Arrays

- Array Basics
- Arrays in Classes and Methods
- Programming with Arrays and Classes
- Array of object

Motivation

- How to organize 100 Student objects?
- 100 different Student object names?
- Organize data for efficient access
  - Class: different data types in one object
  - Array: multiple objects of the same type

Overview

- An array
  - a single name for a collection of data values
  - all of the same data type
  - subscript notation to identify one of the values
- A carryover from earlier programming languages
- More than a primitive type, less than an object
  - like objects when used as method parameters and return types
  - do not have or use inheritance
- Accessing each of the values in an array
  - Usually a `for` loop

Creating Arrays

- General syntax for declaring an array:

```
Base_Type[] Array_Name = new Base_Type[Length];
```

- Examples:
  80-element array with base type `char`:
  ```
  char[] symbol = new char[80];
  ```

  100-element array of `doubles`:
  ```
  double[] reading = new double[100];
  ```

  70-element array of `Species`:
  ```
  Species[] specimen = new Species[70];
  ```

Three Ways to Use [ ] (Brackets) with an Array Name

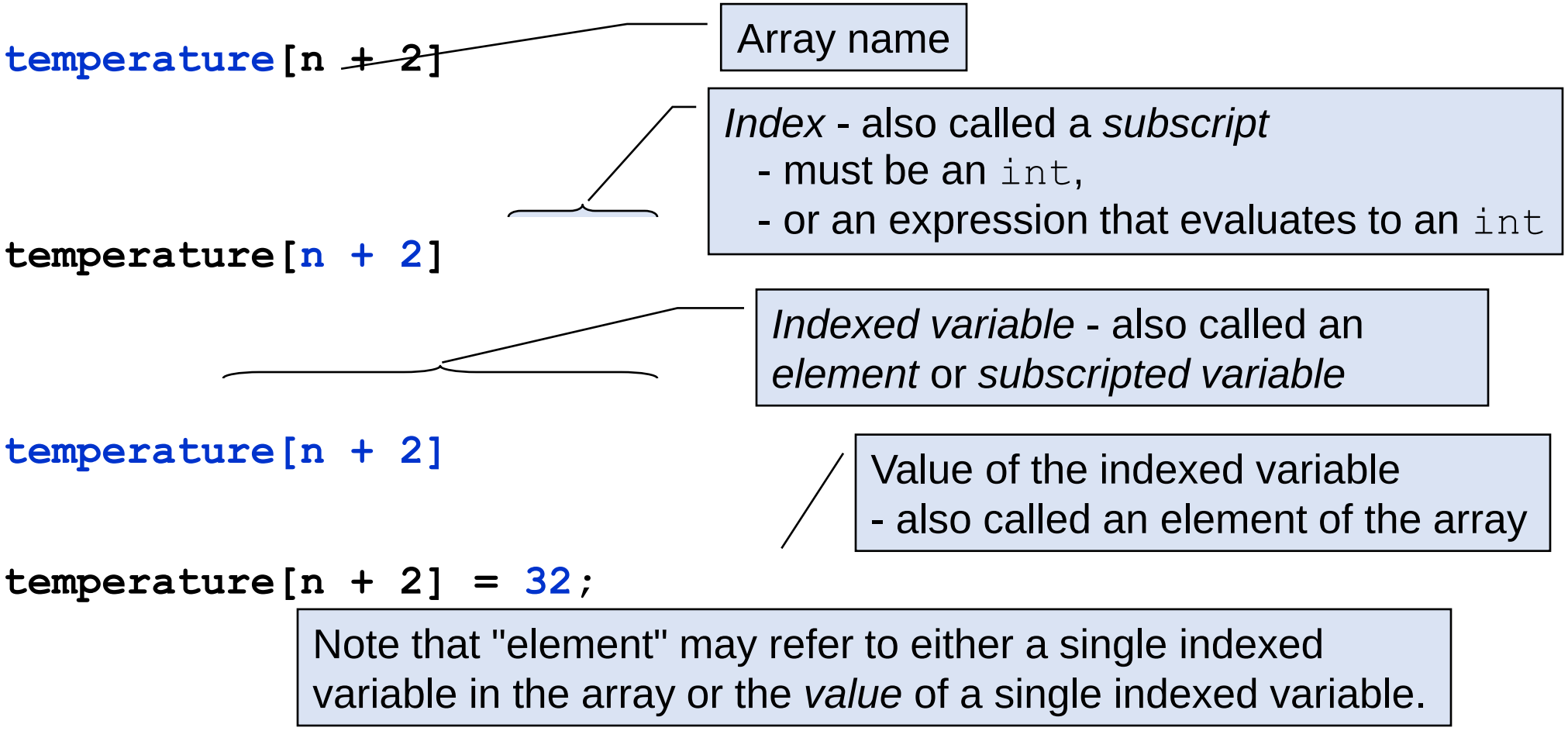
1. Declaring an array: `int[] pressure`
   - creates a name of type "`int` array"
   - types `int` and `int[]` are different
     - `int[]`: type of the array
     - `int` : type of the individual values

2. To create a new array, e.g. `pressure = new int[100];`

3. To refer to a specific element in the array
   - also called *an indexed variable*, e.g.

```
pressure[3] = keyboard.nextInt();
System.out.println("You entered" + pressure[3]);
```

Some Array Terminology



Array Length

- Specified by the number in brackets when created with `new`
  - *maximum* number of elements the array can hold
  - storage is allocated whether or not the elements are assigned values

- the attribute `length`,

```
Species[] entry = new Species[20];
System.out.println(entry.length);
```

- The `length` attribute is established in the declaration and cannot be changed unless the array is redeclared

Subscript Range

- Array subscripts use zero-numbering
  - the first element has subscript 0
  - the second element has subscript 1
  - etc. - the n^{th} element has subscript n-1
  - the last element has subscript `length-1`
- For example: an int array with 4 elements

Subscript:	0	1	2	3
Value:	97	86	92	71

Subscript out of Range Error

- Using a subscript larger than `length-1` causes a *run time* (not a compiler) error
  - an `ArrayOutOfBoundsException` is thrown
    - you do not need to catch it
    - you need to fix the problem and recompile your code
- Other programming languages, e.g. C and C++, do not even cause a run time error!
  - one of the most dangerous characteristics of these languages is that they allow out of bounds array indices.

Array Length Specified at Run-time

```
// array length specified at compile-time
int[] array1 = new int[10];


// array length specified at run-time
// calculate size…
int size = …;
int[] array2 = new int[size];
```

Programming Tip: Use Singular Array Names

- Using singular rather than plural names for arrays improves readability

- Although the array contains many elements the most common use of the name will be with a subscript, which references a *single* value.
- It is easier to read:
  - `score[3]` than
  - `scores[3]`

Initializing an Array's Values in Its Declaration

- can be initialized by putting a comma-separated list in braces
- Uninitialized elements will be assigned some default value, e.g. 0 for `int` arrays (**explicit initialization** is recommended)
- The length of an array is automatically determined when the values are explicitly initialized in the declaration
- For example:

```
double[] reading = {5.1, 3.02, 9.65};
System.out.println(reading.length);
```

- displays `3`, the length of the array `reading`

Initializing Array Elements in a Loop

- A `for` loop is commonly used to initialize array elements
- For example:

```
int i;//loop counter/array index
int[] a = new int[10];
for(i = 0; i < a.length; i++)
   a[i] = 0;
```

  - note that the loop counter/array index goes from `0` to `length - 1`
  - it counts through `length = 10` iterations/elements using the zero-numbering of the array index

Programming Tip:

Do not count on default initial values for array elements

- explicitly initialize elements in the declaration or in a loop

Arrays, Classes, and Methods

An array of a class can be declared and the class's methods applied to the elements of the array.

This excerpt from the Sales Report program in the text uses the `SalesAssociate` class to create an array of sales associates:

create an array of `SalesAssociate`s

each array element is a `SalesAssociate` variable

use the `readInput` method of `SalesAssociate`

```
public void getFigures()
{
   System.out.println("Enter number of sales associates:");
   numberOfAssociates = input.readLineInt();
   SalesAssociate[] record =
            new SalesAssociate[numberOfAssociates];
   for (int i = 0; i < numberOfAssociates; i++)
   {
      record[i] = new SalesAssociate();
      System.out.println("Enter data for associate " + (i + 1));
      record[i].readInput();
      System.out.println();
   }
}
```

Arrays and Array Elements as Method Arguments

- Arrays and array elements can be
  - used with classes and methods just like other objects
  - be an argument in a method
  - returned by methods

Indexed Variables as Method Arguments

`nextScore` is an array of `int`s

an element of `nextScore` is an argument of method `average`

`average` method definition

```
public static void main(String[] arg)
  {
     Scanner keyboard = new Scanner(System.in);a
     System.out.println("Enter your score on exam 1:");
     int firstScore = input.nextInt();
     int[ ] nextScore = new int[3];
     int i;
     double possibleAverage;
     for (i = 0; i < nextScore.length; i++)
        nextScore[i] = 80 + 10*i;
     for (i = 0; i < nextScore.length; i++)
     {
        possibleAverage = average(firstScore, nextScore[i]);
        System.out.println("If your score on exam 2 is "
                           + nextScore[i]);
        System.out.println("your average will be "
                           + possibleAverage);
     }
  }
  public static double average(int n1, int n2)
  {
    return (n1 + n2)/2.0;
  }
```

Passing Array Elements

```
public static void main(String[] arg)
 {
    SalesAssociate[] record = new SalesAssociate[numberOfAssociates];
    int i;
    for (i = 0; i < numberOfAssociates; i++)
    {
       record[i] = new SalesAssociate();
       System.out.println("Enter data for associate " + (i + 1));
       record[i].readInput();
    }
    m(record[0]);
 }
public static void m(SalesAssociate sa)
{
}
```

When Can a Method Change an Indexed Variable Argument?

- primitive types are “call-by-value”
  - only a copy of the value is passed as an argument
  - method *cannot* change the value of the indexed variable
- class types are reference types (“call by reference”)
  - pass the address of the object
  - the corresponding parameter in the method definition becomes an alias of the object
  - the method has access to the actual object
  - so the method *can* change the value of the indexed variable if it is a class (and not a primitive) type

Passing Array Elements

```
int[] grade = new int[10];
obj.method(grade[i]);  // grade[i] cannot be changed

… method(int grade)    // pass by value; a copy
{
}
```

```
Person[] roster = new Person[10];
obj.method(roster[i]); // roster[i] can be changed

… method(Person p)     // pass by reference; an alias
{
}
```

Array Names as Method Arguments

- Use just the array name and no brackets
- Pass by reference
  - the method has access to the original array and can change the value of the elements
- The length of the array passed can be different for each call
  - when you define the method you do not need to know the length of the array that will be passed
  - use the `length` attribute inside the method to avoid `ArrayIndexOutOfBoundsException`s

Example: An Array as an Argument in a Method Call

```
public static void showArray(char[] a)
{
   int i;
   for(i = 0; i < a.length; i++)
      System.out.println(a[i]);
}


-------------
char[] grades = new char[45];
MyClass.showArray(grades);
```

the method's argument is the name of an array of characters

uses the `length` attribute
to control the loop
allows different size arrays
and avoids index-out-of-bounds exceptions

Arguments for the Method **`main`**

- The heading for the `main` method shows a parameter that is an array of `String`s:

  ```
  public static void main(String[] arg)
  ```

- When you run a program from the command line, all words after the class name will be passed to the main method in the arg array.

  ```
  java TestProgram CSC113 Student
  ```

- The following `main` method in the class `TestProgram` will print out the first two arguments it receives:

```
Public static void main(String[] arg)
{
   System.out.println("Hello " + arg[0] + " " + arg[1]);
}
```

- In this example, the output from the command line above will be:

  ```
  Hello CSC113 Student
  ```

Using = with Array Names: Remember They Are Reference Types

```
int[] a = new int[3];
int[] b = new int[3];
for(int i=0; i < a.length; i++)
   a[i] = i;
b = a;
System.out.println(a[2] + " " + b[2]);
a[2] = 10;
System.out.println(a[2] + " " + b[2]);
```

The output for this code will be:

```
2 2
10 10
```

This does not create a copy of array `a`; it makes `b` another *name* for array `a`.

A value changed in `a` is the same value obtained with `b`

Using == with array names: remember they are reference types

```
int i;
int[] a = new int[3];
int[] b = new int[3];
for(i=0; i < a.length; i++)
   a[i] = 0;
for(i=0; i < b.length; i++)
   b[i] = 0;
if(b == a)
   System.out.println("a equals b");
else
   System.out.println("a does not equal b");
```

`a` and `b` are both 3-element arrays of `int`

all elements of `a` and `b` are assigned the value `0`

tests if the *addresses* of `a` and `b` are equal, not if the array values are equal

The output for this code will be "`a does not equal b`" because the *addresses* of the arrays are not equal.

Behavior of Three Operations

	Primitive Type	Class Type	Entire Array	Array Element
Assignment (=)	Copy content	Copy address	Copy address	Depends on primitive/ class type
Equality (==)	Compare content	Compare address	Compare address	Depends on primitive/ class type
Parameter Passing	*Pass by value* (content)	*Pass by reference* (address)	*Pass by reference* (address)	Depends on primitive/ class type

Testing Two Arrays for Equality

- To test two arrays for equality you need to define an `equals` method that returns true if and only the arrays have the same length and all corresponding values are equal

```
public static boolean equals(int[] a,
                             int[] b)
   {
       boolean match = false;
       if (a.length == b.length)
       {
           match = true;
           int i = 0;
           while (match && (i < a.length))
           {
               if (a[i] != b[i])
                   match = false;
               i++;
           }
       }
       return match;
    }
```

Methods that Return an Array

- the address of the array is passed
- The local array name within the method is just another name for the original array

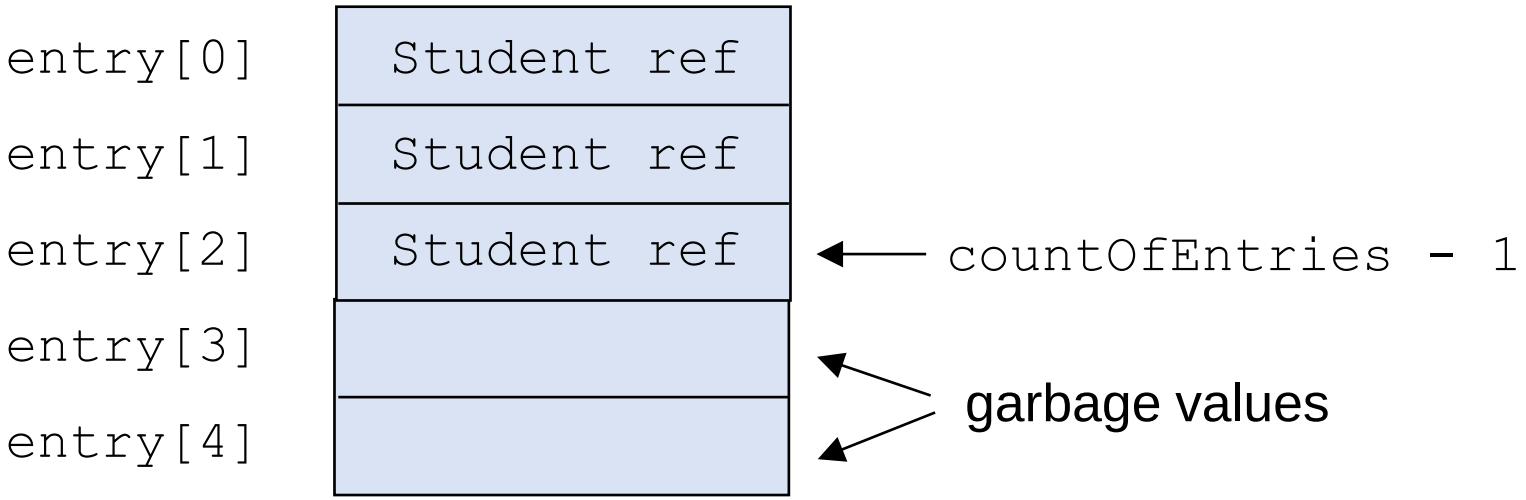
```
public class returnArrayDemo
  {
      public static void main(String arg[])
      {
          char[] c;
          c = vowels();
          for(int i = 0; i < c.length; i++)
             System.out.println(c[i]);
       }
       public static char[] vowels()
       {
            char[] newArray = new char[5];
            newArray[0] = 'a';
            newArray[1] = 'e';
            newArray[2] = 'i';
            newArray[3] = 'o';
            newArray[4] = 'u';
            return newArray;
       }
   }
```

`c`, `newArray`, and the return type of `vowels` are all the same type: `char []`

Partially Filled Arrays

- Sometimes only part of an array has been filled with data
- Array elements always contain something
  - elements which have not been written to
    - contain unknown (*garbage*) data so you should avoid reading them
- There is no automatic mechanism to detect how many elements have been filled
  - *you*, the programmer need to keep track!
- An example: the instance variable `countOfEntries` (in the class `OneWayNoRepeatsList`) is incremented every time `addItem` is called (see the text)

Example of a Partially Filled Array



`countOfEntries` has a value of `3`.

`entry.length` has a value of `5`.

Searching an Array

- There are many techniques for searching an array for a particular value

- *Sequential search*:
  - start at the beginning of the array and proceed in sequence until either the value is found or the end of the array is reached*
    - if the array is only partially filled, the search stops when the last meaningful value has been checked
  - it is not the most efficient way
  - but it works and is easy to program

* Or, just as easy, start at the end and work backwards toward the beginning

Example: Sequential Search of an Array

The `onList` method of `OneWayNoRepeatsList` sequentially searches the array `entry` to see it the parameter `item` is in the array

```
public boolean onList(String item)
{
      boolean found = false;
      int i = 0;
      while ((! found) &&
              (i < countOfEntries))
      {
          if (item.equals(entry[i]))
              found = true;
          else
              i++;
      }

     return found;
}
```

Returning an Array Attribute (Instance Variable)

- Access methods that return references to array instance variables cause problems for information hiding.

  Example:

```
class …
{
  private String[] entry;
  …
  public String[] getEntryArray()
  {
        return entry;
  }
```

  Even though `entries` is declared private, a method outside the class can get full access to it by using `getEntryArray`.

- In most cases this type of method is not necessary anyhow.
- If it is necessary, make the method return a copy of the array instead of returning a reference to the actual array.

Sorting an Array

- Sorting a list of elements is another very common problem (along with searching a list)
  - sort numbers in ascending order
  - sort numbers in descending order
  - sort strings in alphabetic order
  - etc.

- There are many ways to sort a list, just as there are many ways to search a list

Summary

- An array may be thought of as a collection of variables, all of the same type.
- An array is also may be thought of as a single object with a large composite value of all the elements of the array.
- Arrays are objects created with `new` in a manner similar to objects discussed previously.

Summary

- An array element can be used as an argument to a method any place the base type is allowed:
  - if the base type is a primitive type, the method cannot change the array element;
  - if the base type is a class, the method *can* change the array element.
- When you want to store two or more different values (possibly of different data types) for each index of an array,
  - parallel arrays (multiple arrays of the same length)
  - use a class that have multiple types/values.
- An accessor method that returns an array corresponding to a private instance variable of an array type should be careful to return a copy of the array, and not return the private instance variable itself (like any object).

Summary

- Array indexes use zero-numbering:
  - they start at 0, so index *i* refers to the *(i+1)th* element;
  - the index of the last element is `(length-of-the-array - 1)`.
  - Any index value outside the valid range of 0 to length-1 will cause an **array index out of bounds error** when the program *runs.*
- A method may return an array.
- A "partially filled array" is one in which values are stored in an initial segment of the array:
  - use an `int` variable to keep track of how many variables are stored.