Arrays and ArrayList

Lecture 8 - B

Object-Oriented Programming

Array Basics

- An array is a collection of data values.
- If your program needs to deal with 100 integers, 500 Account objects, 365 real numbers, etc., you will use an array.
- In Java, an array is an indexed collection of data values of the same type.

Arrays of Primitive Data Types

- Array Declaration

```
<data type> [ ] <variable>            //variation 1
<data type>   <variable>[ ]   //variation 2
```

- Array Creation

```
<variable>  = new <data type> [ <size> ]
```

- Example

Variation 1

```
double[ ] rainfall;
rainfall
    = new double[12];
```

Variation 2

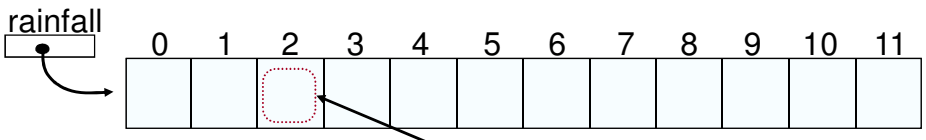
```
double rainfall [ ];
rainfall
    = new double[12];
```

An array is like an object!

Accessing Individual Elements

- Individual elements in an array accessed with the indexed expression.

```
double[] rainfall = new double[12];
```



rainfall[2]

This indexed expression refers to the element at position #2

The index of the first position in an array is 0.

Array Processing – Sample1

```
double[] rainfall = new double[12];

double   annualAverage,
         sum = 0.0;

for (int i = 0; i < rainfall.length; i++) {

    rainfall[i] = Double.parseDouble(
                      JOptionPane.showinputDialog(null,
                      "Rainfall for month " + (i+1) ) );
    sum += rainfall[i];
}

annualAverage = sum / rainfall.length;
```

The public constant length returns the capacity of an array.

Array Processing – Sample 2

- Compute the average rainfall for each quarter.

```
//assume rainfall is declared and initialized properly

double[] quarterAverage = new double[4];

for (int i = 0; i < 4; i++) {
    sum = 0;
    for (int j = 0; j < 3; j++) {
                                        //compute the sum of
        sum += rainfall[3*i + j];       //one quarter
    }
    quarterAverage[i] = sum / 3.0;    //Quarter (i+1) average
}
```

Array Initialization

- Like other data types, it is possible to declare and initialize an array at the same time.

```
int[] number = { 2, 4, 6, 8 };

double[] samplingData = { 2.443, 8.99, 12.3, 45.009, 18.2,
                          9.00, 3.123, 22.084, 18.08 };

String[] monthName = { "January", "February", "March",
                       "April", "May", "June", "July",
                       "August", "September", "October",
                       "November", "December"  };
```

```
number.length ⟶ 4
samplingData.length ⟶ 9
monthName.length ⟶ 12
```


Variable-size Declaration

- In Java, we are not limited to fixed-size array declaration.
- The following code prompts the user for the size of an array and declares an array of designated size:

```
int size;

int[] number;

size= Integer.parseInt(JOptionPane.showInputDialog(null,
                                "Size of an array:"));

number = new int[size];
```

Arrays of Objects

- In Java, in addition to arrays of primitive data types, we can declare arrays of objects
- An array of primitive data is a powerful tool, but an array of objects is even more powerful.
- The use of an array of objects allows us to model the application more cleanly and logically.

The Person Class

- We will use Person objects to illustrate the use of an array of objects.

```
Person latte;

latte = new Person( );
latte.setName("Mr. Latte");
latte.setAge(20);
latte.setGender('F');

System.out.println( "Name: " + latte.getName()   );
System.out.println( "Age : " + latte.getAge()    );
System.out.println( "Sex : " + latte.getGender() );
```

The Person class supports the set methods and get methods.

Creating an Object Array - 1

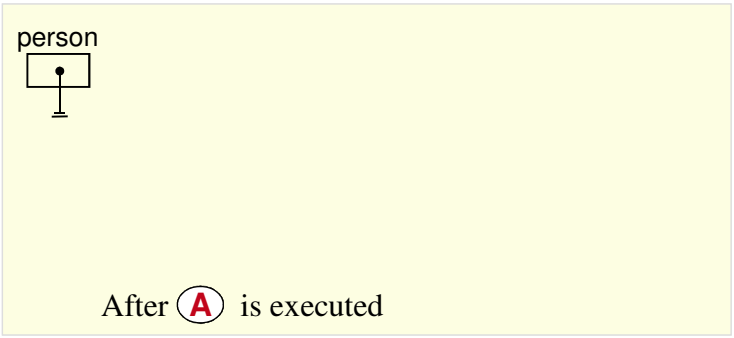
Code

A

```
Person[ ]  person;
person = new Person[20];
person[0] = new Person( );
```

Only the name person is declared, no array is allocated yet.

State of Memory



Creating an Object Array - 2

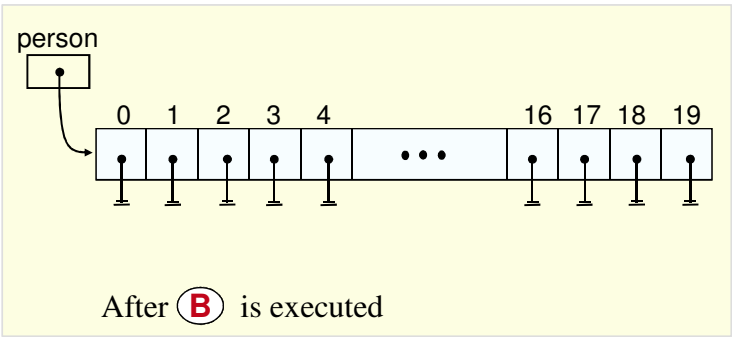
Code

B

```
Person[ ]  person;
person = new Person[20];
person[0] = new Person( );
```

Now the array for storing 20 Person objects is created, but the Person objects themselves are not yet created.

State of Memory



Creating an Object Array - 3

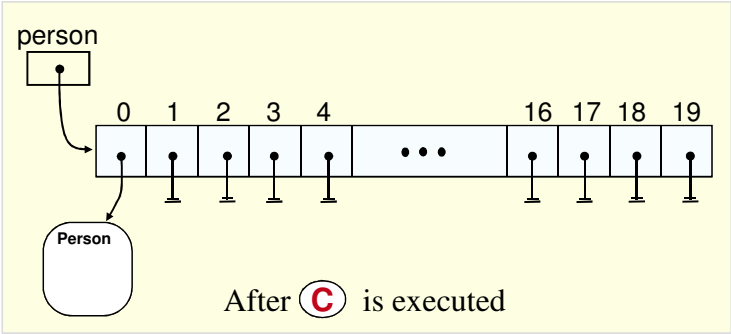
Code

```
Person[ ]  person;
person = new Person[20];
person[0] = new Person( );
```

C

One Person object is created and the reference to this object is placed in position 0.

State of Memory



Person Array Processing – Sample 1

- Create Person objects and set up the person array.

```
String      name, inpStr;
int         age;
char        gender;

for (int i = 0; i < person.length; i++) {
     name    = inputBox.getString("Enter name:");     //read in data values
     age     = inputBox.getInteger("Enter age:");
     inpStr  = inputBox.getString("Enter gender:");
     gender = inpStr.charAt(0);

     person[i] = new Person( );    //create a new Person and assign values

     person[i].setName   ( name   );
     person[i].setAge    (  age   );
     person[i].setGender( gender );
}
```

Person Array Processing – Sample 2

- Find the youngest and oldest persons.

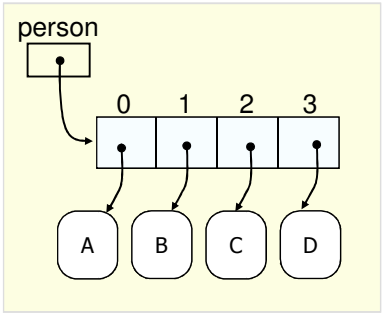
```
int      minIdx = 0;       //index to the youngest person
int      maxIdx = 0;       //index to the oldest person

for (int i = 1; i < person.length; i++) {

    if ( person[i].getAge() < person[minIdx].getAge() ) {
        minIdx       = i;          //found a younger person

    } else if (person[i].getAge() > person[maxIdx].getAge() ) {

        maxIdx = i;  //found an older person
    }
}

//person[minIdx] is the youngest and person[maxIdx] is the oldest
```


Object Deletion – Approach 1

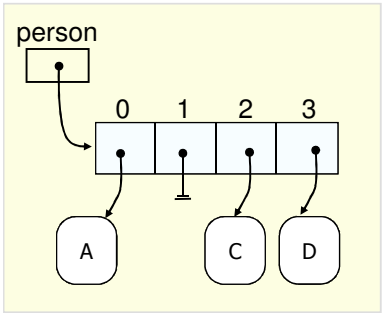
A

```
int delIdx = 1;
person[delIdx] = null;
```

Delete Person B by setting the reference in position 1 to null.



Before A is executed



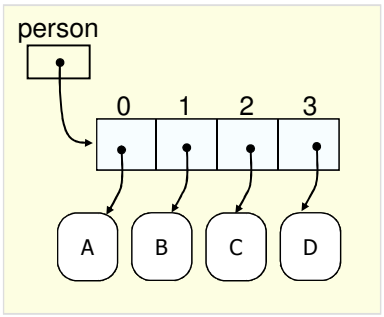
After A is executed

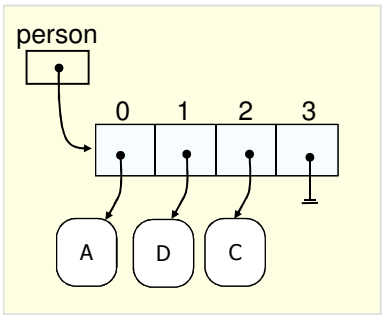
Object Deletion – Approach 2

A

```
int delIndex = 1, last = 3;
person[delIndex] = person[last];
person[last]     = null;
```

Delete Person B by setting the reference in position 1 to the last person.





Before A is executed

After A is executed

Person Array Processing – Sample 3

- Searching for a particular person. Approach 2 Deletion is used.

```
int i = 0;

while ( person[i] != null && !person[i].getName().equals("Latte") ) {
    i++;
}

if ( person[i] == null ) {
    //not found - unsuccessful search
    System.out.println("Ms. Latte was not in the array");

} else {
    //found - successful search
     System.out.println("Found Ms. Latte at position " + i);
}
```

Passing Arrays to Methods - 1

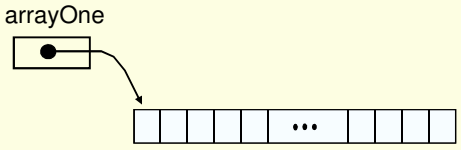
Code

```
minOne
= searchMinimum(arrayOne);
```

A

```
public int searchMinimum(float[]
number))
{
    …

}
```

At A before searchMinimum



State of Memory

A. Local variable number does not exist before the method execution

Passing Arrays to Methods - 2

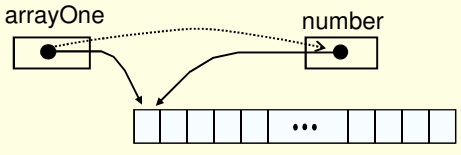
Code

```
minOne
= searchMinimum(arrayOne);
```

```
public int searchMinimum(float[]
number))
{
    …

}
```

B

The address is copied at B



State of Memory

B. The value of the argument, which is an address, is copied to the parameter.

Passing Arrays to Methods - 3

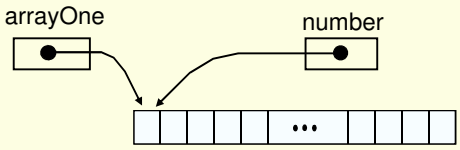
Code

```
minOne
= searchMinimum(arrayOne);
```

```
public int searchMinimum(float[]
number))
{
    …
}
```

C

While at C inside the method



State of Memory

C. The array is accessed via number inside the method.

Passing Arrays to Methods - 4

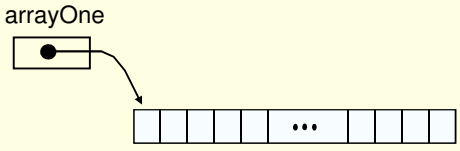
Code

```
minOne
= searchMinimum(arrayOne);
```

D

```
public int searchMinimum(float[]
number))
{
    …
}
```

At D after searchMinimum



State of Memory

D. The parameter is erased. The argument still points to the same object.

Two-Dimensional Arrays

- Two-dimensional arrays are useful in representing tabular information.

Distance Table (in miles)

	Los Angeles	San Francisco	San Jose	San Diego	Monterey
Los Angeles	—	600	500	150	450
San Francisco	600	—	100	750	150
San Jose	500	100	—	650	50
San Diego	150	750	650	—	600
Monterey	450	150	50	600	—

Multiplication Table

	1	2	3	4	5	6	7	8	9
1	1	2	3	4	5	6	7	8	9
2	2	4	6	8	10	12	14	16	18
3	3	6	9	12	15	18	21	24	27
4	4	8	12	16	20	24	28	32	36
5	5	10	15	20	25	30	35	40	45
6	6	12	18	24	30	36	42	48	54
7	7	14	21	28	35	42	49	56	63
8	8	16	24	32	40	48	56	64	72
9	9	18	27	36	45	54	63	72	81

Tuition Table

	Day Students	Boarding Students
Grades 1 – 6	$ 6,000.00	$ 18,000.00
Grades 7 – 8	$ 9,000.00	$ 21,000.00
Grades 9 – 12	$ 12,500.00	$ 24,500.00

Declaring and Creating a 2-D Array

Declaration

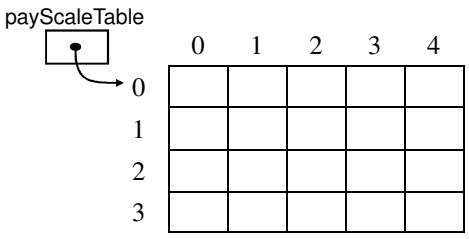
```
<data type> [][] <variable>    //variation 1
<data type>   <variable>[][]   //variation 2
```

Creation

```
<variable>  = new <data type> [ <size1> ][ <size2> ]
```

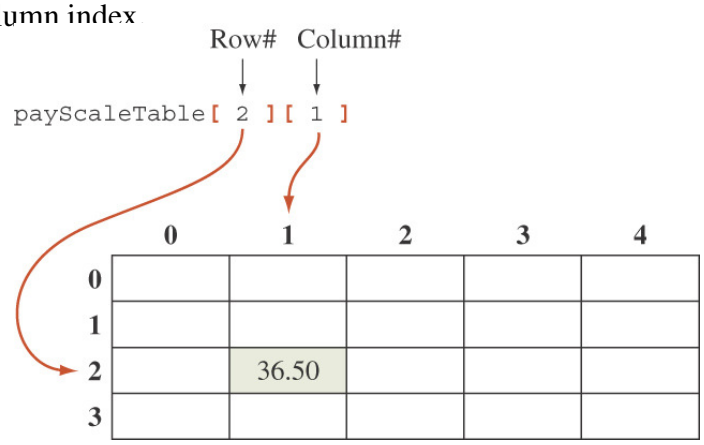
Example

```
double[][] payScaleTable;
payScaleTable
    = new double[4][5];
```



Accessing an Element

- An element in a two-dimensional array is accessed by its row and column index



Sample 2-D Array Processing

- Find the average of each row.

```
double[ ] average = { 0.0, 0.0, 0.0, 0.0 };

for (int i = 0; i < payScaleTable.length; i++) {

    for (int j = 0; j < payScaleTable[i].length; j++) {

        average[i] += payScaleTable[i][j];
    }

    average[i] = average[i] / payScaleTable[i].length;
}
```

Java Implementation of 2-D Arrays

- The sample array creation

```
payScaleTable = new double[4][5];
```

is really a shorthand for

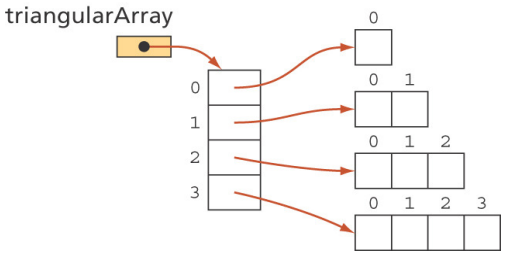
```
payScaleTable = new double [4][ ];

payScaleTable[0] = new double [5];
payScaleTable[1] = new double [5];
payScaleTable[2] = new double [5];
payScaleTable[3] = new double [5];
```


Two-Dimensional Arrays

- Subarrays may have different lengths.
- Executing

```
triangularArray = new double[4][ ];
for (int i = 0; i < 4; i++)
  triangularArray[i] = new double [i + 1];
```

results in an array that looks like:



Copying Arrays

```
int[] a, b; // Declare two arrays
a = new int[] { 10, 20 , 30 };
// Create one
b = a; // Make a copy of array a?
```

- **Will this code create a copy of 'a' in 'b'?**
- **It will only copy the reference of 'a' in 'b'. a and b will be pointing to the same array in the memory.**
- **This is known as shallow copy.**

Copying Arrays (Deep Copy)

```
int[] a, b; // Declare two arrays
a = new int[] { 10, 20 , 30 };
// Create one
// Really make a copy of array 'a'
b = new int[a.length]; // Allocate b
for ( int i = 0; i < a.length; i++ )
{
     b[i] = a[i];
}
```

Common Array Mistakes

- Assigning a scalar to an array

```
int[] myArray = 5;
```

  – 5 is not an array, it is an element

- Assigning an array to a scalar

```
int[] myArray = new int[1000];
int myInt = myArray;
```


Common Array Mistake

- Never try to assign arrays of different dimensions or you will become best friends with something similar to:

```
"Incompatible type for =. Can't
  convert int[] to int[][]"
```

  – similar message for assigning arrays of the wrong type

ArrayList

- ArrayList collections are similar to arrays, except:
  - They grow if you add items to them, automatically
  - They have methods for setting an element, getting an element, and checking the size of the list, rather than using operators or instance variables

Advantages of ArrayList

- When you add an element to the array, it will grow to fit
- When you remove an element from the array, it shrinks to fit
- Adding or removing an element in the middle doesn't leave a "hole" -- elements shift as needed

Array vs. ArrayList

```
String[] cats = {"Fluffy", "Boots", "Puff"};

   vs

ArrayList cats = new ArrayList();
cats.add("Fluffy");
cats.add("Boots");
cats.add("Puff");
```


Array vs. ArrayList

```
cats[0] = "Siamese";
System.out.println(cats[0]+ " cat");

vs

cats.set(0, "Siamese");
System.out.println(cats.get(0)+ " cat");
```

Array vs. ArrayList

```
for(int ind=0;ind<cats.length;ind++){
        System.out.println(cats[ind]);
}

vs.

for(int ind=0;ind<cats.size();ind++){
        System.out.println(cats.get(ind));
}
```


Arrays and ArrayList

- Can one Array instance hold both primitive types and object types?

Wrapper Classes

- A collection can hold only objects (instances of classes), not primitives
- For each primitive, there is a "wrapper class"
  - Integer for int
  - Double for double
  - Character for char

Using Wrapper Classes

- Each wrapper class contains one data member of an appropriate value
- Constructor (one parameter)
  - `Integer myInt = new Integer(5);`
  - `Character myChar = new Character('x');`
- Accessor
  - `int i = myInt.intValue();`
  - `char c = myChar.charValue();`

Autoboxing

- Consider the following code:

```
ArrayList<Integer> numbers;
for(int i=0;i<5;i++){
     numbers.add(i);
}
```

- Before Java 5.0, this code would cause an error (adding a non-Integer to an array of Integers)
- With autoboxing, int is automatically upgraded to Integer, char to Character, etc. when accessing a collection of the same type.

Ensuring Type Safety

- ArrayList could hold many object types which cause type safety issues.
- Starting Java 5.0 it allows generics for type safety.

```
ArrayList<String> s = new
            ArrayList<String>();
ArrayList<String> t = s;  //ok
ArrayList<Point> p = s;  //compiler error
```

Collection Types (Generics)

- You can only add an object of the proper type to a collection

```
ArrayList<String> mylist = new
  ArrayList<String>();
mylist.add("hello");  //adds string to list
mylist.add("12345"); //adds to list
mylist.add(12345); //compiler error
```


Another Example of Generics

```
List<BankAccount> accountList =
                    new ArrayList<BankAccount>();

  accountList.add(new BankAccount("One", 111.11));
  accountList.add(new BankAccount("Two", 222.22));
  accountList.add(new BankAccount("Three", 333.33));
  accountList.add(new BankAccount("Four", 444.44));
  System.out.println(accountList.toString());
```

- ***Output***
  - [One $111.11, Two $222.22, Three $333.33, Four $444.44]

Readings

Book Name: Object Oriented Programming in Java™
Author: Richard L.Halterman
Content: Chapter # 20 & 21

Book Name: Beginning Java Objects
Author: Jacquie Barker
Content: Chapter # 6

Acknowledgements

- While preparing this course I have greatly benefited from the material developed by the following people:
  - Ellen Walker (Hiram College)
  - Richard Halterman (Southern Adventist University)
  - C Thomas Wu (Naval Postgraduate School)