Chapter 9
Arrays & ArrayLists

One of the things you've probably realized is that so far we are very limited in the number of objects we can work with in a game. To illustrate the problem, assume we've got four sheep on screen that we're trying to herd into the corral with a broom. We probably need some code that looks like this.

```java
if (collides(sheep1, broom) == true)
    { corral(sheep1); }
if (collides(sheep2, broom) == true)
    { corral(sheep2); }
if (collides(sheep3, broom) == true)
    { corral(sheep3); }
if (collides(sheep3, broom) == true)
    { corral(sheep3); }
```

This works well if we're only corraling four sheep but would become unwieldy if we had 10 sheep, and beyond ridiculous if we have 50 sheep. There has to be a better way to manage large amounts of data, and in fact there are several.

In this chapter we'll look at arrays, a traditional programming tool to collect a large amount of data into one object that we can process. We'll use them to efficiently create games with more elements. We'll also look at ArrayLists, a Java object that provides a more powerful tool to handle a lot of data. For the Free Java Book you can study arrays or arrays and ArrayLists.

For the Free Java Book you can cover either arrays alone or arrays and ArrayLists to continue your work. You don't have to study both.

### 9.1 Array Basics

In Java an array is an object designed to collect and manage large numbers of any kind of data. Arrays are standard Java, but the notation is quite different from what we've seen before, so things will look a little strange for awhile.

Here's a conceptual visualization of an array, let's call it `words`, that holds the words in the sentence 'Java is a powerful language'.
An array has the following characteristics

- it holds elements of a single primitive data type or object type. An array can’t hold both doubles and booleans. Similarly an array can’t hold both GRects and GOvals, unless it is an array of GObjects, as GRects and GOvals are both GObjects.
- it is a linear list, which means that, conceptually speaking, the stored objects line up one after another and there is a beginning and end of the list
- its positions are indexed (numbered) beginning at 0. There are 5 words in the example sentence so the last one is at position 4. In general if the array contains $N$ elements, then the last position in the array has index $N-1$
- the size is fixed at the time the program is compiled and cannot be changed during a program’s run
- the first position of the array is called the front or head, and the last position is called the rear or tail
- its data can be accessed in any order—head to tail, tail to head or randomly

### 9.1.1 Creating an Array

An array suitable for storing for the sentence above, word by word, can be created with these statements.

```java
final int MAX_WORD = 5;
String[] words = new String[MAX_WORD];
```

and in general arrays are declared and instantiated with

```java
dataType/objectType[ ] arrayName = new dataType/objectType[sizeConstant];
```

Here’s an example array of ints with a size constant.

```java
final int MAX_DATA = 1000;
int[ ] dataValues = new int[MAX_DATA];
```

and here’s an example array of UFOs, assuming UFO is an already created object.

```java
final int MAX_UFO = 10;
UFO[ ] ufos = new UFO[MAX_UFO];
```

The size of an array is almost always declared with a constant as shown above. The constant is very convenient for further array processing.
An array may also be declared and instantiated in separate statements.

```java
String[] words;
words = new String[MAX_WORD];
```

The values in an array are automatically initialized. Numeric values are initialized to 0, `char` values are initialized to the `char` that has ASCII value 0, `booleans` to `false` and `objects` to `null`.

### 9.1.2 Assigning an Element to an Array

Values are assigned to an array by assignment to individual positions in the array. Below are statements that would load `words` as created above.

```java
words[0] = "Java";
words[1] = "is";
words[2] = "a";
words[3] = "powerful";
words[4] = "language";
```

If we wanted to assign a value read from the keyboard value we might have

```java
words[MAX_WORD-1] = readln("Last word? ");
```

Assigning a value to an array position replaces the existing element at that position.

There is no built in method for inserting an element into an array and moving the array elements to make room, or for deleting an element and moving the elements to take up the deleted element’s position.

**The Index Out of Bounds Problem**

The `words` array above has valid positions 0, 1, ..., 4. There is no valid position before position 0 or after position 4. Any attempt to reference an invalid position such as

```java
words[-1] = "We all know";
```

or

```java
words[MAX_WORD] = "Java is fun to learn";
```

results in a program crash and the error message: `java.lang.ArrayIndexOutOfBoundsException` which indicates that an invalid position was specified. Some languages, in particular C++, do not catch a reference to an out of bounds index. It’s still an error, but the program may not crash and the program may run very long time before the error shows up—not good!
9.1.3 Traversing an Array

An array may be traversed, element by element, in order, using a loop.

```java
for(int i=0; i<MAX_WORD; i++)
{
    println(words[i]);
}
```

An array may also be traversed in reverse order.

```java
for(int i=MAX_WORD-1; i >= 0; i--)
{
    println(words[i]);
}
```

`words` may also be traversed forwards (only) using an enhanced `for` loop as shown here.

```java
for(String w : words)
{
    println(w);
}
```

This enhanced `for` loop may be understood as these steps

```java
for every item in words
{
    assign the item to w
    println w
}
```

In general an enhanced `for` loop is written

```java
for(dataType/objectType localName : arrayName)
{
    statements
}
```

Note that `localName` is declared inside the `for` loop and so has scope limited to the `for` loop.

Traversing an array in the head to tail direction is a forward traversal, and the reverse is a backward traversal.

9.1.4 The Length Value

The number of spaces allocated to an array is stored in the public `length` value created when the array object was created. This value can also be used for array traversal.

```java
println(words.length);
```

```java
for(int i=0; i<words.length; i++)
{
    println(words[i]);
}
```
We'll learn more about public values of objects in the next chapter.

### 9.1.5 Arrays and Methods

The examples below show how arrays are passed to and returned from methods. Remember that an array is a object so that a pass to a method is a pass by reference. Changes to the array inside the method are retained outside the method.

```java
public void display(String[] w) {
    for(String wVal : w) {
        println(wVal);
    }
}

public String[] reverse(String[] w) {
    String[] temp = new String[w.length];
    for(int i = w.length - 1; i >= 0; i--)
    {
        temp[w.length - i - 1] = w[i];
    }
    return temp;
}

public void emptyStrings(String[] w) {
    for(int i = 0; i < w.length; i++)
    {
        w[i] = "";
    }
}
```

### 9.1.6 An Example Program

This program illustrates most of the above techniques.

```java
import acm.program.*;

public class ArrayBasics extends ConsoleProgram {
    final int MAX = 5;
    public void run() {
        // pass array to methods
        display(words);
        display(reverse(words));
        emptyStrings(words);
    }
}
```

---

©Daniel L. Schuster, 2009, all rights reserved
```java
String[ ] words = new String[MAX];
display(words);

println("\n");
println("The length of words is: " + words.length);

println("\n");
words[0] = "Java";
words[1] = "is";
words[2] = "a";
words[3] = "powerful";
words[4] = "language";
display(words);

println("\n");
println("Programming");
display(words);

println("\n");
String[ ] revWords = reverse(words);
display(revWords);

println("\n");
println("Now crashing the program...");
println("Going for a walk off the end of the array!");
}
} //run

public void display(String[ ] w)
{
    for(String wVal : w)
    {
        println(wVal + " ");
    }
} //display

public String[ ] reverse(String[ ] w)
{
    String[ ] temp = new String[w.length];
    for(int i=w.length-1; i>=0; i--)
    {
        temp[w.length-i-1] = w[i];
    }
    return temp;
} //reverse
```

And here's a sample run.

null null null null null
The length of words is: 5
Java is a powerful language
Java is a programming language
language programming a is Java

Now crashing the program...

This message appears in the command window when the program crashes

```java
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException
    at ArrayBasics.run(ArrayBasics.java:35)
...several more lines of error information...
```

9.2 Using Arrays in a Game

Arrays give us the capability of working with large number of objects. Let's look at an example game we developed earlier and consider the changes needed when we increase the quantity of objects in the game.

9.2.1 Modifying Cannonade2

Recall the Cannonade2 game where several cannonballs were fired and we attempted to block those shots, much like a game of ping pong. Here's some of the code from Cannonade2.

```java
time = time + WAIT;
cb1.move(xMove1, yMove1);
cb2.move(xMove2, yMove2);
cb3.move(xMove3, yMove3);

cb1Box = cb1.getBounds();
cb2Box = cb2.getBounds();
cb3Box = cb3.getBounds();
blockerBox = blocker.getBounds();

//check for blocking
if (cb1Box.intersects(blockerBox) == true)
{
    yMove1 = -yMove1;
xMove1 = rg.nextInt(-1, 1);
    score++;
}
if (cb2Box.intersects(blockerBox) == true)
{
    yMove2 = -yMove2;
xMove2 = rg.nextInt(-1, 1);
}
```

©Daniel L. Schuster, 2009, all rights reserved
score = score + 2;
}
if (cb3Box.intersects(blockerBox) == true)
{
    yMove3 = -yMove3;
    xMove3 = rg.nextInt(-1, 1);
    score = score + 3;
}

While using the exact same *Cannonball* and *Blocker* classes as before, let's write a version that works with any reasonable number of cannonballs. Some examples of the needed modifications are discussed here.

Instead of declaring each of the cannonballs individually we declare an array of cannonballs and then initialize each individually.

```
Cannonball[] cbs;
cbs = new Cannonball[CB_COUNT];
cbs[0] = new Cannonball();
cbs[1] = new Cannonball(Color.GREEN);
cbs[2] = new Cannonball(Color.YELLOW);
cbs[3] = new Cannonball(Color.RED);
```

We use a *RandomGenerator* `rg` to place all of the *Cannonballs* in the window, which has width `AW`.

```
for(Cannonball c : cbs)
{
    add(c, rg.nextInt(0, AW-50), 0);
}
```

There are unique x and y movements for each cannonball, so we record the x movements in one array and the y movements in another.

```
int[ ] xMoves = new int[CB_COUNT];
int[ ] yMoves = new int[CB_COUNT];
```

```
for(int i=0; i<CB_COUNT; i++)
{
    xMoves[i] = 0; //initially 0, later set randomly to 1, 0 or -1
    yMoves[i] = rg.nextInt(MIN_Y VELOCITY, MAX_Y VELOCITY);
}
```

Missed cannonballs fall out of the window and shouldn't be considered part of the game any longer, so we create an array to record the in-game/out-of-game state of the cannonballs. This array is automatically initialized to `false` which is just what we want because initially none of the
cannonballs are out of the game. When a cannonball falls out of the window we’ll set the appropriate position in `cbsOutOfGame` to `true`. We have

```java
boolean[ ] cbsOutOfGame = new boolean[CB_COUNT];
```

As the game progresses we attempt to block the dropping cannonballs. For every cannonball still in the game, we check it for being blocked and bounce the cannonball if it is.

```java
for(int i=0; i<CB_COUNT; i++)
{
    if ((cbsOutOfGame[i] == false) && (cbBoxes[i].intersects(blockerBox) == true))
    {
        yMoves[i] = -yMoves[i];
        xMoves[i] = rg.nextInt(-1, 1); //left, center or right x movement
        score = score + 1;
    }
}
```

Our game will end when 10 seconds of play have elapsed or all of the cannonballs have escaped the window. The `allDone boolean` method below checks the 'out of game' status of each `Cannonball`. If a `Cannonball` isn’t finished then the loop is exited and the method returns `false`, indicating that not all of the `Cannonballs` are out of the game. If the array traversal loop finishes, then all of the `Cannonballs` are out of the game and so `true` is returned.

```java
public boolean allDone(boolean[ ] cbd)
{
    for(boolean done : cbd)
    {
        if (done == false) return false;
    }
    return true;
}
```

### 9.2.2 CannonadeWithArrays
Below is the full program that implements a version of Cannonade2 with four `Cannonballs`, but it could be extended to many more with very little effort.

```java
//CannonadeWithArrays.java
import acm.program.*;
import acm.graphics.*;
import acm.util.*;
import java.awt.*;
import java.awt.event.*;
public class CannonadeWithArrays extends GraphicsProgram
```
public static final int APPLICATION_WIDTH = 500;
public static final int APPLICATION_HEIGHT = 600;

final int AW = APPLICATION_WIDTH;
final int AH = APPLICATION_HEIGHT;
final int WAIT = 8;
final int MAX_TIME = 10000;
final int CB_SIZE = 30, CB_COUNT = 4;
final int MIN_Y_VELOCITY = 3, MAX_Y_VELOCITY = 5;

Cannonball[ ] cbs;
Blocker blocker;
int time, score;
GLabel scoreLbl;

class Cannonball {
    int x, y, velocity, type;
}

public void init( )
{
    setBackground(Color.BLUE);

time = 0;
score = 0;
scoreLbl = new GLabel("" + score);
scoreLbl.setFont("SansSerif-BOLD-30");
scoreLbl.setColor(Color.RED);
add(scoreLbl, 2, 26);

cbs = new Cannonball[CB_COUNT];
cbs[0] = new Cannonball();
cbs[1] = new Cannonball(Color.GREEN);
cbs[2] = new Cannonball(Color.YELLOW);
cbs[3] = new Cannonball(Color.RED);

RandomGenerator rg = new RandomGenerator();

for(Cannonball c : cbs)
{
    add(c, rg.nextInt(0, AW-50), 0);
}

blocker = new Blocker();
add(blocker, AW/2, AH/2);

addMouseListener( );
waitForClick( );
} //init

public void mouseMoved(MouseEvent e)
double \( x = e.\text{getX}(\ ); \)
\( y = e.\text{getY}(\ ); \)
\( \text{blocker.\text{setLocation}(x-40, y);} //\text{center blocker under mouse} \)

```java
} //mouseMoved

public void run( )
{
    RandomGenerator \( rg = \text{new RandomGenerator( }\); \)
    GRectangle[ ] \( cbBoxes = \text{new GRectangle[CB_COUNT];} \)
    GRectangle \( \text{blockerBox;} \)

    boolean[ ] \( \text{cbsOutOfGame = new boolean[CB_COUNT];} \)

    int[ ] \( xMoves = \text{new int[CB_COUNT];} \)
    int[ ] \( yMoves = \text{new int[CB_COUNT];} \)

    for(int \( i=0; i<\text{CB_COUNT}; i++ \))
        { \( xMoves[i] = 0; //\text{initially 0, later set randomly to 1, 0 or -1} \\
            yMoves[i] = rg.\text{nextInt}(MIN_Y_VELOCITY, MAX_Y_VELOCITY); \)
        } //initialize x and y moves

    while(time < MAX_TIME) //game loop
    {
        pause(WAIT); \( time = time + \text{WAIT;} \)

        for(int \( i=0; i<\text{CB_COUNT}; i++ \))
            { \( \text{cb[i].move(xMoves[i], yMoves[i]);} \) \( cbBoxes[i] = \text{cb[i].getBounds()} ; \)
            } \( \text{blockerBox = \text{blocker.getBound} \text{s()} ;} \)

        for(int \( i=0; i<\text{CB_COUNT}; i++ \))
            { \( \text{if } ((\text{cbsOutOfGame[i]} == \text{false}) \) \) \( \&\& (\text{cbBoxes[i].intersects(blockerBox) == true}) \)
                { \( \text{yMoves[i] = -yMoves[i];} \) \( xMoves[i] = rg.\text{nextInt(-1, 1);} /\text{left, center or right x movement} \)
            } //center blocker under mouse
```

Bounding rectangles of the cannonballs

Track cannonballs that exit the game window

Track x and y movement values of the cannonballs

Initialize x and y movement values

Move the cannonballs and get the bounding rectangles

Check each cannonball - if it's not out of the game window and it was blocked then change the movement so that the cannonball bounces and adjust the score
```java
score = score + 1;

for(int i=0; i<CB_COUNT; i++)
{
    if (cbsOutOfGame[i] == false) && atTop(cbs[i]) == true)
        yMoves[i] = -yMoves[i];
}

for(int i=0; i<CB_COUNT; i++)
{
    if ((cbsOutOfGame[i] == false) && (atLeft(cbs[i]) == true) || (atRight(cbs[i]) == true))
        xMoves[i] = -xMoves[i];
}

for(int i=0; i<CB_COUNT; i++)
{
    if (cbsOutOfGame[i] == false) && (atBottom(cbs[i]) == true))
    {
        remove(cbs[i]);
        cbsOutOfGame[i] = true;
        score--;
    }
}

scoreLbl.setText("+score");
if (allDone(cbsOutOfGame) == true)
    break;
}

//end the game
GLabel doneLbl = new GLabel("Game Over!");
doneLbl.setFont("SansSerif-BOLD-50");
doneLbl.setColor(Color.RED);
add(doneLbl, 2, AH-35);
}

public boolean atTop(Cannonball b) {
    if (b.getY() <= 0) return true;
}
```
null
else return false;
} //atTop

public boolean atRight(Cannonball cb)
{
    if (cb.getX() >= AW) return true;
    else return false;
} //atRight

public boolean atBottom(Cannonball cb)
{
    if (cb.getY() + CB_SIZE >= AH) return true;
    else return false;
} //atBottom

public boolean atLeft(Cannonball cb)
{
    if (cb.getX() <= 0) return true;
    else return false;
} //atLeft

public boolean allDone(boolean[] cbDone)
{
    for(boolean b : cbDone)
    {
        if (b == false) return false;
    }
    return true;
} //alldone
} //CannonadeWithArrays

We'll have a lot more to say about arrays in a later chapter.

9.3 ArrayList Basics

The ArrayList is a powerful standard Java object designed to collect and manage large numbers of any kind of object. In many ways it’s really an array with a bunch of useful methods.

Here's a visual representation of an ArrayList, let's call it words, that holds the words in the sentence 'Java is a powerful language'.

\[
\begin{array}{|c|c|c|c|c|}
\hline
0 & 1 & 2 & 3 & 4 \\
Java & is & a & powerful & language \\
\hline
\end{array}
\]

An ArrayList has the following characteristics
it is a linear list, which means that, conceptually speaking, the stored objects line up one after another and there is a beginning and end of the list

- its positions are indexed (numbered) beginning at 0. There are 5 words in the example sentence so the last one is at position 4
- it contains objects (String, GRect, UFO, etc), *not* primitive data types. Note that this restriction can be worked around by using Java wrapper classes, but we won't study them in this course
- its size will grow automatically as needed
- its data can be added, removed, changed or retrieved at any time and in any order
- the first position of the ArrayList is called the front or head, and the last position is called the rear or tail

An ArrayList is a generic, which means that it is an object or algorithm that is designed to work with many different kinds of objects. You can have an ArrayList of Strings, UFOs, GLabels or any kind of object. You'll learn much more about generics in further study.

### 9.3.1 Declaring an ArrayList

Using an ArrayList requires that the appropriate class be loaded with

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

or

```java
import java.util.*;
```

An ArrayList suitable for storing for the sentence above (‘Java is a powerful language’), word by word, can be created with the statement

```java
ArrayList<String> words = new ArrayList<String>(5);
```

and in general ArrayLists are declared and instantiated with

```java
ArrayList<objectType> arrayListName = new ArrayList<objectType>(optionalBeginningSize);
```

Note the optionalBeginningSize. You may specify the initial size (number of 'spaces' available), but the ArrayList will automatically grow as needed to accommodate more objects added to it. Or you can leave the optionalBeginningSize out, in which case the ArrayList will be initially set to the default value 10. The ArrayList will still grow if needed.

Alternatively you can also declare an ArrayList in one statement and instantiate it in another.

```java
ArrayList<String> words;
```
Like the objects we've worked with before, ArrayLists have many useful methods for manipulation and we'll look at those now.

### 9.3.2 Putting an Element in an ArrayList

An ArrayList has values assigned to it using the add(value) method. Below are statements that would load words as created above.

```java
words.add("Java");
words.add("is");
words.add("a");
words.add("powerful");
words.add("language");
```

The add() method puts elements at the tail of the ArrayList. If the ArrayList is not large enough then it is automatically enlarged to accommodate additional information.

If we wanted to add a value read from the keyboard value we might have

```java
words.add(readln("Last word?
");
```

An element can also be inserted into an ArrayList using the Java method add(index, value) method. Any existing elements 'slide' toward the tail of the list, making room for the new element. Thus

```java
words.add(0, "Sun's");
```

gives us the list

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun's</td>
<td>Java</td>
<td>is</td>
<td>a</td>
<td>powerful</td>
<td>language</td>
</tr>
</tbody>
</table>

and then

```java
words.add(4, "very");
words.add(6, "programming");
```

gives us

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun's</td>
<td>Java</td>
<td>is</td>
<td>a</td>
<td>very</td>
<td>powerful</td>
<td>programming</td>
<td>language</td>
</tr>
</tbody>
</table>

The add(index, value) method may be used to insert an element at the tail of the ArrayList.
words.add(8, """"!""");

which would add a '!' to the end of words, but you cannot adding any farther past the tail of the ArrayList.

### 9.3.3 Traversing an ArrayList Using an Index

Values are retrieved from an **ArrayList** with the `get(index)` method.

```java
for(int i=0; i<8; i++)
{   println(words.get(i));    }
```

An **ArrayList** may also be traversed in reverse order.

```java
for(int i=7; i>= 0; i--)
{   println(words.get(i));    }
```

`words` may also be traversed *forwards only* using an enhanced `for` loop as shown here.

```java
for(String w : words)
{   println(w);    }
```

This enhanced `for` loop may be understood as these steps

*for every String in words*

```java
{    assign the String to w
    println w
}
```

In general an enhanced `for` loop is written

```java
for(objectType localName : arrayListName)
{   ...statements...    }
```

Note that `localName` is declared inside the `for` loop and so has scope limited to the `for` loop and can't be used outside the loop.

Traversing an **ArrayList** in the head to tail direction is a **forward traversal**, and the reverse is a **backward traversal**.

### 9.3.4 Size Related Methods
The number of elements (not necessarily the same as the number of allocated spaces) in words can be retrieved with the size() method.

```java
println(words.size());
```

An ArrayList may be checked for being empty (having no elements) with the isEmpty() boolean method.

```java
if (words.isEmpty() == false) {
   ...doSomething...
}
```

There is no corresponding isFull method. An ArrayList is never full, it automatically expands when more space is needed. (Actually the previous statement is not true. An ArrayList can be full if is uses all available memory but we won’t worry about this possibility and it won’t be a problem for us.)

The expression `arraylist.size() – 1` calculates the index of the last filled space in the ArrayList and can be used to traverse words no matter what the size it is. The code segment below traverses it in reverse order.

```java
for(int i=words.size()-1; i>=0; i--)
   println(words.get(i));
```

### 9.3.5 Removing an Element of an ArrayList

An element may be removed from an ArrayList using the remove(index) method. For example if we start with the original words ArrayList and then execute the statement

```java
words.remove(0);
```

we have

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>is</td>
<td>a</td>
<td>powerful</td>
<td>language</td>
</tr>
</tbody>
</table>

The ArrayList 'collapses' to fill in for the deleted element.

If we then execute

```java
words.remove(1);
words.remove(1);
```

we have

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>is</td>
<td>language</td>
</tr>
</tbody>
</table>

©Daniel L. Schuster, 2009, all rights reserved
because the first remove( ) deleted the "a" in position 1 and collapsed words, and the second remove( ) deleted the "powerful" now in position 1 and again collapsed words.

The collapsing of the ArrayList has significant implications for using the remove( ) method inside a loop. Let’s begin by assuming that words is

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Java</td>
<td>is</td>
<td>a</td>
<td>practical</td>
<td>programming</td>
<td>language</td>
</tr>
</tbody>
</table>

and note that the Strings containing "p" are in positions 3 and 4.

Now, in a loop starting a 0, remove every String that contains a 'p' using this code segment.

```java
for(int i=0; i<5; i++)
{
    String s = words.get(i);
    if (s.indexOf('p') != -1) //if there is a 'p' in the string then
        words.remove(i);     //remove that string from the arraylist
}
```

After the first removal ("practical") we have

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Java</td>
<td>is</td>
<td>a</td>
<td>programming</td>
<td>language</td>
</tr>
</tbody>
</table>

due to collapse of the ArrayList. Also note that the index variable i was 3 when we removed the word "practical" but becomes 4 due to the i++. 'programming' is now in position 3 and has been skipped. So the next remove( ) looks at "language", which does not have a "p". We have

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Java</td>
<td>is</td>
<td>a</td>
<td>programming</td>
<td>language</td>
</tr>
</tbody>
</table>

Because of this using remove( ) inside a loop is very problematic if there is any possibility of more than one item being removed. It is usually best to avoid it.

9.3.6 Modifying an Element of an ArrayList

An existing element of an ArrayList can be modified with set(index, value).

```java
words.set(0, "Java is a");
words.set(1, "powerful language");
```
The `set()` method is very useful for removing elements from consideration in the application. For example, if multiple aliens are in a game, stored in an `ArrayList`, and you are shooting at them, you might have something like

```java
for(int i=0; i<MAX_ALIENS; i++)
{
    if (shotHits(aliens[i]) == true)
    {
        remove(aliens.get(i)); //remove from window, NOT remove from arraylist
        aliens.set(i) == null;
    }
}
```

Then moving the `aliens` becomes

```java
for(int i=0; i <MAX_ALIENS; i++)
{
    if (aliens.get(i) != null)
    {
        aliens.get(i).move(xVal, yVal);
    }
}
```

### 9.3.7 ArrayLists and Methods

ArrayLists are objects and are passed to and returned from methods as we've done with other objects. The examples below illustrate this.

```java
display(words);
display(reverseWords(words));

doctoral void display(ArrayList<String> w)
{
    for(String s : w)
    {
        print(s + " ");
    }
} //display

public ArrayList<String> reverseWords(ArrayList<String> w)
{
    ArrayList<String> temp = new ArrayList<String>( );
    for(int i=w.size()-1; i>=0; i--)
    {
        temp.add(w.get(i));
    }
    return temp;
} //reverseWords
```

### 9.3.8 An Example Program

This program illustrates most of the techniques above.
ArrayListBasics
//ArrayListBasics.java
import java.util.ArrayList;
import acm.program.*;
public class ArrayListBasics extends ConsoleProgram
{
    public void run()
    {
        ArrayList<String> words = new ArrayList<String>(50);

        words.add("Java");
        words.add("is");
        println("Current size: " + words.size());
        words.add("a");
        words.add("powerful");
        words.add(readLine("Last word?")));

        println("Traverse by index");
        for(int i=0; i<5; i++)
            print(words.get(i) + " ");

        println("Reverse traverse by index");
        for(int i=words.size()-1; i>=0; i--)
            print(words.get(i) + " ");

        println("Add values");
        words.add(0, "Sun's");
        words.add(4, "very");
        words.add(6, "programming");
        for(int i=0; i<=words.size()-1; i++)
            print(words.get(i) + " ");

        println("Remove and size 1");
        println("Size: " + words.size());
        words.remove(0);
        for(int i=0; i<=words.size()-1; i++)
            print(words.get(i) + " ");

        println("Remove and size 2");
        println("Size: " + words.size());
        words.remove(3);
words.remove(4);
for(int i=0; i<=words.size()-1; i++)
{
    print(words.get(i) + " ");
}

print("\n\n");
println("Set");
words.set(0, "Suns's Java");
words.set(4, "programming language");
words.add(5, "but I prefer Captain Crunch");
for(int i=0; i<=words.size()-1; i++)
{
    print(words.get(i) + " ");
}
} //run
} //ArrayListBasics

And here's a sample run.

<table>
<thead>
<tr>
<th>Current size: 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last word? language</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Traverse by index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Java is a powerful language</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Reverse traverse by index</th>
</tr>
</thead>
<tbody>
<tr>
<td>language powerful a is Java</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Add values</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sun's Java is a very powerful programming language</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Remove and size 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size: 8</td>
</tr>
<tr>
<td>Java is a very powerful programming language</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Remove and size 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Size: 7</td>
</tr>
<tr>
<td>Java is a powerful language</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Suns's Java is a powerful programming language but I prefer Captain Crunch</td>
</tr>
</tbody>
</table>

### 9.3.9 Iterators and ArrayLists

A **list iterator** is an object designed to traverse other objects in the forward or backward direction. A list iterator may be thought of as a cursor, just like the cursor in word processing.
software, that fits between objects in an `ArrayList`, just as a word processor cursor fits between characters.

Traversing an `ArrayList` using an `int` index works perfectly well, but we can also do it with a list iterator. We'll restrict our application of list iterators to traversals but be aware that there is much more to list iterators than we discuss here.

We declare a list iterator for the `words` object with

```java
ListIterator<String> litr = words.listIterator();
```

`litr` is a `ListIterator` for `String` objects and is positioned before the first `String` in `words`. In general a list iterator is declared with

```java
ListIterator<objectType> iteratorName = objectToIterateOn.listIterator();
```

There are four `ListIterator` methods of interest

- `iteratorName.hasNext()` \(\leftrightarrow\) return `true` if there is an object after the cursor and `false` otherwise
- `iteratorName.hasPrevious()` \(\leftrightarrow\) return `true` if there is an object before the cursor and `false` otherwise
- `iteratorName.next()` \(\leftrightarrow\) return the next object if there is one and moves the cursor forward. If there is no next object it throws a fatal exception which crashes the program
- `iteratorName.previous()` \(\leftrightarrow\) return the previous object if there is one and moves the cursor backward. If there is no next object it throws a fatal exception and crashes the program

Using these to traverse `words` from head to tail is simple.

```java
while(litr.hasNext())
{
    String w = litr.next();
    print(w + " ");
}
```

`litr` has now been moved so that it is positioned after the last `String` in `words`, so traversing from tail to head is now possible.

```java
while(litr.hasPrevious())
{
    String w = litr.previous();
    print(w + " ");
}
```
9.4 Using ArrayLists in a Game

ArrayLists give us the capability of working with large number of objects. Let's look at the CannonadeWithArrays game we developed earlier and consider the changes needed when we increase the quantity of objects in the game.

9.4.1 Modifying CannonadeWithArrays

We replace as many arrays as possible with ArrayLists. Note that because ArrayLists only work with objects and not with primitive data types, any array of primitive data types is unmodified.

Instead of declaring an array of cannonballs we declare an ArrayList of Cannonballs and then initialize each individually.

```java
ArrayList<Cannonball> cbs;
cbs = new ArrayList<Cannonball>(CB_COUNT);
cbs.add(new Cannonball());
cbs.add(new Cannonball(Color.GREEN));
cbs.add(new Cannonball(Color.YELLOW));
cbs.add(new Cannonball(Color.RED));
```

We use a RandomGenerator rg to place all of the Cannonballs from the ArrayList cbs in the window, which has width AW. Note that this is exactly the same enhanced for loop as we used for arrays.

```java
for(Cannonball c : cbs)
{   add(c, rg.nextInt(0, AW-50), 0);  }
```

For every Cannonball still in the game, check it for being blocked and bounce the Cannonball if it is. The code is almost exactly the same as that for arrays.

```java
for(int i=0; i<CB_COUNT; i++)
{
    if ((cbsOutOfGame[i] == false) && (cbBoxes.get(i).intersects(blockerBox) == true))
    {
        yMoves[i] = -yMoves[i];
        xMoves[i] = rg.nextInt(-1, 1);
        score = score + 1;
    }
}
```

Here's the whole program.
CannonadeWithArrayLists

//CannonadeWithArrayLists.java
import acm.program.*;
import acm.graphics.*;
import acm.util.*;
import java.util.*;
import java.awt.*;
import java.awt.event.*;

public class CannonadeWithArrayLists extends GraphicsProgram
{
    public static void main(String [ ] args)
    { new CannonadeWithArrayLists().start(args); }

    public static final int APPLICATION_WIDTH = 500;
    public static final int APPLICATION_HEIGHT = 600;

    final int AW = APPLICATION_WIDTH;
    final int AH = APPLICATION_HEIGHT;
    final int WAIT = 8;
    final int MAX_TIME = 10000;
    final int CB_SIZE = 30, CB_COUNT = 4;
    final int MIN_Y VELOCITY = 3, MAX_Y_VELOCITY = 5;

    ArrayList<Cannonball> cbs;
    Blocker blocker;
    int time, score;
    GLabel scoreLbl;

    public void init( )
    {
        setBackground(Color.BLUE);

        time = 0;
        score = 0;
        scoreLbl = new GLabel(""+score);
        scoreLbl.setFont("SansSerif-BOLD-30");
        scoreLbl.setColor(Color.RED);
        add(scoreLbl, 2, 26);

        cbs = new ArrayList<Cannonball>(CB_COUNT);
        cbs.add(new Cannonball ());
        cbs.add(new Cannonball(Color.GREEN));
        cbs.add(new Cannonball(Color.YELLOW));
        cbs.add(new Cannonball(Color.RED));
RandomGenerator rg = new RandomGenerator();

for(Cannonball c : cbs)
{ add(c, rg.nextInt(0, AW-50), 0); }

blocker = new Blocker();
add(blocker, AW/2, AH/2);

addMouseListeners();
waitForClick();
} //init

public void mouseMoved(MouseEvent e)
{
    double x = e.getX();
    double y = e.getY();
    blocker.setLocation(x-40, y); //center blocker under mouse
} //mouseMoved

public void run()
{
    RandomGenerator rg = new RandomGenerator();
    ArrayList<GRectangle> cbBoxes = new ArrayList<GRectangle>(CB_COUNT);
    GRectangle blockerBox;

    boolean[ ] cbsOutOfGame = new boolean[CB_COUNT];

    int[ ] xMoves = new int[CB_COUNT];
    int[ ] yMoves = new int[CB_COUNT];

    for(int i=0; i<CB_COUNT; i++)
    {
        xMoves[i] = 0; //initially 0, later set randomly to 1, 0 or -1
        yMoves[i] = rg.nextInt(MIN_Y VELOCITY, MAX_Y VELOCITY);
    } //initialize x and y moves

    while(time < MAX_TIME) //game loop
    {
        pause(WAIT); //game loop
        time = time + WAIT;

        for(int i=0; i<CB_COUNT; i++)
        { cbBoxes.add(null); }

        for(int i=0; i<CB_COUNT; i++)
        {
cbs.get(i).move(xMoves[i], yMoves[i]);
cbBoxes.set(i, cbs.get(i).getBounds());
}
blockerBox = blocker.getBounds();

for(int i=0; i<CB_COUNT; i++)
{
    if (((cbsOutOfGame[i] == false) && (cbBoxes.get(i).intersects(blockerBox) == true))
    {
        yMoves[i] = -yMoves[i];
xMoves[i] = rg.nextInt(-1, 1); // left, center or right x movement
score = score + 1;
    }
}

for(int i=0; i<CB_COUNT; i++)
{
    if (((cbsOutOfGame[i] == false) && atTop(cbs.get(i)) == true)
        yMoves[i] = -yMoves[i];
}

for(int i=0; i<CB_COUNT; i++)
{
    if ((cbsOutOfGame[i] == false)
        && ((atLeft(cbs.get(i)) == true) || (atRight(cbs.get(i)) == true)))
        xMoves[i] = -xMoves[i];
}

for(int i=0; i<CB_COUNT; i++)
{
    if (((cbsOutOfGame[i] == false) && (atBottom(cbs.get(i)) == true))
    {
        remove(cbs.get(i));
cbsOutOfGame[i] = true;
score--;
    }
}

scoreLbl.setLabel("" + score);
if (allDone(cbsOutOfGame) == true)
{
    break;
}
} // game loop

// end the game
GLabel doneLbl = new GLabel("Game Over!");
doneLbl.setFont("SansSerif-BOLD-50");
9.5 Arrays vs. ArrayLists

Arrays and ArrayLists served very similar purposes. Both provide randomly accessible linear collections of data.

Arrays have two advantages
- somewhat simpler syntax
- can hold object or primitive data. However, the inability to use ArrayLists with primitive data types can be bypassed using the wrapper classes, so arrays really don't have much of an actual advantage here.
And **ArrayLists** also have two advantages

- the size grows dynamically as needed during the program run. This can be *very* important if you don't know the amount of data you have
- the included methods for adding and removing elements from the middle of an **ArrayList** are easy to use and powerful. The beginning programmer would have difficulty reproducing them for arrays.
Problem Set

Console Programs

ShowEvenPositions Write a program that loads an array with 20 random integers using a loop. Now display the values at position 0, position 2, ..., position 18.

ShowEvenValues Write a program that loads an array with 20 random integers using a loop. Now display only values that are even numbers.

TooBig1 Write a program that loads an array with 10 random doubles using a loop. Now attempt to print 11 values from the array. What happens and why?

LoadSquares Load an array with the integers 25, 36, 49, ..., 144. Traverse the array, calculating the sum which is then displayed.

ArrayOfStrings Write a program that loads 5 Strings from the keyboard using a loop, then displays them in reverse order.

BackAndForth Write a program that loads and array with 10 random integers using a loop. Now print those elements in order, then in reverse order, then prints 5 of them randomly.

ArrayEqual Write a program that loads two arrays of chars from the keyboard. Now write a boolean method isEqual(char[] a1, char[] a2) that tests the arrays for equality.

InsertAndDelete Write a program that declares an array of 10 strings. Now load the array of with 6 strings from the keyboard, putting them in positions 0 through 5 in the array, leaving positions 6 though 9 empty. Now present the user with a menu.
   a) delete string – the user inputs an integer representing a position to delete. If the position is currently occupied then move all of the strings after this position one "forward", overwriting the values already there. Empty spots are filled with "", an empty string. If the position is not currently occupied the deletion is denied.
   b) insert string – the user specifies a position and a string. If the position is already occupied, then the strings are all moved one position "backward", making room for the new string, which is inserted. The last item "drops off" the array if necessary. If the position is not already occupied the insertion is denied.
   c) display array – neatly display the entire array.

Here's a sample of the action on an array.

<table>
<thead>
<tr>
<th>position after loading 6 names</th>
<th>delete at position 3</th>
<th>delete at position 7</th>
<th>insert circle at position 2</th>
<th>after inserting 4 more at various locations</th>
<th>insert spam at position 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 dab tony george</td>
<td>dab tony george</td>
<td>Error: Can't delete</td>
<td>dab tony circle</td>
<td>dab line tony</td>
<td>dab line tony</td>
</tr>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

©Daniel L. Schuster, 2009, all rights reserved
Graphics Program

**PlaceUFOs** Recall the UFO2 class that allowed us to create a default UFO and a custom colored UFO. Write a program that loads an array of 10 UFO2s and adds them randomly in the window.

**UFORace** Write a program that places 4 UFOs at random heights along the left edge of the window. The UFOs move, with differing velocity, to the right edge. When the edge is reached the losing UFOs are removed and the winner flashes on and off.

**UFOShootDown** Create a UFO shooting game that allows you to shoot at a target as it moves across the screen. There is only one target on screen at a given time, the speed, size and starting location of the targets vary. As you shoot at a target you can see the bullets fly upward and you can fire multiple shots at a target. The gun is at the bottom of the window and moves left and right with the mouse and fires with a mouse click.
**UFOShootDown2** Modify the UFOShootDown game to allow for multiple UFOs and multiple bullets. You will need an array of ufos and an array of bullets, and you will need nested loops to check for collision of every ufo with every bullet.

**DrawingProgram** Create a mouse controlled drawing program that allows you to draw rectangles, ovals and lines. You'll need arrays (or arraylists) to store the components of your graphic images as they are created. The program, with a ufo that's been drawn might look like this: