

Chapter 8: Arrays and Array Lists

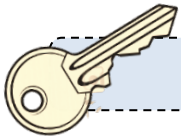
Learning Outcomes:

- ✓ Identify the key advantages associated with using arrays, as opposed to variables
- ✓ Implement both one-dimensional and multidimensional arrays into a program
- ✓ Implement an array list into a program



Prerequisite Knowledge:

- ✓ Complete Chapter 7
- ✓ Be able to confidently use iteration and selection control structures in a program
- ✓ Be able to implement independent methods, that use parameter variables, in a program



Primitive

Multidimensional

Iterator

Array List

Keywords

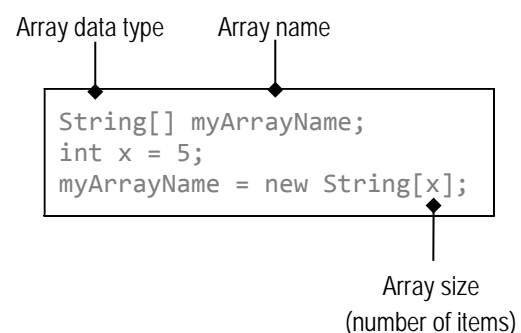
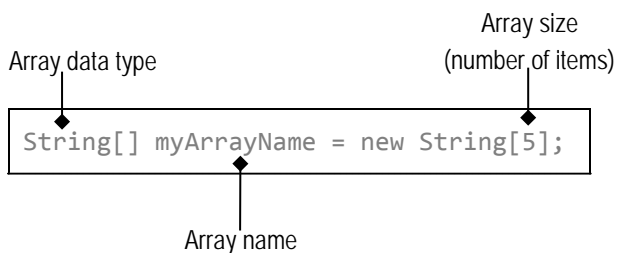
8.1 The Theory: Variables Vs Arrays

A variable can only hold one piece of information at any one instance; for example, `[int x;]` can hold a single integer value, although its value may change numerous times during a program's execution. Although variables are useful, and without them none of the earlier programs would have been possible, the concept of storing a single piece of data per variable will result in some tasks being extremely arduous and tedious. Fortunately, most programming languages provide some form of array; an array is a variable type that can store multiple values instantaneously.



Primitive Arrays

Most modern-day programming languages provide a variety of array types, and Java is no different; perhaps the simplest are known as primitive type arrays. Below are two examples of syntax used to create primitive arrays in Java. The square brackets are used to indicate the declaration of an array; for example, `[int[]]` declares a primitive integer array. The array name follows the array declaration, the same as a variable, and the keyword `[new]` is used to instantiate the array (to create an array object in memory). Finally, the size of an array is set by entering a value into the square brackets that proceeds after the `[new]` keyword and the data type, for example `[new String[5]]`. The second example below illustrates how an array can be instantiated later in a program's existence, as well as setting the array size using an integer variable – a variable which, in theory, could be set once the size of the array needed has been determined. In extension of this, the `[readLine()]` method could be used to retrieve the size of the array needed from a user, before building the array (of specified size) in memory.



8.2 Practical: Primitive Arrays and Loops

Using Primitive Arrays in a Program

Once an array object has been instantiated (after using the `[new]` keyword), an array is ready for use in a program. To store data into an array, just like a variable, the assignment operator `[=]` (the equals symbol) is used. However, because an array holds multiple items, an index of the particular item being referred to must be specified too; this is achieved by entering a numerical index value into the square brackets that proceed after an array's name. Importantly, the index value begins at 0; therefore if an array is declared with a size of 3, the item index will include item 0, item 1 and item 2.

Activity 8.1

Write a program that utilises a one-dimensional, primitive integer array. The array should hold five lucky numbers. Use the array to print the five numbers to the command-line.

The array created in the example below is referred to as one-dimensional; a one-dimensional array can be visualised as a single column in a spreadsheet. Arrays, however, can be multidimensional. The syntax below declares a primitive string array that is used to hold three name values; those names are later printed to the command-line.

```
String[] myArrayName = new String[3];  
myArrayName[0] = "Joe";  
myArrayName[1] = "James";  
myArrayName[2] = "Junaid";  
  
System.out.println(myArrayName[0]);  
System.out.println(myArrayName[1]);  
System.out.println(myArrayName[2]);
```

myArrayName	
0	Joe
1	James
2	Junaid

```
C:\Windows\system32\cmd.exe  
Joe  
James  
Junaid  
Press any key to continue . . .
```

Activity 8.2

Write a simple lottery program that generates five random numbers and a sixth bonus number, before storing them all into a primitive array. The program should then print the numbers (from the array) to the command-line. The lottery numbers should be between 1 and 54.

```
C:\Windows\system32\cmd.exe  
--->The winning lottery numbers are:  
53, 26, 38, 47, 10  
And the bonus ball is; 11  
Press any key to continue . . .
```

Programming Tips!

To generate a random number, use the `Random` class. Import the class first by adding this code `[import java.util.Random;]` to the top of a program. The following syntax can then be used to generate a random number between 1 and 54:

```
Random rand = new Random();  
int num = rand.nextInt(54) + 1;
```

Programming Tips!

Arrays are fundamental in programming. In fact, arrays and loops together are a programmer's best friend, thus are looked at next. While on the topic of arrays, ever noticed `[String[] args]` in the main method? This is a string array, an array that can be used to pass a program additional 'arguments' on start-up – in other words, additional commands when a program first initiates.

Arrays and Loops in Tandem

The previous programs used the `println()` method to print each item stored in a primitive array to the command-line. Although this technique is adequate when working with arrays that contain few items, repeating the same process with a larger array is going to be costly, both in time spent and lines of code written. Fortunately though, iteration (loops) can be easily used in conjunction with an array to output all array items in just a few lines of code – much more efficient than using the `println()` method numerous times. The syntax to achieve this is demonstrated in the example below:

```
String[] myNames = new String[5];  
myNames [0] = "Joe";  
myNames [1] = "James";  
myNames [2] = "Junaid";  
myNames [3] = "John";  
myNames [4] = "Jason";  
  
for(int i = 0; i < 5; i++){  
    System.out.println(myNames [i]);  
}
```



Programming Tips!

There is a shorthand declaration that can be used to instantiate an array with a set of default values. The syntax for this is shown below:

```
char[] wordArray = {'b','o','o','k'};
```

The code above declares a char array with four items; note that when working with char in Java, single quotes are used, as opposed to double quotes.

There is a fundamental problem with 'hard-coding' a loop to cycle through the array items 0 to 4. What if an array's size varies in each life cycle of a program? Suddenly, being able to determine the length of an array becomes very important! Thankfully, however, there is already a function that does just that, `length`.



```
String[] myNames = new String[5];  
myNames [0] = "Joe";  
myNames [1] = "James";  
myNames [2] = "Junaid";  
myNames [3] = "John";  
myNames [4] = "Jason";  
  
for(int i = 0; i < myNames.length; i++){  
    System.out.println(myNames[i]);  
}
```

```
C:\Windows\system32\cmd.exe  
The seven dwarfs are;  
1> Doc  
2> Sleepy  
3> Bashful  
4> Grumpy  
5> Happy  
6> Sneezy  
7> Dopey  
Press any key to continue . . .
```



Programming Tips!

By using the `length` function to determine the number of items stored in the array, the loop is now able to handle an array of any size: 1 to 100, 1 to 10,000, etc.

Finally, be aware that the `length` function has no brackets, and thus accepts no arguments.



Activity 8.3

Write a small utility program that uses a primitive string array to store all of the names of the seven dwarves. Use a loop and the `length` function to print all of the names stored inside of the array to the command-line.

Can't remember the names? That's ok, they are: Doc, Grumpy, Happy, Sleepy, Bashful, Sneezy and Dopey.

8.3 Practical: Arrays in Programs

Arrays and Readlines

The code below demonstrates how an array's size can be set during the life cycle of a program. The program uses two FOR loops, both of which cycle through each item in the array. The first loop reads multiple values from the user, storing each entered value into the array. The second loop prints the value stored inside of each array item to the command-line.

Try the code



```
import java.io.*;
public class MyFriends{

    public static void main(String[] args){
        String[] myFriendList; //String array to hold friend names
        BufferedReader y = new BufferedReader(new InputStreamReader(System.in));
        System.out.println("How many friends do you have?");
        String myInput = null;
        try{ //gets the number of friends
            myInput = y.readLine();
        }catch(IOException e){
            System.out.println("Error: " + e);
        }
        int myNum = Integer.parseInt(myInput); //converts the number to integer
        myFriendList = new String[myNum]; //creates the array and sets the size
        System.out.println("\n\n<-- Enter your " + myNum + " friend names now -->");
        for(int i = 0; i < myFriendList.length; i++){ //loops through the array
            System.out.println("\n -Enter friend number " + (i+1));
            try{
                myFriendList[i] = y.readLine(); //reads each name
            }catch(IOException e){ //and stores into the array
                System.out.println("Error: " + e);
            }
        }
        System.out.println("\n\n<----- Your friends are: ----->");
        for(int i = 0; i < myFriendList.length; i++){ //loops through the array
            System.out.println((i+1)+") " + myFriendList[i]); //prints each name
                                                    //stored in the array
        }
    }
}
```

```
C:\Windows\system32\cmd.exe
How many friends do you have?
3
<----- Enter your 3 friend names now ----->
-Enter friend number 1
Joe
-Enter friend number 2
Dave
-Enter friend number 3
Steve
<----- Your friends are: ----->
1> Joe
2> Dave
3> Steve
Press any key to continue . . .
```

Activity 8.4



Write a simple program that asks the user to input the number of car manufacturers that they are familiar with. The program should then prompt the user to input each manufacturer; the program should store them into a primitive string array. Finally, the program should print all of the array items back to the command-line. Use a WHILE loop to repeat the program if the user would like another attempt.

The Simple Word Guessing Game

The next activity involves building a simple word guessing game; to do so the skills learnt from Chapter 7 will need to be applied too, in particular the use of static global variables. Remember, if something is referred to as 'static', then it belongs to the class, and thus the class name must be used when referencing it, i.e. [MyClassName.myStaticVariableName].



Before writing any code, consider the logical steps needed to create this program. First, two char arrays are required: `wordArray` to store the complete word, and `letterSpaces` to store underscore characters (representing a missing letter) and any letters guessed by the user.

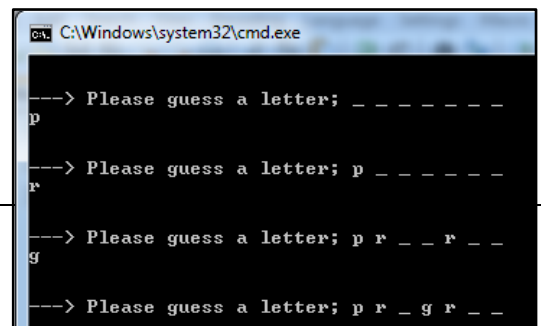
Secondly, print all of the characters stored in the `letterSpaces` array to the command-line, and await the user to input a guessed letter. Thirdly, compare the guessed letter (from the user) to the `wordArray` array; if there is a match store the given letter into the `letterSpaces` array, into the same position as found in the `wordArray` array. Finally, continue to loop the program until there are no more underscore characters present in the `letterSpaces` array.

```
import java.io.*;

public class MyWordGame{
    static char[] wordArray = {'p','r','o','g','r','a','m'}; //static global variables
    static char[] letterSpaces = {'_','_','_','_','_','_','_'}; //belong to the class

    public static void main(String[] args){
        BufferedReader y = new BufferedReader(new InputStreamReader(System.in));
        Boolean count = true;
        while(count==true){
            System.out.print("---> Please guess a letter; ");
            for(int i = 0; i < MyWordGame.letterSpaces.length; i++){
                System.out.print(MyWordGame.letterSpaces[i] + " ");
            }
            //finish code here

        } //end of loop
    }
}
```



Programming Tips!

The first line of code below can be used to obtain the first character found in a string. The remaining code is used to compare the user's guess against the `wordArray` array. If the guess exists in the array, the letter is saved to the same position in the `letterSpaces` array.

```
char userGuess = readLineInput.charAt(0);
for(int i = 0; i < MyWordGame.wordArray.length; i++){
    if(userGuess == MyWordGame.wordArray[i]){
        MyWordGame.letterSpaces[i] =
        userGuess;
    }
}
```

Activity 8.5a

Complete the **simple word guessing game**; the program should continue to loop until the user guesses all of the letters found in the hidden word. To help, some of the code has already been written!



The Advanced Word Guessing Game

The 'simple word guessing game', as featured on the previous page, appears to function adequately to the user, but behind the scenes the code is limited. The code is written into a single method, the `main` method; this results in poor design that provides no support for maintainability or future modifications. Chapter 7 focused, in detail, on how to divide a program up into smaller, more manageable parts; these parts are of course referring to methods and functions. The guessing game in the previous chapter is no different, and would benefit greatly from a complete overhaul of its design. Below are a set of independent functions and methods, ready to be used in a 'well-defined' version of the word guessing game: a version that is of sound design, a version that is better prepared for change.

Introducing the `[checkLetters()]` function;

```
public boolean checkLetters(char[] word, char guessedLetter){
    for(int x = 0; x < word.length; x++){
        if(word[x] == guessedLetter){
            return true;
        }
    }
    return false;
}
```

```
char[] x = {'b','o','o','k'};
char y = 'o';
boolean myLetterCheck = myObj.checkLetters(x,y);
```

This function can be used to determine if a specific character exists in a char array; the function will return either a true or false answer, dependent on the result. The parameters required in order for the function to execute are a char array `[char[] word]` and a char variable `[char guessedLetter]`. The FOR loop iterates from 0 to the number of items found in the `word` character array. The IF statement inside of the loop compares each character item stored in the array to the `guessedLetter` character; if there is a match the function returns true, else if no instances are identified the function returns false.

Introducing the `[addLetters()]` function:

```
public char[] addLetters(char[] word, char[] letterBlanks, char guessedLetter){
    for(int x = 0; x < word.length; x++){
        if(word[x] == guessedLetter){
            letterBlanks[x] = guessedLetter;
        }
    }
    return letterBlanks;
}
```

```
char[] x = {'b','o','o','k'};
char[] z = {'_','_','_','_'};
char y = 'o';
z = myObj.addLetters(x,z,y);
```

If the `[checkLetters()]` function returns true (meaning that the user's guess was correct), the next step is to add the user's guess to the appropriate array; the `[addLetters()]` function will do exactly that. The function will add the user's chosen character to the `letterBlanks` array. The parameters required in order for the function to execute are two char arrays `[char[] word, char[] letterBlanks]`, and a char variable `[char guessedLetter]`. The FOR loop in the function is similar to the `[checkLetters()]` function, except each time the IF statement successfully matches the value of `guessedLetter` to an item found in the `word` array, the inner statement will store the user's guessed letter to the same position in the `letterBlanks` array. Finally, the updated `letterBlanks` character array is returned.

Introducing the [isGameOver()] function:

```
public boolean isGameOver(char[] letterBlanks){
    for(int x = 0; x < letterBlanks.length; x++){
        if(letterBlanks[x] == '_'){
            return true;
        }
    }
    return false;
}
```

```
char[] z = {'_', 'o', 'o', '_'};
boolean ContinueGame = myObj.isGameOver(z);
```

This function can be used to determine whether the game has finished; the function will return either a true or false answer, dependent on the result. The parameter required in order for the function to execute successfully is a char array [char[] letterBlanks]. The function code loops through all items in the char array, checking whether the underscore character is present; if the underscore character is present, then the function returns true (to continue the game), else the function returns false (to end the game).

Introducing the [generateWord()] method:

```
private char[] wordArray; //global properties
private char[] letterSpaces;
public void generateWord(){ //void indicates no return value
    Random rand = new Random();
    int num = rand.nextInt(5) + 1;
    if(num == 1){
        wordArray = new char[] { 'e', 'd', 'u', 'c', 'a', 't', 'i', 'o', 'n' };
    } else if (num == 2) {
        wordArray = new char[] { 'p', 'r', 'o', 'g', 'r', 'a', 'm' };
    } else if (num == 3) {
        wordArray = new char[] { 'e', 'l', 'e', 'p', 'h', 'a', 'n', 't' };
    } else if (num == 4) {
        wordArray = new char[] { 'b', 'o', 'o', 'k' };
    } else if (num == 5) {
        wordArray = new char[] { 'j', 'u', 'r', 'a', 's', 's', 'i', 'c' };
    }
    letterSpaces = new char[wordArray.length];
    for(int i = 0; i < wordArray.length; i++){
        letterSpaces[i] = '_';
    }
}
```

The [generateWord()] method has two purposes: first to select a random word to store in the wordArray character array, and second to generate another character array (the letterSpaces character array) with the same number of items as the first array. The second array is populated with underscore characters, to represent the unguessed letters. The wordArray and letterSpaces array are both global properties, indicating that their scope is available to all methods/functions belonging to the same object. This method in particular takes full advantage of the global status, setting the values of both array properties. The method selects a random word by first generating a random number and then using an IF statement to determine which random word is to be stored in the wordArray array. The loop at the end of the method is used to iterate through the letterSpaces array, setting each item to the underscore character. The size of the letterSpaces array is set to the same as the wordArray array, by using the following line of code [letterSpaces = new char[wordArray.length];].

The pre-written functions/methods on the previous pages cover the major steps that are required to build the 'advanced word guessing game', but the program is still incomplete; it has no start point, no end point and, more importantly, no sequence. To finish the program, the sequential steps of the game still need to be written and this can be expressed in the `main` method. Once an object has been created from the class (see Chapter 7 again for more details), using this line of code [`MyWordGame myWrdObj = new MyWordGame();`], the appropriate methods/functions can be called in the correct order. 'To get the ball rolling,' some of the `main` method has already been written.



```
import java.util.Random;
import java.io.*; //imported classes needed for the word guessing program

public class MyWordGame{
    private char[] wordArray; //global properties
    private char[] letterSpaces;

    public static void main(String[] args){
        MyWordGame myWrdObj = new MyWordGame();
        myWrdObj.generateWord(); //object.callMethod

        //<----- finish program code here
    }
}
```

```
C:\Windows\system32\cmd.exe
Welcome to 'Guess that Word!';
<----->
--> Please guess a letter; _ _ _ _ _
e
---- You have -> 10 <- lives remaining ----
--> Please guess a letter; e _ _ _ _ _
f
---- You have -> 9 <- lives remaining ----
--> Please guess a letter; e _ _ _ _ _
a
---- You have -> 9 <- lives remaining ----
--> Please guess a letter; e _ _ _ a _ _ _ _
```



Programming Tips!

This is a tough challenge, so don't be afraid to revisit Chapter 7 if the whole 'method/function' thing is just not clicking!

The sequence of the guessing game is:

- Enter a WHILE loop to keep repeating the game sequence.
- Call the `[generateWord()]` function to populate both the global array properties; one to hold the word, the other to hold the underscore characters.
- Print the content of the `letterSpaces` character array to the command-line and await a guess from the user.
- Call the `[checkLetters()]` function to see if the user's guess is a match. If there is a match (the function has returned true), call the `[addLetters()]` function to add the guessed letter(s) to the `letterSpaces` array.
- Call the `[isGameOver()]` function to check whether the game (the outer loop) is to continue or end.

Activity 8.5b – Super Hard!



Write the pre-written methods into a class and use them to complete the advanced version of the word guessing game. Note that it is only the `main` method that requires any attention.

Add a 'lives' facility to the game; the user should only have 10 guesses. After each incorrect guess a life should be deducted. The user should be informed of the number of lives remaining after each guess.

8.4 The Theory: Multidimensional Arrays

Two-Dimensional Primitive Arrays

Arrays do not have to be one-dimensional; in fact, they can have numerous dimensions, and multidimensional arrays can be created inside of other multidimensional arrays; what a headache! A two-dimensional array, however, can be visualised as a spreadsheet – a grid in memory that consists of both rows and columns, with each cell having a unique reference. For example, this code `[char[][] x = new char[4][4];]` would generate a grid in memory similar to the table below:

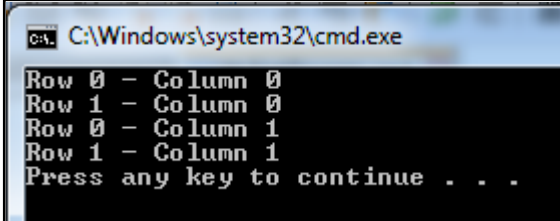
0,0	0,1	0,2	0,3
1,0	1,1	1,2	1,3
2,0	2,1	2,2	2,3
3,0	3,1	3,2	3,3

The code below declares a two-dimensional, primitive string array; the array has two columns and two rows, a total of four items. The assignment operator [=] is used to assign each array item a value, before later printing the values back to the command-line.

```
public class MyTwoDimensionalArray{
    public static void main(String[] args){
        String[][] x = new String[2][2];

        x[0][0] = "Row 0 - Column 0";
        x[1][0] = "Row 1 - Column 0";
        x[0][1] = "Row 0 - Column 1";
        x[1][1] = "Row 1 - Column 1";

        System.out.println(x[0][0]);
        System.out.println(x[1][0]);
        System.out.println(x[0][1]);
        System.out.println(x[1][1]);
    }
}
```



Programming Tips!

The double brackets are used to express a dimension in an array; thus two brackets (`[]`) represent a two-dimensional array and three brackets (`[][]`) represent a three-dimensional array! A third dimension may be visualised as additional worksheets added to a spreadsheet.



Programming Tips!

There is a shorthand version for declaring multidimensional arrays, with default values, too. The syntax below will create an array with the same dimensions and values as the code above:

```
String[][] x = new String[][]{
    {"Row 0 - Column 0", "Row 0 - Column 1"},
    {"Row 1 - Column 0", "Row 1 - Column 1"}
};
```

8.5 Practical: Using Multidimensional Arrays

The Translator

Written below is a function that returns a two-dimensional, primitive string array (`String[][]`); the array is populated with phrases ready to translate words from English to German and/or French. The array is written in the shorthand notation, and has 16 rows and 4 columns, a total of 64 items. The first column holds an ID number (used to identify the phrase), the second column holds an English word, the third column holds a German equivalent translation and the fourth holds a French alternative. The function below is independently written and thus is ready to be used in any translator class:

```
public String[][] generatePhrases(){
    String[][] myPhrases = new String[][] //Rows first, then columns
    {
        {"1","House","Haus","Maison"},
        {"2","Street","Strasse","Rue"},
        {"3","Left","Links","Gauche"},
        {"4","Right","Recht","Droit"},
        {"5","Straight Ahead","Geradeaus","Tout Droit"},
        {"6","Light","Licht","Lumiere"},
        {"7","Time","Zeit","Temps"},
        {"8","Door","Tur","Porte"},
        {"9","Coat","Mantel","Manteau"},
        {"10","Shirt","Hemd","Chemise"},
        {"11","Colour","Fabre","Couleur"},
        {"12","Horse","Pferd ","Cheval"},
        {"13","Cow","Kuh","Vache"},
        {"14","Pig","Schwein","Porc"},
        {"15","Sheep","Schaf","Mouton"},
        {"16","Goat","Ziege","Chevre"}
    };
    return myPhrases;
}
```

```
String[][] myWords = myObjTrans.generatePhrases();
```

```
C:\Windows\system32\cmd.exe
--*-- The Translator --*--
----->Choose a word to translate (enter a number ID):
1> House
2> Street
3> Left
4> Right
5> Straight Ahead
6> Light
7> Time
8> Door
9> Coat
10> Shirt
11> Colour
12> Horse
13> Cow
14> Pig
15> Sheep
16> Goat
3
-----
Left in German is Links
Left in French is Gauche
-----
Press any key to continue . . .
```

Activity 8.6

The Translator – write a program that can be used to translate English words to both German and French. The words have already been prepared in the `[generatePhrases()]` function; the function is complete and ready to be used in the program.

Write the rest of the code into the `main` method; the code should be as concise as possible, therefore use appropriate iteration (loops) to print the required values to the command-line.



8.6 Practical: Array Lists

Using Array Lists in a Program


Arrays are great, there is no doubt about it, but there is one small drawback; once they are defined, they are defined. In other words, once an array is declared with five items, for example, the array will always have five items; there is no changing the size. In context, this would cause an issue with a 'pizza ordering program', as there is no sure way to know exactly how many pizzas will be ordered, unless, of course, the customer is not allowed to change their order once made! This, however, is likely to result in many unhappy pizza-ordering customers. Never fear though, array lists are here! Array lists are dynamic, and therefore their size can grow or shrink as needed. Got an extra pizza? No problem, the array list will simply add it! No longer want the pizza? No problem, the array list will simply delete it!

```
import java.util.ArrayList; //import the ArrayList class
public class MyArrayList{

    public static void main(String[] args){
        ArrayList<String> myWordList = new ArrayList<String>();

        myWordList.add("Joe"); //add item to the array list
        myWordList.add("James");
        myWordList.add("John");

        System.out.println(myWordList.get(0));
        System.out.println(myWordList.get(1));
        System.out.println(myWordList.get(2));
        //retrieve item from array list
    }
}
```



Try the code

```
C:\Windows\system32\cmd.exe
Joe
James
John
Press any key to continue . . .
```



Above is the syntax that is used to declare an array list; note that an array list must have a data type to indicate what the array will hold, i.e. string, integer or even custom defined data types; the data type is declared between the angled brackets [`<String>`]. When using an array list, the class must be imported using the following line of code at the top of a program [`import java.util.ArrayList;`]. Two useful methods that can be used to manipulate an array list are: the [`add()`] method to add an item to the list, and the [`get()`] method to retrieve an item from the list.



```
import java.util.ArrayList;
import java.util.Iterator; //import Iterator class
public class MyArrayList{
    public static void main(String[] args){
        ArrayList<String> myWordList = new ArrayList<String>();
        myWordList.add("Joe");
        myWordList.add("James");
        myWordList.add("John");
        myWordList.add("Dean");

        myWordList.remove(1); //remove item from the array list
        Iterator myIteration = myWordList.iterator(); //Create Iterator Object
        while(myIteration.hasNext()){ //continue loop while there are array items
            System.out.println(myIteration.next()); //write next item to screen
        }
    }
}
```

```
C:\Windows\system32\cmd.exe
Joe
John
Dean
Press any key to continue . . .
```

The example above introduces some other useful techniques when working with array lists. The first is the `remove()` method, which can be used to remove an item from the list; this is achieved by specifying the index number (which in this case is 1). The second useful technique is the use of the `Iterator` class to cycle through the items stored in the array list; an instantiated object from this class has two useful methods, `hasNext()` and `next()`. The `hasNext()` method returns either a true or false value, depending on whether there is another item in the array; this makes it perfect for use in a `WHILE` loop because when there are no more items in the list, the value false will be returned (ending the loop). The `next()` method can be used to retrieve the next item in the array; this can be used, for example, to write the next item to the command-line. Note that the `Iterator` class must be imported by using the following line of code `import java.util.Iterator;` at the top of a program.

Activity 8.7



Lottery Revisited – write a lottery program that generates five random numbers (between 1 and 54) and a bonus number; the program should then add the numbers to an `<Integer>` array list. Use the `Iterator` class to cycle through the items and print the numbers to the command-line.

Now add two more random numbers to the array list and print those too!



```
C:\Windows\system32\cmd.exe
--> The winning numbers are; 9, 47, 41, 25, 39 and the bonus is 36
Press any key to continue . . .
```