ĐẠI HỌC TÔN ĐỨC THẮNG
TON DUC THANG UNIVERSITY

Data Structures and Algorithms I

Collection of Data

Acknowledgement

- The contents of these slides have origin from School of Computing, National University of Singapore.
- We greatly appreciate support from Mr. Aaron Tan Tuck Choy, and Dr. Low Kok Lim for kindly sharing these materials.

Policies for students

- These contents are only used for students PERSONALLY.
- Students are NOT allowed to modify or deliver these contents to anywhere or anyone for any purpose.

Recording of modifications

- Course website address is changed to http://sakai.it.tdt.edu.vn
- Slides “Practice Exercises” are eliminated.
- Course codes cs1010, cs1020, cs2010 are placed by 501042, 501043, 502043 respectively.

Objectives

Using **arrays**

Generics: Allowing operations not tired to a specific data typ

Classes: **Vector** and **ArrayList**

References

Book

- **Array:** Chapter 1, Section 1.1, pages 35 to 38
- **Generics:** Chapter 9, Section 9.4, pages 499 to 507

IT-TDT Sakai → 501043 website → Lessons

- http://sakai.it.tdt.edu.vn

Outline (1/2)

Outline (2/2)

3. Vector
   - 3.1 Motivation
   - 3.2 API Documentation
   - 3.3 Example
4. ArrayList
   - 4.1 Introduction
   - 4.2 API Documentation
   - 4.3 Example

0. Recapitulation

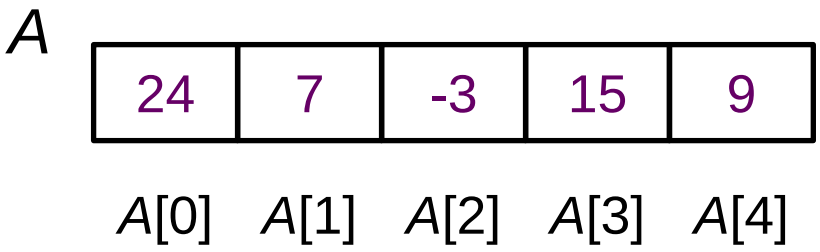
- We explored OOP concepts learned in week 2 in more details (constructors, overloading methods, class and instance methods).
- In week 3, we learned some new OOP concepts (encapsulation, accessors, mutators, “this” reference, overriding methods)
- UML was introduced to represent OO components

1 Array

A collection of homogeneous data

Introduction

- Array is the simplest way to store a collection of data of the same type (homogeneous)
- It stores its elements in contiguous memory
  - Array index begins from zero
  - Example of a 5-element integer array *A* with elements filled



Array in C (1/2)

```c
#include <stdio.h>
#define MAX 6

int scanArray(double [], int);
void printArray(double [], int);
double sumArray(double [], int);

int main(void) {
  double list[MAX];
  int size;

  size = scanArray(list, MAX);
  printArray(list, size);
  printf("Sum = %f\n",
         sumArray(list, size));

  return 0;
}
```

```c
// To read values into arr and return
// the number of elements read.
int scanArray(double arr[], int max_size) {
  int size, i;

  printf("How many elements? ");
  scanf("%d", &size);
  if (size > max_size) {
    printf("Exceeded max; you may only enter");
    printf(" %d values.\n", max_size);
    size = max_size;
  }
  printf("Enter %d values: ", size);
  for (i=0; i<size; i++) {
    scanf("%lf", &arr[i]);
  }
  return size;
}
```

sum_array.c

Array in C (2/2)

sum_array.c

```
// To print values of arr
void printArray(double arr[], int size) {
  int i;

  for (i=0; i<size; i++)
    printf("%f ", arr[i]);
  printf("\n");
}

// To compute sum of all elements in arr
double sumArray(double arr[], int size) {
  int i;
  double sum = 0.0;

  for (i=0; i<size; i++)
    sum += arr[i];
  return sum;
}
```

Array in Java (1/2)

- In Java, **array is an object**.
- Every array has a **public** length attribute (it is not a method!)

TestArray1.java

```
public class TestArray1 {

  public static void main(String[] args) {
    int[] arr; // arr is a reference

    // create a new integer array with 3 elements
    // arr now refers (points) to this new array
    arr = new int[3];

    // using the length attribute
    System.out.println("Length = " + arr.length);

    arr[0] = 100;
    arr[1] = arr[0] - 37;
    arr[2] = arr[1] / 2;
    for (int i=0; i<arr.length; i++)
      System.out.println("arr[" + i + "] = " + arr[i]);
  }
}
```

Declaring an array:
`datatype[] array_name`

Constructing an array:
`array_name = new datatype[size]`

Accessing individual array elements.

```
Length = ?
arr[0] = ?
arr[1] = ?
arr[2] = ?
```

Array in Java (2/2)

- Alternative loop syntax for accessing array elements
- Illustrate toString() method in Arrays class to print an array

TestArray2.java

```
public class TestArray2 {
   public static void main(String[] args) {
      // Construct and initialise array
      double[] arr = { 35.1, 21, 57.7, 18.3 };

      // using the length attribute
      System.out.println("Length = " + arr.length);

      for (int i=0; i<arr.length; i++) {
         System.out.print(arr[i] + " ");
      }
      System.out.println();

      // Alternative way
      for (double element: arr) {
         System.out.print(element + " ");
      }
      System.out.println();
      System.out.println(Arrays.toString(arr));
   }
}
```

```
Length = 4
35.1 21.0 57.7 18.3
35.1 21.0 57.7 18.3
[35.1, 21.0, 57.7, 18.3]
```

Syntax (enhanced for-loop):
`for (datatype e: array_name)`

Go through all elements in the array. "e" automatically refers to the array element sequentially in each iteration

Using toString() method in Arrays class

Array as a Parameter

- The reference to the array is passed into a method
  - Any modification of the elements in the method will affect the actual array

TestArray3.java

```
public class TestArray3 {
  public static void main(String[] args) {
    int[] list = { 22, 55, 33 };

    swap(list, 0, 2);

    for (int element: list)
      System.out.print(element + " ");
    System.out.println();
  }

  // To swap arr[i] with arr[j]
  public static void swap(int[] arr, int i, int j) {
    int temp = arr[i]; arr[i] = arr[j]; arr[j] = temp;
  }
}
```

Detour: String[ ] in main() method

- The main() method contains a parameter which is an array of String objects
- We can use this for command-line arguments

TestCommandLineArgs.java

```
public class TestCommandLineArgs {
  public static void main(String[] args) {

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

```
java TestCommandLineArgs The "Harry Potter" series has 7 books.
args[0] = The
args[1] = Harry Potter
args[2] = series
args[3] = has
args[4] = 7
args[5] = books.
```

Returning an Array

- Array can be returned from a method

TestArray4.java

```
public class TestArray4 {
  public static void main(String[] args) {
    double[] values;

    values = makeArray(5, 999.0);

    for (double value: values) {
      System.out.println(value + " ");
    }
  }

  // To create an array and return it to caller
  public static double[] makeArray(int size, double limit) {
    double[] arr = new double[size];

    for (int i=0; i < arr.length; i++)
      arr[i] = limit/(i+1);

    return arr;
  }
}
```

Return type: `datatype[]`

```
999.0
499.5
333.0
249.75
199.8
```

Common Mistakes (1/3)

- length versus length()
  - To obtain length of a **`String`** object str, we use the **length()** method
    - Example: **`str.length()`**
  - To obtain length (size) of an array arr, we use the **length** attribute
    - Example: **`arr.length`**
- Array index out of range
  - Beware of ArrayIndexOutOfBoundsException

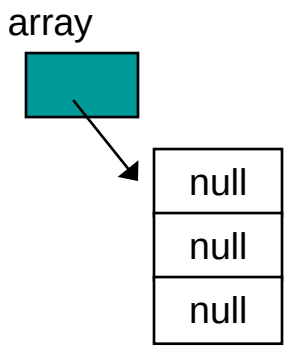
```
public static void main(String[] args) {
  int[] numbers = new int[10];
  . . .
  for (int i = 1; i <= numbers.length; i++)
    System.out.println(numbers[i]);
}
```

Common Mistakes (2/3)

- When you have an array of objects, it's very common to forget to instantiate the array's objects.
- Programmers often instantiate the array itself and then think they're done – that leads to java.lang.NullPointerException
- Example on next slide
  - It uses the `Point` class in the API
  - Refer to the API documentation for details

Common Mistakes (3/3)

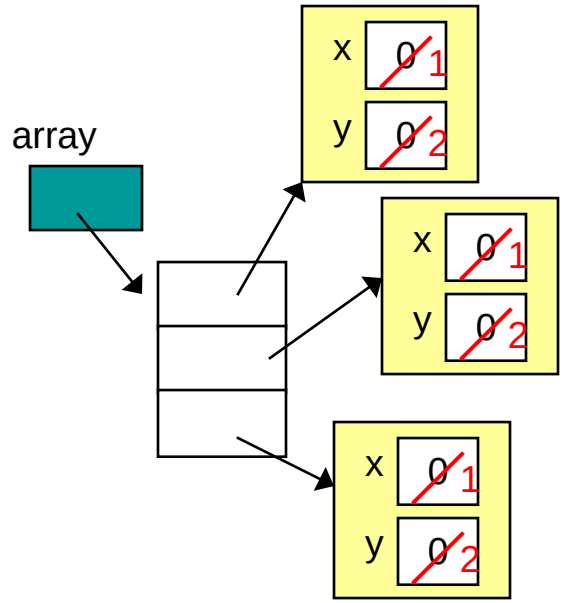
```
Point[] array = new Point[3];
for (int i=0; i<array.length; i++) {
   array[i].setLocation(1,2);
}
```



There are no objects referred to by array[0], array[1], and array[2], so how to call setLocation() on them?!

Corrected code:

```
Point[] array = new Point[3];
for (int i=0; i<array.length; i++) {
   array[i] = new Point();
   array[i].setLocation(1,2);
}
```

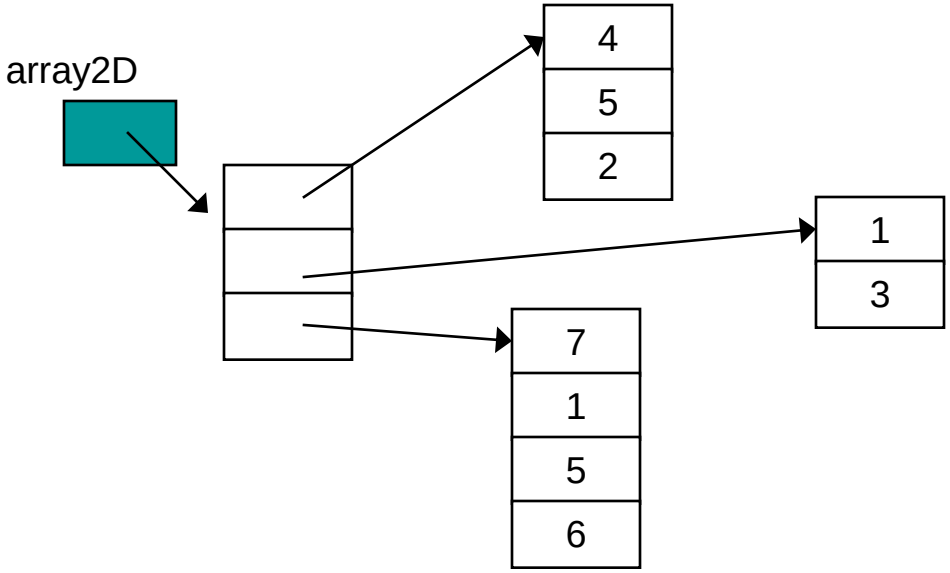


2D Array (1/2)

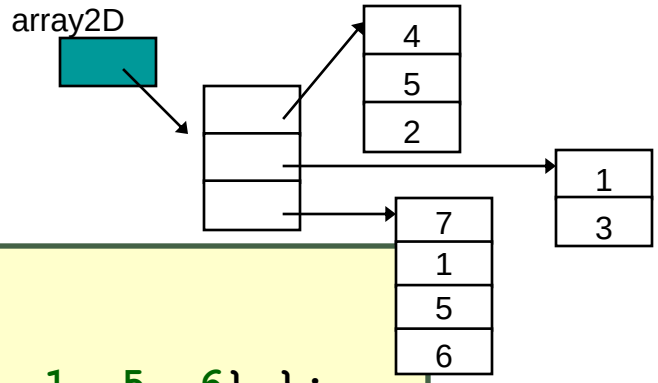
- A two-dimensional (2D) array is an array of array.
- This allows for rows of different lengths.

```
// an array of 12 arrays of int
int[][] products = new int[12][];
```

```
int[][] array2D = { {4,5,2}, {1,3},
                    {7,1,5,6} };
```



2D Array (2/2)




```
public class Test2DArray {
  public static void main(String[] args) {
    int[][] array2D = { {4, 5, 2}, {1, 3}, {7, 1, 5, 6} };

    System.out.println("array2D.length = " + array2D.length);
    for (int i = 0; i < array2D.length; i++)
      System.out.println("array2D[" + i + "].length = "
                         + array2D[i].length);

    for (int row = 0; row < array2D.length; row++) {
      for (int col = 0; col < array2D[row].length; col++)
        System.out.print(array2D[row][col] + " ");
      System.out.println();
    }
  }
}
```

Test2DArray.java

```
array2D.length = 3
array2D[0].length = ?
array2D[1].length = ?
array2D[2].length = ?
?
?
?
```

Drawback

- Array has one major drawback:
  - Once initialized, the array size is fixed
  - Reconstruction is required if the array size changes
  - To overcome such limitation, we can use some classes related to array
- Java has an **Array** class
  - Check API documentation and explore it yourself
- However, we will not be using this Array class much; we will be using some other classes such as **Vector** or **ArrayList**
  - Differences between Vector and ArrayList are in slide 41
- Before doing Vector/ArrayList, we will introduce another concept called Generics

2 Generics

Allowing operation on objects of various types

Motivation

- There are programming solutions that are applicable to a wide range of different data types
  - The code is exactly the same other than the data type declarations
- In C, there is no easy way to exploit the similarity:
  - You need a separate implementation for each data type
- In Java, you can make use of **generic programming**:
  - A mechanism to specify solution without tying it down to a specific data type

Eg: The IntPair Class (non-generic)

- Let's define a class to:
  - Store a pair of integers, e.g. (**74, -123**)
  - Many usages, can represent 2D coordinates, range (min to max), height and weight, etc.

IntPair.java

```
class IntPair {

  private int first, second;

  public IntPair(int a, int b) {
    first = a;
    second = b;
  }

  public int getFirst() { return first; }
  public int getSecond() { return second; }
}
```

Using the IntPair Class (non-generic)

```
// This program uses the IntPair class to create an object
// containing the lower and upper limits of a range.
// We then use it to check that the input data fall within
// that range.
import java.util.Scanner;
public class TestIntPair {

  public static void main(String[] args) {

    IntPair range = new IntPair(-5, 20);
    Scanner sc = new Scanner(System.in);
    int input;

    do {
      System.out.printf("Enter a number in (%d to %d): ",
                        range.getFirst(), range.getSecond());

      input = sc.nextInt();

    } while( input < range.getFirst() ||
             input > range.getSecond() );
  }
}
```

```
Enter a number in (-5 to 20): -10
Enter a number in (-5 to 20): 21
Enter a number in (-5 to 20): 12
```

TestIntPair.java

Observation

- The **`IntPair`** class idea can be easily extended to other data types:
  - **`double`**, **`String`**, etc.
- The resultant code would be almost the same!

```
class StringPair {
   private String first, second;

   public StringPair( String a, String b ) {
      first = a;
      second = b;
   }

   public String getFirst() { return first; }
   public String getSecond() { return second; }

}
```

Only differences are the data type declarations

The Generic Pair Class

```
class Pair <T> {

  private T first, second;

  public Pair(T a, T b) {
    first = a;
    second = b;
  }

  public T getFirst() { return first; }
  public T getSecond() { return second; }

}
```

Pair.java

- Important restriction:
  - The generic type can be substituted by reference data type only
  - Hence, primitive data types are NOT allowed
  - Need to use wrapper class for primitive data type

Using the Generic Pair Class

TestGenericPair.java

```
public class TestGenericPair {

  public static void main(String[] args) {

    Pair<Integer> twoInt = new Pair<Integer>(-5, 20);
    Pair<String> twoStr = new Pair<String>("Turing", "Alan");

    // You can have pair of any reference data types!
    // Print out the integer pair
    System.out.println("Integer pair: (" + twoInt.getFirst()
                       + ", " + twoInt.getSecond() + ")";
    // Print out the String pair
    System.out.println("String pair: (" + twoStr.getFirst()
                       + ", " + twoStr.getSecond() + ")";
  }
}
```

- The formal generic type **<T>** is substituted with the actual data type supplied by the user:
  - The effect is similar to generating a new version of the **Pair** class, where **T** is substituted

Autoboxing/unboxing (1/2)

- The following statement invokes autoboxing

```
Pair<Integer> twoInt = new Pair<Integer>(-5, 20);
```

- `Integer` objects are expected for the constructor, but -5 and 20, of primitive type `int`, are accepted.
- ***Autoboxing*** is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes
  - The primitive values -5 and 20 are converted to objects of `Integer`
- The Java compiler applies autoboxing when a primitive value is:
  - Passed as a parameter to a method that expects an object of the corresponding wrapper class
  - Assigned to a variable of the correspond wrapper class

Autoboxing/unboxing (2/2)

- Converting an object of a wrapper type (e.g.: `Integer`) to its corresponding primitive (e.g: `int`) value is called unboxing.
- The Java compiler applies unboxing when an object of a wrapper class is:
  - Passed as a parameter to a method that expects a value of the corresponding primitive type
  - Assigned to a variable of the corresponding primitive type

```
int i = new Integer(5); // unboxing
Integer intObj = 7;     // autoboxing
System.out.println("i = " + i);
System.out.println("intObj = " + intObj);
```

```
i = 5
intObj = 7
```

```
int a = 10;
Integer b = 10;     // autoboxing
System.out.println(a == b);
```

```
true
```

The Generic NewPair Class

- We can have more than one generic type in a generic class
- Let's modify the generic pair class such that:
  - Each pair can have two values of **different data types**

```
class NewPair <S,T> {

  private S first;
  private T second;

  public NewPair(S a, T b) {
    first = a;
    second = b;
  }

  public S getFirst() { return first; }
  public T getSecond() { return second; }

}
```

You can have multiple generic data types.

Convention: Use single uppercase letters for generic data types.

NewPair.java

Using the Generic NewPair Class

TestNewGenericPair.java

```
public class TestNewGenericPair {

  public static void main(String[] args) {

    NewPair<String, Integer> someone =
      new NewPair<String, Integer>("James Gosling", 55);

    System.out.println("Name: " + someone.getFirst());
    System.out.println("Age: " + someone.getSecond());
  }
}
```

```
Name: James Gosling
Age: 55
```

- This NewPair class is now very flexible!
  - Can be used in many ways

Summary

- Caution:
  - Generics are useful when the code remains unchanged other than differences in data types
  - When you declare a generic class/method, make sure that the code is valid for all possible data types
- Additional Java Generics topics (not covered):
  - Generic methods
  - Bounded generic data types
  - Wildcard generic data types

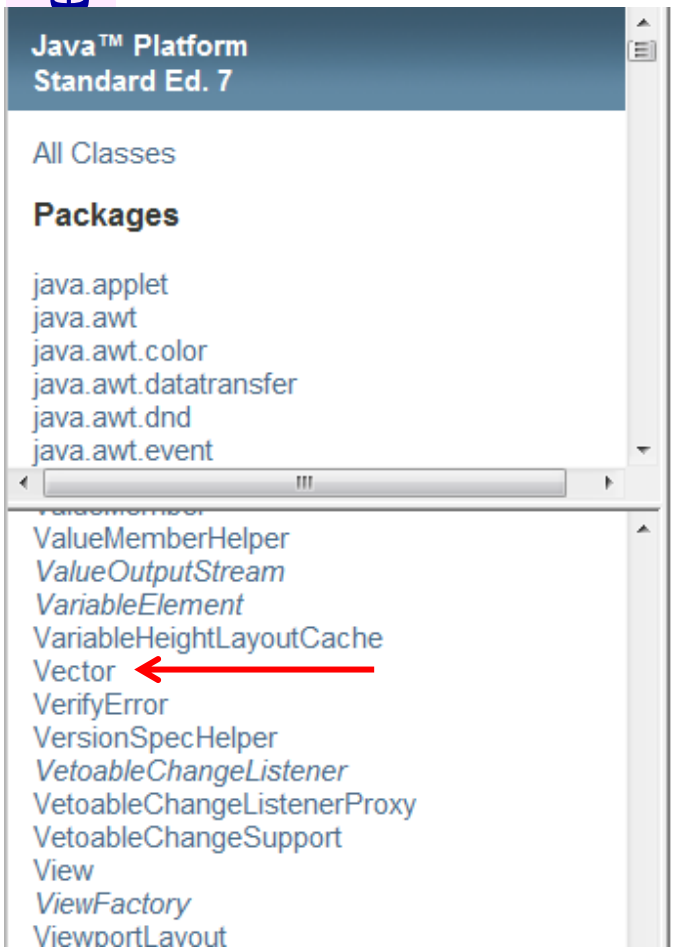
3 Vector class

Class for dynamic-size arrays

Motivation

- Java offers a Vector class to provide:
  - Dynamic size
    - expands or shrinks automatically
  - Generic
    - allows any reference data types
  - Useful predefined methods
- Use array if the size is fixed; use Vector if the size may change.

API documentation (1/3)



Method Summary

Methods

Modifier and Type	Method and Description
boolean	**add**(**E** e) Appends the specified element to the en
void	**add**(int index, **E** elemen Inserts the specified element at the spe
boolean	**addAll**(**Collection**<? ext Appends all of the elements in the spec Collection's Iterator.
boolean	**addAll**(int index, **Colle** Inserts all of the elements in the specifie
void	**addElement**(**E** obj) Adds the specified component to the en
int	**capacity**() Returns the current capacity of this vec
void	**clear**() Removes all of the elements from this V

API documentation (2/3)

PACKAGE

```
import java.util.Vector;
```

SYNTAX

```
//Declaration of a Vector reference
Vector<E> myVector;

//Initialize a empty Vector object
myVector = new Vector<E>;
```

Commonly Used Method Summary	
boolean	***isEmpty*()** Tests if this vector has no components.
int	***size*()** Returns the number of components in this vector .

API documentation (3/3)

Commonly Used Method Summary (continued)	
boolean	***add*(E o)** Appends the specified element to the end of this Vector.
void	***add*(int index, E element)** Inserts the specified element at the specified position in this Vector.
E	***remove*(int index)** Removes the element at the specified position in this Vector.
boolean	***remove*(Object o)** Removes the first occurrence of the specified element in this Vector If the Vector does not contain the element, it is unchanged.
E	***get*(int index)** Returns the element at the specified position in this Vector.
int	***indexOf*(Object elem)** Searches for the first occurrence of the given argument, testing for equality using the equals method.
boolean	***contains*(Object elem)** Tests if the specified object is a component in this vector.

Example

TestVector.java

```
import java.util.Vector;
public class TestVector {

  public static void main(String[] args)

    Vector<String> courses;

    courses = new Vector<String>();

    courses.add("501043");
    courses.add(0, "501042");
    courses.add("502043");

    System.out.println(courses);
    System.out.println("At index 0: " + courses.get(0));

    if (courses.contains("501043"))
      System.out.println("501043 is in courses");

    courses.remove("501043");
    for (String c: courses)
      System.out.println(c);
  }
}
```

Output:

```
[501042, 501043, 502043]
At index 0: 501042
501043 is in courses
501042
502043
```

Vector class has a nice `toString()` method that prints all elements

The enhanced for-loop is applicable to Vector objects too!

4 ArrayList class

Another class for dynamic-size arrays

Introduction (1/2)

- Java offers an ArrayList class to provide similar features as Vector:
  - Dynamic size
    - expands or shrinks automatically
  - Generic
    - allows any reference data types
  - Useful predefined methods
- Similarities:
  - Both are index-based and use an array internally
  - Both maintain insertion order of element
- So, what are the differences between Vector and ArrayList?
  - This is one of the most frequently asked questions, and at interviews!

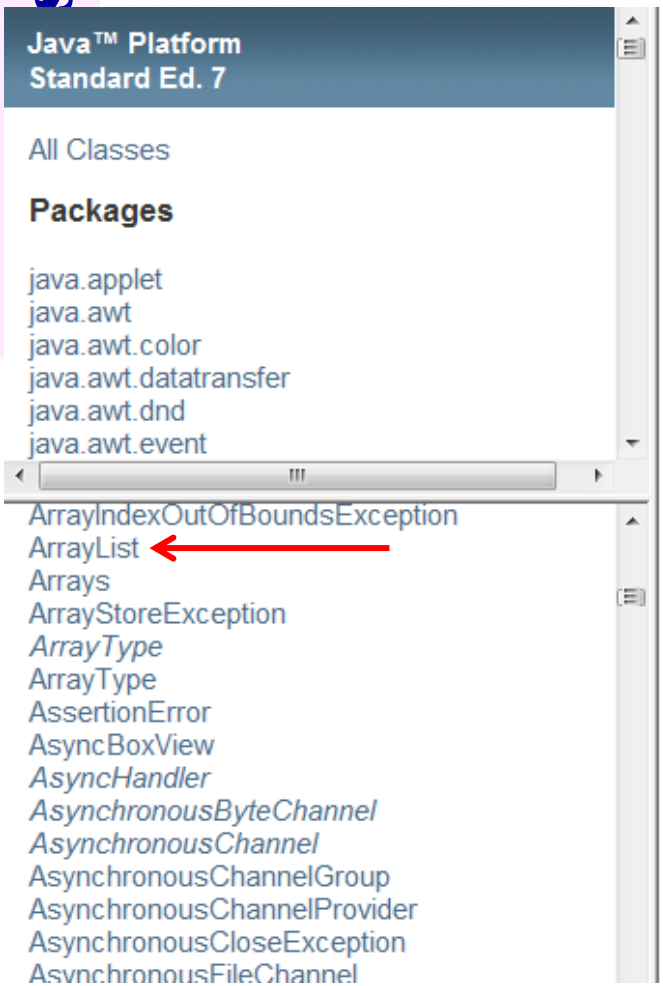
Introduction (2/2)

- Differences between Vector and ArrayList

Vector	ArrayList
Since JDK 1.0	Since JDK 1.2
Synchronised * (thread-safe)	Not synchronised
Slower (price of synchronisation)	Faster (≈20 – 30%)
Expansion: default to double the size of its array (can be set)	Expansion: increases its size by ≈50%

- ArrayList is preferred if you do not need synchronisation
  - Java supports multiple threads, and these threads may read from/write to the same variables, objects and resources. Synchronisation is a mechanism to ensure that Java thread can execute an object's synchronised methods one at a time.
- When using Vector /ArrayList, always try to initialise to the largest capacity that your program will need, since expanding the array is costly.
  - Array expansion: allocate a larger array and copy contents of old array to the new one

API documentation (1/3)



Method Summary

Methods

Modifier and Type	Method and Description
boolean	**add**(**E** e) Appends the specified element to the e
void	**add**(int index, **E** elemen Inserts the specified element at the spe
boolean	**addAll**(**Collection**<? ext Appends all of the elements in the spec Iterator.
boolean	**addAll**(int index, **Colle** Inserts all of the elements in the specifi
void	**clear**() Removes all of the elements from this
Object	**clone**() Returns a shallow copy of this Array
boolean	**contains**(**Object** o) Returns true if this list contains the s
void	**ensureCapacity**(int minC

API documentation (2/3)

PACKAGE

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

SYNTAX

```
//Declaration of a ArrayList reference
ArrayList<E> myArrayList;

//Initialize a empty ArrayList object
myArrayList = new ArrayList<E>;
```

Commonly Used Method Summary	
boolean	***isEmpty*()** Returns **true** if this list contains no element.
int	***size*()** Returns the number of elements in this list.

API documentation (3/3)

Commonly Used Method Summary (continued)	
`boolean`	***add*(E e)** Appends the specified element to the end of this list.
`void`	***add*(int index, E element)** Inserts the specified element at the specified position in this list.
`E`	***remove*(int index)** Removes the element at the specified position in this list.
`boolean`	***remove*(Object o)** Removes the first occurrence of the specified element from this list, if it is present.
`E`	***get*(int index)** Returns the element at the specified position in this list.
`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.
`boolean`	***contains*(Object elem)** Returns **true** if this list contains the specified element.

Example

TestArrayList.java

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

public class TestArrayList {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    ArrayList<Integer> list = new ArrayList<Integer>();

    System.out.println("Enter a list of integers, press ctrl-d to end.");
    while (sc.hasNext()) {
      list.add(sc.nextInt());
    }

    System.out.println(list); // us

    // Move first value to last
    list.add(list.remove(0));

    System.out.println(list);
  }
}
```

Output:

```
Enter a list ... to end.
31
17
-5
26
50
(user pressed ctrl-d here)
[31, 17, -5, 26, 50]
[17, -5, 26, 50, 31]
```

Summary

Java Elements	
	Array: - Declaration and common usage **Generics:** - Allowing operation on objects of various types **Vector** and **ArrayList:** - Dynamic-size arrays - Declaration and useful methods

Pract. Ex. #13

Missing Digits (1/2)

- *[This is adapted from a 501042 exercise in C]*
  Write a program MissingDigits.java to read in a positive integer and list out all the digits that do not appear in the input number. (Assume input value has no leading zeroes.)
- You are to use primitive array, not Vector, ArrayList or any other related classes.
- You should use a boolean array.
- Sample run:

```
Enter a number: 73015
Missing digits in 73015: 2 4 6 8 9
```

Missing Digits (2/2)

- What is the boolean array for? Idea?

Additional Exercise

Detecting Duplicates (1/4)

- Using ArrayList class and random number generation.
  - You may use the Math random() method or the Random class
- Write a program DetectDuplicates.java to read the following values:
  - The number of unique random integers to generate; and
  - Limit of the values: each random number generated should be in the range from 0 (inclusive) to limit (exclusive), or [0, limit – 1].
  - (Certainly, the second input value must not be smaller than the first)
- Each time a random integer is generated, you must check if it is a duplicate of an earlier generated value. If it is, it must be discarded. The program goes on to generate the required number of unique random integers.
- You are to count how many duplicates were detected.

Additional Exercise

Detecting Duplicates (2/4)

- Sample run
  - (In testing your code, each time a random number is generated, you may want to print it to check that the computation is correct)

```
Enter number of unique integers to generate: 10
Enter limit: 20
List: [16, 3, 15, 17, 2, 10, 18, 5, 12, 14]
Duplicates detected: 8
```

Additional Exercise

Detecting Duplicates (3/4)

DetectDuplicates.java

```
import java.util.*;

public class DetectDuplicates {

   public static void main(String[] args) {

      Scanner sc = new Scanner(System.in);
      ArrayList<Integer> list = new ArrayList<Integer>();

      System.out.print("Enter number of unique ...: ");
      int numUnique = sc.nextInt();

      System.out.print("Enter limit: ");
      int limit = sc.nextInt();

      Random rnd = new Random();
      int countUnique = 0;
      int countDuplicates = 0;
      int num; // the random number
```

Additional Exercise

Detecting Duplicates (4/4)

DetectDuplicates.java

```
        System.out.println("List: " + list);
        System.out.println("Duplicates detected: "
                           + countDuplicates);
    }
}
```

End of file