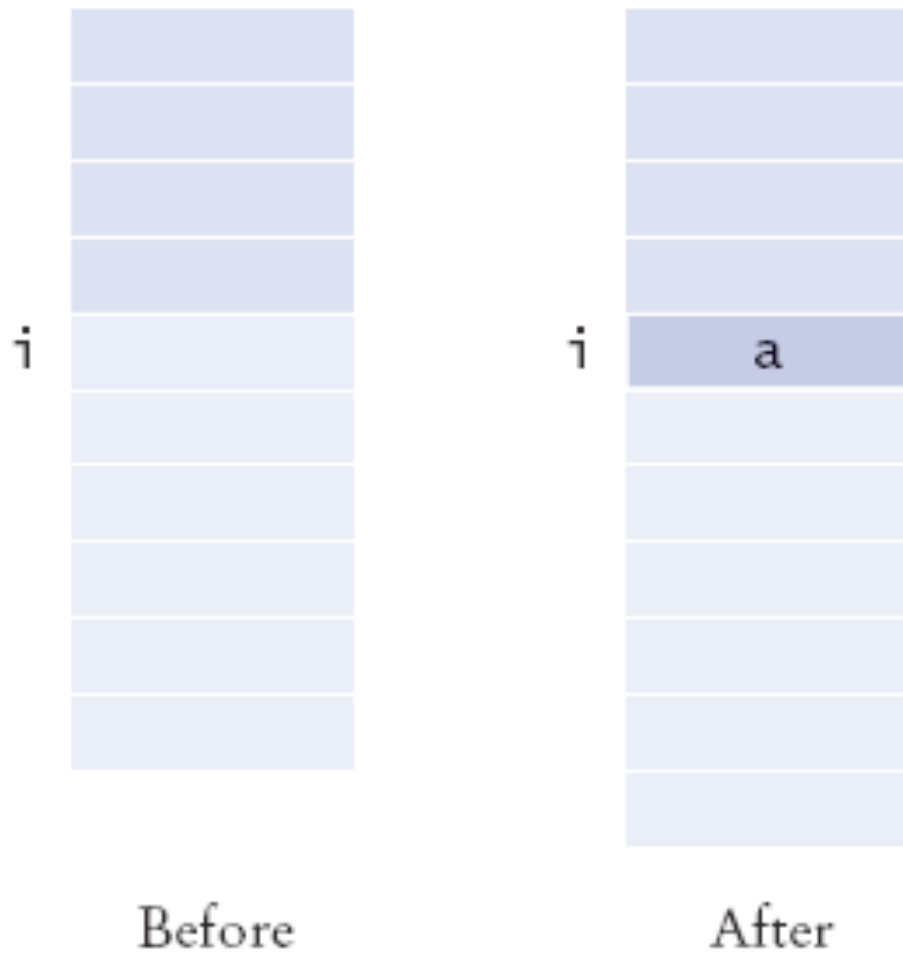


Figure 3 Adding an Element in the Middle of an Array List

Animation 7.1 –

[0]	
[1]	
[2]	
[3]	
[4]	
[5]	
[6]	

This animation demonstrates inserting an element into an array list.

7-01 Inserting into an Array List



Removing Elements

`remove` removes an element at an index

`accounts.remove(i)`

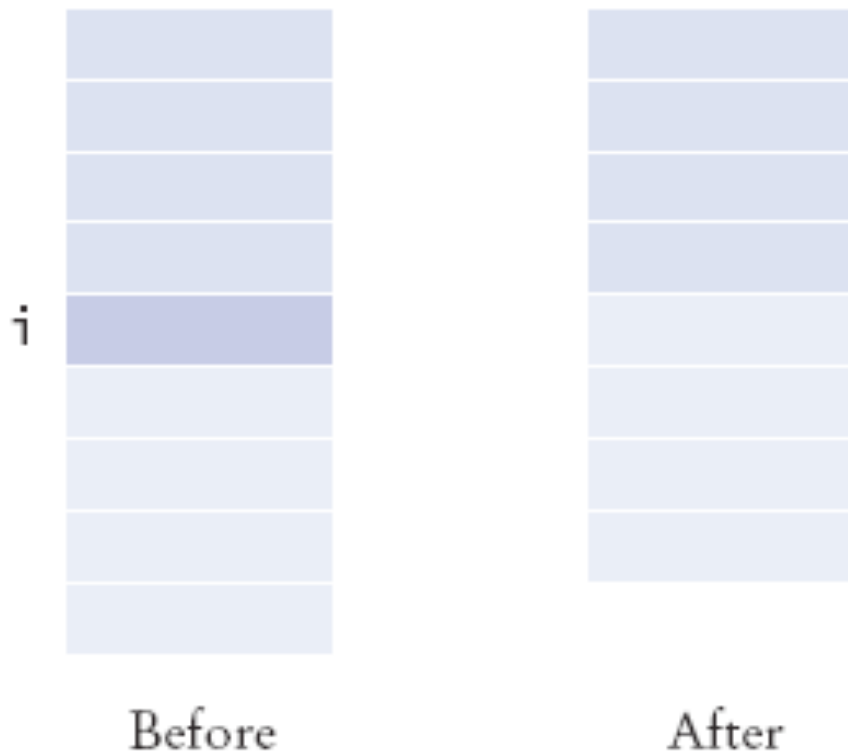


Figure 4 Removing an Element from the Middle of an Array List

Animation 7.2 –

[0]	
[1]	
[2]	
[3]	
[4]	
[5]	
[6]	
[7]	

This animation demonstrates removing an element from an array list.

7-02 Removing from an Array List



ch07/arraylist/ArrayListTester.java

```
01: import java.util.ArrayList;
02:
03: /**
04:     This program tests the ArrayList class.
05: */
06: public class ArrayListTester
07: {
08:     public static void main(String[] args)
09:     {
10:         ArrayList<BankAccount> accounts
11:             = new ArrayList<BankAccount>();
12:         accounts.add(new BankAccount(1001));
13:         accounts.add(new BankAccount(1015));
14:         accounts.add(new BankAccount(1729));
15:         accounts.add(1, new BankAccount(1008));
16:         accounts.remove(0);
17:
18:         System.out.println("Size: " + accounts.size());
19:         System.out.println("Expected: 3");
20:         BankAccount first = accounts.get(0);
```

Continued

ch07/arraylist/ArrayListTester.java (cont.)

```
21:         System.out.println("First account number: "
22:             + first.getAccountNumber());
23:         System.out.println("Expected: 1015");
24:         BankAccount last = accounts.get(accounts.size() - 1);
25:         System.out.println("Last account number: "
26:             + last.getAccountNumber());
27:         System.out.println("Expected: 1729");
28:     }
29: }
```

ch07/arraylist/BankAccount.java

```
01: /**
02:     A bank account has a balance that can be changed by
03:     deposits and withdrawals.
04: */
05: public class BankAccount
06: {
07:     /**
08:         Constructs a bank account with a zero balance
09:         @param anAccountNumber the account number for this account
10:     */
11:     public BankAccount(int anAccountNumber)
12:     {
13:         accountNumber = anAccountNumber;
14:         balance = 0;
15:     }
16:
17:     /**
18:         Constructs a bank account with a given balance
19:         @param anAccountNumber the account number for this account
20:         @param initialBalance the initial balance
21:     */
```

Continued

ch07/arraylist/BankAccount.java (cont.)

```
22:     public BankAccount(int anAccountNumber, double initialBalance)
23:     {
24:         accountNumber = anAccountNumber;
25:         balance = initialBalance;
26:     }
27:
28:     /**
29:      * Gets the account number of this bank account.
30:      * @return the account number
31:      */
32:     public int getAccountNumber()
33:     {
34:         return accountNumber;
35:     }
36:
37:     /**
38:      * Deposits money into the bank account.
39:      * @param amount the amount to deposit
40:      */
41:     public void deposit(double amount)
42:     {
43:         double newBalance = balance + amount;
44:         balance = newBalance;
45:     }
```

Continued

Big Java by Cay Horstmann

Copyright © 2008 by John Wiley & Sons. All rights reserved.

ch07/arraylist/BankAccount.java (cont.)

```
46:
47:     /**
48:         Withdraws money from the bank account.
49:         @param amount the amount to withdraw
50:     */
51:     public void withdraw(double amount)
52:     {
53:         double newBalance = balance - amount;
54:         balance = newBalance;
55:     }
56:
57:     /**
58:         Gets the current balance of the bank account.
59:         @return the current balance
60:     */
61:     public double getBalance()
62:     {
63:         return balance;
64:     }
65:
66:     private int accountNumber;
67:     private double balance;
68: }
```

Continued

ch07/arraylist/BankAccount.java (cont.)

Output:

Size: 3

Expected: 3

First account number: 1008

Expected: 1008

Last account number: 1729

Expected: 1729

Self Check 7.3

How do you construct an array of 10 strings? An array list of strings?

Answer:

```
new String[10];  
new ArrayList<String>();
```

Self Check 7.4

What is the content of `names` after the following statements?

```
ArrayList<String> names = new ArrayList<String>();  
names.add("A");  
names.add(0, "B");  
names.add("C");  
names.remove(1);
```

Answer: `names` contains the strings "B" and "C" at positions 0 and 1

Wrappers

- You cannot insert primitive types directly into array lists
- To treat primitive type values as objects, you must use wrapper classes:

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

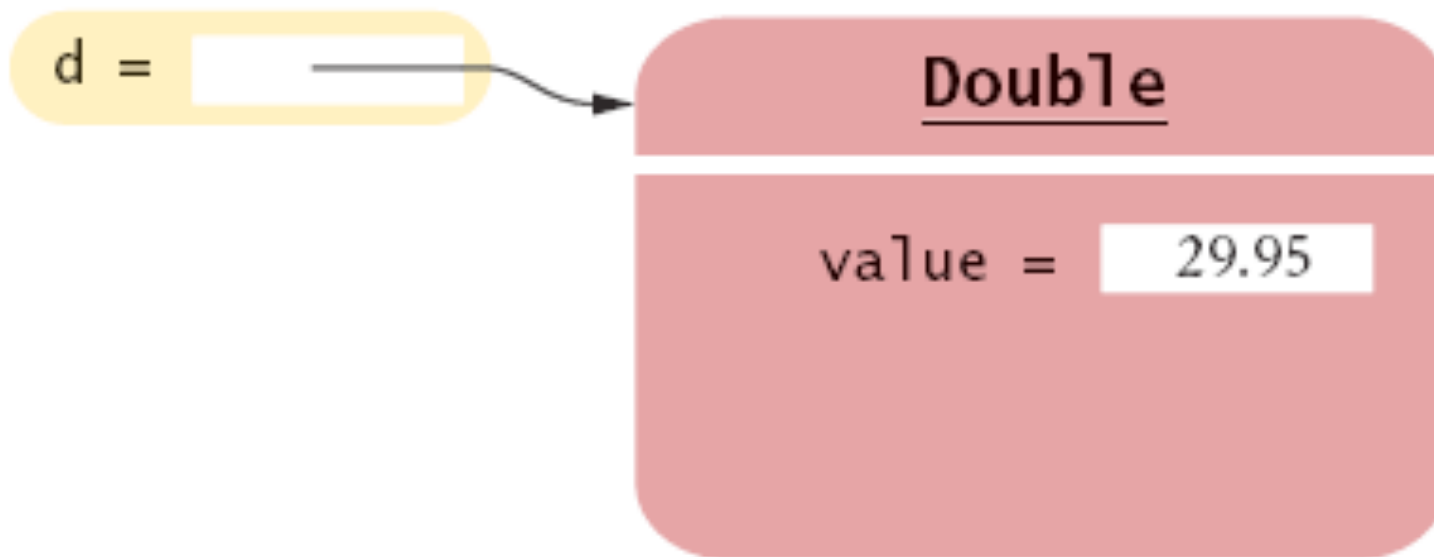


Figure 5 An Object of a Wrapper Class

Wrappers

There are wrapper classes for all eight primitive types:

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

Auto-boxing

- Auto-boxing: Starting with Java 5.0, conversion between primitive types and the corresponding wrapper classes is automatic.

```
Double d = 29.95; // auto-boxing; same as Double d =  
    new Double(29.95);  
double x = d; // auto-unboxing; same as double x =  
    d.doubleValue();
```

- Auto-boxing even works inside arithmetic expressions

```
Double e = d + 1;
```

- 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*

Self Check 7.5

What is the difference between the types `double` and `Double`?

Answer: `double` is one of the eight primitive types. `Double` is a class type.

Self Check 7.6

Suppose `data` is an `ArrayList<Double>` of size > 0 . How do you increment the element with index 0?

Answer: `data.set(0, data.get(0) + 1);`

The Generalized `for` Loop

- Traverses all elements of a collection:

```
double[] data = . . .;
double sum = 0;
for (double e : data) // You should read this loop as
    "for each e in data"
{
    sum = sum + e;
}
```

- Traditional alternative:

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


The Generalized `for` Loop

- Works for `ArrayLists` too:

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

- Equivalent to the following ordinary `for` loop:

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

Syntax 7.3 The "for each" Loop

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

Example:

```
for (double e : data)  
    sum = sum + e;
```

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.

Self Check 7.7

Write a "for each" loop that prints all elements in the array `data`.

Answer:

```
for (double x : data) System.out.println(x);
```

Self Check 7.8

Why is the "for each" loop not an appropriate shortcut for the following ordinary `for` loop?

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

Answer: The loop writes a value into `data[i]`. The "for each" loop does not have the index variable `i`.

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)
    {
        int matches = 0;
        for (BankAccount a : accounts)
        {
            if (a.getBalance() >= atLeast) matches++;
            // Found a match
        }
        return matches;
    }
    . . .
    private ArrayList<BankAccount> accounts;
}
```

Simple Array Algorithms: Finding a Value

Check all elements until you have found a match.

```
public class Bank
{
    public BankAccount find(int accountNumber)
    {
        for (BankAccount a : accounts)
        {
            if (a.getAccountNumber() == accountNumber)
                // Found a match return a;
        }
        return null; // No match in the entire array list
    }
    . . .
}
```

Simple Array Algorithms: Finding the Maximum or Minimum

- Initialize a candidate with the starting element
- Compare candidate with remaining elements
- Update it if you find a larger or smaller value
- Example:

```
BankAccount largestYet = accounts.get(0);
for (int i = 1; i < accounts.size(); i++)
{
    BankAccount a = accounts.get(i);
    if (a.getBalance() > largestYet.getBalance())
        largestYet = a;
}
return largestYet;
```

- Works only if there is at least one element in the array list . . .

Continued

Simple Array Algorithms: Finding the Maximum or Minimum (cont.)

- If list is empty, return `null`:

```
if (accounts.size() == 0) return null;
BankAccount largestYet = accounts.get(0);
...
```


ch07/bank/Bank.java

```
01: import java.util.ArrayList;
02:
03: /**
04:     This bank contains a collection of bank accounts.
05: */
06: public class Bank
07: {
08:     /**
09:         Constructs a bank with no bank accounts.
10:     */
11:     public Bank()
12:     {
13:         accounts = new ArrayList<BankAccount>();
14:     }
15:
16:     /**
17:         Adds an account to this bank.
18:         @param a the account to add
19:     */
20:     public void addAccount(BankAccount a)
21:     {
22:         accounts.add(a);
23:     }
```

Continued

ch07/bank/Bank.java (cont.)

```
24:
25:     /**
26:         Gets the sum of the balances of all accounts in this bank.
27:         @return the sum of the balances
28:     */
29:     public double getTotalBalance()
30:     {
31:         double total = 0;
32:         for (BankAccount a : accounts)
33:         {
34:             total = total + a.getBalance();
35:         }
36:         return total;
37:     }
38:
39:     /**
40:         Counts the number of bank accounts whose balance is at
41:         least a given value.
42:         @param atLeast the balance required to count an account
43:         @return the number of accounts having least the given balance
44:     */
45:     public int count(double atLeast)
46:     {
```

Continued

Big Java by Cay Horstmann

Copyright © 2008 by John Wiley & Sons. All rights reserved.

ch07/bank/Bank.java (cont.)

```
47:         int matches = 0;
48:         for (BankAccount a : accounts)
49:         {
50:             if (a.getBalance() >= atLeast) matches++; // Found a match
51:         }
52:         return matches;
53:     }
54:
55:     /**
56:      Finds a bank account with a given number.
57:      @param accountNumber the number to find
58:      @return the account with the given number, or null if there
59:      is no such account
60:     */
61:     public BankAccount find(int accountNumber)
62:     {
63:         for (BankAccount a : accounts)
64:         {
65:             if (a.getAccountNumber() == accountNumber) // Found a match
66:                 return a;
67:         }
68:         return null; // No match in the entire array list Continued
69:     }
70:
```

ch07/bank/Bank.java (cont.)

```
71:     /**
72:         Gets the bank account with the largest balance.
73:         @return the account with the largest balance, or null if the
74:         bank has no accounts
75:     */
76:     public BankAccount getMaximum()
77:     {
78:         if (accounts.size() == 0) return null;
79:         BankAccount largestYet = accounts.get(0);
80:         for (int i = 1; i < accounts.size(); i++)
81:         {
82:             BankAccount a = accounts.get(i);
83:             if (a.getBalance() > largestYet.getBalance())
84:                 largestYet = a;
85:         }
86:         return largestYet;
87:     }
88:
89:     private ArrayList<BankAccount> accounts;
90: }
```

ch07/bankBankTester.java

```
01: /**
02:     This program tests the Bank class.
03: */
04: public class BankTester
05: {
06:     public static void main(String[] args)
07:     {
08:         Bank firstBankOfJava = new Bank();
09:         firstBankOfJava.addAccount(new BankAccount(1001, 20000));
10:         firstBankOfJava.addAccount(new BankAccount(1015, 10000));
11:         firstBankOfJava.addAccount(new BankAccount(1729, 15000));
12:
13:         double threshold = 15000;
14:         int c = firstBankOfJava.count(threshold);
15:         System.out.println("Count: " + c);
16:         System.out.println("Expected: 2");
17:
18:         int accountNumber = 1015;
19:         BankAccount a = firstBankOfJava.find(accountNumber);
20:         if (a == null)
```

Continued

ch07/bankBankTester.java (cont.)

```
21:         System.out.println("No matching account");
22:     else
23:         System.out.println("Balance of matching account: " +
                a.getBalance());
24:     System.out.println("Expected: 10000");
25:
26:     BankAccount max = firstBankOfJava.getMaximum();
27:     System.out.println("Account with largest balance: "
28:         + max.getAccountNumber());
29:     System.out.println("Expected: 1001");
30: }
31: }
```

Output:

Count: 2

Expected: 2

Balance of matching account: 10000.0

Expected: 10000

Account with largest balance: 1001

Expected: 1001

Self Check 7.9

What does the `find` method do if there are two bank accounts with a matching account number?

Answer: It returns the first match that it finds.

Self Check 7.10

Would it be possible to use a "for each" loop in the `getMaximum` method?

Answer: Yes, but the first comparison would always fail.

Two-Dimensional Arrays

- When constructing a two-dimensional array, you specify how many rows and columns you need:

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

- You access elements with an index pair `a[i][j]`

```
board[i][j] = "x";
```

A Tic-Tac-Toe Board

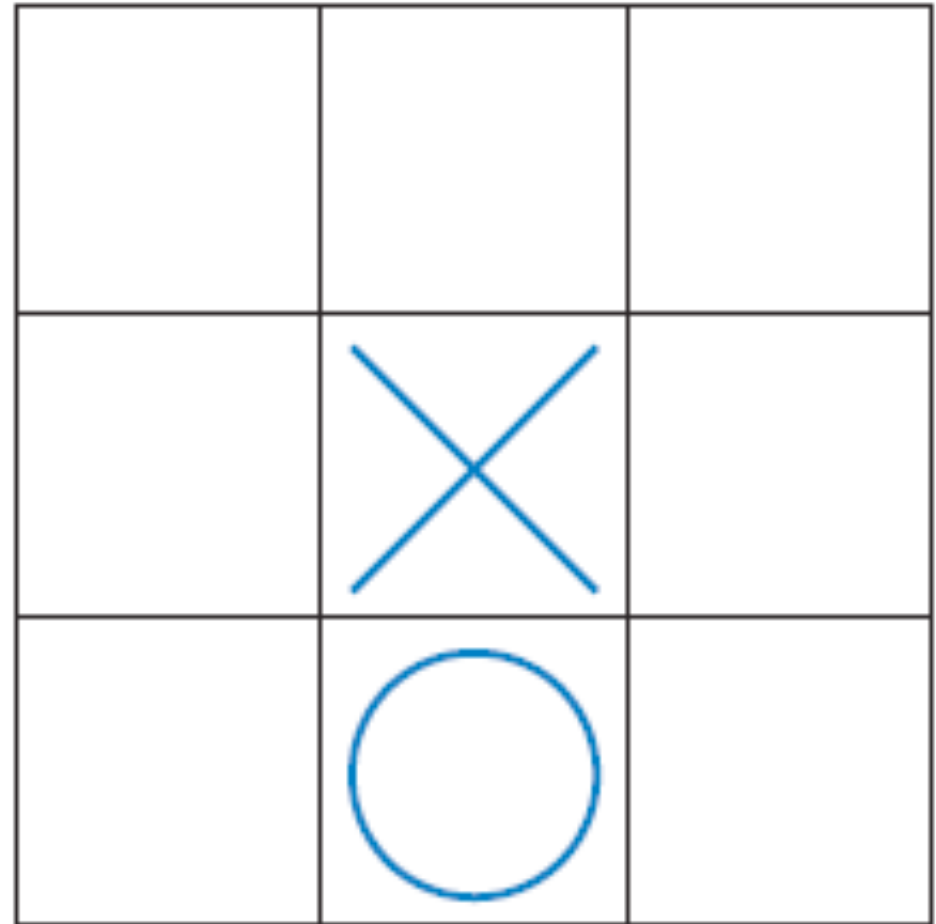


Figure 6

A Tic-Tac-Toe Board

Traversing Two-Dimensional Arrays

It is common to use two nested loops when filling or searching:

```
for (int i = 0; i < ROWS; i++)  
    for (int j = 0; j < COLUMNS; j++)  
        board[i][j] = " ";
```

ch07/twodim/TicTacToe.java

```
01: /**
02:     A 3 x 3 tic-tac-toe board.
03: */
04: public class TicTacToe
05: {
06:     /**
07:         Constructs an empty board.
08:     */
09:     public TicTacToe()
10:     {
11:         board = new String[ROWS][COLUMNS];
12:         // Fill with spaces
13:         for (int i = 0; i < ROWS; i++)
14:             for (int j = 0; j < COLUMNS; j++)
15:                 board[i][j] = " ";
16:     }
17:
18:     /**
19:         Sets a field in the board. The field must be unoccupied.
20:         @param i the row index
21:         @param j the column index
22:         @param player the player ("x" or "o")
23:     */
```

Continued

Big Java by Cay Horstmann

Copyright © 2008 by John Wiley & Sons. All rights reserved.

ch07/twodim/TicTacToe.java (cont.)

```
24:     public void set(int i, int j, String player)
25:     {
26:         if (board[i][j].equals(" "))
27:             board[i][j] = player;
28:     }
29:
30:     /**
31:      * Creates a string representation of the board, such as
32:      * |x  o|
33:      * | x |
34:      * |  o|
35:      * @return the string representation
36:      */
37:     public String toString()
38:     {
39:         String r = "";
40:         for (int i = 0; i < ROWS; i++)
41:         {
42:             r = r + "|";
43:             for (int j = 0; j < COLUMNS; j++)
44:                 r = r + board[i][j];
45:             r = r + "|\n";
```

Continued

ch07/twodim/TicTacToe.java (cont.)

```
46:         }
47:         return r;
48:     }
49:
50:     private String[][] board;
51:     private static final int ROWS = 3;
52:     private static final int COLUMNS = 3;
53: }
```

ch07/twodim/TicTacToeRunner.java

```
01: import java.util.Scanner;
02:
03: /**
04:     This program runs a TicTacToe game. It prompts the
05:     user to set positions on the board and prints out the
06:     result.
07: */
08: public class TicTacToeRunner
09: {
10:     public static void main(String[] args)
11:     {
12:         Scanner in = new Scanner(System.in);
13:         String player = "x";
14:         TicTacToe game = new TicTacToe();
15:         boolean done = false;
16:         while (!done)
17:         {
18:             System.out.print(game.toString());
19:             System.out.print(
20:                 "Row for " + player + " (-1 to exit): ");
21:             int row = in.nextInt();
22:             if (row < 0) done = true;
23:             else
24:                 {
```

Continued

Big Java by Cay Horstmann

Copyright © 2008 by John Wiley & Sons. All rights reserved.

ch07/twodim/TicTacToeRunner.java (cont.)

```
25:         System.out.print("Column for " + player + ": ");
26:         int column = in.nextInt();
27:         game.set(row, column, player);
28:         if (player.equals("x"))
29:             player = "o";
30:         else
31:             player = "x";
32:     }
33: }
34: }
35: }
```


ch07/twodim/TicTacToeRunner.java (cont.)

Output:

```
| |  
| |  
| |  
Row for x (-1 to exit): 1  
Column for x: 2  
| |  
| x |  
| |  
Row for o (-1 to exit): 0  
Column for o: 0  
| o |  
| x |  
| |  
Row for x (-1 to exit): -1
```

Self Check 7.11

How do you declare and initialize a 4-by-4 array of integers?

Answer:

```
int[][] array = new int[4][4];
```

Self Check 7.12

How do you count the number of spaces in the tic-tac-toe board?

Answer:

```
int count = 0;
for (int i = 0; i < ROWS; i++)
    for (int j = 0; j < COLUMNS; j++)
        if (board[i][j] == ' ') count++;
```

Copying Arrays: Copying Array References

Copying an array variable yields a second reference to the same array

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

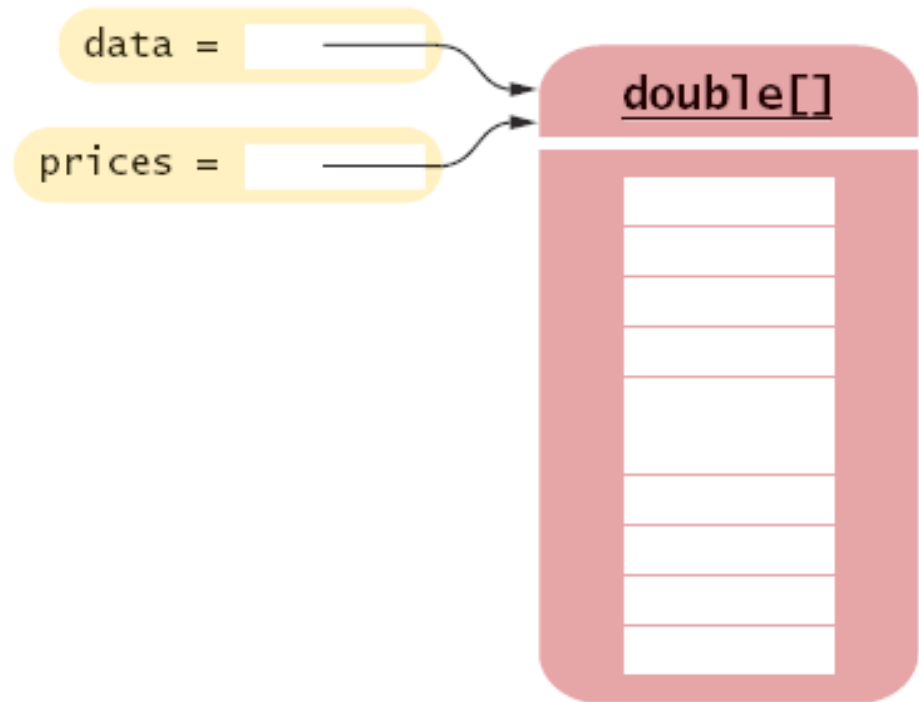


Figure 7 Two References to the Same Array
Big Java by Cay Horstmann
Copyright © 2008 by John Wiley & Sons. All rights reserved.

Copying Arrays: Cloning Arrays

Use `clone` to make true copy

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

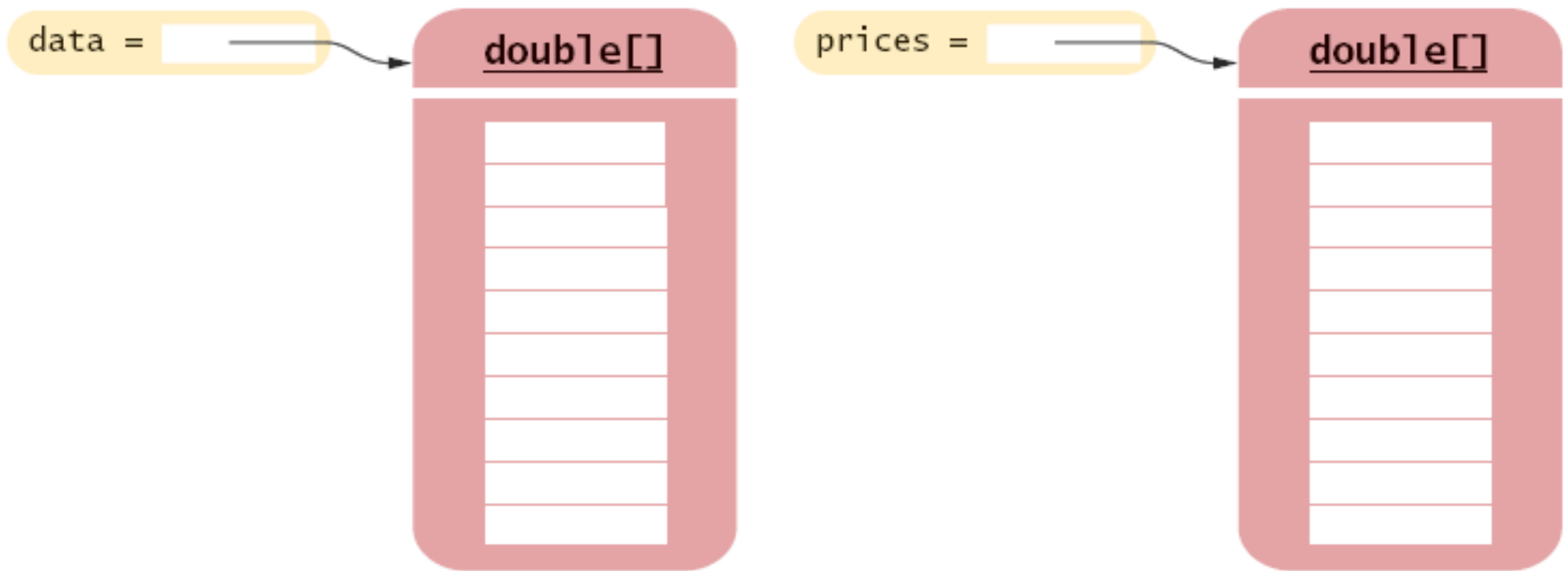


Figure 8 Cloning an Array

Copying Arrays: Copying Array Elements

```
System.arraycopy(from, fromStart, to, toStart, count);
```

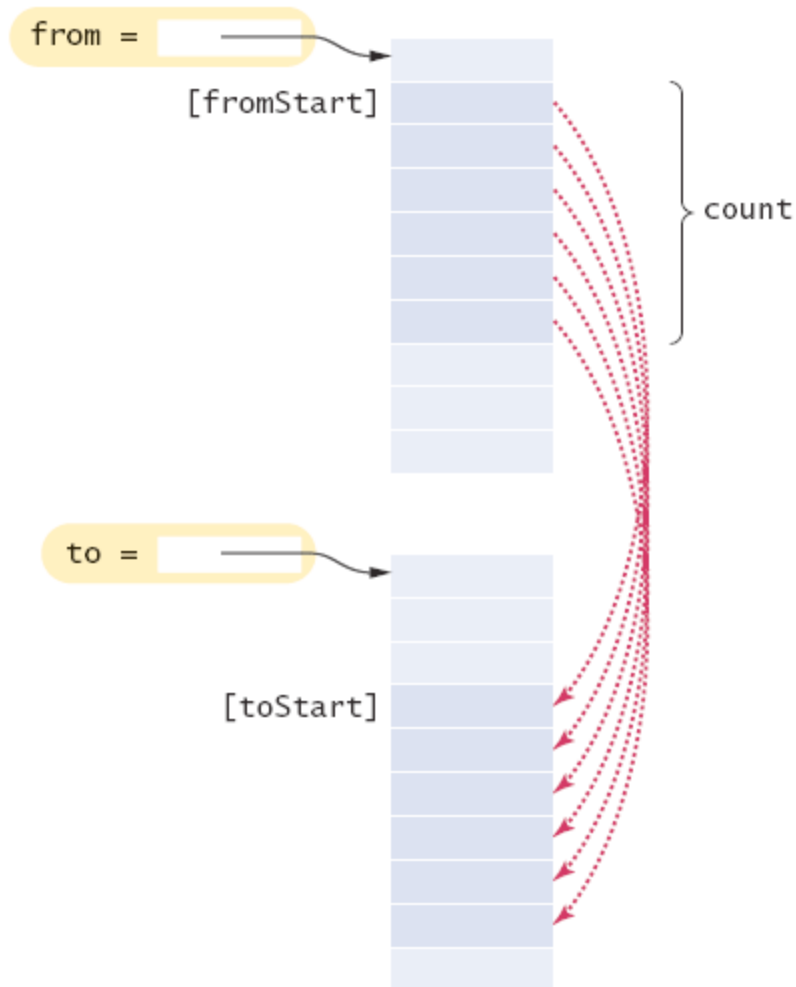


Figure 9 The `System.arraycopy` Method

Adding an Element to an Array

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

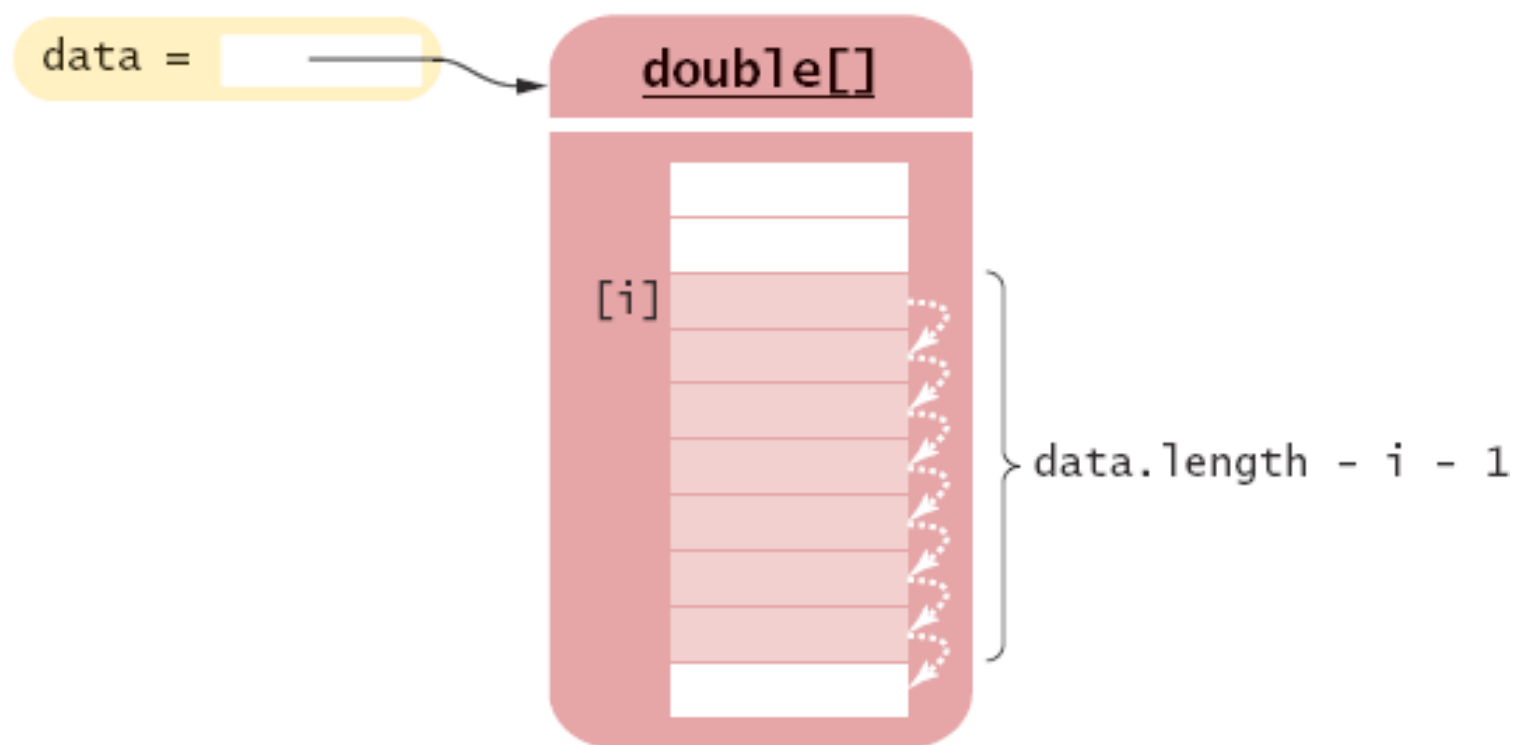


Figure 10 Inserting a New Element into an Array

Removing an Element from an Array

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

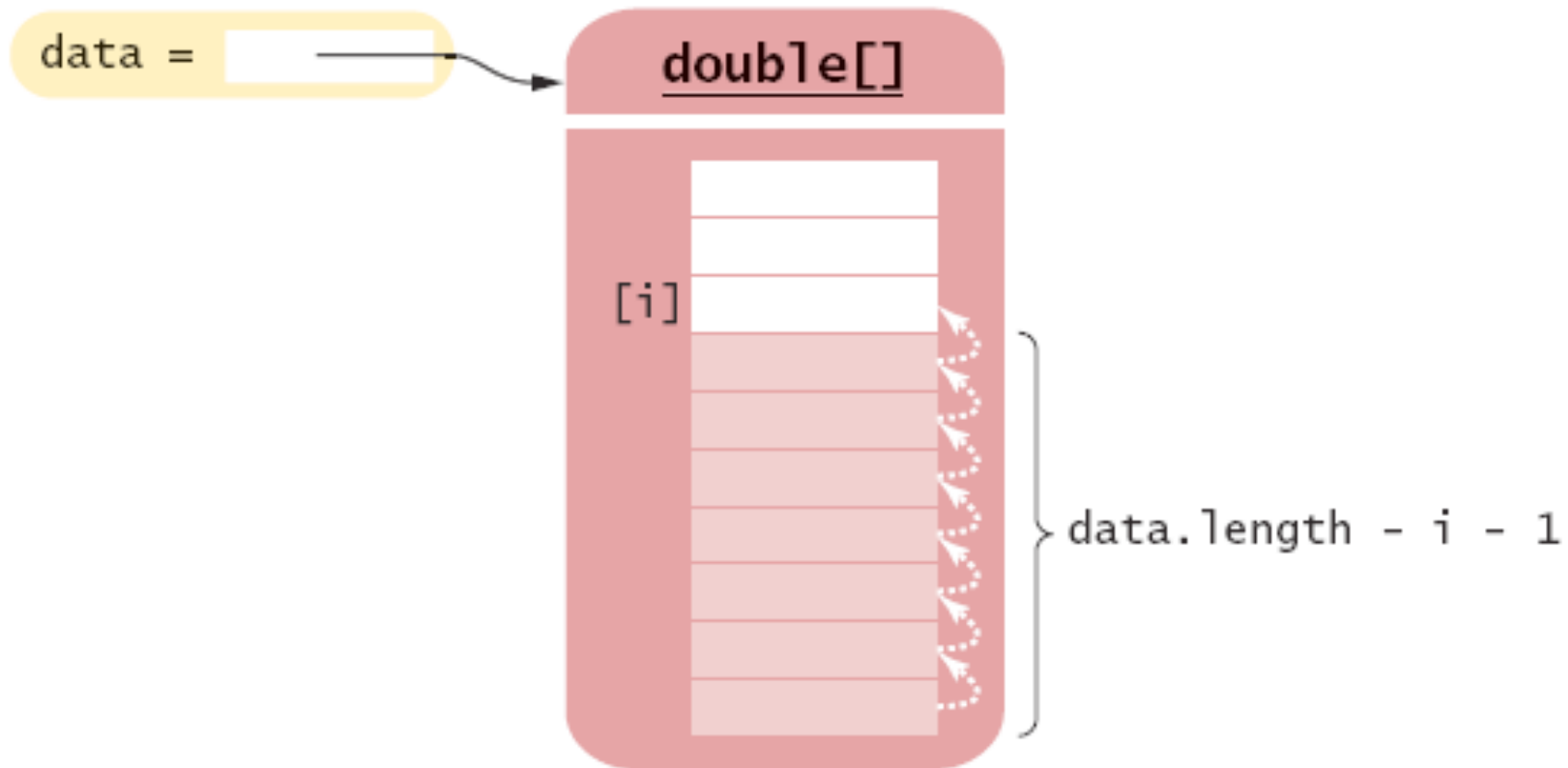


Figure 11 Removing an Element from an Array

Growing an Array

- If the array is full and you need more space, you can grow the array:
- Create a new, larger array:

```
double[] newData = new double[2 * data.length];
```
- Copy all elements into the new array:

```
System.arraycopy(data, 0, newData, 0, data.length);
```
- Store the reference to the new array in the array variable:

```
data = newData;
```

Growing an Array

```
Double[ ] newData = new double[2 * data.length] ①  
System.arraycopy(data, 0, newData, 0, data.length) ②
```

Continued

Growing an Array

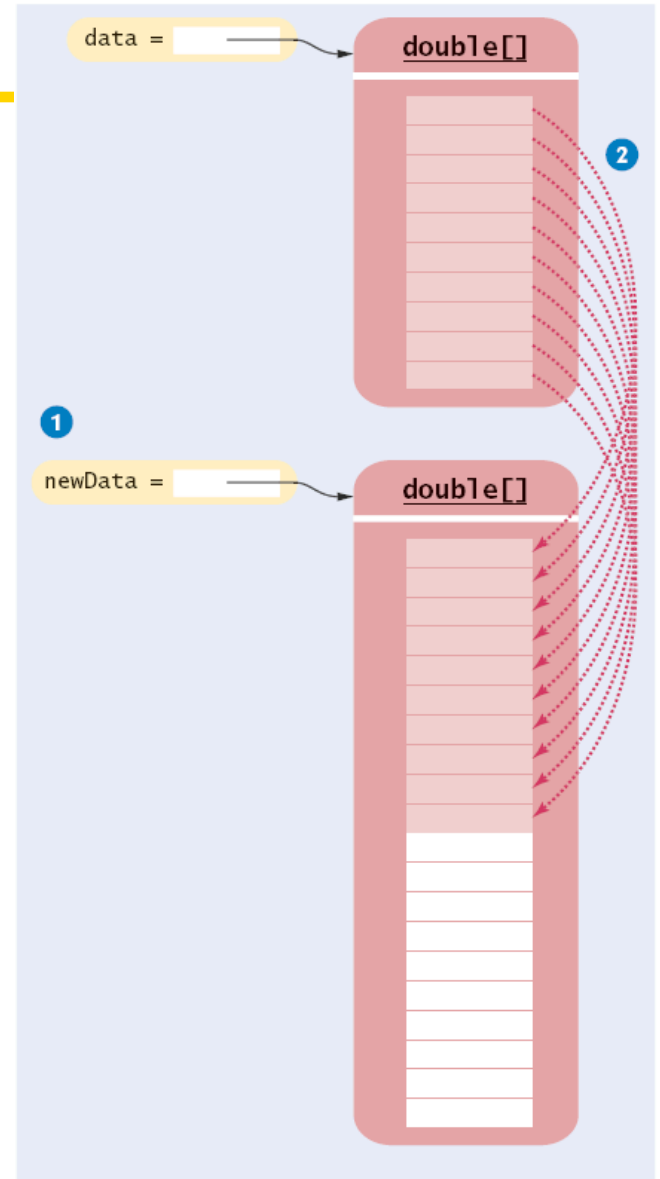


Figure 12 Growing an Array

Growing an Array

```
double[] newData = new double[2 * data.length]; ①  
System.arraycopy(data, 0, newData, 0, data.length); ②  
data = newData; ③
```

Continued

Growing an Array (cont.)

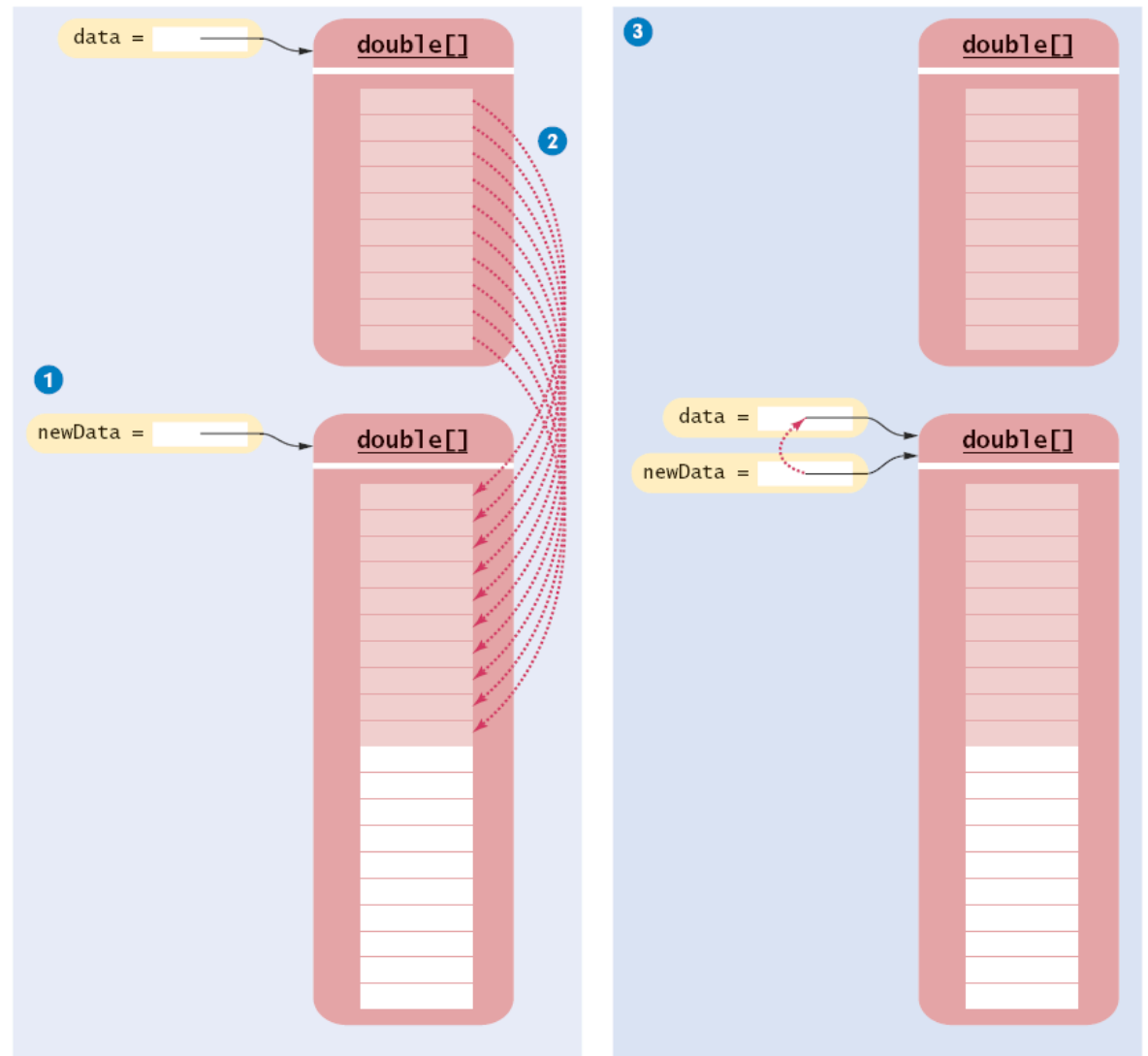


Figure 12 Growing an Array

Self Check 7.13

How do you add or remove elements in the middle of an array list?

Answer: Use the insert and remove methods.

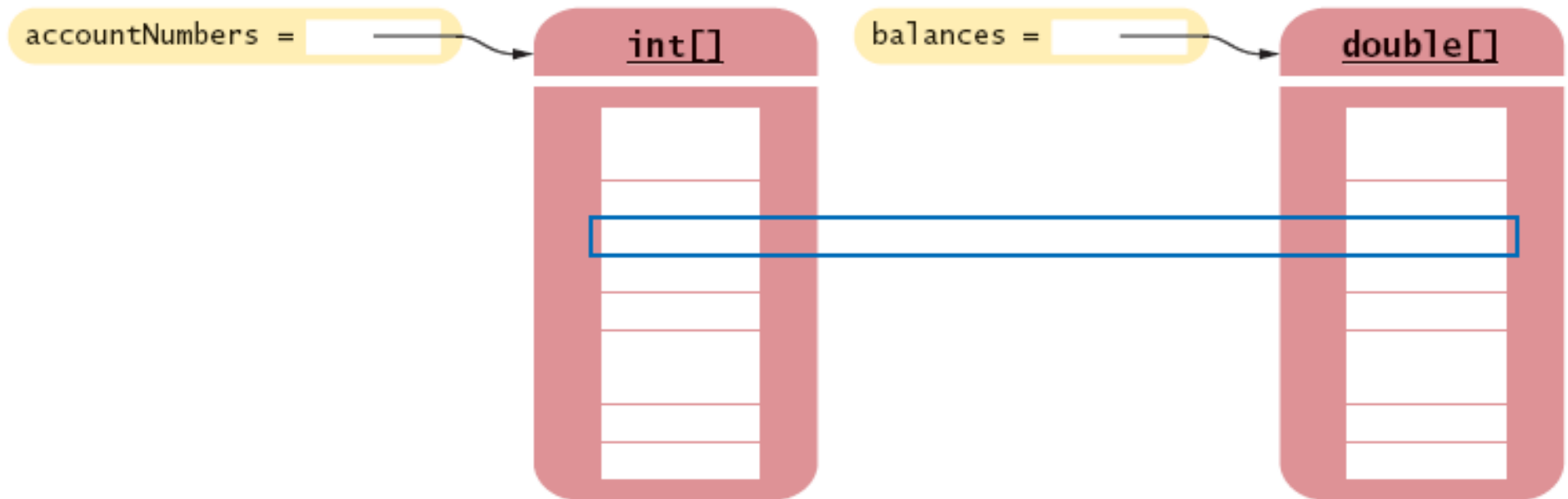
Self Check 7.14

Why do we double the length of the array when it has run out of space rather than increasing it by one element?

Answer: Allocating a new array and copying the elements is time-consuming. You wouldn't want to go through the process every time you add an element.

Make Parallel Arrays into Arrays of Objects

```
// Don't do this  
int[] accountNumbers;  
double[] balances;
```

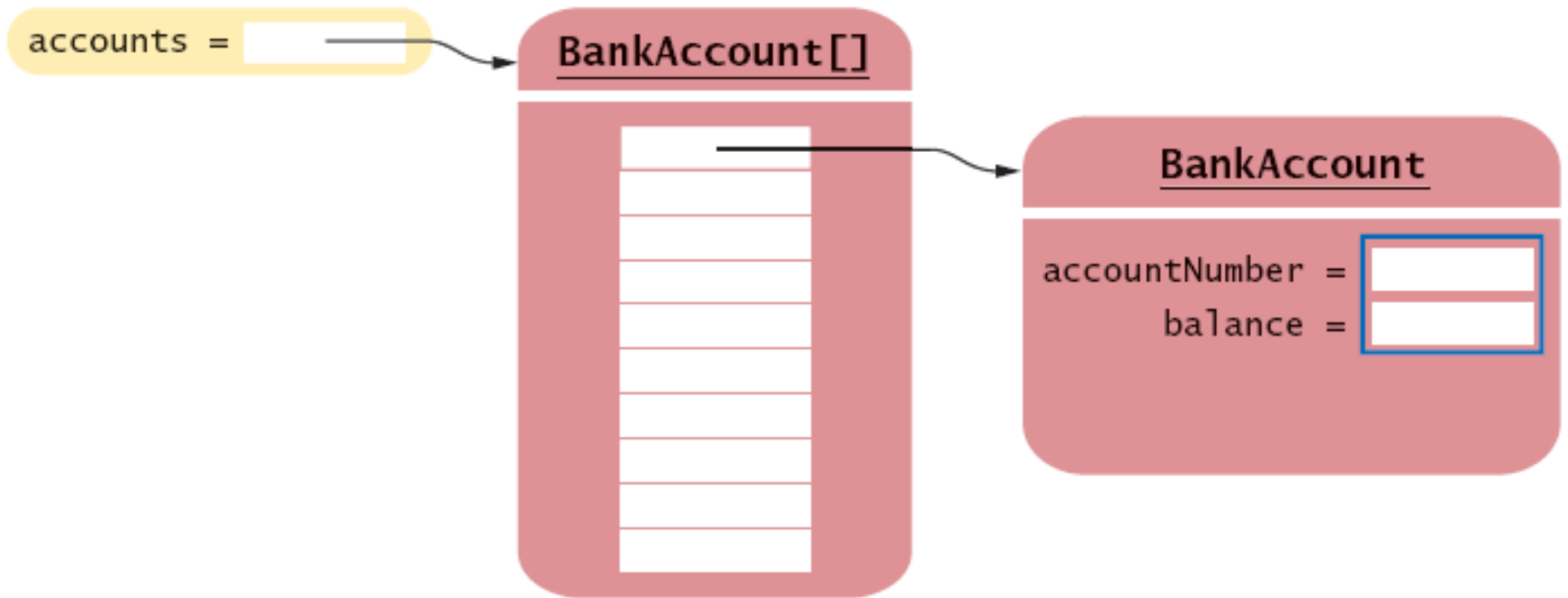


Avoid Parallel Arrays

Make Parallel Arrays into Arrays of Objects

Avoid parallel arrays by changing them into arrays of objects:

```
BankAccount[] = accounts
```



Reorganizing Parallel Arrays into an Array of Objects

Partially Filled Arrays

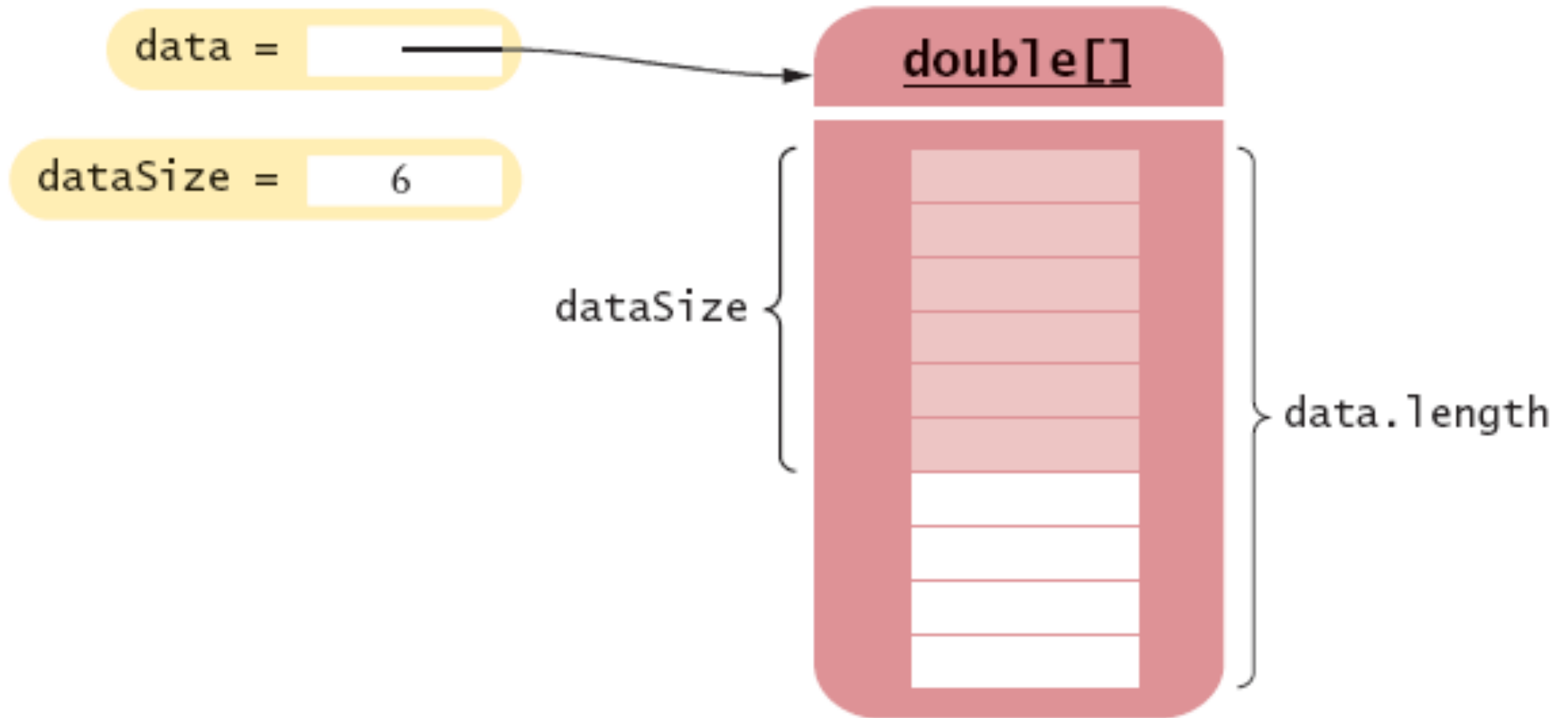
- Array length = maximum number of elements in array
- Usually, array is partially filled
- Need companion variable to keep track of current size
- Uniform naming convention:

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

- Update `dataSize` as array is filled:

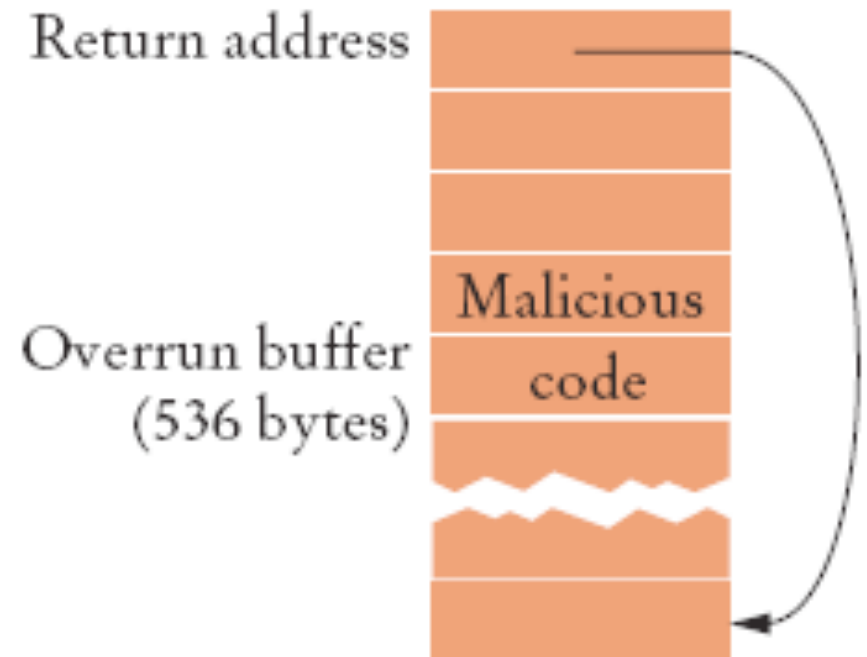
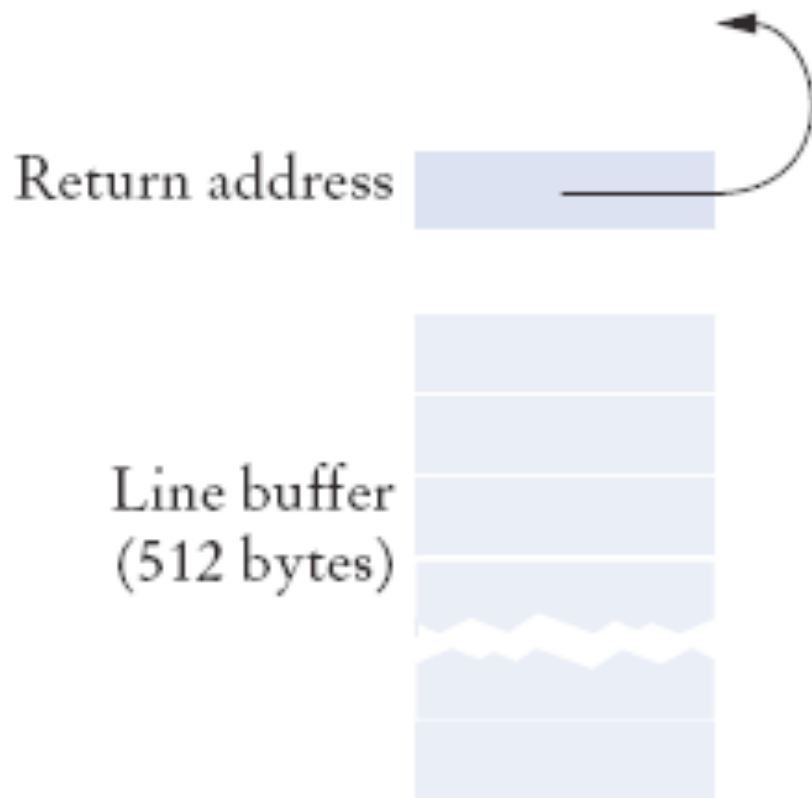
```
data[dataSize] = x;  
dataSize++;
```

Partially Filled Arrays (cont.)



A Partially Filled Array

An Early Internet Worm



A "Buffer Overrun" Attack

Regression Testing

- Save test cases
- Use saved test cases in subsequent versions
- A test suite is a set of tests for repeated testing
- Cycling = bug that is fixed but reappears in later versions
- Regression testing: repeating previous tests to ensure that known failures of prior versions do not appear in new versions

ch07/regression/BankTester.java

```
01: import java.util.Scanner;
02:
03: /**
04:     This program tests the Bank class.
05: */
06: public class BankTester
07: {
08:     public static void main(String[] args)
09:     {
10:         Bank firstBankOfJava = new Bank();
11:         firstBankOfJava.addAccount(new BankAccount(1001, 20000));
12:         firstBankOfJava.addAccount(new BankAccount(1015, 10000));
13:         firstBankOfJava.addAccount(new BankAccount(1729, 15000));
14:
15:         Scanner in = new Scanner(System.in);
16:
17:         double threshold = in.nextDouble();
18:         int c = firstBankOfJava.count(threshold);
19:         System.out.println("Count: " + c);
20:         int expectedCount = in.nextInt();
21:         System.out.println("Expected: " + expectedCount);
22:
```

Continued

ch07/regression/BankTester.java

```
23:         int accountNumber = in.nextInt();
24:         BankAccount a = firstBankOfJava.find(accountNumber);
25:         if (a == null)
26:             System.out.println("No matching account");
27:         else
28:             {
29:                 System.out.println("Balance of matching account: " +
a.getBalance());
30:                 int matchingBalance = in.nextLine();
31:                 System.out.println("Expected: " + matchingBalance);
32:             }
33:     }
34: }
```

Input Redirection

- Store the inputs in a file
- `ch07/regression/input1.txt`:

```
15000  
2  
1015  
10000
```

- Type the following command into a shell window:

```
java BankTester < input1.txt
```

- Output:

```
Count: 2  
Expected: 2  
Balance of matching account: 10000  
Expected: 10000
```


Input Redirection (cont.)

- Output redirection:

```
java BankTester < input1.txt > output1.txt
```

Self Check 7.15

Suppose you modified the code for a method. Why do you want to repeat tests that already passed with the previous version of the code?

Answer: It is possible to introduce errors when modifying code.

Self Check 7.16

Suppose a customer of your program finds an error. What action should you take beyond fixing the error?

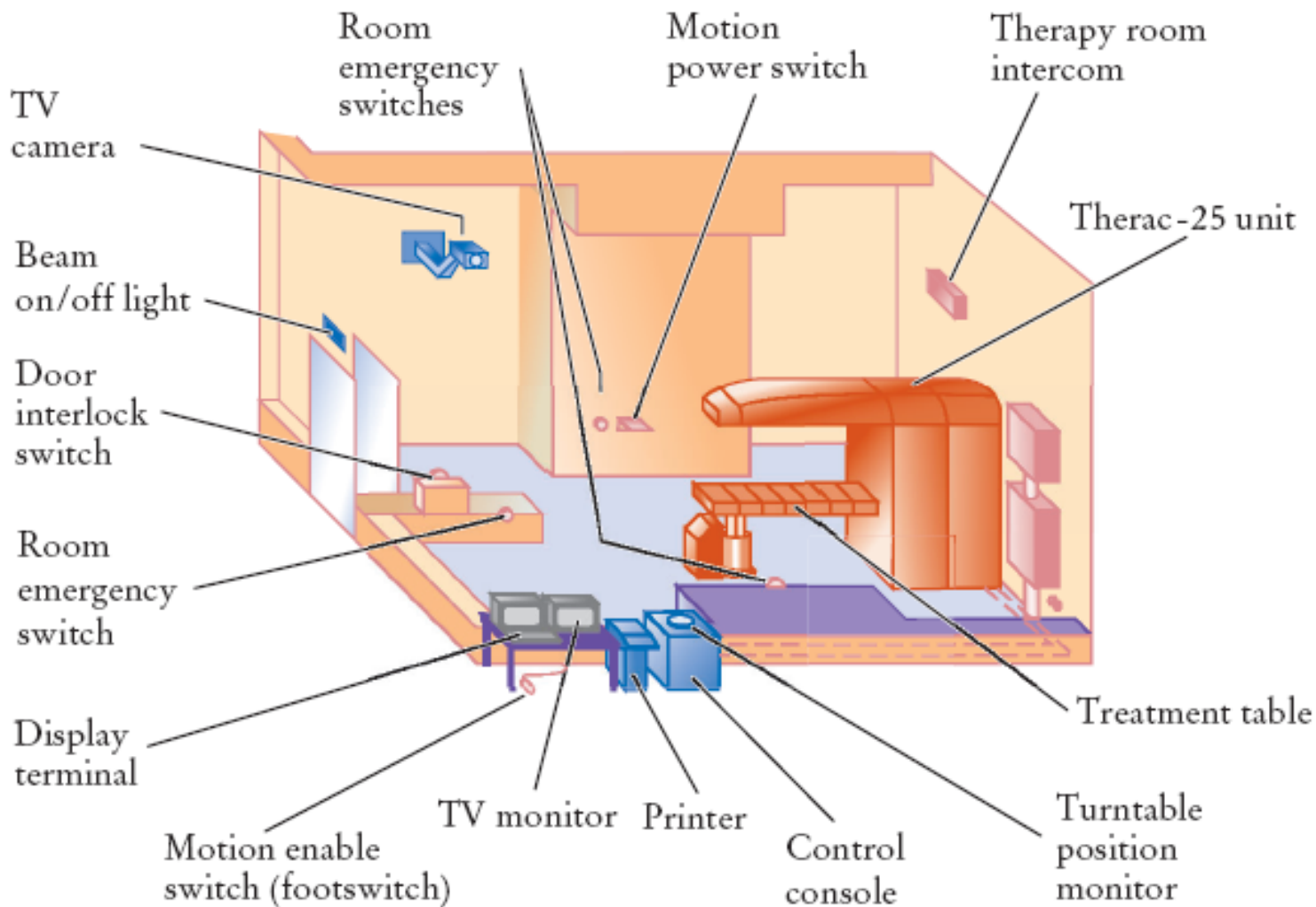
Answer: Add a test case to the test suite that verifies that the error is fixed.

Self Check 7.17

Why doesn't the `BankTester` program contain prompts for the inputs?

Answer: There is no human user who would see the prompts because input is provided from a file.

Therac-25 Facility



Typical Therac-25 Facility