Arrays And ArrayLists

"Should array indices start at 0 or 1? My compromise of 0.5 was rejected without, I thought, proper consideration. "
- S. Kelly-Bootle

Arrays in Java

- Java has built in arrays as well as more complicated classes to automate many array tasks (the ArrayList class).
- Arrays hold elements of the same type:
  - primitive data types or classes
  - space for array must be dynamically allocated with new operator.
    (Size is any integer expression. Due to dynamic allocation does not have to be a constant.)

```java
public void arrayExamples()
{
    int[] intList = new int[10];
    for(int i = 0; i < intList.length; i++)
    {
        assert 0 >= i && i < intList.length;
        intList[i] = i * i * i;
    }
}
```

Array Details

- All arrays must be dynamically allocated.
- Arrays have a public, final field called `length`.
  - Built in size field, no separate variable needed.
  - Don't confuse length (capacity) with elements in use.
- Elements start with an index of zero, last index is `length - 1`.
- Trying to access a non-existent element results in an `ArrayIndexOutOfBoundsException` (AIOBE).

Array Initialization

- Array variables are object variables.
- They hold the memory address of an array object.
- The array must be dynamically allocated.
- All values in the array are initialized (0, 0.0, char 0, false, or null).
- Arrays of primitives and Strings may be initialized with an initializer list:
  ```java
  int[] intList = {2, 3, 5, 7, 11, 13};
  double[] dList = {12.12, 0.12, 45.3};
  String[] sList = {"Olivia", "Kelly", "Isabelle"};
  ```
Arrays of objects
- A native array of objects is actually a native array of *object variables*
  - all object variables in Java are really what?
  - Pointers!

```java
public void objectArrayExamples()
{
    Rectangle[] rectList = new Rectangle[10];
    // How many Rectangle objects exist?
    rectList[5].setSize(5,10);
    // uh oh!
    for(int i = 0; i < rectList.length; i++)
    {
        rectList[i] = new Rectangle();
    }
    rectList[3].setSize(100,200);
}
```

Array Utilities
- In the *Arrays* class
- `binarySearch`, `equals`, `fill`, and `sort` methods for arrays of all primitive types (except boolean) and arrays of Objects
  - overloaded versions of these methods for various data types
- In the System class there is an `arraycopy` method to copy elements from a specified part of one array to another
  - can be used for arrays of primitives or arrays of objects

The ArrayList Class
- A class that is part of the Java Standard Library and a class that is part of the AP subset
- a kind of automated array
- not all methods are part of the ap subset

About Lists (in general)
- A list is an ordered collection or a *sequence*.
- ArrayList implements the List interface
- The user of this interface will have control over where in the list each element is inserted.
- The user can access elements by their integer index (position in the list), and search for elements in the list.
- Items can be added, removed, and accessed from the list
Methods

- `ArrayList()` //constructor
- `void add(int index, Object x)`
- `boolean add(Object x)`
- `Object set(int index, Object x)`
- `Object remove(int index)`
- `int size()`
- `Object get(int index)`
- `Iterator iterator()`

How the methods work

- **add:**
  - `boolean add(Object x)` – **inserts** the Object `x` at the end of the list (size increases by 1), returns true
  - `void add(int index, Object x)` – **inserts** the Object `x` at the given index position (elements will be shifted to make room and size increases by 1)

- **get:**
  - returns the Object at the specified index
  - should cast when using value returned
  - throws `IndexOutOfBoundsException` if index<0 or index>=size

- **set**
  - `replaces` value of Object parameter at the given index
  - size is not changed
How the methods work

- **remove**
  - *removes* the element at the specified index
  - throws IndexOutOfBoundsException if index<0 or index>=size
  - size will be decreased by 1
  - returns Object removed

Examples

```java
ArrayList club = new ArrayList();
club.add("Spanky");
club.add("Darla");
club.add("Buckwheat");
System.out.print(club);
```
Displays:

```
[Spanky, Darla, Buckwheat]
```

//using club from previous slide
```java
club.set(1, "Mikey");
System.out.print(club);
```
Displays:

```
[Spanky, Mikey, Buckwheat]
```

//using club from previous slide
```java
club.add(0, 
    club.remove(club.size()-1));
System.out.print(club);
```
Displays:

```
[Buckwheat, Spanky, Mikey]
```
//ArrayLists only contain Objects!!
ArrayList odds = new ArrayList();
for (int i = 1; i < 10; i += 2)
    odds.add(new Integer(i));
System.out.println(odds);

Displays:
[1, 3, 5, 7, 9]

//ArrayLists only contain Objects!!
ArrayList odds = new ArrayList();
for (int i = 1; i < 10; i += 2)
    
    { Integer x = new Integer(i);
      odds.add(x); }
System.out.println(odds);

Displays:
[1, 3, 5, 7, 9]

Objects and Casting

//Casting when pulling out from ArrayList
ArrayList names = new ArrayList();
names.add("Clint");
names.add("John");
names.add("Robert");
names.add("Henry");
Object obj = names.get(2); //ok
System.out.println(obj.toString());
String str1 = names.get(3); //syntax error
String str2 = (String)(names.get(4)); //ok
char c =
    ((String)(names.get(0))).charAt(0);
//Gack!!

How the methods work

- iterator
  - returns an Iterator object
  - Iterators allow all of the Objects in the list to be accessed one by one, in order
  - methods for an Iterator object
    - hasNext
    - next
    - remove
public boolean hasNext()

- Returns true if the iteration has more elements
- Ex:
  ```java
  while (it.hasNext())
      // do something
  ```

public Object next()

- Returns the next element in the iteration
- Each time this method is called the iterator “moves”
- Ex:
  ```java
  while (it.hasNext())
      Object obj = it.next();
      if ( // obj meets some condition)
          // do something
  ```

public void remove()

- Removes from the collection the last element returned by the iterator
- Can be called only once per call to next

while (it.hasNext())

```java
{ 
    Object obj = it.next();
    if ( // obj meets some condition)
        it.remove();
}
```

Remove Example

```java
public void removeAllLength(ArrayList li, int len) {
    // pre: li contains only String objects
    // post: all Strings of length = len removed
    // wrong way
    String temp;
    for (int i = 0; i < li.size(); i++)
    {
        temp = (String) li.get(i);
        if ( temp.length() == len )
            li.remove(i);
    }
}
```

What if the list contains ["hi", "ok", "the", "so", "do"] and len = 2?
public void removeAllLength(ArrayList li, int len) {
    // pre: li contains only String objects
    // post: all Strings of length = len removed
    // right way using iterator
    String temp;
    iterator it = li.iterator();
    while( it.hasNext() ) {
        temp = (String)li.next();
        if ( temp.length() == len )
            it.remove();
    }
}

What if the list contains ["hi", "ok", "the", "so", "do"] and len = 2?