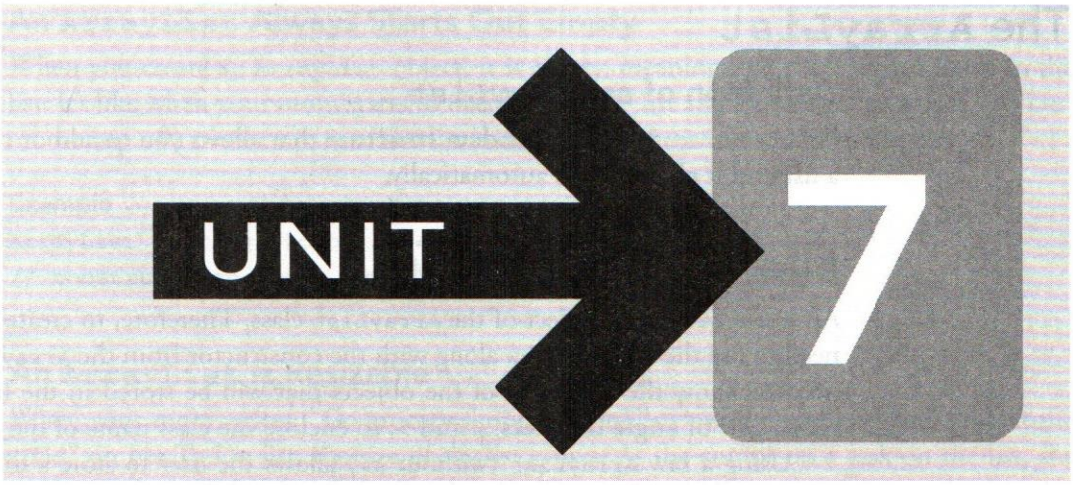


ArrayList

IN THIS UNIT

Summary: In this unit we will discuss the `ArrayList`, along with some of the methods that are used to access and manipulate its elements. In addition to the array algorithms that were introduced in the previous unit (which can also be done with `ArrayLists`), we will look at searching and sorting algorithms that you need to know for the AP Computer Science A Exam.

Key Ideas

- An `ArrayList` is a data structure that can store a list of objects from the same class.
- An `ArrayList` resizes itself as objects are added to and removed from the list.
- To traverse a data structure means to visit each element in the structure.
- The enhanced `for` loop (`for-each` loop) is a special looping structure that can be used by either arrays or `ArrayLists`.
- The sequential search algorithm finds a value in a list.
- The Insertion Sort, Selection Sort, and Merge Sort are sorting routines.

The `ArrayList`

Definition of an `ArrayList`

An `ArrayList` is a **complex data structure** that allows you to add or remove objects from a list and it changes size automatically.

Declaring an `ArrayList` Object

An `ArrayList` is an object of the `ArrayList` class. Therefore, to create an `ArrayList`, you need to use the keyword `new` along with the constructor from the `ArrayList` class. You also need to know the data type of the objects that will be stored in the list. The `ArrayList` uses a pair of **angle brackets**, < and >, to enclose the class name of the objects it will store.

Using a raw `ArrayList` (without <>) allows the user to store a different data type in each position of the `ArrayList`, which is not typically done since Java 5.0 arrived in 2004. The `ArrayList <E>` notation is preferred over `ArrayList` because it allows the compiler to find errors that would otherwise be found at run-time. You might see some released AP Computer Science A Exam questions from previous years using this syntax.

General Form for Declaring an `ArrayList`

```
ArrayList <E> nameOfArrayList = new ArrayList<E>();
```

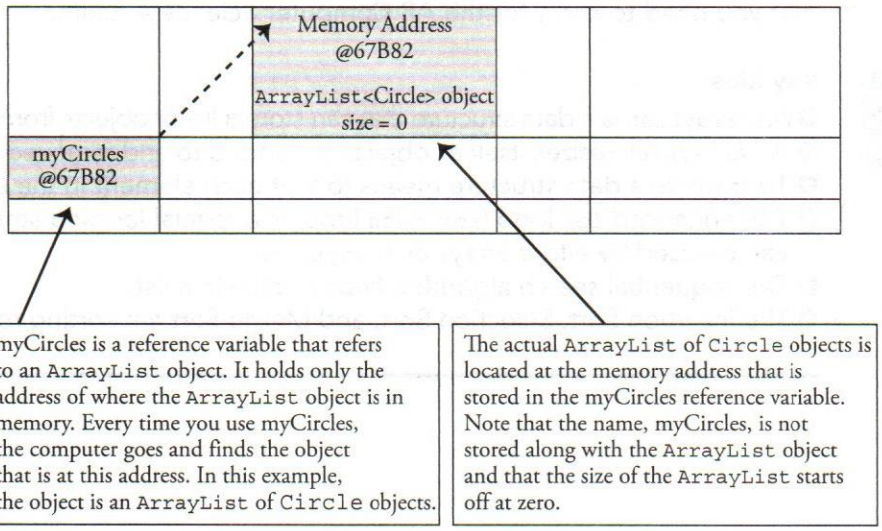
where the generic type E specifies the type of elements that will be stored in the `ArrayList`

The graphic that follows is a visual representation of what the `ArrayList` looks like in memory.

Example

Declare an `ArrayList` of `Circle` objects. Please note that I am referring back to the Circle class from Unit 5 and that the memory address is simulated.

```
ArrayList<Circle> myCircles = new ArrayList<Circle>();
```



An `ArrayList` Always Starts Out Empty

When you create an `ArrayList` object, it is empty, meaning that there are no items in the list. It's like when your mom starts to make a "To Do" list and she writes the words "To Do" on the top of a piece of paper. The list is created but there is nothing in the list.

Example

```
ArrayList <String> toDoList = new ArrayList <String> ();
// constructs an empty list
```

An `ArrayList` Is Resizable

When your mom writes, "Go grocery shopping" or "Buy awesome video game for favorite child" on her To Do list, the size of the list grows. As she completes a task on the list, she crosses it out and the size of the list shrinks. This is exactly how an `ArrayList` is resized.

> **Automatic Resizing**
>
> An awesome feature of the `ArrayList` is its ability to resize itself as elements are added to or removed from the list. The `size()` method (explained later) immediately recalculates how many elements are in the list.

An `ArrayList` Requires an `import` Statement

The `ArrayList` is not part of the built-in Java language package, so you have to let the compiler know that you plan on creating an `ArrayList` by putting an import statement prior to the class declaration.

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

You will not see or need to write any import statements on the AP Computer Science A exam.

An `ArrayList` Can Only Store Objects

Unlike the array, which could store primitive variables like an `int` or `double`, as well as objects, like Strings, the `ArrayList` *can only store objects*. If you want to store an `int` or `double` in an `ArrayList`, you must use the `Integer` or `Double` wrapper classes. This is one of the reasons why the wrapper classes were created (refer back to Unit 2 when they were introduced).

Example

Create an `ArrayList` of `Integer`. Add an `int` to the `ArrayList` and secretly watch as the `int` is automatically converted to an integer using a secret, Java black box technique called **autoboxing** (also introduced back in Unit 2).

```
ArrayList<Integer> myFavoriteIntegers = new ArrayList<Integer>();
int num = 45;
myFavoriteIntegers.add(num);          // The 45 is converted to an Integer
```

Important `ArrayList` Methods

The `ArrayList` class comes with a large array of methods (see my pun). These methods make it easy to work with the objects inside the `ArrayList`. The AP Computer Science A Exam does not require you to know all of the methods from the `ArrayList` class; however, it does require you to know several of them.

The `add` Method

There are two **add** methods for the ArrayList. The **add(E object)** method **appends** the object to the end of the list. It also returns the value true. The size of the `ArrayList` is automatically updated to reflect the addition of the new element.

> **What's with `E`?**
> The data type **E** is known as a **generic type**. It simply means that you can put any kind of data type here. I like to say, "The method takes **E**very kind of data type."

Example

Create an `ArrayList` of `Circle` objects. Add three `Circle` objects to the `ArrayList` where the first has a radius of 8, the second doesn't provide a radius so the radius gets the default value of 0, and finally, the last circle has a radius of 6.5. Note: This example will be used in the explanations for the other `ArrayList` methods.

```
ArrayList<Circle> myCircles = new ArrayList<Circle>();

line 1: myCircles.add(new Circle(8));
line 2: myCircles.add(new Circle());
line 3: myCircles.add(new Circle(6.5));
```

After line 1 is executed: There is one `Circle` object in the `ArrayList`.

index	myCircles
0	Circle with radius 8.0

A `Circle` object is added to the `ArrayList`.

After line 2 is executed: There are two `Circle` objects in the `ArrayList`.

index	myCircles
0	Circle with radius 8.0
1	Circle with radius 0.0

After line 3 is executed: There are three `Circle` objects in the `ArrayList`.

index	myCircles
0	Circle with radius 8.0
1	Circle with radius 0.0
2	Circle with radius 6.5

Another `add` Method

The **add(int index, E object)** method inserts the object into the position `index` in the `ArrayList`, shifting the object that was previously at position index and each of the objects after it over one index. The index for each of the objects affected by the **add** is incremented. The method does not return a value.

An `ArrayList` Is Like the Lunch Line at School

An `ArrayList` can be visualized like the lunch line at school. Imagine there are 10 students in line and they are numbered 0 through 9. Person 0 is the first person in line.

Suppose the unthinkable happens and a student walks up and cuts the line. They have just inserted themselves into the list. If the *cutter* is now the third person in line, then they have just performed an add(2, "cutter"). This action impacts everyone who is in line *after* the cutter. The person who used to be in position 2, is now in position 3. The person who used to be at position 3 is now in position 4, and so on. The index for each person behind the cutter was *incremented by one.*

```
line 4:  myCircles.add(1, new Circle(4));
```

After line 4 is executed: The `Circle` object with a radius of 4 was inserted into the position with an index of 1 (the second position). All of the `Circle` objects after it had to move over one slot.

index	myCircles
0	Circle with radius 8.0
1	Circle with radius 4.0
2	Circle with radius 0.0
3	Circle with radius 6.5

The `add(index, object)` method inserts an object into the list at specific index. All objects after it move over to accommodate it.

The `size` Method

The **size()** method returns the number of elements in the `ArrayList`. Notice that this is different from the **length** field of the array, which tells you how many slots were set aside and not the actual number of valid items stored in the array.

```
line 5: int howManyCircles = myCircles.size();    // howManyCircles is 4
```

`IndexOutOfBoundsException`

As in the `1D` array and the `2D` array, you will get an error if you try to access objects outside of the range of the list. The error when doing this with an `ArrayList` is called the `IndexOutOfBoundsException`.

```
myCircles.add(88, new Circle());          //  IndexOutOfBoundsException
```

The index, 88, is not in the range of 0 ≤ index ≤ myCircle.size().

The remove Method

The **remove(int index)** method deletes the object from the list that is at index. Each of the objects after this shifts down one index. The method also returns a reference to the object that was removed from the list.

```
line 6: Circle someCircle = myCircles.remove(2); // someCircle's radius is 0.0
```

After line 6 is executed: the Circle object that used to be in the slot at index 2 is removed (the circle with a radius of 0.0). The Circle objects that were positioned after it all move down one slot and someCircle now points to the Circle object that was removed.

index	myCircles
0	Circle with radius 8.0
1	Circle with radius 4.0
2	Circle with radius 6.5

The remove(2) method deleted the object that was at index 2. It returns a reference to the object that used to be at index 2.

The get Method

The **get(int index)** method returns the object that is located in the ArrayList at position index. It doesn't remove the object; it just returns a copy of the object reference. This way, an alias is created (more than one object reference pointing to the same object). The value of index must be greater than or equal to zero and less than the size of the ArrayList.

Example

Get the Circle object at index 2 and assign it to a different Circle reference variable:

```
line 7: Circle someCircle = myCircles.get(2);    // someCircle's radius is 6.5
```

The set Method

The **set(int index, E object)** method replaces the object in the ArrayList at position index with object. It returns a reference to the object that was previously at index.

Example

Replace the Circle object at index 0 with a new Circle object with a radius of 20:

```
line 8: Circle c = myCircles.set(0, new Circle(20)); // c's radius is 8.0
```

index	myCircles
0	Circle with radius 20.0
1	Circle with radius 4.0
2	Circle with radius 6.5

The set(0, Circle(20)) call replaces the object at index 0. It returns a reference to the object that was previously at index 0.

You will receive a Java Quick Reference sheet to use on the Multiple-choice and Free-response sections which lists the ArrayList class methods that may be included on the exam. Make sure you are familiar with it *before* you take the exam.

length Versus size() Versus length()

To find the number of slots in an array, use the length field.
To find the number of objects in an ArrayList, use the size() method.
To find the number of characters in a string, use the length() method.

Traversing an `ArrayList` Using a `for` Loop

Traversing is a way of accessing each element in an array or `ArrayList` through the use of iteration. The following code uses a `for` loop to print the area of each of the `Circle` objects in the `ArrayList`. The `get` method is used to retrieve each `Circle` object and then the `getArea` method is used on each of these `Circle` objects.

```
// print the area of each circle in the ArrayList using a for loop
for (int i = 0; i < myCircles.size(); i++)
{
   Circle temp = myCircles.get(i);
   System.out.println(temp.getArea());            // print each area
}
```

```
OUTPUT
1256.636
50.265
132.732
```

Traversing an `ArrayList` Using the Enhanced `for` Loop

The enhanced `for` loop (`for-each` loop) can be used with an `ArrayList`. The following code prints the area of each of the `Circle` objects in the `ArrayList`. Notice that the temporary variable, element, is the same data type as the objects in the `ArrayList`. In contrast to the general `for` loop shown above, the enhanced `for` loop does not use a loop control variable. Therefore, there is *no need for the get(i)* method since the variable `element` is a copy of each of the `Circle` objects, one at a time and the `getArea` method can be used directly with circle.

```
// print the area of each circle in the ArrayList using a for-each loop
for (Circle element : myCircles)
{
   System.out.println(element.getArea()); // no index used in for-each loop
}
```

```
OUTPUT
1256.636
50.265
132.732
```

Printing the Contents of an `ArrayList`

Unlike an array, the contents of an `ArrayList` can be displayed to the console by printing the reference variable. The contents are enclosed in brackets and separated by commas.

```
ArrayList<Double> myDoubles = new ArrayList<Double>();
myDoubles.add(23.5);
myDoubles.add(50.1);
myDoubles.add(7.5);
System.out.println(myDoubles);    // print the contents of the ArrayList
```

```
OUTPUT
[23.5, 50.1, 7.5]
```

Note: If you want to format the output when printing the contents of an `ArrayList`, you should traverse the `ArrayList`.

Do Not Use `remove()` in the Enhanced `for` Loop

Deleting elements during a traversal requires special techniques to avoid skipping elements. Never use the enhanced `for` loop (`for-each`) loop to remove an item from an `ArrayList`. You need the index to perform this process and the `for-each` loop doesn't use one. However, if you have to perform a remove, use an index variable with a `while` loop. Increment the index to get to the next element in the `ArrayList`. But, if you perform a remove, then don't increment the index.

```
int index = 0;
while (index < myCircles.size())
{
   if (myCircles.get(index).getRadius() <= 0)
      myCircles.remove(index);     // remove any circle whose radius <= 0
   else
      index++;          // only increment index when not removing a circle
}
```

Avoid Run-time Exceptions

Since the indices for an `ArrayList` start at 0 and end at the number of elements - 1, accessing an index value outside of this range will result in an `ArrayIndexOutOfBoundsException` being thrown.

Changing the size of an `ArrayList` while traversing it using an enhanced `for` loop can result in a `ConcurrentModifcationException` being thrown. Therefore, when using an enhanced `for` loop to traverse an `ArrayList`, you should not add or remove elements.

You will receive a Java Quick Reference sheet to use on the Multiple-choice and Free-response sections, which lists the `String` class methods that may be included in the exam. Make sure you are familiar with it before you take the exam.

ArrayList Class	
`int size()`	Returns the number of elements in the list
`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 adds 1 to size
`E get(int index)`	Returns the element at position `index` in the list
`E set(int index, E obj)`	Replaces the element at position `index` with `obj`; returns the element formerly at position `index`
`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 subtracts 1 from size; returns the elements formerly at position `index`

array vs ArrayList

Both an array and an ArrayList hold collections of data, so you just have to choose which one you want to use in your code (it's great to have choices, am I right?). Both have some limitations. The size of an array needs to be established when declaring and cannot be changed, whereas an ArrayList grows and shrinks as needed. An ArrayList can only have elements that are objects (such as String, Circle, Integer) but an array can have either primitive (int, double, boolean) or objects. To add or remove elements from an ArrayList a method needs to be called, but with an array you have to write (in most cases) a loop to shift all of the elements to accommodate the change.

Here is a summary of the differences between an array and an ArrayList.

Task	array	ArrayList
Declaring	`int [] arr = new int[100];`	`ArrayList<Integer> list = new ArrayList <Integer>();`
Length needs to be defined when created?	yes	no
Type of elements	primitive or reference	reference (objects)
Number of elements	`arr.length`	`list.size()`
Size can change?	no	yes
Accessing an element	`int value = arr[5];`	`int value = list.get(5);`
Assigning an element	`arr[5] = 16;`	`list.add(5,16);` or `list.add(16);`
Change a value	`arr[5] = 21;`	`list.set(5,21);`
Remove an element	user must write code	`list.remove(3);`

The Accumulate Advanced Algorithm

You have been hired by the high school baseball team to write a program that calculates statistics for the players on the team. You decide to have a class called `BaseballPlayer`. Every `BaseballPlayer` has a name, a numberOfHits, and a numberOfAtBats. `BaseballRunner` has an array of `BaseballPlayer` objects called roster. Your goal is to find the team batting average.

```
public class BaseballPlayer
{
   private String name;
   private int hits;
   private int atBats;
   public BaseballPlayer(String name, int hits, int atBats)
   {
    this.name = name;
    this.hits = hits;
    this.atBats = atBats;
   }
   public String getName()
   {   return name;   }

   public int getHits()
   {   return hits;   }

   public int getAtBats()
   {   return atBats;   }

   public double getBattingAverage()
   {    return (double) atBats / hits;   }

   /*  Additional implementation not shown   */
}
```

General Problem: Given a roster of baseball players, find the team's batting average.

Refined Problem: You are given a list of baseball players. Every baseball player knows his number of hits and number of at-bats. The team average is found by first computing the sum of the at-bats and the sum of the hits and then dividing the total hits by the total at-bats. Write a method that computes the team batting average.

Final Problem: Write a method called `findTeamAverage` that has one parameter: a `BaseballPlayer` array. The method should return a double that is the team's batting average. Compute the team average by dividing the total hits by the total at-bats. Make sure a player exists before processing him. Also, perform a check against zero before computing the team average. If the team total at-bats is 0, return a team batting average of 0.0. Finally, adapt your algorithm to work with an `ArrayList` of `BaseballPlayer` objects.

Algorithm:

Step 1: Create a variable called totalHits and set it equal to 0
Step 2: Create a variable called totalAtBats and set it equal to 0
Step 3: Look at each player on the roster
Step 4: Get the number of hits for the player and add it to totalHits
Step 5: Get the number of atBats for the player and add it to totalAtBats
Step 6: Continue until you reach the end of the list
Step 7: Compute the teamBattingAverage by dividing the totalHits by the totalAtBats
Step 8: Return the teamBattingAverage

Pseudocode:

```
set totalHits = 0
set totalAtBats = 0
for (iterate through all the players in the roster)
{
      totalHits = totalHits + player's hits
      totalAtBats = totalAtBats + player's at-bats
}
if (totalAtBats = 0)
{
   return 0
}
else
{
   return totalHits / totalAtBats
}
```

Java Code 1: Java code using an `ArrayList` (`for` loop)

```
public static double findTeamAverage(ArrayList<BaseballPlayer> arr)
{
    int totalHits = 0;
    int totalAtBats = 0;

    for (int i = 0; i < arr.size(); i++)
    {
      totalHits += arr.get(i).getHits();
      totalAtBats += arr.get(i).getAtBats();
    }

    if (totalAtBats == 0)
       return 0;
    else
       return (double)totalHits / totalAtBats;
}
```

Java Code 2: Java code using an `ArrayList` (`for-each` loop)

```
public static double findTeamAverage(ArrayList<BaseballPlayer> arr)
{
    int totalHits = 0;
    int totalAtBats = 0;

    for (BaseballPlayer player : arr)
    {
      totalHits += player.getHits();
      totalAtBats += player.getAtBats();
    }

    if (totalAtBats == 0)
       return 0;
    else
       return (double)totalHits / totalAtBats;
}
```

```
import java.util.ArrayList;
public class BaseballRunner
{
   public static void main(String[] args)
   {
ArrayList <BaseballPlayer> roster = new ArrayList <BaseballPlayer> ( );

      roster.add(new BaseballPlayer("Doug", 2, 4));
      roster.add(new BaseballPlayer("Sam", 3, 4));
      roster.add(new BaseballPlayer("Chase", 2, 4));
      roster.add(new BaseballPlayer("Brody", 3, 4));
      roster.add(new BaseballPlayer("Tom", 1, 3));
      roster.add(new BaseballPlayer("Triston", 2, 3));
      roster.add(new BaseballPlayer("Larry", 1, 3));
      roster.add(new BaseballPlayer("Charlie", 1, 3));
      roster.add(new BaseballPlayer("Chaser", 2, 3));

      System.out.println("Team average:  " + findTeamAverage(roster));
   }

   public static double findTeamAverage (ArrayList <BaseballPlayer> arr)
   {
      /* implementation is either Code 1 or Code 2 described above */
   }
}
```

```
OUTPUT
Team average:  0.5483870967741935
```

The Find-Highest Advanced Algorithm

General Problem: Given a roster of baseball players, find the name of the player that has the highest batting average.

Refined Problem: You are given a list of baseball players. Every baseball player has a name and knows how to calculate his batting average. Write a method that gets the batting average for every player in the list, and find the name of the player that has the highest batting average.

Final Problem: Write a method called findBestPlayer that has one parameter: a `BaseballPlayer` array. The method should return a string that is the name of the player that has the highest batting average. Make sure a player exists before processing him. If two players have the same high average, only select the first player. Also, adapt your algorithm to work with an ArrayList of BaseballPlayer objects.

Note: We will use the BaseballPlayer class from the previous example.

Algorithm:

Step 1: Create a variable called highestAverage and assign it a really small number
Step 2: Create a variable called bestPlayer and assign it the empty string
Step 3: Look at each of the players in the roster
Step 4: If the batting average of the player is greater than highestAverage, then set the highestAverage to be that player's average and set bestPlayer to be the name of the player
Step 5: Continue until you reach the end of the list
Step 6: Return the name of bestPlayer

Pseudocode:

```
set highestAverage = the smallest available number in Java
set bestPlayer = ""
for (iterate through all the players in the list)
{
   if (current player's batting average > highestAverage)
   {
      set highestAverage = current player's batting average
      set bestPlayer = name of current player
   }
}
return bestPlayer
```

Java Code 1: Returning one player with highest score

```
public static String findBestPlayer(ArrayList <BaseballPlayer>
players)
{
  double highestAverage = Integer.MIN_VALUE;
  double battingAverage;
  String bestPlayer = "";
  for (BaseballPlayer player : players)
  {
        battingAverage = player.getBattingAverage();
        if (battingAverage > highestAverage)
        {
              highestAverage = battingAverage;
              bestPlayer = player.getName();
        }

  }
  return bestPlayer;
}
```

Java Code 2: Returns each player with highest score

```
public static String findBestPlayer(ArrayList <BaseballPlayer> players)
{
   double highestAverage = Integer.MIN_VALUE;
   double battingAverage;
   String bestPlayer = "";
   for (BaseballPlayer player : players)
   {
         battingAverage = player.getBattingAverage();
         if (battingAverage > highestAverage)
         {
               highestAverage = battingAverage;
         }
   }
   for (BaseballPlayer player : players)
   {
         battingAverage = player.getBattingAverage();
         if (battingAverage == highestAverage)
         {
               bestPlayer += player.getName() + " ";
         }
   }
   return bestPlayer;
}
```

Example: Runner Program

```
import java.util.ArrayList;
public class BaseballRunner
{
   public static void main(String[] args)
   {
       ArrayList <BaseballPlayer> roster = new ArrayList <BaseballPlayer> ( );

       roster.add(new BaseballPlayer("Doug", 2, 4));
       roster.add(new BaseballPlayer("Sam", 3, 4));
       roster.add(new BaseballPlayer("Chase", 2, 4));
       roster.add(new BaseballPlayer("Brody", 3, 4));
       roster.add(new BaseballPlayer("Tom", 1, 3));
       roster.add(new BaseballPlayer("Triston", 2, 3));
       roster.add(new BaseballPlayer("Larry", 1, 3));
       roster.add(new BaseballPlayer("Charlie", 1, 3));
       roster.add(new BaseballPlayer("Chaser", 2, 3));

       System.out.println("Best Player:  " + findBestPlayer(roster));
   }

public static String findBestPlayer(ArrayList <BaseballPlayer> players)
   {
       /* implementation is either Code 1 or Code 2 described above  */
   }
}
```

```
OUTPUT Code 1
Best Player: Sam

OUTPUT Code 2
Best Player: Sam  Brody
```

The Twitter-Sentiment-Analysis Advanced Algorithm

Twitter is very popular and Twitter Sentiment Analysis has grown in popularity. The idea behind Twitter Sentiment is to pull emotions and/or draw conclusions from the tweets. A large collection of tweets can be used to make general conclusions of how people feel about something. The important words in the tweet are analyzed against a library of words to give a tweet a sentiment score.

One of the first steps in processing a tweet is to remove all of the words that don't add any real value to the tweet. These words are called **stop words,** and this step is typically part of a phase called **preprocessing**. For this problem, your job is to remove all the stop words from the tweet. There are many different ways to find meaningless words, but for our example, the stop words will be all words that are three characters long or less.

General Problem: Given a tweet, remove all of the meaningless words.

Refined Problem: You are given a tweet as a list of words. Find all of the words in the list that are greater than three characters and add them to a new list. Leave the original tweet unchanged. The new list will contain all the words that are not stop words.

Final Problem: Write a method called `removeStopWords` that has one parameter: an `ArrayList` of Strings. The method should return a new `ArrayList` of Strings that contains only the words from the original list that are greater than three characters long. Do not modify the original `ArrayList`. Also, modify the code to work on a `String` array.

Solution 1: Using an `ArrayList`

Algorithm 1:
Step 1: Create a list called longWords
Step 2: Look at each word in the original list
Step 3: If the word is greater than three characters, add that word to longWords
Step 4: Continue until you reach the end of the original list
Step 5: Return longWords

Pseudocode 1:

```
create a list called longWords
for (every word in the original list)
{
    if (word length > 3)
    {
        add the word to longWords
    }
}
return longWords
```

Java Code 1: Using an `ArrayList` (`for-each` loop)

```
public static ArrayList<String> removeStopWords(ArrayList<String> words)
{
    ArrayList<String> longWords = new ArrayList<String>();
    for (String word : words)
    {
        if (word.length() > 3)
        {
            longWords.add(word);
        }
    }
    return longWords;
}
```

Solution 2: Using an Array

This solution requires more work than the `ArrayList` solution. Since we don't know how big to make the array, we need to count how many words are greater than three characters before creating it.

Algorithm 2:

Step 1: Create a counter and set it to zero
Step 2: Look at each word in the original list
Step 3: If the length of the word is greater than three characters, add one to the counter
Step 4: Continue until you reach the end of the list
Step 5: Create an array called longWords that has a length of counter
Step 6: Create a variable that will be used as an index for longWords and set it to zero
Step 7: Look at each word in the original list
Step 8: If the length of the word is greater than three characters, add the word to longWords and also add one to the index that is used for longWords
Step 9: Continue until you reach the end of the list
Step 10: Return the longWords array

Pseudocode 2:

```
set counter = 0
for (iterate through every word in the original list)
{
    if (word length > 3)
    {
        counter = counter + 1
    }
}
create an array called longWords whose length is the same as counter
set index = 0
for (iterate through every word in the original list)
{
    if (word length > 3)
    {
        add the word to longWords using index
        set index = index + 1
    }
}
return longWords
```

Java Code 2: Using an array (using a `for-each` loop and a `for` loop)

```
public static String[] removeStopWords(String[] words)
{
   int count = 0;
   for (String word : words)
   {
      if (word.length() > 3)
      {
         count++;
      }
   }
   String[] longWords = new String[count];
   index = 0;
   for (int i = 0; i < words.length; i++)
   {
      if (words[i].length() > 3)
      {
         longWords[index] = words[i];
         index++;
      }
   }
   return longWords;
}
```

Example

Using an `ArrayList`:

```
public class TweetSentimentRunner
{
   public static void main(String[] args)
   {
      ArrayList<String> tweet = new ArrayList<String>();

      tweet.add("If");
      tweet.add("only");
      tweet.add("Bradley's");
      tweet.add("arm");
      tweet.add("was");
      tweet.add("longer");
      tweet.add("best");
      tweet.add("photo");
      tweet.add("ever");

      ArrayList<String> processedTweet = removeStopWords(tweet);

      System.out.println(processedTweet);      // print the ArrayList
   }

   public static ArrayList<String> removeStopWords(ArrayList<String> arr)
   {
      /*  implementation is described above Java code 1 */
   }
}
```

```
OUTPUT
[only, Bradley's, longer, best, photo, ever]
```

The Sequential (or Linear) Search Algorithm

When you search iTunes for a song, how does it actually find what you are looking for? It may use a sequential search to find the name of a song. A **sequential (or linear) search** is the process of looking at each of the elements in a list, one at a time, until you find what you are looking for. Sequential (or linear) search is one standard algorithm for searching. Another search algorithm, the binary search, will be discussed in Unit 10.

General Problem: Search for "Sweet Melissa" in a list of song titles.

Refined Problem: Write a method that allows you to search an array of song titles for a specific song title.

Final Problem: Write a method called `searchForTitle` that has two parameters: a string array and a string that represents the search target. The method should return the index of the target string if it is found and -1 if it does not find the target string. Note: I provide two solutions to solve this problem and compare their efficiency afterward.

The Search Target

When you search for a specific item in a list, the item you are looking for is often referred to as the search target. If the target is not found, a value of -1 is often returned.

Solution 1: Look at every item in the list.

Algorithm 1:

Step 1: Create a variable called `foundIndex` and set it equal to -1
Step 2: Look at each of the titles in the array one at a time
Step 3: Compare each title to the target title
Step 4: If the title is equal to the target title, then assign `foundIndex` to value of the current index
Step 5: Continue looking until you reach the end of the list
Step 6: Return the value of `foundIndex`

Pseudocode 1:

```
set foundIndex = -1
for (iterate through every element in the list)
{
  if (title = target title)
  {
    set foundIndex = current index
  }
}
return the value of foundIndex
```

Java Code

```
public static int searchForTitle(ArrayList <String> titles, String target)
{
   int foundIndex = -1;
   for (int i = 0; i < titles.size(); i++)
   {
      if (titles.get(i).equals(target))
      {
         foundIndex = i;                    // remember this index
      }
   }
   return foundIndex;                        // returns the foundIndex
}
```

Did you notice that the example used a for loop? That was on purpose. Remember, if you need to access the index of an element in an array or `ArrayList`, then a for-each loop cannot be used.

Solution 2: Stop looking if you find the search target.

Algorithm 2:

Step 1: Look at each of the titles in the array one at a time
Step 2: Compare the target title to each of the titles
Step 3: If the title is equal to the target title, then stop looking and return the current index
Step 4: Continue looking until you reach the end of the list
Step 5: Return -1

Pseudocode 2:

```
index = 0
while (not at the end of the list)
{
  if (title = target title)
  {
    return index
  }
  index++
}
return -1
```

Java Code

```
public static int searchForTitle(ArrayList <String> titles, String target)
{
    int index = 0;
    while (index < titles.size())
    {
        if (titles.get(index).equals(target))
        {
            return index;              // target was found
        }
        index ++;                      // target was not found yet
    }
    return -1;  // after searching entire list, the target was not found
}
```

Efficiency

Analyze the two solutions that searched for the title of a song.

The second algorithm is *more efficient* than the first because the method stops looking for the title as soon as it finds it. The first algorithm is *less efficient* than the second because it continues to compare each of the titles in the list against the search target even though it may have already found the search target.

Efficiency

An efficient algorithm does its job in the fastest way it can. Efficient programs are optimized to reduce CPU (central processing unit) time and memory usage.

Sorting Data

The idea of sorting is quite easy for humans to understand. In contrast, teaching a computer to sort items all by itself can be a challenge. Through the years, hundreds of sorting algorithms have been developed, and they have all been critiqued for their efficiency and memory usage. The AP Computer Science A Exam expects you to be able to read and analyze code that uses three main sorts: the Insertion Sort, the Selection Sort, and the Merge Sort (to be discussed in Unit 10).

The most important thing to remember is different sorting algorithms are good for different things. Some are easy to code, some use the CPU efficiently, some use memory efficiently, some are good at adding an element to a pre-sorted list, some don't care whether the list is pre-sorted or not, and so on. Of our three algorithms, Merge Sort is the fastest, but it uses the most memory.

The Swap Algorithm

Recall the swapping algorithm that you learned in Unit 6: Array. It is used in some of the sorting algorithms in this concept.

Sorting Algorithms on the AP Computer Science A Exam

You will not have to write the full code for any of the sorting routines described in this unit in the Free-Response Section. You should, however, understand how each works because they might appear in the Multiple-Choice Section.

Insertion Sort

The **Insertion Sort** algorithm is similar to the natural way that people sort a list of numbers if they are given the numbers one at a time. For example, if you gave a person a series of number cards one at a time, the person could easily sort the numbers "on the fly" by inserting the card where it belonged as soon as the card was received. Insertion Sort is considered to be a relatively simple sort and works best on very small data sets.

To write the code that can automate this process on a list of numbers requires an algorithm that does a lot of comparing. Let's do an example of how Insertion Sort sorts a list of numbers from smallest to largest.

Insertion Sort	1st number	2nd number	3rd number	4th number	5th number	6th number
Original list	67	23	12	54	35	18
After 1st pass	**23**	**67**	12	54	35	18
After 2nd pass	**12**	**23**	**67**	54	35	18
After 3rd pass	**12**	**23**	**54**	**67**	35	18
After 4th pass	**12**	**23**	**35**	**54**	**67**	18
After 5th pass	**12**	**18**	**23**	**35**	**54**	**67**

This algorithm uses a temporary variable that I will call `temp`. The first step is to put the number that is in the second position into the temporary variable (`temp`). Compare `temp` with the number in the first position. If `temp` is less than the number in the first position, then move the number that is in the first position to the second position and put `temp` in the first position. Now, the first pass is completed and the first two numbers are sorted from smallest to largest.

Next, put the number in the third position in `temp`. Compare `temp` to the number in the second position in the list. If `temp` is less than the second number, move the second number into the third position and compare `temp` to the number in the first position. If `temp` is less than the number in the first position, move the number in the first position to the second position, and move `temp` to the first position. Now, the second pass is completed and the first three numbers in the list are sorted. Continue this process until the last number is compared against all of the other numbers and inserted where it belongs.

I'll Pass

A **pass** in programming is one iteration of a process. Each of these sorts makes a series of passes before it completes the sort. An efficient algorithm uses the smallest number of passes that it can.

Implementation

The following class contains a method that sorts an array of integers from smallest to greatest using the Insertion Sort algorithm. Note, an `ArrayList` could have also been used. Also important to note is that any type could be stored in an array/ArrayList. The examples that follow use the int type.

```
public class InsertionSort
{
   public static void main(String[] args)
   {
      int[] myArray = {67, 23, 12, 54, 35, 18};
      insertionSort(myArray);
      for (int i : myArray)
      {
         System.out.print(i + "\t");
      }
   }

   /**
    * This method sorts an int array using Insertion Sort.
    *
    * @param element the array containing the items to be sorted
    *
    * Postcondition:  arr contains the original elements and
    *                 elements are sorted in ascending order
    */
   public static void insertionSort(int[] arr)
   {
      for (int j = 1; j < arr.length; j++)
      {
         int temp = arr[j];
         int index = j;
         while (index > 0 && temp < arr[index - 1])
         {
            arr[index] = arr[index - 1];
            index--;
         }
         arr[index] = temp;
      }
   }
}
```

```
OUTPUT
12    18    23    35    54    67
```

Selection Sort

The **Selection Sort** algorithm forms a sorted list by repeatedly finding and selecting the smallest item in a list and putting it in its proper place.

Selection Sort	1st number	2nd number	3rd number	4th number	5th number	6th number
Original list	67	23	12	54	35	18
After 1st pass	**12**	23	67	54	35	18
After 2nd pass	**12**	**18**	67	54	35	23
After 3rd pass	**12**	**18**	**23**	54	35	67
After 4th pass	**12**	**18**	**23**	**35**	54	67
After 5th pass	**12**	**18**	**23**	**35**	**54**	67
After 6th pass	**12**	**18**	**23**	**35**	**54**	**67**

To sort a list of numbers from smallest to largest using Selection Sort, search the entire list for the smallest item, select it, and swap it with the first item in the list (the two numbers change positions). This completes the first pass. Next, search for the smallest item in the remaining list (not including the first item), select it, and swap it with the item in the second position. This completes the second pass. Then, search for the smallest item in the remaining list (not including the first or second items), select it, and swap it with the item in the third position. Repeat this process until the last item in the list becomes (automatically) the largest item in the list.

Implementation

The following class contains a method that sorts an array of integers from smallest to greatest using the Selection Sort algorithm. Note, an `ArrayList` could have also been used. Also important to note is that any type could be stored in an array/ArrayList. The examples that follow use the int type.

```
public class SelectionSort
{
   public static void main(String[] args)
   {
      int[] myArray = {67, 23, 12, 54, 35, 18};
      selectionSort(myArray);
      for (int i : myArray)
      {
         System.out.print(i + "\t");
      }
   }
  /**
    * This method sorts an int array using Selection Sort.
    *
    * @param arr  the array to be sorted
    *
    * Postcondition:  arr contains the original elements and
    *                 elements are sorted in ascending order
    */
   public static void selectionSort(int[] arr)
   {
      for (int j = 0; j < arr.length - 1; j++)
      {
         int index = j; // index of smallest element
         for (int k = j + 1; k < arr.length; k++)
         {
             if (arr[k] < arr[index]) // current element is smaller
             index = k;               // the new smallest index

         }

        // swap the two elements
         int temp = arr[j];
         arr[j] = arr[index];
         arr[index] = temp;
      }
   }
}
```

```
OUTPUT
12    18    23    35    54    67
```

› Rapid Review

The `ArrayList`

- The `ArrayList` is a complex data structure that can store a list of objects.
- The two general forms for declaring an `ArrayList` are:

```
ArrayList <ClassName> generic = new ArrayList <ClassName> ( );
ArrayList raw = new ArrayList ( );
```

- An `ArrayList` can only hold a list of objects; it cannot hold a list of primitive data.
- Use the `Integer` or `Double` classes to make an `ArrayList` of `int` or `double` values.

- The initial size of an `ArrayList` is 0.
- The `add(E object)` method appends the object to the end of the `ArrayList`. It also returns true.
- To append means to add on to the end of a list.
- The `add(int index, E object)` method inserts object at position index (Note: index must be in the interval: [0,size]). As a result of this insertion, the elements at position index and higher move 1 index farther from the 0 index.
- The `get(int index)` method returns a reference to the object that is at index.
- The `set(int index, E object)` method replaces the element at position index with object and returns the element that was formerly at index.
- The `remove(int index)` method removes the element at position index, and subsequently subtracts one from the indices of the elements at positions index + 1 and greater. It also returns the element that was removed.
- The `size()` method returns an int that represents the number of elements that are currently in the `ArrayList`.
- The size of the `ArrayList` grows and shrinks by either adding or removing elements.
- The `size()` method adjusts accordingly as elements are added or removed.
- Using an index that is not in the range of the `ArrayList` will throw an `IndexOutOfBoundsException`.
- The `ArrayList` requires an import statement since it is not in the standard library, but this will never be required on the AP Computer Science A exam.

Traversing an `ArrayList`

- To traverse an `ArrayList` means to visit each of the elements in the list.
- There are many ways to traverse an `ArrayList`, but the most common way is to start at the beginning and work toward the end.
- If a `for` loop is used to traverse an `ArrayList`, then the `get()` method will be required to gain access to each of the elements in the `ArrayList`.
- If an enhanced `for` loop is used to traverse an `ArrayList`, then the `get()` method is not required to gain access to each of the elements in the `ArrayList`, since each object is automatically retrieved by the loop, one at a time.
- If you need to remove elements from an `ArrayList`, you may consider starting at the end of the list and working toward the beginning.

Sequential (or Linear) Search Algorithm

- The sequential search algorithm searches a list to find a search target.
- It looks at each element in the list one at a time comparing the element to the search target.
- If the element is found, the index is returned.
- If the element is not found, -1 is usually returned.

Insertion Sort

- Insertion Sort uses an algorithm that repeatedly compares the next number in the list to the previously sorted numbers in the list and inserts it where it belongs.

Selection Sort

- Selection Sort uses an algorithm that repeatedly selects the smallest number in a list and swaps it with the current element in the list.

› Review Questions

Basic Level

1. Assume that cities is an `ArrayList<String>` that has been correctly constructed and populated with the following items.

```
["Oakland", "Chicago", "Milwaukee", "Seattle", "Denver", "Boston"]
```

Consider the following code segment.

```
cities.remove(2);
cities.add("Detroit");
cities.remove(4);
cities.add("Cleveland");
```

What items does `cities` contain after executing the code segment?

(A) `["Cleveland", "Detroit", "Oakland", "Chicago", "Seattle", "Denver", "Boston"]`
(B) `["Oakland", "Milwaukee", "Seattle", "Boston", "Detroit", "Cleveland"]`
(C) `["Oakland", "Chicago", "Seattle", "Boston", "Detroit", "Cleveland"]`
(D) `["Oakland", "Milwaukee", "Denver", "Boston", "Detroit", "Cleveland"]`
(E) `["Oakland", "Chicago", "Seattle", "Denver", "Detroit", "Cleveland"]`

2. Consider the following code segment.

```
ArrayList<String> subjects = new ArrayList<String>();
subjects.add("French");
subjects.add("History");
subjects.set(1, "English");
subjects.add("Art");
subjects.remove(1);
subjects.set(2, "Math");
subjects.add("Biology");
System.out.println(subjects);
```

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

(A) `[French, English, Math, Biology]`
(B) `[French, Art, Biology]`
(C) `[French, English, Art, Math, Biology]`
(D) `[French, Math, Biology]`
(E) `IndexOutOfBoundsException`

Questions 3-6 refer to the following information.

Array `arr` has been defined and initialized as follows

```
int[] arr = {5, 3, 8, 1, 6, 4, 2, 7};
```

3. Which of the following shows the elements of the array in the correct order after the first pass through the outer loop of the Insertion Sort algorithm?

(A) `1 5 3 8 6 4 2 7`
(B) `1 3 8 5 6 4 2 7`
(C) `5 3 7 1 6 4 2 8`
(D) `3 5 8 1 6 4 2 7`
(E) `1 2 3 4 5 6 7 8`

4. Which of the following shows the elements of the array in the correct order after the fourth pass through the outer loop of the Insertion Sort algorithm?

(A) 1 3 5 6 8 4 2 7
(B) 1 2 3 4 6 5 8 7
(C) 1 2 3 4 5 6 7 8
(D) 3 5 8 1 6 4 2 7
(E) 1 2 3 4 5 6 8 7

5. Which of the following shows the elements of the array in the correct order after the first pass through the outer loop of the Selection Sort algorithm?

(A) 1 2 3 5 6 4 8 7
(B) 1 3 8 5 6 4 2 7
(C) 1 3 5 8 6 4 2 7
(D) 1 5 3 8 6 4 2 7
(E) 5 3 1 6 4 2 7 8

6. Which of the following shows the elements of the array in the correct order after the fourth pass through the outer loop of the Selection Sort algorithm?

(A) 3 1 4 2 5 6 7 8
(B) 3 1 4 2 5 8 6 7
(C) 1 2 3 4 5 6 7 8
(D) 1 2 3 4 5 8 6 7
(E) 1 2 3 4 6 5 8 7

Advanced Level

7. Consider the following code segment.

```
int total = 3;
ArrayList<Integer> integerList = new ArrayList<Integer>();
for (int k = 7; k < 11; k++)
{
   integerList.add(k + 3);
}
for (Integer i : integerList)
{
   total += i;
   if (total % 2 == 1)
   {
      total -= 1;
   }
}
System.out.println(total);
```

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

(A) 34
(B) 37
(C) 46
(D) 47
(E) 49

8. Assume that `msg` is an `ArrayList<String>` that has been correctly constructed and populated with the following items.

```
["can", "i", "delete", "words", "starting", "with", "letters",
      "between", "n", "and", "z"]
```

Which code segment removes all `String` objects starting with a letter from the second half of the alphabet (n–z) from the `ArrayList`?

Precondition: all `String` objects will be lowercase

Postcondition: msg will contain only `String` objects from the first half of the alphabet (a–m)

I.
```
for (int i = 0; i < msg.size(); i++)
{
   if (msg.get(i).compareTo("n") >= 0)
   {
      msg.remove(i);
   }
}
```

II.
```
for (int i = msg.size() - 1; i >= 0; i--)
{
   if (msg.get(i).compareTo("n") >= 0)
   {
      msg.remove(i);
   }
}
```

III.
```
int i = 0;
while (i < msg.size())
{
   if (msg.get(i).compareTo("n") >= 0)
   {
      msg.remove(i);
   }
   else
   {
      i++;
   }
}
```

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

9. Consider the following code segment that implements the Insertion Sort algorithm.

```
public void insertionSort(int[] arr)
{
   for (int i = 1; i < arr.length; i++)
   {
      int key = arr[i];
      int j = i - 1;
      while (j >= 0 && /* condition */)
      {
         arr[j + 1] = arr[j];
         j--;
      }
      arr[j + 1] = key;
   }
}
```

Which of the following can be used to replace `/* condition */` so that `insertionSort` will work as intended?

(A) `arr[i] > key`
(B) `arr[j] > key`
(C) `arr[i + 1] > key`
(D) `arr[j + 1] > key`
(E) `arr[i - 1] > key`

10. Determine if the numbers in an `ArrayList` are always increasing.

Write a method that determines if the values in an `ArrayList` are always increasing. The method takes one parameter: an `ArrayList` of `Integer`. The method returns true if the elements in the array are continually increasing and false if the elements are not continually increasing.

```
ArrayList<Integer> listOfIntegers = /* contains values from table below  */
boolean result = isIncreasing(listOfIntegers);   // result is true
```

For example, if the ArrayList passed to the method is:

34	35	36	37	38	40	42	43	51	52

then the value true is returned

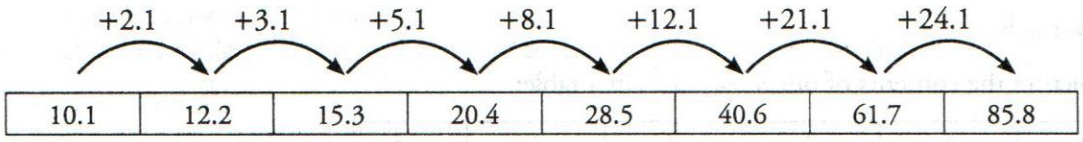
```
public boolean isIncreasing(ArrayList<Integer> arr)
{
   // Write the implementation
}
```

11. Determine if the rate is increasing.

If a stock price is increasing, that means the value of it is going up. If the *rate* that the stock price is increasing is increasing, it means that the price is rising at a faster rate than it was the day before. Write a method that determines if the rate at which a stock price increases is increasing. The method has one parameter: an `ArrayList` of `Double` values that represent the stock prices. Return true if the rate that the stock is increasing is increasing, and false if the rate is not increasing.

```
ArrayList<Double> prices = /* contains the values from the table below  */
boolean result = rateIsIncreasing(prices);   // result is true
```

For example, if stockPrices contains the following:



then the value true is returned

```
public boolean rateIsIncreasing(ArrayList<Double> stockPrices)
{
   // Write the implementation
}
```

12. Count the empty lockers.

A high school has a specific number of lockers so students can store their belongings. The office staff wants to know how many of the lockers are empty so they can assign these lockers to new students. Consider the following classes and complete the implementation for the `countEmptyLockers` method.

```
public class Locker
{
   private boolean inUse;

   public boolean isInUse()
   {   return inUse;   }

   public void setInUse(boolean isInUse)
   {   inUse = isInUse;   }
}

public class School
{
   private ArrayList <Locker> lockerlist;

   /*   Additional implementation not shown   */

   /**
    * This method counts the number of empty lockers in the list of lockers.
    * @param lockers the list of Locker objects
    * @return the number of empty lockers
    * PRECONDITION:  No object in the list lockers is null
    *                (every locker has a true/false value)
    */
   public static int countEmptyLockers(ArrayList <Locker> lockers)
   {
      /* to be implemented */
   }
}
```