Integer.MAX_VALUE

- **a constant**
- **holds the largest value for the int data type**
- **$2^{31} - 1$**

Integer.MIN_VALUE

- **a constant**
- **holds the smallest value for the int data type**
- **-2^{31}**

static variables

- **variable is connected to the whole class, not individual objects**
- **all variables of the class share this one variable**
- **with the keyword *final* it is used to create a constant**
- **also called*: class variables***

final

- keyword used to make a variable a constant

ArrayList

- resizable array
- uses the *List* interface
- only holds objects
- built in methods for inserting, deleting, etc
- Can use the for-each loop or regular for-loop

```
int x = 7;
x = x /2;
```

what does x equal?

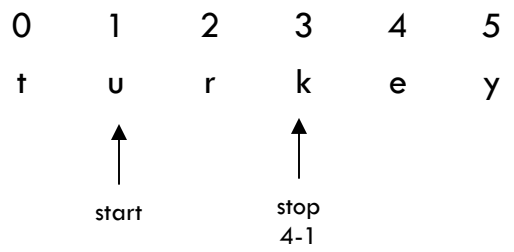
- answer: 3
- Remember that with integer division the decimal is cut off, no rounding happens

String s = "turkey";

what is:

s.substring(1, 4);

- **answer: *urk***
- **remember:**
  - **Strings start at 0**
  - **substring stops at the second number –1**



int x = 3 % 7;

what is x?

- **answer: 3**
- ***7* divides into 3 zero times, leaving 3 as a remainder**

Which is correct?

A. int x = (double) 4.5;
B. int x = (int) 4.5;
C. int x = 4.5;

- **answer: B**
- **Since 4.5 is a double you must use a *numeric cast* to change it to an int.**

String w = "ans: "+ 4 + 5;

What is w?

- **answer: ans 45**
- **Since the first thing after the = is a String, java uses concatenation**
- **so it changes 4 to a String, puts it on the end of ans , then changes 5 to a String and puts it at the end**

String concatenation

- **What Interface does it implement?**
- **What methods are included?**

What are:

A) \\
B) \n
C) \"

- **called *escape sequences***
- **used for some characters that cannot be typed\displayed easily**

**A) **
B) new line
C) "

Short circuit evaluation

- Java doesn't always test both halves of a Boolean operation—sometimes it uses a shortcut
- && (and): if the first part is false, the whole thing will be false, so it never tests the second part
- || (or): if the first part is true, the whole thing will be true, so it never tests the second part

What is output?

```
for (int a = 1; a < 5; a++){
        for (int b = a; b < 5; b += a)
                System.out.print(b);
        System.out.println();
}
```

- answer:

```
1 2 3 4
2 4
3
4
```

- Be careful! The first time a = 1, so the second loop counts by 1. Next time a = 2, so it counts by 2, etc.
- For these problems [ay special attention to where they start, where they stop and what they count by.

Why is it best to use .equals instead of == with objects?

- == only does a primitive test for equality
- in objects this means it is testing the memory address, not the values stored in the object
-

overloading

- **In the same class, many methods with the same name**
- **Java tells them apart using the *signature***
- **Signature: the number a type of parameters**
- **CANNOT use the methods return type to tell them apart**

Integer

- **Wrapper class that holds ints**
- **used to store ints in an ArrayList**
- **holds the MAX_VALUE and MIN_VALUE**

simple arrays

- **array can hold primitive types or class types**
- **all elements are of the same type**
- **not resizable**
- **Use the regular for-loop**
- **Use .length to find the size**

new

- **creates a reference to an object in memory**
- **calls the constructor in the object**

constructor

- **builds an object in memory**
- **sets up all the variables in the objects**
- **has the same name as the class**
- **NEVER has a return type or void**
- **can *overload* the constructor—Java tells them apart by the number and type of the parameters**

void

- **means a method does not return a value**

public int [] doStuff()

What type does this method return?

- An array of integers
- If this is a free response question the first thing you should do is:

```
int [] list = new int [10];
```

```
return list;
```

- This is usually worth 1/2 to 1 point!

What is returned by the call mystery(5)?

```
public static int   mystery(int num){
    if (num ==0)
        return 0;

    return num + mystery(num - 1);
    }
```

How about mystery (3)? mystery(2)?

- answer: 15
- mystery(3)? 6
- mystery(2)? 3

```
int list [] = { 5, 7, 2, 4};
for (int i = 0; i < list.length; i++)
    list [i] = list[i] * 2;
```

What is stored in *list* after this loop?

- answer:

0	1	2	3
10	14	4	8

null

- **What Interface does it implement?**
- **What methods are included?**

int stuff [] [] = new int [3][5];

How many columns?
How many rows?

Write the for loops to initialize this to:

01234
12345
23456

- **2-D arrays are *row-major*—> the rows come first.**
- **rows = 3, columns = 5**

```
for(int r = 0; r < stuff.length; r++)
    for (int c = 0; c < stuff[r].length; c++)
            stuff[r][c] = r + c;
```

public vs. private

- **Public means the variable or method can be accessed outside the class, private means it cannot**
- **On the AP exam all variables in a class are expected to be private**

//

- **Create a one-line comment**

/* */

- **Makes a comment**
- **Can block out several lines of code**

Simplify:

!(x != 5 && y <=0)

- **answer: (x == 5 || y>0)**
- **This is an example of DeMorgan's Law**
- **To simplify you distribute the ! and take the opposite of each operation**
- **Be careful! The opposite of > is <=**
- **Review this - there are always a few of these in the MC section - this should be a fast and easy question**

What is 1101_2 in base 10?

- answer: 13
- You should know binary, octal and hex. They are in the AP course description.

int x[];

Assume this array has been initialized. Write a loop to find the index of the largest element.

```
int maxIndex = 0;

for (int i = 1; i < x.length; i++){
        if (x[maxIndex] < x[i])
                maxIndex = i;
}
```

- Why do we set maxIndex to 0 before the loop?
- Why does the loop start at 1?

static methods

- method connected to the class, not an object
- ex: Math.random
- You do not declare a variable of Math type to get to the random method

interface

- **Used to define a set of behavior for a group of classes**
-
- **example: List interface, Comparable interface**

How are abstract classes different than interfaces?

- **Interfaces cannot have variables or code in methods. They can only have constants and a list of abstract methods**
- **Abstract classes can have some code along with the abstract methods**

You have an abstract class called first. A child class called second extends first. What must be true for second to NOT be abstract?

- **second must include code for all of the abstract methods in first.**
- **If second does not have code for all the abstract methods in first, second is also abstract**
- **The abstract methods are like a to-do list, once a child class has code for all the abstract methods, the abstraction is lifted**

implements

- **Used to show a class uses an interface**
- **example:**

public class Binary implements Comparable{

Bug b1 = null;
b1.act();

What's wrong?

- **Since b1 is null, it does not point to a location in memory. We cannot use any methods on a null object.**
- **throws a** NullPointerException,

What is output if x = 7;

```
if( x % 2 ==1)
    System.out.print("A");
else
    System.out.print("B");
```

- **answer: A**
- **% finds the remainder**
- **%2 finds if a number is even or odd**

extends

- **Connects a child class to it's parent**
- **The child class *inherits* all the features of the parent class**

- **Example:**

public class RockMonster extends Monster

super

- **calls the constructor of the parent class**
- **must be the first line in the child class' constructor**

abstract class

- **A class set up to be a parent to a subclass**
- **can have abstract methods, these are methods with no code**
- **an abstract class cannot be instantiated (cannot create an object from it)**
- **If the child class does not implement all the abstract methods it is also abstract**

class

- **template for a object**
- **can include variables and methods**

Hints:

- **According to the AP Java subset *all* variables should be private!**
- **On some old free response questions there has been a 1/2 deduction for not using private**

Object

- **All classes are children of this class**
- **has a toString and equals method**
  - **toString returns the variable's memory address**
  - **equals tests if two variables memory addresses are the same**
- **the equals and toString methods are usually rewritten in child classes**

object

- **a variable of a class type**
- **can hold data (variables) and have methods**