UIL Written Test
JAVA Questions

Math

Math. Functions

`Math.abs``(int a) -> int:` Takes the absolute value of `a`.

`Math.floor``(double a) -> double:` Rounds `a` *down* to an integer (removes the decimal value)

`Math.ceil``(double a) -> double:` Rounds `a` *up* to an integer (remove the decimal value and add one)

Math.min()
Chooses the least of the two arguments and returns it

Math.max()
Chooses the highest of the two arguments and returns it

Math.random()
The java.lang.Math.random() returns a double value with a positive sign, greater than or equal to 0.0 and less than 1.0.

> **Example 1**
> To generate a random number between -5 inclusively and 46 inclusively
>
> ```
> int x = (int) (Math.random()*52 - 5);
> ```

Order of Operations

Java follows same order of operations as regular math PEMDAS as far as the math operations are concerned. For full operator precedence see https://introcs.cs.princeton.edu/java/11precedence/

Division

For division of integers, the outcome is the integer quotient without the remainder.

- 5/2=2
- 15/3=5

For division of doubles, the outcome is the decimal value of the quotient.

- 5.0/2=2.5
- 15.0/3=5.0

Modulus

The mod operator (%) gives the remainder of two divided values

- 13%4=1

- 39.02%9=3.02
- 4%10=4

Common Exceptions

ArithmeticException throws when arithmetic rules are broken, such as overflow and underflow (when the number goes out of range) and if trying to divide by zero

Example Problems

Question 6.

What is the output of the code segment to the right?

A) 0.111
B) 0.125
C) 6.000
D) 8.000
E) 9.000

```
double raw = -10.0 / 4;
double floor = Math.abs(Math.floor(raw));
double ceil = Math.abs(Math.ceil(raw));
out.printf("%.3f", Math.pow(floor, ceil));
```

Question 1.

Which of the following is equivalent to $3D_{16} * 13_8$?

A) 101001111_2 **B)** 22131_4 **C)** 1327_8 **D)** 761_{10} **E)** $29F_{16}$

Question 2.

What is the value of z in the code segment to the right?

A) 0.0 **B)** 2.0 **C)** 3.0 **D)** 6.0
E) No output due to an error.

```
double x = 6;
double y = 1/2;
double z = x * y;
```

Answer is A because when y is calculated the two integers `1` and `2` are divided, so the answer is also an integer which rounds down to `0` (double to integer conversions always round down) and then is converted to a double, and multiplying anything by `0` comes out to be `0`, but in this case as a double: `0.0`.

Primitive Types

There are **8 primitive data types** (see Java Primitive Data Types)

byte/short/int/long are 8/16/32/64 signed **two's complement** integers
float/double are 32/64-bit floating point numbers
char is a single 16-bit unicode character
boolean has only two possible values - true or false

To convert to/from **two's complement** see Mr. Mueller's slides.

In addition to the eight primitive data types listed above, the Java programming language also provides special support for character **strings** via the java.lang.String class. Enclosing your character string within double quotes will automatically create a new String object; for example, String s = "this is a string";. String objects are immutable, which means that once created, their values cannot be changed. The String class is not technically a primitive data type, but considering the special support given to it by the language, you'll probably tend to think of it as such.

Example Problems

Question 14.

Which pair of Java primitive data types occupies the same number bits of storage in memory?

A) `byte, char` **B)** `int, double` **C)** `short, char` **D)** `float, double` **E)** `short, float`

Answer is C because short and char both have a size of 16 bits. The byte gets 8 bits, int and float get 32 bits, and double gets 64 bits.

What is the output of

```
float f = 0.0f;
for(int i = 0; i < 10; i++) {
     f += 0.1f;
}
System.out.println(f == 1.0f);
```

?

Answer: false.

The reason is that floating point numbers are in base 2 not base 10, and 0.1 does not round perfectly into base 2 (just like how ⅓ is 0.3333333333 in base 10). There are rounding errors when adding it ten times. These rounding errors are 100% predictable and you can learn them, as Java follows IEEE standards for floating point numbers. The real value of `f` at the end of the loop is 1.0000001.

What is the decimal value of the number to the right if it is being stored as a signed byte data byte? $\mathbf{10101010_2}$

Since the number is stored in two's complement and is negative (i.e. top bit is 1) we determine the numbers magnitude by taking the one's complement of 10101010_2 to get 01010101_2, adding 1 to get 01010110_2, converting to decimal to get 2+4+16+64 = 86. Thus the answer is **-86**.

Miscellaneous

printf formatting

This type of formatting is available in many programming languages
https://www.cs.colostate.edu/~cs160/.Summer16/resources/Java_printf_method_quick_reference.pdf

casting to (int)

Casting to an int implicitly drops any decimal. No need to call Math.floor() (assuming positive numbers) Simply typecast with (int), e.g.:

```
System.out.println((int)(99.9999)); // Prints 99
```

It does have a different behavior from `Math.floor` which rounds towards negative infinity

For Loops

There are two types of for loops in Java

The generic:

```
for (int i=0; i<10;i++){}
```

This will run 10 times from 0<=i<10

And the more complex:

```
for (int myint : list){}
```

This will run list.size() times and will store each element of the list into int myint for to use.

Below are two list which both do the same thing

```
List<String> mylist = new List<String>(); // Making a list
mylist.add("sup1"); // Adding stuff to the list
mylist.add("sup2");
mylist.add("sup3");

for (String string : list) { // LIST A
      System.out.println(string);
}

for (int i=0; i<list.size();i++) { // LIST B
      System.out.println(list.get(i));
}
```

This prints out

```
sup1
sup2
sup3
```

```
sup1
sup2
sup3
```

Example Problems

Question 18.

What is the output of the code segment to the right?

A) `[10, 10, 9, 7]`

B) `[6, 9, 9, 6]`

C) `[6, 6, 5, 3]`

D) `[10, 14, 15, 13]`

E) No output due to an error.

```
int[][] table = new int[4][5];
int[] rows = new int[table.length];

for (int i = 0; i < table.length; i++)
  for (int j = i; j < table[i].length; j++)
    table[i][j] = i + j;

int i = 0;
for (int[] row : table) {
  int tot = 0;
  for (int n : row)
    tot += n;
  rows[i++] = tot;
}
out.println(Arrays.toString(rows));
```

Arrays/Arraylists

Arrays:

- collection of variables of the same type
- **Declaration:** int[] intArray;
- **Instantiation:** intArray = new int[10];
- **Literal:** int[] ints2 = { 1,2,3,4,5,6,7,8,9,10 };
- **Accessing:** intArray[0] = 0;
  int firstInt = intArray[0];
- **Multidimensional**: int[][] intArray = new int[10][20];

Arraylist:

- Like arrays but with dynamic size
- **Instantiation:** ArrayList al = new ArrayList();
- **Adding Values:** al.add("C");
- **Removing Values:** al.remove("F");
- **Accessing:** al.get(index);

Example Problems:

QUESTION 18

What is output by the code to the right?

A. `[4, 2]` B. `[4]` C. `[2]`

D. `[2, 4]` E. `[0, 2, 4]`

```
ArrayList<Integer> nums =
                        new ArrayList<Integer>();
nums.add(2);
nums.add(4);
System.out.println(nums);
```

QUESTION 23

What is output by the code to the right?

A. 1234 B. 521

C. 3632 D. 2521

E. There is no output due to a an `ArrayOutOfBoundsException`.

```
int[] list = {2, 5, 2, 1};
for( int val : list )
  System.out.print( val + 1 );
```

Alex Example:

What is outputted by the code to the left?

A.
B.
C.
D.