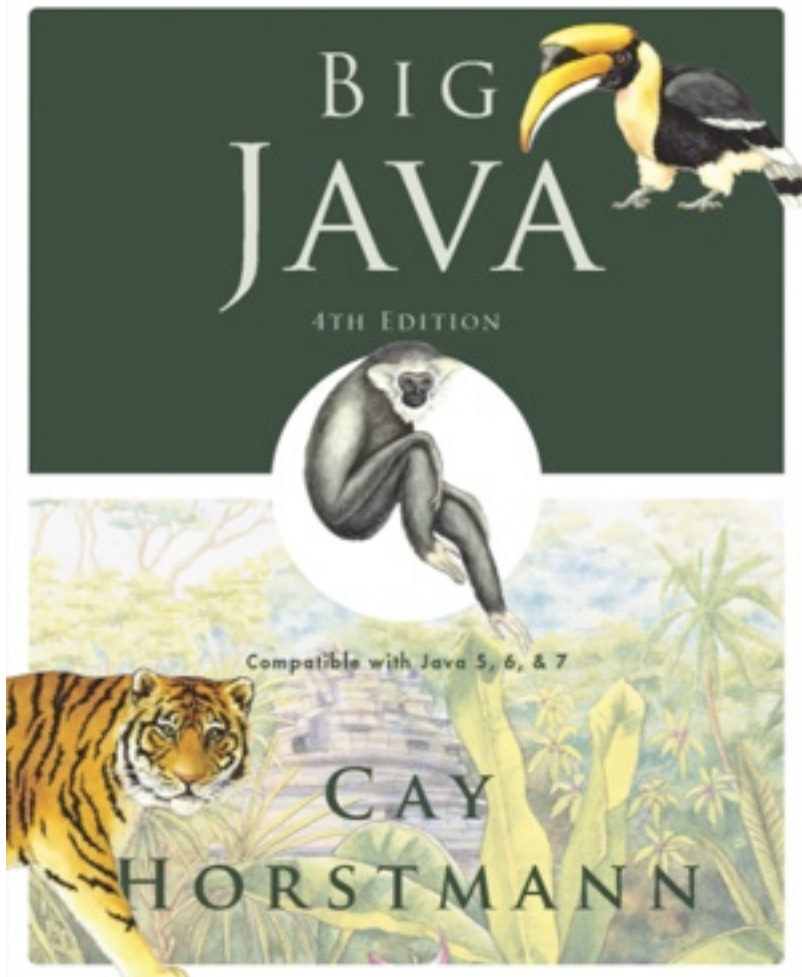# ICOM 4015: Advanced Programming

Lecture 7

Reading: Chapter Seven: Arrays and ArrayLists

Chapter 7 – Arrays and Array Lists

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

- **T** 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];`

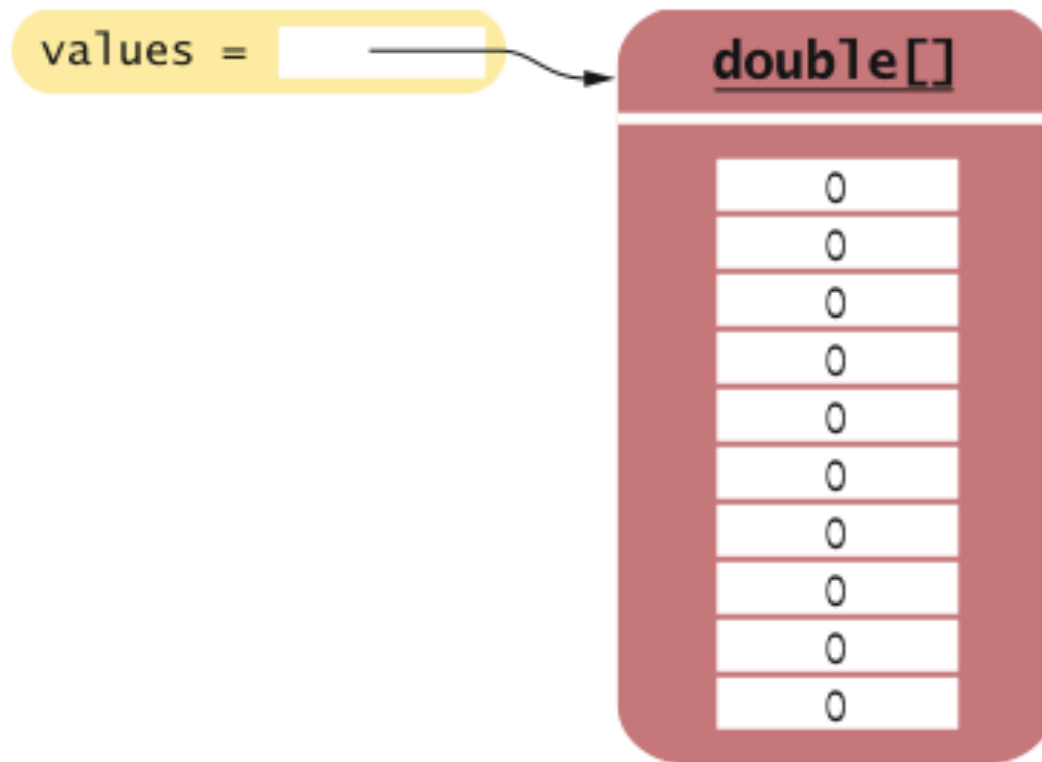- When array is created, all values are initialized depending on array type:

  - *Numbers:* `0`

  - *Boolean :* `false`

  - *Object References:* `null`

Arrays

Figure 1
An Array Reference
and an Array

```
values =
```

double[]

0
0
0
0
0
0
0
0
0
0

Arrays

Use `[]` to access an element:

```
values[2] = 29.95;
```
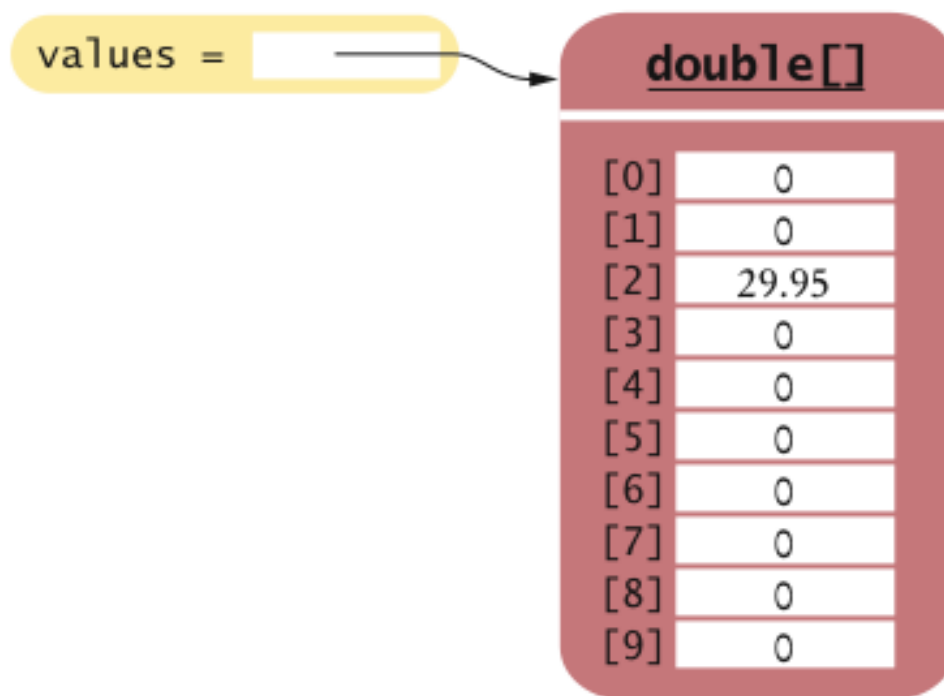

Figure 2
Modifying an
Array Element

Arrays

- Using the value stored:

```
System.out.println("The value of this data item is "
    + values[2]);
```

- Get array length as `values.length` (Not a method!)

- Index values range from `0` to `length - 1`

- Accessing a nonexistent element results in a **bounds error**:

```
double[] values = new double[10];
values[10] = 29.95; // ERROR
```

- Limitation: Arrays have fixed length

Declaring Arrays

Table 1 Declaring Arrays

`int[] numbers = new int[10];`	An array of ten integers. All elements are initialized with zero.
`final int NUMBERS_LENGTH = 10;` `int[] numbers = new int[NUMBERS_LENGTH];`	It is a good idea to use a named constant instead of a "magic number".
`int valuesLength = in.nextInt();` `double[] values = new double[valuesLength];`	The length need not be a constant.
`int[] squares = { 0, 1, 4, 9, 16 };`	An array of five integers, with initial values.
`String[] names = new String[3];`	An array of three string references, all initially `null`.
`String[] friends = { "Emily", "Bob", "Cindy" };`	Another array of three strings.
`double[] values = new int[10]`	**Error:** You cannot initialize a `double[]` variable with an array of type `int[]`.

Syntax 7.1 Arrays

Syntax   To construct an array: `new` *typeName* [*length*]

To access an element: *arrayReference* [*index*]

Example

Type of array variable

Name of array variable

Element type

Length

Initialized with zero

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

```
double[] moreValues = { 32, 54, 67.5, 29, 35 };
```

Initialized with these elements

Use brackets to access an element.

```
values[i] = 29.95;
```

The index must be ≥ 0 and < the length of the array.

Self Check 7.1

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

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

Make Parallel Arrays into Arrays of Objects

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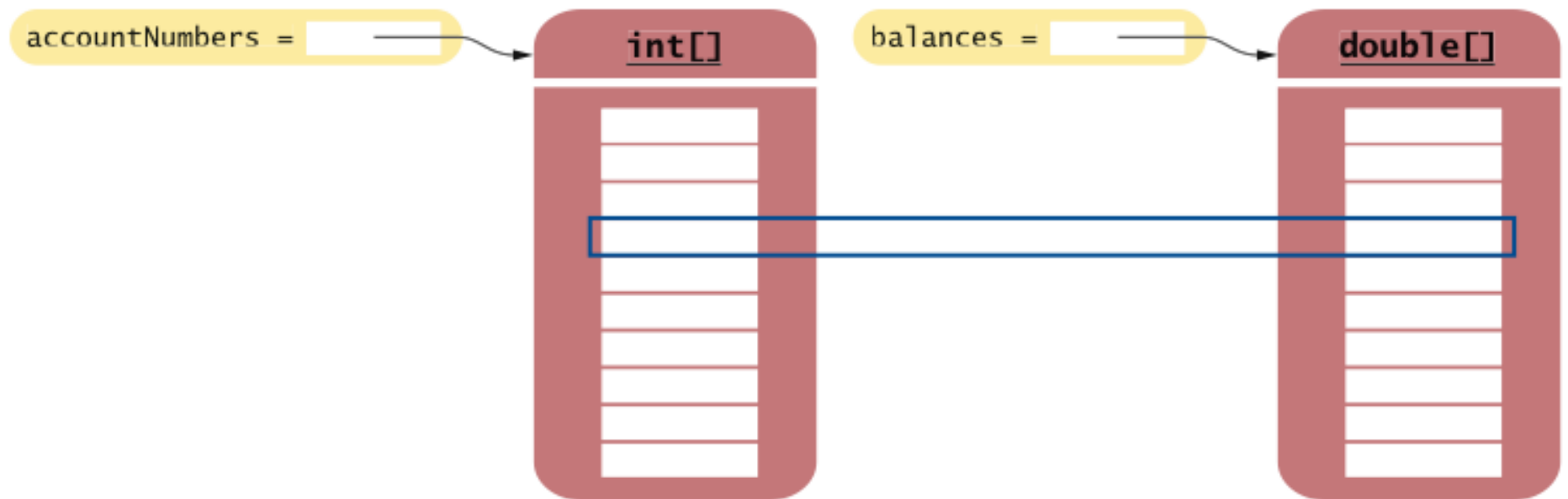

Figure 3 Avoid Parallel Arrays

Make Parallel Arrays into Arrays of Objects

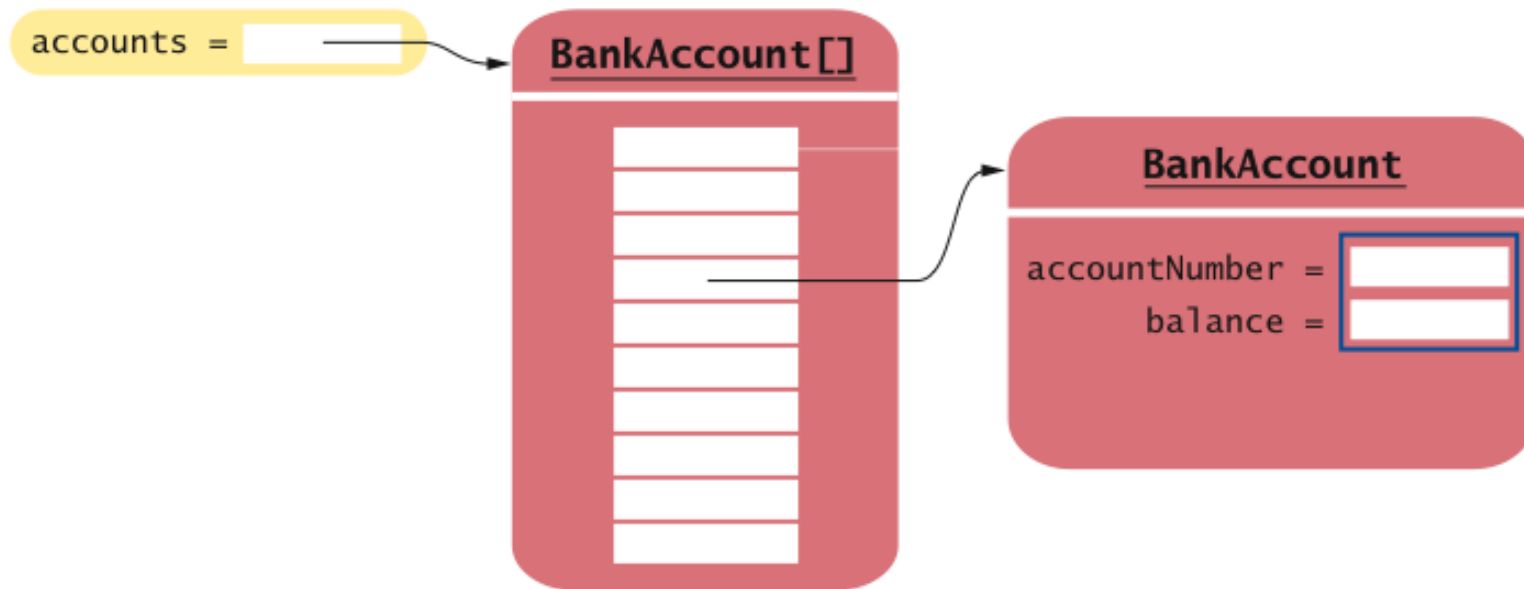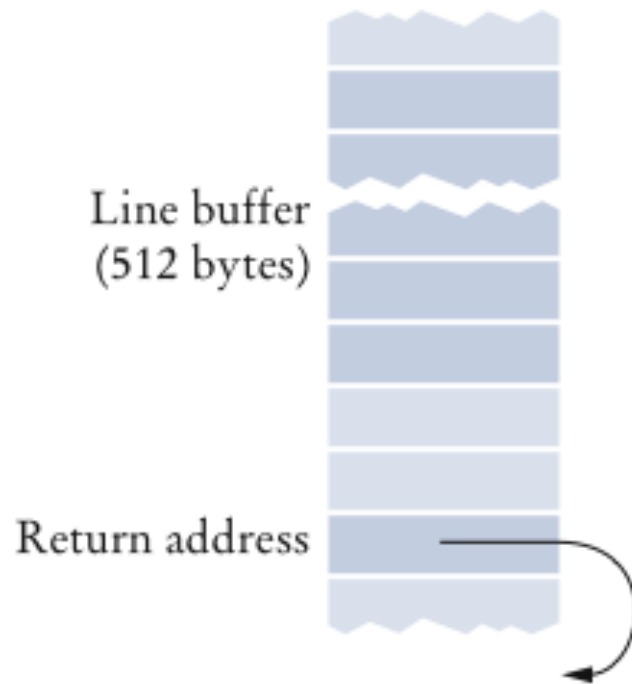Avoid parallel arrays by changing them into arrays of objects:

`BankAccount[] accounts;`

Figure 4 Reorganizing Parallel Arrays into an Array of Objects

An Early Internet Worm
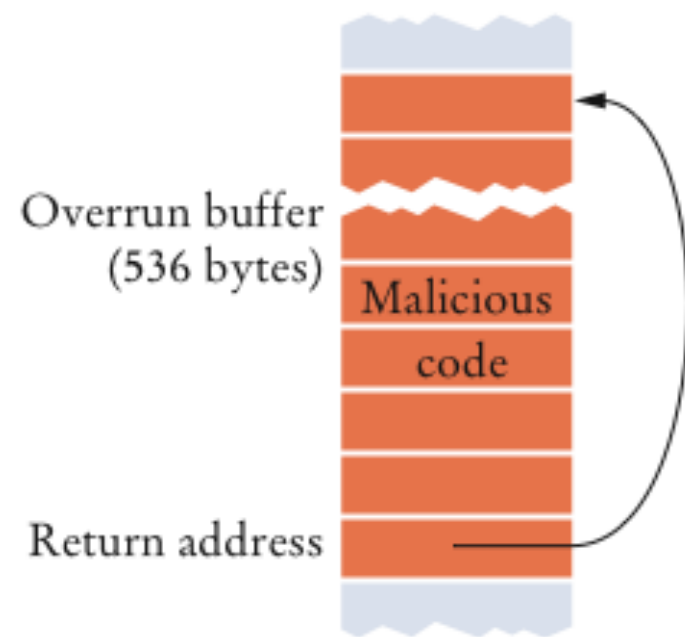
Array Lists

- `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

- `ArrayList` is a **generic class**:

  `ArrayList<T>`

  collects objects of **type parameter** `T`:

  ```
  ArrayList<String> names = new ArrayList<String>();
  names.add("Emily");
  names.add("Bob");
  names.add("Cindy");
  ```

- `size` method yields number of elements

Adding Elements

To add an object to the end of the array list, use the `add` method:

```
names.add("Emily");
names.add("Bob"); ❶
names.add("Cindy"); ❷
```
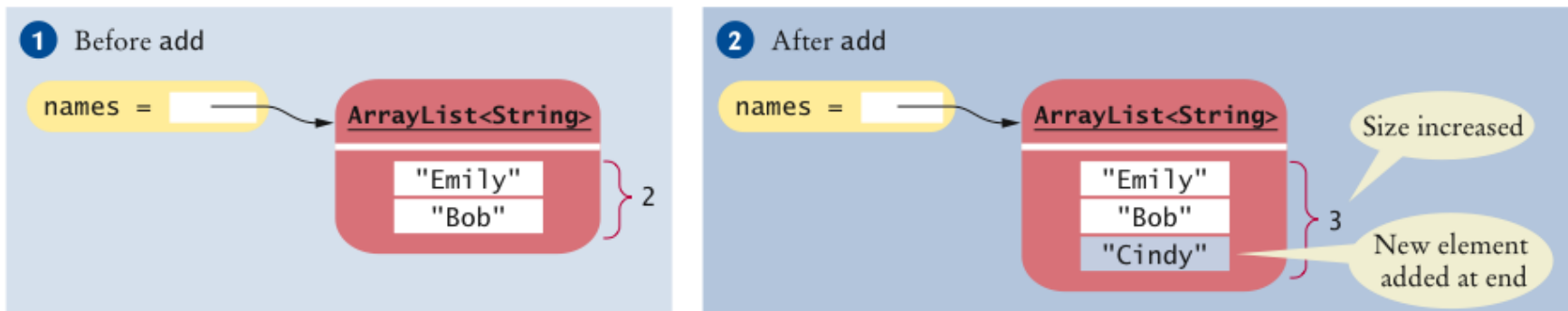

Figure 5 Adding an Element with add

Retrieving Array List Elements

- To obtain the value an element at an index, use the `get` method

- Index starts at 0

- `String name = names.get(2);`
`// gets the third element of the array list`

- Bounds error if index is out of range

- Most common bounds error:

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

Setting Elements

- To set an element to a new value, use the `set` method:

```
names.set(2, "Carolyn");
```

Removing Elements

- To remove an element at an index, use the `remove` method:

```
names.remove(1);
```

Adding and Removing Elements

```
names.add("Emily");
names.add("Bob");
names.add("Cindy");
names.set(2, "Carolyn"); ❶
names.add(1, "Ann"); ❷
names.remove(1); ❸
```

❶ **Before add**

names = → **ArrayList<String>**
"Emily"
"Bob"
"Carolyn"

❷ **After** names.add(1, "Ann")

names = → **ArrayList<String>**
"Emily"
"Ann" — New element added at index 1
"Bob" — Moved from index 1 to 2
"Carolyn" — Moved from index 2 to 3

❸ **After** names.remove(1)

names = → **ArrayList<String>**
"Emily"
"Bob" — Moved from index 2 to 1
"Carolyn" — Moved from index 3 to 2
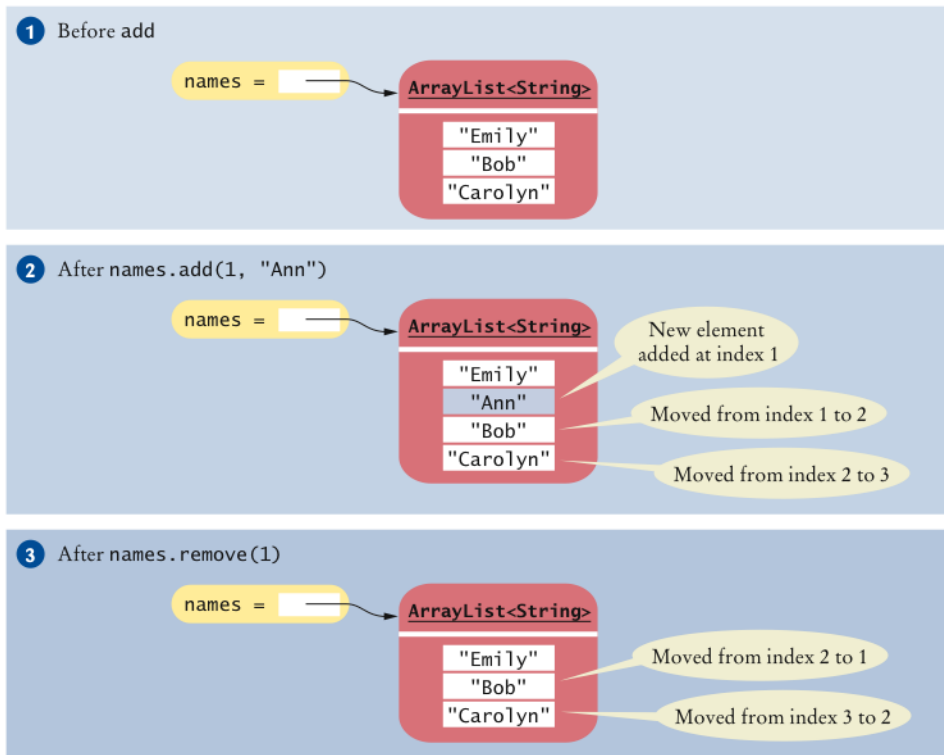
Figure 6 Adding and Removing Elements in the Middle of an Array List

Working with Array Lists

`ArrayList<String> names =` `    new ArrayList<String>();`	Constructs an empty array list that can hold strings.
`names.add("Ann");` `names.add("Cindy");`	Adds elements to the end.
`System.out.println(names);`	Prints `[Ann, Cindy]`.
`names.add(1, "Bob");`	Inserts an element at index 1. `names` is now `[Ann, Bob, Cindy]`.
`names.remove(0);`	Removes the element at index 0. `names` is now `[Bob, Cindy]`.
`names.set(0, "Bill");`	Replaces an element with a different value. `names` is now `[Bill, Cindy]`.

Working with Array Lists (cont.)

`String name = names.get(i);`	Gets an element.
`String last =` `    names.get(names.size() - 1);`	Gets the last element.

`ArrayList<Integer> squares =`
`    new ArrayList<Integer>();`
`for (int i = 0; i < 10; i++)`
`{`
`    squares.add(i * i);`
`}`
	Constructs an array list holding the first ten squares.

Syntax 7.2 Array Lists

Syntax   To construct an array list: `new ArrayList<typeName>()`

To access an element: `arraylistReference.get(index)`
`arraylistReference.set(index, value)`

Example

Variable type   Variable name   An array list object of size 0

```
ArrayList<String> friends = new ArrayList<String>();
```

```
friends.add("Cindy");
String name = friends.get(i);
friends.set(i, "Harry");
```

The add method appends an element to the array list, increasing its size.

Use the get and set methods to access an element.

The index must be ≥ 0 and < `friends.size()`.

ch07/arraylist/ArrayListTester.java

```java
 1    import java.util.ArrayList;
 2
 3    /**
 4       This program tests the ArrayList class.
 5    */
 6    public class ArrayListTester
 7    {
 8       public static void main(String[] args)
 9       {
10          ArrayList<BankAccount> accounts = new ArrayList<BankAccount>();
11          accounts.add(new BankAccount(1001));
12          accounts.add(new BankAccount(1015));
13          accounts.add(new BankAccount(1729));
14          accounts.add(1, new BankAccount(1008));
15          accounts.remove(0);
16
17          System.out.println("Size: " + accounts.size());
18          System.out.println("Expected: 3");
19          BankAccount first = accounts.get(0);
20          System.out.println("First account number: "
21                + first.getAccountNumber());
22          System.out.println("Expected: 1008");
23          BankAccount last = accounts.get(accounts.size() - 1);
24          System.out.println("Last account number: "
25                + last.getAccountNumber());
26          System.out.println("Expected: 1729");
27       }
28    }
```

```java
1   /**
2       A bank account has a balance that can be changed by
3       deposits and withdrawals.
4   */
5   public class BankAccount
6   {
7       private int accountNumber;
8       private double balance;
9
10      /**
11          Constructs a bank account with a zero balance.
12          @param anAccountNumber the account number for this account
13      */
14      public BankAccount(int anAccountNumber)
15      {
16          accountNumber = anAccountNumber;
17          balance = 0;
18      }
19
```

Continued

```java
20      /**
21          Constructs a bank account with a given balance
22          @param anAccountNumber the account number for this account
23          @param initialBalance the initial balance
24      */
25      public BankAccount(int anAccountNumber, double initialBalance)
26      {
27          accountNumber = anAccountNumber;
28          balance = initialBalance;
29      }
30
31      /**
32          Gets the account number of this bank account.
33          @return the account number
34      */
35      public int getAccountNumber()
36      {
37          return accountNumber;
38      }
39
```

Continued

```
40      /**
41          Deposits money into the bank account.
42          @param amount the amount to deposit
43      */
44      public void deposit(double amount)
45      {
46          double newBalance = balance + amount;
47          balance = newBalance;
48      }
49
50      /**
51          Withdraws money from the bank account.
52          @param amount the amount to withdraw
53      */
54      public void withdraw(double amount)
55      {
56          double newBalance = balance - amount;
57          balance = newBalance;
58      }
59
```

Continued

```
60      /**
61          Gets the current balance of the bank account.
62          @return the current balance
63      */
64      public double getBalance()
65      {
66          return balance;
67      }
68  }
```

Program Run:

```
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

Wrapper Classes

- For each primitive type there is a **wrapper class** for storing values of that type:

```
Double d = new Double(29.95);
```
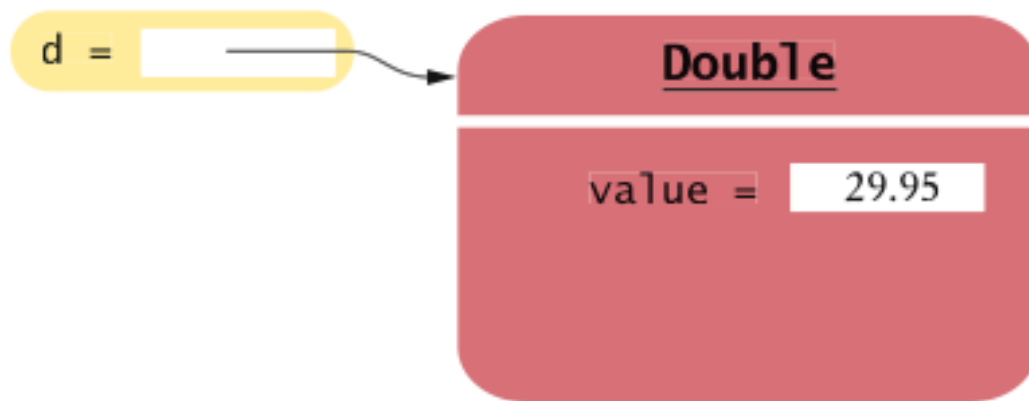

Figure 7 An Object of a Wrapper Class

- Wrapper objects can be used anywhere that objects are required instead of primitive type values:

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

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:** Automatic conversion between primitive types and the corresponding wrapper classes:

```
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:

```
d = 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* `d`

Auto-boxing and Array Lists

- To collect numbers in an array list, use the wrapper type as the type parameter, and then rely on auto-boxing:

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

- Storing wrapped numbers is quite inefficient

  - *Acceptable if you only collect a few numbers*

  - *Use arrays for long sequences of numbers or characters*

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

Answer:

```
values.set(0, values.get(0) + 1);
```

The Enhanced `for` Loop

- Traverses all elements of a collection:

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

- Read the loop as "for each `element` in `values`"

- Traditional alternative:

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

The Enhanced `for` Loop

- Works for `ArrayLists` too:

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

- Equivalent to the following ordinary `for` loop:

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

The Enhanced `for` Loop

- The "for each loop" does not allow you to modify the contents of an array:

```
for (double element : values)
{
    element = 0;
    // ERROR—this assignment does not
    // modify array element
}
```

- Must use an ordinary `for` loop:

```
for (int i = 0; i < values.length; i++)
{
    values[i] = 0; // OK
}
```

Syntax 7.3 The "for each" Loop

Syntax   for (*typeName variable* : *collection*)
         statement

Example

This variable is set in each loop iteration.
It is only defined inside the loop.

An array or array list

```
for (double element : values)
{
    sum = sum + element;
}
```

These statements are executed for each list element.

The variable contains an element, not an index.

Self Check 7.7

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

Answer:

```
for (double element : values)
    System.out.println(element);
```

Self Check 7.8

What does this "for each" loop do?

```
int counter = 0;
for (BankAccount a : accounts)
{
   if (a.getBalance() == 0) { counter++; }
}
```

Answer: It counts how many accounts have a zero balance.

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 VALUES_LENGTH = 100;
double[] values = new double[VALUES_LENGTH];
int valuesSize = 0;
```

- Update `valuesSize` as array is filled:

```
values[valuesSize] = x;
valuesSize++;
```
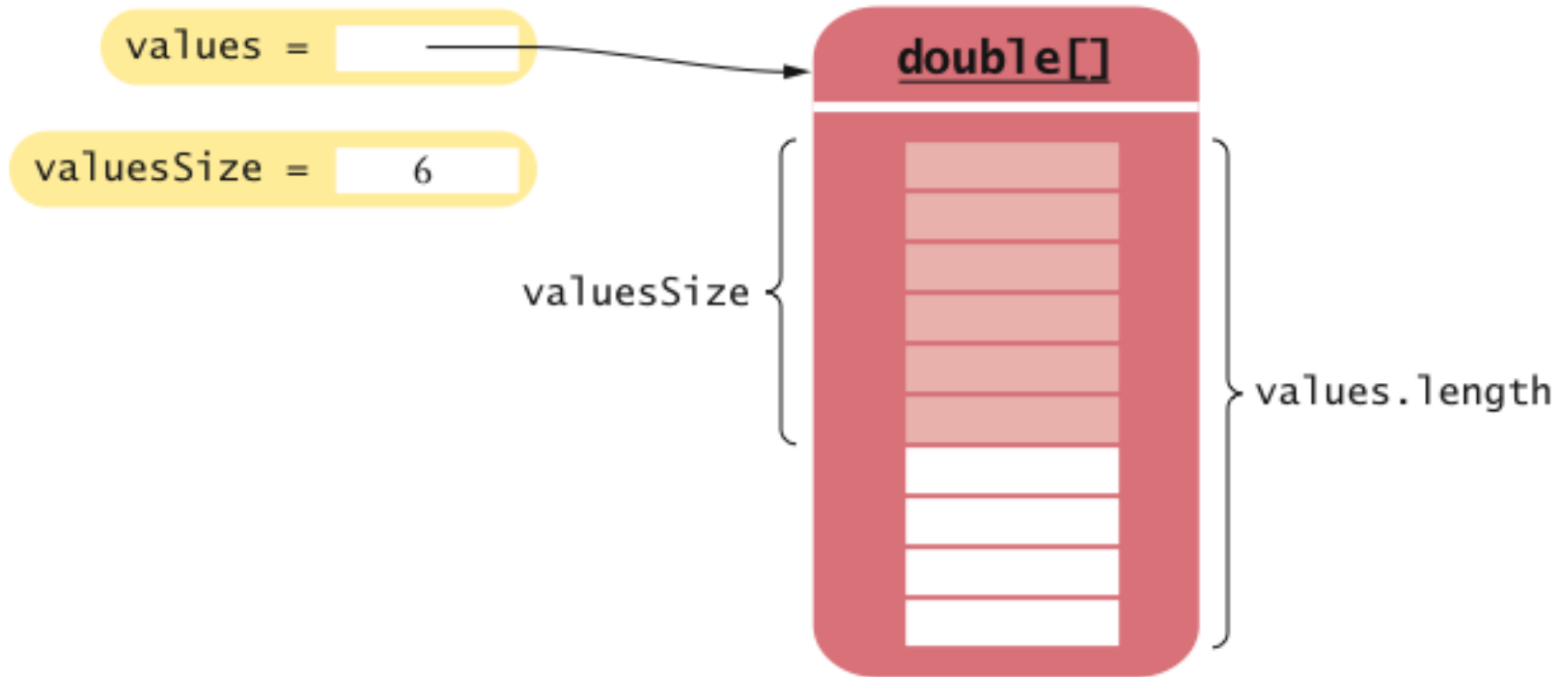
Partially Filled Arrays

values =

valuesSize = 6

double[]

valuesSize

values.length

Figure 8 A Partially Filled Array

Partially Filled Arrays

- Example: Read numbers into a partially filled array:

```java
int valuesSize = 0;
Scanner in = new Scanner(System.in);
while (in.hasNextDouble())
{
    if (valuesSize < values.length)
    {
        values[valuesSize] = in.nextDouble();
        valuesSize++;
    }
}
```

- To process the gathered array elements, use the companion variable, not the array length:

```java
for (int i = 0; i < valuesSize; i++)
{
    System.out.println(values[i]);
}
```

Self Check 7.9

Write a loop to print the elements of the partially filled array `values` in reverse order, starting with the last element.

Answer:

```java
for (int i = valuesSize - 1; i >= 0; i--)
   System.out.println(values[i]);
```

Self Check 7.10

How do you remove the last element of the partially filled array `values`?

Answer:

`valuesSize--;`

Self Check 7.11

Why would a programmer use a partially filled array of numbers instead of an array list?

Answer: You need to use wrapper objects in an `ArrayList<Double>`, which is less efficient.

Common Array Algorithm: Filling

- Fill an array with zeroes:

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

- Fill an array list with squares (0, 1, 4, 9, 16, ...):

```
for (int i = 0; i < values.size(); i++)
{
    values.set(i, i * i;
}
```

Common Array Algorithm: Computing Sum and Average

- To compute the sum of all elements, keep a running total:

```
double total = 0;
for (double element : values)
{
   total = total + element;
}
```

- To obtain the average, divide by the number of elements:

```
double average = total /values.size();
// for an array list
```

- Be sure to check that the size is not zero

Common Array Algorithm: Counting Matches

- Check all elements and count the matches until you reach the end

- Example: Count the number of accounts whose balance is at least as much as a given threshold:

```java
public class Bank
{
   private ArrayList<BankAccount> accounts;

   public int count(double atLeast)
   {
      int matches = 0;
      for (BankAccount account : accounts)
      {
         if (account.getBalance() >= atLeast) matches++; // Found a
match
      }
      return matches;
   }
   . . .
}
```

Common Array Algorithm: 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

Common Array Algorithm: Finding the Maximum or Minimum

- Example: Find the account with the largest balance in the bank:

```
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 — if list is empty, return `null`:

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

Common Array Algorithm: Searching for a Value

- Check all elements until you have found a match

- Example: Determine whether there is a bank account with a particular account number in the bank:

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

Common Array Algorithm: Searching for a Value

- The process of checking all elements until you have found a match is called a **linear search**

Common Array Algorithm: Locating the Position of an Element

- Problem: Locate the position of an element so that you can replace or remove it

- Use a variation of the linear search algorithm, but remember the position instead of the matching element

- Example: Locate the position of the first element that is larger than 100:

```java
int pos = 0;
boolean found = false;
while (pos < values.size() && !found)
{
    if (values.get(pos) > 100) { found = true; }
    else { pos++; }
}
if (found) { System.out.println("Position: " + pos); }
else { System.out.println("Not found"); }
```

Common Array Algorithm: Removing an Element

- Array list ⇒ use method `remove`

- Unordered array ⇒

  1. *Overwrite the element to be removed with the last element of the array*

  2. *Decrement the variable tracking the size of the array*

     ```
     values[pos] = values[valuesSize - 1];
     valuesSize--;
     ```

Common Array Algorithm: Removing an Element

- Ordered array ⇒

  1. *Move all elements following the element to be removed to a lower index*

  2. *Decrement the variable tracking the size of the array*

     ```
     for (int i = pos; i < valuesSize - 1; i++)
     {
         values[i] = values[i + 1];
     }
     valuesSize--;
     ```
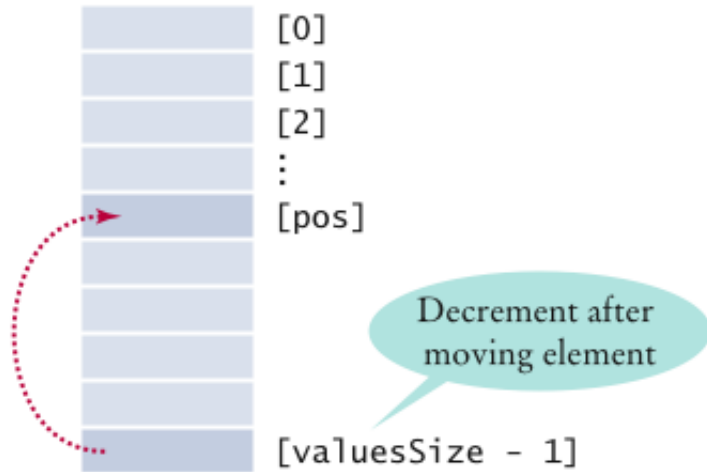
Common Array Algorithm: Removing an Element

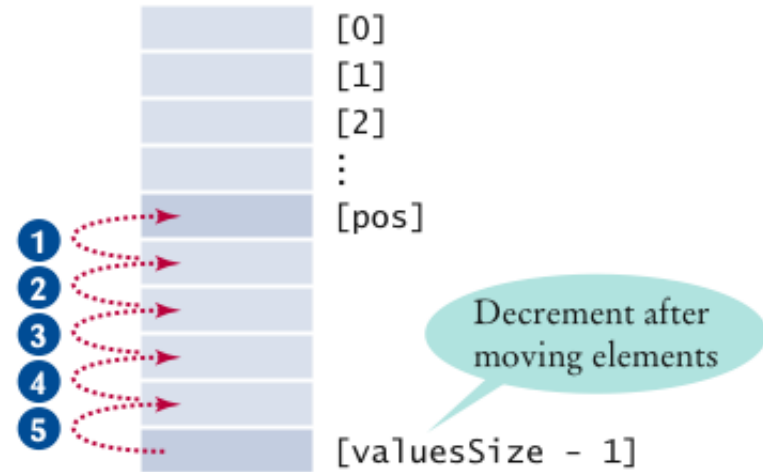**Figure 9**
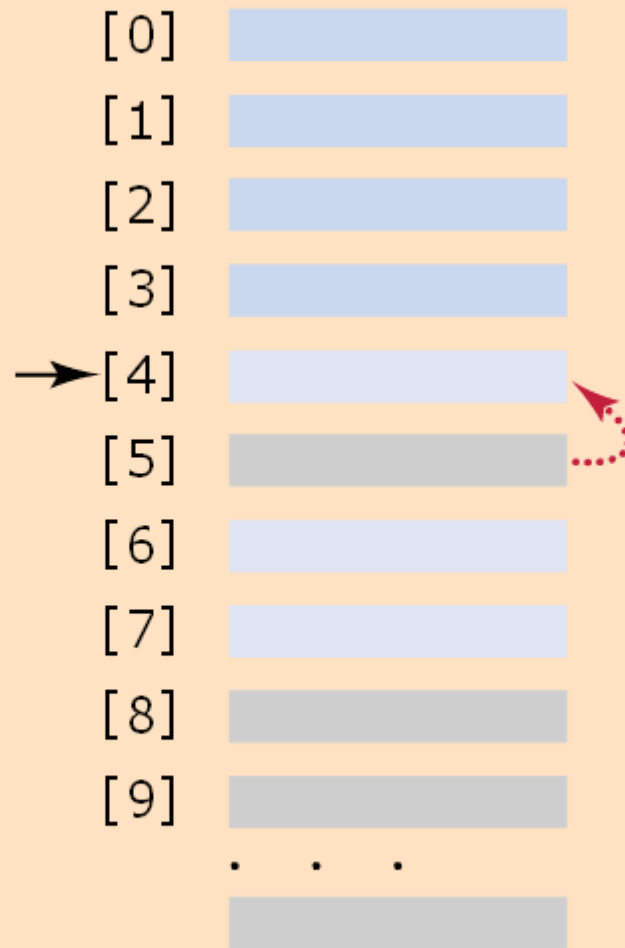Removing an Element in an Unordered Array

Figure 10
Removing an Element in an Ordered Array

Animation 7.1: Removing from an Array

The next element replaces the removed element.

[0]
[1]
[2]
[3]
→ [4]
[5]
[6]
[7]
[8]
[9]
. . .

Removing from an Array

Common Array Algorithm: Inserting an Element

- Array list ⇒ use method `add`

- Unordered array ⇒

  1. *Insert the element as the last element of the array*

  2. *Increment the variable tracking the size of the array*

```
if (valuesSize < values.length)
{
    values[valuesSize] = newElement;
    valuesSize++;
}
```

Common Array Algorithm: Inserting an Element

- Ordered array ⇒

  1. *Start at the end of the array, move that element to a higher index, then move the one before that, and so on until you finally get to the insertion location*

  2. *Insert the element*

  3. *Increment the variable tracking the size of the array*

```
if (valuesSize < values.length)
{
    for (int i = valuesSize; i > pos; i--)
    {
        values[i] = values[i - 1];
    }
    values[pos] = newElement;
    valuesSize++;
}
```
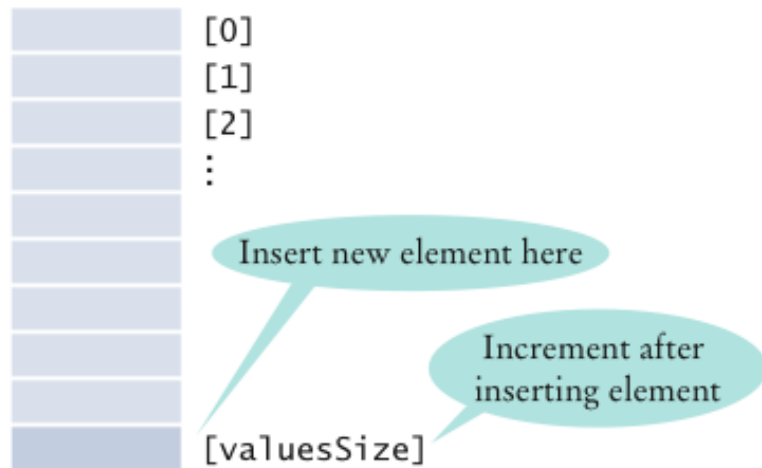
Common Array Algorithm: Inserting an Element

Figure 11
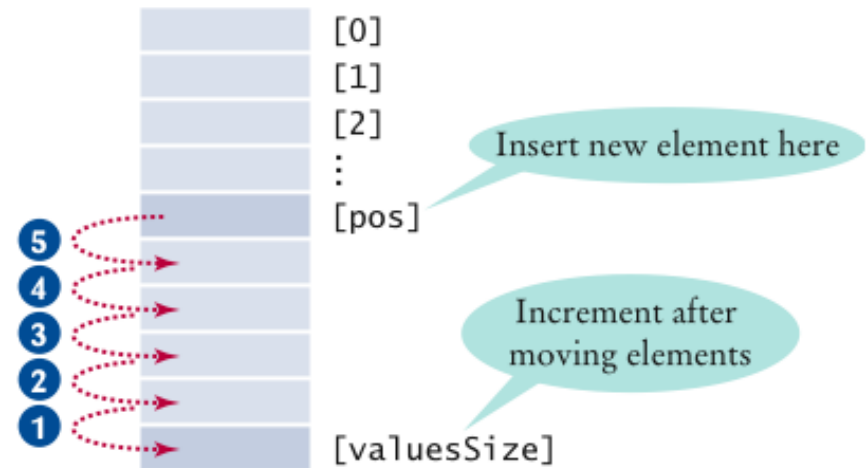Inserting an Element in an Unordered Array

Figure 12
Inserting an Element in an Ordered Array

Animation 7.2: Inserting into an Array

Elements are moved, starting from the removal location.

[0]
[1]
[2]
[3]
→ [4]
[5]
[6]
[7]
[8]
[9]
. . .
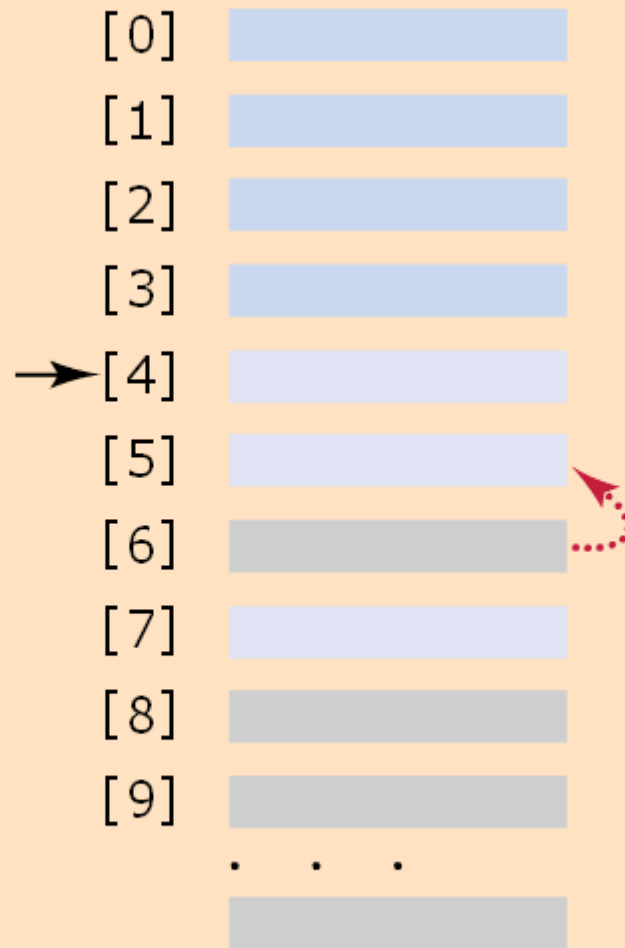
Removing from an Array

Common Array Algorithm: Copying an Array

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

```
double[] values = new double[6];
. . .   // Fill array
double[] prices = values; ①
```

① After the assignment prices = values

values =

prices =

double[]

| 32 |
| 54 |
| 67.5 |
| 29 |
| 35 |
| 47.5 |

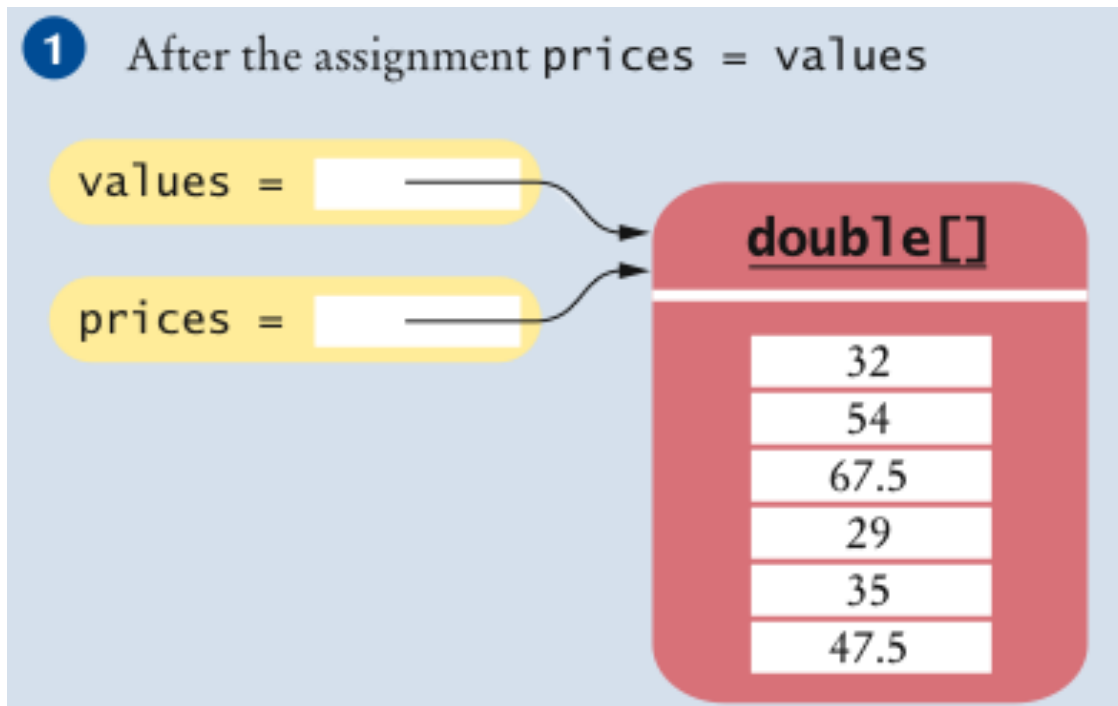# Common Array Algorithm: Copying an Array

- To make a true copy of an array, call the `Arrays.copyOf` method:

```
double[] prices = Arrays.copyOf(values, values.length); ②
```


② After calling `Arrays.copyOf`

values =   →   double[]

32
54
67.5
29
35
47.5

prices =   →   double[]

32
54
67.5
29
35
47.5
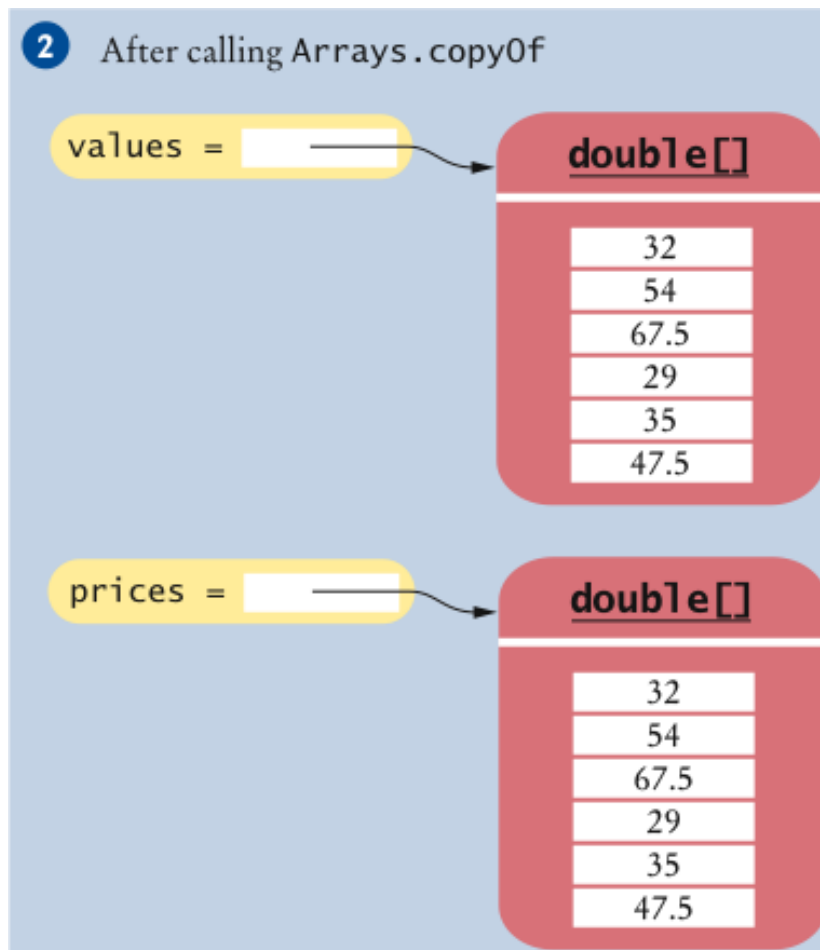
Common Array Algorithm: Copying an Array

- To grow an array that has run out of space, use the `Arrays.copyOf` method:

  `values = Arrays.copyOf(values, 2 * values.length);`

Figure 14 Growing an Array

Common Array Algorithm: Growing an Array

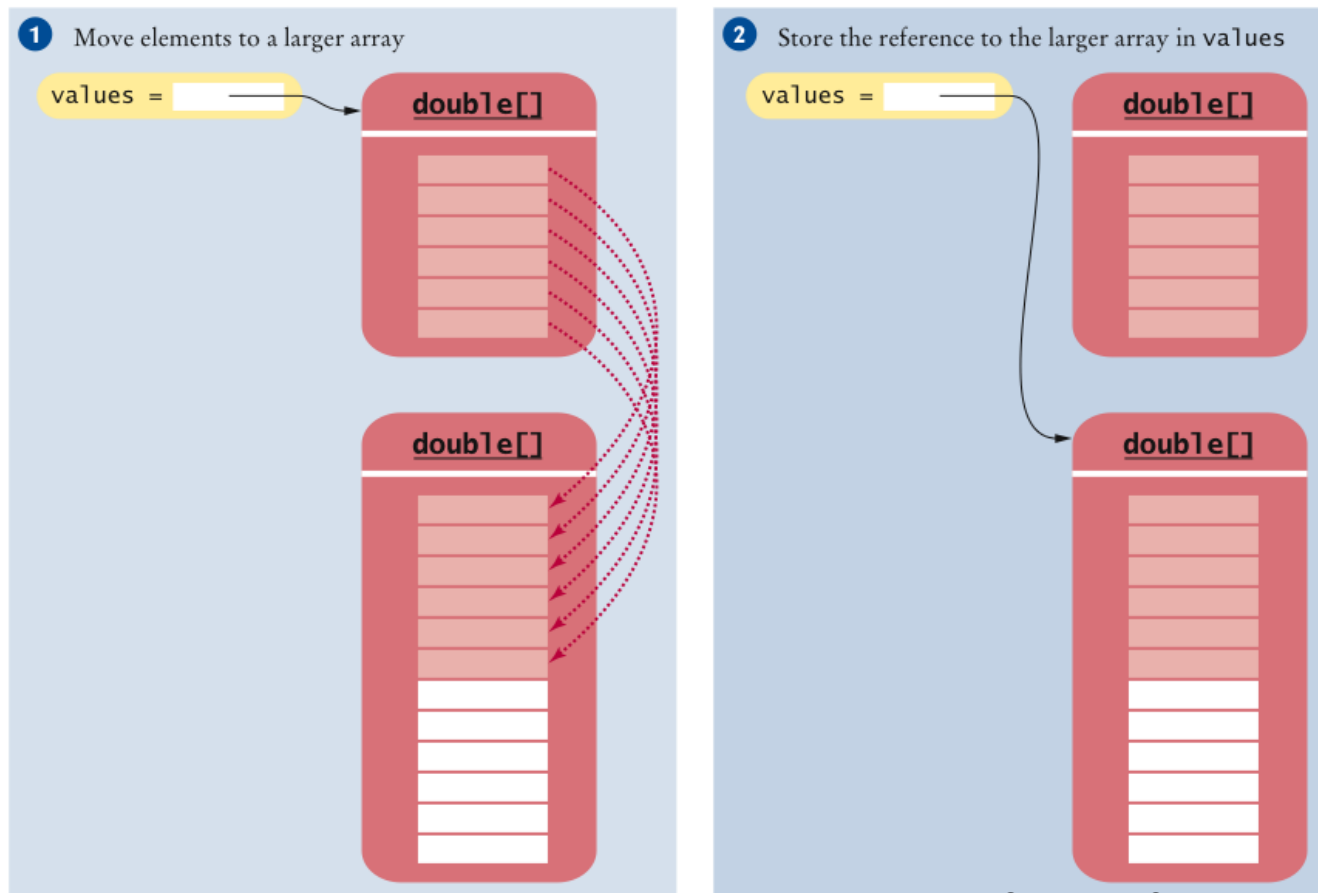- Example: Read an arbitrarily long sequence numbers into an array, without running out of space:

```
int valuesSize = 0;
while (in.hasNextDouble())
{
   if (valuesSize == values.length)
      values = Arrays.copyOf(values, 2 * values.length);
   values[valuesSize] = in.nextDouble();
   valuesSize++;
}
```

Common Array Algorithm: Printing Element Separators

- When you display the elements of an array or array list, you usually want to separate them:

  ```
  Ann | Bob | Cindy
  ```

- Print the separator before each element *except the initial one* (with index 0):

  ```java
  for (int i = 0; i < names.size(); i++)
  {
     if (i > 0)
     {
        System.out.print(" | ");
     }
     System.out.print(names.get(i));
  }
  ```

- `Bank` class stores an array list of bank accounts

- Methods of the `Bank` class use some of the previous algorithms:

```
 1  import java.util.ArrayList;
 2
 3  /**
 4      This bank contains a collection of bank accounts.
 5  */
 6  public class Bank
 7  {
 8     private ArrayList<BankAccount> accounts;
 9
10     /**
11         Constructs a bank with no bank accounts.
12     */
13     public Bank()
14     {
15        accounts = new ArrayList<BankAccount>();
16     }
17
```

Continued

```
18      /**
19          Adds an account to this bank.
20          @param a the account to add
21      */
22      public void addAccount(BankAccount a)
23      {
24          accounts.add(a);
25      }
26
27      /**
28          Gets the sum of the balances of all accounts in this bank.
29          @return the sum of the balances
30      */
31      public double getTotalBalance()
32      {
33          double total = 0;
34          for (BankAccount a : accounts)
35          {
36              total = total + a.getBalance();
37          }
38          return total;
39      }
40
```

Continued

```java
41      /**
42          Counts the number of bank accounts whose balance is at
43          least a given value.
44          @param atLeast the balance required to count an account
45          @return the number of accounts having least the given balance
46      */
47      public int countBalancesAtLeast(double atLeast)
48      {
49          int matches = 0;
50          for (BankAccount a : accounts)
51          {
52              if (a.getBalance() >= atLeast) matches++; // Found a match
53          }
54          return matches;
55      }
56
```

Continued

```
57      /**
58          Finds a bank account with a given number.
59          @param accountNumber the number to find
60          @return the account with the given number, or null if there
61          is no such account
62      */
63      public BankAccount find(int accountNumber)
64      {
65          for (BankAccount a : accounts)
66          {
67              if (a.getAccountNumber() == accountNumber) // Found a match
68                  return a;
69          }
70          return null; // No match in the entire array list
71      }
72
```

Continued

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

```java
1   /**
2       This program tests the Bank class.
3   */
4   public class BankTester
5   {
6      public static void main(String[] args)
7      {
8         Bank firstBankOfJava = new Bank();
9         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 count = firstBankOfJava.countBalancesAtLeast(threshold);
15        System.out.println("Count: " + count);
16        System.out.println("Expected: 2");
17
```

Continued

```java
18          int accountNumber = 1015;
19          BankAccount account = firstBankOfJava.find(accountNumber);
20          if (account == null)
21              System.out.println("No matching account");
22          else
23              System.out.println("Balance of matching account: "
24                  + account.getBalance());
25          System.out.println("Expected: 10000");
26
27          BankAccount max = firstBankOfJava.getMaximum();
28          System.out.println("Account with largest balance: "
29                  + max.getAccountNumber());
30          System.out.println("Expected: 1001");
31      }
32  }
```

Program Run:

```
Count: 2
Expected: 2
Balance of matching account: 10000.0
Expected: 10000
Account with largest balance: 1001
Expected: 1001
```

Big Java by Cay Horstmann

Self Check 7.12

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 7.13

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

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

Self Check 7.14

When printing separators, we skipped the separator before the initial element. Rewrite the loop so that the separator is printed *after* each element, except for the last element.

Answer:

```
for (int i = 0; i < values.size(); i++)
{
   System.out.print(values.get(i));
   if (i < values.size() - 1)
   {
      System.out.print(" | ");
   }
}
```

Now you know why we set up the loop the other way.

Self Check 7.15

The following replacement has been suggested for the algorithm that prints element separators:

```
System.out.print(names.get(0));
for (int i = 1; i < names.size(); i++)
   System.out.print(" | " + names.get(i));
```

What is problematic about this suggestion?

Answer: If `names` happens to be empty, the first line causes a bounds error.

Regression Testing

- **Test suite:** 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

```java
1   import java.util.Scanner;
2
3   /**
4       This program tests the Bank class.
5   */
6   public class BankTester
7   {
8      public static void main(String[] args)
9      {
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

```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.nextInt();
31             System.out.println("Expected: " + matchingBalance);
32          }
33       }
34    }
```

Regression Testing: 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
  ```

- Program Run:

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

Regression Testing: Output Redirection

- Output redirection:

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

Self Check 7.16

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 7.17

Suppose a customer of your program finds an error. What action should you take beyond fixing the error?
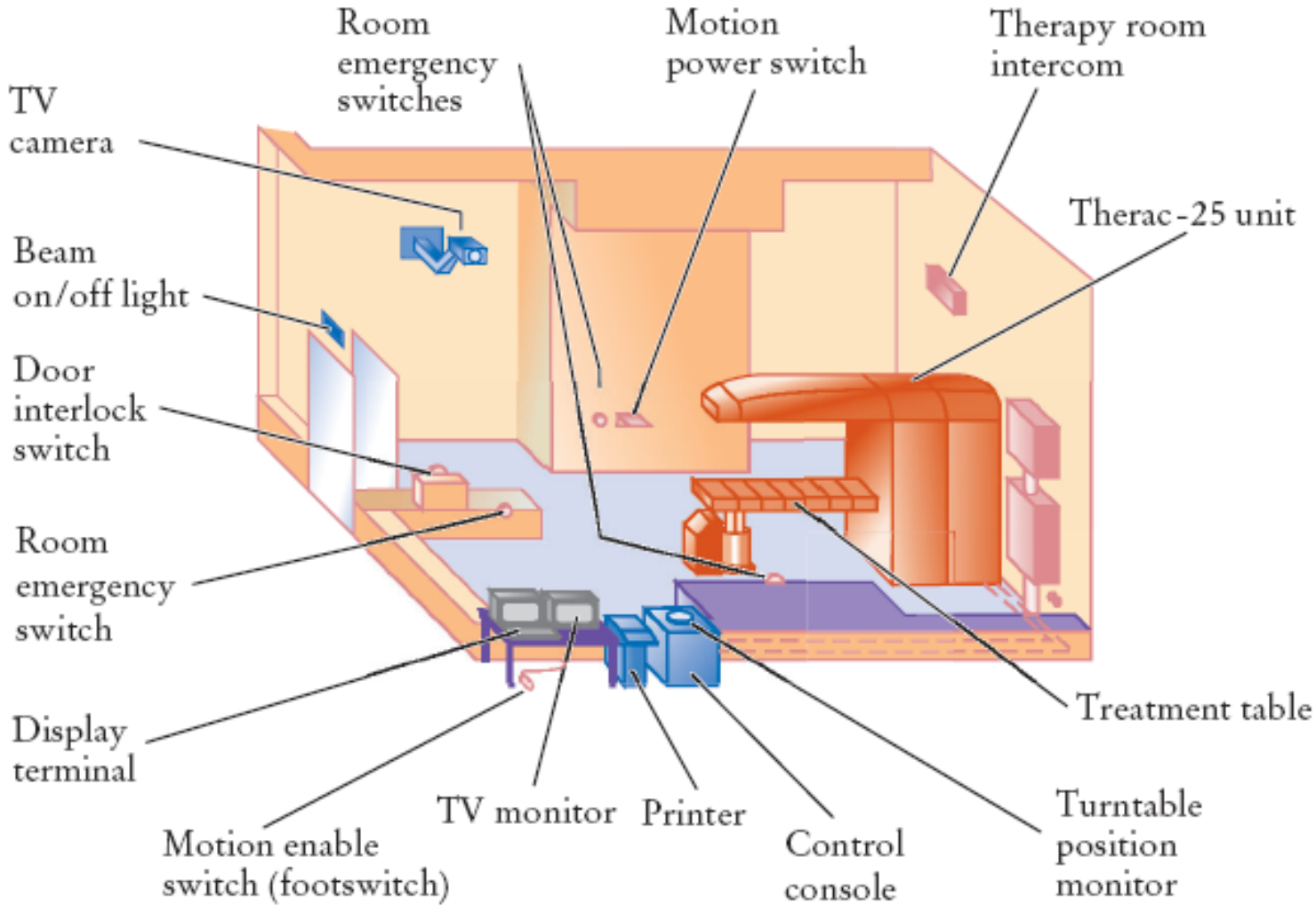
Answer: Add a test case to the test suite that verifies that the error is fixed.

Self Check 7.18

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

TV camera

Beam on/off light

Door interlock switch

Room emergency switch

Display terminal

Room emergency switches

Motion power switch

Therapy room intercom

Therac-25 unit

Treatment table

Motion enable switch (footswitch)

TV monitor

Printer

Control console

Turntable position monitor

Typical Therac-25 Facility
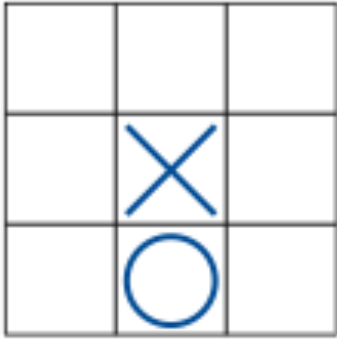
Two-Dimensional Arrays

Figure 15 A Tic-Tac-Toe Board

- When constructing a two-dimensional array, specify how many rows and columns are needed:

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

- Access elements with an index pair:

```
board[1][1] = "x";
board[2][1] = "o";
```

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] = " ";
```

Traversing Two-Dimensional Arrays

- You can also recover the array dimensions from the array variable:

  - *board.length is the number of rows*

  - *board[0].length is the number of columns*

- Rewrite the loop for filling the tic-tac-toe board:

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

```java
1  /**
2      A 3 x 3 tic-tac-toe board.
3  */
4  public class TicTacToe
5  {
6     private String[][] board;
7     private static final int ROWS = 3;
8     private static final int COLUMNS = 3;
9
10    /**
11       Constructs an empty board.
12    */
13    public TicTacToe()
14    {
15       board = new String[ROWS][COLUMNS];
16       // Fill with spaces
17       for (int i = 0; i < ROWS; i++)
18          for (int j = 0; j < COLUMNS; j++)
19             board[i][j] = " ";
20    }
21
```

Continued

```java
22      /**
23          Sets a field in the board. The field must be unoccupied.
24          @param i the row index
25          @param j the column index
26          @param player the player ("x" or "o")
27      */
28      public void set(int i, int j, String player)
29      {
30          if (board[i][j].equals(" "))
31              board[i][j] = player;
32      }
33
```

Continued

```
35              Creates a string representation of the board, such as
36              |x  o|
37              | x |
38              |  o|
39              @return the string representation
40          */
41          public String toString()
42          {
43              String r = "";
44              for (int i = 0; i < ROWS; i++)
45              {
46                  r = r + "|";
47                  for (int j = 0; j < COLUMNS; j++)
48                      r = r + board[i][j];
49                  r = r + "|\n";
50              }
51              return r;
52          }
53  }
```

```java
 1   import java.util.Scanner;
 2
 3   /**
 4       This program runs a TicTacToe game. It prompts the
 5       user to set positions on the board and prints out the
 6       result.
 7   */
 8   public class TicTacToeRunner
 9   {
10      public static void main(String[] args)
11      {
12         Scanner in = new Scanner(System.in);
13         String player = "x";
14         TicTacToe game = new TicTacToe();
```

Continued

```java
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              {
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  }
```

Program Run:

```
 |   |
 |   |
 |   |
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 7.19

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

Answer:

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

Self Check 7.20

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++;
```