Chapter 5 - Decisions

The Do Loop

- Executes the body of a loop at least once and performs the loop test **after** the body is executed.
- Use for input validation

  To force the user to enter a value less than 100

```
int value;
do
{
  System.out.print("Enter an integer < 100: ");
  value = in.nextInt();
}
while (value >= 100);
```

Self Check 6.34

What is wrong with the following loop for finding the position of the first space in a string?

```
boolean found = false;
for (int position = 0; !found && position < str.length(); position++)
{
  char ch = str.charAt(position);
  if (ch == ' ') { found = true; }
}
```

Answer: The loop will stop when a match is found, but you cannot access the match because neither `position` nor `ch` are defined outside the loop.

Nested Loop Examples

Table 3 Nested Loop Examples

Nested Loops	Output	Explanation

`for (i = 1; i <= 3; i++)`
`{`
`  for (j = 1; j <= 4; j++)  { Print "*" }`
`  System.out.println();`
`}`
	****
	****
	****
		Prints 3 rows of 4 asterisks each.

`for (i = 1; i <= 4; i++)`
`{`
`  for (j = 1; j <= 3; j++) { Print "*" }`
`  System.out.println();`
`}`
	***
	***
	***
	***
		Prints 4 rows of 3 asterisks each.

Application: Random Numbers and Simulations

- In a simulation, you use the computer to simulate an activity.
- You can introduce randomness by calling the random number generator.
- To generate a random number

  create an object of the `Random` class

  Call one of its methods

Method	Returns
`nextInt(n)`	A random integer between the integers 0 (inclusive) and `n` (exclusive)
`nextDouble()`	A random floating-point number between 0 (inclusive) and 1 (exclusive)

Application: Random Numbers and Simulations

- To simulate the cast of a die:

```
Random generator = new Random();
int d = 1 + generator.nextInt(6);
```

- The call `generator.nextInt(6)` gives you a random number between 0 and 5 (inclusive).
- Add 1 to obtain a number between 1 and 6.

Self Check 6.46

How do you generate a random floating-point number >= 0 and < 100?

Answer:

```
generator.nextDouble() * 100.0
```

Chapter 7 – Arrays and Array Lists

Arrays

- An array collects a sequence of values of the same type.
- Create an array that can hold ten values of type double:

```
new double[10]
```

  - The number of elements is the `length` of the array
  - The `new` operator constructs the array

- The type of an array variable is the type of the element to be stored, followed by `[]`.
- To declare an array variable of type `double[]`

```
double[] values;
```

- To initialize the array variable with the array:

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

- By default, each number in the array is 0
- You can specify the initial values when you create the array

```
double[] moreValues = { 32, 54, 67.5, 29, 35, 80, 115, 44.5, 100, 65
};
```

Arrays

- To access a value in an array, specify which “slot” you want to use
  - use the `[ ]` operator

```
values[4] = 35;
```

- The “slot number” is called an index.
- Each slot contains an element.
- Individual elements are accessed by an integer index `i`, using the notation `array[i]`.
- An array element can be used like any variable.

```
System.out.println(values[4]);
```

Arrays

- The elements of arrays are numbered starting at 0.
- The following declaration creates an array of 10 elements:

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

- An index can be any integer ranging from 0 to 9.
- The first element is `values[0]`
- The last element is `values[9]`
- An array index must be at least zero and less than the size of the array.
- Like a mailbox that is identified by a box number, an array element is identified by an index.

Declaring Arrays

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

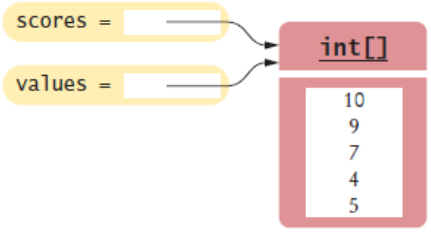
Array References

- An array reference specifies the location of an array.
- Copying the reference yields a second reference to the same array.
- When you copy an array variable into another, both variables refer to the same array

```
int[] scores = { 10, 9, 7, 4, 5 };
int[] values = scores; // Copying array reference
```

- You can modify the array through either of the variables:

```
scores[3] = 10;
System.out.println(values[3]);
// Prints 10
```



Using Arrays with Methods

- Arrays can occur as method arguments and return values.
- An array as a method argument

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

- To call this method

```
int[] scores = { 10, 9, 7, 10 };
fred.addScores(scores);
```

- A method with an array return value

```
public int[] getScores()
```

Partially Filled Arrays

- A loop to fill the array

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

- At the end of the loop, `currentSize` contains the actual number of elements in the array.
- Note: Stop accepting inputs when `currentSize` reaches the array length.

Make Parallel Arrays into Arrays of Objects

- Don't do this

```
int[] accountNumbers;
double[] balances;
```

- Don't use parallel arrays

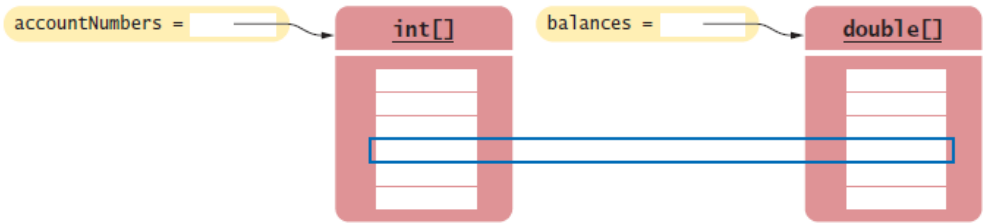


Figure 4 Avoid Parallel Arrays

Make Parallel Arrays into Arrays of Objects

Avoid parallel arrays by changing them into arrays of objects:

```
BankAccount[] accounts;
```

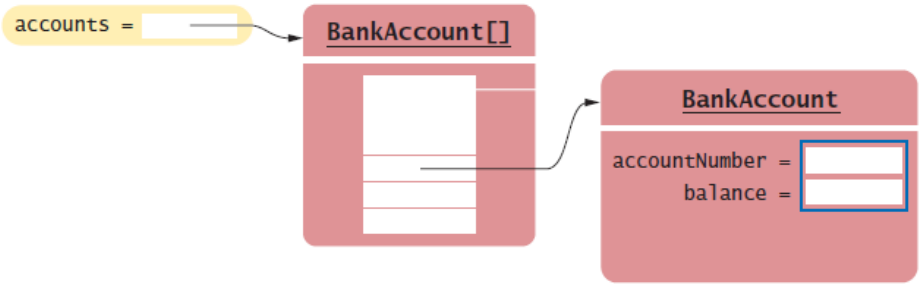


Figure 5 Reorganizing Parallel Arrays into an Array of Objects

The Enhanced `for` Loop

- You can use the enhanced `for` loop to visit all elements of an array.
- Totaling the elements in an array with the enhanced `for` loop

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

- The loop body is executed for each element in the array `values`.
- Read the loop as “for each `element` in `values`.”
- Traditional alternative:

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

The Enhanced `for` Loop

- Not suitable for all array algorithms.
- Does not allow you to modify the contents of an array.
- The following loop does not fill an array with zeros:

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

- Use a basic `for` loop instead:

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

- Use the enhanced `for` loop if you do not need the index values in the loop body.
- The enhanced `for` loop is a convenient mechanism for traversing all elements in a collection.

Common Array Algorithm: Maximum or Minimum

- Finding the maximum in an array

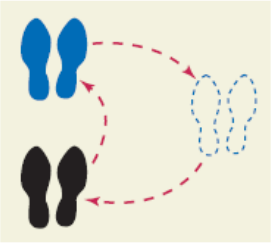
```
double largest = values[0];
for (int i = 1; i < values.length; i++)
{
  if (values[i] > largest)
  {
 largest = values[i];
  }
}
```

The loop starts at 1 because we initialize largest with `values[0]`.

- Finding the minimum: reverse the comparison.
- These algorithms require that the array contain at least one element.

Common Array Algorithm: Swapping Elements

- To swap two elements, you need a temporary variable.



- We need to save the first value in the temporary variable before replacing it.

```
double temp = values[i];
values[i] = values[j];
```

- Now we can set `values[j]` to the saved value.

```
values[j] = temp;
```

Common Array Algorithm: Swapping Elements

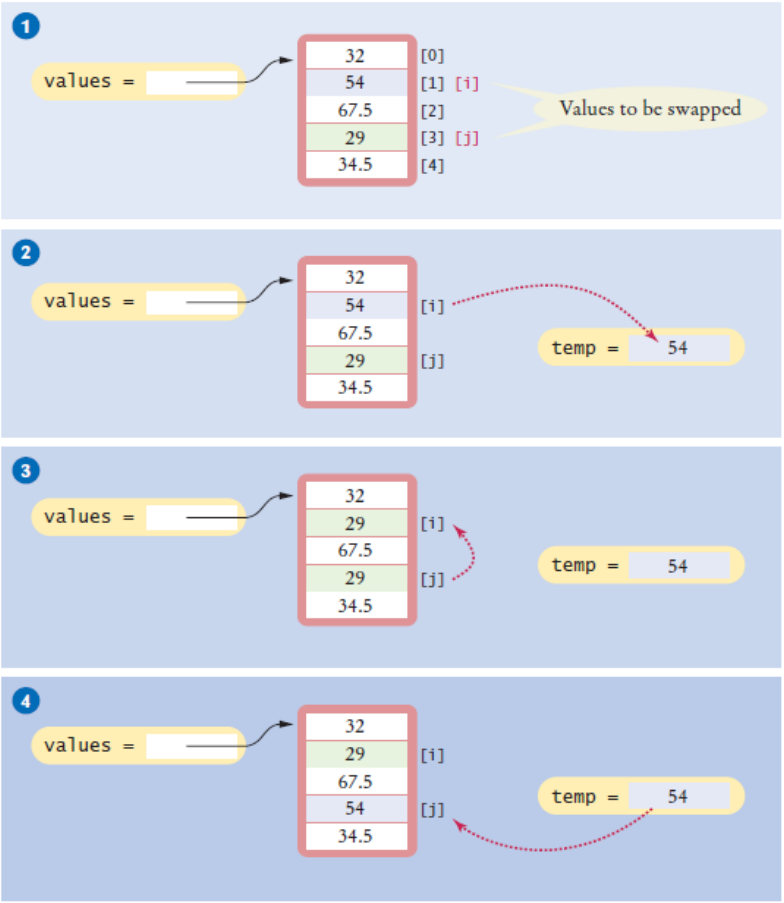


Figure 10 Swapping Array Elements

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;
```

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

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

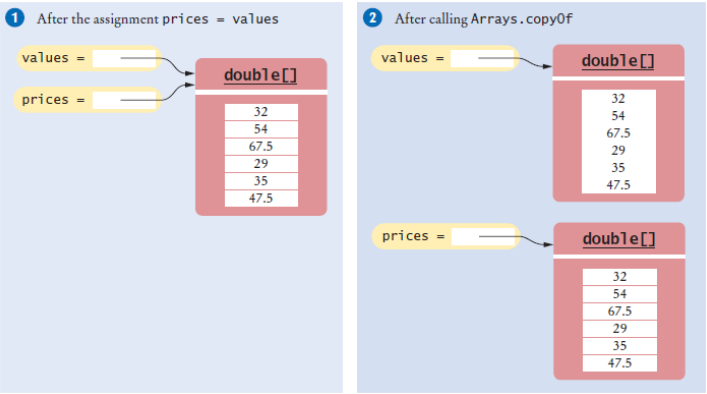


Figure 11 Copying an Array Reference versus Copying an Array

- To use the `Arrays` class, you need to add the following statement to the top of your program

```
import java.util.Arrays;
```

Common Array Algorithm: Growing an Array

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

```
double[] newValues = Arrays.copyOf(values, 2 * values.length);    ❶
values = newValues;    ❷
```

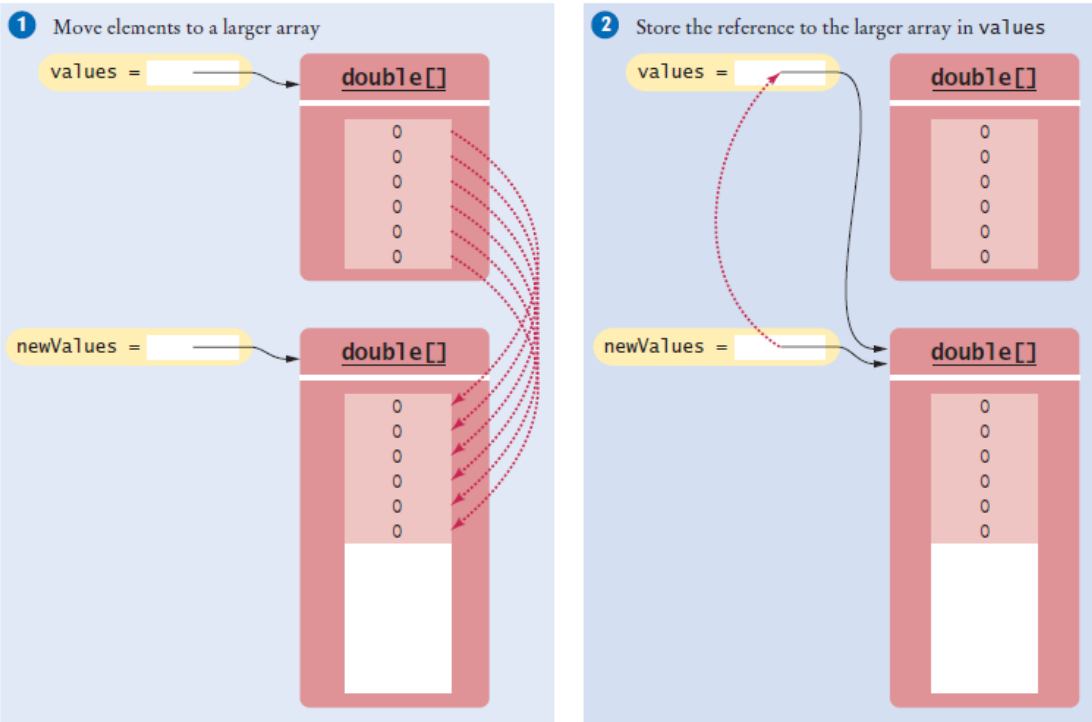


Figure 12 Growing an Array

Reading Input

- To read a sequence of arbitrary length:

  Add the inputs to an array until the end of the input has been reached.
  Grow when needed.

```
double[] inputs = new double[INITIAL_SIZE];
int currentSize = 0;
while (in.hasNextDouble())
{
  // Grow the array if it has been completely filled
  if (currentSize >= inputs.length)
  {
    inputs = Arrays.copyOf(inputs, 2 * inputs.length);
    // Grow the inputs array
  }
  inputs[currentSize] = in.nextDouble();
  currentSize++;
}
```

Discard unfilled elements.

```
inputs = Arrays.copyOf(inputs, currentSize);
```

Self Check 7.18

When finding the position of a match, we used a `while` loop, not a `for` loop. What is wrong with using this loop instead?

```
for (pos = 0; pos < values.length && !found; pos++)
{
  if (values[pos] > 100)
  {
    found = true;
  }
}
```

Answer: If there is a match, then `pos` is incremented before the loop exits.

Two-Dimensional Arrays

- An arrangement consisting of rows and columns of values

  Also called a matrix.

- Example: medal counts of the figure skating competitions at the 2014 Winter Olympics.

	Gold	Silver	Bronze
Canada	0	3	0
Italy	0	0	1
Germany	0	0	1
Japan	1	0	0
Kazakhstan	0	0	1
Russia	3	1	1
South Korea	0	1	0
United States	1	0	1

Figure 13 Figure Skating Medal Counts

- Use a two-dimensional array to store tabular data.
- When constructing a two-dimensional array, specify how many rows and columns are needed:

```
final int COUNTRIES = 8;
final int MEDALS = 3;
int[][] counts = new int[COUNTRIES][MEDALS];
```

Two-Dimensional Arrays

- You can declare and initialize the array by grouping each row:

```
int[][] counts =
{
   { 0, 3, 0 },
   { 0, 0, 1 },
   { 0, 0, 1 },
   { 1, 0, 0 },
   { 0, 0, 1 },
   { 3, 1, 1 },
   { 0, 1, 0 },
   { 1, 0, 1 }
};
```

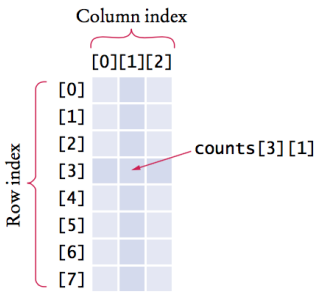
- You cannot change the size of a two-dimensional array once it has been declared.

Accessing Elements

- Access by using two index values, `array[i][j]`

```
int medalCount = counts[3][1];
```

- Use nested loops to access all elements in a two-dimensional array.
- Example: print all the elements of the `counts` array




```
for (int i = 0; i < COUNTRIES; i++)
{
  // Process the ith row
  for (int j = 0; j < MEDALS; j++)
  {
    // Process the jth column in the ith row
    System.out.printf("%8d", counts[i][j]);
  }
  System.out.println(); // Start a new line at the end of the row
}
```

Accessing Elements

- Number of rows: `counts.length`
- Number of columns: `counts[0].length`
- Example: print all the elements of the `counts` array

```
for (int i = 0; i < counts.length; i++)
{
  for (int j = 0; j < counts[0].length; j++)
  {
    System.out.printf("%8d", counts[i][j]);
  }
  System.out.println();
}
```

Array Lists

- An array list stores a sequence of values whose size can change.
- An array list can grow and shrink as needed.
- `ArrayList` class supplies methods for many common tasks, such as inserting and removing elements.
- An array list expands to hold as many elements as needed.

Syntax 7.4 Array Lists

Syntax

To construct an array list:	`new ArrayList<`*typeName*`>()`
To access an element:	*arraylistReference*`.get(index)` *arraylistReference*`.set(index, value)`

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().

Declaring and Using ArrayLists

- To declare an array list of strings

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

- To use an array list

```
import java.util.ArrayList;
```

- `ArrayList` is a **generic class**
- Angle brackets denote a **type parameter**

  Replace `String` with any other class to get a different array list type

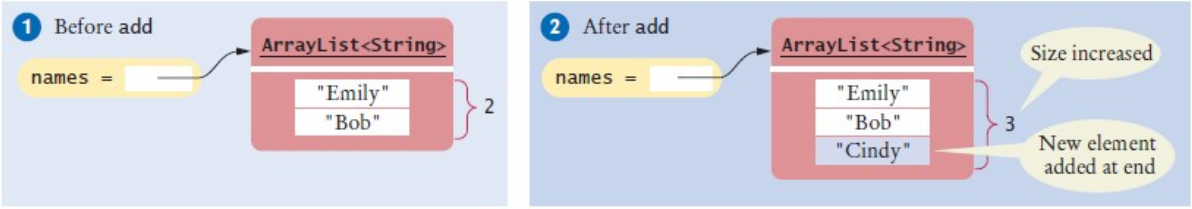
Declaring and Using ArrayLists

- `ArrayList<String>` is first constructed, it has size 0
- Use the `add` method to add an object to the end of the array list:

```
names.add("Emily"); // Now names has size 1 and element "Emily"
names.add("Bob"); // Now names has size 2 and elements "Emily", "Bob"
names.add("Cindy"); // names has size 3 and elements "Emily", "Bob", and "Cindy"
```

- The `size` method gives the current size of the array list.

Size is now 3



Declaring and Using ArrayLists

- To obtain an array list element, use the `get` method

  Index starts at 0

- To retrieve the name with index 2:

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

- The last valid index is `names.size() - 1`

  A common bounds error:

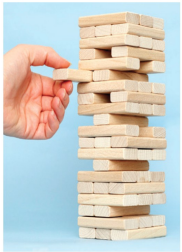
```
int i = names.size();
name = names.get(i); // Error
```

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

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

Declaring and Using ArrayLists

- An array list has methods for adding and removing elements in the middle.



- This statement adds a new element at position 1 and moves all elements with index 1 or larger by one position.

```
names.add(1, "Ann")
```

- The remove method,

  removes the element at a given position
  moves all elements after the removed element down by one position
  and reduces the size of the array list by 1.

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

- To print an array list:

```
System.out.println(names); // Prints [Emily, Bob, Carolyn]
```

Declaring and Using ArrayLists

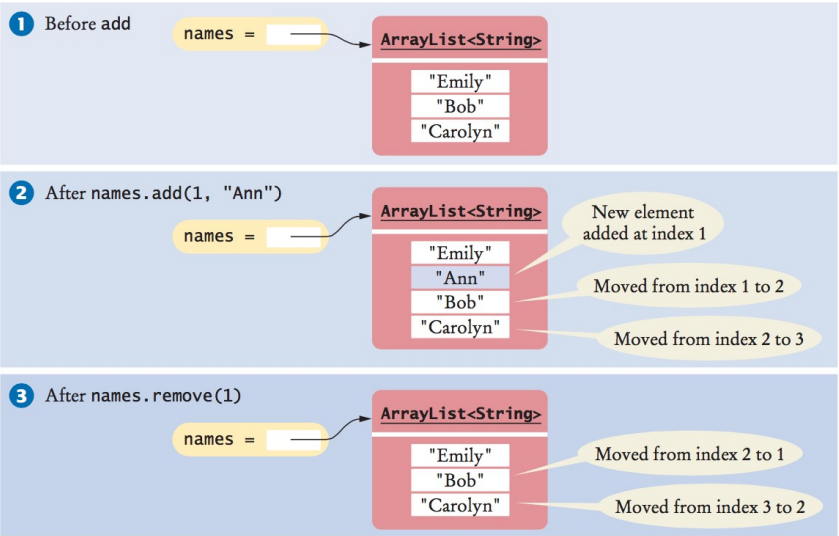


Figure 18 Adding and Removing Elements in the Middle of an Array List

Using the Enhanced for Loop with Array Lists

- You can use the enhanced for loop to visit all the elements of an array list

```
ArrayList<String> names = . . . ;
for (String name : names)
{
   System.out.println(name);
}
```

- This is equivalent to:

```
for (int i = 0; i < names.size(); i++)
{
   String name = names.get(i);
   System.out.println(name);
}
```

Copying Array Lists

- Copying an array list reference yields two references to the same array list.
- After the code below is executed

  Both `names` and `friends` reference the same array list to which the string `"Harry"` was added.

```
ArrayList<String> friends = names;
friends.add("Harry");
```

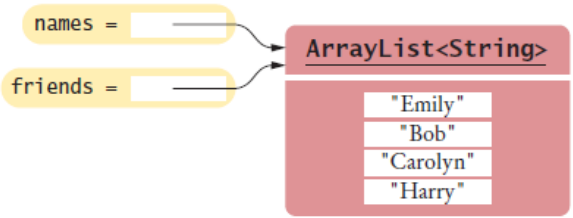


Figure 19 Copying an Array List Reference

- To make a copy of an array list, construct the copy and pass the original list into the constructor:

```
ArrayList<String> newNames = new ArrayList<String>(names);
```