AP CS A Study Guide – 2019

By Laura Gray
GSMST

Laura_Gray@gwinnett.k12.ga.us.

AP Computer Science Exam: Tuesday, 5/17/19 at Noon in MLH

Be in the area outside the MLH at 11:55

- The exam will start around Noon, as soon as the lecture halls are cleared from morning testing.
- **No students will be allowed to enter late or leave early.**

All students need to bring:

- Black or blue pen
- Sharpened #2 pencils (it is suggested to bring 2 of each).

All students taking AP tests are marked excused all day (school activity).

- You can come to school for 2nd lunch at 11:30.
- Those of you who come to school at regular start time are expected to attend 1st – 4th periods.
- You are NOT ALLOWED to leave campus and come back onto campus.

•Your AP Testing will end after normal school dismissal (around 4:00pm). You need to arrange a ride home if you normally take the bus.

Reminders and tips:

- No food or drink in the testing room.
- **No smart watches are allowed!**
- Cell phones/electronic devices are not allowed to be on your possession while testing and cannot be used during the break.
- No scratch paper allowed.
- All book bags, purses, laptops must be left at an assigned area in the room.

You will receive scores online in July.
Sign up/login for College Board account at ***apscore.org***

Exam Appendix — Java Quick Reference

Accessible methods from the Java library that may be included on the exam

class java.lang.Object

- `boolean equals(Object other)`
- `String toString()`

class java.lang.Integer

- `Integer(int value)`
- `int intValue()`
- `Integer.MIN_VALUE` // minimum value represented by an `int` or `Integer`
- `Integer.MAX_VALUE` // maximum value represented by an `int` or `Integer`

class java.lang.Double

- `Double(double value)`
- `double doubleValue()`

class java.lang.String

- `int length()`
- `String substring(int from, int to)` // returns the substring beginning at `from`
  // and ending at `to-1`
- `String substring(int from)` // returns `substring(from, length())`
- `int indexOf(String str)` // returns the index of the first occurrence of `str`;
  // returns `-1` if not found
- `int compareTo(String other)` // returns a value < 0 if `this` is less than `other`
  // returns a value = 0 if `this` is equal to `other`
  // returns a value > 0 if `this` is greater than `other`

class java.lang.Math

- `static int abs(int x)`
- `static double abs(double x)`
- `static double pow(double base, double exponent)`
- `static double sqrt(double x)`
- `static double random()` // returns a `double` in the range [0.0, 1.0)

interface java.util.List<E>

- `int size()`
- `boolean add(E obj)` // appends `obj` to end of list; returns `true`
- `void add(int index, E obj)` // inserts `obj` at position `index` ($0 \le \text{index} \le \text{size}$),
  // moving elements at position `index` and higher
  // to the right (adds 1 to their indices) and adjusts size
- `E get(int index)`
- `E set(int index, E obj)` // replaces the element at position `index` with `obj`
  // returns the element formerly at the specified position
- `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 adjusts size
  // returns the element formerly at the specified position

class java.util.ArrayList<E> implements java.util.List<E>

Exam Format

Section I

Multiple Choice — 40 Questions | 1 Hour, 30 Minutes | 50% of Exam Score

Question topics will include:

- Programming Fundamentals
- Data Structures
- Logic
- Algorithms/Problem Solving
- Object-Oriented Programming
- Recursion
- Software Engineering

Classification Category	Percentage of Multiple-Choice Items
Programming Fundamentals	55-75%
Data Structures	20-40%
Logic	5-15%
Algorithms/Problem Solving	24-45%
Object-Oriented Programming	15-25%
Recursion	5-15%
Software Engineering	2-10%

Section II

Free Response — 4 Questions | 1 Hour, 30 Minutes | 50% of Exam Score

The free-response section tests your students' ability to solve problems involving more extended reasoning.

Question 1: Methods and Control Structures, where students call methods and work with control structures without the added complexity of data structures.
Question 2: Class, where students design and implement a described class.
Question 3: Array/ArrayList, where students complete program code that uses array or ArrayList objects.
Question 4: 2-D Array, where students complete program code that uses 2-D arrays.

VARIABLES AND OBJECTS

ALWAYS define and assign a value to variables when doing free response. If your loop or conditional logic does not run, you want to assign a value to return.

Example:

```
public static int example(){
int a;
if(a>5) return a=5;
return a;
}

//This will not store a value for a if it is not greater than 5.
```

The fix:

```
public static int example(){
int a=0;
if(a>5) return a=5;
return a;
}
```

**ONLY CLASS LEVEL VARIABLES HAVE DEFAULT VALUES!

Primitive types:

Primitive types in Java (pass by value) that you need to care about: int, double, boolean, char.

```
boolean = true;  //don't use quotes as this is a primitive type, not a String!
int x = 6;
int y = 6.5; //will cause error
int x = 7; //will cause error as x is already declared
double z = 8.5;
double a = x; //will work fine and store a as 6.0, a double
int b = (int) z; //will work as I cast z to an int of the value of 8.5
double c = 6/5; //will store a 1.0 as these are two integers and we have round off
double d = 6.0/5 //will store 1.2 as we have a double in the calculation
double e = (int)6.0/5 //will store 1.0
double f = (int) 6.0/5; //will store 1.0
double g = (int) 6/5.0 //will store 1.2
double h = (double)3/4 //will store 0.75
double I = (double)(3/4) //will store 0.0 (3/4 = 0 as they are ints and casting a 0 to a double is 0.0
```

You CANNOT cast to a String it is not primitive. You have to use a method to do that. You will not need to use a conversion method for this exam. If you do, you are overcomplicating things.

Remember, Strings are Weird (immutable)

String a = "Hi";

String b = "Hi";

System.out.println(a==b); //prints true, same object location

System.out.println(a.equals(b)); //prints true, same Strings

b = new String("Hi"); //divorced with the word new. String b wants a new life

System.out.println(a==b); //prints false, different object location

System.out.println(a.equals(b)); //prints true, same Strings

Strings are NOT primitive types. Objects will behave much differently than Strings as we can overwrite them if passed to methods.

This is an important one to review, as they seem to come up on the multiple choice exam:

```
public static void example(int a, int[]b){
a = 5;
for(int i = 0; i<b.length; i++)
        b[i]=i;}

public static void main(String[]args){
int[] c = {1, 8, 7, 2};
ind d = 9;
example(d, c);
System.out.print(d); //prints 9. Method is void and doesn't return a value
For(int e: c)
        System.out.print(e);//prints 0, 1, 2, 3; Arrays and Arraylists are permanently changed, unless I place them into a new arraylist. This is the case for all objects except Strings and primitive types
```

```java
class Main {
  public static void main(String[] args) {
    String[] a = {"Hi", "Hi"};
    int b = 5;
    String c = "Hi";
    main2(a, b, c);

    for(String d:  a)
    System.out.print(d + " ");

    System.out.println("\n" + b);
    System.out.println(c);

  }


  public static void main2(String[]a, int b, String c){
    a[0] = "Bye";
    b = 7;
    c = "Bye";

  }
}

```

This will print
Bye Hi
5
Hi

Remember voids cannot change primitive types or Strings, but it will change objects I pass, such as arrays, Arraylists, etc.

To change primitive types or Strings, we have to return the item and save it as that value (see next example.

```java
class Main {
  public static void main(String[] args) {
    String[] a = {"Hi", "Hi"};
    int b = 5;
    String c = "Hi";
    b = main2(a, b, c);

    for(String d:  a)
    System.out.print(d + " ");

    System.out.println("\n" + b);
    System.out.println(c);

  }


  public static int main2(String[]a, int b, String c){
    a[0] = "Bye";
    b = 7;
    c = "Bye";
    return b;

  }
}
```

This prints
Bye Hi
7
Hi

= vs ==

int a;

a=7; //Assigns a value of 7 to integer a

System.out.print(a==8); //prints a true/false statement if a is equal to 8

String b = “Hi”;

System.out.print(b.equals(“Hi”)); //checks if the string values are the same == is for object location for non primitive types

In free response, if they have created a string, array or other object as an instance variable without instantiating it. . .You generally need to instantiate it in your code! They do this a lot with arrays and ArrayLists:

```
public class Teacher{
private CourseInfo course1;

public Teacher(String subject ){
        course1 = new CourseInfo(subject); }
```

Class Level Variables ("Global" variables):

//ALWAYS CIRCLE OR HIGHLIGHT PRIVATE AND FINAL VARIABLES SO YOU DO NOT MISS THEM!

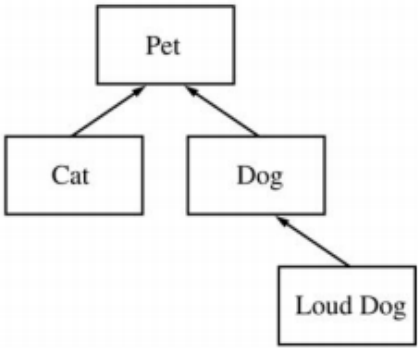
```
public class example{
public static final int a = 2;  //This variable value cannot be changed as it is a final or constant
public static int b =3;


public static void main(String[]args)
{
int a = 4;  //local variable will override class level in this method unless noted
System.out.print(a); //prints 4
System.out.print(example.a); //prints 2
System.out.print(b); //prints 3 as there is no local variable b
}}
```

CLASS LEVEL VARIABLES HAVE DEFAULT VALUES, LOCAL VARIABLES DO NOT.

Defaults:
String or Object = null
boolean = false
double = 0.0
int = 0

Sometimes, you need to cast an object if working with inheritance in order to use it properly.

Consider the hierarchy of classes shown in the following diagram.



Note that a `Cat` "is-a" `Pet`, a `Dog` "is-a" `Pet`, and a `LoudDog` "is-a" `Dog`.

The class `Pet` is specified as an abstract class as shown in the following declaration. Each `Pet` has a name that is specified when it is constructed.

```
public abstract class Pet
{
  private String myName;

  public Pet(String name)
  {  myName = name;  }

  public String getName()
  {  return myName;  }

  public abstract String speak();
}
```

The subclass `Dog` has the partial class declaration shown below.

```
public class Dog extends Pet
{
  public Dog(String name)
  {  /* implementation not shown */  }

  public String speak()
  {  /* implementation not shown */  }
}
```

(a) Given the class hierarchy shown above, write a complete class declaration for the class `Cat`, including implementations of its constructor and method(s). The `Cat` method `speak` returns `"meow"` when it is invoked.

(b) Assume that class `Dog` has been declared as shown at the beginning of the question. If the `String` *dog-sound* is returned by the `Dog` method `speak`, then the `LoudDog` method `speak` returns a `String` containing *dog-sound* repeated two times.

Given the class hierarchy shown previously, write a complete class declaration for the class `LoudDog`, including implementations of its constructor and method(s).

(c) Consider the following partial declaration of class `Kennel`.

```
public class Kennel
{
  private ArrayList petList;     // all elements are references
                                 // to Pet objects

  // postcondition: for each Pet in the kennel, its name followed
  //                by the result of a call to its speak method
  //                has been printed, one line per Pet
  public void allSpeak()
  {  /* to be implemented in this part */  }

  // ... constructor and other methods not shown
}
```

Write the `Kennel` method `allSpeak`. For each `Pet` in the kennel, `allSpeak` prints a line with the name of the `Pet` followed by the result of a call to its `speak` method.

In writing `allSpeak`, you may use any of the methods defined for any of the classes specified for this problem. Assume that these methods work as specified, regardless of what you wrote in parts (a) and (b). Solutions that reimplement functionality provided by these methods, rather than invoking these methods, will not receive full credit.

Complete method `allSpeak` below.

```
// postcondition: for each Pet in the kennel, its name followed
//                by the result of a call to its speak method
//                has been printed, one line per Pet
public void allSpeak()
```

For part c in the problem:

```
1  public void allSpeak() {
2    for (Object o: petList) {
3      Pet p = (Pet)o;
4      p.speak();
5    }
6  }
```

I need to cast to a pet object b/c the ArrayList stores objects. Java needs to know the object is a pet to call the correct methods. The list can store multiple pets, and through method overriding, it will run the correct methods in Pet (or overwrite them if the object is a Dog that extends Pet. and has a method that also exists in it's parent, Pet)

When to use the keyword new:
new = object instantiation (or provisioning a storage location and space for the object). When we create a new object, we should use this keyword.

When coding in free response, as yourself, is this a brand new thing that needs to be created?
For example:

```
public class Reservation{
        int reservationNum;
        String cust;
        public Reservation(int res, String cust){
                reservationNum = res;
                this.cust = cust;
        //need to use this keyword b/c we have a local and class level variable by the same name}}

public class Hotel{
        Reservation[] hotelrooms;
        public Hotel(int roomnums){
                hotelrooms = new Reservation[roomnums];
        }

        public String bookRoom(int resnum; String custnam){
                for(int i = 0; i<hotelrooms.length; i++){
                        if(hotelrooms[i]==null)
                                {hotelrooms[i]=new Reservation(resnum, custname);
                                //we need keywork new here b/c we need to store create a new reservation
                                return "Reservation booked;}}}


        public String bookRoom(Reservation booking){
                        for(int i = 0; i<hotelrooms.length; i++){
                                if(hotelrooms[i]==null)
                                        {hotelrooms[i]=booking);
                                        //reservation was already instantiated and sent to us, so no need for a new one
                                        return "Reservation booked;}}
                        return "Reservation not booked";}}
```

We can set objects such as String or a class level Object that we store in an Array or Arraylist with the value of null. (not in quotes). That being said, if we try to run a non-static method with that object or utilize an instance variable, we will receive the null pointer exception (as the null keyword does not point to the reference location of an object). When going through the content, you might need to program to avoid this exception in free response:

Let's say I have student objects that can be stored on a class roster and I want to add a student to that roster:

```
public class Main {
  public static void main(String[] args) {
    Student john = new Student("John Doe");
    Roster math = new Roster();
    math.addToRoster(john);
    math.printRoster();

  }
}

public class Roster{
  private Student[] roster = new Student[25];

  public void addToRoster(Student stud){
    for(int i = 0; i<roster.length; i++)
      if(roster[i]==null)
         {roster[i]=stud;
         return;}}

public void printRoster(){
  for(int i = 0; i<roster.length; i++)
     if(roster[i]!=null)
         System.out.println(roster[i].getName());}}

public class Student{
  private String name;

  public Student(String name){this.name=name;}
  public String getName(){return name;}}
```

MATH FUNCTIONS:

Random Number Generation:

```
int a = (int)(Math.random()*4);  //Options of 0, 1, 2, 3

int b = (int)(Math.random()*3+5);  //Options of 5, 6, 7

int c = (int)Math.random()*3+5;  //Option of 0
```

Remember, to talk the square root of something, you have to use:

```
Math.sqrt(16);  Returns double.
```

Math.pow(16, ½) will return a value of 1 as it will perform ½ = 0 (as they are integers) prior to calling the method. It will then take 16 to the power of 0, returning a double with a value of 1.0.

% and Dividing:

a%b calculates the remainder of dividing a by b

4%7 = 4

6%4 = 2

If you are asked on free response to work with placing the digits of a number into an Array or ArrayList DO NOT CONVERT TO STRING. You should never have to convert a number to a String:

```
import java.util.*;
class Main {
 public static void main(String[] args) {
        int a = 12345;
        ArrayList<Integer> b = new ArrayList<Integer>();
        do{
                b.add(0, a%10);
                //because I'm always taking the last integer, need to add to front of list
                a/=10;
        }while(a>0);
        System.out.print(b);
 }
}
```

A quick way to find the max or min value of 3-4 values:

int a = 3; int b = 4; int c = 5; int d = 7;

int smallest = Math.min(Math.min(a, b), Math.min(c, d));
int largest = Math.max(Math.max(a, b), Math.max(c, d));

STRING FUNCTIONS:

String commands:

String a = "Ms.Graytired";

String b = "Student tired";

- System.out.print(a.charAt(4)); //will print char at index 4, r
- System.out.print(a.compareTo(b)); //compares lexicographics, remember case matters, 0 = strings have same values, positive value = non-alphabetical order (Upper case then lower case); Negative number = alphabetical (Upper case ordered before lowercase), This will give a negative value
- System.out.print(a.equals(b)); //compares string values and returns true/false if they are equal. This will return false.
- System.out.print(a.indexOf("Gray")); //finds index value of string provided returns -1 if not found (Very useful in free response). This returns 3 beware of out of bounds exception error
- System.out.print(b.indexOf("t", 7); //finds index value of a string after a particular index, returns -1 if not found. This will return 8. Will return -1 if not found. Beware of out of bounds exception!
- String[] c = b.split(" "); //will split into an array of 2 items "Student" and "tired". Remember to add back spaces if using in free response
- String[] d = a.split(""); //will split into an array with each letter stored in a new index
- System.out.print(a.substring(3)); //Will print from the index until the end of the string Graytired
- System.out.print(a.substring(3, 6); //Will print from the first index until the second index (non included): gra
- a.trim(); //will trim any leading or spaces at the end of the string
- a.toLowerCase(); a.toUpperCase(); //will convert String case
- System.out.print(a.contains("Hi"); //will return a boolean value if an instance of "Hi" is found in String a

.split()

You can use the .split function to make Free Response easier by partitioning a string into an array. Because it is not in scope for the exam, you must get the syntax correct or they will auto deduct a point.

String t = "This is an example string.";

String[] arr1 = t.split(" "); //will produce array of {"This", "is", "an", "example", "string"}

//It can be helpful when trying to ignore the spaces in a String

//comments

To add comments to free response use // before the comment. They can take off for items that causes compiler errors if it's not commented out (//oops)

Java Escape Sequences ("Special Characters")

\n	newline	Advances the cursor to the next line for subsequent printing
\t	tab	Causes the cursor to skip over to the next tab stop
\b	backspace	Causes the cursor to back up, or move left, one position
\r	carriage return	Causes the cursor to go to the beginning of the current line, not the next line
\\	backslash	Causes a backslash to be printed
\'	single quote	Causes a single quotation mark to be printed
\"	double quote	Causes a double quotation mark to be printed

The compareTo() function can be used for sorting strings in alphabetical order:

String a = “a”;
String b = “b”;
a.compareTo(b); //returns -1 (Unicode for letter “a” – “b”) so String a should be sorted before b
b.compareTo(a); //returns 1 (Unicode for letter “b” – “a”) so String a should be sorted before b
a.compareTo(a); //returns 0 (Strings are EXACTLY the same)

.remove()

String a = “agraya”;
a.replace(“a”, “”); //removes all occurrences of a and replaces with “” making it gry
a = “agraya”
a.replaceFirst(“a”, “e”) //returns egraya

CONDITIONALS:

Logic Operations:

Order of operations:

&& before ||, read from left to right

Example: a = true; b = false; c = true;

a && c || b && a

true || b && a

true || false

true

Conditional formats:

if(Condition){ Code to run if condition is true}

else if(Condition){ Code to run if condition is true, must be attached to an if statement}

else if(Condition){ Code to run if the if statement condition is false, must be attached to an if statement}

Beware of the AP Exam using two if statements in a row instead of an if and else if. Run all if statements when evaluating!

DeMorgan's (There is ALWAYS 1-2 questions that are much better solved using DeMorgan's law and converting):

!(a&&b) = !a || !b

!(a||b) = !a && !b

!(x>2) && !(y==5) = (x<=2) && (y!=5)

!(x>2 && y==5) = (x<=2) || (y!=5)

= vs ==

```
int a;
a=7; //Assigns a value of 7 to integer a
System.out.print(a==8); //prints a true/false statement if a is equal to 8
String b = “Hi”;
System.out.print(b.equals(“Hi”)); //checks if the string values are the same == is for object location for non primitive types
```

LOOPS:

For loops:

```
for(variable for iteration; condition; iteration)
for(int i = 5; i<7; i++)
        System.out.print(i);

//prints 56 and i will no longer exist after the loop is executed

int i = 5;
for( ; i<7; i++){
        System.out.print(i);}
System.out.print(i);
//prints 567 and i will exist with a value of 7 after the loop is executed

while(condition){
}

int i = 5;
while(i<7){
System.out.print(i++);}
//prints 56 and i will exist with a value of 7 after the loop executes
```

//If conditionals or loops do not have {} brackets, then java will simply read up until the first semicolon as part of that conditional or loop

for loops for arrays/arraylists:

```
int[]a = {1, 2, 3, 4};
for(int i = 0; i<a.length; i++)
        System.out.print(a[i]);  //prints 1234
int[][]a = {{1, 2, 3, 4}, {5. 6. 7}};
for(int i = 0; i<a.length; i++){
        for(int j = 0; j<a[i].length; j++)
                System.out.print(a[i][j]);
        System.out.println("\t");}          prints 1234      567
```

For each:

```
String[]b = {"This", "is", "an", "example"};
for(String c:  b)
        System.out.print(c+ " ");  //Prints This is an example

String[][]d = {{"This", "is", "an", "example"}, {"See", "this", "work"}};
for(String[] c:  d)
        for(String e:  c)
                System.out.print(e+ " ");  //Prints This is an example See this work
```

Continue/break:

```
for(int i = 0; i<a.size(); i++)
        {if(a.get(i)==5)  continue;  //will not run the add statement, but will continue with the loop
        else if(a.get(i)==7) break;  //will break out of the internal loop
        a.add(i);}
```

```
if(a.size()==0) return 0; //returns 0 and does not run rest of the method
return a.get(0);  //must have a return statement outside of the loop to ensure there is a return statement for all cases, or compile error
```

METHODS:

Method structure:

Modifier (public static, private, etc) ReturnTypeValue (int[], String, void (nothing returned), etc.) Name (Parameters accepting)

public static void example1 (int[]a, String b){}

//This is a method visible to all classes that uses static variables (no object instance variables). It returns no value, and takes an integer array and String when it is called

private int[] example2()

//This is a non-static method only visible within the class. It will be called with an object (obj.example2()) and can utilize object instance variables. This method does not accept parameters

public Students(String name; double gpa)

//This is a constructor method as it is non-static and has no return type. It accepts variables String name, double gpa. It will be called like this:

Student gray = new Student("Laura Gray", 2.57);

This is an important one to review, as they seem to come up on the multiple choice exam:

```
public static void example(int a, int[]b){
a = 5;
for(int i = 0; i<b.length; i++)
        b[i]=i;}

public static void main(String[]args){
int[] c = {1, 8, 7, 2};
ind d = 9;
example(d, c);
System.out.print(d); //prints 9. Method is void and doesn't return a value
For(int e: c)
        System.out.print(e);//prints 0, 1, 2, 3; Arrays and Arraylists are permanently changed, unless I place them into a new arraylist. This is the case for all objects except Strings and primitive types
```

The Dreaded Instance Variables:

They are non-static, so methods that use them must be non-static. We must create objects and call the methods with the object names!

```
public class Student{
        private String name = "";
        private String subject = "";
        private int grade = 0;
        private Teacher teach;
        //MAKE ALL INSTANCE VARIABLES PRIVATE!!!

        //THIS IS THE CONSTRUCTOR METHOD
        public Student(String a, String b, int c, Teacher d){
                name = a;
                subject = b;
                grade = c;
                Teacher = d;
        }

        //accessor methods
        public int getAPGrade(){return grade + 10;}
        public int getgrade(){return grade;}
        public String getname(){return name;}
        public Teacher getteacher(){return teach;}
        public String getsubject(){return subject;}

        //mutator method
        public String getName (){name = name.toUpperCase();}}

        public class Teacher{
        private String name = "";
        private double room = 0;

        //constructor
        public Teacher(String name, double room){
        this.name = name;
        this.room = room;}
        /accessor
        public String getName(){return name;}
        public double getRoom(){return room;}}


         public class main{
         public static void main(String[]args){
         Teacher gray = new Teacher("Gray", 5.342);
         Student you = new Student("Your Name", "APCSA", 84, gray);
         System.out.print(you.getAPGrade());//prints 94
         System.out.print(you.getgrade ());//prints 84
         System.out.print(you.getteacher().getName);//prints Gray}
```

//You can have more than one constructor method (that accepts different variables via method overloading. Also, the public Student() method exists even though it's not coded here, it will create object instance variables with default values

Method overloading:

```
public static void methoda(int a){System.out.print(a);}
public static void methoda(int a, int b){System.out.print(b-a);)
public static void main(String[]args){
        int c = 3;
        int d = 7;
        methoda(c); //will print 3
        methoda(c, d); //will print 4 (7-3)
```

If a method returns something, make sure you return that object type. That's a FULL POINT, *SEAN EVA!!!!!!!!*

ALWAYS have a return statement outside of conditionals and loops. If your loop or conditional never executes, you will have an error as your method has nothing to return (and you will lose a point!)

```
public static void main2(String[] a, String b)
    {       for(String c:  a)
                If(c.equals(b){
                    return true;}
                else {return false;}
    }
```

The above will cause a compiler error b/c there is not a guaranteed return statement. What if the array a is null?

The above will cause a compiler error b/c there is not a guaranteed return statement. What if the array a is null?

Java will not read code in a method after a return:

```
if(a>b)
{return a;
a+=b;  //will never be read as the method has returned}
```

You can also get out of a method that is void by simply using return:

```
public static void test(int a)
{
if(a<0) return;
System.out.print("Positive value");
}
```

ARRAYS/ARRAYLISTS:

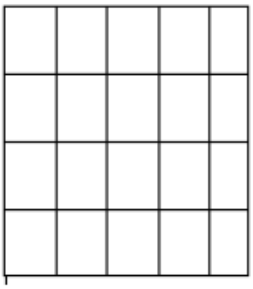
Finding the size/length of Strings, Arrays, and ArrayLists:

Strings:

```
String a = "Hello";
System.out.println(a.length() ) ; //Prints 5;
//Index of 0 to a.length()-1 (0 to 4)
```

Arrays:

```
int[][] b = new int[4][5]  //Creates a 4 row by 5 column Array
System.out.println(b.length);  //Prints row length of 4
System.out.println(b[0].length);  //Prints row 0 column length of 5
//Row Index of 0 to b.length-1 (0 to 3)
//Column 1  Index of 0 to b[1].length-1 (0 to 4)
//2D Arrays are simply a series of rows, each containing a new array
```



ArrayLists:

```
List<Integer> c = new ArrayList<Integer>();
System.out.println(c.size( ) )  //Prints the length of the ArrayList
//Index of 0 to c.size()-1 (0 to ?)
```

ALWAYS CHECK MULTIPLE CHOICE FOR POTENTIAL OUT OF BOUND INDEX ISSUES, ESPECIALLLY WHEN THEY ARE USING FOR/WHILE LOOPS. ALSO CHECK YOUR OWN FREE RESPONSE TO ENSURE THERE ARE NO OUT OF BOUNDS EXCEPTION ERRORS WITH STRING, ARRAY, AND ARRAYLIST INDEXES!

ARRAYS STORE DEFAULT VALUES (O, null, false) ARRAYLISTS DO NOT!

Be careful when writing commands for Arrays vs ArrayLists in free response. Misuse of commands is an automatic point off!

Remember, you must set a size of your Arrays, and ArrayLists sizes can be constantly changed/built. If you are writing a free response in which the sizing will change (items will be added/removed) then use an ArrayList (even if they send an Array, instantiate an ArrayList and use a for or while loop to populate the ArrayList).

ArrayList Commands:

c.get(0);	//returns value at index 0
c.add(1);	//adds 1 to the end of the array, returns true if added
c.add(0, 1);	//adds 1 at index 0 to the array, everything at index 0 and beyond will move 1 position to the right, returns true if added
c.set(2, 4);	//changes the value at index 2 to a value of 4. Returns value it replaced
c.remove("yo");	//removes the first instance of the String "yo" that it finds. All items after "yo" will be shifted to the left. Returns the value that was removed
c.remove(4);	//removes the item at index 4. Shifts all items at index 5 and beyond to the left 1 index. Returns the value it removed

Be VERY careful with the remove function in loops! There will be 1 multiple choice question on this, and you want to ensure you don't make this mistake on free response as it will be a point deduction. When writing a loop to remove items, work from the back to the front to avoid this error! See below:

```
public static void main(String[]args){
ArrayList<String> color = new ArrayList<>();
color.add("red");
color.add("red");
color.add("green");
color.add("red");
color.add("red");
color.add("green");
color.add("yellow");

for(int i = 0; i<color.size(); i++){
if(color.get(i).equals("red")){
color.remove(i); }}}
```

Index	0	1	2	3	4	5	6
Value	red	red	green	red	red	green	yellow

When i=0:

Index	0	1	2	3	4	5	6
Value	red	green	red	red	green	yellow	

When i=1:

Index	0	1	2	3	4	5	6
Value	red	green	red	red	green	yellow	

When i=2:

Index	0	1	2	3	4	5	6
Value	red	green	red	green	yellow		

When i=3:

Index	0	1	2	3	4	5	6
Value	red	green	red	green	yellow		

When i=4:

Index	0	1	2	3	4	5	6
Value	red	green	red	green	yellow		

```
public static void main(String[]args){
ArrayList<String> color = new ArrayList<>();
color.add("red");
color.add("red");
color.add("green");
color.add("red");
color.add("red");
color.add("green");
color.add("yellow");

for(int i = 0; i<color.size(); i++){
if(color.get(i).equals("red")){
color.remove(i);
i--; }}}
```

Index	0	1	2	3	4	5	6
Value	red	red	green	red	red	green	yellow

When i=0:

Index	0	1	2	3	4	5	6
Value	red	green	red	red	green	yellow	

When i=0:

Index	0	1	2	3	4	5	6
Value	green	red	red	green	yellow		

When i=0:

Index	0	1	2	3	4	5	6
Value	green	red	red	green	yellow		

When i=1:

Index	0	1	2	3	4	5	6
Value	green	red	green	yellow			

When i=1:

Index	0	1	2	3	4	5	6
Value	green	green	yellow				

When i=2:

Index	0	1	2	3	4	5	6
Value	green	green	yellow				

Arrays:

```
System.out.print(a[5]); //prints item at index 5 in the array

a[3]=5;  //sets value to 5 at index 3

System.out.print(a[1][3])  //prints value of row 1, column 3 in array a
```

int[][] a = new int[3][4]
int[]b = a[2]; (saves a 1 dimensional array equal to all of the values in row 2, no need for a loop!)

SEARCHING/SORTING ALGORITHMS:

Big hint to save time. They usually give you the type of the sorting array in multiple choice questions along with a long segment of code. If you already know how the algorithm works, you just need to check if it is sorting from lowest to highest or highest to lowest. After that, ignore the code!

Selection Sort: # of comparisons made (n-1)(n)/2

Insertion Sort: Best case: List is already ordered (more efficient) 0(n): Worst Case: List is in reverse O(n^2)

Merge Sort: Generally more efficient than other sorts Best O(n), Worst O(log n)

**You will likely not have to calculate run times with log n!

Selection Sort: When sorting lowest to highest, finds the lowest value and swaps it with value in index 0, then finds next lowest and swaps with index 1, etc.

	2	1	8	3	4
1st Iteration	1	2	8	3	4
2nd Iteration	1	2	8	3	4
3rd Iteration	1	2	3	8	4
4th Iteration	1	2	3	4	8

Selection sort: Look for a code that has a loop that starts at the first item in the list and an embedded loop that checks all items in the array/arraylist after that item (j=i+1) it will find the highest or lowest value and then perform a swap

```
public static void sort(int[] x)
{
   for (int i = 0; i < x.length - 1; i++)
   {
      int minIndex = i;

      for (int j = i + 1; j < x.length; j++)
         if (x[j] < x[minIndex]) minIndex = j;

      swap(x, i, minIndex);
   }
}

private static void swap(int[] x, int i, int j)
{
   int temp = x[i];
   x[i] = x[j];
   x[j] = temp;
}
```

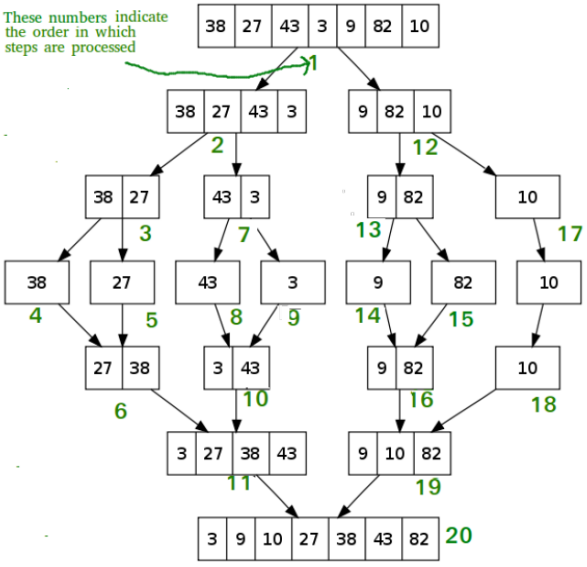
Insertion Sort: Sorts the first 2 elements, sorts the first 3 elements, sorts the first 4 elements, etc.

	2	1	8	3	4
1st Iteration	1	2	8	3	4
2nd Iteration	1	2	8	3	4
3rd Iteration	1	2	3	8	4
4th Iteration	1	2	3	4	8

Insertion sort: Look for a code that has a loop that starts at the second item in the list and an embedded list that checks all items before it in the array/arraylist (j-1) it will stop when the value is greater than the item and perform a swap.

```
public static void sort(int[] x)
{
   for (int i = 1; i < x.length; i++)
   {
      int next = x[i];
      int j = i;

      while (j > 0 && next < x[j - 1])
      {
         x[j] = x[j - 1];
         j--;
      }

      x[j] = next;
   }
}
```

Merge Sort: Performed through recursion.




```
private static void sort(int[] a, int low, int high)
   {
      if (low < high)
      {
         int mid = (low + high) / 2;
         sort(a, low, mid);
         sort(a, mid + 1, high);
         merge(a, low, mid, high);
      }
   }
```

Linear or Sequential Search: List does not have to be ordered. Goes through each element in order and searches for item.

```
for(int i = 0; i<c.size(); i++){
        if(c.get(i).equals("search"))  return true;}
return false;
```

```
public static int sequentialSearch(int[] x, int key)
{
 for (int i = 0; i < x.length; i++)
       if (x[i] == key) return i;
 return -1;
}
```

Binary Search: List MUST be ordered to perform a binary search.

Formula to find index to check: (min + max)/2

index	0	1	2	3	4	5	6	7
Value	3	4	8	17	22	23	40	42

To find 37:

1	min=0; max =7 (0+7)/2 = 3	17!=37 17<37, so the min changes Index 3+1
2	min=4; max = 7 (4+7)/2 = 5	23!=37 23<37, so the min changes Index 5 +1
3	min=6; max = 7 (6+7)/2 = 6	40!=37 40>37, so the max changes Index 6-1
4	min=6; max = 5 max<min, so the loop stops and generally returns -1	

To find 8:

1	min=0; max =7 (0+7)/2 = 3	17!=8 17>8, so the max changes Index 3-1
2	min=0; max = 2 (0+2)/2 = 1	4!=8 4<8, so the min changes index 1+1
3	min=2; max = 2 (2+2)/2 = 2	8==8 So the loop would stop and return index 2

The match number of searches a binary search will conducted is binary. For example, the max number of indexes a binary will check for an array of 500 records Merge sorts will use recursion and identify a max, min, and mid index:

500/2 = 250

250/2 = 125

125/2 = 63

63/2 = 32

32/2 = 16

16/2 = 8

8/2 = 4

4/2 = 2

2/1 = 1

9 index checks will occur in this binary search!

For max searches in binary given a record count, convert to Binary and count the number of digits. That is your answer

```
public static int binarySearch(int[] x, int key)
{
  return binarySearch(x, key, 0, x.length - 1);
}

private static int binarySearch(int[] x, int key, int start, int end)
{
  if (start > end)
    return -start - 1;
  else
  {
    int mid = (start + end) / 2;
    int result = key - x[mid];

    if (result < 0)
    { return binarySearch(x, key, start, mid - 1); }
    else
    {
      if(result > 0)
        return binarySearch(x, key, mid + 1, end);
      else return mid;
    }
  }
}
```

INHERITANCE/POLYMORPHISM (Overriding, Overloading in Methods):

```
Inheritance:  Keyword extends!

public class Student{

        private String name = "";
        private int grade = 0;


public Student(String a, int b){
        name = a;
        grade = b;
        }

        //accessor methods
        public int getgrade(){return grade;}
        public String getname(){return name;}
        public void print(){System.out.print(name);}}

public class Senior extends Student{  //children can only extend to ONE parent
        private boolean sce = false;

        public Senior(String a, int b, boolean c){
                super(a, b);  //Must construct parent objects FIRST!
                sce = c;
        }

        public int getgrade(){return super.getgrade() + 10;}

        public boolean getSCE(){return sce;}}
public class main{

public static void main(String[]args){

Senior a = new Senior("Jessica", 80, true);
System.out.print(a.getSCE()); //prints true
System.out.print(a.getgrade()); //prints 90
System.out.print(a.print()); //prints Jessica

Student b = new Senior("Andy", 90, true);
System.out.print(b.getSCE()); //will fail.  This method is not listed in Student and cannot be overwritten, you'd have to cast the object to a senior first!
System.out.print(b.getgrade()); //prints 100
System.out.print(b.print()); //prints Andy

Student c = new Student("Laura", 63);
System.out.print(c.getSCE()); //will fail.  This method is not listed in Student!
System.out.print(c.getgrade()); //prints 63
System.out.print(c.print()); //prints Laura}}

    Senior d = new Student(....///just stop you CANNOT do this.  We can't create a parent at the child class level.
```

//Children can only have 1 parent, but can implement multiple interfaces. Parents can have multiple children.

//When methods (print statements, etc) are different for children, create a local method to incorporate the changes, never create instance variables that exist, use method calls to parent objects to obtain and use them. You can also use things you need in parent method by using the super.methodname() call. If the child behaves exactly as the parent, don't implement another method in the child on free response, method overriding will take care of it for you. Only implement methods required (abstract methods listed in abstract parent or implemented interface) or methods that are the same name but behave differently (method overriding)

10-13-1: Given the following class declarations, what is the output from `Student s1 = new GradStudent();` followed by `s1.getInfo();` ?

```
public class Student {
   public String getFood() {
      return "Pizza";
   }
   public String getInfo()  {
      return this.getFood();
   }
}

public class GradStudent extends Student {
   public String getFood() {
      return "Taco";
   }
}
```

- (A) Won't compile since `GradStudent` doesn't have a `getInfo` method
- (B) Taco
- (C) Pizza
- (D) Won't compile since you are creating a `GradStudent` , not a `Student`
- (E) Won't compile since you use `this.getFood()`

This is a GradStudent object that is cast at the Student class. It validates the getInfo method exists in Student first. There is no method that overrides getInfo so it runs the one in student that it inherits. That calls getFood() the object is cast as a Student, so it validates the getFood() method exists in Student, but then overrides it with the GradStudent getFood() method b/c that object is still a GradStudent object.

B. Taco

10-13-3: Given the following class declarations and initializations in a client program, which of the following is a correct call to `method1` ?

```
public class Test1
{
   public void method1(Test2 v1, Test3 v2)
   {
      // rest of method not shown
   }
}

public class Test2 extends Test1
{
}

public class Test3 extends Test2
{
}

The following initializations appear in a different class.
Test1 t1 = new Test1();
Test2 t2 = new Test2();
Test3 t3 = new Test3();
```

- (A) t1.method1(t1,t1);
- (B) t2.method1(t2,t2);
- (C) t3.method1(t1,t1);
- (D) t2.method1(t3,t2);
- (E) t3.method1(t3,t3);

We can cast child objects (sub classes) up to a parent level (super class) but we cannot take an object instantiated at a parent level and cast down to a child without instantiating (keyword new) again.

Therefore answer is E

10-14-1: Assume that `Base b = new Derived();` appears in a client program. What is the result of the call `b.methodOne();` ?

```
public class Base
{
   public void methodOne()
   {
      System.out.print("A");
      methodTwo();
   }

   public void methodTwo()
   {
      System.out.print("B");
   }
}

public class Derived extends Base
{
   public void methodOne()
   {
      super.methodOne();
      System.out.print("C");
   }

   public void methodTwo()
   {
      super.methodTwo();
      System.out.print("D");
   }
}
```

- (A) ABDC
- (B) AB
- (C) ABCD
- (D) ABC

The correct answer is ABDC

This method is instantiated as a Derived object but cast as a Base.

It checks that methodOne exists in Base, then overrides it. It calls super, which prints A and calls methodtwo()

It validates methodTwo exists in base and again overrides it from the child b/c it is a Derived object. It will call the super MethodTwo() and print B. Then it will finish running the child overriding methodTwo and print D. It then needs to return to the overriding Method One and finish by printing C

INTERFACES AND ABSTRACT CLASSES:

Abstract classes:

Can contain instance variables and static/non-static methods. Cannot be instantiated:

```
public abstract class Student{
        private String name = "";
        private int grade = 0;

public Student(String a, int b){
        name = a;
        grade = b;
        }

        //accessor methods
        public int getgrade(){return grade;}
        public String getname(){return name;}
        public void print(){System.out.print(name);}
        public abstract boolean getSCE(); //abstract method is just listed, meaning it must be defined in the child class!}

public class Senior extends Student{
        private boolean sce = false;

        public Senior(String a, int b, boolean c){
                super(a, b);  //Must construct parent objects FIRST! super() is implied if not coded and will run parent constructor first!
                sce = c;
        }

        public int getgrade(){return super.getgrade() + 10;}

        public boolean getSCE(){return sce;}}

public class main{

public static void main(String[]args){

Senior a = new Senior("Jessica", 80, true);
System.out.print(a.getSCE()); //prints true
System.out.print(a.getgrade()); //prints 90
System.out.print(a.print()); //prints Jessica

Student b = new Senior("Andy", 90, true);
System.out.print(b.getSCE()); //will fail.  This method is not listed in Student to be overrided
System.out.print(b.getgrade()); //prints 100
System.out.print(b.print()); //prints Andy

Student c = new Student("Laura", 63);//will fail, cannot instantiate objects in abstract classes

Senior d = new Student(....///just stop you CANNOT do this.  We can't create a parent at the child class level.}}
```

Interfaces:

Interfaces cannot implement instance variables!!!!!!!!

```
public interface EatFood(){
        void eatSnack();  //all interface methods listed are public and abstract the MUST be programmed in classes that implement them
        String[] goodFood();
}
public class SchoolFood implements EatFood, SnackFood //Snackfood interface not shown
{
        private String[] snacks;

        public SchoolFood(String[] snacks){
                this.snacks = snacks;}

        public void eatSnack(){
                int a = (int)(Math.random()*snacks.length);
                System.out.print(snacks[a]);}

        public String[] goodfood(){
                return snacks;}

        public String[] print(){
                if(snacks.length>0) System.out.print(snacks[0];}}


public class main{
        public static void main(String[]args){
                String[] food = {"coke", "Cheetos"};
                EatFood example = new SchoolFood(food);//because we have implemented the interface with this object, we can only
run the eatSnack() and goodFood() methods.  We cannot execute the print() method.  The object cannot see it.
```

Even if it is not shown, super constructors are ALWAYS called first, even when instantiating a subclass or child. It is implied that super(); is the first line in the constructor, if it is not written in the code.

RECURSION:

Recursion:

```
public static void printSeriesInReverse(int n){
  if (n > 0){
    System.out.print(n);
    printSeriesInReverse(n-1);
  }
```

```
printSeriesInReverse(int 4); //prints 4321
```

```
public static void printSeriesInReverse(int n){
  if (n > 0){
    printSeriesInReverse(n-1);
    System.out.println(n);
}
```

```
printSeriesInReverse(int 4); //prints 1234
```

```
public static void printSeriesInReverse(int n){
  if (n > 0){
            System.out.println(n);
    printSeriesInReverse(n-1);
    System.out.println(n);
}
```

```
printSeriesInReverse(int 4); //prints 43211234
```

There will be one exactly like the one above on the exam!!!!!!!!

11-7-3: Given the following method declaration, what value is returned as the result of the call `product(5)` ?

```
public int product(int n)
{
   if (n <= 1)
      return 1;
   else
      return n * product(n - 2);
}
```

- a) 1
- b) 10
- c) 25
- d) 3125
- e) 15

product(5)→
  5*product(3)→
    5* product(1)→
      1
=5*5*1 = 25

11-7-2: Given the following method declaration, what value is returned as the result of the call `mystery(5)` ?

```
public static int mystery(int n)
{
   if (n == 0)
      return 1;
   else
      return 3 * mystery (n - 1);
}
```

- a) 243
- b) 0
- c) 3
- d) 81
- e) 27

mystery(5)→
3*mystery(4)→
3* mystery(3)→
3* mystery(2)→
3* mystery(1)→
3* mystery(0)→1
=3*3*3*3*3*1 = A

11-8-1: Given the following method declaration, this method will return true if and only if:

```
public boolean check(String s)
{
   return s.length() >= 2 && (s.charAt(0) == s.charAt(1) || check(s.substring(1)));
}
```

- a) The string `s` contains two or more of the same characters.
- b) The string `s` starts with two or more of the same characters.
- c) The string `s` contains two or more of the same character that are next to each other.
- d) The string `s` ends with two or more of the same characters

C

11-8-2: Given the following method declaration, what will `redo(82, 3)` return?

```
public int redo(int i, int j)
{
   if (i==0)
      return 0;
   else
      return redo(i/j, j)+1;
}
```

a) 5

b) 4

c) 6

d) 7

e) The method never returns due to infinite recursion.

Redo(82, 3)→
Redo(82/3=9, 3)+1→
Redo(9/3=3, 3)+1→
Redo(3/3=1, 3)+1→
Redo(1/3=0, 3)+1→0
=1+1+1+1+0=4
B

Consider the following method:

public int recur(int n){

if (n<12) return n*3;
else return recur(recur(n/4));}

What value is returned as a result of the call recur(33)?

recur(33)->
=recur(recur(33/4 = 8)=recur(recur(8)) (recur(8) returns 24 from base case) = recur(24)

recur(24)->
=recur(recur(24/4 = 6) = recur(recur(6)) (recur(6) returns 18) = recur(18)

recur(18)->
=recur(recur(18/4 = 4) = recur(recur(4)) (recur(4) returns 12) = recur(12)

recur(12)->
=recur(recur(12/4 = 3) = recur(recur(3)) (recur(3) returns 9) = recur(9)

recur(9) returns 27 per the base case, so this entire thing returns 27!