Table of Contents

Unit 5 Notes

5.1 Anatomy of a Class

- Private: Restricts access to the class you're in
  Public: Allows access from other classes
- Classes should be public
  Instance variables should be private
  Constructors should be public
  Methods may be either public or private
- Blueprints -> Class
  Use of the class (instances) -> Objects
  Attributes of objects -> instance variables
  Behaviors of objects -> methods
- Accessor/mutator methods can allow client code to modify data

5.2 Constructors

- Public + name of class
- "Has-a": each instance of a class "has-a" private instance variable
- Default constructor: default values for instance variables
- Overloaded constructor: parameters imputed to set each instance variable's value
- Only one constructor used to set initial state of the instance variables
- Even if constructor only setting one value, still need to provide default value for other instance variables
- Parameters in constructor are local variables
- If no constructor provided, Java provides a default constructor: int - 0, double - 0.0, String and other objects- null

5.3 Documentation with Comments

- Describe functionality/use of code in a program through comments
- Only help people understand the code; ignored by the interpreters
- Can be used to test alternative code without losing it
- Syntax
  - // Single line comment
  - /* Multi line

        Comment */
    - /** Documentation
      * to create Javadoc
      */
- Precondition: condition that must be true before executing a part of the code
    - Comments for a method
    - Ex. The state of a parameter
- Postcondition: describe outcome of execution; state of an object or what is returned
    - Responsibility of programmer to ensure these are true; write code to ensure conditions will be true
    - Comments for the method
    - Things that must be true after the method is executed

5.4 Accessor Methods

- Allow safe access to instance variables
- get methods / getters
- Must be public, return type must match type of instance variable to be accessed
- No parameters
- Getter returns the value, must print the returned values yourself
- **toString method:**
    - Returns a string with info about instance variable
    - No parameters
    - When System.out.println(object) is called on an object, toString method is called

5.5 Mutator Methods

- Void methods don't return values
- A mutator method (modifier method, set method, setters) is a void method changing the values of instance or static variables
    - Necessary for another class to modify instance variables

        - Must be public visibility, void return, setNameOfVar, parameter matching type of instance variable being modified
        - Can be used if you used a default constructor initially but later got information to set the instance variables
        - Parameter is optional, depends on method content

5.6 Writing Methods

- Need to use public accessor method when trying to get value of instance variable
- If comparing two objects of same class, if we are inside the class, we can use a method like isSamBreed(Dog otherDog){ return breed.equals(otherDog.breed);
    - Here, we can use private instance variables (for both dogs) for comparison as we are inside the dog class
    - Need to use getters to find value of instance variables not inside class
    - Method header:
        - Acces level: set by access modifier; public or private
        - Ownership: set by whether or not static is included
        - Return type: data type of value returned by method, can be primitive, reference or void (no value returned)
        - Identifier: name of method
        - Parameter list: enclosed in parentheses, states data type and identifier for each parameter
    - Can call methods within other methods
    - Parameters can be primitive or reference
        - If the parameter is a reference, when a new object is created, the parameter will point to the same object as the formal parameter (alias). Need to be careful as this can end up mutating the object

5.7 Static Variables and Methods

- Define behaviors of a class through static methods
- Static methods are associated with the class, not its objects
    - Cannot access or change instance variables
    - Don't have "this" reference, and can't call non static methods
    - Static can only call other static variables/methods
- Static variables can be defined - belong to the class
    - All objects of the class share a single static variable
    - Static variables can be public or private
    - Used with class name and "." operator - not the object name as they're associated with the class itself, not its objects
    - Can mutate the variables directly in the class
- Should be used when no instance variables are

5.8 Scope and Access

- Instance variables: accessible everywhere in the class vs.
- local variables: parameters local to constructors and methods
    - Cannot be declared to public or private
- Method decomposition: breaking down a task into multiple methods, each with its own parameters

5.9 'This' Keyword

- 'This' references the current object (the one whose method/constructor is being called) within a non static method/constructor
- Can be used to pass the current object as a parameter in a method call
- In some situations, 'this' is optional (to make code easier to read)
- Required to differentiate between instance variables and local variables
- this.(instance variable name) = (value) will set the instance variable

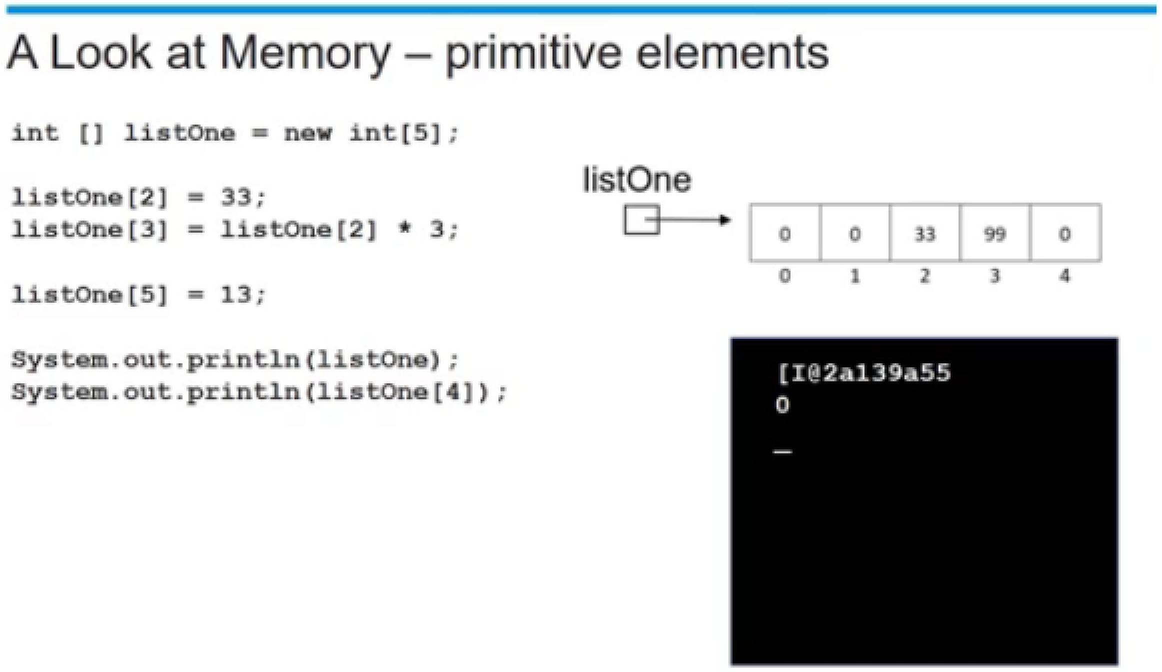
5.10 Ethical and Social Implications of Computing Systems

- Golden Rule: Do unto others as you would have them do unto you
    - Computer Professionals for Social Responsibility
    - Association for Computing Machinery
    - Institute of Electrical and Electronics Engineers

- Copyright: in trying to find a solution to a coding problem, we often seek answers via internet or reaching out to other programmers
    - Open source code does not mean we can use the code without giving credit
    - Credit any code that we use in programs that does not belong to us by using comments
- Impact of Software: At higher levels, code can have much greater impact on society, economies, and/or culture

Unit 6 Notes

6.1 Array Creation and Access

- Array: data structure use to implement a list of primitive or object reference data
- Element is a single value in the array, index is the position
- In Java, first element at index 0
- Length is the number of elements. Length is a public final data member (can be accessed in any class, and we cannot change an array's length)
- Last element at index list.length-1
- 

  A Look at Memory – primitive elements

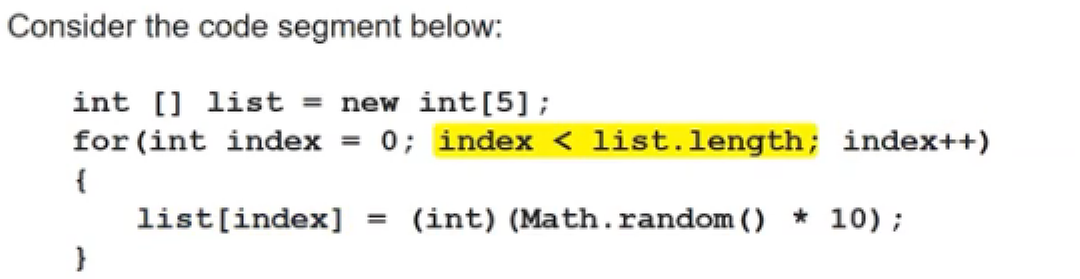
```
int [] listOne = new int[5];

listOne[2] = 33;
listOne[3] = listOne[2] * 3;

listOne[5] = 13;

System.out.println(listOne);
System.out.println(listOne[4]);
```

```
[I@2a139a55
0
_
```

- Here, listOne[5] will cause an ArrayIndexOutofBoundsException
- Elements initialized based on their type: int to 0, reference type to null, double to 0.0, boolean to false
- Initializer list: {x,y,z,...}
-

6.2 Traversing Arrays

- Since array indexed from 0 to number of elements-1, for loop can be used for traversal


Consider the code segment below:

```
int [] list = new int[5];
for(int index = 0; index < list.length; index++)
{
   list[index] = (int)(Math.random() * 10);
}
```

-
- Make sure to check initial value, condition, and increment in the loop headers to ensure each element of the array is accessed
-
- Traversal: Accessing each element in an array
    - Iteration can be used to traverse arrays
    - Access elements using their indices (remember arrays are zero-indexed)
- Can create a list using a function
- Off by one errors are common; will result in an array index out of bounds exception. Read through code to ensure this won't happen

6.3 Enhanced for Loop for Arrays

- Also called a for-each loop
- As opposed to a normal for loop, a for each loop has 2 parts separated by a colon
    - Type/name of variable : data structure
    - Ex. int number : values (Where values is an array with integers stored)
        - This will loop for each item in the array, with number being set to the current value for each iteration (without using index)
- Cannot access indices of array/subscript notation in the loop, but you can access the value stored in the variable
- Can be used with objects we create
- Changing the variable in the for each loop will not change the array

6.4 Develop Algorithms Using Arrays

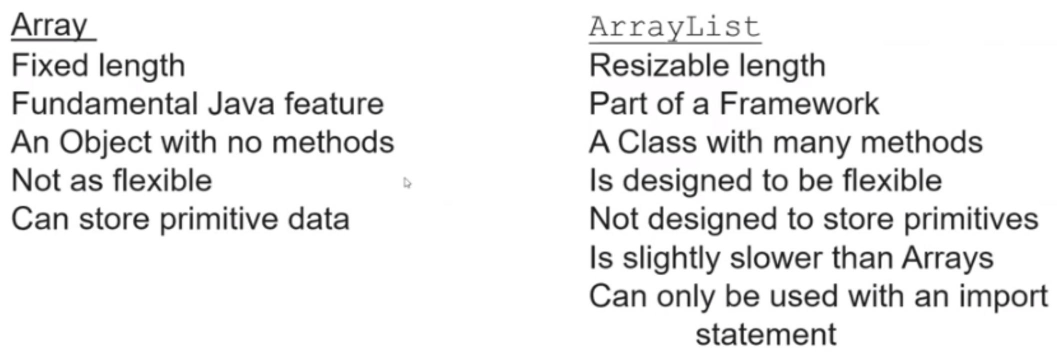
- Creating a minimum/maximum algorithm from an array
    - Goal is to identify the largest or smallest value in an array

        - Need a local variable to store the current min/max value to compare in each iteration. If the new variable is smaller/larger, update the var
- Finding the average of data members within an array
    - Goal is to find the average value within an array
    - Double var to store accumulated values
        - Must be declared before, not during the loop
    - For loop to traverse array and add current to total val
    - After counting values, divide by total values and return
- Shifting array values by a specified index
    - Contents of an array may need to be shifted to use the stored data
    - Make an empty array of the same size then iterate over the original array and copy values into the new array with the adjusted index
    - Assign the new array to the original value
    - Use a standard for loop (not enhanced) as the index is needed
    - Shifting to the left:
        - Outer loop will execute the number of shifts
        - Inner loop will save the first index value, move everything left, then put saved value at the end
    - Shifting to the right
        - Create new array, pull from current index, add shift number to index, and insert into new array

Unit 7 Notes

7.1 Introduction to ArrayList

- ArrayList is a collection of object data of the same type; store references to objects
    - Mutable and contains object references
        - Differences between Arrays and ArrayLists



Array	ArrayList
Fixed length	Resizable length
Fundamental Java feature	Part of a Framework
An Object with no methods	A Class with many methods
Not as flexible	Is designed to be flexible
Can store primitive data	Not designed to store primitives
	Is slightly slower than Arrays
	Can only be used with an import statement

        - ArrayList class is implemented using Arrays
    - ArrayList() constructs an empty list, ArrayList<E> specifies element type; reference parameters and return type
        - ```ArrayList<DataType> varName;``` declares a variable to reference an ArrayList object
        - ```new ArrayList<DataType>();``` instantiates an ArrayList object
            - DataType can be any **nonprimitive** data type
                - boolean, char, double, int
            - Wrapper classes can be used as a workaround; store primitive values as objects
                - Boolean, Character, Double, Integer
        - Ex. ```ArrayList<Integer> a1 = new ArrayList<Integer>();```
    - ArrayLists are part of the java.util package
        - ```import java.util.ArrayList;```
- Frameworks are prewritten code that can use objects of groups of data

7.2 ArrayList Methods

- int size() -- Returns the number of objects in the list
    - ```arrayListName.size()```
- boolean add(E obj) -- Append obj to end of list and returns true

    - ```arrayListName.add(obj)```
- void add(int index, E obj) -- Inserts obj at position index, moving elements at index and higher to the right and increasing the size by 1
    - ```arrayListName.add(index, obj)```
    - To add objects to an ArrayList, objects must be the same data type used to instantiate the list
- E get(int index) -- Returns element and the index specified
    - ```objectName varName = arrayListName.get(index)```
- E set(int index, E obj) -- Replaces element at specified index with obj and returns the former value
    - ```objectName varName = arrayListName.set(index, obj)```
- E remove(int index) -- Removes element at specified index, moving elements at index + 1 and higher to the left, decreases size by 1. Returns the removed value
    - ```objectName varName = arrayListName.remove(index)```
        - Removes specified element and saves to varName
    - ```arrayListName.remove(arrayListName.size() - 1)```
        - Removes last element in ArrayList
- ArrayList is a reference object; when passed as a parameter in a method they're passed as a reference to the address, not the copy of the value
    - If a method updates elements of an ArrayList, the elements will be updated in both the method and the ArrayList
- ArrayList<E> where E is the type of elements
    - Reference parameters and return type from methods are type E
    - ArrayList<E> allows compiler to find errors that would otherwise be found in runtime
    - When returning an ArrayList, it's best to specify the data type of the stored elements

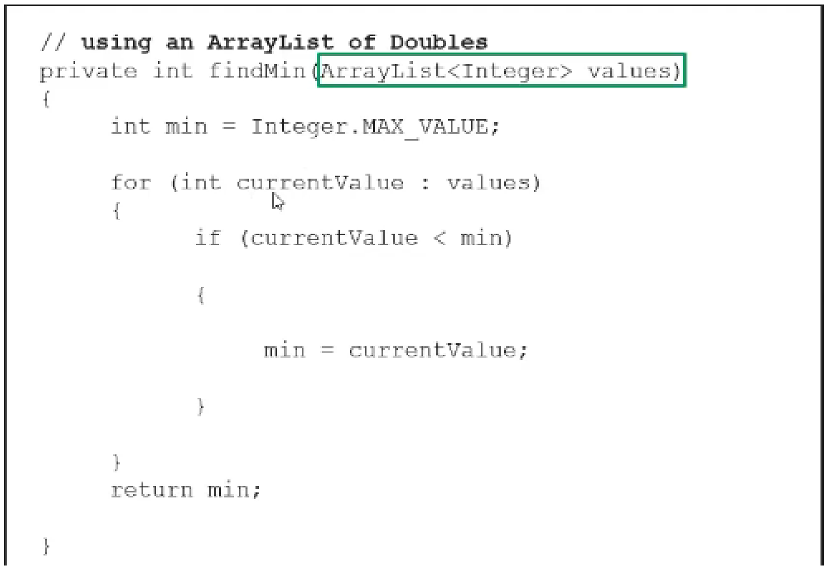
7.3 Traversing ArrayLists

- For and while loops may be used to traverse ArrayLists
    - Accesses all elements in the ArrayList
        - Use arrayList.get() method
    - If you remove an element from an ArrayList, the indices will be shifted.

        - To avoid skipping checking entries, you can start at the end and decrement for each iteration, as even if an element is removed, it'll still go through each element
    - Remember indices start at 0 and end at elements -1; be careful to avoid ArrayIndexOutOfBoundsException
        - You can start at 0 and iterate until the end of the ArrayList, increasing the index by 1 per iteration or start at the end of the ArrayList and iterate to 0, decreasing index by 1 per iteration
- Enhanced for loops may also be used to traverse ArrayLists
    - Can't change the size of an ArrayList within an enhanced for loop; will cause a ConcurrentModificationException
    - Iterates from first to last index, though indices aren't used explicitly; copies of the current element are made while iterating
        - This means that you can't change the ArrayList itself through an enhanced for loop
- Other common mistakes
    - Arrays are different from ArrayLists - must use different notation

7.4 Developing Algorithms Using ArrayList

- Compared to array algorithms, remember to use ArrayList methods (get, set, size, etc.)
- Ex: finding the minimum value in the array list



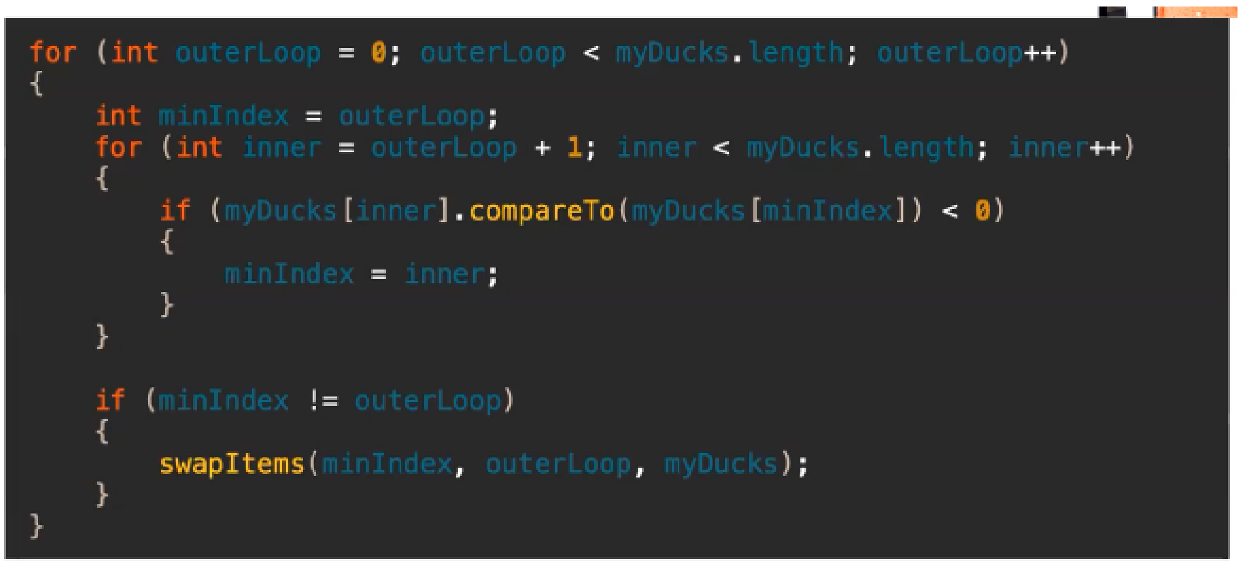
```
// using an ArrayList of Doubles
private int findMin(ArrayList<Integer> values)
{
     int min = Integer.MAX_VALUE;

     for (int currentValue : values)
     {
           if (currentValue < min)

           {

                 min = currentValue;

           }

     }
     return min;

}
```

7.5 Searching

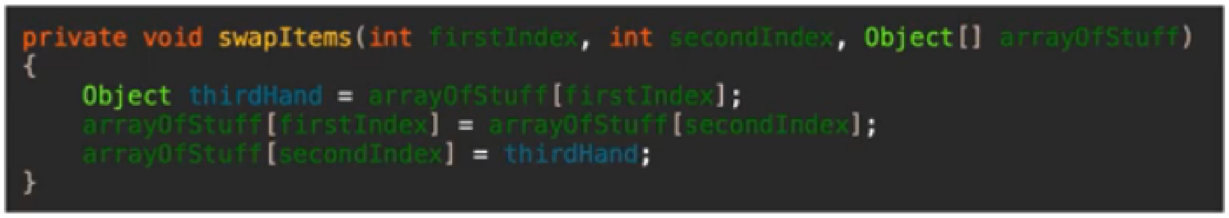
- Linear searching fits a for loop. We specify each element one at a time and don't need to track the index after execution
- We retrieve the value in every iteration and if it matches, we return the index; otherwise the loop keeps going
- Can be used for arrays and ArrayLists
- Linear search best when we do not have any idea about the order of the data
- Different data types require different comparisons (int: == ; double- do some math to check if the value is close by; Object instances should use .equals(otherThing) method to check for a match)
- Standard for loop with an if loop to test for matching condition
- Else condition: return -1. This means match found nowhere
- Don't use == when looking for an Object because that would only return true if the variable and the element at that index point to the same memory location
- Questions to ask:
    - Does order matter? When searching to remove a value, if we search forward, we have to make sure to adjust the loop control variable, or else some values may end up getting skipped because of the change in index. If we start from the back, the index will be adjusted appropriately. Or, you can include x-- within the if statement
    - Can we use an enhanced for loop? Can use if we are just checking for existence, because they do not track the index in the structure
    - Can we do this faster? We need to check EVERY index in order

7.6 Sorting

- Selection sort: identifies either max or min of compared values and iterates over the structure, checking if the element at the index matches the condition; if so, it will swap the value stored at that index and continue. Uses a helper method for the swap operation (since variables can only hold one value at a time)

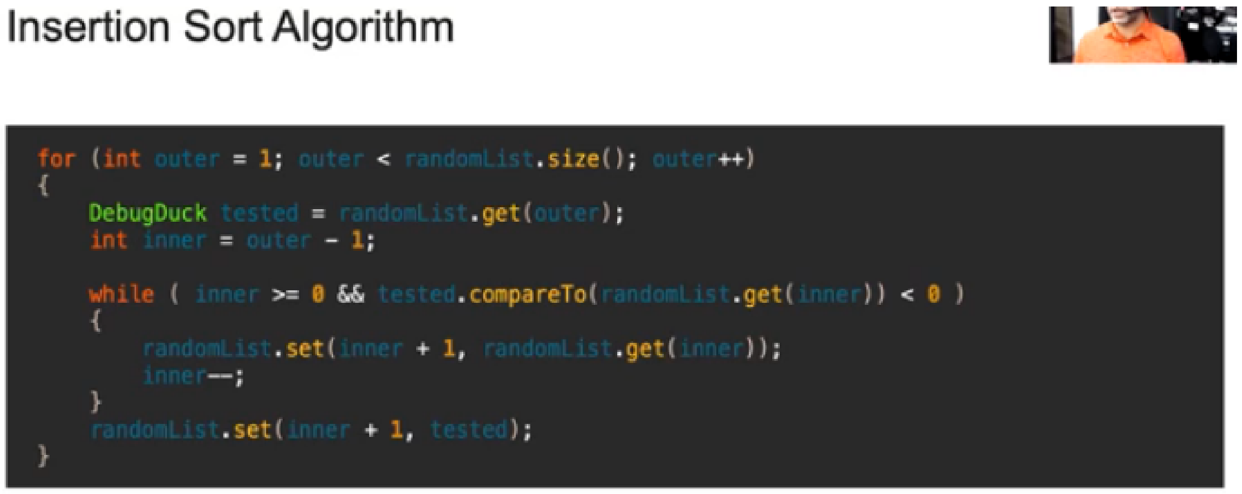
- 

```java
for (int outerLoop = 0; outerLoop < myDucks.length; outerLoop++)
{
    int minIndex = outerLoop;
    for (int inner = outerLoop + 1; inner < myDucks.length; inner++)
    {
        if (myDucks[inner].compareTo(myDucks[minIndex]) < 0)
        {
            minIndex = inner;
        }
    }

    if (minIndex != outerLoop)
    {
        swapItems(minIndex, outerLoop, myDucks);
    }
}
```

- 

```java
private void swapItems(int firstIndex, int secondIndex, Object[] arrayOfStuff)
{
    Object thirdHand = arrayOfStuff[firstIndex];
    arrayOfStuff[firstIndex] = arrayOfStuff[secondIndex];
    arrayOfStuff[secondIndex] = thirdHand;
}
```

- Swap function uses a temporary, third variable to hold on to the swapped value
- Can't use enhanced for loop as we need to know the index when swapping
- Insertion sort: builds sorted structure as it proceeds. Inserts each value it finds at the appropriate location. Accomplished by using while loop as the inner loop



Insertion Sort Algorithm

```java
for (int outer = 1; outer < randomList.size(); outer++)
{
    DebugDuck tested = randomList.get(outer);
    int inner = outer - 1;

    while ( inner >= 0 && tested.compareTo(randomList.get(inner)) < 0 )
    {
        randomList.set(inner + 1, randomList.get(inner));
        inner--;
    }
    randomList.set(inner + 1, tested);
}
```

- Counting steps: can compare algorithms' by their efficiency (how fast do they execute)

  - How often does each statement get executed?
  - Each of the basic sorting algorithms (selection and insertion) are quadratic functions since a loop over an entire array / ArrayList means that the inner and outer loops execute as many times as the number of elements to compare (even if one of the loops is optimized so it doesn't execute for all elements (count-n), algorithm still a function of count^2

7.7 Ethical Issues Around Data Collection

- Need to ensure we take steps to remove identifying information as part of writing an application
- Removing Data
- Minimizing info: don't use private information as an identifier. Never ask for confidential information. Use values that can at least partially anonymize what you are storing
- Object method hashCode() can be used to create an identifier

Unit 9 Notes

9.1 Creating Superclasses and Subclasses

- Inheritance
    - Subclasses inherit attributes/behaviors of superclasses
    - Allows for code to be reused, prevents repeating it
    - Easier to maintain - more organized and easier to read
- Class hierarchy developed by putting common attributes of related classes together into a superclass
    - Each subclass can only have one superclass
    - Subclasses extend superclasses, take from the attributes/behaviors of the superclass (inheritance)
    - Extending a subclass from a superclass creates an “is-a” relationship from the subclass to the superclass
        - A subclass ‘is a’ subdivision of the superclass
- Keyword `extend`
    - Establishes the inheritance relationship between the subclass and superclass
    - Syntax:

```
public class Subclass_Name extends Superclass_Name{}
```

9.2 Writing Constructors for Subclasses

- Constructors aren’t inherited
    - Java will call the no argument constructor if a subclass’s constructor doesn’t call the superclass’s constructor
    - Process of calling superclass constructors continues until Object constructor is called; all constructors in hierarchy execute then
- Keyword `super`
    - Superclass constructor can be called from the first line of a subclass constructor
        - Needs to pass appropriate parameters
        - Parameters passed in the call to superclass constructor provide values used to initialize instance variables

9.3 Overriding Methods

- Overriding Methods

    - Occurs when a public method in subclass has the same signature as a public method in the superclass
    - Used to write a new implementation of a method that already exists in the superclass
- Any method called must be defined in its own class or superclass
- Subclasses are usually designed to have modified or additional methods/instance variables compared to the superclass
- Subclasses inherit all public methods from the superclass, and they remain public
    - Important for accessing private instance variables from superclass

9.4 Super Keyword
- Keyword `super` can be used to call a superclass's constructors and methods
    - Allows things from superclass to be called in subclass
    - Even if a method is overridden, using the super keyword will allow the overridden method to be called from the superclass

9.5 Creating References Using Inheritance Hierarchies
- A reference variable is **polymorphic** when it can refer to objects from different classes at different points in the code
- Ex: WritingUtensil writer2 = new Marker(); because Marker "is-a" WritingUtensil
- Done so that different data types can be stored within the same ArrayList. This way we can use each object's unique methods
- In place of a method parameter, we can provide a subclass of the parameter to the method as well

9.6 Polymorphism
- Reference variable can store a reference to its declared class or to any subclass of its declared class
- Metho considered polymorphic when it's overridden in at least one subclass
- At compile time, the method has to exist in the declared type of the variable. At run time, the method will execute from the actual type of the object (for non-static method calls)

- We can downcast an object to use its methods: ex- ((Comedian) talisa) . tellJoke(). This prevents a compile-time error

9.7 Object Superclass

- Object class is the superclass of all other classes in Java
- toString(): Object's method used when an object is passed as a parameter to print() or println() method
- Returns the class name "@" the hashCode() value of object
- Classes can override toString() to be more specific, and the modified toString() method will be inherited by all subclasses
- equals() method inherited from Object
- Returns whether calling object == parameter object, meaning the two variables reference the same object
- Class-specific implementation: use if statement to check if one is not an instance of another. If true→ return false. If false → downcast the parameter object and then compare appropriate variable values

Unit 10 Notes

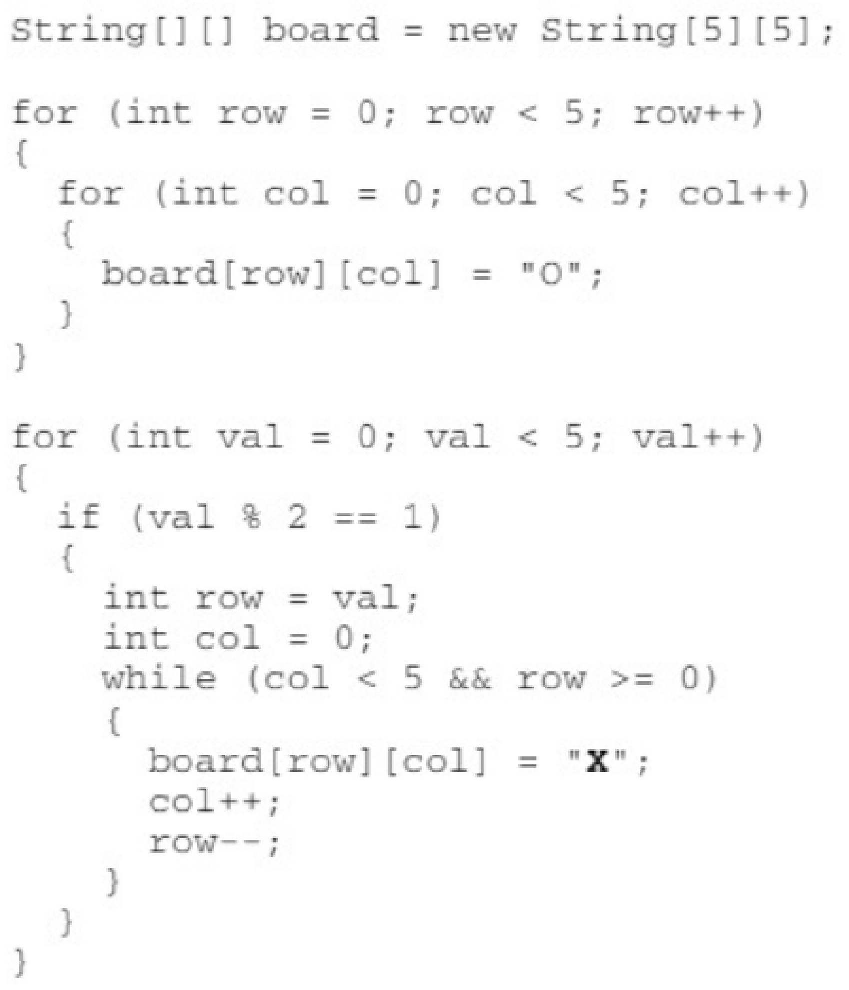
10.1 Recursion

- A recursive method is a method that calls itself
  - Must contain a base case (to stop the recursion) and a recursive call (to repeat the code)
    - Multiple of each may be used
      - When using multiple recursive calls, go left to right all the way down to the bottom level
  - Each recursive call has its own set of local variables
  - While executing a recursive process, parameter values will track the progress similar to a loop
- Any recursive code can also be done with iteration; any iterative code can be written recursively
- Must ensure the base case will be reached to avoid a CallStackOverflowException (infinite loop)
- When returning values while using recursion, return output + methodCall(param);
- Recursion can be used to traverse String, array, and ArrayList objects

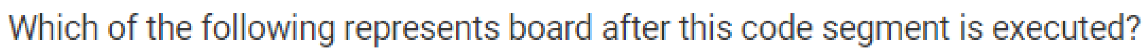
Test 1 Review

MCQ 2015

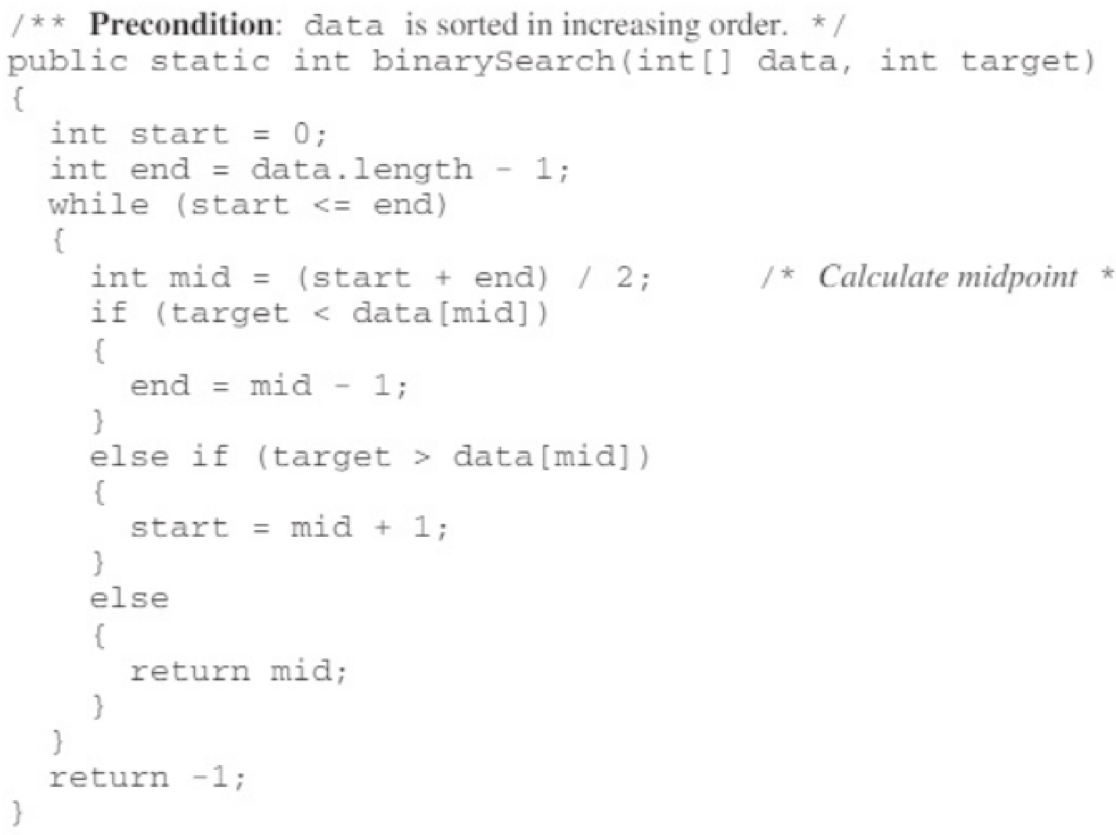
- Overall Score
    -
      2015 Practice Exam MCQ 04/19 08:00 AM 04/19 11:59 AM - Scored 35/39
    - I finished all the questions except for 27 which I skipped and didn't have time to go back to, so out of the questions I completed, I scored 35/38
- Missed Questions: 31, 35, 37
    - Question 31



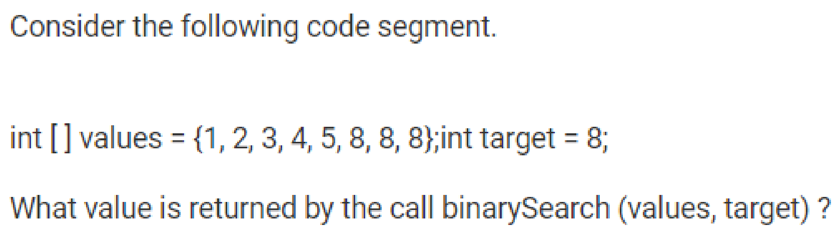
```
String[][] board = new String[5][5];

for (int row = 0; row < 5; row++)
{
  for (int col = 0; col < 5; col++)
  {
    board[row][col] = "O";
  }
}

for (int val = 0; val < 5; val++)
{
  if (val % 2 == 1)
  {
    int row = val;
    int col = 0;
    while (col < 5 && row >= 0)
    {
      board[row][col] = "X";
      col++;
      row--;
    }
  }
}
```

- 
  Which of the following represents board after this code segment is executed?
- I answered B, though the answer was E

        - Looking at the code, I thought it would just be a checkerboard pattern with X's at 0,1 and 1,0 and then down the board
        - However, the pattern stops at the 0,4 and 4,0 diagonal since the code doesn't reach the bottom right corner of the board due to the col++ and row--
    - Question 35



```
/** Precondition: data is sorted in increasing order. */
public static int binarySearch(int[] data, int target)
{
  int start = 0;
  int end = data.length - 1;
  while (start <= end)
  {
    int mid = (start + end) / 2;      /* Calculate midpoint *
    if (target < data[mid])
    {
      end = mid - 1;
    }
    else if (target > data[mid])
    {
      start = mid + 1;
    }
    else
    {
      return mid;
    }
  }
  return -1;
}
```

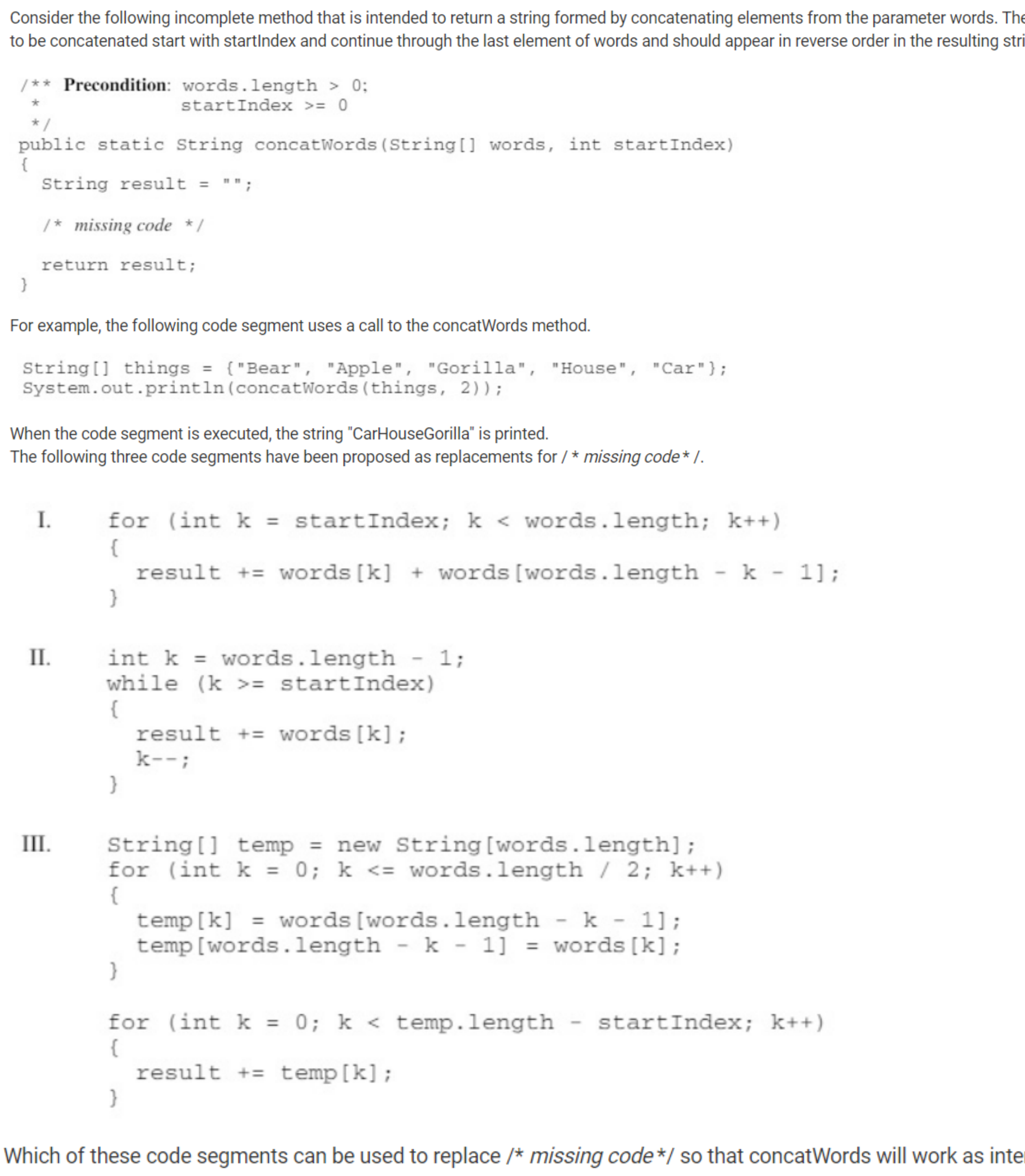


Consider the following code segment.

int [ ] values = {1, 2, 3, 4, 5, 8, 8, 8};int target = 8;

What value is returned by the call binarySearch (values, target) ?

        -
        - I answered D, though the answer was C
        - Because the code is calculating midpoints and then looking for the target from there, the first index of target found is 6, not 5
        - I thought it would just find the first instance of the target number, so I didn't look at the order in which code was executed
    - Question 37

- 

  Consider the following incomplete method that is intended to return a string formed by concatenating elements from the parameter words. The
  to be concatenated start with startIndex and continue through the last element of words and should appear in reverse order in the resulting stri

  ```
  /** Precondition: words.length > 0;
   *                startIndex >= 0
   */
  public static String concatWords(String[] words, int startIndex)
  {
    String result = "";

    /* missing code */

    return result;
  }
  ```

  For example, the following code segment uses a call to the concatWords method.

  ```
  String[] things = {"Bear", "Apple", "Gorilla", "House", "Car"};
  System.out.println(concatWords(things, 2));
  ```

  When the code segment is executed, the string "CarHouseGorilla" is printed.
  The following three code segments have been proposed as replacements for / * *missing code* * /.

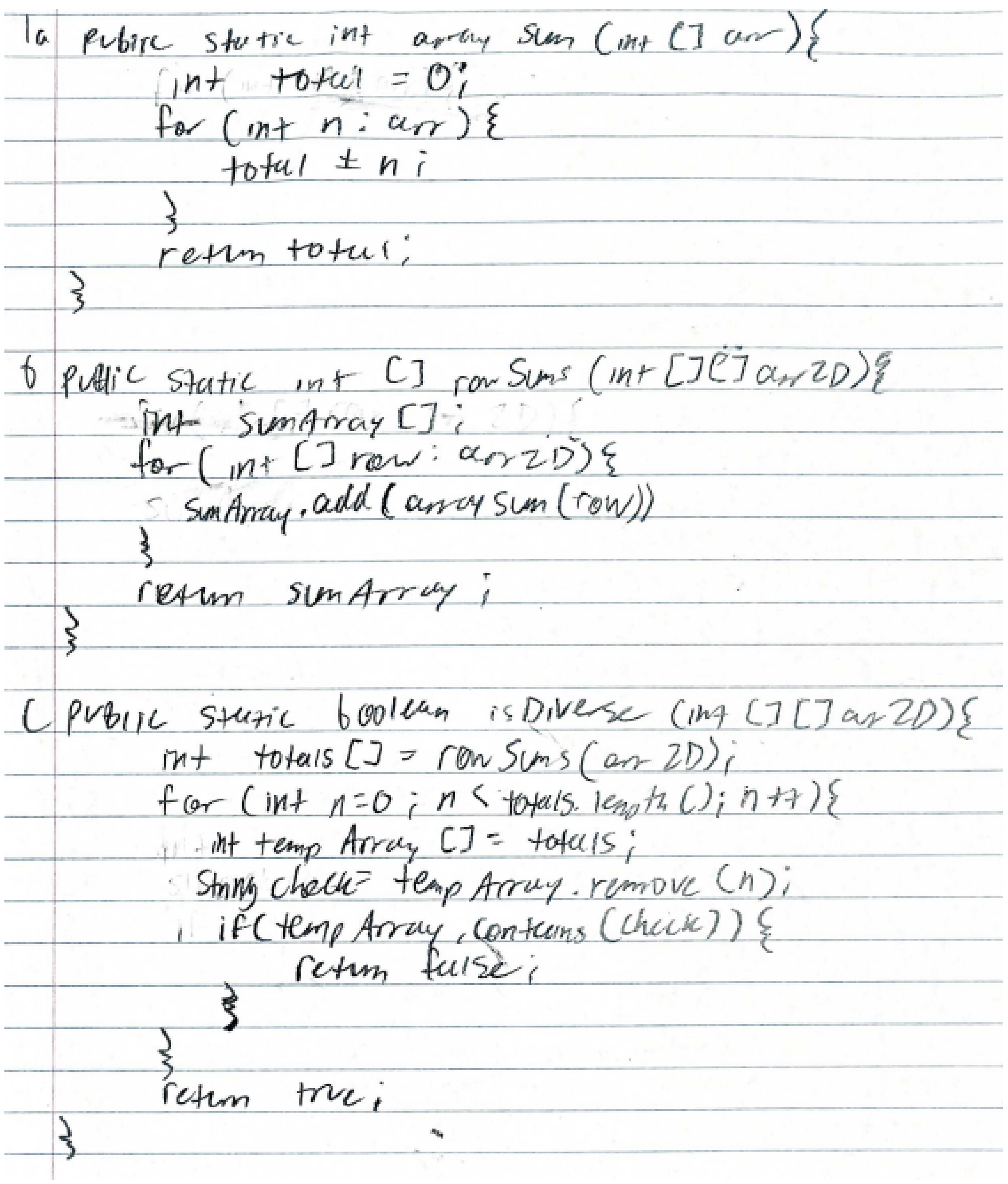
  ```
  I.    for (int k = startIndex; k < words.length; k++)
        {
          result += words[k] + words[words.length - k - 1];
        }

  II.   int k = words.length - 1;
        while (k >= startIndex)
        {
          result += words[k];
          k--;
        }

  III.  String[] temp = new String[words.length];
        for (int k = 0; k <= words.length / 2; k++)
        {
          temp[k] = words[words.length - k - 1];
          temp[words.length - k - 1] = words[k];
        }

        for (int k = 0; k < temp.length - startIndex; k++)
        {
          result += temp[k];
        }
  ```

  Which of these code segments can be used to replace /* *missing code* */ so that concatWords will work as inte
-
- I answered D (I and II), though the answer was E (II and III)
- Option I doesn't work, since it would add things to the result in the wrong order
- Option III works and would return the proper string upon executing code

FRQ 2015

- Repl with FRQs

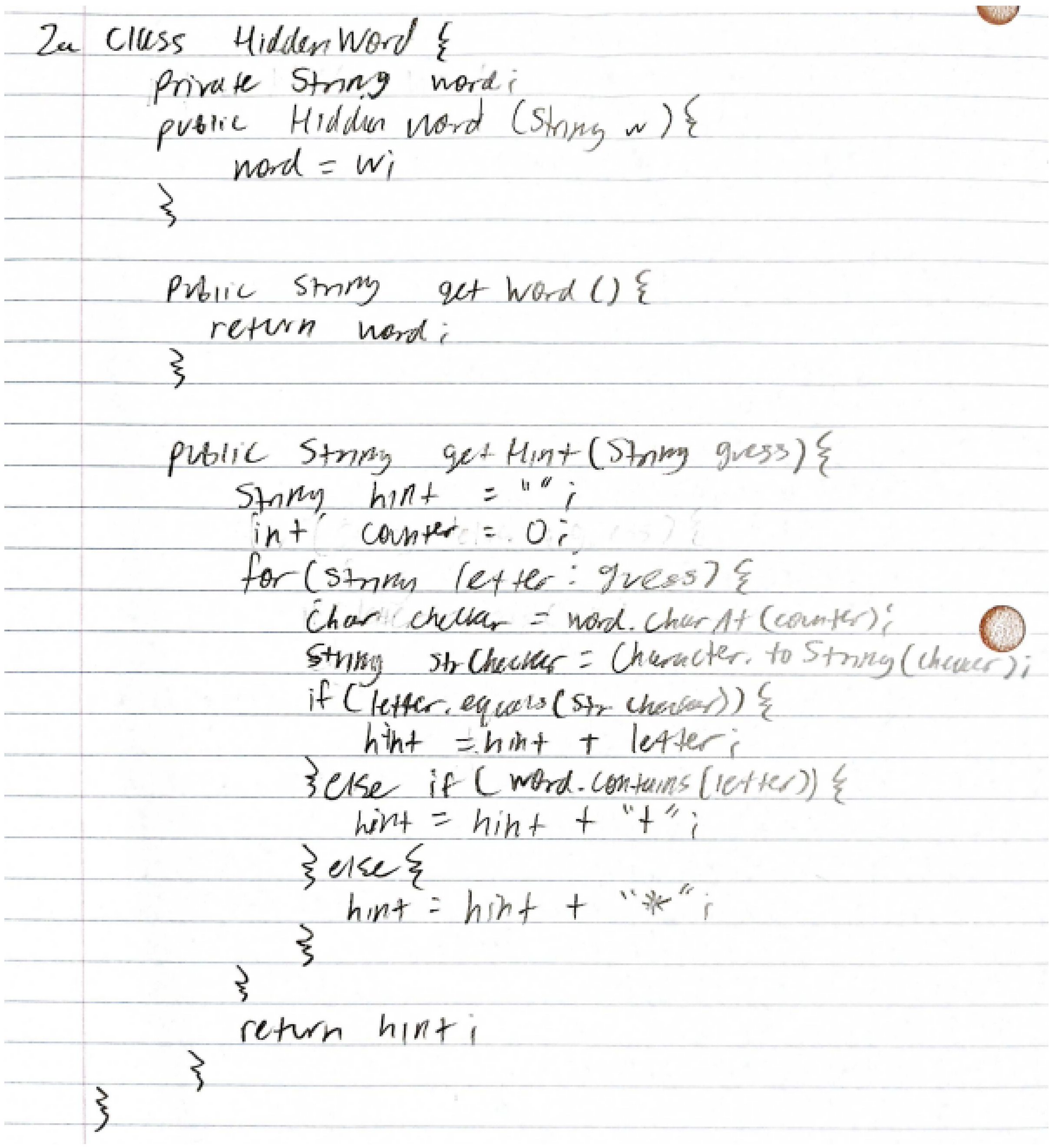
- Question 1



```
1a public static int arraySum (int [] arr){
        int total = 0;
        for (int n : arr){
            total ± n ;
        }
        return total;
   }

b public static int [] rowSums (int [][] arr2D){
       int sumArray [];
       for ( int [] row : arr2D){
          sumArray.add (arraySum (row))
       }
       return sumArray ;
   }

c public static boolean isDiverse (int [] [] arr2D){
        int totals [] = rowSums (arr2D);
        for (int n=0 ; n < totals.length (); n++){
          int tempArray [] = totals ;
           String check= tempArray.remove (n);
           if(tempArray.contains (check)) {
                return false;
           }
        }
        return true;
   }
```

    ○

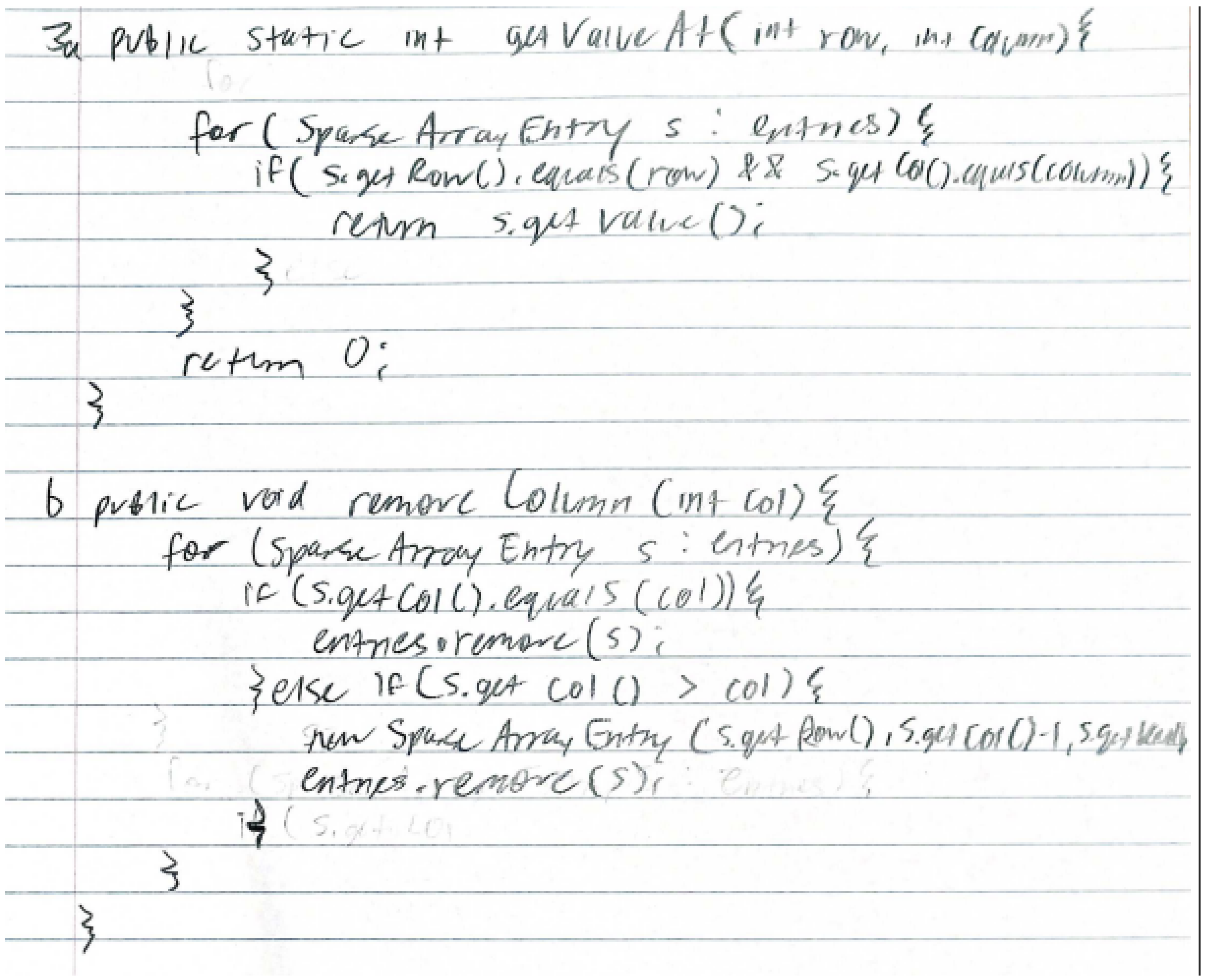
- Question 2



```
2a class HiddenWord {
     private String word;
     public Hidden word (String w) {
          word = w;
     }

     Public String getWord () {
        return word;
     }

     public String getHint(String guess) {
          String hint = "";
          int counter = 0;
          for(String letter : guess) {
               char checker = word.charAt(counter);
               String strChecker = (Character. toString(checker));
               if (letter.equals(strChecker)) {
                    hint = hint + letter;
               } else if (word.contains(letter)) {
                    hint = hint + "+";
               } else {
                    hint = hint + "*";
               }
          }
          return hint;
     }
}
```

○

- Question 3



```
3a public static int getValueAt(int row, int column){

        for (SparseArrayEntry s : entries){
            if( s.getRow().equals(row) && s.getCol().equals(column)){
                return s.getValue();
            }
        }
        return 0;
   }

b public void removeColumn(int col){
      for (SparseArrayEntry s : entries){
          if (s.getCol().equals(col)){
              entries.remove(s);
          }else if(s.getCol() > col){
              new SparseArrayEntry(s.getRow(), s.getCol()-1, s.getValue
              entries.remove(s);
          }
      }
  }
```