Java Coding 6

Collections

Arrays of Objects

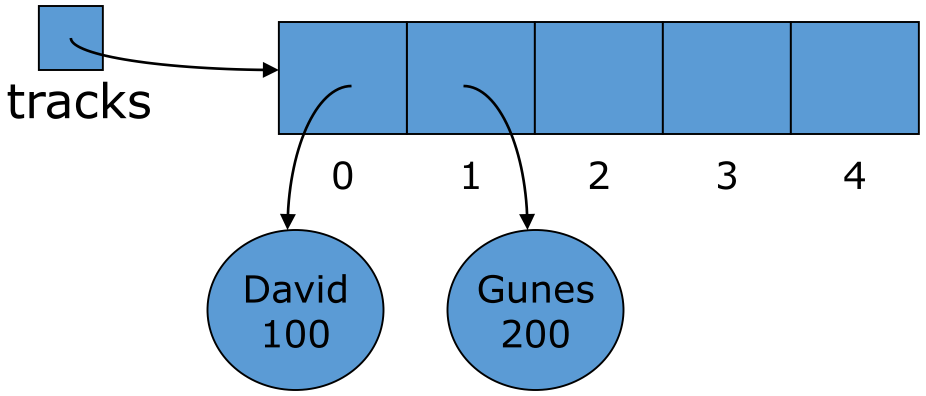
Arrays of objects

- Array contains only references to objects

```
Track[] tracks;
tracks = new Track[5];
```

- Still need to create actual objects

```
tracks[0] = new Track( "David", 100);
tracks[1] = new Track( "Gunes", 200);
```



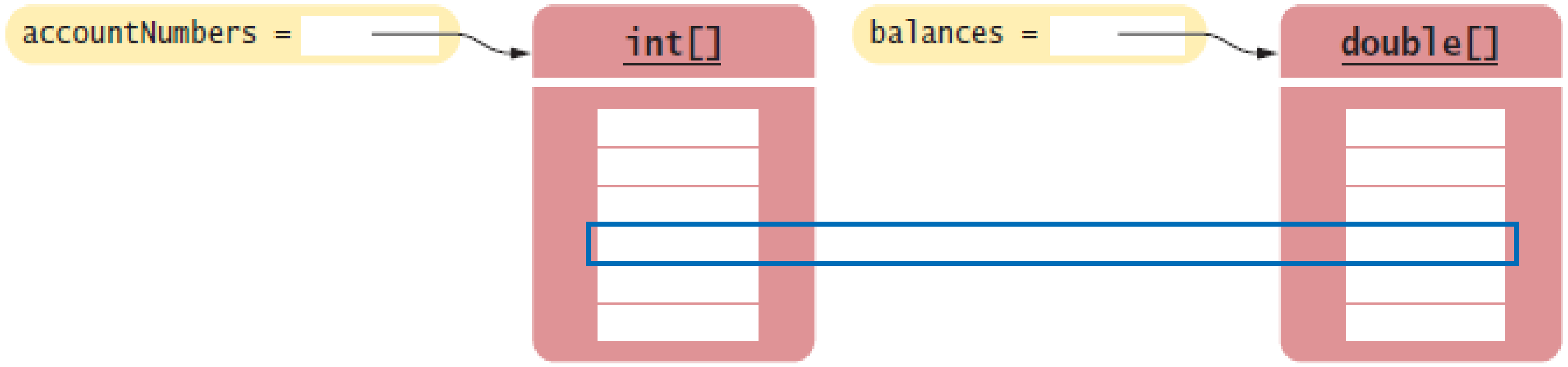
```
tracks[0].getTitle()
tracks[4].getTitle()
```

Make Parallel Arrays into Arrays of Objects

- Don't do this

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

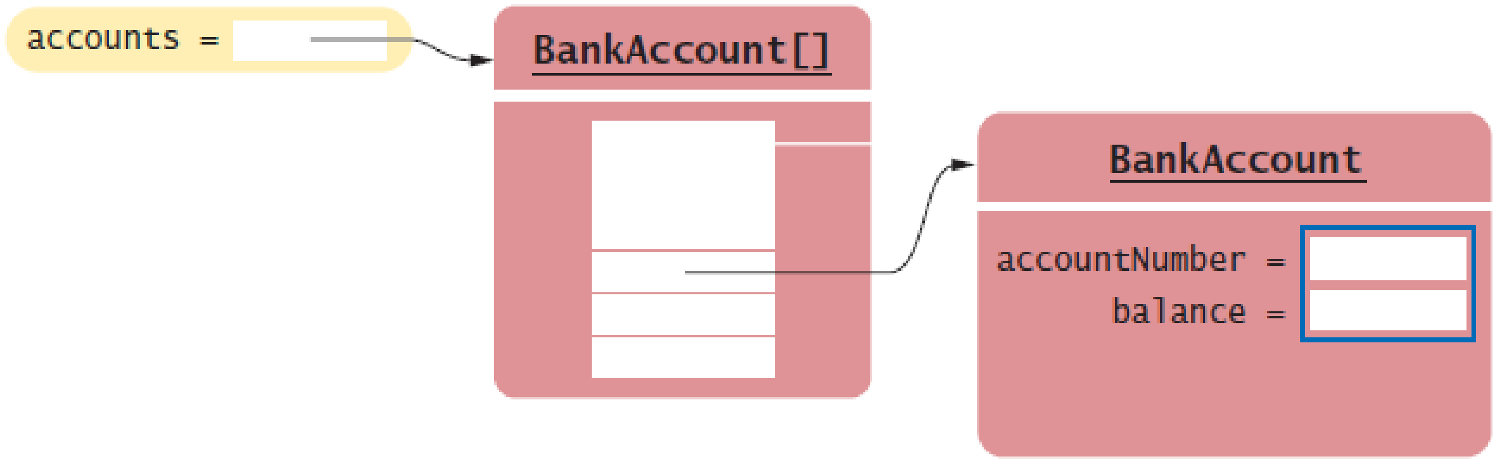
- Don't use parallel arrays



Make Parallel Arrays into Arrays of Objects

Avoid parallel arrays by changing them into arrays of objects:

```
   BankAccount[] accounts;
```



Fig

ArraysofObjects - Example

Date class

Properties: day, month, year

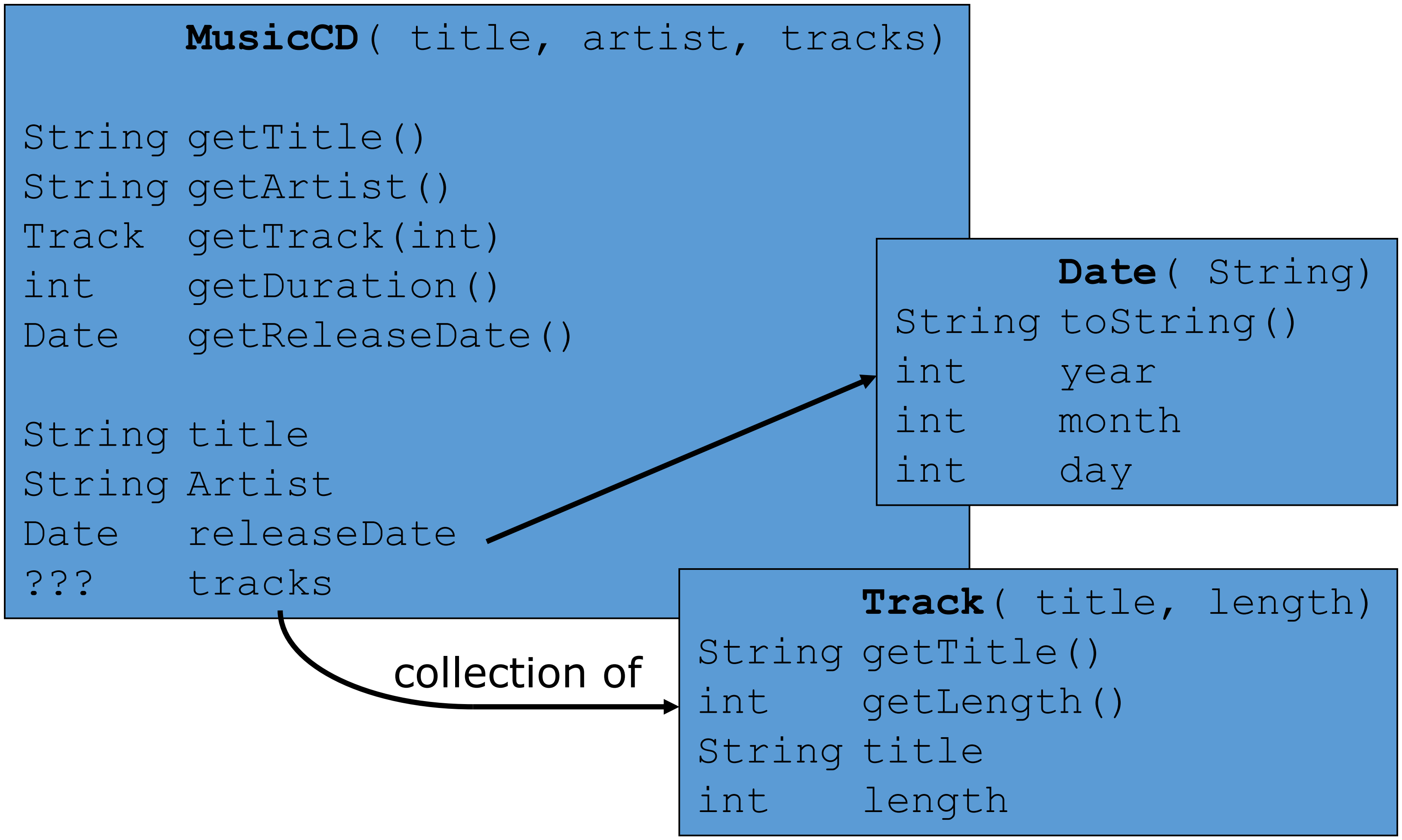
Constructors: copy constructor and others

Methods: get methods, clone, equals, compareTo, toString

Main method
Create an array to keep date objects
Ask the user to enter date objects until a sentinel value
Print the contents of the array

Hints: Make sure array has space
Keep the number of objects in the array, etc.

Inner class Example - MusicCD Class



```
          MusicCD( title, artist, tracks)

String getTitle()
String getArtist()
Track  getTrack(int)
int    getDuration()
Date   getReleaseDate()

String title
String Artist
Date   releaseDate
???    tracks
```

```
          Date( String)
String toString()
int    year
int    month
int    day
```

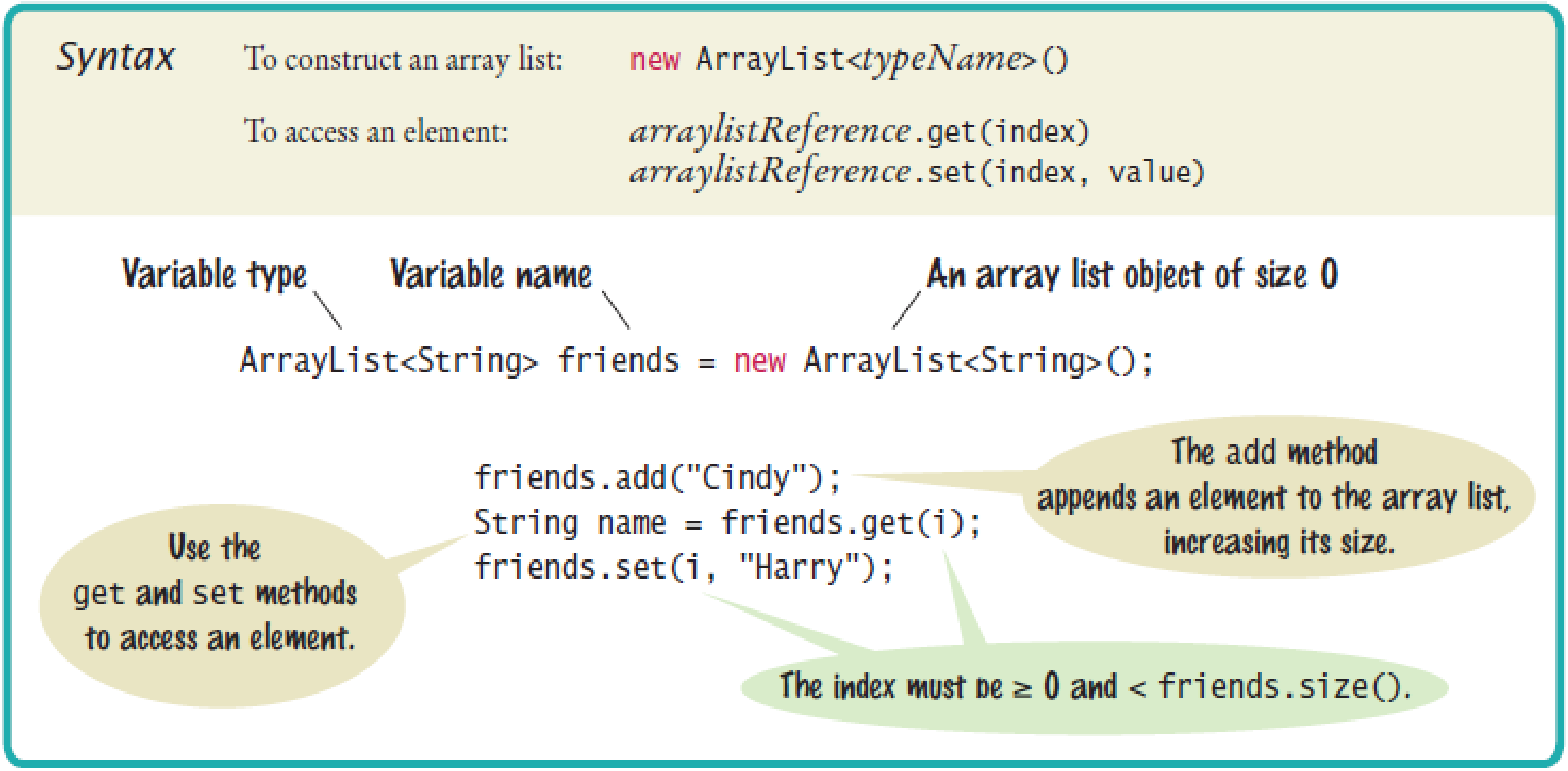
```
          Track( title, length)
String getTitle()
int    getLength()
String title
int    length
```

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.

© digital94086/iStockphoto.

Syntax 6.4 Array Lists


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

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

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

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

Variable type
Variable name
An array list object of size 0
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 Array Lists

- 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

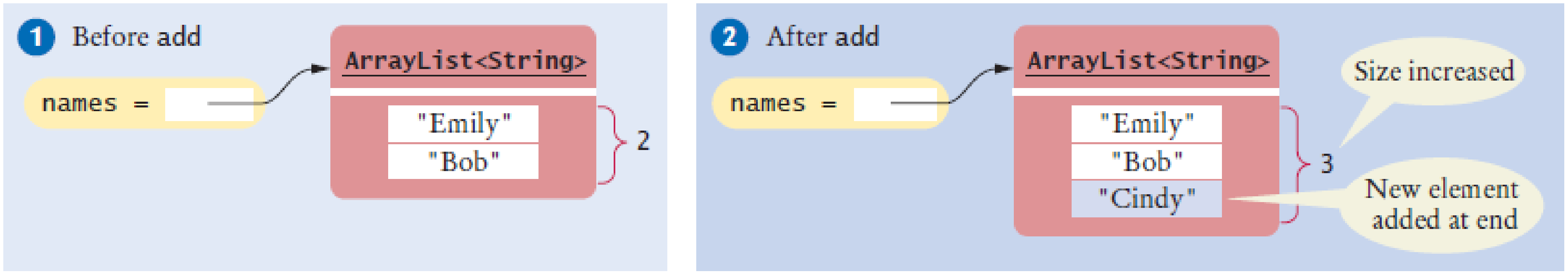
Declaring and Using Array Lists

- `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

Figure 17 Adding an Array List Element with add



Declaring and Using Array Lists

- 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 Array Lists

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

© Danijelm/iStockphoto.

- 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");
```

Declaring and Using Array Lists

- 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 Array Lists

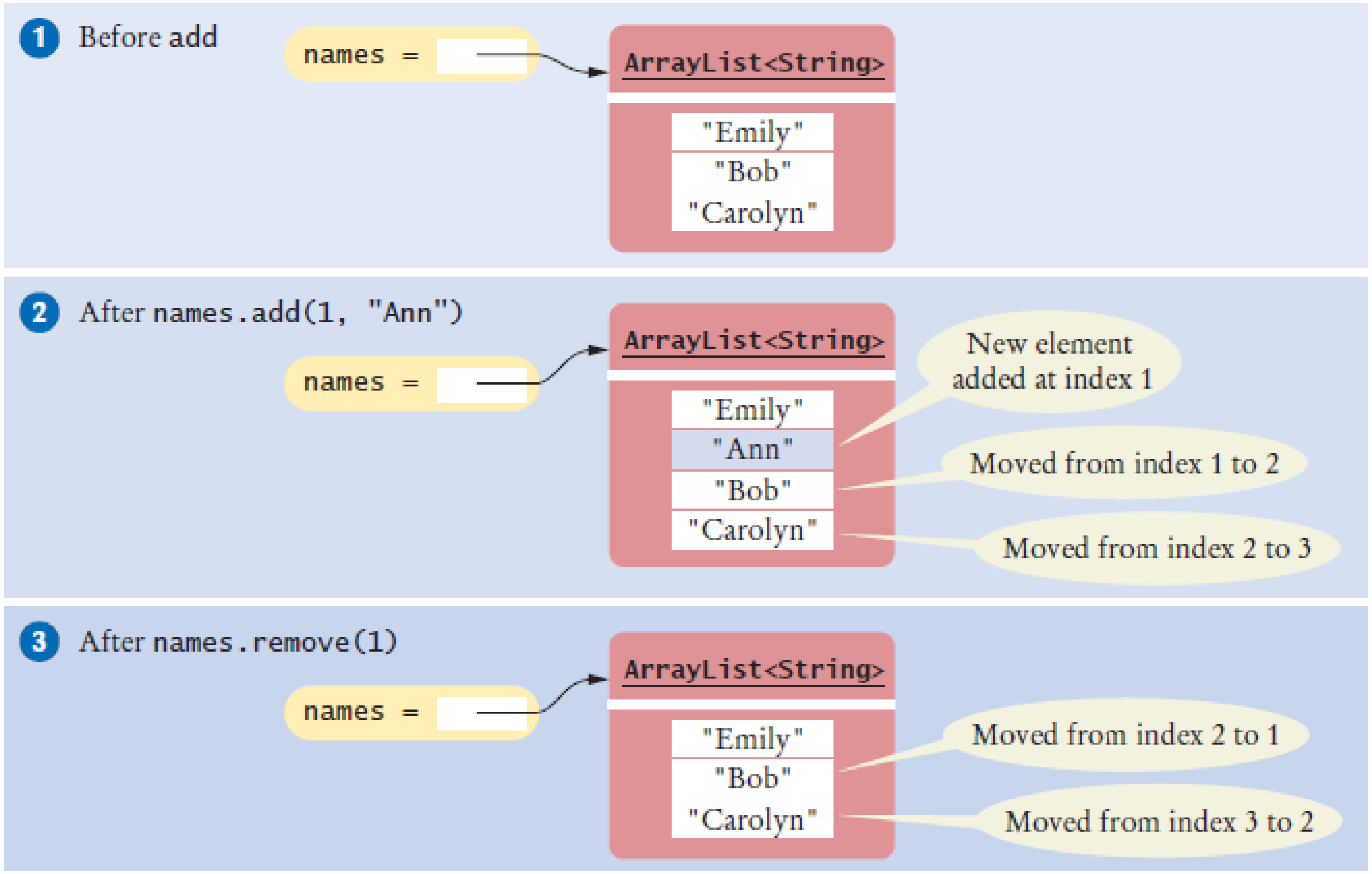


Figure 18 Adding and Removing Elements in the Middle of an Array List

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");
```

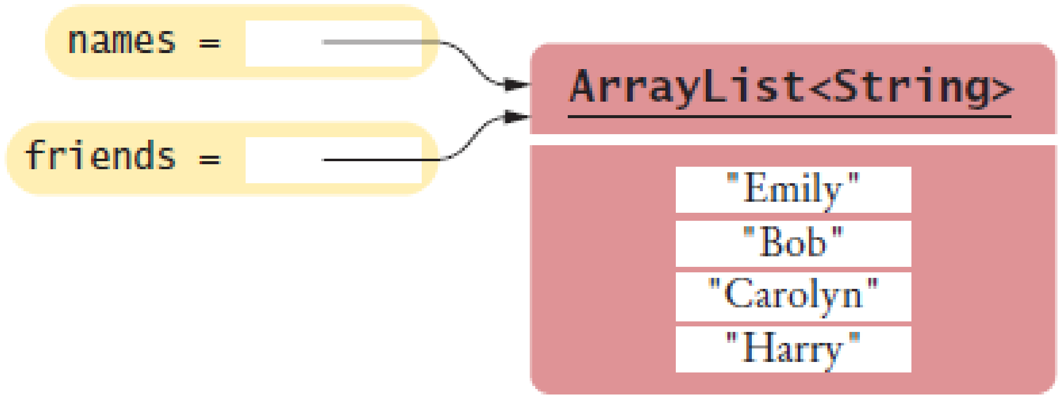


Figure 19 Copying an Array List Reference

Copying Array Lists

- 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);
```

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].
`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.

Wrapper Classes

- **You cannot directly insert primitive type values into array lists.**
- Like truffles that must be in a wrapper to be sold, a number must be placed in a wrapper to be stored in an array list.

© sandoclr/iStockphoto.

- Use the matching wrapper class.

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

Wrapper Classes

- To collect double values in an array list, you use an `ArrayList<Double>`.
- If you assign a `double` value to a `Double` variable, the number is automatically "put into a box"
- Called **auto-boxing:**
  - Automatic conversion between primitive types and the corresponding wrapper classes:

```
Double wrapper = 29.95;
```

  - Wrapper values are automatically "unboxed" to primitive types

```
double x = wrapper;
```

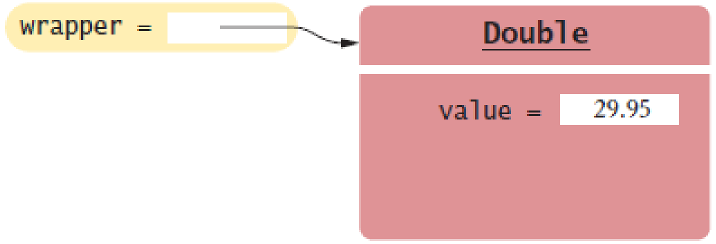


Figure 20 A Wrapper Class Variable

Self Check 6.35

Declare an array list `primes` of integers that contains the first five prime numbers (2, 3, 5, 7, and 11).

Answer:

```
ArrayList<Integer> primes =
   new ArrayList<Integer>();
primes.add(2);
primes.add(3);
primes.add(5);
primes.add(7);
primes.add(11);
```

Self Check 6.36

Given the array list `primes` declared in Self Check 35, write a loop to print its elements in reverse order, starting with the last element.

Answer:

```
for (int i = primes.size() - 1; i >= 0; i--)
{
   System.out.println(primes.get(i));
}
```

Self Check 6.37

What does the array list `names` contain after the following statements?

```
ArrayList<String> names = new ArrayList<String>;
names.add("Bob");
names.add(0, "Ann");
names.remove(1);
names.add("Cal");
```

Answer: `"Ann", "Cal"`

Self Check 6.39

Consider this method that appends the elements of one array list to another:

```
public void append(ArrayList<String> target,
   ArrayList<String> source)
{
   for (int i = 0; i < source.size(); i++)
   {
      target.add(source.get(i));
   }
}
```

What are the contents of `names1` and `names2` after these statements?

```
ArrayList<String> names1 = new ArrayList<String>();
names1.add("Emily");
names1.add("Bob");
names1.add("Cindy");
ArrayList<String> names2 = new ArrayList<String>();
names2.add("Dave");
append(names1, names2);
```

Continued

Self Check 6.39

Answer: `names1` contains `"Emily", "Bob", "Cindy", "Dave"`; `names2` contains "`Dave`"

ArrayListPlay - Play with collections of Date objects

indexOf and contains

- public int indexOf(Object o)
- Returns the index of the first occurrence of the specified element in this list, or -1 if this list does not contain the element
- More formally, returns the lowest index i such that (o==null ? get(i)==null : o.equals(get(i))), or -1 if there is no such index

- public Boolean contains(Object o)
- Returns true if this list contains the specified element
- More formally, returns true if and only if this list contains at least one element e such that (o==null ? e==null : o.equals(e)).

Easy Problem

- Read in a set of positive integer values and then print out a table showing the average, each of the values and their difference from the average.

Umm… must remember all the values we read in in order to print the table.

Could use ArrayList… BUT integers are not Objects!
(use Integer wrapper class)

Example output…

```
Average is 5

Value     Diff
--------------
10        5
3         -2
6         1
1         -4
--------------
```

Easy Problem

- Algorithm

  1. read set of values (how? Fixed number, e.g. 4, ask user how many?, use sentinel?)

  2. compute average of set of values (divide by zero error?)

  3. print table using average & set of values

- Cannot directly store primitive types in ArrayList

  + solution create own wrapper class, MyInt

  + use Java’s wrapper classes Integer, Double, etc.

  + utilise autoboxing/unboxing.

- Java’s wrapper classes have other useful methods, e.g. valueOf to convert string to int or double.

Example

- Write a program that

- Reads a sequence of values and
- Prints them, marking the largest value

section_7/LargestInArrayList.java

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

/**
   This program reads a sequence of values and prints them, marking the largest value.
*/
public class LargestInArrayList
{
   public static void main(String[] args)
   {
      ArrayList<Double> values = new ArrayList<Double>();

      // Read inputs

      System.out.println("Please enter values, Q to quit:");
      Scanner in = new Scanner(System.in);
      while (in.hasNextDouble())
      {
         values.add(in.nextDouble());
      }

```

Continued

section_7/LargestInArrayList.java

```java
        // Find the largest value

        double largest = values.get(0);
        for (int i = 1; i < values.size(); i++)
        {
           if (values.get(i) > largest)
           {
              largest = values.get(i);
           }
        }

        // Print all values, marking the largest

        for (double element : values)
        {
           System.out.print(element);
           if (element == largest)
           {
              System.out.print(" <== largest value");
           }
           System.out.println();
        }
     }
  }
```

Continued

section_7/LargestInArrayList.java

Program Run

```
Please enter values, Q to quit:
35 80 115 44.5 Q
35
80
115 <== largest value
44.5
```

Using Array Algorithms with Array Lists

- The array algorithms can be converted to array lists simply by using the array list methods instead of the array syntax.
- Code to find the largest element in an **array**:

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

- Code to find the largest element in an **array list**:

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

Storing Input Values in an Array List

- To collect an unknown number of inputs, array lists are much easier to use than arrays.
- Simply read the inputs and add them to an array list:

```
ArrayList<Double> inputs = new ArrayList<Double>();
while (in.hasNextDouble())
{
   inputs.add(in.nextDouble());
}
```

Removing Matches

- To remove elements from an array list, call the `remove` method.

```
ArrayList<String> words = ...;
for (int i = 0; i < words.size(); i++)
{
   String word = words.get(i);
   if (word.length() < 4)
   {
      Remove the element at index i.
   }
}
```

Error: skips the element after the moved element

Removing Matches

- Should not increment `i` when an element is removed
- Pseudocode

```
If the element at index i matches the condition
    Remove the element.
Else
    Increment i.
```

Removing Matches

- Use a `while` loop, not a `for` loop

```
int i = 0;
while (i < words.size())
{
   String word = words.get(i);
   if (word.length() < 4) { words.remove(i); }
   else { i++; }
}
```

Choosing Between Array Lists and Arrays

- For most programming tasks, array lists are easier to use than arrays
  - Array lists can grow and shrink.
  - Arrays have a nicer syntax.
- Recommendations
  - If the size of a collection never changes, use an array.
  - If you collect a long sequence of primitive type values and you are concerned about efficiency, use an array.
  - Otherwise, use an array list.

Choosing Between Array Lists and Arrays

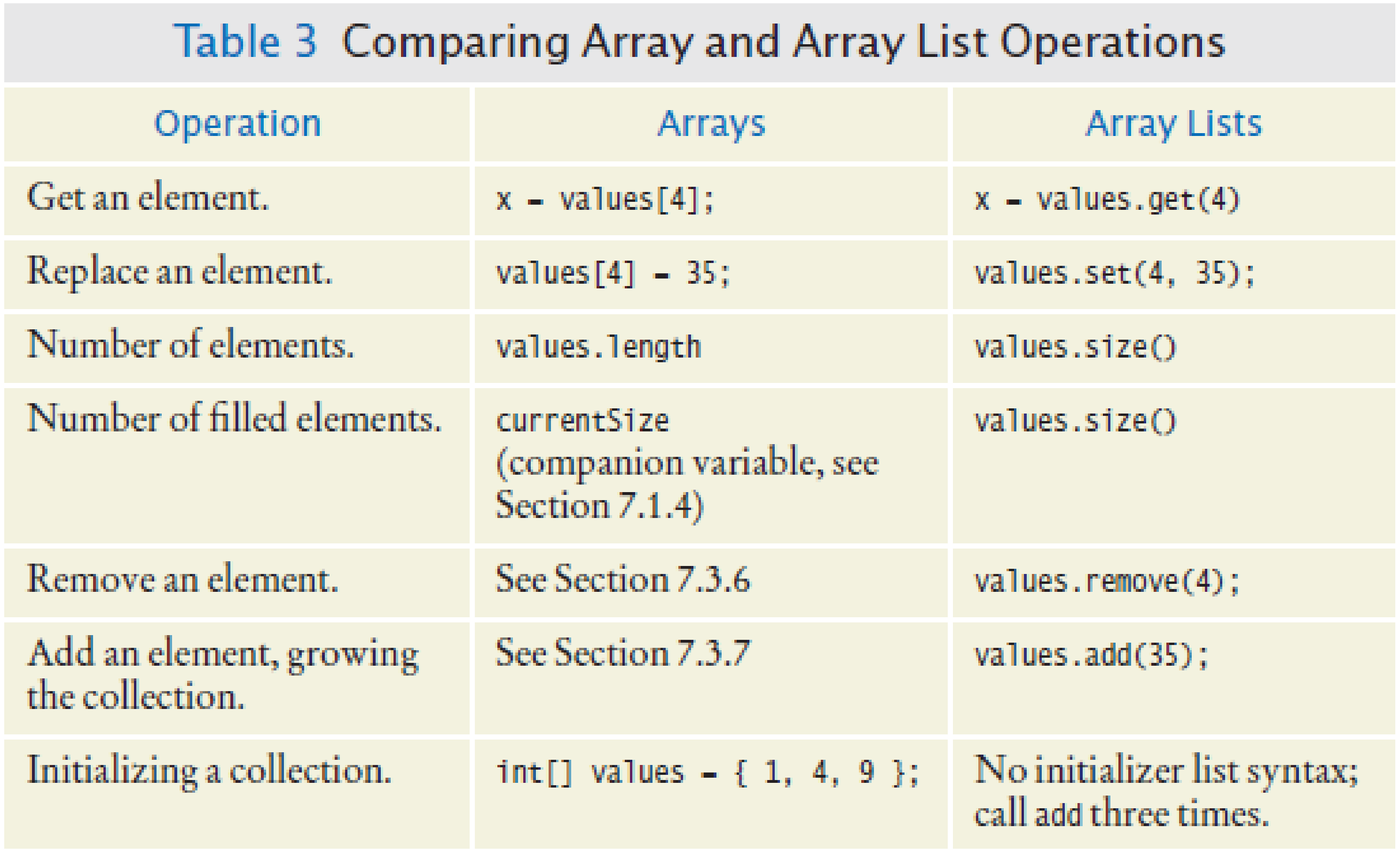


Table 3 Comparing Array and Array List Operations

Operation	Arrays	Array Lists
Get an element.	`x = values[4];`	`x = values.get(4)`
Replace an element.	`values[4] = 35;`	`values.set(4, 35);`
Number of elements.	`values.length`	`values.size()`
Number of filled elements.	`currentSize` (companion variable, see Section 7.1.4)	`values.size()`
Remove an element.	See Section 7.3.6	`values.remove(4);`
Add an element, growing the collection.	See Section 7.3.7	`values.add(35);`
Initializing a collection.	`int[] values = { 1, 4, 9 };`	No initializer list syntax; call `add` three times.