GC01 Introductory Programming

Week 1 – Lecture

Dr Ghita Kouadri Mostefaoui
Department of Computer Science

Outline

- Arrays
- ArrayLists
- Vectors
- *COMPGC01 Coursework*

Java Arrays

Arrays

- So far we have worked primarily with primitive variables
  - One characteristic of primitive variables is that they hold one value at a time.
    - `int x = 100` – can only hold one integer value at a time (100 at this time)
- Imagine that you want to hold the names of 100 people at a time.
  - What would you have to do?
    - Declare 100 variables, one for each name ☹
  - Better solution: Use Arrays.
    - Arrays are complex variables that can hold multiple values of the *same data type*.
    - Now we can declare a single array that holds all the names.

Arrays

To declare an integer array:

```
int[] numbers;
```

- It is just like declaring any primitive variable, EXCEPT for the `[]` between the data type and the identifier.
- Just like any other object, this does not create the array, it creates a reference variable that can point to an array.
- To create an array object for this reference variable:

```
 numbers = new int[5];
```

- The number inside of the square brackets is called the array's <u>size declarator</u>.
  - An array's Size Declarator indicates the number of <u>elements</u>, or values the array can hold.
    - Individual values inside of an array are called <u>elements</u>.
  - So the `numbers` array can hold 5 integers.
- You can declare and create on the same line like any other object:

```
int[] numbers = new int [5];
```

Arrays

- The elements in an array are numbered starting from 0.
  - So the first element has index 0.
- The size declarator must be a non-negative integer expression.
- Once an array is created the size cannot change!
- Arrays can be of any type we discussed up to this point.

Accessing Array Elements

- Even though, the array has one name, we can access the individual elements by their index (or subscript).
- To access an element of an array, simply use the array name followed by the element's index inside of square brackets.
  - The following statement assigns 20 to the first element in the `numbers` array:

```
numbers[0] = 20;
```

Array Elements

- Individual array elements can be treated like any simple variable of that type.

```
int[] numbers = new int[5];
int x = 5;
numbers[1] = x + 5;
x = numbers[1] + 20;
numbers[2] = numbers[1] / 5;
numbers[2]++;
```

- Also, since arrays are objects in Java, they have methods and data members.
  - For instance the length data member holds the number of elements in the array.
    - `int size = numbers.length;`
    - This is helpful for looping through an array.

Bounds Checking and Off-by-One Errors

- What is the problem with this?

```
int[] numbers = new int [3];
numbers[5] = 10;
```

  - Answer: The index 5 is out of the bounds defined by the size declarator.
- Java performs <u>bounds checking</u> at runtime.
  - In the case with Java that means that if you attempt to index an element outside of the array's bounds, it will throw an exception (cause an error and terminate your program).
    - Exceptions will be discussed in a future lecture.
- What is the problem with this?

```
int[] numbers = new int [3];
numbers[3] = 10;
```

  - Answer: The index 3 is still out of the array's bounds.
  - Because indexing starts at 0, the last element of the array is the size of the array minus 1.
  - This is called an <u>Off-by-One</u> error.

Array Declaration Notes

- You can initialize Arrays just like any other variables.

```
int[] numbers = {1, 2, 3, 4, 5};
```

  – This array has an implicit size of 5 and the values in the initialization are indexed from left to right.

- Java also allows for two different syntactical forms for declaring an array:

```
int[] numbers;
int numbers[];
```

- The difference is when you declare multiple arrays on the same line:

```
int[] numbers1, numbers2, numbers3;
```

  – This declares three reference variables to integer arrays

```
int numbers1[], numbers2, numbers3;
```

  – This declares one reference variable to an integer array and two primitive integer variables

```
int numbers1[], numbers2[], numbers3[];
```

  – This, again, declares three reference variables to integer arrays

Array traversal

- This seems repetitive…
  - For example, I have to use a separate segment of code for printing out the students names.
- Is there a better way?
  - Answer: Yes, using loops.
- Arrays work great with loops, because you can loop through the elements of the array and perform operations on them.

Array traversal example

```
public class ArrayTraversal {

  public static void main(String[] args){
      String[] students =  {"Bob", "Ela" , "Alex"};

      // traversing an array
      for (int i=0; i<3; i++){
          System.out.println("student: " + students[i]);
      }

  }
}
```

Enhanced `for` Loop

- The last kind of loop we will be discussing in this course is called the enhanced `for` loop.
  - The Enhanced `for` Loop is a special loop that iterates through the elements in a collection (in this case array) and allows access to each element.
  - Each iteration of the loop corresponds to an element in the array.
  - Introduced in Java 5
- General Form:

```
for(dataType elementVariable : array)
    block or statement
```

  - *`dataType`* – The type of the array
  - *`elementVariable`* – A variable that holds the value of the current iteration's element.
  - *`array`* – The array in which we are iterating through.

Enhanced `for` loop example

```
public class EnhancedForLoop {

  public static void main(String[] args){
      String[] students =  {"Bob", "Ela" , "Alex"};

      // traversing an array
      for (String s: students){
          System.out.println("student: " + s);
      }

  }
}
```

When NOT To Use the Enhanced `for` Loop

- The enhanced `for` loop is useful for when you need only all the values of the array, but there are many situations when you should NOT use it:
  1. If you need to change the contents of an array element
  2. If you need to work through the array elements in reverse order
  3. If you need to access some of the array elements, but not all of them
  4. If you need to simultaneously work with two or more arrays within the loop
  5. If you need to refer to the index of a particular element

- **Note**: Enhanced for loops are simple but "inflexible". They can be used when you wish to step through the elements of the array in first-to-last order, and you do not need to know the index of the current element. In all other cases, the "standard" for loop should be preferred.

Allowing the User to Specify an Array's Size

- Since the size declarator is an integer, you can let the user enter an integer, and then create an array of that size.

```
import java.util.Scanner;

public class UserSizeArray {
     public static void main(String[] args) {
          int studentsNumber;
          String[] students;

          Scanner console = new Scanner(System.in);

          System.out.print("How many students you want to register? ");
          studentsNumber = console.nextInt();

          students = new String [studentsNumber];


          for(int i = 0; i < studentsNumber; i++) {
               System.out.print("Enter student name " + (i+1) + ": ");
               students[i] = console.next();
          }

          System.out.println("\nThe names you entered are:");
          for(int i = 0; i < studentsNumber; i++) {
               System.out.print(students[i] + " ");
          }
     }
}
```

Java `arrayList`

What's an Array List

- `ArrayList` is
  - a class in the standard Java libraries that can hold any type of object
  - an object that can grow and shrink while your program is running (unlike arrays, which have a fixed length once they have been created)

- In general, an `ArrayList` serves the same purpose as an array, except that an `ArrayList` can *change length while the program is running.*

Using the `ArrayList` Class

- In order to make use of the **`ArrayList`** class, it must first be imported

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

- An **`ArrayList`** is created and named in the same way as object of any class, except that you specify the base type as follows:

```
ArrayList<BaseType> aList =
        new ArrayList<BaseType>();
```

Creating an `ArrayList`

- An initial capacity can be specified when creating an `ArrayList` as well
  - The following code creates an `ArrayList` that stores objects of the base type `String` with an initial capacity of 20 items

```
ArrayList<String> list = new ArrayList<String>(20);
```

  - Specifying an initial capacity does not limit the size to which an `ArrayList` can eventually grow
  - Question: Why start an ArrayList with an initial capacity?
  - http://stackoverflow.com/questions/15430247/why-start-an-arraylist-with-an-initial-capacity (optional reading)

Primitive data types are not allowed in an **`ArrayList`**

- Note that the base type of an ***`ArrayList`*** cannot be a primitive type but an object.
  - *int -> Integer*
  - *boolean -> Boolean*
  - *double -> Double*
  - *byte -> Byte etc...*

- *Why? Optional research question.*

Adding elements to an `ArrayList`

- The `add` method is used to add an element at the "end" of an `ArrayList`

```
list.add("something");
```

- There is also a two argument version that allows an item to be added at any currently used index position or at the first unused position

How many elements?

- The `size` method is used to find out how many indices already have elements in the `ArrayList`

```
int howMany = list.size();
```

- The `set` method is used to replace any existing element, and the `get` method is used to access the value of any existing element

```
list.set(index, "something else");
String thing = list.get(index);
```

- `size` is NOT capacity
  - size is the number of elements currently stored in the ArrayList
  - Capacity is the maximum number of elements which can be stored. Capacity will automatically increase as needed

ArrayList code Example

```
// Note the use of Integer, rather than int
public static void main( String[ ] args)
{
   ArrayList<Integer> myInts = new ArrayList<Integer>();
   System.out.println( “Size of myInts = “ + myInts.size());
   for (int k = 0; k < 10; k++)
       myInts.add( 3 * k );
   myInts.set( 6, 44 );
   System.out.println( “Size of myInts = “ + myInts.size());
   for (int k = 0; k < myInts.size(); k++)
       System.out.print( myInts.get( k ) + “, “  );
}
```

Methods in the Class **`ArrayList`**

- The tools for manipulating arrays consist only of the square brackets and the instance variable **`length`**

- **`ArrayLists`**, however, come with a selection of powerful methods that can do many of the things for which code would have to be written in order to do them using arrays

More example code

```
// Note the use of Integer instead of int
public static void main( String[ ] args)
{
   ArrayList<Integer> myInts = new ArrayList<Integer>(25);
   System.out.println( “Size of myInts = “ + myInts.size());
   for (int k = 0; k < 10; k++)
   myInts.add( 3 * k );
   myInts.set( 6, 44 );
   myInts.add( 4, 42 );
   myInts.remove( new Integer(99) );
   System.out.println( “Size of myInts = “ + myInts.size());
   for (int k = 0; k < myInts.size(); k++)
    System.out.print( myInts.get( k ) + “, “  );
   if (myInts.contains( 57 ) )
    System.out.println(“57 found”);
   System.out.println (“44 found at index “ + myInts.indexOf(44));
}
```

The enhanced `for` loop

- The *enhanced for loop* discussed earlier can also be used with an `ArrayList`.
- The enhanced loop is also known as the *'for-each loop'*. It has been added to java since version 5.0.
- This kind of loop has been designed to cycle through all the elements in a collection (like an `ArrayList`)

“for-each” example

```
public class ForEach
{
   public static void main(String[ ] args)
   {
    ArrayList<Integer> list = new ArrayList<Integer>();
    list.add( 42 );
    list.add( 57 );
    list.add( 86 );

    // “for each Integer, i, in list”
    for( Integer i : list )
       System.out.println( i );
   }
}
//-- Output ---
42
57
86
```

`ArrayList` vs `Array`

Why use an array instead of an `ArrayList`

1. An `ArrayList` is less efficient than an array
2. `ArrayList` does not have the convenient square bracket notation
3. The base type of an `ArrayList` must be a class type. It cannot accept primitive types.

`ArrayList` vs `Array`

Why use an `ArrayList` instead of an array?

1. Arrays can't grow. Their size is fixed at compile time.
   - `ArrayList` grows and shrinks as needed while your program is running
2. You need to keep track of the actual number of elements in your array (recall partially filled arrays).
   - `ArrayList` will do that for you.
3. Arrays have no methods (just `length` instance variable)
   - `ArrayList` has powerful methods for manipulating the objects within it

Java `Vectors`

The `Vector` Class

- The Java standard libraries have a class named **`Vector`** that behaves almost exactly the same as the class **`ArrayList`**

Differences

- Vectors are synchronized, **`ArrayLists`** are not.
- Data Growth Methods.
  - A Vector defaults to doubling the size of its array, while the ArrayList increases its array size by 50 percent.
  - More info here: http://beginnersbook.com/2013/12/difference-between-arraylist-and-vector-in-java/

The `Vector` Class

- The class Vector can be used to implement a list, replacing a simple array

- The Vector will automatically grow to accommodate the number of elements you put in it.

The **`Vector`** Class (continued)

- The class Vector is contained in the package `java.util`

- Programs must include either:
  - `import java.util.*;`
  - `import java.util.Vector;`

Vector Declaration

- Declare/initialize

```
Vector<String> students = new Vector<String>();
```

- The syntax <…> is used to declare the type of object that will be stored in the Vector

- If you add a different type of object, an exception is thrown

Vector Size/Capacity

- The *size* of a vector is the number of elements.
- The *capacity* of a vector is the maximum number of elements before more memory is needed.
- If size exceeds capacity when adding an element, the capacity is automatically increased.
- By default, the capacity doubles each time

Setting Initial Capacity

- If you know you will need a large vector, it may be faster to set the initial capacity to something large

```
Vector<Student> students = new Vector<Student>(1000);
```

Size and capacity

- Get the current size and capacity:

```
Vector<Student> students = new Vector<Student>(1000);
students.size();      // returns 0
students.capacity();  // returns 1000
```

`Vector` and the for-each loop

- Each `Vector` object is a collection of elements
  - You can use a for-each loop to process its elements
  - Exactly like using a foreach loop with an array or arrayList.
- Syntax:

```
for (type identifier : vectorObject)
  statements
```