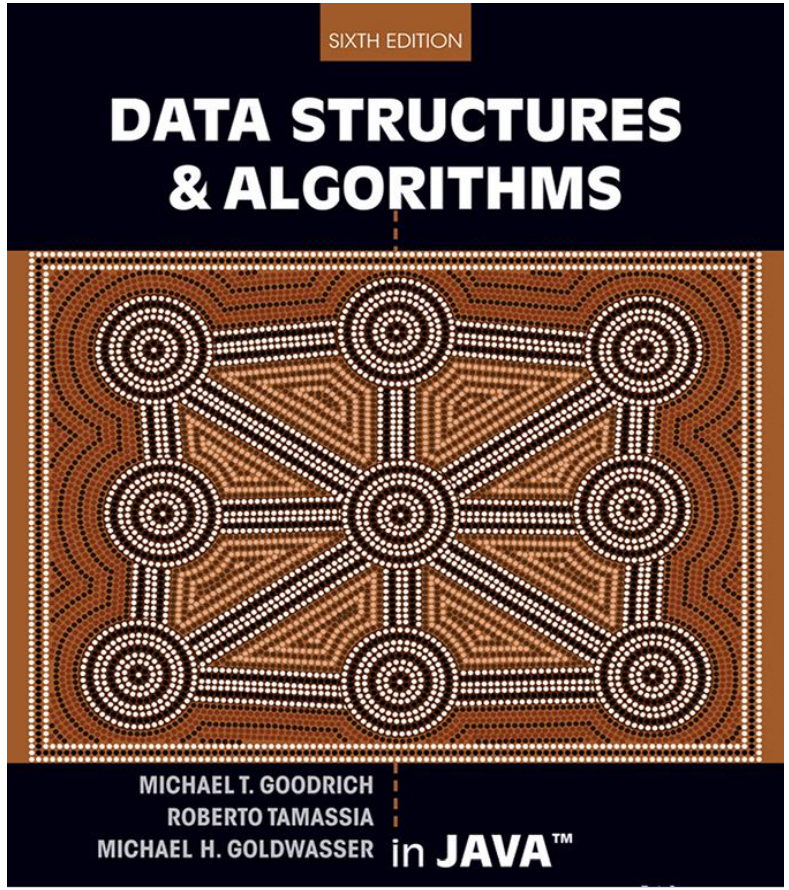
DATA STRUCTURES & ALGORITHMS



Data Structures & Algorithms

Lecture 2

Fundamental Data Structures

BTEC Alliance with FPT Education

Pearson

Topics

- Using Arrays
- Singly Link Lists

Loops for Processing an Array (1/3)

To access the values in an array, 3 kinds of loops are useful:

1. For Loop

```
for(int k = 0; k < array.length; k++) {
        // Accessing array elements using index k
        double x = array[k];
        // ... // do something with x
}
```

Loops for Processing an Array (2/3)

To access the values in an array, 3 kinds of loops are useful:

2. For Each Loop

```
for(double x : array) {
        // Accessing array elements directly using x
        // ... // do something with x
}
```

Loops for Processing an Array (3/3)

To access the values in an array, 3 kinds of loops are useful:

3. While Loop

```
int k = 0;
while(k < array.length) {
        // Accessing array elements using index k
        double x = array[k]; // ... // do something with x
        k++;  // increment index
}
```

"foreach" Version of "for" Loop

To iterate over every element of an array you can use a special "for" syntax

```
int[] a = { 2, 4, 6, 8, 10 };
        // iterate over all elements
        // assign each element to the loop variable x
for (int x : a) {
        System.out.println(x);
}
```

***Note**: In C#, Perl, and VisualBasic this is called a "foreach" loop. Java doesn't use the name "foreach" to avoid creating another reserved word.*

Reading Data into an Array

Suppose we want to read some words from the input into an array. Maybe we know that the input will never contain more than 100 words. We could write...

```
// create Scanner to read input
Scanner input = new Scanner(System.in); // create array of words (Strings)
String[] words = new String[100]; // read the data
int count = 0;
while (input.hasNext() && count < words.length) {
    words[count] = input.next();
    count++;
}
```

Sort Data in an Array

java.util.Arrays - provides utility methods for arrays.

One method is: Arrays.sort(array[ ])

```
/** Sort the words[ ] array from last slide */
/** You must "import java.util.Arrays".     */
Arrays.sort(words);
```

Input data

dog
cat
frog
DOGS
ANT

Arrays.sort();

Result

words[0] = "ANT"
words[1] = "DOGS"
words[2] = "cat"
words[3] = "dog"
words[4] = "frog"

Sort part of an Array

The previous slide is not quite correct. Since we only have data for part of the array, we should only sort that part.

Use: Arrays.sort(*array, start_index, end_index*);

```
// sort elements 0 until count (exclusive)
Arrays.sort(words, 0, count);
```

This sorts only the elements

words[0] words[1] ... words[count - 1]

Output the Elements of an Array

The Now lets print the values of the array.

```
// write a loop to display each array element
for (int k = 0; k < count; k++) {
        System.out.printf("%d: %s\n", k, words[k]);
}
```

Output

```
0: ANT
1: DOGS
2: cat
3: dog
4: frog
```

Example: Compute quiz average

The We have a file containing student quiz scores.

Each line of the file looks like this:

```
6054101234,9,8.5,10,10,7,9,9.5,10
```

student ID quiz scores

How to split the quiz scores?

- String has a method named split() that splits a string into an array, using a delimiter

```
String fruit = "apple, banana, orange";
String[] x = fruit.split(",");
System.out.println(x[0]); // "apple"
System.out.println(x[1]); // "banana"
System.out.println(x[2]); // "orange"
```

- Syntax:

  String[] result = string.split(delimiter);

  delimiter - a *regular expression* used to split the string

Code for Quiz Average

```
public static void main(String[] args) {
String scores = "100,95,85,90,92";
        printQuizAverage(scores);
}
public static void printQuizAverage(String scores) {
        String[] data = scores.split(",");
        String studentId = data[0];
        double sum = 0.0;
for (int k = 1; k < data.length; k++) {
        sum += Double.parseDouble(data[k]);
}
        double average = sum / (data.length - 1);
        System.out.printf("Student %s has quiz average %.2f\n", studentId, average);
}
```

Auto-sizing Arrays at Initialization

```
int[] primes = {2, 3, 5, 7, 11, 13, 17, 19, 23, 29};
```

- This ***creates*** an array object of size equal to the number of data values, and ***assigns values*** to the array elements:

  primes[0] **= 2**;

- What is the value of ***primes.length*** ?
- What is the index of the last element of ***primes***?
- What is the value of ***primes[5]*** ?
- For what index k is ***primes[k] == 11*** ?

An Array Variable is a *Reference*

```
int[] a = new int[4];
int[] b = new int[5];
for (int k = 0; k < 5; k++) b[k] = 2 * k;
// what does this statement do?
a = b;
```

What does this statement do? Choices:

1. It copies each element of b[ ] into a[ ]

   Like for(k=0; k<a.length; k++) a[k] = b[k];

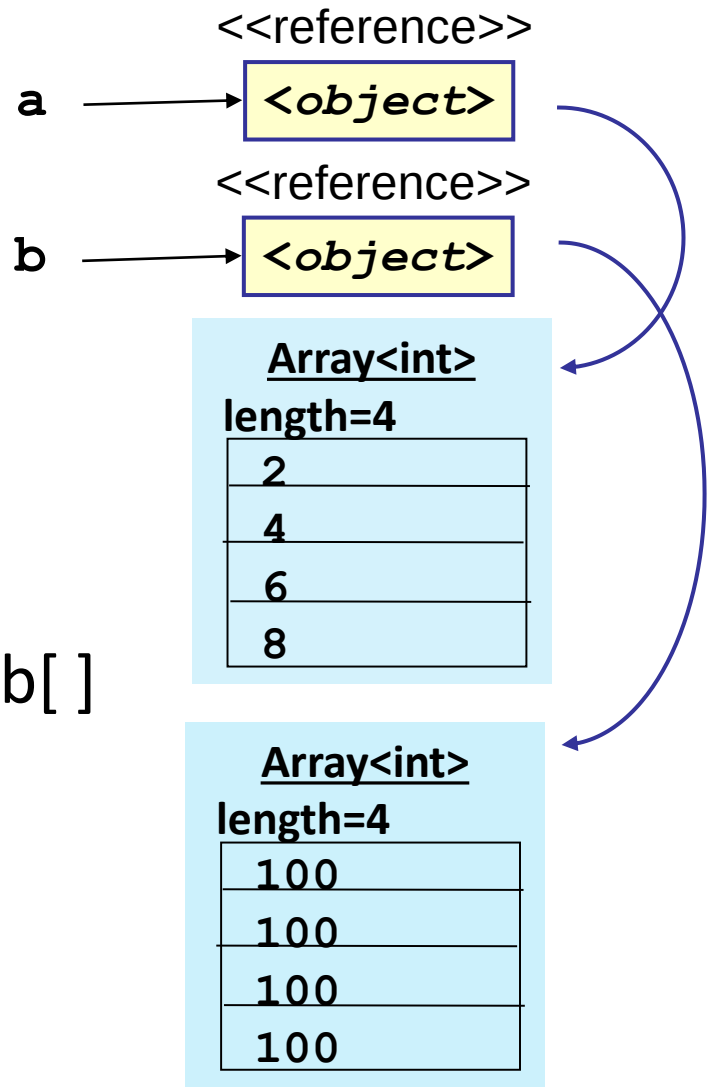
2. Makes a and b refer to the same array

3. Error -- the array sizes aren't the same

Copying An Array (1/6)

Let's create two arrays: a and b

```
import java.util.Arrays;
int[] a = {2, 4, 6, 8};
int[] b = new int[4];
Arrays.fill(b, 100);
```



a and b each refer to an array object of 4 integers

`Arrays.fill`(b) stores 100 in each element of b[ ]

How can we copy the array **a** into the array **b**?

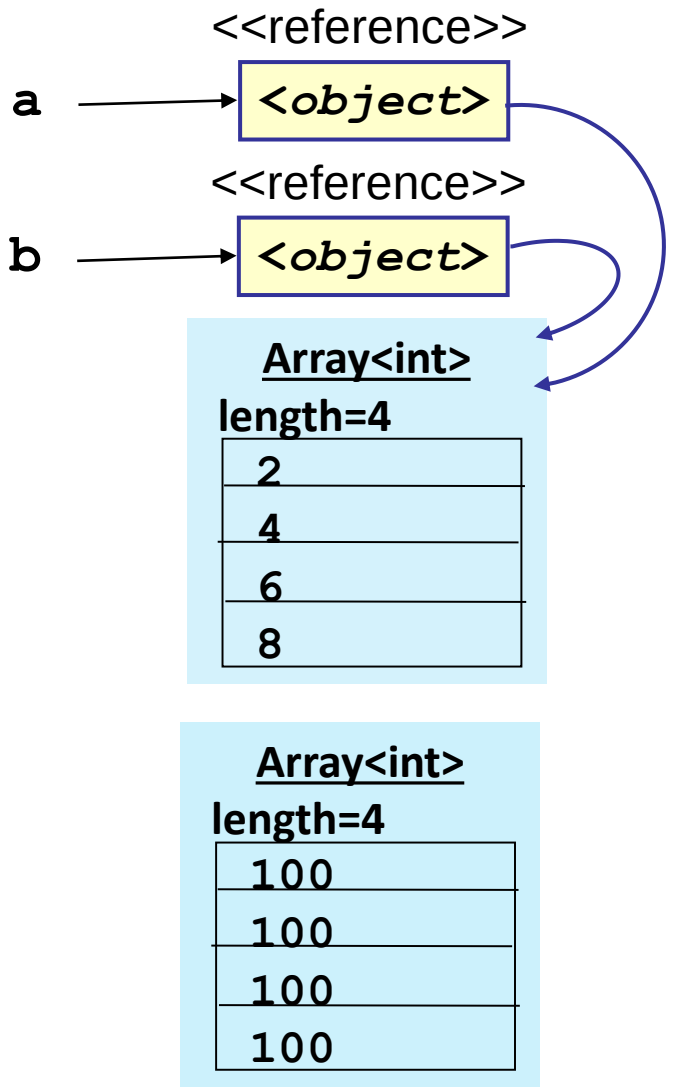
Copying An Array (2/6)

Consider this statement:

```
b = a
```

a and **b** are array references.
This statement copies the *reference*.
It does *not* copy the array elements.

*This statements makes **b** refer to the same array (object) as **a**.*



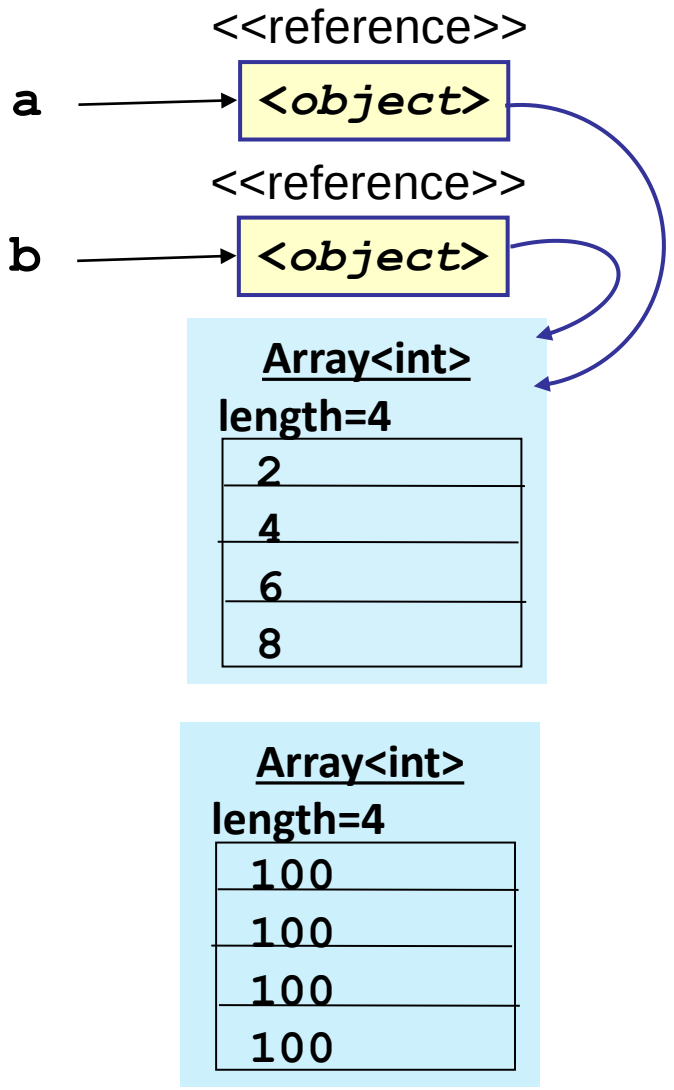
Copying An Array (3/6)

Consider this statement:

```
b = a
```

After this statement, the old "b" array object is lost

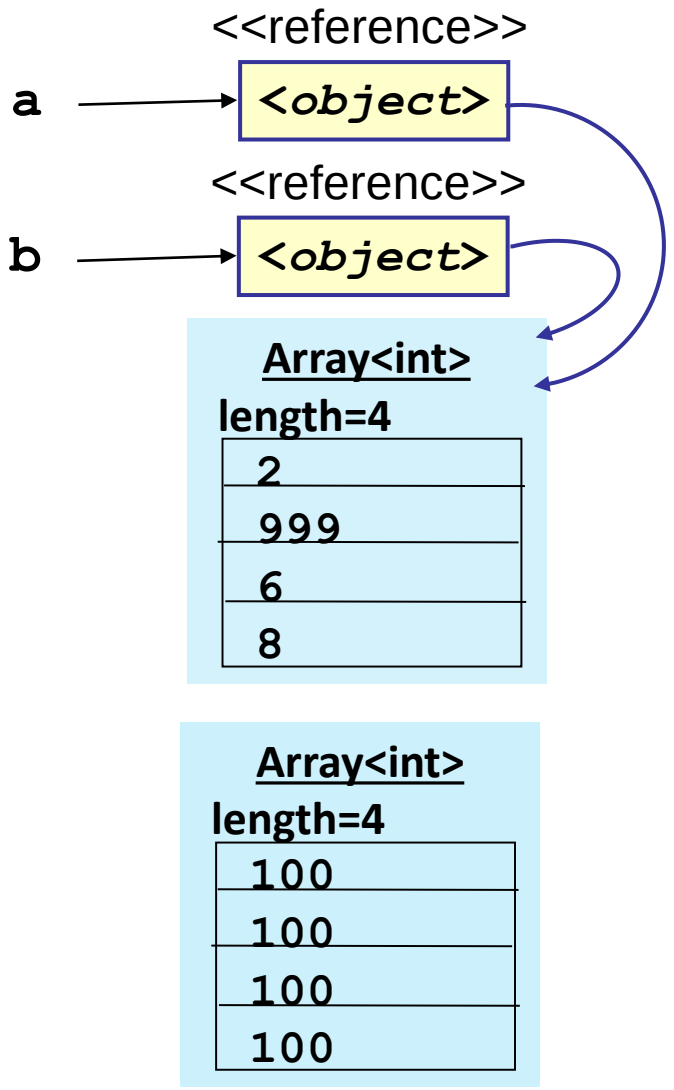
It becomes unreferenced *garbage*



Copying An Array (4/6)

Consider this statement:

```
b = a;
System.out.println( b[1] );
a[1] = 999;
System.out.println( b[1] );
```



To verify this, change an element of a.

The element of b changes, too!

Output:

```
4
99
```

Copying An Array (5/6)

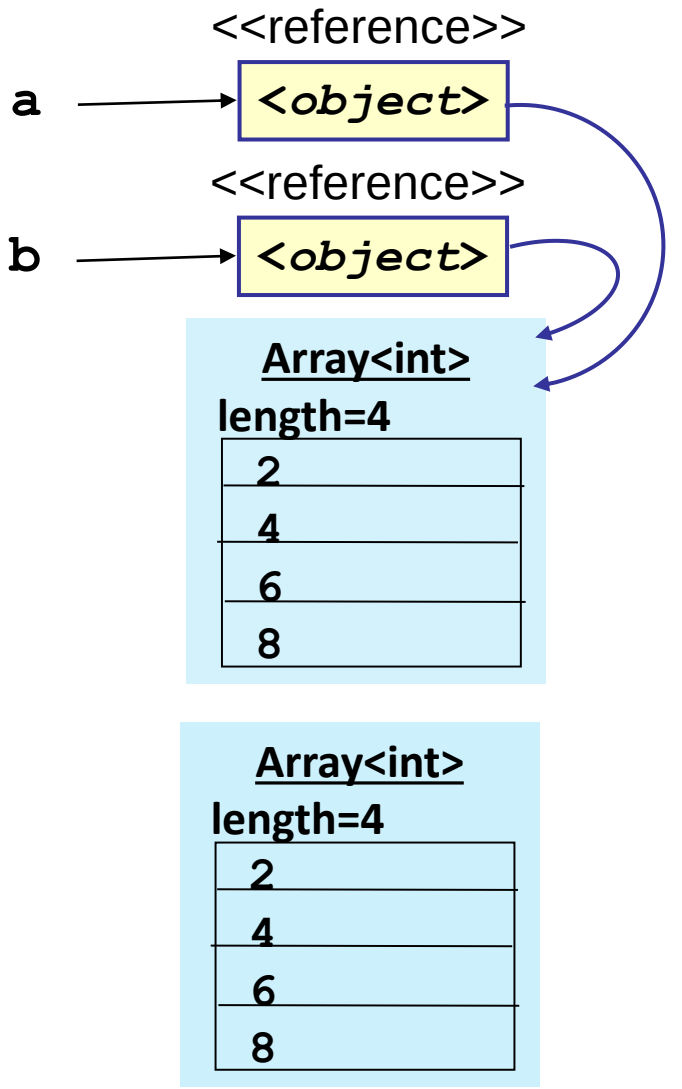
To copy the contents of an array, you must copy each element.

```
// copy each element of array
for (int k = 0; k < a.length; k++) {
        b[k] = a[k]; }
```

This copies all the elements.

But there is an easier way....

System.arraycopy(a, 0, b, 0, 4);



Copying An Array (6/6)

System.arraycopy(src, src_start, dest, dest_start, length)

copies elements from src array to dest array.

It copies length elements only.

```
// copy each element of array
System.arraycopy(a, 0, b, 0, a.length);
```

"foreach" Syntax

```
for (datatype x : a)
        statement;
for (datatype x : a) {
        statement;
        // more statements...
}
```

a is an array object
or any ***Iterable*** collection.

x is a variable or reference of a ***datatype*** that is *assignment compatible* with elements of **a**.

Example: Find the maximum value

Let's write a method to find the maximum value of an array.

Example: double [ ] a = { 0.5, 2.8 -3.7, 18.0, 9.5 };

max(a) is 18.0

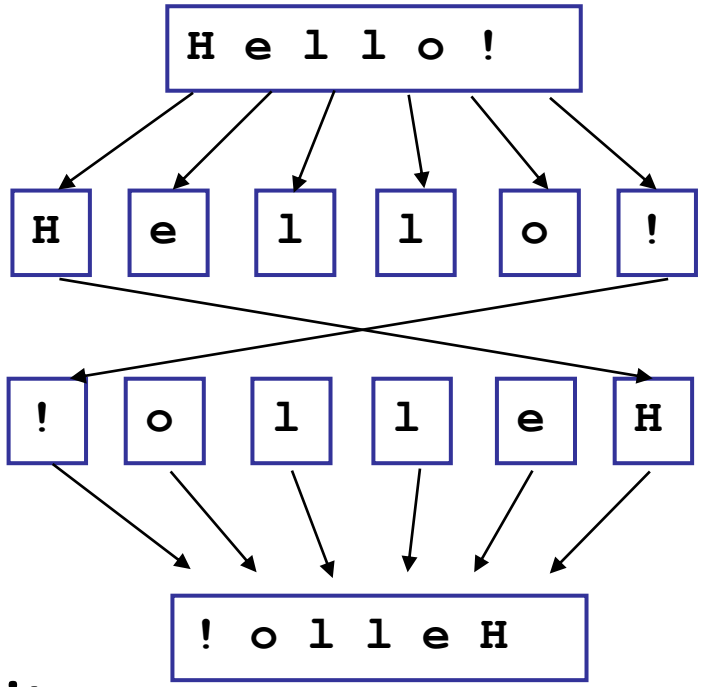
```
/** Find the maximum value of array elements. param a is an array of double
values. Return the maximum value in a. */
   public static double max(double[] a) {
         double maxval = a[0]; // initialize max value to first element
         for (double x : a) // compare all elements of a to maxval
               if (x > maxval) maxval = x;
   return maxval;
   }
```

Example: Reverse a String (1/2)

- Write a method named reverse to reverse order of a String.
- Example: reverse("Hello there") returns "ereht olleH"
- Algorithm:

1. Convert the parameter (String)
   to an array of characters.
   Use: string.toCharArray( )
2. Iterate over the 1st half of the char array.

   Swap characters with the 2nd half.
3. Convert char array into a String and return it.



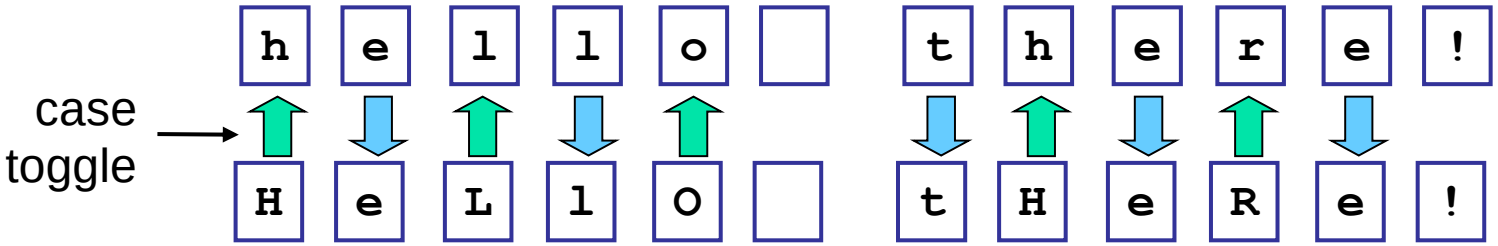
Example: reverse a String (2/2)

The Java API has methods to help you implement this:
"*string".toCharArray( )* - convert string to char array
new String(char [ ] c) - create String from char array

```
/** Reverse the order of a String. Param text is the String to reverse.
Return String with text in reverse order. */
public static String reverse(String text) {
    char[] c = text.toCharArray(); // reverse the chars
    for (int k = 0; k < c.length / 2; k++) {
        int k2 = c.length - k - 1;
        char temp = c[k];
            c[k] = c[k2];
            c[k2] = temp;
    }
return new String(c);
}
```

Example: Case-mangle a String (1/3)

- Write a method to mangle the case of a String, LiKe ThIs
- Example: mangle("hello there") returns "HeLlO tHeRe!"
- Algorithm:
  1. Convert the parameter (String)to an array of characters.
     Use: *string.toCharArray( )*
  2. Iterate over each character.
     2a. if the character is not a letter do nothing.
     2b. if the character is a letter change case and record what was the last change (*to uppercase or to lowercase*).
  3. Convert char array into a String and return it.



Example: Case-mangle a String (2/3)

We can use a boolean variable to keep track of whether the next letter should be uppercase or lowercase.

1. Always flip the case of the first letter in the String.
2. Use a flag to indicate the first letter found.
3. When you see the first letter, decide mangling like this:

uppercase = true if first letter in String is lowercase

uppercase = false if first letter in String is uppercase

```
if ( first ) { // first letter in String
      uppercase = Character.isLowerCase(c[k] );
      first = false;
}
// mangle the letter
if (uppercase)
      c[k] = Character.toUpperCase(c[k] );
      else  c[k] = Character.toLowerCase(c[k] );
```

Example: Case-mangle a String (3/3)

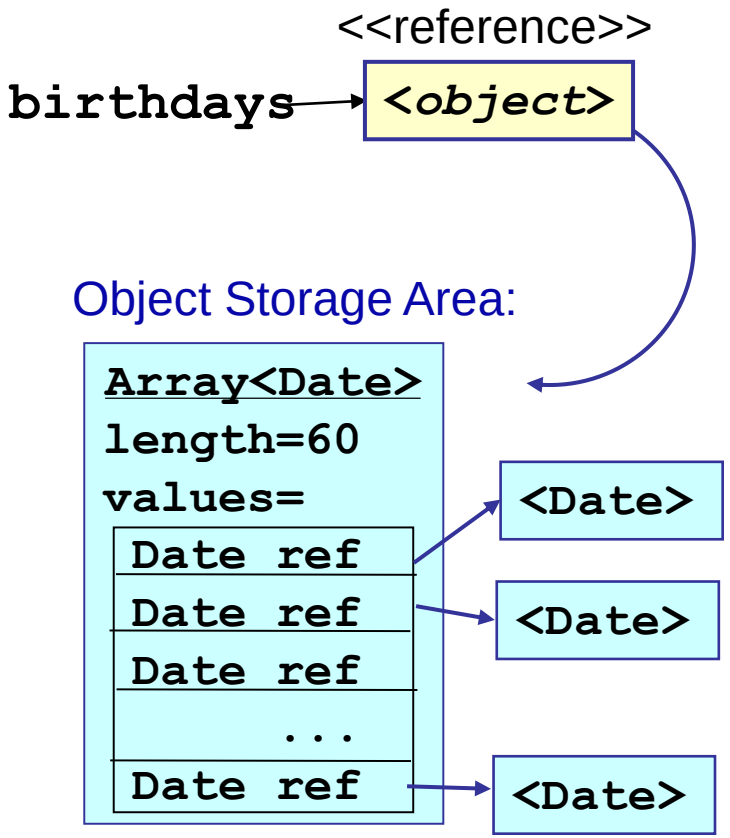
```
/** Mangle case of letters is a String. */
public static String mangle(String text) {
      boolean first = true, uppercase = false;      char[] c = text.toCharArray();
for (int k = 0; k < c.length; k++) {
 // if not a letter then go to next character
      if (!Character.isLetter(c[k])) continue;
      if (first) {
            uppercase = Character.isLowerCase(c[k]);
            first = false;
}
// how to mangle this letter?
if (uppercase)
      c[k] = Character.toUpperCase(c[k]);
else
      c[k] = Character.toLowerCase(c[k]);
uppercase = !uppercase; // for next letter
}
return new String(c);
}
```

Creating an Array of Objects

```
// 1. define array reference
Date[] birthdays;
// 2. allocate storage
birthdays = new Date[60];
// 3. create the objects
// that go in the array!
for (int k = 0; k < birthdays.length; k++) {
    birthdays[k] = new Date();
}
```

birthdays[k] is an *object reference.*

You must create the Date object that it will refer to.



Example: Array of Objects

Suppose we have a file on student names and student ID, like this:

```
48540017    Watchara    Srisawasdi
48540165    Kan         Boonprakub
48540181    Keerati     Tangjitsomkid
48540223    Thunthoch   Laksulapan
48540231    Thanyawan   Tarnpradab
48540249    Palawat     Palawutvichai
48540256    Pitchatarn  Lertudomtana
........    more data
```

We want to store the Student ID and name of each student in an array for further data processing.

Example: Array of Objects

Define a simple Student class with attributes for name and Student ID.

```
public class Student {
    String firstName; // attributes of student
    String lastName;
    String studentID;
    /** constructor for new Student object */
public Student(String fn, String ln, String id) {
    studentID = id; // set the attributes
    firstName = fn; // using parameters of
    lastName = ln; // the constructor
    } // remainder of class definition omitted
}
```

Example: Array of Objects

We can create a new Student object like this:

```
Scanner input = new Scanner(System.in);
/* read data for a student */
String id = input.next();
String first = input.next();
String last = input.next();
/* create a new student object */
Student s = new Student(first, last, id);
```

Example: Array of Objects

To read all the data and save in an array of Student objects, we can do something like this:

```
// Read the data from input
Scanner input = new Scanner(System.in);
// Create an array for Students
Student[] iup = new Student[60];
int count = 0; // How many students?
while (input.hasNext()) {
    String id = input.next();
    String first = input.next();
    String last = input.next();
    iup[count] = new Student(first, last, id);
    count++;
}
```

Create the array object

Create Student object for each element of the array

Useful Array Methods

a.length is an *attribute*, not a method.

Array methods are defined in java.util.Arrays:

Arrays.fill(a, 5 **);**

Set all elements of **a**[ ] to a **value**

Arrays.sort(a);

Sorts elements of **a**[ ]. sort works for primitive data types, Strings, and array of any type where the objects can be compared using their own compareTo() method

Arrays.sort(a, start_index, end_index);

Sorts elements of **a**[ ] beginning at **start_index** (inclusive) and ending at **end_index** (exclusive)

Useful Array Methods

Arrays.binarySearch(a, value)

return index of element in **a**[ ] equal to **value**

array **a**[ ] must already be sorted.

Arrays.equals(a, b)

returns **true** of **a**[] and **b**[] are the same size,

same type, and all elements are equal.

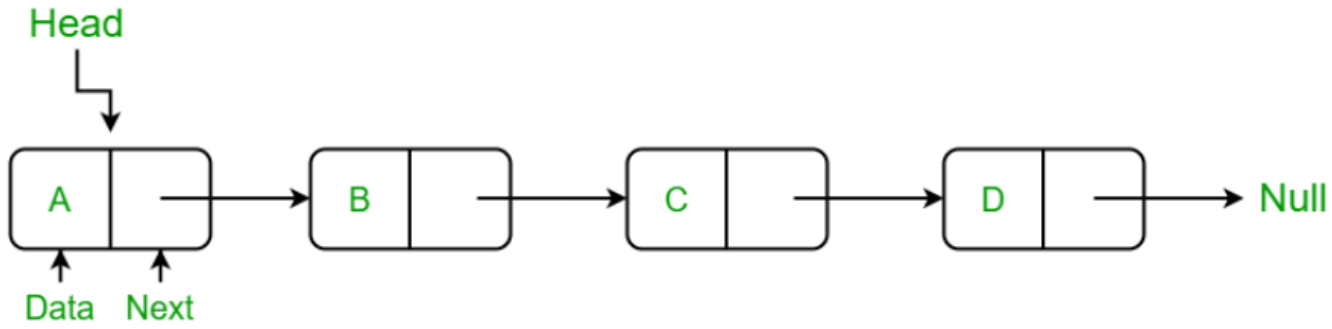
If the elements of **a**[ ] and **b**[ ] are objects (like Date) then the object's equals method is used to compare them.

Arrays.toString(a)

return a string representation of **a**, such as:
["apple", "banana", "carrot", "durian"]

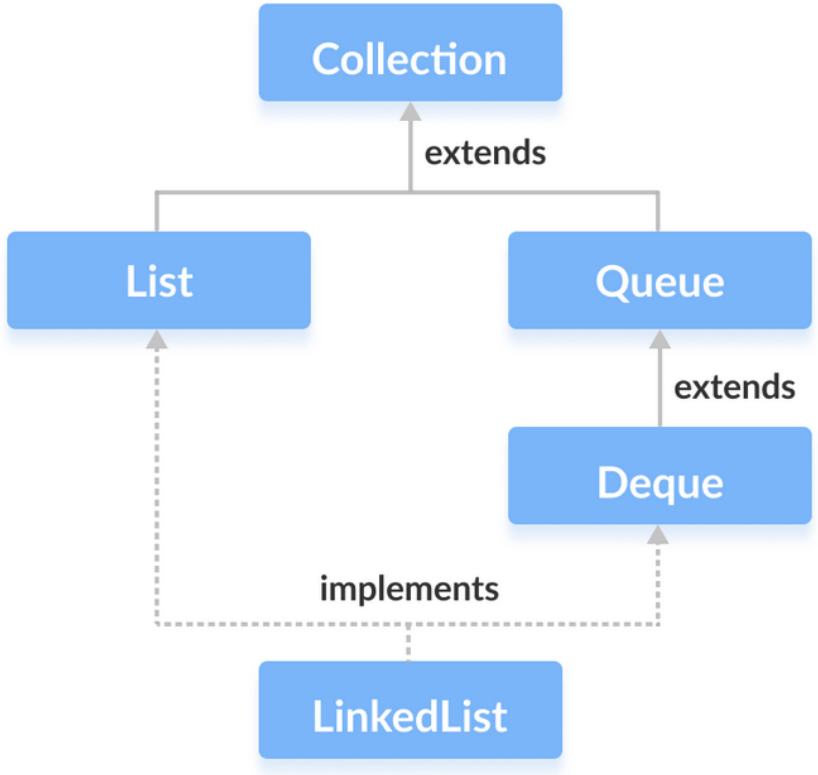
Introduction to LinkedList

- A **LinkedList** is a linear data structure.
- Elements in a LinkedList are not stored adjacent to each other like arrays.
- Each element in a LinkedList is linked to the next element through a pointer, meaning each element will reference the address of the next element.



Using LinkedList in Java

- The LinkedList class in Java implements a doubly linked list. Each element in a LinkedList is referred to as a node.



Using LinkedList in Java

It consists of 3 fields:

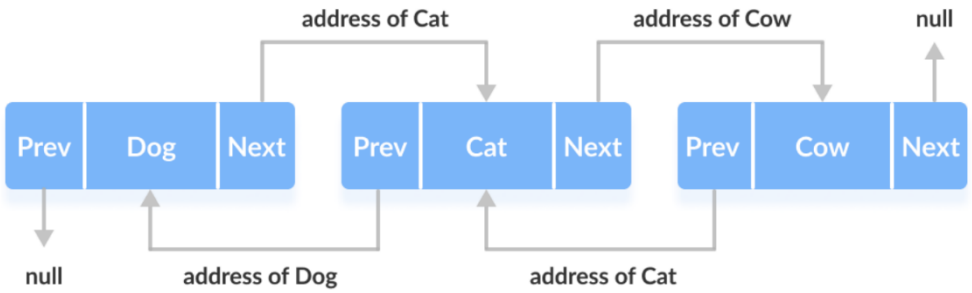
- **Prev** - Stores the address of the previous element in the list. Returns null for the first elements.
- **Next** - Stores the address of the next element in the list. Returns null for the last elements.
- **Data** - Stores the actual data.

Các phần tử trong LinkedList không được lưu trữ theo trình tự. Thay vào đó, chúng nằm rải rác và được kết nối thông qua các liên kết (Prev và Next).

Using LinkedList in Java

There are **3 elements** in a LinkedList:

- Dog - This is the first element, with the previous address being null and the next address pointing to Cat.



LinkedList Implementation in Java

- Cat - This is the second element, with the previous address pointing to Dog and the next address pointing to Cow.
- Cow - This is the last element, with the previous address pointing to Cat and the next address being null.

How to create a LinkedList in Java

Here is how we can create LinkedLists in Java:

```
LinkedList<Type>linkedList = new LinkedList<>();
```

Here, Type is the type of the LinkedList. For example:

```
// create Integer type linked list
LinkedList<Integer> linkedList = new LinkedList<>();

// create String type linked list
LinkedList<String> linkedList = new LinkedList<>();
```

Creating a LinkedList Using an Interface

```
List<String> animals1 = new LinkedList<>();
```

Here, we've declared a LinkedList named 'animals1' using the List interface. LinkedList can only access functions provided by the List interface. Let's see another example

```
Queue<String> animals2 = new LinkedList<>();
Deque<String> animals3 = new LinkedList<>();
```

Here, 'animals2' can access functions from the Queue interface. However, 'animals3' can only access functions from both the Deque and Queue interfaces. This is because Deque is a subinterface of Queue.

Adding Elements to a LinkedList

To add an element (node) to the end of a LinkedList, we use the **add()**

```
import java.util.LinkedList;
class Main {
    public static void main(String[] args){
        LinkedList<String> animals = new LinkedList<>();
        // Add elements to LinkedList
        animals.add("Dog");
        animals.add("Cat");
        animals.add("Horse");
        System.out.println("LinkedList: " + animals);
    }    // LinkedList: [Dog, Cat, Horse]
}
```

Adding Elements from One LinkedList to Another

To add all elements from one LinkedList to another, we use the **addAll()**

```
LinkedList<String> mammals = new LinkedList<>();
    mammals.add("Dog");
    mammals.add("Cat");
    mammals.add("Horse");
    System.out.println("Mammals: " + mammals);
LinkedList<String> animals = new LinkedList<>();
    animals.add("Crocodile");

    // Add all elements of mammals in animals
    animals.addAll(mammals);
    System.out.println("Animals: " + animals);
```

```
run:
Mammals: [Dog, Cat, Horse]
Animals: [Crocodile, Dog, Cat, Horse]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Adding Elements from One LinkedList to Another

Furthermore, we can also use the **listsIterator()** method. To use it, we need to import the **java.util.ListIterator** package:

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

// Creating an object of ListIterator
ListIterator<String> listIterate =animals.listIterator();
  listIterate.add("Dog");
  listIterate.add("Cat");

 System.out.println("LinkedList: " + animals);
// LinkedList: [Dog, Cat]
```

Accessing Elements in a LinkedList

To access an element from a LinkedList, we can use the **get()** method.

```
LinkedList<String> animals= new LinkedList<>();

    // Add elements in the linked list
    animals.add("Dog");
    animals.add("Cat");
    animals.add("Horse");
    System.out.println("LinkedList: " + animals);

    String str = animals.get(1); // Get the element from the linked list
    System.out.println("Element at index 1: " + str);
```

```
run:
LinkedList: [Dog, Horse, Cat]
Element at index 1: Horse
```

Accessing Elements in a LinkedList

Besides this method, to iterate through the elements of a LinkedList, we can use the **iterator()** method. We need to import the **java.util.Iterator**

```
// Creating an object of Iterator
Iterator<String> iterate = animals.iterator();
System.out.print("LinkedList: "); // LinkedList: Dog, Horse, Cat,
while(iterate.hasNext()) {
    System.out.print(iterate.next());
    System.out.print(", ");
}
```

- The **hasNext()** method - returns true if there is a next element.
- The **next()** method - returns the next element.

Accessing Elements in a LinkedList

We can also use the **listIterator()** method to iterate through the elements of a LinkedList. To use this method, we need to import the **java.util.ListIterator** package. The **listIterator()** method is more versatile in LinkedList. This is because **listIterator()** objects can also iterate in reverse direction

Note:

- Hàm **hasNext()** trả về true nếu có phần tử tiếp theo
- Hàm **next()** trả về phần tử tiếp theo
- Hàm **hasPrevious()** trả về true nếu có các phần tử trước
- Hàm **previous()** trả về phần tử trước

Accessing Elements in a LinkedList

```
// Create an object of ListIterator
ListIterator<String> listIterate = animals.listIterator();
System.out.print("LinkedList: ");
while(listIterate.hasNext()) {
    System.out.print(listIterate.next());
    System.out.print(", "); // LinkedList: Dog, Horse, Cat,
}
// Iterate backward
System.out.print("\nReverse LinkedList: ");
while(listIterate.hasPrevious()) {
    System.out.print(listIterate.previous());
    System.out.print(", "); // Reverse LinkedList: Cat, Horse, Dog,
}
```

Searching for Elements in a LinkedList

```
LinkedList<String> animals = new LinkedList<>();

// Add elements in the linked list
    animals.add("Dog");
    animals.add("Cat");
    animals.add("Horse");
System.out.println("LinkedList: " + animals);

// Checks if Dog is in the linked list
if(animals.contains("Dog")) {
   System.out.println("Dog is in LinkedList.");
}
```

```
run:
LinkedList: [Dog, Horse, Cat]
Dog is in LinkedList.
BUILD SUCCESSFUL (total time: 0 seconds)
```

Searching for an Element's Position in a LinkedList

Note:

- The **indexOf()** method - returns the first occurrence index of an element, or -1 if the specified element is not found.
- The **lastIndexOf()** method - returns the index of the last occurrence of an element, or -1 if the specified element is not found.

Searching for an Element's Position in a LinkedList

```
// Add elements in the linked list
    animals.add("Dog");
    animals.add("Cat");
    animals.add("Horse");
System.out.println("LinkedList: " + animals);

// First Occurrence of Dog
System.out.println(animals.indexOf("Dog"));  //0

// Last Occurrence of Dog
System.out.println(animals.lastIndexOf("Dog")); //3
```

Modifying Elements in a LinkedList

To modify elements in a LinkedList, we can use the **set()** method

```
animals.add("Dog");
    animals.add("Cat");
    animals.add("Horse");
    animals.add("Dog");
System.out.println("LinkedList: " + animals);
// Change elements at index 3
animals.set(3, "Zebra");
System.out.println("New LinkedList: " + animals);
```

```
run:
LinkedList: [Dog, Horse, Cat, Dog]
New LinkedList: [Dog, Horse, Cat, Zebra]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Modifying Elements in a LinkedList

We can also modify elements in a LinkedList using the **listIterator()**

```
// Add elements in the linked list
    animals.add("Dog");
    animals.add("Cat");
    animals.add("Horse");
System.out.println("LinkedList: " + animals);
// Creating an object of ListIterator
ListIterator<String> listIterate = animals.listIterator();
listIterate.next();
// Change element returned by next()
listIterate.set("Cow");
System.out.println("New LinkedList: " + animals);
```

```
run:
LinkedList: [Dog, Cat, Horse]
New LinkedList: [Cow, Cat, Horse]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Removing Elements from a LinkedList

To remove an element from a LinkedList, we can use the **remove()**

```
// Add elements in the linked list
    animals.add("Dog");
    animals.add("Cat");
    animals.add("Horse");
    animals.add("Zebra");
System.out.println("LinkedList: " + animals);
// Remove elements from index 1
String str = animals.remove(1);
System.out.println("Removed Element: " + str);

System.out.println("New LinkedList: " + animals);
```

```
run:
LinkedList: [Dog, Horse, Cat, Zebra]
Removed Element: Horse
New LinkedList: [Dog, Cat, Zebra]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Removing Elements from a LinkedList

We can also remove elements from a LinkedList using the **listIterator()**

```
// Add elements in the linked list
    animals.add("Dog");
    animals.add("Cat");
    animals.add("Horse");
System.out.println("LinkedList: " + animals);
// Creating an object of ListIterator
ListIterator<String> listIterate = animals.listIterator();
listIterate.next();
// Remove element returned by next()
listIterate.remove();
System.out.println("New LinkedList: " + animals);
```

```
run:
LinkedList: [Dog, Horse, Cat]
New LinkedList: [Horse, Cat]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Removing Elements from a LinkedList

To remove all elements from a LinkedList, we can use the **clear()**

```
// Add elements in the linked list
    animals.add("Dog");
    animals.add("Cat");
    animals.add("Horse");
System.out.println("LinkedList: " + animals);

// Remove all the elements
animals.clear();
System.out.println("New LinkedList: " + animals);
```

```
run:
LinkedList: [Dog, Horse, Cat]
New LinkedList: []
BUILD SUCCESSFUL (total time: 0 seconds)
```

Removing Elements from a LinkedList

Note: We can also use the **removeAll()** method to remove all elements. However, the **clear()** method is considered more efficient than **removeAll()**.

We can also remove elements from a LinkedList if they satisfy a certain condition. To do this, we use the removeIf() method.

Note: In the example below, **(Integer i)->i<4** is a lambda expression. To learn more about **lambda** expressions, please refer to Java Lambda Expressions

Removing Elements from a LinkedList

```
LinkedList<Integer> animals= new LinkedList<>();

// Add elements in LinkedList
 animals.add(2);
 animals.add(3);
 animals.add(4);
 animals.add(5);
System.out.println("LinkedList: " + animals);

// Remove all elements less than 4
animals.removeIf((Integer i)->i < 4);
System.out.println("New LinkedList: " + animals);
```

```
run:
LinkedList: [2, 3, 4, 5]
New LinkedList: [4, 5]
BUILD SUCCESSFUL (total time: 0 seconds)
```

Summary

- Operations using arrays
- Understand and use singly linked lists

BTEC Alliance with FPT Education

Pearson