ARRAYS

(Chapter 5)

Arrays

How to create and handle a very large number of data items?

```
public static void main(String[] args)
{
        // maybe declare 7 variables to hold readings?
        double temperature1, temperature2, temperature3,
               temperature4,  temperature5, temperature6,
               temperature7;
        // more code will go here
}
```

How to enter 7 temperature readings?

Try using a **for** loop?

```
for (int i = 1; i <= 7; i++)
{
    // what goes here?
}
```

- No obvious instruction we could write in the for loop that will allow a value to be entered into a different variable each time the loop repeats
  - **Each variable has a distinct name**

What is an array?

- An array is a data type that stores **a collection of items**
- These items are sometimes referred to as the elements of the array
- All elements must be of the **same type** BUT there is no restriction on which type this is. For example,
  - Arrays can be used to hold a collection of `int` values
  - Or a collection of `char` values
  - But they cannot be used to hold a mixture of `int` and `char` values

How to create an array?

- It is a two-stage process:
  - Declare an array variable
  - Allocate memory to store the array elements

Declaring an array variable

- In declaration, we use square brackets that indicate the variable is an array allowing many elements of the same type to be stored

- To declare an array to hold a collection of integer variables

```
int[] someArray;
```

- To declare an array temperature containing double values

```
double[] temperature;
```

Allocating memory to store the array elements

```
double[] temperature;
```

- It defines temperature to be a variable. However, the memory that will eventually hold these double values has not been allocated yet.
- In allocation, it is necessary to state
  - the **size** of the array
  - the **type** of each individual array element

Allocating memory to store the array elements

```
arrayName = new int[10];
```

- **new operator** creates the space in memory for an array of the given size and element type
  - The array size should not be negative
  - Once the size of the array is set it cannot be changed

Returning to the temperature array

- The temperature array holds seven double values

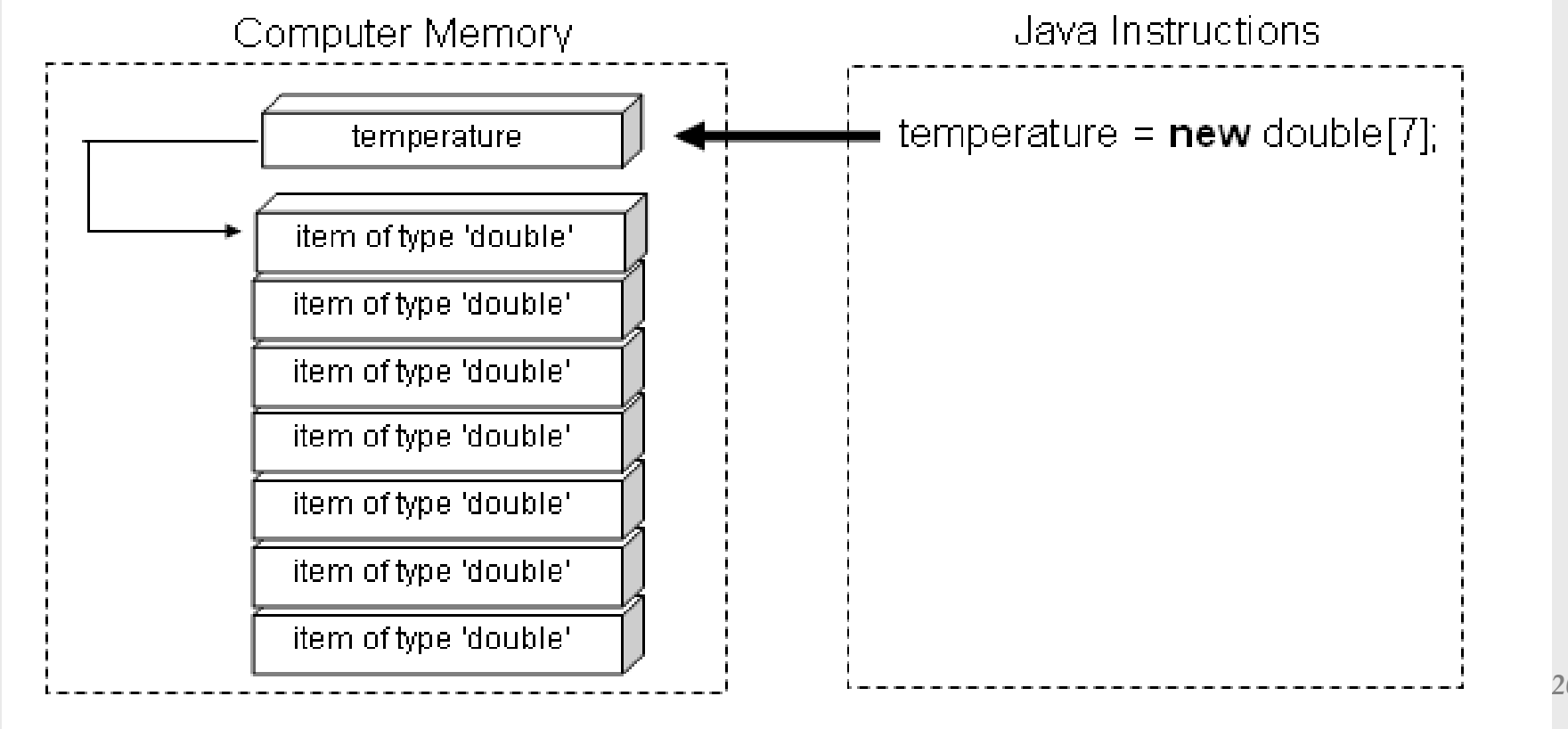
```
double[] temperature;
temperature = new double[7];
```

- The two stages of array creation can be combined into one step as follows:

```
double[] temperature = new double[7];
```

The effect on computer memory of declaring an array

Array allocation has created seven elements and linked the temperature variable to these seven elements. The temperature variable is said to hold a **reference** to the array elements.



Naming the array elements

- The individual elements are uniquely identified by an index number
- In Java, array indices start from 0 and not from 1
  - The first element in the temperature array is temperature[0], the second element is temperature[1], the last element is temperature[6]
- If you access an invalid element (such as temperature[7]), the system generates the following program error

```
java.lang.ArrayIndexOutOfBoundsException
```

Initializing an array

```
double[] temperature = {9, 11.5, 11, 8.5, 7, 9, 8.5} ;
```

- This is the only instance in which all the elements of an array can be assigned explicitly by listing out the elements in a single assignment statement
- Once an array has been created, elements must be accessed individually

Accessing array elements

- Array elements can be used like any other variable of the given type in Java
  - You must specify which element to be used

- For example,

```
System.out.println(temperature[5]);

for (int i = 0; i < 7; i++){
    System.out.println
            ("max temp for day " + (i+1));
    temperature[i] = sc.nextDouble();
}
```

The length attribute

- **The length attribute** returns the size of an array

```
for (int i = 0; i < temperature.length; i++){
    System.out.println
            ("max temp for day " + (i+1));
    temperature[i] = sc.nextDouble();
}
```

Passing arrays as parameters

- Arrays can be used both as parameters and as return values

- Suppose, we want to write a program that uses two helper methods to take 7 temperatures from the user and display them on the screen, respectively

```
import java.util.*;

public class TemperatureReadings2{

   private static void displayTemps(double[] T){
      for (int i = 0; i < T.length; i++)
         System.out.println
                  ("day " + (i+1) + ":  " + T[i]);
   }

   private static void enterTemps(double[] T){
      Scanner sc = new Scanner(System.in);
      for (int i = 0; i < T.length; i++){
         System.out.println
                  ("temperature for day " + (i+1));
         T[i] = sc.nextDouble();
      }
   }
```

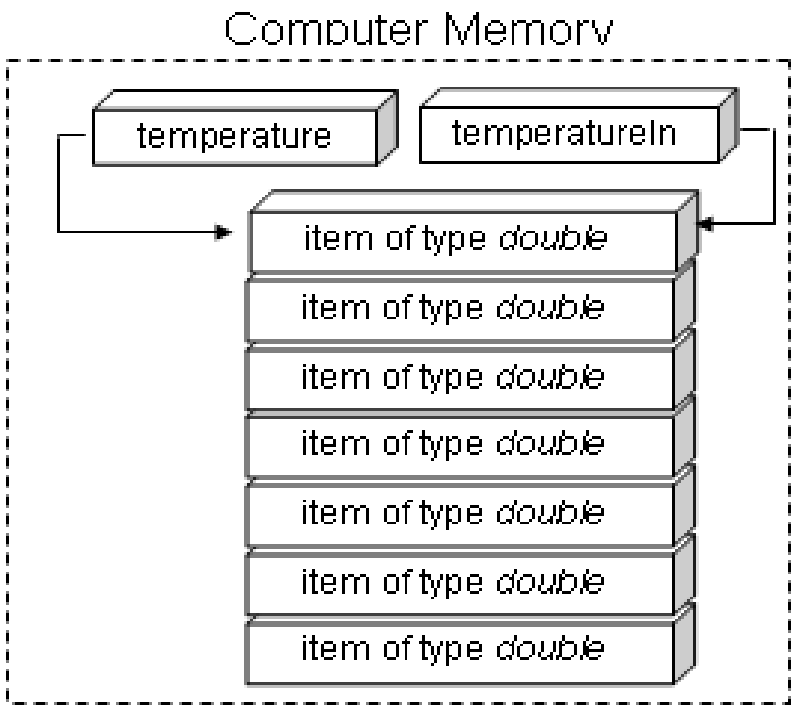
```
    public static void main(String[] args){
        double[] temperature = new double[7];

        enterTemps(temperature);
        displayTemps(temperature);
    }
}
```

The effect on computer memory of passing an array as a parameter

- We mentioned that a parameter just receives a copy of the original variable
  - The original variable is unaffected outside a method even though the corresponding parameter is altered within that method
- So how can enterTemps method fill in the original array?

The effect on computer memory of passing an array as a parameter

- In the case of arrays, the value sent as a parameter is not a copy of each array element, but instead a copy of the array reference



Java Instructions

```
main(String[] args)
{
      // call method with local array
      enterTemps(temperature);
}

// method receives array reference
enterTemps(double[] temperatureIn)
```

What is the output?

```
public class arrayParameters{
    public static void main(String[] args){
        double[] temperature = {5, 5, 5};
        System.out.println("before enterTemps");
        displayTemps(temperature);
        enterTemps(temperature);
        System.out.println("after enterTemps");
        displayTemps(temperature);
    }
    private static void enterTemps(double[] T){
        for (int i = 0; i < T.length; i++)
            T[i] = i;
        System.out.println("in enterTemps");
        displayTemps(T);
    }
    private static void displayTemps(double[] T){
        for (int i = 0; i < T.length; i++)
            System.out.println(T[i]);
    }
```

What is the output?

```
before enterTemps
5.0
5.0
5.0
in enterTemps
0.0
1.0
2.0
after enterTemps
0.0
1.0
2.0
```

What is the output?

```
public class arrayParameters{
    public static void main(String[] args){
        double[] temperature = {5, 5, 5};
        System.out.println("before enterTemps");
        displayTemps(temperature);
        enterTemps(temperature);
        System.out.println("after enterTemps");
        displayTemps(temperature);
    }
    private static void enterTemps(double[] T){
        T = new double[4];
        for (int i = 0; i < T.length; i++)
            T[i] = i;
        System.out.println("in enterTemps");
        displayTemps(T);
    }
    private static void displayTemps(double[] T){
        for (int i = 0; i < T.length; i++)
            System.out.println(T[i]);
    }
}
```

What is the output?

```
before enterTemps
5.0
5.0
5.0
in enterTemps
0.0
1.0
2.0
3.0
after enterTemps
5.0
5.0
5.0
```

Returning an array from a method

- A method can return an array as well
- Suppose that we want to create an array with the specified size within a method and fill this array with values

```
private static double[] enterValues(int no){
    double[] T = new double[no];
    for (int i = 0; i < T.length; i++)
        T[i] = i;
    return T;
}
```

How to call it?

- We need to modify the caller method so that the returned array value is used to set the value of the original array

```
double[] myArray;
myArray = enterValues(7);
```

The enhanced "for" loop

- The enhanced for loop iterates through elements of an array without the need for an array index
- For example, suppose that we want to display on the screen each value from the temperature array
- Java 5.0 provides an enhanced version of the for loop

```
for (double item: temperature)
    System.out.println(item);
```

The loop header is to be read as "for each item in the temperature array"

- Which is equivalent to

```
for (int i = 0; i < temperature.length; i++)
    System.out.println(temperature[i]);
```

When to use the enhanced "for" loop?

- You should use an enhanced for loop **only when**
  - You wish to access the entire array (and not just part of the array)
  - You wish to read the elements in the array, not modify them
  - You do not require the array index for additional processing

Some useful array methods

- Apart from the length feature, an array does not come with any useful built in routines
- We will develop some of our own methods for processing an array
- In this example, we will use a simple integer array

```
import java.util.*;
public class arrayMethods{
   public static void main(String[] args){
      Scanner sc = new Scanner(System.in);
      int [] someArray = read();
      System.out.println("Max is " + max(someArray));
      System.out.println("Sum is " + sum(someArray));

      System.out.print("Enter a value: ");
      int value = sc.nextInt();
      if (contains(someArray,value))
         System.out.println(value + " is in the array");
      else
         System.out.println(value + " is not in the array");
      int index = search(someArray,value);
      if (index == -1)
         System.out.println(value + " is not in the array");
      else
         System.out.println(value + " is  at index " + index);
   }
   // implement the helper methods here
   // ...
}
```

```
// it reads the elements of an array from the user
private static int [] read(){
    Scanner sc = new Scanner(System.in);
    int [] A;
    int size;

    System.out.print("Enter the size: ");
    size = sc.nextInt();

    A = new int[size];

    for (int i = 0; i < A.length; i++){
        System.out.print("Enter a value: ");
        A[i] = sc.nextInt();
    }
    return A;
}
```

```
// it finds the maximum element in an array
private static int max(int [] A){
    int result = A[0];

    for (int item: A)
        if (item > result)
            result = item;
    return result;
}


// it finds the sum of the elements in an array
private static int sum(int [] A){
    int total = 0;

    for (int item: A)
        total += item;
    return total;
}
```

```
// it checks whether or not an item is in an array
private static boolean contains(int [] A, int value){

    // enhanced "for" loop can be used here
    for (int item: A)
        if (item == value)
            return true;
    return false;
}


// it returns the position of an item; if the item
// is not in the array it returns -1
private static int search(int [] A, int value){

    // enhanced "for" loop should not be used here
    for (int i = 0; i < A.length; i++)
        if (A[i] == value)
            return i;
    return -1;
}
```