Java ArrayList Class

Download form TREES→CODE→Week12:
ALProbs.java
ArrayListTester.java

We will MODIFY StudentTester.java

An `ArrayList` also store large amounts of data in a single collection that can be referred to with a single variable.

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

- The `ArrayList` class provides a dynamic array implementation
  - stores elements of the same type sequentially
  - grows as needed (double its size)
  - useful if the size of the array is unknown
  - implements the List interface
- Declaring

```
ArrayList <E> arrayReferenceVariable;
```

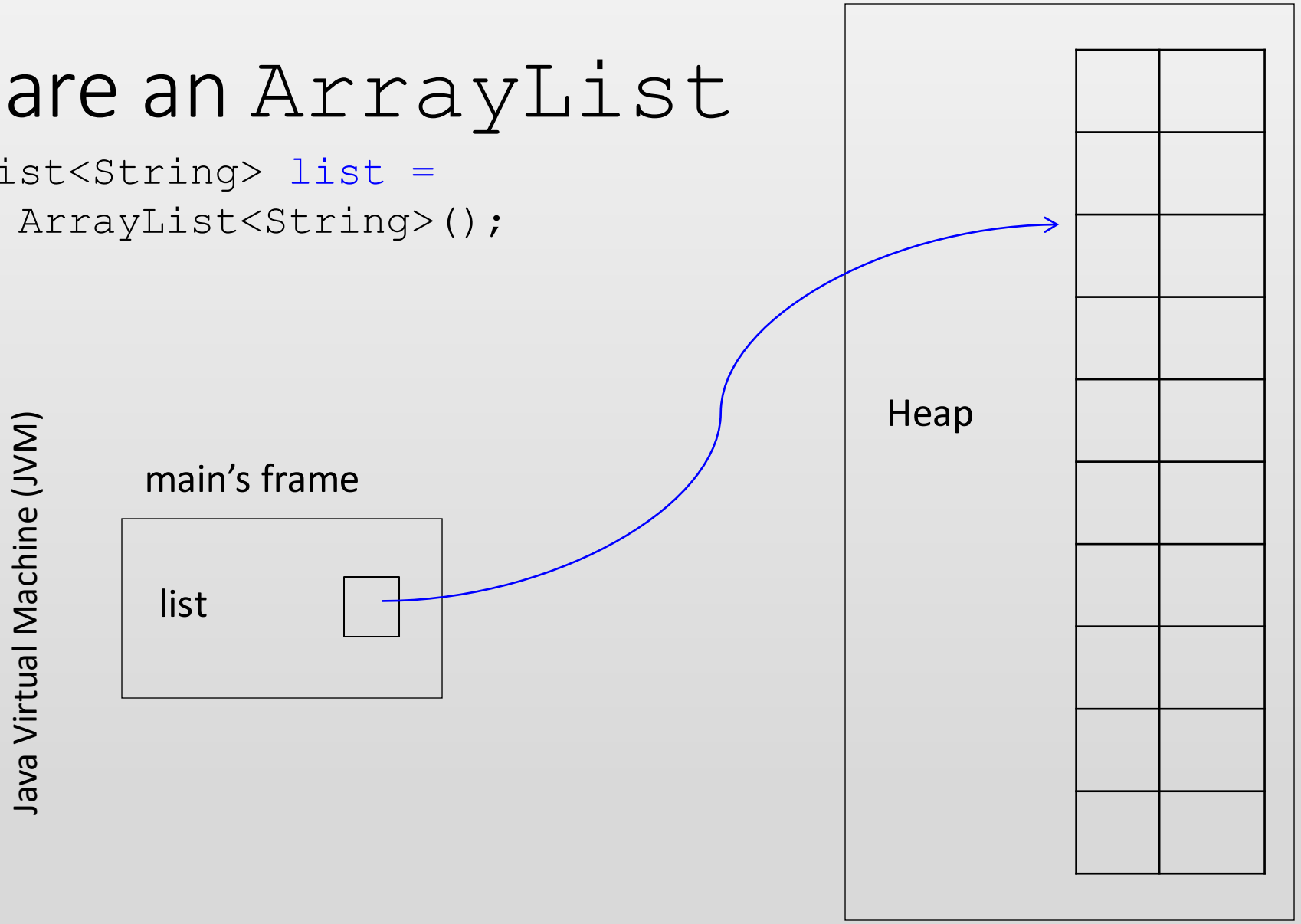
Item data type

ArrayList

- Each element in the sequence can be accessed separately.
- We can explicitly overwrite an object at a specified position in the sequence, thus changing its value.
- We can inspect the object at a specified location in the sequence.
- We can add an object into a specified position of the sequence.
- We can add an object to the end of the sequence.
- We can remove an object from a specified location in the sequence.

Declare an `ArrayList`

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



ArrayList

- An `ArrayList` implements the `List` interface
- `ArrayLists` can only hold **objects**, and **not primitive** types such as `ints`.

- To declare and instantiate an `ArrayList`:
  - **`ArrayList<String>`** `names = new ArrayList<String>`**`();`**

Some methods for the **`ArrayList`**:

`ArrayList<E>`

```
ArrayList<E> arrayReferenceVariable;
```



ArrayList Declaration- E is the data type

- `int size()`
  `// returns the number of elements in this list`
- `boolean add(x)`
  `// appends x to the end of list; returns true`
- `void add(index, x)`
  `// inserts x at position index, sliding elements`
  `// at position index and higher to the right`
  `// (adds 1 to their indices) and adjusts size`
- `E get(index)`
  `// returns the element at the specified position in this list.`
- `E set(index, x)`
  `// replaces the element at index with x`
  `// returns the element formerly at the specified position`
- `E remove(index)`
  `// removes element from position index, sliding subsequent elements to the left (subtracts 1 from their indices)`
  `// and adjusts size and returns the element at the specified position in this list.`

Some methods for the **`ArrayList`**:
`ArrayList<E>`

- `boolean remove(obj)`

  // removes the first occurrence of `obj` from this list, if it is present. If the list
  // does not contain the element, it is unchanged. Returns **true** if the specified element was
  // removed.

- `int indexOf(obj)`

  // If `obj` is found in the list, then the position number where it is found is returned. If the
  // `obj` is not found, then -1 is returned.

An example (What's printed?)

```
public static void main(String[] args){
     ArrayList <String> names = new ArrayList<String>();
      names.add("Ann");
      names.add("Robert");
      names.add("Tim");
      names.add(1,"Michael");
      names.add(3,"Susan");
      printList(names);
      names.remove(2);
      printList(names);
      names.set(2,"Lenny");
      printList(names);
      System.out.println("The name in position 1 is :" + names.get(1));
   }
```

An example (What's happening?)

```
public static void main(String[] args){
    ArrayList<String> names = new ArrayList<String>();
    names.add("Ann");                    // Ann
    names.add("Robert");                 // Ann  Robert
    names.add("Tim");                    // Ann Robert Tim
    names.add(1,"Michael");              // Ann Michael Robert Tim
    names.add(3,"Susan");                // Ann Michael Robert Susan Tim
    System.out.println(names);           // [Ann,Michael,Robert,Susan,Tim]
    names.remove(2);                     // Ann Michael Susan Tim
    System.out.println (names);          // [Ann,Michael,Susan,Tim]
    names.set(2,"Lenny");                // Ann Michael Lenny Tim
    System.out.println (names);          // [Ann, Michael, Lenny,Tim]
    System.out.println("The name in position 1 is : " + names.get(1));
  }                                      // The name in position 1 is : Michael
```

An example

```
System.out.println("ArrayListTest");
ArrayList<String> aList = new ArrayList<String>();
aList.add("Dan");
aList.add("George");
aList.add("Mary");

System.out.println("aList contains:");
for(int x = 0; x < aList.size(); x++){
  System.out.println(aList.get(x));
}
```

```
System.out.println(aList)
[Dan, George, Mary]
```

```
for(String e : aList){
        System.out.println(e);
    }
```

ArrayListTest

aList contains:

Dan

George

Mary

Another Example:

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

students.add("Mary");
students.add("James");
students.add("Kevin");
students.add(1, "Tanya");
String temp = students.get(3);
System.out.println(temp);
students.remove(2);
students.set(1, "John");
System.out.println(students.size());
```

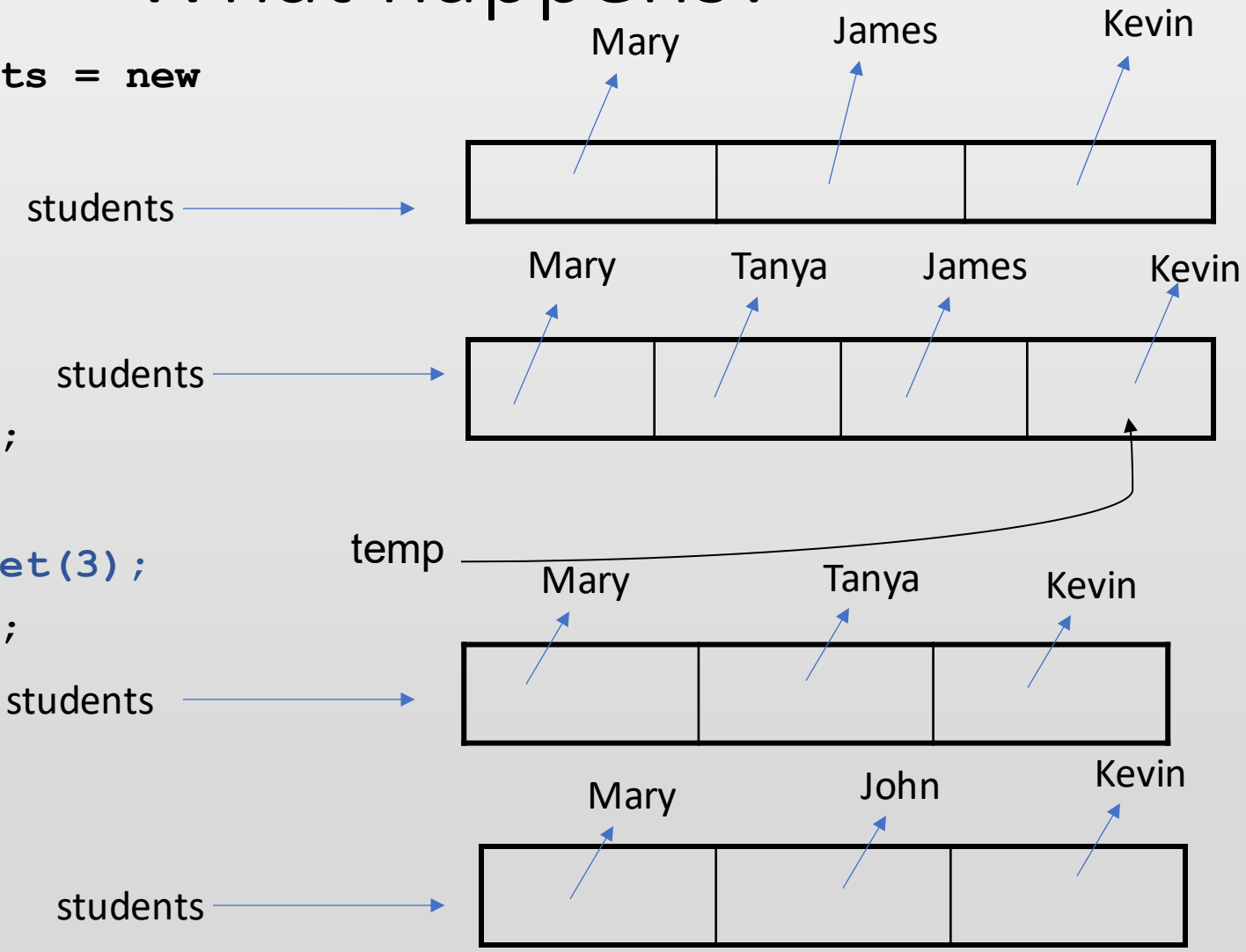
What happens?

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

students.add("Mary");
students.add("James");
students.add("Kevin");
students.add(1, "Tanya");

String temp = students.get(3);
System.out.println(temp);
students.remove(2);

students.set(1, "John");
```



- ArrayLists can hold any kind of object.
  - `ArrayList<Athlete> team = new ArrayList<Athlete>();`
  - `ArrayList<Student> cs111 = new ArrayList<Student>();`
  - `ArrayList<BankAccount> accounts = new ArrayList<BankAccount> ();`

ArrayLists DO NOT hold primitives.

But Java "wraps" primitives! (Don't worry about this yet-112)

  - `ArrayList<Integer> nums = new ArrayList<Integer> ();`

Reminder about number of elements

Data Type	Number of Elements
`Array`	`a.length`
`ArrayList`	`a.size()`
`String`	`a.length()`

Array and Array List Manipulations:

- Access elements
- Traverse arrays
- Find the maximum/minimum value –ARRAYS for primitives
- Accumulate values – ARRAYS for primatives
- Work with consecutive terms
- Add an element
- Remove an element

Student.java StudentTest.java studentNames.txt

- We will modify our test program to use an ArrayList of Student objects instead of an Array of Student objects.
  - **What is the main advantage of an ArrayList in this scenario?**
- Modify your `StudentTester` program (name it `StudentTesterAL.java`)
- Modify the method `findHighest` in your client program that will return the name of the student with the highest gpa. Method `main` will display that information.

Arrays and ArrayLists

The appropriate data structure:

- Arrays are a fixed length.
- ArrayLists grow and shrink as needed.
- Arrays can hold primitive values or objects.
- ArrayLists hold objects.

Arrays and Array Lists

Selecting the appropriate data structure:

	Arrays	`ArrayList`
Can access each element separately	`arr[i]`	`list.get(i)`
Can explicitly overwrite an element in a specific position	`arr[i] = "Tom";`	`list.set(i,"Tom");`
Can inspect the object at a specified location in the sequence	`if arr[i]...`	`if list.get(i)...`

Arrays and ArrayLists

Selecting the appropriate data structure:

	Arrays	`ArrayList`
Can add an object into a specified position of the sequence	Lots of work	`list.add(i,"Joe");`
Can add an object to the end.	Only if there is room and you are keeping track of what and where you are adding.	`list.add("Joe");`
Can remove an object from a specified location	Lots of work	`list.remove(i);`

Array and ArrayList

- Arrays are a fixed length.
- ArrayLists grow and shrink as needed.
- Arrays can hold primitive values or objects.
- ArrayLists hold objects.

	Array	ArrayList	Best case	Worst case
Can access each element separately	`arr[i]`	`list.get(i);`	O(1)	O(1)
Can explicitly overwrite an element in a specific position	`arr[i] = "Tom";`	`list.set(i,"Tom");`	O(1)	O(1)
Can add an object into a specified position in the sequence	`If there is space, copy items i…length-1 to the right, then insert object into position i`	`list.add(i,"Joe");`	O(1)	O(n)
Can add an object to the end	`If there is space, add object at the end (must keep track of last added element)`	`list.add("Joe");`	O(1)	O(1)
Can remove an object from a specified location	`Copy items (i+1)…(length-1) to the left. Remove last element.`	`list.remove(i);`	O(1)	O(n)

Problem 1: *ALProbs.java*

- Write a method, `numWordsOfLength`, that has two parameters, an `ArrayList<String> list`, and an `int len`. The method returns the number of words in `list` that are exactly `len` letters long. For example,
  - If `list = ["cat", "dog", "mouse", "rat", "frog"]`
  - `numWordsOfLength(list, 3) returns 3;`
  - `numWordsOfLength(list, 4) returns 1;`
  - `numWordsOfLength(list, 2) returns 0;`

Problem 2

- Write a method, `removeWordsOfLength`, that has two parameters, an ArrayList `list`, and an `int len`. The method removes all words in `list` that are exactly `len` letters long. For example,
  - If `list = ["cat", "dog", "mouse", "rat", "frog"]`
    - `removeWordsOfLength(list, 3)`
      - `Results with  list = ["mouse", "frog"]`
    - `removeWordsOfLength(list, 4)`
      - `Results with list = ["cat", "dog", "mouse", "rat"]`
    - `removeWordsOfLength(list, 2)`
      - `Results with list = ["cat", "dog", "mouse", "rat", "frog"]`

Summary

- ArrayListTest.java
  - *This program compares the Array data structure to the ArrayList data structure by writing code to do several tasks using each data structure.*