AP Computer Science A Practice Exam 1

Multiple-Choice Questions

ANSWER SHEET

1	A	B	C	D	E		21	A	B	C	D	E
2	A	B	C	D	E		22	A	B	C	D	E
3	A	B	C	D	E		23	A	B	C	D	E
4	A	B	C	D	E		24	A	B	C	D	E
5	A	B	C	D	E		25	A	B	C	D	E
6	A	B	C	D	E		26	A	B	C	D	E
7	A	B	C	D	E		27	A	B	C	D	E
8	A	B	C	D	E		28	A	B	C	D	E
9	A	B	C	D	E		29	A	B	C	D	E
10	A	B	C	D	E		30	A	B	C	D	E
11	A	B	C	D	E		31	A	B	C	D	E
12	A	B	C	D	E		32	A	B	C	D	E
13	A	B	C	D	E		33	A	B	C	D	E
14	A	B	C	D	E		34	A	B	C	D	E
15	A	B	C	D	E		35	A	B	C	D	E
16	A	B	C	D	E		36	A	B	C	D	E
17	A	B	C	D	E		37	A	B	C	D	E
18	A	B	C	D	E		38	A	B	C	D	E
19	A	B	C	D	E		39	A	B	C	D	E
20	A	B	C	D	E		40	A	B	C	D	E

AP Computer Science A Practice Exam 1

Part I (Multiple Choice)

Time: 90 minutes
Number of questions: 40
Percent of total score: 50

Directions: Choose the best answer for each problem. Use available space for scratch work and hand tracing. Some problems take longer than others. Consider how much time you have left before spending too much time on any one problem.

Notes:

- You may assume all import statements have been included where they are needed.
- You may assume that the parameters in method calls are not null and the methods are called when their preconditions are met.
- You may assume that declarations of variables and methods appear within the context of an enclosing class.
- You may refer to the Java Quick Reference sheet as needed.

1. Consider the following code segment.

```
int myValue = 17;
int multiplier = 3;
int answer = myValue % multiplier + myValue / multiplier;
answer = answer * multiplier;
System.out.println(answer);
```

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

(A) `9`

(B) `10`

(C) `7`
(D) `30`
(E) `21`

2. Assume that `a`, `b`, and `c` have been declared and correctly initialized with `int` values. Consider the following expression.

```
boolean bool = !(a < b || b <= c) && !(a < c || b >= a);
```

Under what conditions does `bool` evaluate to true?

(A) `a = 1, b = 2, c = 3`
(B) `a = 3, b = 2, c = 1`
(C) `a = 3, b = 1, c = 2`
(D) All conditions; `bool` is always `true`.
(E) No conditions; `bool` is always `false`.

3. Consider the following code segment.

```
int[] myArray = {2, 3, 4, 1, 7, 6, 8};
int index = 0;
while (myArray[index] < 7)
{
    myArray[index] += 3;
    index++;
}
```

What values are stored in `myArray` after executing the code segment?

(A) `{2, 3, 4, 1, 7, 6, 8}`
(B) `{2, 3, 4, 1, 7, 9, 11}`
(C) `{5, 6, 7, 4, 7, 9, 8}`
(D) `{5, 6, 7, 4, 7, 6, 8}`
(E) `{5, 6, 7, 4, 0, 9, 11}`

4. Consider the following class used by a company to represent the items it has available for online purchase.

```
public class OnlinePurchaseItem
{
          public double getPrice()          // implementation not shown
          public String getItem()           // implementation not shown
          public String getMonth()          // implementation not shown
                                            // instance variables, constructors, and
                                            // other methods not shown
}
```

The company bills at the end of the quarter. Until then, it uses an `ArrayList` of `OnlinePurchaseItem` objects to track a customer's purchases.

```
private ArrayList<OnlinePurchaseItem> items;
```

The company decides to offer a 20-percent-off promotion on all items purchased in September. Which of the following code segments properly calculates the correct total price at the end of the quarter?

I.
```
double total = 0.0;
for (int i = 0; i < items.size(); i++)
{
   if (items.get(i).getMonth().equals("September"))
      total += 0.80 * items.get(i).getPrice();
   else
      total += items.get(i).getPrice();
}
```

II.
```
double total = 0.0;
for (OnlinePurchaseItem purchase : items)
{
   if (purchase.get(i).getMonth().equals("September"))
      total = total + 0.80 * purchase.getPrice();
   else
      total += items.get(i).getPrice();

}
```

III.
```
double total = 0.0;
for (int i = items.size(); i >= 0; i--)
{
   if (items.get(i).getMonth().equals("September"))
      total = total + 0.80 * items.get(i).getPrice();
   else
      total += items.get(i).getPrice();
}
```

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

5. Consider the following method.

```
public int mystery(int n)
{
    if (n <= 1)
        return 1;
    return 2 + mystery(n - 1);
}
```

What value is returned by the call `mystery(5)`?

(A) `1`
(B) `7`
(C) `8`
(D) `9`
(E) `10`

6. Consider the following code segment.

```
String myString = "H";
int index = 0;
while (index < 4)
{
    for (int i = 0; i < index; i++)
        myString = "A" + myString + "A";
    index ++;
}
```

What is the value of `myString` after executing the code segment?

(A) `"H"`
(B) `"AHA"`
(C) `"AAAHAAA"`
(D) `"AAAAHAAAA"`
(E) `"AAAAAAHAAAAAA"`

7. Consider the following statement.

```
int var = (int)(Math.random() * 50) + 10;
```

What are the possible values of `var` after executing the statement?

(A) All integers from 1 to 59 (inclusive)
(B) All integers from 10 to 59 (inclusive)

(C) All integers from 10 to 60 (inclusive)
(D) All real numbers from 50 to 60 (not including 60)
(E) All real numbers from 10 to 60 (not including 60)

8. Consider the following statement.

```
System.out.print(13 + 6 + "APCSA" + (9 - 5) + 4);
```

What is printed as a result of executing the statement?

(A) `136APCSA(9 - 5)4`
(B) `19APCSA8`
(C) `19APCSA44`
(D) `136APCSA8`
(E) `136APCSA44`

9. Assume `truth1` and `truth2` are `boolean` variables that have been properly declared and initialized.

Consider this expression.

```
(truth1 && truth2) || ((!truth1) && (!truth2))
```

Which expression below is its logical equivalent?

(A) `truth1 != truth2`
(B) `truth1 || truth2`
(C) `truth1 && truth2`
(D) `!truth1 && !truth2`
(E) `truth1 == truth2`

10. Consider the following method.

```
/** Precondition: numbers.size() > 0
 */
public int totalValue(ArrayList<Integer> numbers)
{
    int total = 0;
    for (Integer val : numbers)
    {
        if (val > 1 && numbers.size() - 3 > val)
            total += val;
    }
    return total;
}
```

Assume that the `ArrayList` passed as a parameter contains the following `Integer` values.

`[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]`

What value is returned by the call `totalValue`?

(A) `20`
(B) `21`
(C) `27`
(D) `44`
(E) `45`

Questions 11–14 refer to the following class definition.

```
public class Depot
{
   private String city;
   private String state;
   private String country = "USA";
   private boolean active = true;

   public Depot(String myCity, String myState, String myCountry, boolean myActive)
   {
      city = myCity;
      state = myState;
      country = myCountry;
      active = myActive;
   }

   public Depot(String myState, String myCity)
   {
      state = myState;
      city = myCity;
   }

   public Depot(String myCity, String myState, boolean myActive)
   {
      city = myCity;
      state = myState;
      active = myActive;
   }

   public String toString()
   {
      return city + ", " + state + " " + country + " " + active;
   }

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

11. The `Depot` class has three constructors. Which of the following is the correct term for this practice?

    (A) Overriding
    (B) Procedural abstraction
    (C) Encapsulation
    (D) Polymorphism
    (E) Overloading

12. Consider the following code segment in another class.

```
Depot station = new Depot("Oakland", "California");
System.out.println(station);
```

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

    (A) `California, Oakland USA true`

(B) `California, Oakland USA active`
(C) `USA, California USA true`
(D) `Oakland, California USA active`
(E) `Oakland, California USA true`

13. Consider the following class definition.

```
public class WhistleStop extends Depot
```

Which of the following constructors compiles without error?

I.
```
public WhistleStop()
{
    super();
}
```

II.
```
public WhistleStop()
{
    super("Waubaushene", "Ontario", "Canada", true);
}
```

III.
```
public WhistleStop(String city, String province)
{
    super(city, province);
}
```

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

14. Assume a correct no-argument constructor has been added to the `WhistleStop` class.

Which of the following code segments compiles without error?

I.
```
ArrayList<Depot> stations = new ArrayList<Depot>();
stations.add(new WhistleStop());
stations.add(new Depot("Mansonville", "Quebec", "Canada", false));
```

II.
```
ArrayList<WhistleStop> stations = new ArrayList<Depot>();
stations.add(new WhistleStop());
stations.add(new Depot("Orinda", "California", true));
```

III.
```
ArrayList<WhistleStop> stations = new ArrayList<WhistleStop>();
stations.add(new WhistleStop());
stations.add(new Depot("Needham", "Massachusetts", true));
```

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

15. Consider the following code segment.

```
String crazyString = "crazy";
ArrayList<String> crazyList = new ArrayList<String>();
crazyList.add("weird");
crazyList.add("enigma");
for (String s : crazyList)
{
   crazyString = s.substring(1, 3) + crazyString.substring(0, 4);
}
System.out.println(crazyString);
```

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

(A) `nicraz`
(B) `nieicr`
(C) `nieicraz`
(D) `einicraz`
(E) `weienicraz`

16. Consider the following method.

```
public int pathways(int n)
{
    int ans;
    if (n < -5)
       ans = 1;
    else if (n < 0)
       ans = 2;
    else if (n > 10)
       ans = 3;
    else
       ans = 4;
    return ans;
}
```

Which of the following sets of data tests every possible path through the code?

(A) `-6, -1, 15, 12`
(B) `-5, -3, 12, 15`
(C) `-8, -5, 8, 10`
(D) `-6, 0, 20, 7`
(E) `-10, -5, 10, 12`

Questions 17–18 refer to the following scenario.

A resort wants to recommend activities for its guests, based on the temperature (degrees F) as follows:

76° and above	go swimming
61°–75°	go hiking
46°–60°	go horseback riding
45° and below	play ping pong

17. Consider the following method.

```
public String chooseActivity(int temperature)
{
    String activity;
    if (temperature > 75)
       activity = "go swimming";
    if (temperature > 60)
       activity = "go hiking";
    if (temperature > 45)
       activity = "go horseback riding";
    else
       activity = "play ping pong";
    return activity;
}
```

Consider the following statement.

```
System.out.println("We recommend that you " +
chooseActivity(temperature));
```

For which temperature range is the correct suggestion printed (as defined above)?

(A) `temperature > 75`

(B) `temperature > 60`

(C) `temperature > 45`

(D) `temperature <= 60`

(E) Never correct

18. After discovering that the method did not work correctly, it was rewritten as follows.

```
public String chooseActivity(int temperature)
{
    String activity;
    if (temperature > 45)
       activity = "go horseback riding";
    else if (temperature > 60)
       activity = "go hiking";
    else if (temperature > 75)
       activity = "go swimming";
    else
       activity = "play ping pong";
    return activity;
}
```

Consider the following statement.

```
System.out.println("We recommend that you " +
chooseActivity(temperature));
```

What is the largest temperature range for which the correct suggestion is printed (as defined above)?

(A) Always correct

(B) Never correct

(C) `temperature <= 75`

(D) `temperature <= 60`

(E) `temperature <= 45`

19. Assume `int[] arr` has been correctly instantiated and is of sufficient size. Which of these code segments results in identical arrays?

I.
```
int i = 1;
while (i < 6)
{
    arr[i / 2] = i;
    i += 2;
}
```

II.
```
for (int i = 0; i < 3; i++)
{
    arr[i] = 2 * i + 1;
}
```

III.
```
for (int i = 6; i > 0; i = i - 2)
{
    arr[(6 - i) / 2] = 6 - i;
}
```

(A) I and II only
(B) II and III only
(C) I and III only
(D) I, II, and III
(E) All three arrays are different.

20. Consider the following code segment.

```
ArrayList<String> nations = new ArrayList<String>();
nations.add("Argentina");
nations.add("Canada");
nations.add("Australia");
nations.add("Cambodia");
nations.add("Russia");
nations.add("France");
for (int i = 0; i < nations.size(); i++)
{
    if (nations.get(i).length() >= 7)
       nations.remove(0);
}
System.out.println(nations);
```

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

(A) `[Canada, Russia, France]`

(B) `[Cambodia, Russia, France]`

(C) `[Australia, Cambodia, Russia, France]`

(D) `[Canada, Cambodia, Russia, France]`

(E) Nothing is printed. `IndexOutOfBoundsException`

21. Consider the following class used to represent a student.

```
public class Student
{
    private String name;
    private int year;

    public Student()
    {
        name = "name";
        year = 0;
    }

    public Student(String myName, int myYear)
    {
        name = myName;
        year = myYear;
    }
}
```

Consider the `ExchangeStudent` class that extends the `Student` class.

```
public class ExchangeStudent extends Student
{
    private String country;
    private String language;

    public ExchangeStudent(String myName, int myYear,
                           String myCountry, String myLanguage)
    {
        super(myName, myYear);
        country = myCountry;
        language = myLanguage;
    }
}
```

Which of the following constructors could also be included in the `ExchangeStudent` class without generating a compile-time error?

I.
```
public ExchangeStudent(String myName, int myYear, String myCountry)
{
    super(myName, myYear);
    country = myCountry;
    language = "English";
}
```

II.
```
public ExchangeStudent(String myCountry, String myLanguage)
{
    super("name", "2015");
    country = myCountry;
    language = myLanguage;
}
```

III.
```
public ExchangeStudent()
{
}
```

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

22. Consider the following method.

```
public int countLetters(String word, String letter)
{
    int count = 0;
    for (int index = 0; index < word.length(); index++)
    {
        if ( /*  missing code  */ )
            count++;
    }
    return count;
}
```

What could replace `/* missing code */` to allow the method to return the number of times `letter` appears in `word`?

(A) `word.substring(index).equals(letter)`

(B) `word.substring(index, index + 1) == letter`

(C) `word.indexOf(letter) == letter.indexOf(letter)`

(D) `word.substring(index, index + 1).equals(letter)`

(E) `letter.equals(word.substring(index).indexOf(letter))`

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

`["okapi", "aye-aye", "cassowary", "echidna", "sugar glider", "jerboa"]`

Consider the following code segment.

```
for (int i = 0; i < obscureAnimals.size(); i++)
{
   if (obscureAnimals.get(i).compareTo("pink fairy armadillo") < 0)
      obscureAnimals.remove(i);
}
System.out.print(animals);
```

What will be printed as a result of executing the code segment?

(A) `[]`

(B) `[sugar glider]`

(C) `[aye-aye, echidna, sugar glider]`

(D) `[aye-aye, echidna, sugar glider, jerboa]`

(E) Nothing will be printed. There is an `ArrayListIndexOutOfBoundsException`.

24. Consider the following method.

```
public boolean whatDoesItDo(String st)
{
   for (int i = 0; i < st.length() / 2; i++)
   {
      String sub1 = st.substring(i, i + 1);
      String sub2 = st.substring(st.length() - i - 1, st.length() - i);
      if (!sub1.equals(sub2))
      {
         return false;
      }
   }
   return true;
}
```

What is the purpose of the method `whatDoesItDo`?

(A) The method returns `true` if `st` has an even length, `false` if it has an odd length.

(B) The method returns `true` if `st` contains any character more than once.

(C) The method returns `true` if `st` is a palindrome (spelled the same backward and forward), `false` if it is not a palindrome.

(D) The method returns `true` if `st` begins and ends with the same letter.

(E) The method returns `true` if the second half of `st` contains the same sequence of letters as the first half, `false` if it does not.

25. Consider the following code segment.

```
int value = 33;
boolean calculate = true;
while (value > 5 || calculate)
{
   if (value % 3 == 0)
      value = value - 2;
   if (value / 4 < 3)
      calculate = false;
}
System.out.print(value);
```

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

(A) `0`
(B) `1`
(C) `8`
(D) `33`
(E) Nothing will be printed. It is an infinite loop.

Questions 26–28 refer to the following classes.

```
public class Person
{
   private String firstName;
   private String lastName;

   public Person(String fName, String lName)
   {
      firstName = fName;
      lastName = lName;
   }

   public String getName()
   {
      return firstName + " " + lastName;
   }
}

public class Adult extends Person
{
   private String title;

   public Adult(String fname, String lName, String myTitle)
   {
      super(fname, lName);
      title = myTitle;
   }

   /*  toString method to be implemented in question 27  */
}

public class Child extends Person
{
   private int age;

   /*  Constructor to be implemented in question 26  */
}
```

26. Which of the following is a correct implementation of a `Child` class constructor?

(A)
```
public Child(String first, String last, String t, int a)
{
    super(first, last, t);
    age = a;
}
```
(B)
```
public Child()
{
    super();
}
```
(C)
```
public Child(String first, String last, int a)
{
    super(first, last);
    age = a;
}
```
(D)
```
public Child extends Adult (String first, String last, String t, int a)
{
    super(first, last, t);
    age = a;
}
```
(E)
```
public Child extends Person(String first, String last, int a)
{
    super(first, last);
    age = a;
}
```

27. Which of the following is a correct implementation of the `Adult` class `toString` method?

(A)
```
public String toString()
{
    System.out.println(title + " " + super.toString());
}
```
(B)
```
public String toString()
{
    return title + " " + super.toString();
}
```
(C)
```
public String toString()
{
    return title + " " + firstName + " " + lastName;
}
```
(D)
```
public void toString()
{
    System.out.println(title + " " + super.getName());
}
```
(E)
```
public String toString()
{
    return title + " " + super.getName();
}
```

28. Consider the following method declaration in a different class.

```
public void findSomeone(Adult someone)
```

Assume the following variables have been correctly instantiated and initialized with appropriate values.

```
Person p;
Adult a;
Child c;
```

Which of the following method calls compiles without error?

I. `findSomeone(p);`
II. `findSomeone(a);`
III. `findSomeone(c);`
IV. `findSomeone((Adult) p);`
V. `findSomeone((Adult) c);`

(A) II only

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

29. Consider the following method. The method is intended to return `true` if the value `val` raised to the power `power` is within the tolerance `tolerance` of the target value `target`, and `false` otherwise.

```
public boolean similar(double val, double power, double target, double tolerance)
{
   /* missing code */
}
```

Which code segment below can replace `/* missing code */` to make the method work as intended?

(A)
```
double answer = Math.pow(val, power);
if (answer - tolerance >= target)
   return true;
return false;
```

(B)
```
return (Math.abs(Math.pow(val, power))) - target <= tolerance;
```

(C)
```
double answer = Math.pow(Math.abs(val), power);
return answer - target <= tolerance;
```

(D)
```
double answer = Math.pow(val, power) - target;
boolean within = answer - tolerance >= 0;
return within;
```

(E)
```
return (Math.abs(Math.pow(val, power) - target) <= tolerance);
```

30. Consider the following method.

```
public int changeEm(int num1, int num2, int[] values)
{
    num1 = values[num2];
    values[num1] = num2;
    num2++;
    return num2;
}
```

Consider the following code segment.

```
int[] values = {1, 2, 3, 4, 5};
int num1 = 2;
int num2 = 3;
num2 = changeEm(num1, num2, values);
System.out.println("num1 = " + num1 + "  values[" + num2 + "] = " + values[num2]);
```

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

(A) `num1 = 2 values[4] = 4`

(B) `num1 = 2 values[4] = 5`

(C) `num1 = 2 values[4] = 3`

(D) `num1 = 4 values[3] = 3`

(E) `num1 = 4 values[3] = 4`

31. Consider the following method.

```
public int firstLetterIndex(String s2)
{
    String s1 = "abcdefghijklmnopqrstuvwxyz";
    return s1.indexOf(s2.substring(s2.length() - 2));
}
```

What is returned as a result of the call `firstLetterIndex("on your side")`?

(A) `-1`

(B) `0`

(C) `3`

(D) `4`

(E) `5`

32. The following code segment appears in the main method of another class.

```
AnotherClass[] acArray = new AnotherClass[10];
acArray[2] = new AnotherClass("two");
acArray[3] = new AnotherClass("three");
for (int i = 0; i < acArray.length; i++)
{
    /* missing condition */
    {
        System.out.println(acArray[i].getData());
    }
}
```

Which of the following statements should be used to replace `/* missing condition */` so that the code segment will not terminate with a `NullPointerException`?

(A) `if (acArray[i] != null)`

(B) `if (acArray.get(i) != null)`

(C) `if (acArray.getData() != null)`

(D) `if (acArray.getData().equals("two") || acArray.getData().equals("three"))`

(E) Since not all of acArray's entries contain an `AnotherClass` object, there is no way to loop through the array without a `NullPointerException`.

33. A programmer intends to apply the standard Binary Search algorithm on the following array of integers. The standard Binary Search algorithm returns the index of the search target if it is found and -1 if the target is not found. What is returned by the algorithm when a search for 50 is executed?

```
int[] array = {9, 100, 11, 45, 76, 100, 50, 1, 0, 55, 99};
```

(A) `-1`

(B) `0`

(C) `5`

(D) `6`

(E) `7`

34. Consider the following code segment.

```
int[][] nums = { {0, 1, 2}, {3, 4, 5}, {6, 7, 8} };

for (int x = 0; x < nums.length; x++)
{
    int temp = nums[x][0];
    nums[x][0] = nums[x][2];
    nums[x][2] = temp;
}
```

What are the values in array nums after the code segment is executed?

(A) `{ {0, 1, 0}, {3, 4, 3}, {6, 7, 6} }`
(B) `{ {0, 0, 0}, {3, 3, 3}, {6, 6, 6} }`
(C) `{ {2, 1, 0}, {5, 4, 3}, {8, 7, 6} }`
(D) `{ {0, 1, 2}, {3, 4, 5}, {6, 7, 8} }`
(E) `{ {8, 7, 6}, {5, 4, 3}, {2, 1, 0} }`

35. Consider the following code segment.

```
ArrayList<Integer> newList = new ArrayList<Integer>();
Integer[] list = {2, 4, 3, 3, 2, 5, 1};
newList.add(list[0]);

for (int i = 1; i < list.length; i++)
{
    if (list[i] > newList.get(newList.size() - 1))
       newList.add(list[i]);
}

System.out.println(newList);
```

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

(A) `[2, 4, 5]`
(B) `[2, 2, 1]`
(C) `[2, 4, 3, 5, 1]`
(D) `[1, 2, 3, 3, 4, 5]`
(E) `[2, 4, 3, 3, 5, 1]`

36. Consider the following code segment.

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

for (int r = 0; r < a.length; r++)
   for (int c = r + 1; c < a[0].length; c++)
      a[r][c] = 9;

for (int d = 0; d < a.length; d++)
   a[d][d] = d;

System.out.println(a[1][2] + " " + a[3][3] + " " + a[4][3]);
```

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

(A) `9 9 9`
(B) `0 0 0`
(C) `0 3 9`
(D) `9 3 9`
(E) `9 3 0`

37. Consider the following sorting method.

```
public void mysterySort(int[] arr)
{
    for (int j = 1; j < arr.length; j++)
    {
        int temp = arr[j];
        int index = j;
        while (index > 0 && temp < arr[index - 1])
        {
            arr[index] = arr[index - 1];
            index--;
        }
        arr[index] = temp;
    }
}
```

Consider the following code segment.

```
int[] values = {9, 1, 3, 0, 2};
mysterySort(values);
```

Which of the following shows the elements of the array in the correct order after the second pass through the outer loop of the sorting algorithm?

(A) `{0, 1, 2, 9, 3}`
(B) `{0, 1, 3, 9, 2}`
(C) `{0, 1, 9, 3, 2}`
(D) `{1, 3, 9, 0, 2}`
(E) `{1, 3, 0, 2, 9}`

38. Consider the following method.

```
public int puzzle(int m, int n)
{
    if (n == 1)
        return m;
    return m * puzzle(m, n - 1);
}
```

What is returned by the method call `puzzle(3, 4)`?

(A) 9
(B) 64
(C) 81
(D) 128
(E) Nothing is returned. Infinite recursion causes a stack overflow error.

39. Consider the following code segment.

```
int[][] table = new int[5][6];
int val = 0;
for (int k = 0; k < table[0].length; k++)
{
    for (int j = table.length - 1; j >= 0; j--)
    {
        table[j][k] = val;
        val = val + 2;
    }
}
```

What is the value of `table[3][4]` after executing the code segment?

(A) `22`
(B) `30`
(C) `34`
(D) `42`
(E) `46`

40. Consider the following method.

```
public void selectionSort(String[] arr)
{
    for (int i = 0; i < arr.length; i++)
    {
        int small = i;
        for (int j = i; j < arr.length; j++)
        {
            /*  missing code */
            small = j;
        }
        String temp = arr[small];
        arr[small] = arr[i];
        arr[i] = temp;
    }
}
```

Assume `String[] ray` has been properly instantiated and populated with `String` objects.

Which line of code should replace `/* missing code */` so that the method call `selectionSort(ray)` results in an array whose elements are sorted from least to greatest?

(A) `if (arr[j].equals(arr[small])`
(B) `if (arr[j] > arr[small])`
(C) `if (arr[j] < arr[small])`
(D) `if (arr[j].compareTo(arr[small]) > 0)`
(E) `if (arr[j].compareTo(arr[small]) < 0)`

STOP. End of Part I.

AP Computer Science A
Practice Exam 1

Part II (Free Response)

Time: 90 minutes
Number of questions: 4
Percent of total score: 50

Directions: Write all of your code in Java. Show all your work.

Notes:

- You may assume all import statements have been included where they are needed.
- You may assume that all preconditions are met when making calls to methods.
- You may assume that all parameters within method calls are not null.
- Be aware that you should, when possible, use methods that are defined in the classes provided as opposed to duplicating them by writing your own code. Doing so you will not receive full credit.
- You may use the Java Quick Reference sheet as needed.

1. A `Password` class contains methods used to determine information about passwords. You will write two methods of the class.

```
public class Password
{
    private String upper = "ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    private String lower = "abcdefghijklmnopqrstuvwxyz";
    private String symbols = "!@#$%^&*";
    private int minLength;
    private int maxLength;

    public Password(int min, int max)
    {
        /* implementation not shown   */
    }

    public boolean isValid(String password)
    {
        /* implemented in part (a)    */
    }

    public String generatePassword()
    {
        /* implemented in part (b)     */
    }

    // There may be instance variables, constructors, and other methods not shown.

}
```

(a) Write the method `isValid`, which returns true if a password is valid and false otherwise. A password is considered valid if

- Its length is a valid length. The length must be at least minimum characters and at most maximum characters, inclusive.
- It contains at least one uppercase letter, one lowercase letter, and one symbol.

Statement	Value Returned	Comment
Password p1 = new Password(3,15);		Passwords can be any length between 3 and 15 inclusive.
p1.isValid("HelloWorld!");	true	Password has length 11, contains at least 1 uppercase letter, 1 lowercase letter, and 1 symbol.
p1.isValid("Aloha")	false	Password does not include a symbol
p1.isValid("#APComputerScience Rocks");	false	Password has length 24

Complete method `isValid` below.

```
public boolean isValid(String password)
```

(b) Write the method `generatePassword`, which returns a `String` representing a valid password. In writing `generatePassword` you must call `isValid`. Assume `isValid` works correctly regardless of what you wrote in part (a).

The `generatePassword` method must randomly choose the length of the password and randomly choose characters from the uppercase, lowercase, and symbol strings.

Statement	Value Returned	Comment
Password p2 = new Password(4,10);		Passwords can be any length between 4 and 10 inclusive.
p2.generatePassword();	Bkb@mHI	Password has length 7, contains at least 1 uppercase letter, 1 lowercase letter, and 1 symbol.
p2.generatePassword();	k&iHBe	Password has length 6, contains at least 1 uppercase letter, 1 lowercase letter, and 1 symbol.
p2.generatePassword();	hY*G	Password has length 4, contains at least 1 uppercase letter, 1 lowercase letter, and 1 symbol.

Complete method `generatePassword` below.

```
public String generatePassword()
```

2. An ISBN (International Standard Book Number) is a numeric book identifier that is assigned to each published book. Prior to 2007, the ISBN numbers were 10 digits in length. The ISBN number is broken down into sections (group, publisher, title, and check digit). The last digit, or check digit, is used for error detection.

To generate a valid check digit, the first nine digits in the ISBN is multiplied by a weight. Each weight is multiplied by the corresponding digit, and then the products are added together.

ISBN with no check digit	1	2	6	0	4	5	4	9	1
Weight	10	9	8	7	6	5	4	3	2
Product	10	18	48	0	24	25	16	27	2
Sum	$10 + 18 + 48 + 0 + 24 + 25 + 16 + 27 + 2 = 170$								
Check Digit	$11 - (\text{sum} \% 11) = 11 - (170\%11) = 6$								

ISBN with no check digit	0	3	0	6	4	0	6	1	1
Weight	10	9	8	7	6	5	4	3	2
Product	0	27	0	42	24	0	24	3	2
Sum	$0 + 27 + 0 + 42 + 24 + 0 + 24 + 3 + 2 = 122$								
Check Digit	$11 - (\text{sum} \% 11) = 11 - (122\%11) = 10 = \text{X}$								

If the value of the check digit is 10, then the check digit becomes "X".

An `ISBN` object is created with a parameter that contains a nine-digit ISBN number. The `ISBN` class provides two methods: `calculateCheckDigit` and `generateNumber`.

- `calculateCheckDigit` will perform the calculations described above to determine the check digit.
- `generateNumber` will join together the nine-digit number and the check digit generated from the method defined above.

For example, consider the following code that is in a class other than the `ISBN` class.

```
ISBN book1 = new ISBN(126045491);
String check1 = book1.calculateCheckDigit();         //  "6" will be returned
String isbnNumber1 = book1.generateNumber();         // "126045491-6" will be returned

ISBN book2 = new ISBN(030640611);
String check2 = book2.calculateCheckDigit();         //  "X" will be returned
String isbnNumber2 = book2.generateNumber();         // "030640611-X" will be returned
```

Write the complete `ISBN` class, including the constructor and any required instance variables and methods. Your implementation must meet all specifications to confirm to the example.

3. A train is composed of an engine and any number of train cars. The engine is rated to pull up to a maximum weight, based on its power. The weight of all the train cars combined, including the engine, must be below this maximum weight or the engine will not be able to move the train.

   Trains are represented by the `Train` class and contain an engine followed by any number of cars. An `ArrayList` will be used to

store the weight of the engine (in element 0), and the weights of each of the cars (in elements 1, 2, 3, etc.).

The weight of a `Train` object is calculated by adding the weight of the engine to the weight of each of the objects in the `trainCars` ArrayList [to be completed in part (a)].

Trains need to be checked to make sure that their weight can be pulled by their engine. If the train is overweight, train operators must remove train cars from the end of the train until the train is within the acceptable weight range [to be completed in part (b)].

```
public class Train
{
   private double maxWeight;
   private ArrayList <Double> trainCars;

   public Train(Double max, ArrayList <Double> tc)
   {
      maxWeight = max;
      trainCars = tc;
   }

   public double getMaximumWeight()
   /*       implementation not shown    */

   /** Finds the total weight of the train.
    *
    *    @return double value containing the sum of the weights of the train
    */
   public double getTotalWeight()
   {     /*     To be implemented in part (a)    */     }


   /** Removes Double objects from the end of the train
    *    until the train can be pulled by the Engine.
    *
    *    @return ArrayList<Double> containing the removed cars in
    *            the order they were removed (the last car is
    *            item 0, etc.). If no cars are removed, the returned
    *            list will be empty.
    */
   public ArrayList<Double> removeExcessTrainCars()
   {     /*     To be implemented in part (b)    */     }

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

(a) Write the `getTotalWeight` method that will calculate the total weight of the train by adding the weight of each of the train

cars to the weight of the engine (element 0).

```
/** Finds the total weight of the train.
 *
 * @return double value containing the sum of the weights of the train cars
 */
public double getTotalWeight()
{  /*     To be implemented in part (a)    */     }
```

(b) Write the `removeExcessTrainCars` method of the `Train` class that removes `Double` objects one at a time from the end of the train until the train weight is less than or equal to the maximum weight allowed as given by the `getMaximumWeight` method of the `Train` object. The removed train cars are added to the end of an `ArrayList` of `Double` objects as they are removed. This `ArrayList` of removed `Double` objects is returned by the method. If no `Double` objects need to be removed, an empty `ArrayList` is returned. You may assume what you wrote for part (a) works as intended.

Example:
A train is composed of the cars listed below. The engine has a maximum weight rating of 475,000 pounds.

Car	Weight
Engine/ `trainCars[0]`	200,000
`trainCars[1]`	100,000
`trainCars[2]`	140,000
`trainCars[3]`	50,000
`trainCars[4]`	100,000
`trainCars[5]`	60,000

In the example, the train initially weighs 650,000 pounds. Train cars need to be removed one by one from the end of the train until the total weight is under the maximum allowed weight of 475,000.

- Removing the train car from index 5 lowers the weight to $590{,}000$
- Removing the train car from index 4 lowers the weight to $490{,}000$
- Removing the train car from index 3 lowers the weight to $440{,}000$

At $440{,}000$ pounds, the train is now in the acceptable weight range. The following `ArrayList` is returned by the method:

```
[60000.0, 100000.0, 50000.0]
```

Complete the `removeExcessTrainCars` method.

```
/** Removes TrainCar objects from the end of the train
 *  until the train can be pulled by the Engine.
 *
 *  @return ArrayList<TrainCar> containing the removed cars in
 *          the order they were removed (the first car removed is
 *          item 0, etc.) If no cars are removed, the returned
 *          list will be empty.
 */
public ArrayList <Double> removeExcessTrainCars()
```

4. The colors of the pixels on a TV or computer screen are made by combining different quantities of red, green, and blue light. Each of the three colors is represented by a value between 0 and 255, where 0 means none of that color light and 255 means as much of that color as there can be. This way of representing colors is referred to as RGB, for red-green-blue, and the color values are written in ordered triples like this $(80, 144, 240)$. That particular combination means red at a value of 80, green at a value of 144, and blue at a value of 240. The resulting color is a kind of medium blue. The combination $(0, 0, 0)$ produces black and $(255, 255, 255)$ produces white.

   Pixels can be modeled by the `Pixel` class shown below.

```
public class Pixel
{
    private int red;
    private int green;
    private int blue;

    public Pixel(int myRed, int myGreen, int myBlue)
    {
        red = myRed;
        green = myGreen;
        blue = myBlue;
    }
    public String toString()
    {
        return "(" + red + ", " + green + ", "   + blue + ")";
    }
}
```

An artist wants to manipulate the pixels on a large computer display. She begins by creating three two-dimensional arrays the size of the screen, `rows` by `columns`. One array contains the red value of each pixel, one contains the blue value of each pixel, and one contains the green value of each pixel. She then calls methods of the `AlterImage` class to generate and then manipulate the data.

```
public class AlterImage

{
   /** Converts 3 arrays of color values into one array
    *  of Pixel objects
    *
    *  @param reds the red color values
    *  @param greens the green color values
    *  @param blues the blue color values
    *  @return the Pixel array generated with information
    *          in the red, green and blue arrays
    *
    *  Precondition:  The three color arrays are all the same
    *                 size and contain int values in the range 0-255.
    *  Postcondition: The returned array is the same size as the
    *                 3 color arrays.
    */
   public Pixel[][] generatePixelArray(int[][] reds, int[][] greens, int[][] blues)
   {  /* to be implemented in part (a) */  }
```

```
    /** Flips a 2D array of Pixel objects either
     *  horizontally or vertically
     *
     *  @param image the 2D array of Pixel objects to be processed
     *  @param horiz true: flip horizontally, false: flip vertically
     *  @return the processed image
     */
    public Pixel[][] flipImage(Pixel[][] image, boolean horiz)
    {  /* to be implemened in part (b) */  }

    /* Additional implementation not shown */

}
```

(a) Given the three arrays `reds`, `greens`, and `blues`, and the `Pixel` and `AlterImage` classes, implement the `generatePixelArray` method that converts the raw information in the three color arrays into a `rows` by `columns` array of `Pixel` objects.

Precondition: The three color arrays are all the same size and contain `int` values in the range 0–255.
Postcondition: The returned array is the same size as the three color arrays.

```
/** Converts 3 arrays of color values into one array
 *  of Pixel objects
 *
 *  @param reds the red color values
 *  @param greens the green color values
 *  @param blues the blue color values
 *  @return the Pixel array generated with information
 *          from the red, green and blue arrays
 *
 *  Precondition:    The three color arrays are all the same
 *                   size and contain int values in the range 0-255.
 *  Postcondition:   The returned array is the same size as the
 *                   3 color arrays.
 */
public Pixel[][] generatePixelArray(int[][] reds, int[][] greens, int[][] blues)
```

(b) The artist uses various techniques to modify the image displayed on the screen. One of these techniques is to flip the image, either horizontally (along the *x*-axis, top-to-bottom) or vertically (along the *y*-axis, left-to-right), so that a mirror image is produced as shown below.

In this example, the data in the cells are the RGB values stored in the `Pixel` objects. To make the example clearer,

red is set to the original row number, green is set to the original column number, and blue is constant at 100.

Original array:

0, 0, 100	0, 1, 100	0, 2, 100	0, 3, 100
1, 0, 100	1, 1, 100	1, 2, 100	1, 3, 100
2, 0, 100	2, 1, 100	2, 2, 100	2, 3, 100
3, 0, 100	3, 1, 100	3, 2, 100	3, 3, 100

Flipped horizontally:

3, 0, 100	3, 1, 100	3, 2, 100	3, 3, 100
2, 0, 100	2, 1, 100	2, 2, 100	2, 3, 100
1, 0, 100	1, 1, 100	1, 2, 100	1, 3, 100
0, 0, 100	0, 1, 100	0, 2, 100	0, 3, 100

Flipped vertically:

0, 3, 100	0, 2, 100	0, 1, 100	0, 0, 100
1, 3, 100	1, 2, 100	1, 1, 100	1, 0, 100
2, 3, 100	2, 2, 100	2, 1, 100	2, 0, 100
3, 3, 100	3, 2, 100	3, 1, 100	3, 0, 100

Write the `flipImage` method, which takes as parameters a 2D array of `Pixel` objects and a `boolean` direction to flip, where true means horizontally and false means vertically. The method returns the flipped image as a 2D array of `Pixel` objects.

```
/** Flips a 2D array of Pixel objects either
 *  horizontally or vertically
 *
 *  @param image the 2D array of Pixel objects to be processed
 *  @param horiz true: flip horizontally, false: flip vertically
 *  @return the processed image
 */
public Pixel[][] flipImage(Pixel[][] image, boolean horiz)
```

STOP. End of Part II.

Practice Exam 1
Answers and Explanations

Part I (Multiple Choice)

Answers and Explanations

Bullets mark each step in the process of arriving at the correct solution.

1. The answer is E.
   - Lines 1 and 2 give us: myValue = 17 and multiplier = 3
   - Line 3: answer = 17 % 3 + 17 / 3
     - Using integer division: 17 / 3 = 5 remainder 2, giving us:
     - 17 / 3 = 5
     - 17 % 3 = 2
     - The expression simplifies to answer = 2 + 5 = 7 (remember order of operations)
   - Line 4: answer = 7 * 3 = 21
2. The answer is B.
   - Let's use De Morgan's theorem to simplify the expression.
     `!(a < b || b <= c) && !(a < c || b >= a)`
   - Distribute the first !. Remember to change || to &&. This gives:

`!(a < b) && !(b <= c)`

- Distribute the second ! This gives:
  `!(a < c) && !(b >= a)`
- It's easy to simplify all those !s. Remember that !< → >= and !<= → > (the same with < and >=).
  `(a >= b) && (b > c) && (a >= c) && (b < a)`
- Since these are all &&s, all four conditions must be true. The first condition says a >= b and the fourth condition says b < a. No problem there. Options B and C both have a > b. Option A can be eliminated.
- The second condition says b > c. That's true in option B. Option C can be eliminated.
- The third condition says b < a. Still true in option B, so that's the answer.
- You could also solve this problem with guess and check by plugging in the given values, but all those ands and ors and nots tend to get pretty confusing. Simplifying at least a little will help, even if you are going to ultimately plug and chug.

3. The answer is D.
   - The `while` loop condition is (myArray[index] < 7). If you did not read carefully, you may have assumed that the condition was (index < 7), which, along with the index++ is the way you would write it if you wanted to access every element in the array.
   - Because the condition is (myArray[index] < 7), we will start at element 0 and continue until we come to an element that is greater than or equal to 7. At that point we will exit the loop and no more elements will be processed.
   - Every element before the 7 in the array will have 3 added to its value.

4. The answer is A.
   - Option I correctly accesses and processes every element in the `ArrayList`.
   - Option II uses a `for-each` loop. That is an excellent option for this problem, since we need to process each element in

exactly the same way, but option II does not access the element correctly. Read the `for-each` loop like this: “For each OnlinePurchaseItem (which I will call purchase) in items.…” The variable purchase already holds the needed element. Using get(i) to get the element is unnecessary and there is no variable i.

- Option III uses a `for` loop and processes the elements in reverse order. That is acceptable. Since we need to process every item, it doesn't matter what order we do it in. However, option III starts at i = items.size(), which will cause an `IndexOutOfBoundsException`. Remember that the last element in an `ArrayList` is list.size() – 1 (just like a String or an array).

5. The answer is D.
   - This is a recursive method. Let's trace the calls. The parts in *italics* were filled in on the way back up. That is, the calls in the plain type were written top to bottom until the base case returned a value. Then the answers were filled in *bottom to top*.
     - mystery(5) = 2 + mystery(4) = 2 + *7* = ***9*** which gives us our final answer.
     - mystery(4) = 2 + mystery(3) = 2 + *5* = *7*
     - mystery(3) = 2 + mystery(2) = 2 + *3* = *5*
     - mystery(2) = 2 + mystery(1) = 2 + *1* = *3*
     - mystery(1) Base Case! return 1

6. The answer is E.
   - The outer loop will execute three times, starting at index = 0 and continuing as long as index < 4, increasing by one each time through. So index will equal 0, 1, 2, 3 in successive iterations through the loop.
   - The only statement inside the outer loop is the inner loop. Let's look at what the inner loop does in general terms.
   - The inner loop executes from i = 0 to i < index, so it will execute index times.
   - Each time through it puts an "A" in front of and after myString.

- Putting it all together:
  - The first iteration of the outer loop, index = 0, the inner loop executes 0 times, myString does not change.
  - The second iteration of the outer loop, index = 1, the inner loop executes one time, adding an "A" in front of and after myString. myString = "AHA"
  - The third iteration of the outer loop, index = 2, the inner loop executes two times, adding two "A"s in front of and after myString. myString = "AAAHAAA"
  - The fourth (and last) iteration of the outer loop, index = 3, the inner loop executes three times, adding three more "A"s in front of and after myString. myString = "AAAAAAHAAAAAA"
- Putting it another way, when index = 1, we add one "A"; when index = 2, we add two "A"s; when index = 3, we add three "A"s. That's six "A"s all together added to the beginning and end of "H" → "AAAAAAHAAAAAA"

7. The answer is B.
   - The general form for generating a random number between *high* and *low* is:
     `(int)(Math.random() * (high - low + 1)) + low`
   - high – low + 1 = 50, low = 10, so high = 59
   - The correct answer is integers between 10 and 59 inclusive

8. The answer is C.
   - This question requires that you understand the two uses of the + symbol.
   - First, we execute what is in the parentheses. Now we have:
     `13 + 6 + "APCSA" + 4 + 4`
   - Now do the addition left to right. That's easy until we hit the string:
     `19 + "APCSA" + 4 + 4`
   - When you add a number and a string in any order, Java turns the number into a string and then concatenates the strings:
     `"19APCSA" + 4 + 4`

- Now every operation we do will have a string and a number, so they all become string concatenations, not number additions.
  `"19APCSA44"`

9. The answer is E.
   - Let's consider what the expression is telling us.
     `(truth1 && truth2) || ((!truth1) && (!truth2))`
   - The expression is true if both truth1 and truth2 are true OR if both truth1 and truth2 are false.
   - Therefore, the expression is true as long as truth1 == truth2.
10. The answer is A.
    - The for-each loop looks at every item in the ArrayList and adds it to sum if:
      - the value of the item > 1 (all values except 0 and 1) AND
      - the size of the list - 3 > the value of the item. Since the list has 10 elements, we can rewrite this as the value of the item < 7, which is true for all elements 6 and below.
      - Both conditions are true for 2, 3, 4, 5, 6; so the sum is 20.
11. The answer is E.
    - Methods or constructors with the same name (and return type, in the case of methods) may have different parameter lists. The parameters may differ in type or number. This is called *overloading*.
12. The answer is A.
    - We can eliminate options B and D immediately by noticing that active is a `boolean` and will therefore print as either "true" or "false" and not as "active" even though that is more informative.
    - We would expect option E to be correct by reading the code, but the constructor was called incorrectly. The overloaded constructor that is being called is the two-parameter constructor, and that constructor is going to assign its first parameter to state and its second parameter to city. Since we

passed city, state, in that order, the city name and the state name have been assigned incorrectly and will therefore be printed incorrectly.

- Note that this is a confusing way to write an overloaded constructor! The order of the variables changes for no reason.

13. The answer is D.
    - Option I is incorrect. Most classes have a no-argument constructor; either the default constructor provided automatically if no other constructor is provided, or one written by the programmer. However, the `Depot` class does not. The super call causes a compile-time error.
    - Option II is correct. It implements a no-argument constructor in the `WhistleStop` class, which passes the needed parameters to the Depot constructor.
    - Option III is also correct. It takes two parameters and correctly passes them to the Depot constructor via the call to super. However, this code will probably not give you the result you are expecting. Just like in question 12, the order of the city and province has been switched.

14. The answer is A.
    - Option I is correct. The `ArrayList` is correctly instantiated as an `ArrayList` of `Depot` objects. Since a `WhistleStop` object *is-a* `Depot` object, a `WhistleStop` object may be added to the `ArrayList`. `WhistleStop` contains a no-argument constructor, and the Depot constructor is called correctly as well.
    - Option II is incorrect. A `WhistleStop` *is-a* `Depot`, but not the other way around. We cannot put `Depot` objects into an `ArrayList` that is declared to hold `WhistleStop` objects. The `ArrayList` declaration will generate a compile-time error.
    - Option III is incorrect. The `ArrayList` is correctly instantiated as an `ArrayList` of `WhistleStop` objects. We can add `WhistleStop` objects to this `ArrayList`, but we cannot add `Depot` objects. The second add will generate a compile-time error.

15. The answer is B.
    - When we start the `for-each` loop, the variable crazyString = "crazy" and the `ArrayList` crazyList has two elements ["weird," "enigma"].
    - You can read the `for-each` like this: “For each String (which I am going to call s) in crazyList.…”
    - So for each String s in crazyList we execute the assignment statement:
      ```
      crazyString = s.substring(1, 3) +
      crazyString.substring(0, 4);
      ```
      which takes two characters starting at index 1 from s and concatenates the first four characters of crazyString.
    - The first time through the loop, s = "weird" and crazyString = "crazy" so the assignment becomes:
      `crazyString = "ei" + "craz" = "eicraz"`

      Remember that substring starts at the first parameter and ends *before* the second parameter, and remember that we start counting the first letter at 0.
    - The second time through the loop, s = "enigma" and crazyString = "eicraz" so the assignment becomes:
      `crazyString = "ni" + "eicr" = "nieicr"`
16. The answer is E.
    - There are four possible paths through the code. We need one data element that will pass through each path.
      - The first path is executed if n < -5.
      - The second path is executed if -5 ≤ n < 0 (because if n < -5, the first path is executed and we will not reach the second path).
      - The third path is executed if n > 10.
      - The fourth path is executed in all other cases.
    - We need to be sure that we have at least one piece of data for all four of those cases.

- Option A does not test the fourth path, because 12 and 15 are both > 10.
- Option B does not test the first path, because it doesn't have a number that is < -5.
- Option C does not test the third path, because it doesn't have a number that is > 10.
- Option D does not test the second path, because 0 is not less than 0.
- Option E tests all four paths.

17. The answer is D.
    - This statement should be written with if-else ladder. When the clause associated with a true condition is executed, the rest of the statements should be skipped. Unfortunately, that is not how it is written. For example, if the parameter temperature = 80, the first if is evaluated and found to be true and activity = "go swimming"; however, since there is no else, the second if is also evaluated and found to be true, so activity = "go hiking". Then the third if is evaluated and found to be true, so activity = "go horseback riding". The else clause is skipped and "go horseback riding" is returned (incorrectly).
    - All temperatures that are true for multiple `if` statements will cause activity to be set incorrectly. Only the values of temperature that will execute the last if or the else clause will return the correct answer.
    - In order for this code to function as intended, all ifs except the first one should be preceded with *else*.

18. The answer is D.
    - This time, there is an if-else ladder, but they are in the wrong order. It is important to write the most restrictive case first. Let's test this code by using 80 as an example. 80 > 75, and that is the condition we intend to execute, but unfortunately, it is also > 45, which is the first condition in the code, so that is the if clause that is executed and activity is set to "go horseback riding". Since the code is written as an if-else

ladder, no more conditions are evaluated and flow of control jumps to the return statement.

- Two sets of values give the right answer:
  - Values that execute the first if clause correctly—that is, values that are greater than 45, but less than or equal to 60.
  - Values that fail all the if conditions and execute the else clause—that is, values that are less than or equal to 45.
- Those two conditions can be combined into temperatures <= 60.
- In order for this code to function as intended, the order of the conditions needs to be reversed.

19. The answer is A.
    - Let's look at the three code segments one by one.
    - Option I:
      - When we enter the loop, i = 1.
      - arr[1 / 2] = arr[0] = 1 (don't forget integer division)
      - i = i + 2 = 3, 3 < 6 so we enter the loop again.
      - arr [3 / 2] = arr[1] = 3
      - i = i + 2 = 5, 5 < 6 so we enter the loop again.
      - arr[5 / 2] = arr[2] = 5
      - i = i + 2 = 7, 7 is not < 6 so we exit the loop.
      - The array arr = [1, 3, 5]
    - Option II:
      - When we enter the loop, i = 0.
      - arr[0] = 2 * 0 + 1 = 1
      - increment i to 1, 1 < 3 so we enter the loop again.
      - arr[1] = 2 * 1 + 1 = 3
      - increment i to 2, 2 < 3 so we enter the loop again.
      - arr[2] = 2 * 2 + 1 = 5
      - increment i to 3, 3 is not < 3 so we exit the loop.
      - The array arr = [1, 3, 5]
    - Option III:
      - When we enter the loop, i = 6.

    - arr[(6 – 6) / 2] = arr[0] = 6 – 6 = 0
    - Since we know the first element should be 1, not 0, we can stop here and eliminate this option.
- Options I and II produce the same results.

20. The answer is C.
- Let's picture our `ArrayList` as a table. After the add statements, `ArrayList` nations looks like this:

Argentina	Canada	Australia	Cambodia	Russia	France

- It looks like perhaps the loop is intended to remove all elements whose length is greater than or equal to 7, but that is not what this loop does. Let's walk through it.
- i = 0. nations.size() = 6, 0 < 6 so we enter the loop.
    - nations.get(0).length() gives us the length of "Argentina" = 9.
    - 9 >= 7, so we remove the element at location 0. Now our `ArrayList` looks like this:

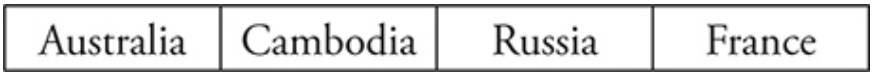
Canada	Australia	Cambodia	Russia	France

- increment i to 1. nations.size() = 5, 1 < 5, so we enter the loop.
    - nations.get(1).length() gives us the length of "Australia" = 9. (We skipped over "Canada" because it moved into position 0 when "Argentina" was removed.)
    - 9 >= 7, so we remove the element at location 0. Now our `ArrayList` looks like this:

Australia	Cambodia	Russia	France

- increment i to 2. nations.size() = 4, 2 < 4 so we enter the loop.
    - nations.get(2).length() gives us the length of "Russia" = 6.
    - 6 is not >= 7 so we skip the if clause.
- increment i to 3. nations.size() = 4, 3 < 4 so we enter the loop.
    - nations.get(3).length() gives us the length of "France" = 6.
    - 6 is not >= 7 so we skip the if clause.

- increment i to 4. nations.size() = 4, 4 is not < 4 so we exit the loop with the `ArrayList`:



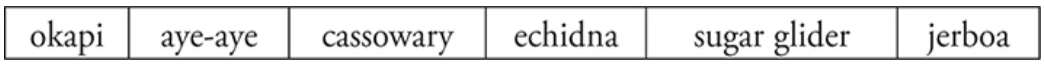
- Note: You have to be really careful when you remove items from an `ArrayList` in the context of a loop. Keep in mind that when you remove an item, the items with higher indices don't stay put. They all shift over one, changing their indices. If you just keep counting up, you will skip items. Also, unlike an array, the size of the `ArrayList` will change as you delete items.

21. The answer is D.
    - Option I is correct. It calls the super constructor, correctly passing on 2 parameters, and then sets its own instance variables, one to the passed parameter, one to a default value.
    - Option II is incorrect. It attempts to call the student class 2 parameter constructor, passing default values, but the year is being passed as a String, not as an int.
    - Option III is correct. There will be an implicit call to Student's no-argument constructor.

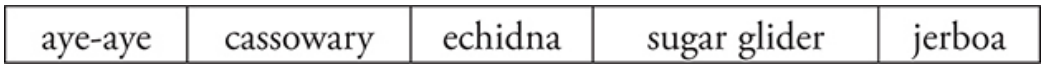
22. The answer is D.
    - index starts at 0 and goes to the end of the String. We want a condition that will look at the letter in word at index, compare it to the letter parameter, and count it if it is the same.
    - Option A is incorrect. The one-parameter substring includes everything from the starting index to the end of the word. We need one letter only.
    - Option B is incorrect. We cannot compare Strings with ==.
    - Option C is incorrect. indexOf will tell us whether a letter appears in a word and where it appears, but not how many times. It is possible to count the occurrences of letter using indexOf, but not this way. Option C does not change the if condition each time through the loop. It just asks the same question over and over.

- Option D is correct. It selects one letter at index and compares it to letter using the `equals` method, incrementing count if they match.
- Option E is incorrect. It will not compile. It tries to compare letter to the int returned by indexOf.

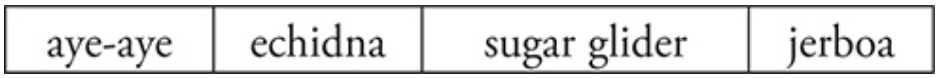
23. The answer is C.
    - You might expect that the loop will remove any animal whose name comes before pink fairy armadillo lexicographically, but removing elements from an ArrayList in the context of a loop is tricky. Let's walk through it.
    - We start with:



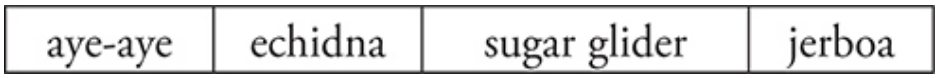
    - The `for` loop begins, i = 0. Compare "okapi" with "pink fairy armadillo", and since "o" comes before "p", remove "okapi".



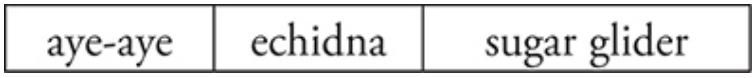
    - The next iteration begins with i = 1. The remove operation has caused all the elements' index numbers to change. We skip "aye-aye", which is now element 0, and look at "cassowary", which we remove.



    - The next iteration begins with i = 2. Again the elements have shifted, so we skip "echidna", compare to "sugar glider" but that comes after "pink fairy armadillo" so it stays put leaving the ArrayList unchanged.



    - The next iteration begins with i = 3. Remove "jerboa".



- i = 4, which is not < animals.size(). We exit the loop and print the ArrayList.
- Remember that if you are going to remove elements from an ArrayList in a loop, you have to adjust the index when an element is removed so that no elements are skipped.
- Note that if we used the loop condition i < 6 (the size of the original ArrayList), then there would have been an `IndexOutOfBoundsException`, but because we used animals.size(), the value changed each time we removed an element.

24. The answer is C.
    - This method takes a String parameter and loops through the first half of the letters in the string.
    - The first substring statement assigns the letter at index i to sub1. You may have to run through an example by hand to discover what the second substring does.
    - If our string is “quest”:
        - the first time through the loop, i = 0, so sub1 = "q" and sub2 = st.substring(5 - 0 - 1, 5 - 0) = "t",
        - the second time, i = 1, so sub1 = "u" and sub2 = st.substring(5 - 1 - 1, 5 - 1) = "s".
    - This loop is comparing the first letter to the last, the second to the second to last, and so on, which will tell us if the string is a palindrome.

25. The answer is E.
    - The loop will continue to execute until either value <= 5 or calculate = false.
    - When we enter the loop, value = 33, calculate = true.
        - Since 33 % 3 = 0 we execute the first if clause and set value = 31.
        - 31 / 4 = 7, which is not < 3, so we do not execute the second if clause.
    - The second iteration of the loop begins with value = 31 and calculate = true.

    - Since $31 \% 3 != 0$, we do not execute the first if clause.
    - Since $31 / 4 = 7$, which is not < 3, we do not execute the second if clause.
  - At this point, we notice that nothing is ever going to change. Value will always be equal to 31 and calculate will always be true. This is an infinite loop.

26. The answer is C.
  - Option A is incorrect. If Child extended Adult, this would be the correct constructor, but Child extends Person. The Person constructor does not take a title, only a first name and last name.
  - Option B is incorrect because Person does not have a no-argument constructor. The call to super() will cause a compile time error.
  - Options D and E are incorrect because of the "extends" phrase. The keyword `extends` is used in the class declaration, not the constructor declaration.
  - Option C is the correctly written constructor.

27. The answer is E.
  - Options A and D are incorrect because they print information to the console rather than returning it as a string. Option A also calls super.toString, which does not exist.
  - Option B is incorrect because, like option A, it calls super.toString, which does not exist.
  - Option C is incorrect because firstName and lastName are private instance variables of the `Person` class and cannot be accessed by the `Adult` class.
  - Option E is a correctly written `toString` method.

28. The answer is C.
  - Method findSomeone is expecting a parameter of type Adult. The question is, which of the options correctly provides a parameter of type Adult?

- Option I is incorrect. Here, the parameter is of type Person. Adult extends Person, not the other way around, so a Person is not an Adult.
- Option II is correct. The parameter is of type Adult.
- Option III is incorrect. Here, the parameter is of type Child. Child extends Person, not Adult, so a Child is not an Adult.
- Option IV is correct. A Person reference variable can be downcast to an Adult since Adult extends Person. At run-time, if the reference variable does not reference an Adult object as promised by the cast, the program will crash with a run-time error.
- Option V is incorrect. Child is not a superclass of Adult. Child cannot be downcast to Adult.

29. The answer is E.
    - The basic formula for tolerance is:
      ```
      Math.abs(a - b) <= tolerance;
      ```
    - In our case, a is val^{power}, or Math.pow(val, power), and b is target, so the equation becomes:
      ```
      Math.abs(Math.pow(val, power) - target) <= tolerance;
      ```
    - That evaluates to a `boolean`. We could assign it to a `boolean` variable and return the variable, but we can also just return the expression as in option E.

30. The answer is C.
    - This problem requires you to understand that primitives are passed by value and objects are passed by reference.
    - When a primitive argument (or actual parameter) is passed to a method, its value is copied into the formal parameter. Changing the formal parameter inside the method has no effect on the value of the variable passed in. The caller's num1 will not be changed by the method.
    - The caller's num2 is not changed when the num2++ is executed inside the method. That new value, however, is returned by the method, and the caller uses the returned value to reset num2. The value of num2 will change.

- When an object is passed to a method, its reference is copied into the formal parameter. The actual and formal parameters become aliases of each other; that is, they both point to the same object. Therefore, when the object is changed inside the method, those changes will be seen outside of the method. Changes to array values inside the method will be seen in array values outside of the method.
- num1 remains equal to 2, even though the method's num1 variable is changed to 4. num2 is reset to 4, because the method returns that value, and values[4] is set to 3.

31. The answer is C.
    - Let's figure this out step by step. The length of "on your side" is 12 (don't forget to count spaces), so `s1.indexOf(s2.substring(s2.length() - 2))` simplifies to `s1.indexOf(s2.substring(10))`
    - That gives us the last two letters of s2, so our statement becomes `s1.indexOf("de")`
    - Can we find "de" in s1? Yes. It starts at index 3 and that is what is returned.

32. The answer is A.
    - acArray is an array of objects. Initially, all the entries in the array are null. That is, they do not reference anything. If a program tries to use one of these entries as if it were an object, the program will terminate with a `NullPointerException`.
    - To avoid the `NullPointerException`, we must not attempt to use a null entry as if it contained a reference to an object.
    - Option B is incorrect. It uses `ArrayList` syntax, not array syntax, so it will not even compile.
    - Options C and D are incorrect. They attempt to call getData() on a null entry and so will terminate with a `NullPointerException`.
    - Option A is correct. It checks to see whether the array entry is null before allowing the System.out.println statement to treat it

as an object.

33. The answer is A.
    - The Binary Search algorithm should not be applied to this array! Binary Search only works on a sorted array. However, the algorithm will run; it just won't return reasonable results. Let's take a look at what it will do.
    - First look at the middle element. That's 100. $50 < 100$, so eliminate the second half of the array.

9	100	11	45	76	~~100~~	~~50~~	~~1~~	~~0~~	~~55~~	~~99~~

    - Look at the middle element of the remaining part; $50 > 11$, so eliminate the lower half.

9	~~100~~	~~11~~	45	76	~~100~~	~~50~~	~~1~~	~~0~~	~~55~~	~~99~~

    - 50 does not appear in the remaining elements, return -1 (incorrectly, as it turns out).

34. The answer is C.
    - After the array is instantiated, it looks like this:

0	1	2
3	4	5
6	7	8

    - The `for` loop will execute three times: x = 0, x = 1, and x = 2.
    - You should recognize that the code in the loop is a simple swap algorithm that swaps the value at nums[x][0] with the value at nums[x][2].
    - The result is that in each row, element 0 and element 2 will be swapped.
    - Our final answer is:

2	1	0
5	4	3
8	7	6

35. The answer is A.

- Let's trace the loop and see what happens. On entry we have:

list (which will not change):

2	4	3	3	2	5	1

newList:

2

- i = 1. In pseudocode, the `if` statement says "if (element 1 of list > the last element in newList) add it to newList". Since 4 > 2, add 4 to newList, which becomes:

2	4

- i = 2. "if (element 2 of list > the last element in newList) add it to newList" 3 is not > 4 so we do not add it.
- i = 3. "if (element 3 of list > the last element in newList) add it to newList" 3 is not > 4, we do not add it. (By this point, you may recognize what the code is doing and you may be able to complete newList without tracing the remaining iterations of the loop.)
- i = 4 "if (element 4 of list > the last element in newList) add it to newList" 2 is not > 4, we do not add it.
- i = 5. "if (element 5 of list > the last element in newList) add it to newList" 5 > 4, add 5 to newList which becomes:

2	4	5

- i = 6. "if (element 6 of list > the last element in newList) add it to newList" 1 is not > 5, so we do not add it.
- We have completed the loop; we exit and print newList.

36. The answer is E.

- After the array is instantiated, it is filled with 0s because that is the default value for an int, so anything we don't overwrite will be a 0.
- Let's look at the first loop. r goes from 0 to 4, so all rows are affected. c starts at r+1 and goes to 4, so not all columns are affected. When r = 0, c will = 1,2,3,4; when r = 1, c will = 2,3,4; when r = 2, c will = 3,4; when r = 3, c will = 4. That will assign 9 to the upper-right corner of the grid like this.

0	9	9	9	9
0	0	9	9	9
0	0	0	9	9
0	0	0	0	9
0	0	0	0	0

- In the second loop, d goes from 0 to 4, so all rows are affected, and c always equals r, so the diagonals are filled in with their row/column number.

0	9	9	9	9
0	1	9	9	9
0	0	2	9	9
0	0	0	3	9
0	0	0	0	4

- Now we just have to look for the three cells specified in the problem.

37. The answer is D.
    - The first thing to do is to identify the sorting algorithm implemented by mysterySort.
    - We can tell it's not Merge Sort, because it is not recursive.
    - There are several things we can look for to identify whether a sort is Insertion Sort or Selection Sort. Some easy things to

look for:

- In the inner loop, Insertion Sort shifts items over one slot.
- After the loops are complete, Selection Sort swaps two elements.

- We can see items being shifted over and we can't see a swap, so this is Insertion Sort.
- Let's look at the state of the array in table form. Before the sort begins, we have this. The sort starts by saying that 9 (all by itself) is already sorted.

9	1	3	0	2

- The first iteration through the loop puts the first two elements in sorted order.

1	9	3	0	2

- The second iteration through the loop puts the first three elements in sorted order.

1	3	9	0	2

- Notice that in Insertion Sort, the elements at the end of the array are never touched until it is their turn to be "inserted." Selection Sort will change elements all through the array.

38. The answer is C.

- This is a recursive method. Let's trace the calls. The parts in *italics* were filled in on the way back up. That is, the calls in the plain type were written top to bottom until the base case returned a value. Then the answers were filled in *bottom to top.*
  - puzzle (3, 4) = 3 * puzzle (3, 3) = 3 ** 27* = ***81*** which gives us our final answer
  - puzzle (3, 3) = 3 * puzzle (3, 2) = 3 ** 9* = *27*
  - puzzle (3, 2) = 3 * puzzle (3, 1) = 3 ** 3* = *9*
  - puzzle (3, 1) Base Case! return 3

- It is interesting to notice that this recursive method finds the first parameter raised to the power of the second parameter.

39. The answer is D.
    - This nested `for` loop is traversing the array in column major order. I can tell this is the case because the outer loop, the one that is changing more slowly, is controlling the column variable. For every time the column variable changes, the row variable goes through the entire row.
    - In addition, although the outer loop is traversing the columns from least to greatest, the inner loop is working backward through the rows.
    - Since we increase val by 2 each time, values are being filled in like this:

8	18	28	38	48	58
6	16	26	36	46	56
4	14	24	34	44	54
2	12	22	32	42	52
0	10	20	30	40	50

    - You probably didn't need to fill in the whole table to figure out that table[3][4] = 42.

40. The answer is E.
    - Since this is a Selection Sort algorithm, we know that the inner loop is looking for the smallest element.
    - small is storing the index of the smallest element we have found so far. We need to find out if the current element being examined is smaller than the element at index small, or, in other words, if arr[j] is less than arr[small].
    - Option C would be correct if we were sorting an array of ints, but < doesn't work with strings. We have to use the method `compareTo`. `compareTo` returns a negative value if the calling

element is before the parameter value. We need:
`if (arr[j].compareTo(arr[small] < 0))`

Part II (Free Response)

Answers and Explanations

Please keep in mind that there are multiple ways to write the solution to a Free-response question, but the general and refined statements of the problem should be pretty much the same for everyone. Look at the algorithms and coded solutions, and determine if yours accomplishes the same task.

General Penalties (assessed only once per problem):

-1 using a local variable without first declaring it

-1 returning a value from a void method or constructor

-1 confusing array/ArrayList access

-1 overwriting information passed as a parameter

-1 including unnecessary code that causes a side effect such as a compile error or console output

1. Password

(a) **General Problem:** Write the `isValid` method for the `Password` class.

Refined Problem: Determine if a password is valid by comparing the checking if the password has the correct length and the correct combination of letters and symbols.

Algorithm:

- Determine the length of the password.
- If the length is not between the minimum length and the maximum length, then return false. Otherwise, continue checking.
- Initialize counters for the number of uppercase letters, lowercase letters, and symbols to zero.

- For each character in the password increment the appropriate counter.
- If the counter for each isn't zero (meaning there was at least one occurrence) and the sum of all three counters equal the length of the password (meaning there were no symbols outside those that are acceptable), then it is a valid password.
- Return the result.

Java Code:

```
public boolean isValid(String password)
{
if (password.length() < minLength || password.length() > maxLength)
      return false;
int countUpper =0, countLower = 0, countSymbol = 0;
for (int i = 0; i < password.length(); i++)
{
      String letter = password.substring(i, i+1);
      if (upper.indexOf(letter) > -1)
           countUpper++;
      else if (lower.indexOf(letter) > -1)
            countLower++;
      else if (symbols.indexOf(letter) > -1)
            countSymbol++;
  }
return (countUpper > 0) && (countLower > 0) && (countSymbol > 0)
        && (countUpper + countLower + countSymbol == password.length());
  }
```

Common Errors:

- Not taking into account that there might be characters other than letters or acceptable symbols.
- Not checking for the length of the password.
- Not initializing or declaring the necessary variables.

(b) **General Problem:** Write the `generatePassword` method for the `Password` class.

Refined Problem: Use the random number generator to choose upper case, lower case, and symbols from the given strings. Determine if the generated password is valid by using the method written in part (a).

Algorithm:

- Concatenate the upper case, lower case and symbol strings.
- Determine if a valid password has been generated.
  - Use the random number generator to choose a valid length of the password.
  - Use the random number generator to randomly choose a letter from the concatenated upper/lower/symbol string.
  - Continue randomly choosing letters until the string has reached the desired length.
  - Test to see if the password is valid.
- If the password is not valid, start the process again until a valid password has been generated.
- Return the generated password string.

Java Code:

```
public String generatePassword()
{
      String password = "";
      String allPossible = upper + lower + symbols;
      int position;
      while (!isValid(password))
      {
        password = "";
        int length = (int) (Math.random() * (maxLength - minLength + 1)) + 1;
        for (int i = 1; i <= length; i++)
        {
            position = (int) (Math.random() * allPossible.length());
            password += allPossible.substring(position, position+1);
        }
      }
      return password;
}
```

Common Errors:

- Not checking to see if the generated password is valid.
- Not randomly choosing a length for the new password.
- Not initializing or declaring necessary variables.
- Not returning the generated password.

Scoring Guidelines: Password

Part (a) **isValid** **5 points**

+1 Determine if the password has a valid length
+1 Examine each character of the password one at a time
+1 Determine if the password has upper case, lower case, and valid symbols
+1 Ensure there are no other characters besides upper case, lower case, and valid symbols
+1 Return the appropriate value

Part (b) **generatePassword** **4 points**

+1 Calls the `isValid` method correctly
+1 Randomly chooses a length of the password
+1 Randomly choose upper case, lower case, and valid symbols
+1 Returns the generated password

Sample Driver:

There are many ways to write these methods. Maybe yours is a bit different from our sample solutions and you are not sure if it works. Here is a sample driver program. Running it will let you see if your code works, and will help you debug it if it does not.

Copy PasswordDriver into your IDE along with the complete `Password` class (including your solutions).

```
public class PasswordDriver
{
    public static void main(String[] args)
    {
      Password pass1 = new Password(3,15);
      String pass = "HelloWorld!";
      boolean valid = pass1.isValid(pass);
      System.out.println(pass + " is valid: " + valid);
      pass = pass1.generatePassword();
      System.out.println(pass);

      Password pass2 = new Password(3,15);
      pass = "#APComputerScienceRocks";
      valid = pass2.isValid(pass);
      System.out.println(pass + " is valid: " + valid);
      pass = pass2.generatePassword();
      System.out.println(pass);

    }
}
```

2. ISBN

General Problem: Write an `ISBN` class.
Refined Problem: Define necessary instance variables, a constructor with one integer parameter, and two methods.

Algorithm:

- Declare a private instance variable to hold the 9-digit ISBN number.
- Define a constructor with one integer parameter and assign the value of the parameter to the instance variable.
- Write a `calculateCheckDigit` method that returns a string containing the check digit.
  - For each of the 9 digits in the ISBN number, find the product of digit and the weighted value.
  - Maintain a sum of those products.
  - Determine the check digit by using the formula given.
- Write a `generateNumber` method that concatenates the instance variable with the result of the call to the `calculateCheckDigit` method.

Java Code:

```
public class ISBN
{
   private int isbnNumber;

   public ISBN(int number)
   {
      isbnNumber = number;
   }

   public String calculateCheckDigit()
   {
          int temp = isbnNumber;
          int digit, product;
          int multiple = 2;
          int sum = 0;
          for (int i = 1; i <= 9; i++)
          {
               digit = temp % 10;
               temp /= 10;
               product = digit * multiple;
               sum += product;
               multiple++;
          }
          int check = 11 - sum % 11;
          if (check == 10)
               return "X";
          return "" + check;
   }

   public String generateNumber()
   {
          return isbnNumber + "-" + calculateCheckDigit();
   }
}
```

Scoring Guidelines: ISBN

ISBN class **1 point**

+1 Complete, correct header for ISBN

state maintenance **1 point**

+1 Declares a private instance variable capable of maintaining the 9-digit ISBN number

ISBN Constructor **2 points**

+1 Correctly formed header
+1 Sets appropriate state variables based on parameter

calculateCheckDigit **3 points**

+1 Correctly formed header
+1 Finds the sum of the products of digits and weighted values
+1 Returns correct check digit

generateNumber **2 points**

+1 Correctly formed header
+1 Returns 10-digit ISBN number

Sample Driver:

There are many ways to write these methods. Maybe yours is a bit different from our sample solutions and you are not sure if it works. Here is a sample driver program. Running it will let you see if your code works, and will help you debug it if it does not.

Copy ISBNDriver into your IDE along with the complete `ISBN` class (including your solutions).

```
public class ISBN Driver
{
   public static void main(String [] args)
   {
        ISBN book1 = new ISBN(126045491);
        String check1 = book1.calculateCheckDigit();     //  "6" will be returned
        String isbnNumber1 = book1.generateNumber();     // "126045491-6" will be
                                                         // returned

        ISBN book2 = new ISBN(030640611);
        String check2 = book2.calculateCheckDigit();     //  "X" will be returned
        String isbnNumber2 = book2.generateNumber();     // "030640611-X" will be
                                                         //  returned
   }
}
```

3. Train

(a) **General Problem:** Write the `getTotalWeight` method to find the total weight of the engine and train.

Refined Problem: The method should loop through each element of the `ArrayList` and add it to a sum variable.

Algorithm:

- Initialize a sum variable to 0.
- Loop through each element of the `ArrayList`.
- Add the value of the current element to the accumulated sum.
- Return the accumulated sum.

Java Code:

```
public double getTotalWeight()
{
     double weight = 0;
     for (Double car : trainCars)
        weight += car;
     return weight;
}
```

Common Errors:

- Not initializing the sum to 0.
- Not accessing each element of the `ArrayList`.
- Not returning the sum.

Java Code Alternate Solution:

```
public double getTotalWeight()
{
  double weight = 0;
  for (int i = 0; i < trainCars.size(); i++)
     weight += trainCars.get(i);
  return weight;
}
```

(b) **General Problem:** Write the `removeExcessTrainCars` method of the `Train` class that removes cars from the train until the train is light enough to be pulled by the engine.

Refined Problem: Calculate the initial weight of the train. If the weight exceeds the weight limit, TrainCars are removed from the end of the train until the weight is in the acceptable

range. All train cars that are removed are returned in an `ArrayList` in the order they are removed from the train.

Algorithm:

- Create an `ArrayList` of Double to hold the removed cars.
- While the total weight is greater than the Engine's maximum weight,
  - Remove a train car from the end of the train
  - Add it to the `ArrayList` to be returned
- Return the `ArrayList` of removed cars (which may be empty).

Java Code:

```
public ArrayList<Double> removeExcessTrainCars()
{
    if (getTotalWeight() < getMaximumWeight())
      return null;
  ArrayList<Double> removed = new ArrayList<Double>();

    while (getTotalWeight() > getMaximumWeight())
    {
       Double removedCar = trainCars.remove(trainCars.size() - 1);
       removed.add(removedCar);
    }
    return removed;
}
```

Common Errors:

- Not creating a temporary `ArrayList` when needed.
- Remember that remove also returns the removed element. You do not need a `get` followed by a `remove`.
- Not returning the temporary `ArrayList`.

Scoring Guidelines: Train

Part (a) getTotalWeight 4 points

+1 Declare and initialize a weight variable
+1 Accesses the weight of every element of the `ArrayList`; no bounds errors, no missed elements
+1 Calculates the total weight of the train
+1 Returns the accumulated weight

Part (b) removeExcessTrainCars 5 points

+1 Declares and instantiates an `ArrayList` for the removed values
+3 Removes necessary cars from the train
  +1 Writes a loop with an appropriate terminating condition
  +1 Removes a car from the end of the train
  +1 Adds the car to the end of the declared `ArrayList`
+1 Returns the correctly generated `ArrayList`

Sample Driver:

There are many ways to write these methods. Maybe yours is a bit different from our sample solutions and you are not sure if it works. Here is a sample driver program. Running it will let you see if your code works, and will help you debug it if it does not.

Copy TrainDriver and the complete `Train` class into your IDE. Add your `getTotalWeight` and `removeExcessTrainCars` methods to the bottom of the `TrainDriver` class.

```
import java.util.ArrayList;

public class TrainDriver
{
      public static void main(String[] args)
      {
            ArrayList <Double> trainCars = new ArrayList <Double>();
            trainCars.add(200000.0);
            trainCars.add(100000.0);
            trainCars.add(140000.0);
            trainCars.add(50000.0);
```

```
            trainCars.add(100000.0);
            trainCars.add(60000.0);

            Train t = new Train(475000, trainCars);
            double weight = t.getTotalWeight();
            System.out.println("The total weight is " + weight);
            ArrayList <Double> removed = t.removeExcessTrainCars();
            System.out.println("The removed train cars are " + removed);
        }
}

import java.util.ArrayList;

public class Train
{
    private double maxWeight;
    private ArrayList <Double> trainCars;

    public Train (Double max, ArrayList<Double> tc)
    {
     maxWeight = max;
     trainCars = tc;
    }

    public double getMaximumWeight()
    {
     return maxWeight;
    }


/* insert your code here */

}
```

4. Pixels

(a) **General Problem:** Write the `generatePixelArray` method to convert three 2-D int arrays into a 2-D array of `Pixel` objects.

Refined Problem: Use a nested `for` loop to instantiate a `Pixel` object for each element of the `Pixel` object array. Use the corresponding values in the red, green, and blue arrays as parameters to the Pixel constructor. When the loops are complete, return the completed array.

Algorithm:

- Create a 2-D array of type Pixel to hold the new `Pixel` objects. The dimensions of this array are the same as any of the color arrays because there is a precondition saying all the arrays must be the same size.

- Write a `for` loop that goes through all the rows of the new array.
  - Nested in that `for` loop, write a `for` loop that goes through all the columns of the new array. (These can be switched. Row-major order is more common, but column-major order will also work here.)
    - Instantiate a new `Pixel` object, passing the values in the red, green, and blue arrays at the position given by the two loop counters, and assign it to the element of the Pixel array given by the two loop counters.
- When the loops are complete and every element has been processed, return the completed Pixel array.

Java Code:

```
public static Pixel[][] generatePixelArray(int[][] reds, int[][] greens, int[][] blues)
{
   Pixel[][] pix = new Pixel[reds.length][reds[0].length];
   for (int r = 0; r < pix.length; r++)
      for (int c = 0; c < pix[r].length; c++)
         pix[r][c] = new Pixel(reds[r][c], greens[r][c], blues[r][c]);
   return pix;
}
```

(b) **General Problem:** Write a `flipImage` method that takes a 2-D array of `Pixel` objects and flips it into a mirror image, either vertically or horizontally.

Refined Problem: Create a new array to hold the altered image. Determine whether to flip the image horizontally or vertically. Write a nested `for` loop to move all the Pixels to their mirror-image location in the new array, either horizontally or vertically. When the loops are complete, return the new array.

Algorithm:

- Instantiate a 2-D Pixel array that has the same dimensions as the array passed as a parameter. This array will hold the altered image.

- Create an `if-else` statement with one clause for a horizontal flip and one for a vertical flip.
- If the flip is horizontal, write a nested `for` loop that goes through the array in row-major order.
  - For each iteration of the loop, an entire row is moved into its new "flipped" place in the altered array.
- Otherwise, the flip is vertical. Write a nested `for` loop that goes through the array in column-major order.
  - For each iteration of the loop, an entire column is moved into its new "flipped" place in the altered array.
- Return the altered array.

Java Code:

```
public static Pixel[][] flipImage(Pixel[][] image, boolean horiz)
{
    Pixel[][] flipped = new Pixel[image.length][image[0].length];
    if (horiz)
       for (int r = 0; r < image.length; r++)
          for (int c = 0; c < image[0].length; c++)
             flipped[r][c] = image[image.length - 1 - r][c];
    else
        for (int c = 0; c < image[0].length; c++)
           for (int r = 0; r <image.length; r++)
              flipped[r][c] = image[r][image[0].length - 1 - c];
    return flipped;
}
```

Common Errors:

- Watch off-by-one errors. Always think: do I want length or length - 1? Using variables, like in the alternate solution, can help you be consistent.
- Be sure you understand the difference between row-major and column-major order.
- Check your answer with both even and odd numbers of rows and columns.
- Create a new array to hold the “flipped” version. Do not overwrite the array that is passed in. This is called *destruction of persistent data* and incurs a penalty.

Java Code Alternate Solution:

- This solution only loops halfway through the array. It flips a pair of lines on each iteration.

  This solution also uses a few extra variables to keep things easier to read. This is not necessary but it reduces the amount of typing and the chance of an off-by-one error with length, as opposed to length – 1.

```
public static Pixel[][] flipImage(Pixel[][] image, boolean horiz)
{
   Pixel[][] flipped = new Pixel[image.length][image[0].length];
   int maxR = image.length - 1;
   int maxC = image[0].length - 1;
   if (horiz)
   {
      for (int r = 0; r <= maxR / 2; r++)
      {
         for (int c = 0; c <= maxC; c++)
         {
            flipped[r][c] = image[maxR - r][c];
            flipped[maxR - r][c] = image[r][c];
         }
      }
   }
   else
   {
      for (int c = 0; c <= maxC / 2; c++)
      {
         for (int r = 0; r <= maxR; r++)
         {
            flipped[r][c] = image[r][maxC - c];
            flipped[r][maxC - c] = image[r][c];
         }
      }
   }
   return flipped;
}
```

Scoring Guidelines: Pixels

Part (a) **generatePixelArray** **3 points**

+1 Instantiates a 2-D array of `Pixel` objects with the correct dimensions

+1 Instantiates a new `Pixel` object with the corresponding elements of the red, green, and blue arrays for every element of the array; no bounds errors, no missed elements

+1 Returns the properly constructed and filled array

Part (b) **flipImage** **6 points**

+1 Correctly determines whether the image should be flipped vertically or horizontally

+2 Flips the array horizontally

- **+1** Correctly repositions at least one element
- **+1** Correctly repositions all elements

+2 Flips the array vertically

- **+1** Correctly repositions at least one element
- **+1** Correctly repositions all elements

+1 Returns the correctly generated flipped array

Sample Driver:

There are many ways to write these methods. Maybe yours is a bit different from our sample solutions and you are not sure if it works. Here is a sample driver program. Running it will let you see if your code works, and will help you debug it if it does not.

Copy PixelDriver and the complete `Pixel` class into your IDE. Add your `generatePixelArray` and `flipImage` methods to the bottom of the `PixelDriver` class.

```
public class PixelDriver {

   public static void main(String[] args) {
      int[][] red = { { 0, 0, 0, 0 }, { 1, 1, 1, 1 }, { 2, 2, 2, 2 },
                      { 3, 3, 3, 3 } };
      int[][] green = { { 0, 1, 2, 3 }, { 0, 1, 2, 3 }, { 0, 1, 2, 3 },
                        { 0, 1, 2, 3 } };
      int[][] blue = { { 100, 100, 100, 100 }, { 100, 100, 100, 100 },
                       { 100, 100, 100, 100 }, { 100, 100, 100, 100 } };
      System.out.println("generatePixelArray test");
      System.out.println("Expecting:");
      for (int r = 0; r < red.length; r++) {
         for (int c = 0; c < red[r].length; c++)
            System.out.print("(" + red[r][c] + ", " + green[r][c] +
                  ", " + blue[r][c] + ") ");
         System.out.println();
      }
      Pixel[][] pix = generatePixelArray(red, green, blue);
      System.out.println("\nYour Answer:");
      printIt(pix);

      final boolean HORIZ = true;
      final boolean VERT = !HORIZ;
      System.out.println("\nflipImage - Horizontal test");
      System.out.println("Expecting:");
      for (int r = red.length-1; r >=0; r--) {
         for (int c = 0; c < red[r].length; c++)
            System.out.print("(" + red[r][c] + ", " + green[r][c] +
                  ", " + blue[r][c] + ") ");
         System.out.println();
      }
      System.out.println("\nYour Answer:");
      Pixel[][] pix2 = flipImage(pix, HORIZ);
      printIt(pix2);

      System.out.println("\nflipImage - Vertical test");
      System.out.println("Expecting:");
      for (int r = 0; r < red.length; r++) {
         for (int c = red[r].length - 1; c >= 0; c--)
            System.out.print("(" + red[r][c] + ", " + green[r][c] +
                  ", " + blue[r][c] + ") ");
         System.out.println();
      }
      System.out.println("\nYour Answer:");
      Pixel[][] pix3 = flipImage(pix, VERT);
      printIt(pix3);
   }

   public static void printIt(Pixel[][] pix) {
      for (int r = 0; r < pix.length; r++) {
         for (int c = 0; c < pix[r].length; c++) {
            System.out.print(pix[r][c] + " ");
         }
         System.out.println();
      }
   }
   // Enter your generatePixelArray and flipImage methods here
}
```

Scoring Worksheet

This worksheet will help you to approximate your performance on Practice Exam 1 in terms of an AP score of 1–5.

Part I (Multiple Choice)

Number right (out of 40 questions) = _____

Part II (Short Answer)

See the scoring guidelines included with the explanations for each of the questions and award yourself points based on those guidelines.

Question 1: Points Obtained (out of 9 possible) _____

Question 2: Points Obtained (out of 9 possible) _____

Question 3: Points Obtained (out of 9 possible) _____

Question 4: Points Obtained (out of 9 possible) _____

Add the points and multiply by 1.1111 _____ × 1.1111 = _____

Add your totals from both parts of the test (round to nearest whole number) Total Raw Score _____

Approximate conversion from raw score to AP score

Raw Score Range	AP Score	Interpretation
62–80	5	Extremely Well Qualified
44–61	4	Well Qualified
31–43	3	Qualified
25–30	2	Possibly Qualified
0–24	1	No Recommendation