14. ArrayList and HashMap

[ECE20016/ITP20003] Java Programming

Agenda

- Array-based Data Structures
- Collection Framwork and HashMap

Class *ArrayList*

- Consider limitations of Java arrays
  - Array length is not dynamically changeable
    - Possible to create a new, larger array and copy elements – but this is awkward, contrived

- More elegant solution is use an instance of ArrayList
  - Length is changeable at run time

Class *ArrayList*

- Drawbacks of using ArrayList
  - Less efficient than using an array
  - Can only store objects
    - Cannot store primitive types

- Implementation
  - Actually does use arrays
  - Expands capacity in manner previously suggested

Class *ArrayList*

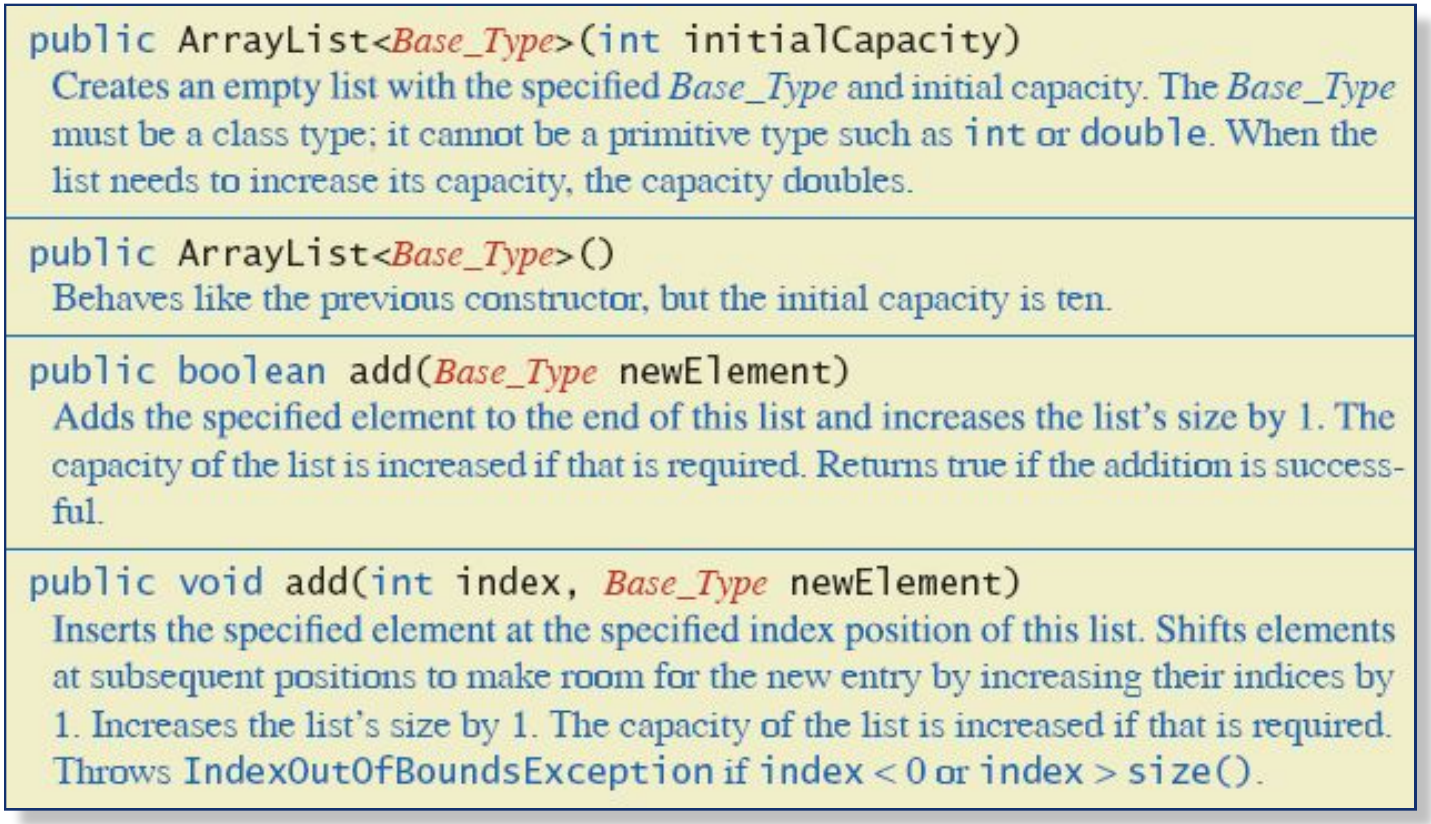
- Class ArrayList is an implementation of an Abstract Data Type (ADT) called a list

- Elements can be added
  - At end, at beginning, or in between items

- Possible to edit, delete, access, and count entries in the list

Class *ArrayList*

- Methods of class ArrayList



public ArrayList<*Base_Type*>(int initialCapacity)

Creates an empty list with the specified *Base_Type* and initial capacity. The *Base_Type* must be a class type; it cannot be a primitive type such as int or double. When the list needs to increase its capacity, the capacity doubles.

public ArrayList<*Base_Type*>()

Behaves like the previous constructor, but the initial capacity is ten.

public boolean add(*Base_Type* newElement)

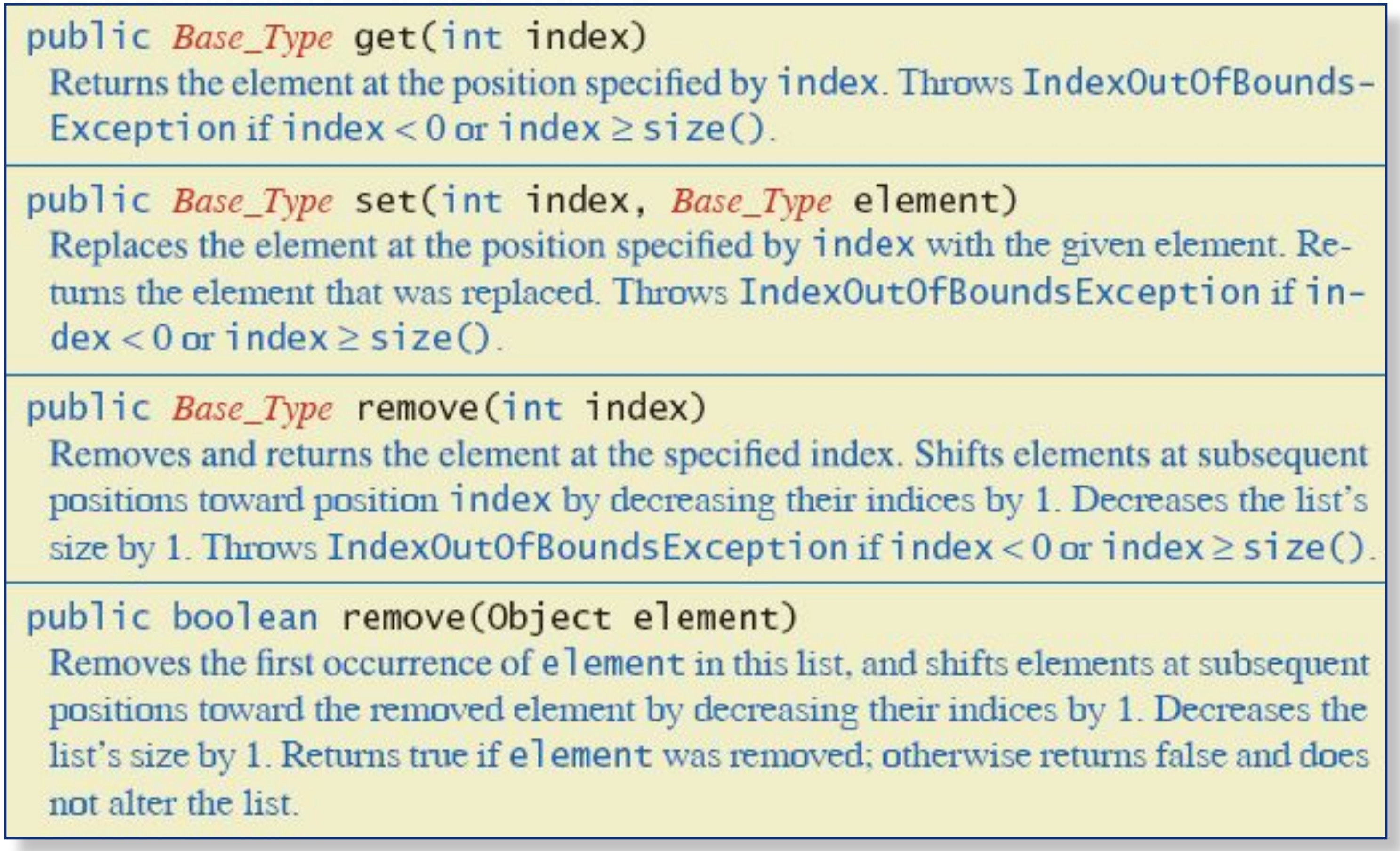
Adds the specified element to the end of this list and increases the list's size by 1. The capacity of the list is increased if that is required. Returns true if the addition is successful.

public void add(int index, *Base_Type* newElement)

Inserts the specified element at the specified index position of this list. Shifts elements at subsequent positions to make room for the new entry by increasing their indices by 1. Increases the list's size by 1. The capacity of the list is increased if that is required. Throws IndexOutOfBoundsException if index < 0 or index > size().

Class *ArrayList*

- Methods of class ArrayList



`public Base_Type get(int index)`

Returns the element at the position specified by `index`. Throws `IndexOutOfBoundsException` if `index` < 0 or `index` ≥ `size()`.

`public Base_Type set(int index, Base_Type element)`

Replaces the element at the position specified by `index` with the given element. Returns the element that was replaced. Throws `IndexOutOfBoundsException` if `index` < 0 or `index` ≥ `size()`.

`public Base_Type remove(int index)`

Removes and returns the element at the specified index. Shifts elements at subsequent positions toward position `index` by decreasing their indices by 1. Decreases the list's size by 1. Throws `IndexOutOfBoundsException` if `index` < 0 or `index` ≥ `size()`.

`public boolean remove(Object element)`

Removes the first occurrence of `element` in this list, and shifts elements at subsequent positions toward the removed element by decreasing their indices by 1. Decreases the list's size by 1. Returns true if `element` was removed; otherwise returns false and does not alter the list.

Creating Instance of ArrayList

- Necessary to import java.util.ArrayList

- Create and name instance

  Ex) ArrayList<String> *list* = new ArrayList<String>();
  ArrayList<String> *list2* = new ArrayList<String>(*20*); // you may set its length but it is not necessary.

- This list will
  - Hold String objects
  - The second list Initially hold up to 20 elements. But its length is decided according to how many realy String instances are in the list.

Using Methods of ArrayList

- Object of an ArrayList used like an array
  - But methods must be used
  - Not square bracket notation

- Given

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

  - Assign a value with

```
aList.add("Hi Mom");
aList.add(index, "Hi Mom");
aList.set(index, "Yo Dad");
```

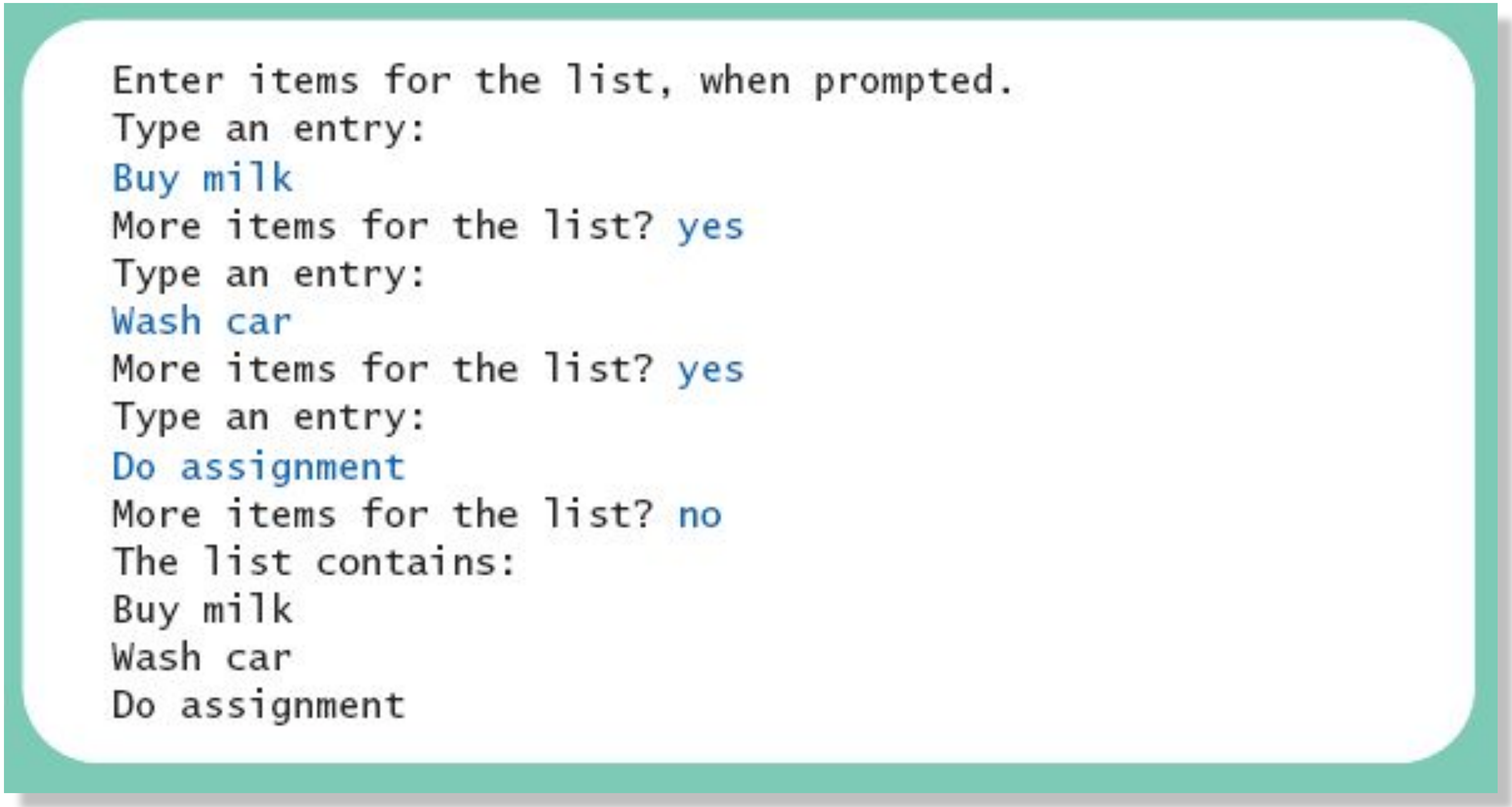
Programming Example

- A To-Do List
  - Maintains a list of everyday tasks
  - User enters as many as desired
  - Program displays the list

- class ArrayListDemonstrator
  - https://github.com/lifove/ArrayListDemonstrator/blob/master/src/main/java/edu/handong/csee/java/example/ArrayListDemonstrator.java

Programming Example

- Result



```
Enter items for the list, when prompted.
Type an entry:
Buy milk
More items for the list? yes
Type an entry:
Wash car
More items for the list? yes
Type an entry:
Do assignment
More items for the list? no
The list contains:
Buy milk
Wash car
Do assignment
```

Programming Example

- class IntegerReader
  - https://github.com/lifove/ArrayListDemonstrator/blob/master/src/main/java/edu/handong/csee/java/example/IntegerReader.java
  - Result

```
Put integer numbers in one line (seperator is space):
12 43 23 56 76 12 67
The number of integer numbers from you: 7
12
43
23
56
76
12
67
```

Programming Example

- When accessing all elements of an ArrayList object
  - Use a for-each loop

```
Ex) for(String s : toDoList)
          System.out.println(s);
```

- Use the trimToSize method to save memory
  - Trims the capacity of this ArrayList instance to be the list's current size.
- To copy an ArrayList
  - Do not use just an assignment statement
    - □ Copies only the reference of the object
  - Use the clone method
    - □ Creates a deep copy
    - □ Example: https://github.com/lifove/ArrayListDemonstrator/blob/master/src/main/java/edu/handong/csee/java/example/DeepCopyTester.java

for-each Statement

- Syntax

  for(*FormalParameter* : *Expression*)

      Statement

  - *Expression* must be Iterable or an array type
  - For more, see http://docs.oracle.com/javase/specs/jls/se7/html/jls-14.html#jls-14.14.2

Parameterized Classes, Generic Data Types

- Possible to declare our own classes which use types as parameters

  Ex) ArrayList<String> toDoList = new ArrayList<String>();

cf. Note earlier versions of Java had a type of ArrayList that was not parameterized

Agenda

- Array-based Data Structures
- **Collection framework and HashMap**

The Java Collections Framework

- A collection of interfaces and classes that implement useful data structures and algorithms
- The Collection interface specifies how objects can be added, removed, or accessed from a Collection
- Brief introduction to HashMap
  - See other references for more info

HashMap Class

- Used like a database to efficiently map from a key to an object
- Uses the same <> notation as an ArrayList to specify the data type of both the key and object

  Ex) HashMap<String, Integer> mountains =
  new HashMap<String, Integer>();
- class HashMapDemo

HashMapDemo

```
public void run()
{
        HashMap<String, Integer> mountains = new HashMap<String, Integer>();
        mountains.put("Everest",29029);
        mountains.put("K2",28251);
        mountains.put("Kangchenjunga",28169);
        mountains.put("Denali",20335);
        printMap(mountains);

        System.out.println("Denali in the map: " + mountains.containsKey("Denali"));
        System.out.println();

        System.out.println("Changing height of Denali.");
        mountains.put("Denali", 20320); // Overwrites old value for Denali
        printMap(mountains);

        System.out.println("Removing Kangchenjunga.");
        mountains.remove("Kangchenjunga");
        printMap(mountains);
}
```

HashMapDemo

```
public void printMap(HashMap<String, Integer> map)
{
    System.out.println("Map contains:");
    for (String keyMountainName : map.keySet())
    {
        Integer height = map.get(keyMountainName);
        System.out.println(keyMountainName +
                              " --> " + height.intValue() + " feet.");
    }
    System.out.println();
}
```

Programming Example

- SalesReporter using HashMap instead of SalesAssociate class