More on Collections

Lecture 3

Motivation for this study

- What type of 1-dimensional data structure is supported by Java?

- How do we create 1-dimensional data structure under Java?
- How do we use them? How to maintain them?
- Can we iterate through a 1-dimensional data structure under Java?

- More examples?

Menu

- Collections and List
- Using List and ArrayList
- Iterators

Comments on code style

- We will drop “this.” except when needed.
  - instead of `this.loadFromFile(fname)`
    just `loadFromFile(fname)`

- We will leave out { } when surrounding just one statement
  - instead of

```
while (i < name.length) {
   name[i] = null;
}
```

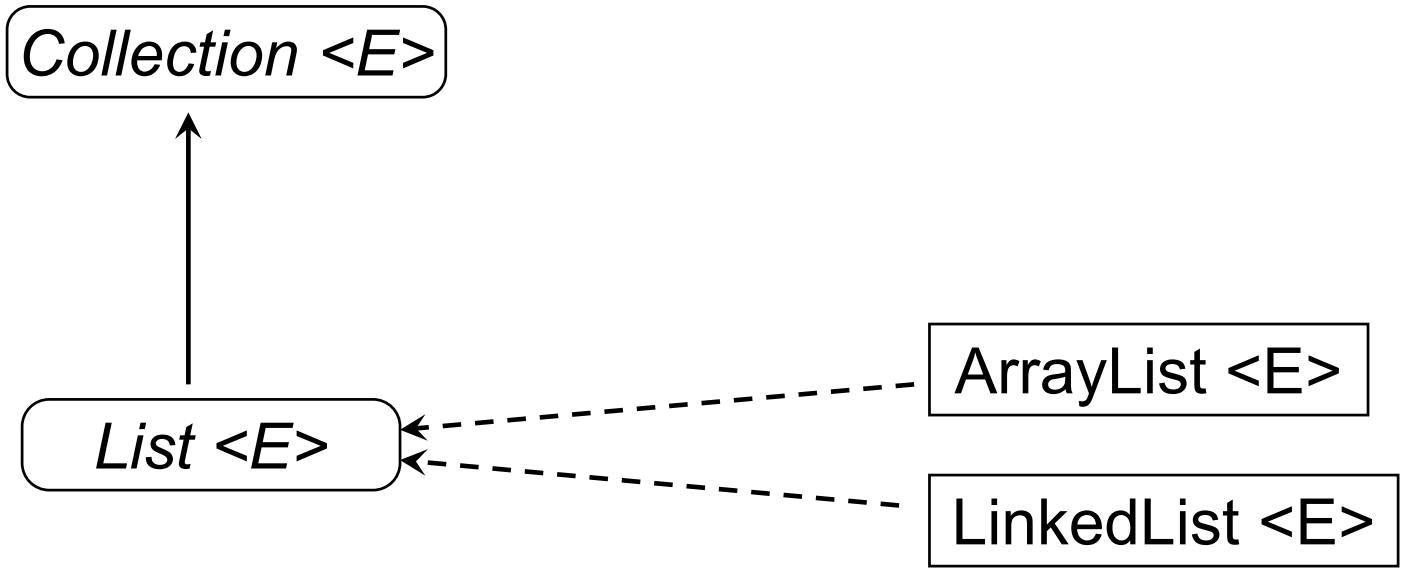
    just

```
while (i < name.length)
   name[i] = null;
```

Collection Types

Interfaces

Classes



Interfaces can **extend** other interfaces:

The **sub** interface has all the methods of the **super** interface plus its own methods (**sub** means? **super** means?)

Methods on Collection and List

- Collection <E>
  - isEmpty() → boolean
  - size() → int
  - contains(E elem) → boolean
  - add(E elem) → boolean (whether it succeeded)
  - remove(E elem) → boolean (whether it removed an item)
  - iterator() → iterator <E>
  - …
- List <E>
  - add(int index, E elem)
  - remove(int index) → E (returns the item removed)
  - get(int index) → E
  - set(int index, E elem) → E (returns the item replaced)
  - indexOf(E elem) → int
  - subList(int from, int to) → List<E>
  - …

Methods on all types of collections

Additional methods on all Lists

Using a collection type

- Variable or field declared to be of the interface type
  - Specify the type of the collection
  - Specify the type of the value

```
private List <Task>  tasks;
```

  - The type between "<" and ">" is the type of the elements

- Create an object of a class that implements the type:
  - Specify the class
  - Specify the type of the value

```
tasks = new ArrayList <Task> ();
```

- Call methods on the object to access or modify

Example

- TodoList – collection of tasks, in order they should be done.
- Collection type: List of tasks

- Requirements of TodoList:
  - read list of tasks from a file,
  - display all the tasks
  - add task, at end, or at specified position
  - remove task,
  - move task to a different position.

Example (TodoList program)

```
public class TodoList implements ActionListener{
    :
  private List<Task> tasks;
    :
  /* read list of tasks from a file, */
  public void readTasks(String fname){
    try {
        Scanner sc = new Scanner(new File(fname));
        tasks = new ArrayList<Task>();
        while ( sc.hasNext() )
            tasks.add(new Task(sc.next()));
        sc.close();
     } catch(IOException e){…}
    displayTasks();
  }
```

Iterating through List:

```
public void displayTasks(){
    textArea.setText(tasks.size() +“ tasks to be done:\n”);
    for (Task task : tasks){
        textArea.append(task + "\n");
}
```

Automatically calls toString() method. What is being displayed?

Or

```
for (int i=0; i<tasks.size(); i++)
    textArea.append(tasks.get(i) + "\n");
```

Or

```
iterator <Task> iter = tasks.iterator();
while (iter.hasNext()){
    textArea.append(iter.next() +  "\n");
```

More of the TodoList example:

```
public void actionPerformed(ActionEvent e){
    String but = e.getActionCommand();
    if ( but .equals("Add") )            tasks.add(askTask());
    else if (but.equals("Remove") ) tasks.remove(askTask());
    else if (but.equals( "AddAt" ) )
        tasks.add(askIndex("add where?"), askTask());
    else if (but.equals("RemoveFrom") )
        tasks.remove(askIndex("from"));
    else if (but.equals("MoveTo") ){
        int from= askIndex(“move from position: ");
        int to = askIndex(“move to position: ")
                    Task task= tasks.get(from);
                    tasks.remove(from);
                    tasks.add(to, task);
        }
    displayTasks();
}
```

Iterators

How does the “for each” work?

```
for (Task task : tasks){
    textArea.append(task + "\n");
```

An iterator object attached to tasks that will keep giving you the next element

- **Turns into**

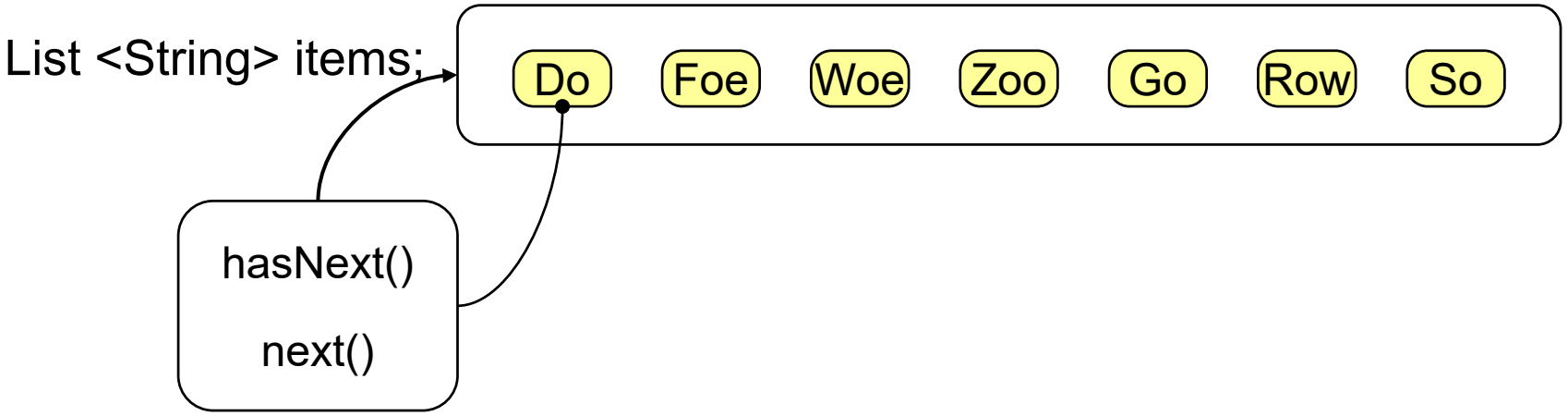
```
Iterator <Task> iter = tasks.iterator();
while (iter.hasNext()){
    Task  task = iter.next();
     textArea.append(task +  "\n");
```

Iterator is an interface:

A Scanner is a fancy iterator

```
public interface Iterator <E> {
    public boolean hasNext();
    public E next();
}
```

Iterators




```
for ( String str : items)   System.out.print(str + ", ");

Iterator<String> iter = items.iterator();
while (iter.hasNext()){
   String str = iter.next();
   System.out.print(str + ", ");
}
```

Q&A: Compare List against Array in the following aspects

- Lists are nicer than arrays:
  - … …
  - … …
  - List

	Array
jobList.set(ind, value)	?
jobList.get(ind)	?
jobList.size()	?
jobList.add(value)	?
jobList.add(ind, value)	?
jobList.remove(ind)	?
jobList.remove(value)	?

  - **for** (Task t : tasks) *vs* **for**(…) { … …}

List *vs* Array

- Lists are nicer than arrays:
  - No size limit!!! They grow bigger as necessary
  - Lots of code written for you:

jobList.set(ind, value)	jobArray[ind] = value
jobList.get(ind)	jobArray[ind]
jobList.size()	? *(Not the length!!!)*
jobList.add(value)	? *(Where is the last value? What happens if it's full?)*
jobList.add(ind, value)	? *(Have to shift everything up!!!)*
jobList.remove(ind)	? *(Have to shift everything down!!!)*
jobList.remove(value)	? *(Have to find value, then shift things down!!!)*

  - **for** (Task t : tasks) *vs* **for**(int i = 0; i< ???; i++){
    Task t = taskArray[i];

Q&A: How does ArrayList work?

- What does it store inside?
- How does it keep track of the size?
- How does it grow when necessary?
- How does its iterator work?

Summary

- Collections and List
- Using List and ArrayList
- Iterators

Readings

- [Mar07] Read 3.4
- [Mar13] Read 3.4