



Object Oriented Programming

Instructor Name:

Lecture-8-9

Today's Lecture

- **Arrays in Java**
- **foreach Loop**
- **Grouping Objects**
- **Making use of Library Classes**

Array

What is an Array

- An *array* is a container object that holds a fixed number of values of a single type.
- The length of an array is established when the array is created.
- Here is the way to create an array in java

```
int[] array;
```

```
array=new int[10];
```

or

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

Arrays

Printing an Array

```
class arrayTest{  
  
    public static void main(String arg[]){  
        int a[]={1,2,3,4,5,6,7};  
        for(int i=0; i<a.length;i++)  
            System.out.println(a[i]);  
    }  
}
```

OUTPUT

1
2
3
4
5
6
7

Arrays

Sorting Array Using Built-in Functions

```
import java.util.Arrays;
class sortAttay{

    public static void main(String arg[]){
        int a[]={11,2,23,4,15,6,17};
        for(int i=0; i<a.length;i++)
            System.out.println(a[i]);
        Arrays.sort(a);
        for(int i=0; i<a.length;i++)
            System.out.println(a[i]);
    }
}
```

This will sort the array in ascending order.

Arrays

Limitations with Arrays

- Elements can not be added dynamically.
- Elements cannot be removed automatically
- Array size must be define at start

Collection in Java

Strength of Java

- As we have seen great strength of java is its rich set of predefined classes that you can reuse rather than “reinventing the Wheel” again and again
- The classes are grouped into packages- named group of related classes and are collectively referred to as java class library or java application programming interface(API).
- One of the API is **Collection** which is used to store the group of related objects
- A Collection object can store an arbitrary number of other objects.
- Efficient method to store , organize and retrieve the data with out requiring additional knowledge of how to store it.
- Have the ability to dynamically change the size to accommodate to more element
- Reduce size when deleting element from collection.

Collection in Java

Collection Examples

- Here are some examples of collection that are related to programming context.
- Electronic calendars store event notes about appointments, meetings, birthdays, and so on. New notes are added as future events are arranged, and old notes are deleted as details of past events are no longer needed.
- Libraries record details about the books and journals they own. The catalog changes as new books are bought and old ones are put into storage or discarded.
- Universities maintain records of students. Each academic year adds new records to the collection, while the records of those who have left are moved to an archive collection. Listing subsets of the collection will be common: all the students taking first-year CS or all the students due to graduate this year, for instance.

ArrayList<T>

What is ArrayList

- The *ArrayList* is a Java class that is inherited from (or extends) the *AbstractList* class. The *ArrayList* class implements the *List* interface.

How to Use ArrayList

- In order to use the *ArrayList* class in your projects, you must import this class as follows:

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

Declaring an ArrayList

- After importing the Java *ArrayList* class, it is time to declare an object of the *ArrayList* class. The default way is:

```
ArrayList array_name = new ArrayList();
```

- An array “*array_name*” of the ten size is created. The *ArrayList* default constructor creates an array of ten size, by default.

ArrayList<T>

Key Points of ArrayList

- A Java ArrayList is a dynamic array, an array that can grow after it is created.
- The standard [array](#) are not dynamic i.e. once a standard array is created it cannot grow. These are of fixed size and you cannot add or remove elements after these are created whereas the ArrayList allows that.
- ArrayList in Java gives fast iteration.
- It also gives fast random access.
- The ArrayList provides more powerful insertion than the standard arrays.
- Search mechanism in ArrayList is better than standard arrays.
- The ArrayList manipulation is slower as a lot of shifting occur.
- Works with non-primitive data types only.

ArrayList<T>

Most Common Methods

Methods	Description
Add	Add an element at the end of collection
Clear	Remove all element
Contains	Return true if element is present otherwise false
Get	Return element at specific index
indexOf	Return the index of first occurrence
Remove	Remove from speicific index
Size	Size f arrayList
trimToSize	Trim to capacity to current number of elements

ArrayList<T>

ArrayList Example

```
import java.util.ArrayList;
public class ArrayListDemo {
    public static void main(String args[]) {
        ArrayList<String> al = new ArrayList<String>();
        System.out.println("Initial size of al: " + al.size());
        al.add("C");
        al.add("A");
        al.add("E");
        al.add("B");
        al.add("D");
        al.add("F");
        al.add(1, "A2");
        System.out.println("Size of al after additions: " +
al.size());
        System.out.println("Contents of al: " + al);
        al.remove("F");
        al.remove(2);
        System.out.println("Size of al after deletions: " +
al.size());
        System.out.println("Contents of al: " + al); } }
```

ArrayList<T>

ArrayList Example – Output

Initial size of al: 0

Size of al after additions: 7

Contents of al: [C, A2, A, E, B, D, F]

Size of al after deletions: 5

Contents of al: [C, A2, E, B, D]

A Notebook Example



- This interface helps the implementor decide on the class's operation/methods.

The Notebook Class

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

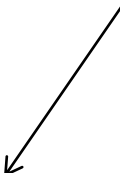
```
public class Notebook  
{
```

```
    private ArrayList<String> notes;
```

```
    public Notebook()
```

```
    {    notes = new ArrayList<String>();    }
```

The list will store
String objects, and
is called notes



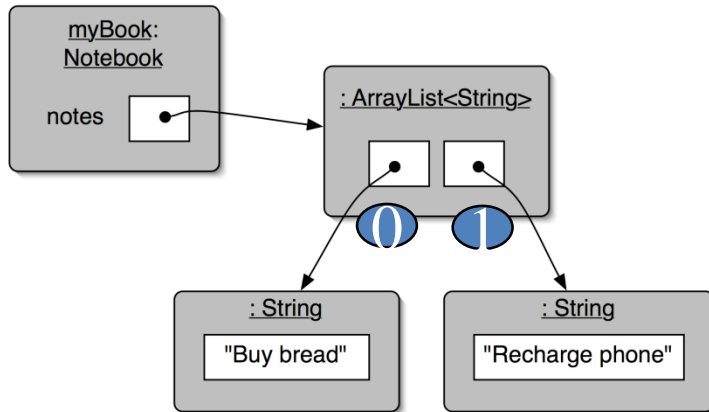
continued

ArrayList.add(),
ArrayList.remove()
ArrayList.size()

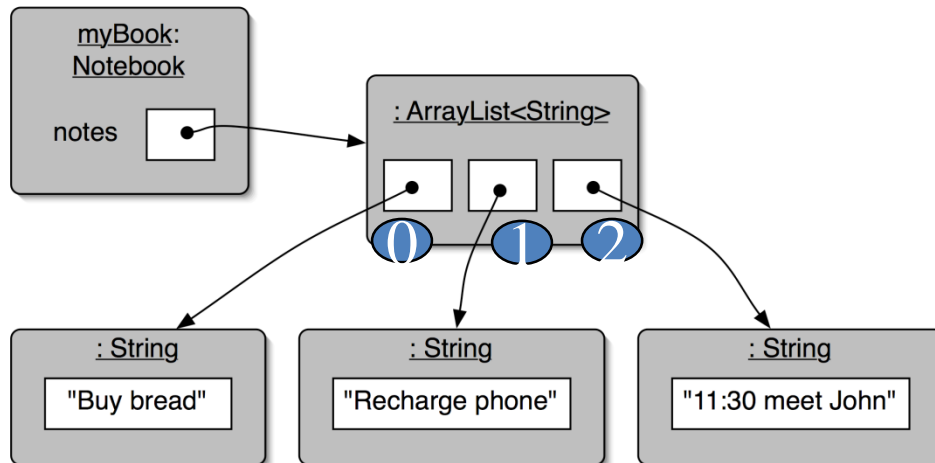
```
public void storeNote(String note)
// add a note (a string) to the notebook
{  notes.add(note); }
```

```
public void removeNote(int noteIdx)
// Remove a note from the notebook if it exists.
{
    if ((noteIdx >= 0) && (noteIdx < notes.size()))
        // a valid note number
        notes.remove(noteIdx);
}
```


Using add()

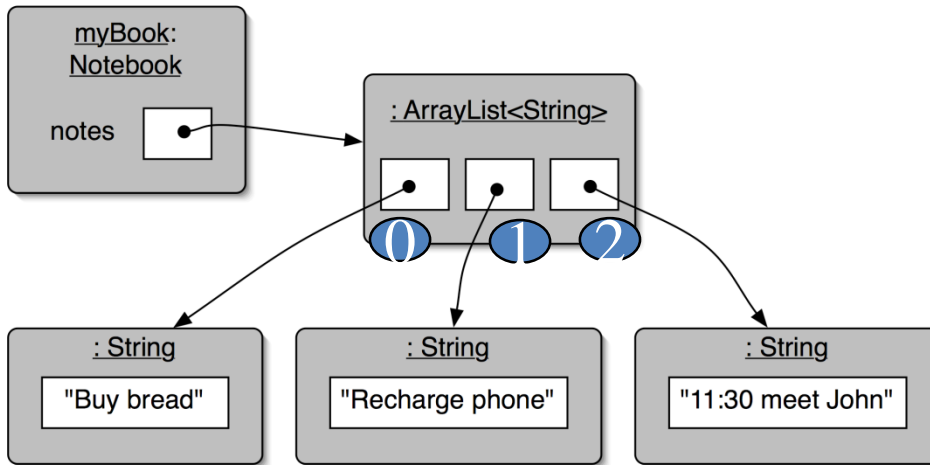


`notes.add("11:30 meet John");`

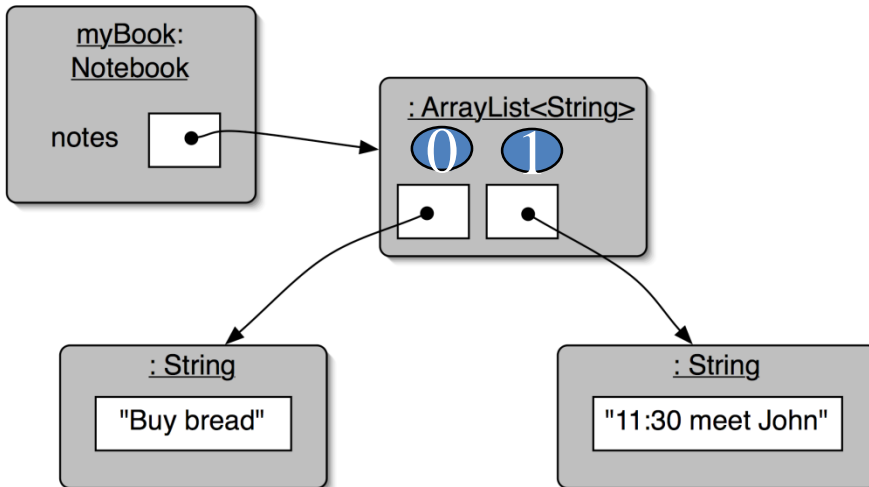


`notes.size()` now
returns 3

Using remove()



`notes.remove(1);`



The index position of the "meet" note changes when the second object is removed.

`notes.size()` is now 2

The Notebook Class (continued)

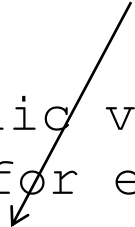
```
public void showNote(int noteIdx)
{
    if ((noteIdx >= 0) &&
        (noteIdx < notes.size()))
        // if a valid note number
        System.out.println( notes.get(noteIdx) );
}
```

```
public int numNotes()
{   return notes.size(); }
```

ArrayList.get()
returns a link to the
object at index
noteIdx

continued

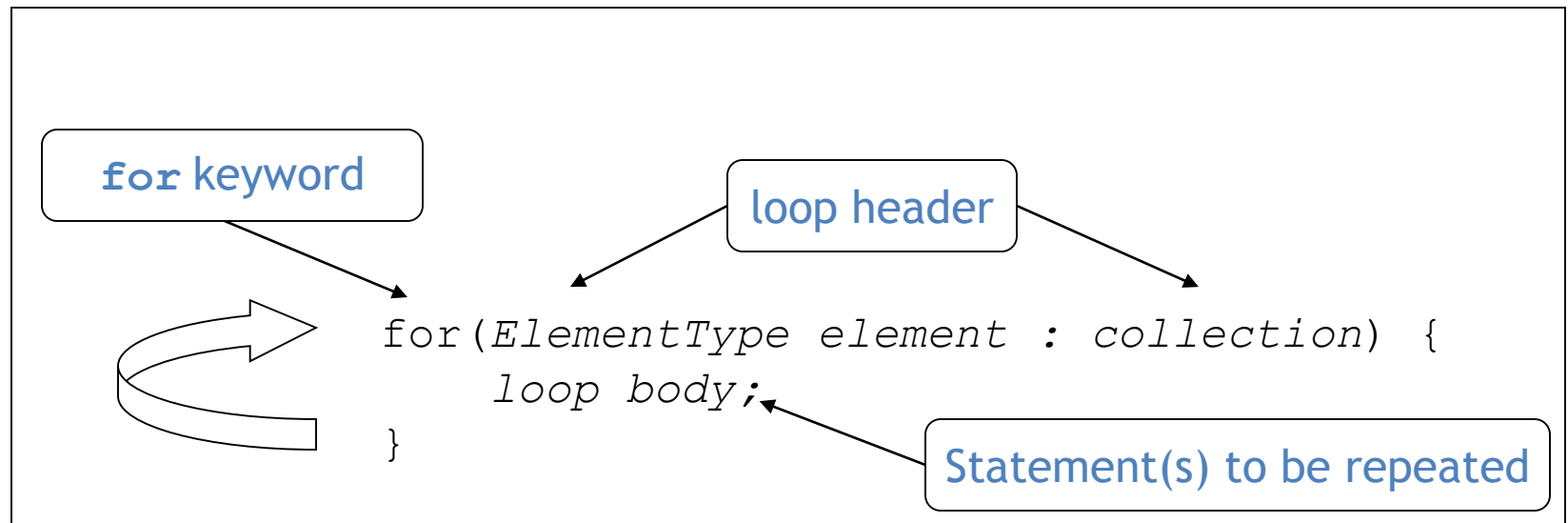
The Java for-each loop



```
public void listNotes()  
    // for each note in notes, print it  
    { for (String note : notes)  
        System.out.println(note);  
    }
```

```
}    // end of Notebook class
```

The For-each Loop



For each element in collection,
do the statements in the loop body.

Collection

Generic Vs Non Generic Collection

- Java collection framework was non-generic before JDK 1.5. Since 1.5, it is generic.
- Java new generic collection allows you to have only one type of object in collection. Now it is type safe so typecasting is not required at run time.
- Let's see the old non-generic example of creating java collection.

```
ArrayList al=new ArrayList();
```

- Let's see the new generic example of creating java collection.

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

- In generic collection, we specify the type in angular braces. Now ArrayList is forced to have only specified type of objects in it. If you try to add another type of object, it gives *compile time error*.

Generic Classes

- Collections are known as *parameterized* or *generic* classes.
- The type parameter says what we want in the list:
 - `ArrayList<String>`
 - `ArrayList<TicketMachine>`
 - etc.
- `ArrayList` implements list functionality, and there are other classes in `java.util` that implement queues, sets, maps, etc.

Using Notebook

```
public class NotebookDemo
{
    public static void main(String[] args)
    {
        Notebook book = new Notebook();

        System.out.println("Store note: \"Teaching maths\");
        book.storeNote("Teaching maths");

        System.out.println("Store note: \"Teaching Java\");
        book.storeNote("Teaching Java");

        System.out.println("No. of notes: " + book.numNotes());
        :
```



```
System.out.println("Note 1: ");
book.showNote(1);
System.out.println("Note 2: ");
book.showNote(2);

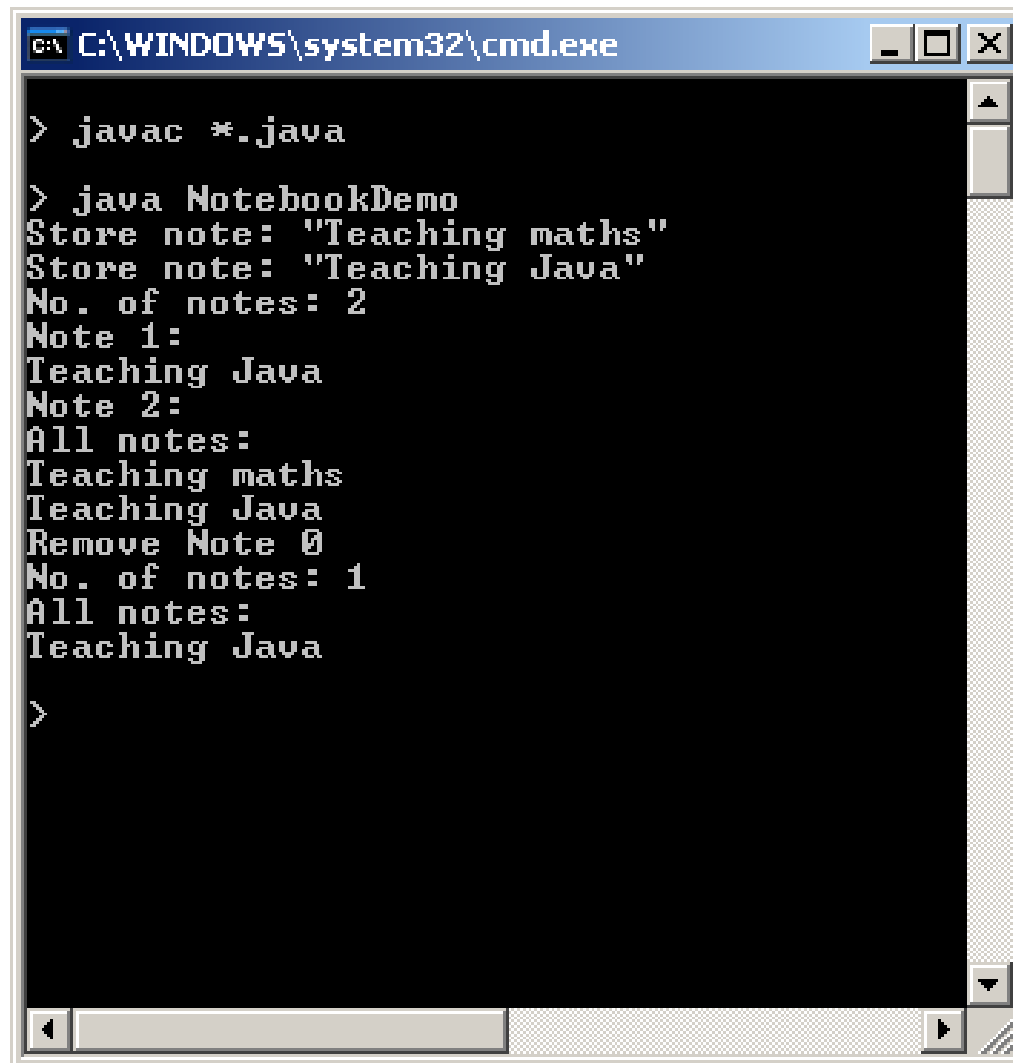
System.out.println("All notes: ");
book.listNotes();

System.out.println("Remove Note 0");
book.removeNote(0);

System.out.println("No. of notes: " + book.numNotes());
System.out.println("All notes: ");
book.listNotes();
} // end of main()

} // end of NotebookDemo class
```

Execution



```
C:\WINDOWS\system32\cmd.exe

> javac *.java

> java NotebookDemo
Store note: "Teaching maths"
Store note: "Teaching Java"
No. of notes: 2
Note 1:
Teaching Java
Note 2:
All notes:
Teaching maths
Teaching Java
Remove Note 0
No. of notes: 1
All notes:
Teaching Java

>
```

Iteration (looping)

- We often want to perform some actions an arbitrary number of times.
 - e.g., print all the notes in the notebook
- Java has several sorts of loop statement
 - familiar ones: for, while, do-while
 - new one: for-each

listNotes() using 'while'

compare with the
for-each version
on slide 37

```
public void listNotes()  
{  
    int index = 0;  
    while(index < notes.size()) {  
        System.out.println(notes.get(index));  
        index++;  
    }  
}
```

while the value of *index* is less than the size of the collection,
print the next note, and then increment *index*

for-each versus while

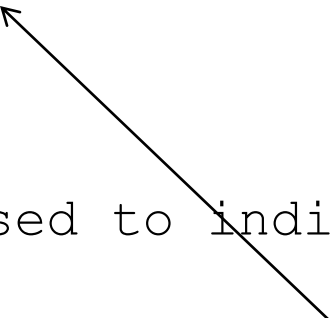
- for-each:
 - easier to write
 - safer: it's guaranteed to finish
- while:
 - processing order can be varied
 - 'while' can be used with data structures other than collections
 - take care: a 'while' loop can go into an *infinite loop* (never stops)

'While' Without a Collection

```
// print all even integers from 0 to 30
int index = 0;
while(index <= 30) {
    System.out.println(index);
    index = index + 2;    // steps of 2
}
```

Searching a Collection

```
public int findNote(String searchString)
{
    int index = 0;
    while(index < notes.size()) {
        String note = notes.get(index);
        if(note.contains(searchString))
            return index;
        index++;
    }
    return -1;    // used to indicate failure
}
```



contains() is from the
String class

R@CAP

