CSA Review Guide

Unit 10

Recursion

- method that calls itself within its own method
- Another way to do an iterative process

Consider the following recursive method:

```
public static void simpleRecur(int n)
{
    System.out.println(n);
    if (n > 2)
        simpleRecur(n-1);
    System.out.println(n);
}
```

Let's trace the call `simpleRecur(4)`:

Call Stack	Variable trace for call
`simpleRecur(4)`	`n = 4` `4>2 T`
[illegible]	[illegible]
[illegible]	[illegible]

```
4
3
2
2
3
4
```

-
- Recursion traces down the call stack until the if statement terminates
- traces back up the call stack until returning to the original value and terminating

Recursive Searching and Sorting

- Binary search algorithm: starts at middle of arraylist, eliminates half of array each iteration until all elements are eliminated or desired value is found
- Uses high and low parameters
  - take a middle point and see its value in relation to target
  - adjust up or down and eliminate values in wrong direction
- Sort data first

Unit 9

super Keyword

- calling the superclass's implementation of the method
  - super(); or super(arguments); calls just the super constructor if put in as the first line of a subclass constructor.
  - super.method(); calls a superclass' method (not constructors).
- allows us to not have to duplicate code twice across multiple classes

For Example:

Consider the following class declarations.

```
public class Parent
{
     public void first()
{
System.out.print("P");
second();
}
public void second()
{
     System.out.print("Q");
}
}
public class Child extends Parent
{
public void first()
{
     super.first();
}
public void second()
{
     super.second();
     System.out.print("R");
}
}
public class Grandchild extends Child
{
     public void first()
     {
```

```
            super.first();
            System.out.print("S");
        }
        public void second()
        {
            super.second();
            System.out.print("T");
        }
    }
```

Which of the following code segments, if located in another class, will produce the output "PQRTS" ?

- Grandchild d = new Grandchild();
  d.first();

    - The variable d is instantiated as a Grandchild object
        - call d.first() is invoked, it calls super.first(), which calls super.first() (first method of child class), invoking the first method of the Parent class
            - Prints "P" and then calls second.
    - second method of the Grandchild class is invoked.
        - series of calls to super.second(), prints "Q",
        - second method of the Child class prints "R",
        - second method of the Grandchild class prints "T".
    - control returns to the first method of the Grandchild class and it completes execution by printing "S".

Unit 8

Initialize a 2D Array

- new DataType[r][c]

- Each row of a 2D array has its own initializer list (with {})
  - To separate elements, you use a comma
    - {a,b}
  - To separate the rows, you also use a comma
    - {a,b},{c,d}

For Example:

Consider the following code segment, which is intended to create and initialize the two-dimensional (2D) integer array num so that columns with an even index will contain only even integers and columns with an odd index will contain only odd integers.

```
int[][] num = /* missing code */;
```

Which of the following initializer lists could replace /* missing code */ so that the code segment will work as intended?

- {{0, 1, 2}, {4, 5, 6}, {8, 3, 6}}

column 0	column 1	column 2
0	1	2
4	5	6
8	3	6

  - column 0: 0, 4, 8 (all even)
  - column 1: 1, 5, 3 (all odd)
  - column 2: 2, 4, 6 (all even)

Size of a 2D array

- Number of rows is the same as the length of the array because a 2D array is an array of arrays
  - row = array.length
- Number of columns is based on the size of each row. So determine the number of elements in the first row of the 2D array
  - column = array[0].length

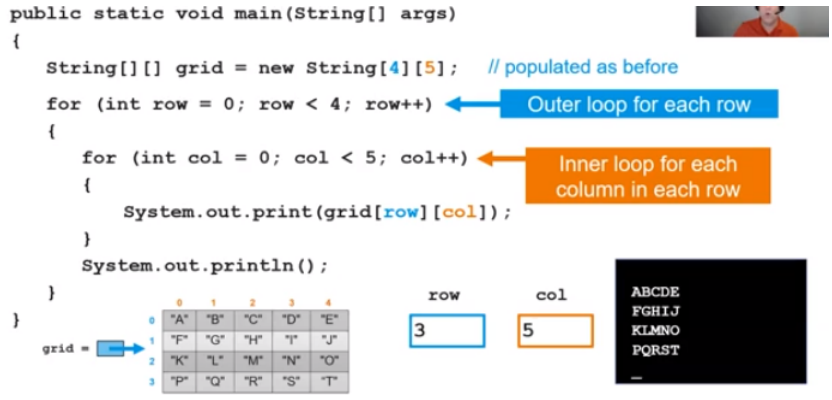
    - zero stands for row zero

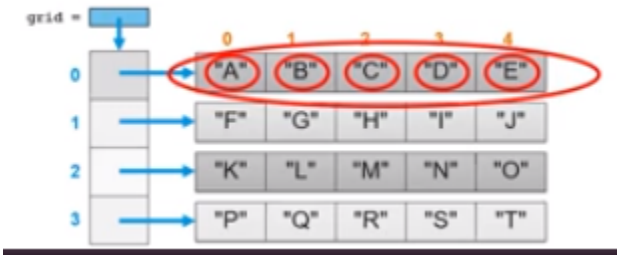
Accessing Elements

- index notation
    - array[row][column]
    - to access the last element
        - array[array.length-1][array[0].length-1]
- to change the value of the element, set the location of the element equal to the new value

Traversing Arrays

- For loops





    - Use an outer for loop to loop through each row
    - Use an inner for loop to loop through each column in each row
    - If you don't know the dimensions of the array, use array.length (for rows) and array[0].length (for columns) in the condition statement for the for loops

For Example:
Consider the following code segment.
int[][] mat = {{10, 15, 20, 25},

```
            {30, 35, 40, 45},
            {50, 55, 60, 65}};
for (int[] row : mat)
{
     for (int j = 0; j < row.length; j += 2)
     {
          System.out.print(row[j] + " ");
     }
     System.out.println();
}
```

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

- Iterate through each row of the row 2D array
  - print every other element of each row (start with element 0)

- 10 20
  30 40
  50 60

- Enhanced for loop

```
public static void printArray(String[][] grid)
{
   for (String[] row : grid)
   {
      for (String letter : row)
      {
         System.out.print(letter);
      }
      System.out.println();
   }
}
```

  - Get each (string) array within the grid
  - Get each (string) letter out of each row
  - Print out the letter
  - Print new line after each row

- Row-Major order goes row by row (across) whereas Column-Major order goes column by column (down)

- Column-Major order
  - For loop
    - Columns on outer loop and rows in inner loop

- Search for a value

- For loop

```
public static boolean search(String[][] chart, String name)
{
   for (int r = 0; r < chart.length; r++)
   {
      for (int c = 0; c < chart[0].length; c++)
      {
         if (chart[r][c].equals(name))
            return true;
      }
   }
   return false;
}
```

  - loops through each column

- Enhanced for loop

```
public static boolean search(String[][] chart, String name)
{
   // if chart was a 1D array...

      for (String item : chart)
      {
         if (item.equals(name))
            return true;
      }

   return false;
}
```

  - loops through each item of each row

Unit 7

- Initialize ArrayList
  - ArrayList<Type> [NAME] = new ArrayList<Type>();
    - this is empty
- Add to arraylist
  - [NAME].add([CONTENT]);
- Set values
  - [NAME].set([INDEX], [VALUE]);
- Remove from arraylist
  - [NAME].remove([INDEX]);

Arrays

- have to call the array within the available indices or it will throw an error
- can use to
  - get average
  - get specific information

- know the different types of inits
  - initializing a string array with integers would return empty

ArrayList

- Imported to java, not native to java
  - import java.util.ArrayList;

Properties

- ArrayLists are collections of object reference data
- mutable
- Arrays are static in size, one defined cannot be changed
- ArrayLists are not, resizable length

- ArrayList
  - Resizable length
  - Part of a framework
  - Class with many methods
  - Is designed to be flexible
  - Not Designed to store primitives
  - Use wrapper classes (Integer)

Creating ArrayLists

```
ArrayList<DataType> variableName;
ArrayList<DataType> variableName = new ArrayList<DataType>();
```

- creating new array list and defining its default constructor

```
ArrayList<DataType> variableName = new ArrayList<DataType>(n);
```

- DataType can be any nonprimitive data type
- n is the initial number of elements in ArrayLIst
- ArrayList objects are designed to only store references to objects, not primitive value. A workaround is to use Wrapper classes which store primitive values as objects
    - Primitive:Wrapper
        - boolean:Boolean
        - char:Character
        - double:Double
        - integer:Integer

ArrayList Method

- a1.add(n) adds element at an index of 2
- a1.add(x, y) adds element y at an index of x
- a1.remove(n) removes the element at an index of n
- a1.set(x, y) sets the element at an index of x to y
- a1.remove() can also return a variable
    - Integer thatOneThingWeRemoved = a1.remove(2);
    - System.out.println(thatOneThingWeRemoved);
- a1.get(n) returns the element at an index of n

Unit 6

Traversing Arrays

- Iteration statements can be used to call all elements in array
  -
- Traversing an array with an indexed for or while loop requires elements to be accessed with their indexes
- Since indexes for an array start at 0 and end at one less than the number of elements, be careful to not be "off by one"
  - this will throw an ArrayIndexOutOfBoundsException

Enhanced for Loop for Arrays

- Enhanced for loop header includes a variable (enhanced for loop variable)
- For each iteration of the enhanced for loop, the enhanced for loop variable is assigned a copy of an element without its index
- Assigning a new value to the enhanced for loop variable does not change the value stored in the array
- Program code written using an enhanced for loop to traverse and access elements in an array can be rewritten using an indexed for loop or while loop

For Example:

Segment 1:

```
int[] arr = {1, 2, 3, 4, 5};
for (int x = 0; x < arr.length; x++)
{
     System.out.print(arr[x + 3]);
}
```

Segment 2:

```
int[] arr = {1, 2, 3, 4, 5};
for (int x : arr)
{
System.out.print(x + 3);
}
```

Which of the following best describes the behavior of code segment I and code segment II ?

Code segment I will cause an ArrayIndexOutOfBoundsException and code segment II will print 45678.

- Segment 1:
  - 1st iteration: arr[0 + 3] = 4
  - 2nd iteration: arr[1 + 3] = 5
  - 3rd iteration: arr[2 + 4] =
  - Output: ArrayIndexOutOfBoundsException
    - Array only has 5 elements (0-4), arr[6] will throw an error
- Segment 2:
  - 1st iteration: 1 + 3 = 4
  - 2nd iteration: 2 + 3 = 5
  - 3rd iteration: 3 + 3 = 6
  - 4th iteration: 4 + 3 = 7
  - 5th iteration: 5 + 3 = 8
  - Output: 45678

Developing Algorithms Using Arrays

- Standard algorithms can utilize array traversals to
  - Determine minimum or maximum value
  - Compute a sum, average, or mode
  - Determine if at least one element has a particular property
  - Determine if all elements have a particular property
  - Access all consecutive pairs of elements
  - Determine the Presence or Absence of duplicate elements
  - Determine the area of elements meeting specific criteria
- Standard array algorithms utilize traversals to
  - Shift or rotate elements (left or right)
  - Reverse the order of elements

For Example:

The code segment below is intended to set the boolean variable duplicates to true if the int array arr contains any pair of duplicate elements. Assume that arr has been properly declared and initialized.

```
boolean duplicates = false;
for (int x = 0; x < arr.length - 1; x++)
{
     /* missing loop header */
```

```
            {
                  if (arr[x] == arr[y])
                  {
                        duplicates = true;
                  }
            }
      }
```

Which of the following can replace /* missing loop header */ so that the code segment works as intended?

```
      for (int y = x + 1; y < arr.length; y++)
```

- y++ : loop will iterate through all elements of the array
- y = x + 1 : causes each inner loop to examine all the elements that come after the current element for each iteration. Ensures no element will be compared with itself
- y < arr.length : loop will terminate when there are no more elements in the array

Unit 5

Static Variables and Methods

- Static Variable Behaviors
  - Static methods are associated with the class, not objects of the class.
  - Static methods include the keyword static in the header before the method name
  - Static methods cannot access or change the values of instance variables and can only access or change the values of static variables
  - Static methods do not have a "this" reference and are unable to use the class's instance variables or call non-static methods.
- static variables that belong to the class
  - Static variables belong to the class, with all objects of a class sharing a single static variable.
  - Static variables can be designated as either public or private and are designated with the static keyword before the variable type.
  - Static variables are used with the class name and the dot operator, since they are associated with a class, not objects of a class.

For Example:

```
public class Beverage
{
      private int numOunces;
      private static int numSold = 0;
      public Beverage(int numOz)
      {
            numOunces = numOz;
      }
      public static void sell(int n)
      {
            /* implementation not shown */
      }
}
```

- numSold can be accessed and updated; numOunces cannot be accessed or updated.
  - Static methods can access and update static variables like numSold but not instance variables like numOunces

Scope and Access

- where variables can be used in the program code
  - Local variables can be declared in the body of constructors and methods. These variables may only be used within the constructor or method and cannot be declared to be public or private.
  - When there is a local variable with the same name as an instance variable, the variable name will refer to the local variable instead of the instance variable.
  - Formal parameters and variables declared in a method or constructor can only be used within that method or constructor.
  - Through method decomposition, a programmer breaks down a large problem into smaller subproblems by creating methods to solve each individual subproblem.

For Example:

```
public class Student
{
      private String firstName;
      private String lastName;
      private int age;
      public Student(String firstName, String lastName, int age)
      {
            firstName = firstName;
            lastName = lastName;
            age = age;
      }
      public String toString()
      {
            return firstName + " " + lastName;
      }
}
```

The following code segment appears in a method in a class other than Student. It is intended to create a Student object and then to print the first name and last name associated with that object.

```
Student s = new Student("Priya", "Banerjee", -1);
System.out.println(s);
```

- The code segment will compile, but the instance variables will not be initialized correctly because the variable names firstName, lastName, and age refer to the local variables inside the constructor.
    - When there is a local variable with the same name as an instance variable, the variable name will refer to the local variable instead of the instance variable
    - scoping issue can be resolved by using parameter names that are different from the instance variable names.

Unit 4

while Loops

- Iteration statements change the flow of control by repeating the code a set amount of times until a condition is met
- In loops, boolean expression is evaluated before each iteration (including the first)
  - Evaluates true, loop is executed
  - Evaluates false, iteration ceases
- Infinite loop if boolean expression always evaluates true
- If the boolean initially evaluates false, the loop will not execute
- Return statement will stop and exit
- Standard algorithms used to
  - identify if integer is/is not evenly divisible by another integer
  - Identify individual digits in an integer
  - Determine frequency of condition
  - Determine minimum or maximum
  - Calculate sum, average, mode

For Example:

```
int a = 1;
string result = "";
while (a < 20)
{
     result += a;
     a += 5;
}
System.out.println(result);
```

- First iteration: 1
- Second iteration: 6
- Third iteration: 11
- Fourth iteration: 16
- Stops after fourth iteration because 21 is greater than 20

for Loops

- parts of a loop header: initialization, boolean expression, increment/decrement statement
- initialization statement only executed once before first boolean expression evaluation

    - variable is called loop control variable
- increment statement executed after the entire loop before the boolean expression
- for loops and while loops can be rewritten as equivalents
- Off by one error is when iteration statement loops one too many or too few times

For Example:

```
Code Segment I
for (int i = 0; i < 10; i++)
{
     System.out.print( "*" );
}

Code Segment II
for (int i = 1; i <= 10; i++)
{
     System.out.print( "*" );
}
```

- The output of the code segments is the same because the loops in both code segments iterate 10 times
    - Code segment I causes the print statement to be executed 10 times, for values of i between 0 and 9, inclusive
    - Code segment II also causes the print statement to be executed 10 times, for values of i between 1 and 10, inclusive.

```
int n = 6;
for (int i = 1; i < n; i = i + 2)  // Line 2
{
     System.out.print(i + " ");
}
```

Which of the following best explains how changing i < n to i <= n in line 2 will change the result?

- There will be no change to the program output because the loop will iterate the same number of times.
- Original code
    - i is initialized to 1 and is increased by 2 repeatedly until it is at least 6
    - loop iterates three times, producing the output "1 3 5 "
    - when i is 7, the loop terminates.

- New code
  - the code segment will still initialize i to 1 and increase it by 2 repeatedly until it exceeds 6
  - loop iterates three times, producing the output "1 3 5 ".
  - when i is 7, the loop terminates.

Unit 3

Boolean Expressions

- Boolean values return either true or false
- Primitive values and reference values can be compared using relational operators
  - == equal to
  - != not equal to
- Arithmetic expression values can be compared using relational operators
  - < less than
  - > greater than
  - <= less than or equal to
  - >= greater than or equal to
- An expression involving relational operators evaluates to a Boolean value.

For example:

```
boolean a = true;
boolean b = false;
System.out.print((a == !b) != false);
```

- First line sets the boolean variable named a as true
- Second line sets the boolean variable named b as false
- Third line
  - a == !b: does true == not false?
    - true is equal to true, therefore returns true
  - true != false: does true != false?
    - true is not equal to false, therefore returns true
  - Code prints true

If-else statement

- uses logical conditions (<, <=, >, >=, ==, !=)
- Use if to specify a block of code to be executed, if a specified condition is true
- Use else to specify a block of code to be executed, if the same condition is false
- Use else if to specify a new condition to test, if the first condition is false

```
if (CONDITION) {
```

```
        CODE TO EXECUTE
    }
    else {
        CODE TO EXECUTE
    }
```

For example:

```
    String weather;

    if (temp <= 31)
    {
        weather = "cold";
    }
    else
    {
        weather = "cool";
    }
    if (temp >= 51)
    {
        weather = "moderate";
    }
    else
    {
        weather = "warm";
    }
    System.out.print(weather);
```

Which of the following test cases can be used to show that the code does NOT work as intended?

temp = 30
temp = 51
temp = 60

- temp = 30
  - weather = "cold"
  - weather = " warm"
  - Incorrect
- temp = 51
  - weather = "cool"
  - weather = "moderate"
- temp = 60
  - weather = "cool"
  - weather = "moderate"

Unit 2

String Objects - Concatenation

- Concatenation = joining of data
- operators are + and +=
    - + joins
    - += joins && redefines
        - = means becomes

For example:

```
String str = "AP";
str += "CS " + 1 + 2;
System.out.println(str);
```

- first line sets AP as the string variable str
- line 2 concatenates CS 12 to the variable str
    - the + in the second line (between "CS", 1, and 2) is not an addition symbol, it is telling the computer to join the string "CS", the integer 1, and the integer 2
    - str becomes AP CS12
- prints AP CS12

Unit 1

Variable Declaration

int *VARIABLE ;*

- Stores variable as an integer

boolean *VARIABLE ;*

- Stores variable as a boolean
  - True or False

double *VARIABLE ;*

- Stores a variable as both the whole number and the fractional parts

For example:

Which statement correctly declares a variable that can store a temperature rounded to the nearest tenth of a degree?

double patientTemp ;

Using double will allow the variable to accept values that are decimals

Expressions and Assignment Statements

<u>Temporary Storing in Variables</u>

Values can be temporary stored in another variable

For Example:

```
double volume = pi * r * r ;
volume = volume * h ;
```

- The first line temporarily stores the base area in the volume variable
- This value is then multiplied by the height (h) to get the correct volume

Math Functions

+ : Addition, Add values

- : Subtraction, Subtract values

* : Multiplication, Multiply values

/ : Division, Divide values
- Division rounds down

% : Modulo, Find the remainder of the values

For Example:

```
1 / (5 % 3)

     = 1 / 2
     = 1
```

```
2 / 5 + 1

     =  0 + 1
     = 1
```

```
int x = 10;
int y = 20;
System.out.print(y + x / y);

     = 20 + 10 / 20
     = 20 + 1/2
     = 20 + 0
```

Compound Assignment Operators

x -= y

- subtract y from x and store this difference as the variable x

x += y

- add x and y and story this sum as the variable x

For example:

```
int x = 4;
int y = 6;
x -= y;
y += x;
```

```
x = x - y = 4 - 6 = -2
y = y + x = 6 + (-2) = 4
```

Casting and Ranges of Variables

(int)*VARIABLE*

- stores variable as an integer
- if a decimal or fraction, digits to the right of the decimal are truncated

(double)*VARIABLE*

- stores variable as a double
  - accepts decimals and fractions

For example:

```
double d = 0.25;
int i = 3;
double diff = d - i;
System.out.print((int)diff - 0.5);
```

```
d - i = 0.25 - 3 = -2.75
(int)d will round this to -2
-2 - 0.5 = -2.5
```

```
double a = 7;
int b = (int) (a / 2);
double c = (double) b / 2;
System.out.print(b);
System.out.print(" ");
System.out.print(c);
```

```
7 / 2 = 3.5
int will round this to 3
3 / 2 = 1.5
```

double will let variable c accept decimals
output will be "3 1.5"

```
double p = 10.6;
double n = -0.2;
System.out.println((int) (p + 0.5));
System.out.print((int) (n - 0.5));
```

p + 0.5 = 10.5 + 0.5 = 11
(int) will store this as 11
n - 0.5 = -0.2 - 0.5 = -0.7
(int) will round this as 0
output will be "11 0"

```
int num1 = 5;
int num2 = 10;
double ans = num1 / num2;
System.out.print(ans);
```

in order to get the output to be 0.5, you need to cast either num 1 or num 2 to double. This will ensure that ans can accept decimal values. If you cast num1 / num2 to double, it will return an error as the expression will divide by 0