Data Structures and Algorithms

Lecture 3

Mark Cudden
Mark.Cudden@ncirl.ie

3. Java Collections: ArrayList

3.1 Java Collections: ArrayList

Features

- ArrayList = An auto-resizing array that can hold any type of object
- In several respects an ArrayList may be considered a better array.
  - **It is generic – it holds elements of class Object**
  - **It dynamically adjusts its capacity to meet the storage needs**
  - **It automatically adjusts its contents when an object is inserted or removed at an arbitrary location (index)**

- ArrayList class is part of `java.util` package
- So, you have to import `java.util.*;` in order to use this class.

- ArrayList is basically the same as the older Vector class

3.1 Java Collections: ArrayList

Features

- Generic class
  - A type of class in Java that is written to accept another type as part of itself.
  - Generic ("parameterized") classes were added to Java (1.6 version) to improve the type safety of Java's collections.
  - A parameterized type has one or more other types' names written between < and >.
- `ArrayList<E>` is a generic class.
  - The `<E>` is a placeholder in which you write the type of elements you want to store in the `ArrayList`.
  - Example:

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

    - Now the methods of the object `words` will manipulate and return Strings

3.1 Java Collections: ArrayList

Features

- An ArrayList can only hold objects (descendants of class Object)
  - Primitive values are not objects
  - Primitive types must be “wrapped” before they are inserted into an ArrayList.
  - If you want to store primitives in an `ArrayList`, you must declare it using a "wrapper" class as its type

```
ArrayList<Integer> list = new ArrayList<Integer>();
```

  - Objects removed from an ArrayList must be cast into their appropriate derived class

Primitive type	Wrapper class
int	Integer
double	Double
char	Character
boolean	Boolean

3.1 Java Collections: ArrayList

Features

- Primitive type vs Wrapper class
- Using primitive type

```
int i;
i=24;
System.out.println(i);
```

- Using Wrapper class

```
Integer x;
x= new Integer(24);
System.out.println(x.intValue());
```

3.1 Java Collections: ArrayList

Features

- Example of code

```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add(new Integer(13));
list.add(new Integer(47));
list.add(new Integer(15));
list.add(new Integer(9));
```

- This code is correct, too!

```
Integer num;
ArrayList<Integer> list;
list = new ArrayList<Integer>();
num = new Integer(13);
list.add(num);
num = new Integer(47);
list.add(num);
```

3.1 Java Collections: ArrayList

Methods

Method name	Description
`add(`*value*`)`	adds the given value to the end of the list
`add(`*index*`, `*value*`)`	inserts the given value before the given index
`clear()`	removes all elements
`contains(`*value*`)`	returns `true` if the given element is in the list
`get(`*index*`)`	returns the value at the given index
`indexOf(`*value*`)`	returns the first index at which the given element appears in the list (or -1 if not found)
`lastIndexOf(`*value*`)`	returns the last index at which the given element appears in the list (or -1 if not found)
`remove(`*index*`)`	removes value at given index, sliding others back
`size()`	returns the number of elements in the list

3.2 ArrayList: Adding elements

- Elements are added dynamically to the end of the list

```
ArrayList<String> list = new ArrayList<String>();
list.add("John");
list.add("Paul");
list.add("Denise");
```

- Elements can be added in a given position in the list

```
list.add(2, "Anne-Marie");
```

3.3 ArrayList: Printing elements

- Elements can be printed using toString() method

```
ArrayList<String> list = new ArrayList<String>();
System.out.println("Elements of the list are:"+list.toString());
```

- Printing process can be personalised using an iterator

3.4 ArrayList: Parsing the elements

- An Iterator object can be attached to the ArrayList to access the elements one by one and to perform an operation on each element
  - Example of operations: printing / changing the value / setting a value of each element
- Code example

```
Iterator iter;
iter =  list.iterator();
while (iter.hasNext())
{
    System.out.println("Element:" + iter.next());
}
```

3.5 ArrayList: Removing elements

- One element can be removed by index

```
list.remove(1);
```

- One element can also be removed by value

```
list.remove("John");
```

- All elements can be removed in one operation

```
list.clear();
```

3.6 ArrayList: Getting one element

- The value of one element can be obtained without removing it
- The value of the element at the specified position (index)

```
String val;
val = list.get(1);
```

3.7 ArrayList: Searching elements

- You can search the ArrayList for particular elements

```
if (list.contains("John")) {
    System.out.println("John is in the list");
} else {
    System.out.println("John is not found.");
}

int index;
if (list.contains("John")) {
    index = list.indexOf("John");
    System.out.println(index + " " +
  list.get(index));
}
```

- `contains` tells you whether an element is in the list or not, and `indexOf` tells you at which index you can find it.

3.8 ArrayList: Example of Application

- Develop ArrayListExample project (in NetBeans) to practice some operations performed on an ArrayList object
- The skeleton NetBeans code is a available on the Moodle Web site
  - Download the ArrayListExample_skeleton.zip file
  - Unzip the file in your home folder => an ArrayListExample folder is created
  - Open the project from NetBeans IDE
    - File => Open Project
  - Add your Java code to perform the tasks indicated as comments in the code
  - Compile and Run the program

3.9 ArrayList: Ordering the elements

- An `ArrayList` of orderable values can be sorted using the `Collections.sort` method

```
ArrayList<String> words = new ArrayList<String>();
words.add("four");
words.add("score");
words.add("and");
words.add("seven");
words.add("years");
words.add("ago");

// show list before and after sorting
System.out.println("before sort, words = " + words.toString());
Collections.sort(words);
System.out.println("after sort, words = " + words.toString());
```

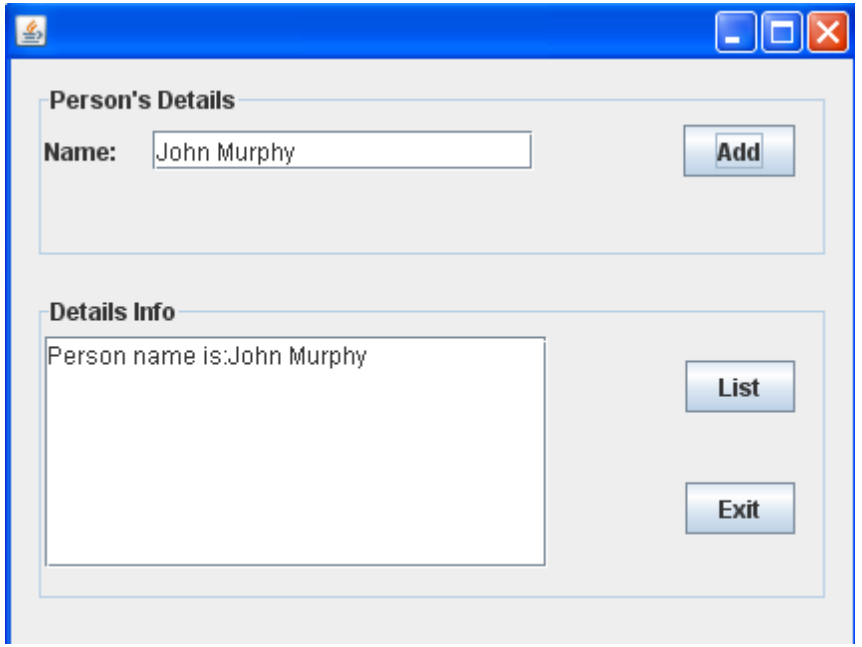
- Output:

```
before sort, words = [four, score, and, seven, years, ago]
after sort, words = [ago, and, four, score, seven, years]
```

- To shuffle the elements use `Collections.shuffle` method

3.10 LAB: GUIApp using ArrayList

- Week 1: Learn about NetBeans and GUI development
  - We developed GUIApp in NetBeans



- Each name we typed in was displayed in the JTextArea
  - Added functionality to the Add and Exit buttons

3.10 LAB: GUIApp using ArrayList

- Extend your application such as:
  - Each name we type in is added into an ArrayList object
  - **List** button functionality => all elements of the ArrayList are printed in JTextArea

- Add functionality to the List button
  - Right Click on the Add button. Events --> Action --> ActionPerformed.
  - IDE automatically adds an ActionListener to the Add button and opens up the Source Code window
  - Write code for jButton2ActionPerformed() method

```
}
}
```

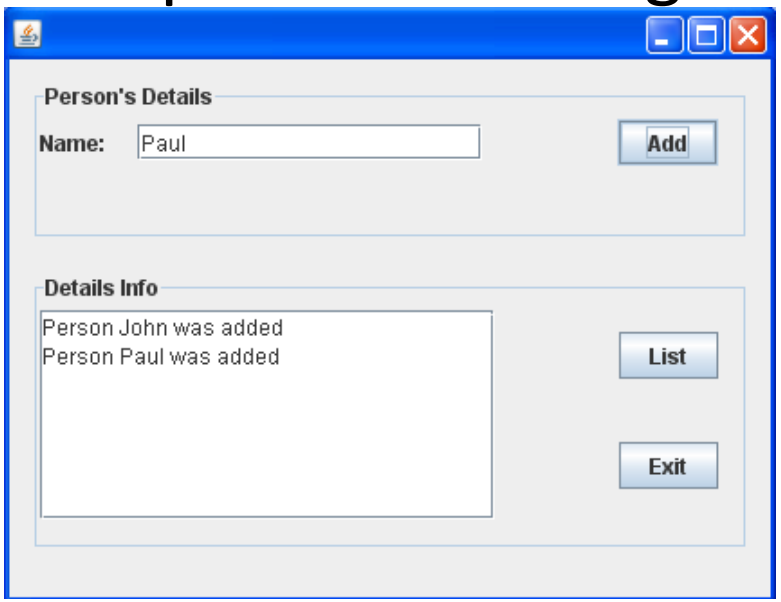
Import java.util.*

NOTE:

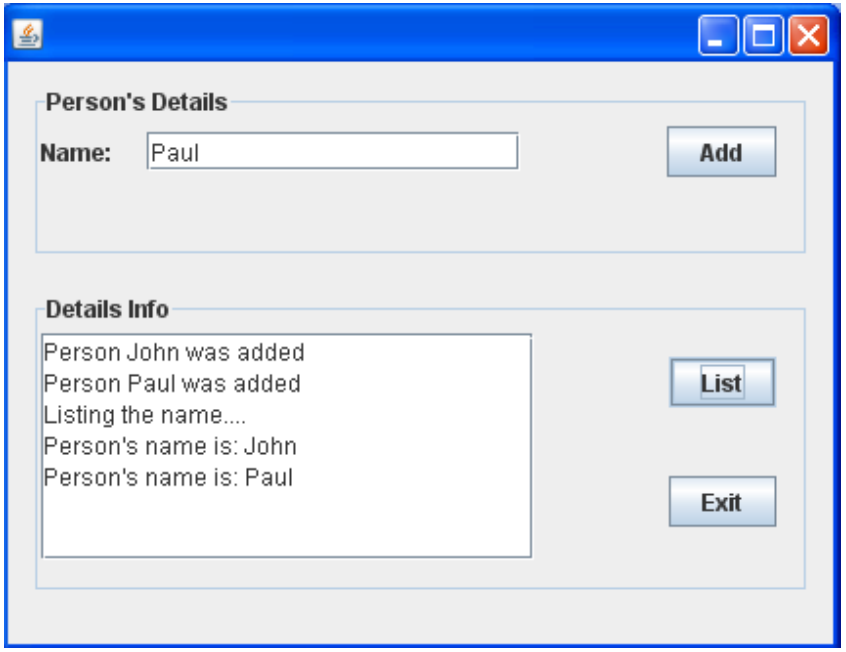
1) **Need to declare the ArrayList Object as a member of the PersonsGUI class**
2) **Need to create the ArrayList Object in the constructor of the class**
3) **We need to update the code from actionPerformed Add button to put the name into the ArrayList object**

3.10 LAB: GUIApp using ArrayList

- Output after adding 2 names



- Output after pressing List



3.10 LAB: GUIApp using ArrayList

- Optional task: Extend again your application such as:
  - Add **Remove** button – Removes a person with a given name from the ArrayList
  - Add **Sort** button – Sorts alphabetical the names from the ArrayList collection

Learning Outcome

- Java Collections: ArrayList
  - Declaring and creating an ArrayList object
  - Adding/ searching/ removing/ getting an element
  - Parsing and printing elements using
    - toString() method
    - iterator