ArrayLists

The **ArrayList<E>** class *implements* the **List<E>** interface. This is a dynamic data structure; its size adjusts as objects are added or removed. The elements must be *object references,* unlike an array that can store object references or primitives. Like an array, all elements must be of the same type. The data type is determined in the declaration, where **<E>** refers to the element type.

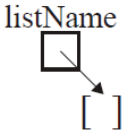
Declaring an ArrayList:

```
ArrayList<String> listName; //sets aside memory to store a reference
-OR-                        //nothing has been constructed
List<Integer> numberListName;
```

Declaring **numberListName** to be of type **List<E>** facilitates code reuse and polymorphism. Even though the `ArrayList` class is the only "list" included in the testable Java subset, there are other implementations.

Instantiating an ArrayList:

```
listName = new ArrayList<String>();    //listName refers to empty list
numberListName = new ArrayList<Integer>(); // reference to empty list
```



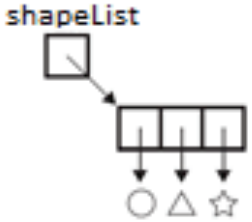
Adding objects to an ArrayList:

Add elements to the end of the list (append) by invoking the one-parameter **add** method:

```
listName.add("George");
numberListName.add(new Integer(6));
```

Objects can be inserted into a given position using the two-parameter version of add. The existing element and all objects in higher positions are moved to the right and the size is adjusted. For example, given the following lines of code, this list would be built:

```
shapeList = new ArrayList<Shape>();
shapeList.add(new Circle());
shapeList.add(new Star());
shapeList.add(1, new Triangle());
```



Accessing ArrayList elements:

To find the number of elements in `listName`, use _____

The first element in an `ArrayList` has a subscript of _____, and the last index is _____

Use an enhanced `for` loop (`for-each`) or the `get` method in a regular indexed `for` loop to access all elements:

Enhanced `for` loop	Indexed `for` loop

`for (int n : nums)`
`{`
`  System.out.println(_____);`
`}`
	`for (int i = 0; i < nums.size(); i++)`
	`{`
	`  System.out.println(_____);`
	`}`

Impossible to go out of bounds	Use when you need the index of element
Uses copies of elements, values can't change	Use when you want to change value of elements

ArrayList methods:

`interface java.util.List<E>`

- `int size()`
- `boolean add(E obj)` `//` appends `obj` to end of list; returns `true`
- `void add(int index, E obj)` `//` inserts `obj` at position `index` (0 ≤ `index` ≤ size),
  `//` moving elements at position `index` and higher
  `//` to the right (adds 1 to their indices) and adjusts size
- `E get(int index)`
- `E set(int index, E obj)` `//` replaces the element at position `index` with `obj`
  `//` returns the element formerly at the specified position
- `E remove(int index)` `//` removes element from position `index`, moving elements
  `//` at position `index + 1` and higher to the left
  `//` (subtracts 1 from their indices) and adjusts size
  `//` returns the element formerly at the specified position

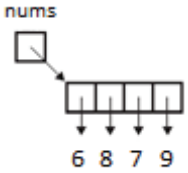
`class java.util.ArrayList<E> implements java.util.List<E>`

Which method replaces the element at a given position? _____

Which method removes the element at a given position? _____

Suppose `nums` contains four Integer objects as shown:

Write code to swap the elements in the middle two positions so the contents will be [6, 7, 8, 9].



ArrayLists as parameters:

Use ***List<E> listName*** in the parameter list. This allows the method to work with an object of any class that implements the **List<*E*>** interface or of one of its subclasses.

```
public double getIncome (List<Double> sales)
{   . . .
}
```

Use an `ArrayList<Double>` name in the call:

```
double revenue = getIncome(dailyReceipts);
```

Remember that the name in the parameter list is an **alias**. Changes made inside a method affect the original data structure. Also, the elements are all references to objects.

Multiple Choice

1. Consider the following code segment:

```
ArrayList<String> items = new ArrayList<String>();
items.add("A");
items.add("B");
items.add("C");
items.add(0, "D");
items.remove(3);
items.add(0, "E");
System.out.println(items);
```

What is printed as a result of executing the code segment?

(A) `[A, B, C, E]`
(B) `[A, B, D, E]`
(C) `[E, D, A, B]`
(D) `[E, D, A, C]`
(E) `[E, D, C, B]`

2. Consider the following instance variable and method.

```
private ArrayList<Integer> list;

public void mystery(int n)
{
    for (int k = 0; k < n; k++)
    {
        Integer num = list.remove(0);
        list.add(num);
    }
}
```

Assume that `list` has been initialized with the following `Integer` objects.
`[12, 9, 7, 8, 4, 3, 6, 11, 1]`

Which of the following represents the list as a result of a call to `mystery(3)`?

(A) `[12, 9, 8, 4, 3, 6, 11, 1, 7]`
(B) `[12, 9, 7, 8, 4, 6, 11, 1, 3]`
(C) `[12, 9, 7, 4, 3, 6, 11, 1, 8]`
(D) `[8, 4, 3, 6, 11, 1, 12, 9, 7]`
(E) `[1, 11, 6, 12, 9, 7, 8, 4, 3]`

3. Consider the following method which is intended to remove all of the odd values in a list.

```
public static void removeOdd(List<Integer> allNums)
{
   for (int i = 0; i < allNums.size(); i++)
   {
      if (allNums.get(i).intValue() % 2 == 1)
      {
         allNums.remove(i);
      }
   }
}
```

Which of the code segments shown below will build a list that makes `removeOdd` appear to work as intended?

I.
```
List<Integer> list = new ArrayList<Integer>();
for (int i = 0; i < 12; i++)
{
   list.add(new Integer(i));
}
```

II.
```
List<Integer> list = new ArrayList<Integer>();
for (int i = 0; i < 12; i += 2)
{
   list.add(new Integer(i));
}
```

III.
```
List<Integer> list = new ArrayList<Integer>();
for (int i = 1; i < 12; i +=2)
{
   list.add(i);
}
```

(A) I only
(B) II only
(C) III only
(D) I and II
(E) II and III

4. Consider the following declaration of the class `NumSequence`, which has a constructor that is intended to initialize the instance variable `seq` to an `ArrayList` of `numberOfValues` random floating-point values in the range [0.0, 1.0).

```
public class numSequence
{
   private List<Double> seq;

   // precondition:     numberOfValues > 0
   // postcondition:    seq has been initialized to an ArrayList of
   //                   numberOfValues elements; each contains a random
   //                   Double value in the range [0.0, 1.0)
   public NumSequence(int numberOfValues)
   {
         /* missing code */
   }
}
```

Which of these code segments could be used to replace ***/* missing code */*** so that the constructor will work as intended?

I.
```
List<Double> seq = new ArrayList<Double>();
for (int i = 0; i < numberOfValues; i++)
{
   seq.add(new Double(Math.random()));
}
```

II.
```
seq = new ArrayList<Double>();
for (int i = 0; i < numberOfValues; i++)
{
   seq.add(new Double(Math.random()));
}
```

III.
```
ArrayList<Double> temp = new ArrayList<Double>();
for(int i = 0; i < numberOfValues; i++)
{
  temp.add(new Double(Math.random()));
}
seq = temp;
```

(A) II only
(B) III only
(C) I and II only
(D) I and III only
(E) II and III only

Free Response

1. An organization raises money by selling boxes of cookies. A cookie order specifies the variety of cookie and the number of boxes ordered. The declaration of the `CookieOrder` class is shown below.

```
public class CookieOrder
{
    /** Constructs a new CookieOrder object. */
    public CookieOrder(String variety, int numBoxes)
    { /* implementation not shown */ }

    /** @return the variety of cookie being ordered
     */
    public String getVariety()
    { /* implementation not shown */ }

    /** @return the number of boxes being ordered
     */
    public int getNumBoxes()
    { /* implementation not shown */ }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

The `MasterOrder` class maintains a list of the cookies to be purchased. The declaration of the `MasterOrder` class is shown below.

```
public class MasterOrder
{
    /** The list of all cookie orders */
    private List<CookieOrder> orders;

    /** Constructs a new MasterOrder object. */
    public MasterOrder()
    { orders = new ArrayList<CookieOrder>(); }

    /** Adds theOrder to the master order.
     * @param theOrder the cookie order to add to the master order
     */
    public void addOrder(CookieOrder theOrder)
    { orders.add(theOrder); }

    /** @return the sum of the number of boxes of all of the cookie orders
     */
    public int getTotalBoxes()
    { /* to be implemented in part (a) */ }

    /** Removes all cookie orders from the master order that have the same variety of
     * cookie as cookieVar and returns the total number of boxes that were removed.
     * @param cookieVar the variety of cookies to remove from the master order
     * @return the total number of boxes of cookieVar in the cookie orders removed
     */
    public int removeVariety(String cookieVar)
    { /* to be implemented in part (b) */ }

    // There may be instance variables, constructors, and methods that are not shown.
}
```

(a) The `getTotalBoxes` method computes and returns the sum of the number of boxes of all cookie orders. If there are no cookie orders in the master order, the method returns 0.

Complete method `getTotalBoxes` below.

```
/** @return the sum of the number of boxes of all of the cookie orders
 */
public int getTotalBoxes()
```

(b) The `removeVariety` method updates the master order by removing all of the cookie orders in which the variety of cookie matches the parameter `cookieVar`. The master order may contain zero or more cookie orders with the same variety as `cookieVar`. The method returns the total number of boxes removed from the master order.

For example, consider the following code segment.

```
MasterOrder goodies = new MasterOrder();
goodies.addOrder(new CookieOrder("Chocolate Chip", 1));
goodies.addOrder(new CookieOrder("Shortbread", 5));
goodies.addOrder(new CookieOrder("Macaroon", 2));
goodies.addOrder(new CookieOrder("Chocolate Chip", 3));
```

After the code segment has executed, the contents of the master order are as shown in the following table.

"Chocolate Chip"	"Shortbread"	"Macaroon"	"Chocolate Chip"
1	5	2	3

The method call `goodies.removeVariety("Chocolate Chip")` returns 4 because there were two Chocolate Chip cookie orders totaling 4 boxes. The master order is modified as shown below.

"Shortbread"	"Macaroon"
5	2

The method call `goodies.removeVariety("Brownie")` returns 0 and does not change the master order.

Complete method `removeVariety` below.

```
/** Removes all cookie orders from the master order that have the same variety of
 *  cookie as cookieVar and returns the total number of boxes that were removed.
 *  @param cookieVar  the variety of cookies to remove from the master order
 *  @return  the total number of boxes of cookieVar in the cookie orders removed
 */
public int removeVariety(String cookieVar)
```

2. A music web site keeps track of downloaded music. For each download, the sites uses a `DownloadInfo` object to store a song's title and the number of times it has been downloaded. A partial declaration for the `DownloadInfo` is class is shown below.

```
public class DownloadInfo
{
   /** Creates a new instance with the given unique title and sets the
    *    number of times downloaded to 1.
    *   @param title the unique title of the downloaded song
    */
   public DownloadInfo(String title)
   { /* implementation not shown */ }

   /** @return the title */
   public String getTitle()
   { /* implementation not shown */ }

   /** Increment the number times downloaded by 1 */
   public void incrementTimesDownloaded()
   { /* implementation not shown */ }

   // There may be instance variables, constructors, and methods that are not shown.
}
```

The list of downloaded information is stored in a `MusicDownloads` object. A partial declaration for the `MusicDowloads` class is shown below.

```
public class MusicDownloads
{
    /** The list of downloaded information
     *  Guaranteed not to be null and not to contain duplicate titles.
     */
    private List<DownloadInfo> downloadList;

    /** Creates the list of downloaded information */
    public MusicDownloads()
    { downloadList = new ArrayList<DownloadInfo>(); }

    /** Returns a reference to the DownloadInfo object with the requested title if it exists.
     *  @param title the requested title
     *  @return a reference to the DownloadInfo object with the title that matches the
     *          parameter title if it exists in the list; null otherwise.
     *  Postcondition: no changes were made to downloadList.
     */
    public DownloadInfo getDownloadInfo(String title)
    { /* to be implemented in part (a) */ }

    /** Updates downloadList with information from titles.
     *  @param titles a list of song titles
     *  Postcondition:
     *          - there are no duplicate titles in downloadList.
     *          - no entries were removed from downloadList.
     *          - all songs in titles are represented in downloadList.
     *          - for each existing entry in downloadList, the download count is
     *            increased by the number of times its title appeared in titles
     *          - the order of the existing entries in downloadList is not changed.
     *          - the first time an object with a title from titles is added to downloadList,
     *            it is added to the end of the list.
     *          - new entries in downloadList appear in the same order
     *            in which they appear in titles
     *          - for each new entry in downloadList, the download count is equal to
     *            the number of times its title appeared in titles.
     */
    public void updateDownloads(List<String> titles)
    { /* to be implemented in part (b) */ }

    // There may be instance variable, constructors, and methods that are not shown.
}
```

(a) Write the `MusicDownloads` method `getDownloadInfo`, which returns a reference to a `DownloadInfo` object if an object with a title that matches the parameter `title` exist in the `downloadList`. If no song in `downloadList` has a title that matches the parameter `title`, the method returns `null`.

For example, suppose variable `webMusicA` refers to an instance of `MusicDowloads` and that the table below represents the contents of `downloadList`. The list contains three `DownloadInfo` objects. The object at position 0 has a title of "Hey Jude" and a download count of 5. The object at position 1 has a title of "Soul Sister" and a download count of 3. The object at position 2 has a title of "Aqualung" and a download count of 10.

0	1	2
"Hey Jude" 5	"Soul Sister" 3	"Aqualung" 10

The call `webMusicA.getDownloadInfo("Aqualung")` returns a reference to the object in position 2 of the list.

The call `webMusica.getDownloadInfo("Happy Birthday")` returns `null` because there are no `DownloadInfo` objects with that title in the list.

Class information repeated from the beginning of the question

```
public class DownloadInfo

public DownloadInfo(String title)
public String getTitle()
public void incrementTimesDownloaded()

public class MusicDownloads


private List<DownloadInfo> downloadList
public DownloadInfo getDownloadInfo(String title)
public void updateDownloads(List<String> titles)
```

WRITE YOUR ANSWER ON THE NEXT PAGE

Complete method `getDownloadInfo` below.

```
/** Returns a reference to the DownloadInfo object with the requested title if it exists.
 *  @param title the requested title
 *  @return a reference to the DownloadInfo object with the
 *          title that matches the parameter title if it exists in the list;
 *          null otherwise.
 *  Postcondition: no changes were made to downloadList.
 */
public DownloadInfo getDownloadInfo(String title)
```

(b) Write the `MusicDownloads` method `updateDownloads`, which takes a list of song titles as a parameter. For each title in the list, the method updates `downloadList`, either by incrementing the download count if a `DownloadInfo` object with the same title exists, or by adding a new `DownloadInfo` object with that title and a download count of 1 to the end of the list. When a new `DownloadInfo` object is added to the end of the list, the order of the already existing entries in `downloadList` remains unchanged.

For example, suppose variable `webMusicB` refers to an instance of `MusicDownloads` and that the table below represents the content of the instance variable `downloadList`.

0	1	2
"Hey Jude" 5	"Soul Sister" 3	"Aqualung" 10

Assume that the variable `List<String> songTitles` has been defined and contains the following entries.

```
{"Lights", "Aqualung", "Soul Sister", "Go Now", "Lights", "Soul Sister"}
```

The call `webMusicB.updateDownloads(songTitles)` results in the following `downloadList` with incremented download counts for the objects with the titles of "Soul Sister" and "Aqualung". It also has a new `DownloadInfo` objects with a title of "Lights" and a download count of 2, and another `DownloadInfo` object with a title of "Go Now" and a download count of 1. The order of the already existing entries remains unchanged.

0	1	2	3	4
"Hey Jude" 5	"Soul Sister" 5	"Aqualung" 11	"Lights" 2	"Go Now" 1

Class information repeated from the beginning of the question

```
public class DownloadInfo

public DownloadInfo(String title)
public String getTitle()
public void incrementTimesDownloaded()

public class MusicDownloads

private List<DownloadInfo> downloadList
public DownloadInfo getDownloadInfo(String title)
public void updateDownloads(List<String> titles)
```

In writing your solution, you must use the `getDownloadInfo` method. Assume that `getDownloadInfo` works as specified, regardless of what you wrote for part (a).

Complete method `updateDownloads` below.

```
/** Updates downloadList with information from titles.
 *  @param titles a list of song titles
 *  Postcondition:
 *      - there are no duplicate titles in downloadList.
 *      - no entries were removed from downloadList.
 *      - all songs in titles are represented in downloadList.
 *      - for each existing entry in downloadList, the download count is
 *        increased by the number of times its title appeared in titles
 *      - the order of the existing entries in downloadList is not changed.
 *      - the first time an object with a title from titles is added to downloadList,
 *        it is added to the end of the list.
 *      - new entries in downloadList appear in the same order
 *        in which they appear in titles
 *      - for each new entry in downloadList, the download count is equal to
 *        the number of times its title appeared in titles.
 */
public void updateDownloads(List<String> titles)
```