# CS 312 – Sample Credit by Exam 2

**1. Short Answer 1 - Expressions. 1 point each, 10 points total.**

For each Java expression in the left hand column, indicate the resulting value in the right hand column.

**You must show a value of the appropriate type. For example, 7.0 rather than 7 for a double, "7" instead of 7 for a String, and '7' for a char.**

**Answers that do not indicate the data type correctly are wrong.**

A. `(3 / 2 == 0) + "H" + 2 * 5 + 1.5` _____

B. `73983 % 1000 / 100 + 10 * 5 * 1.0` _____

C. `10 / 3 + 17 % 30 + 2.5 % 2.0` _____

D. `(int) 2.752 * 10 + 5.0` _____

E. `"CS312".charAt(2) == '3'` _____

F. `12 % 3 != 0 || 3 == '3'` _____

G. `Math.floor(-2.31) + Math.round(-1.5)` _____

H. `"cat".toUpperCase() == "CAT"` _____

I. `"longhorns_or+tv".indexOf("or") * 2` _____

J. `"super_tas".indexOf("TAs") - 3` _____

**2. Code Tracing - 1 point each, 24 points total.** For each code snippet state the exact output to the screen. **Placed your answers to the right of the snippet and put a box around each answer.**
Assume all necessary imports have been made.
If the snippet contains a syntax error or other compile error, answer COMPILE ERROR.
If the snippet results in a runtime error or exception, answer RUNTIME ERROR.
If the snippet results in an infinite loop, answer INFINITE LOOP.

A.
```
int[] a1 = {5, 2, 4, 6, 1};
a(a1);
System.out.print(Arrays.toString(a1));

public static void a(int[] arr) {
     arr[0] = arr[3];
     arr[arr.length - 2] -= 3;
     arr[arr[1]] += 3;
     arr[arr.length / 2] += 5;
     arr[0] = arr[3];
     arr[3] = 7;
}
```

B.
```
double[] b1 = {.5, 2, 1.5};
double tot1 = 0.0;
for (double a : b1) {
     tot1 += a;
}
System.out.print(tot1 + " " + b1[1]);
```

C.
```
int c1 = 80;
int t2 = 0;
while (c1 > 0 && t2 < 20) {
     t2++;
     c1 /= 2;
}
System.out.print(c1 + " " + t2);
```

D.
```
String d1 = "CAT_DOG";
String d2 = "DOG";
String d3 = d1.substring(4);
String d4 = d2;
d2.substring(1);
System.out.print((d2 == d4) + " " + (d1 == d3));
```

E.

```
boolean[] es = new boolean[6];
System.out.println(es[2] + " " + (es[3] == es[4]));
```

F.

```
String[][] namTab = new String[10][4];
int ft = 0;
for (int r = 0; r < namTab.length; r++) {
     ft += namTab[r].length;
}
System.out.print(namTab.length + " " + ft);
```

G.

```
int[] g1 = {5};
g(g1);
System.out.print(Arrays.toString(g1));

public static void g(int[] arr) {
     for (int i = 0; i < arr.length; i++) {
          for (int j = 0; j < i; j++) {
               arr[i] += arr[j];
          }
     }
}
```

H. (uses method from part G)

```
int[] h1 = {2, 1, 4};
g(h1);
System.out.print(Arrays.toString(h1));
```

I. (uses method from part G)

```
int[] i1 = {1, 1, 1, 1, 1, 2};
g(i1);
System.out.print(Arrays.toString(i1));
```

J. (uses method from part G)

```
int[] j1 = {-3, 0, 5, -2, 7, -10};
g(j1);
System.out.print(Arrays.toString(j1));
```

K. (Consider the given enum)

```
System.out.print(CS_Area.values().length + " "
               + CS_Area.values()[2]);

public enum CS_Area {
     PROGRAMMING, THEORY, SYSTEMS
}
```

L.

```
int[] va = new int[3];
va[1]++;
va[va.length - 1]--;
va++
va[1] += va[2];
System.out.print(Arrays.toString(va));
```

M.

```
ArrayList<String> w = new ArrayList<>();
w.add("K");
w.add("P");
w.add("B");
w.add(1, "G");
w.add("G");
w.remove(2);
w.add(3, "A");
w.remove("G");
System.out.print(w);
```

N.

```
ArrayList<String> xs = new ArrayList<>(5);
System.out.print(xs.size());
```

For the short answer questions O through X consider these classes:

```
public class Room {
     private int seats;

     public Room(int s) { seats = s; }

     public void expand() { seats += 5; }

     public int getSeats() { return seats; }

     public String toString() { return "room: " + seats; }
}

public class Meeting extends Room {
     private boolean tele;

     public Meeting(int s) { super(s); }

     public void expand() {tele = true;}

     public String toString() {return super.toString() + " " + tele; }
}

public class Classroom extends Room {

     private int plugs;

     public Classroom(int p, int s) {
          super(s * 2);
          plugs = p;
     }

     public void add(int p) {
          plugs += p;
          expand();
          expand();
     }
}
```

O. Consider the following statements. For each, circle if it is legal or if it causes a syntax error.

```
Object obj = Room();               // legal       syntax error

Room r1 = new Classroom(10, 10);   // legal       syntax error
```

P. Consider the following statements. For each, answer circle it is legal or if it causes a syntax error.

```
Meeting m1 = new Object(5);              // legal       syntax error

Meeting m2 = new Classroom(10, 10);      // legal       syntax error
```

For each code snippet what is output?

Q.
```
Classroom c3 = new Classroom(10, 10);
c3.add(2);
System.out.print(c3);
```

R.
```
Room or = new Room(20);
o(or);
System.out.print(or);

public static void o(Room r) {
     r.expand();
     r.expand();
}
```

S.
```
Room pr = new Room(20);
p(pr);
System.out.print(pr);

public static void p(Room r) {
     r.expand();
     r = new Room(0);
}
```

T.
```
Room qr = new Room(10);
qr.seats -= 5;
System.out.print(qr.toString());
```

U.
```
Classroom cr1 = new Classroom(5, 5);
Classroom cr2 = new Classroom(10, 10);
cr2.expand();
cr2 = cr1;
cr1.expand();
cr2.expand();
System.out.print(cr1);
```

V.

```
Room cs = new Meeting(10);
cs.expand();
System.out.print(cs);
```

W.

```
Classroom ct1 = new Classroom(25, 10);
Classroom ct2 = new Classroom(25, 10);
System.out.print(ct1.toString().equals(ct2.toString()) + " "
          + ct1.equals(ct2));
```

X.

```
int[] ua = {-1, 2};
System.out.print(um(ua, 2) + " " + ua[1] + " " + um(ua, 1));

public static int um(int[] ar, int x) {
     ar[0] *= x;
     ar[1] += x;
     System.out.print(Arrays.toString(ar));
     return ar[0] + ar[1];
}
```

**3. Critters - 16 Points.** Implement the `JumpingBean` class from the Critters assignment. The only methods you must override from the `Critter` class are the **`fight`** and **`getMove`** methods. Do not override the other methods. Add a constructor if you believe it is necessary.

A `JumpingBean` sits still until it is involved in a fight. If it wins the fight it gets excited and starts moving. The `JumpingBean` knows it wins the fight if it is asked to move some time later.

When it moves after a successful fight the `JumpingBean` picks a random direction to move, either NORTH or WEST. Each direction has an equal chance of being chosen. The `JumpingBean` moves twice in the chosen direction and then resumes its behavior of sitting still when asked to move. If the `JumpingBean` wins a second fight, it picks a random direction again and moves 4 times in the chosen direction. Each time the `JumpingBean` wins a fight it picks a random direction to move and adds 2 to the number of times it moves in the chosen direction.

`JumpingBean`s fight with a SCRATCH if they are sitting still (CENTER), but `FORFEIT` if they fight when the next time asked to move they would move instead of staying put.

Example of `JumpingBean` movement (C for Center, N for North, and W for West)

```
CCCC (wins fight, random direction West) WW C (wins fight, random
direction West again) WWWW CCCCCC (wins fight, random direction N) NNNNNN
(fights after the last move North, when movement complete, so JumpingBean
sitting still, random direction N) NNNNNNNN CCC (wins fight, random
direction N) NNN (in a fight and loses)
```

```
public abstract class Critter {

     public boolean eat() {  return false; }

     public Attack fight(String opponent) {
          return Attack.FORFEIT;
     }

     public Color getColor() { return Color.BLACK; }

     public Direction getMove() { return Direction.CENTER; }

     public String toString() { return "?"; }

     // constants for directions
     public static enum Direction {
          NORTH, SOUTH, EAST, WEST, CENTER
     };

     // constants for fighting
     public static enum Attack {
          ROAR, POUNCE, SCRATCH, FORFEIT
     };
```

Include the class header and instance variables. Assume any necessary imports are already made.

**You may use the `Math.random` method, the `Critter` class, and the `Attack` and `Direction` enumerated types, but no other built in Java classes or methods.**

**4. Data Processing 18 points**. Write a method that determine how much money each person listed in a file has. For this question we are using Wizarding World denominations.
The denominations are Knut, Sickle, and Galleon. The conversion rates are:

29 Knuts = 1 Sickle
17 Sickles = 1 Galleon

The file uses abbreviations K, S, and G for Knut, Sickle, and Galleon.

Here is an example file:

```
Mike Scott 493 K 1 G 12 * 2465 K
Isabelle 20 G 13 S       11 K   103 K   15 S
Hailey Mothershead   85 k 12 s
Omer 'Omeezy' Mahtey   101 G  357 S     51 S  12 X
Carla R. 8 S    261 K    12  #  10 g
```

Each line of the file represents a single person. The format for each line is
<NAME> <N S><N S><N S>...

Names will be one or more tokens that are not integers. Each name is followed by 0 or more pairs of ints and symbols.

You may assume each int is greater than 0.
You may assume each int is followed by a non-integer symbol of length 1.
Some of the non-integer symbols may be invalid. Any symbol besides K, S, and G and its associated integer shall be ignored. Symbols are case sensitive, so k is not a valid symbol.
There are no blank lines in the file.

Your method shall print out one line of output for each line in the original file and the total value of the person's money in Galleons. For example, given the file above, your method would produce the following output:

```
Mike Scott 7.0 Galleons
Isabelle 21.878296146044626 Galleons
Hailey Mothershead 0.0 Galleons
Omer 'Omeezy' Mahtey 125.0 Galleons
Carla R. 1.0 Galleon
```

You may create new `Scanner`s and use the methods from the `Scanner` class. You may use `String` concatenation and the `length`, `charAt`, `contains`, and `equals` methods from the String class.

Do not use any other Java classes or methods.

Complete the method on the next page.

```
public static void printMoney(Scanner sc) {
```

**5. Arrays - 12 Points.** Write a method `getReversedSubList`. The method accepts 3 parameters, an array of `int`s, and two `int`s, start and stop, that indicate the starting index and stopping index of the sublist. The method creates and returns a new array with the elements from the indicated range of the given array, but with the elements in reverse order.

The sublist contains elements from the start index inclusive to the stop index exclusive, just like the `substring` method from the `String` class.

Examples of calls and results of various calls to `getReversedSubList`:

```
getReversedSubList({0, 1, 2, 3, 4, 5}, 2, 4) -> returns {3, 2}

getReversedSubList({0, 1, 2, 3, 4, 5}, 2, 3) -> returns {2}

getReversedSubList({0, 1, 2, 3, 4, 5}, 2, 2) -> returns {}

getReversedSubList({0, 1, 2, 3, 4, 5}, 6, 6) -> returns {}

getReversedSubList({0, 1, 2, 3, 4, 5}, 0, 6)
                                        -> returns {5, 4, 3, 2, 1, 0}
```

**You may create an array of `int`s to return.**

**You may not use any other Java classes or methods other than the array length field.**

**Assume 0 <= startIndex <= array length, 0 <= stopIndex <= array length,**
**and startIndex <= stopIndex**

```
public static int[] getReversedSubList(int[] vals, int startIndex,
                                                   int stopIndex) {
```

**6. Arrays - 16 points.** Write a method `insertElementsAtFront`. The method accepts two arrays as parameters and inserts all of the elements from the second array at the beginning of the first array.

Consider these examples:

```
first array:  {5, 12, 6, 3, 9, 0, -2}
second array: {4, -3, 2}
first array becomes:  {4, -3, 2, 5, 12, 6, 3}
```

Notice how the 3 elements of the second array are added to the front of the first array. The elements of the first array are shifted down appropriately with the last 3 elements of the original array being removed.

Other examples:

```
first array:  {5, 12, 6, 3, 9, 0, -2}
second array: {4}
first array becomes:  {4, 5, 12, 6, 3, 9, 0}

first array:  {2, 6, 1}
second array: {17, 19, 37, 41}
first array becomes:  {17, 19, 37}
```

Notice in the last example the second array is larger than the first array. The first N elements of the second array that will fit are inserted into the first array.

You may assume the second array length is greater than 0.

**The second array is not altered by the method.**

**You may not use any other Java classes or methods other than the array length field. This means you may not create any new arrays.**

**Your method should be reasonably efficient.**

```
// two is not altered as a result of this method call
public static void insertElementsAtFront (int[] one, int[] two) {
```

**7. ArrayLists - 16 points.** Write a method that given an `ArrayList` that contains `String` objects, removes all values from the `ArrayList` that contain a given target character in the first n (another parameter, an `int`) characters.

For example, if we had the following `ArrayList`:

```
["CK", "Omer", "Carla", "Fatima", "Anthony", "Aish", "Hailey", "Bri",
"Dayanny", "Chris", "Olivia"]
```

the target `char` of `'a'`, and `n` equal to `3`, then the `ArrayList` becomes:

```
["CK", "Omer", "Anthony", "Aish", "Bri", "Chris", "Olivia"]
```

All of the elements of the `ArrayList` that contained one or more `'a'` characters in the first 3 characters have been removed. The relative order of the remaining `String`s must remain the same.

The method returns the number of elements removed from the given `ArrayList`, in this case 4.

Recall the following methods from the `ArrayList` class. These are the only `ArrayList` methods you may use in your answer.

```
int size() Number of elements in the list.

get(int pos)  Returns the element at the given position.

remove(int pos) Remove and return the value at the given position.
```

You may use whatever `String` methods you want, but you should not create any new `String`s.

**Do not create or use any other data structures such as arrays or other `ArrayList`s.**

**Do not use any methods besides the given `ArrayList` methods and methods from the `String` class.**

```
public static int removeValues(ArrayList<String> list, char c, int n){
```

**8. 2D arrays - 18 Points.** Write a method that determine the number of coins a robot will collect when moving across a field represented by a 2d array of integers.

The robot follows a strict protocol when moving and collecting coins. The robot always starts out in the left most column. The starting row is sent as a parameter to the method. The robot can only move up, down, or right, one space at a time. It cannot move diagonally and it cannot move left. **The robot stops when it first enters the right most column of the 2d array, but does collect the coins in that cell.**

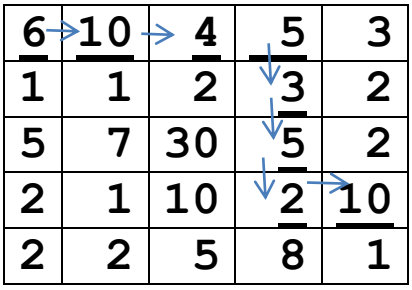
In the case of a tie the robot will prefer moves in the following order: up, down, right.

The robot never re-enters a cell it has already been in.

Following the movement rules the robot always moves to the adjacent cell with the highest number of coins. (This is a greedy algorithm, doing what appears to be best at the time. It turns out for some problems greedy algorithms give the best solution, but for others they do not.)

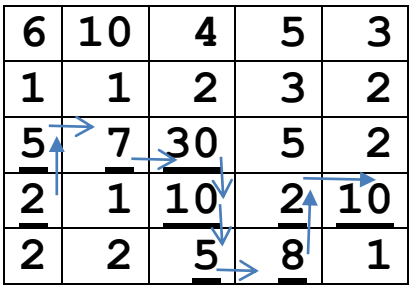
Consider the following example 2d array. The actual 2d array can of course, be very different. In the first example, assume the robot starts in the cell at the top row, the cell that contains 6 coins. Based on the robot's rules of movement it takes the path shown by the arrows and collects a total of 45 coins (6 + 10 + 4 + 5 + 3 + 5 + 2 + 10). Notice how the robot is faced with a tie when it reaches the first 5. The cell to the right and below both contain 3 coins. It breaks the tie by preferring to move down over moving to the right. Once the robot reaches the right most column, it stops.

| | | | | |
|---|---|---|---|---|
| 6 | 10 | 4 | 5 | 3 |
| 1 | 1 | 2 | 3 | 2 |
| 5 | 7 | 30 | 5 | 2 |
| 2 | 1 | 10 | 2 | 10 |
| 2 | 2 | 5 | 8 | 1 |




Here is another example with the same 2d array, but with the robot starting in the 4th row, the first 2 in the left most column:

| | | | | |
|---|---|---|---|---|
| 6 | 10 | 4 | 5 | 3 |
| 1 | 1 | 2 | 3 | 2 |
| 5 | 7 | 30 | 5 | 2 |
| 2 | 1 | 10 | 2 | 10 |
| 2 | 2 | 5 | 8 | 1 |




In this example the robot collects a total of 79 coins (2 + 5 + 30 + 10 + 5 + 8 + 2 + 10)

Complete the method on the next page. **Do not create any new arrays. Do not use any other Java classes or methods. <u>You may alter the given array</u>. All elements in the given array are initially greater than 0**.

```
/* The 2d array mat is a rectangular 2d array. All elements in mat are
initially greater than 0. Your method may alter the 2d array.
Assume 0 <= initialRow M mat.length */
public int coinsCollected(int[][] mat, int initialRow) {
```