Name: ________________

AP Computer Science

Everything You Need to Know

1. **How long a minute is, depends on which side of the bathroom door you're on. ~Zall's Second Law**

In other words…save time!

- Do not read a problem top to bottom! As nuts as it sounds it's true. **Before** you read a long section of code **Read the Question!** No point in reading all that code if you don't know what you are looking for. Plus, they sometimes add a lot of stuff you don't need.

- Think about what they are really trying to get at.
    - Loop problems are often about WHERE a loop ends. You can often just look at the boundary cases (starting and stopping points)

    - if-else problems are testing if you understand nesting and the { }. Where one of stops and the next begins. This means you can skip sections of code.

Try It!

Consider the following method:

```
Public void conditionalTest (int a, int b) {
        if(( a > 0) && (b>0)) {
                if  (a>b)
                        System.out.println("A");
                else
                        System.out.println("B");
        } else if ((b < 0) || (a <0 ))
                        System.out.println("C");
        else
                        System.out.println("D");

}
```

Read this first 👉

What is printed as a result of the call conditionalTest (3, -2)?

A) A
B) B
C) C
D) D
E) Nothing is printed.

Notice if you read the question first then you can skip a lot of this code. Do it right and you only read 4 lines, instead of 10!!

2. **Plug and Play**

- Sometimes the easiest way to solve a problem is to plug in some known values. The trick is picking good values.
- On loop problems try boundary cases…where the loop should stop and start. Does it do what it is supposed to?
- On array problems…make a smaller array of 3 - 4 items:

Try it! Don't forget: Read the question BEFORE you start on the code:

```
private int[] arr;
//precondition : arr.length > 0
public void mystery () {
        int s1 = 0;
        int s2 = 0;
        for (int k = 0; k < arr.length; k++){
                int num = arr[k];
                if ((num >0) && (num % 2 == 0)
                        s1 += num;
                else if (num <0)
                        s2 += num;
                }
                System.out.println(s1);
                System.out.println(s2);
        }
```

Which of the following best describes the value of s1 output by the method *mystery*?

A) The sum of all positive values in arr
B) The sum of all positive even values in arr
C) The sum of all positive odd values in arr
D) The sum of all values greater than 2 in the arr
E) The sum of all negative odd values in arr

Huh? Since the answers all have to do with even/odd & positive/negative try the following array:

-9	**7**	**2**	**-8**	**6**	**5**	**1**

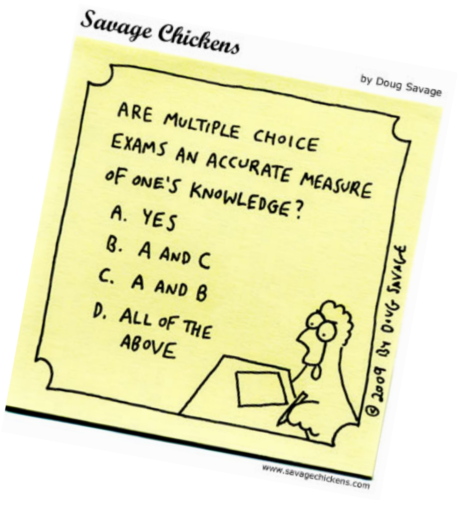
What answer does this give?

Also, if you read the question first, you knew to ignore all that code about s2.

3. **Know the Lingo**

If you don't understand what it is asking, **how are you going to answer?**

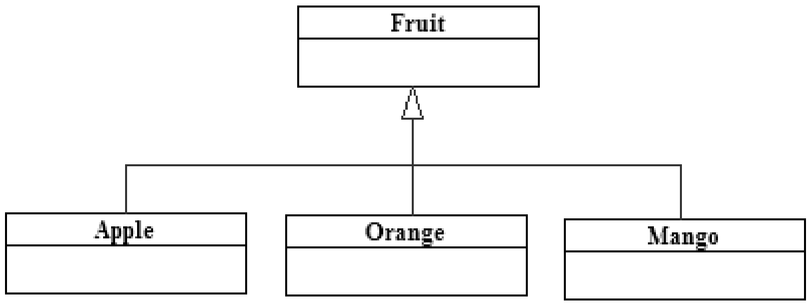
- **Compare and contrast Classes and Objects.** How are they related?



- What does **static** do to a:
    - method?
    - variable?
- **Primitive** and **class** data:
    - Give examples of each:
        - primitive:
        - Class:
    - describe the difference in how these are passed as **parameters**
- **Constructors**
    - What are they?
    - What is constructor chaining?
    - With inheritance…what's the gotcha when using a constructor from a super class?
- **Interfaces**
    - Can they contain constructors?
    - Can they be instantiated?
    - While we are on the subject, what the heck is **instantiated**?
    - All methods in an interface are abstract- true or false?
- **Public** and **Private**
    - What impact do they have on inheritance?

4. Inherit the Wind!

- **Fact**: Inheritance is a major topic on the exam
- For Example



  - An Apple ***has a*** seed, so they would not use inheritance.
  - An Apple ***is a*** Fruit, so you would use inheritance
  - Fruit is the parent, Apple is the child
  - What does Apple inherit from Fruit?
  - How would Apple access the constructor in Fruit?
  - How does public/private info impact Apple and Fruit?

```
public class Vehicle {
        private int numWheels;

        public Vehicle (int n) {
                numWheels = n;
        }
}
```

Write a class Scooter that extends Vehicle. The constructor should set the number of wheels to 2.

5. History Repeating

- These kinds of problems are about keeping track of what variables equal when.
- MAKE A CHART!! Writing things down makes them easier to remember.
- **Loops**

```
int num1 = 0;
int num2 = 3;

while ((num2 != 0) && ((num1/num2) >= 0)) {
          num1 = num1 + 2;
          num2 = num2 – 1;
}
```

What are the values of num1 and num2 after the while loop completes its execution?

1. num1 = 0, num2 = 3
2. num1 = 8, num2 = -1
3. num1 = 4, num2 = 1
4. num1 = 6, num2 = 0
5. The loop will never complete its execution because a division by zero will generate an ArithmeticException

- **Recursion**: a function that calls itself

```
Private static void recur (int n) {
          if (n != 0) {
                    recur (n-2);
                    System.out.print(n + " ");
          }
}
```

What will be printed : recur(7) ?

A) -1 1 3 5 7
B) 1 3 5 7
C) 7 5 3 1
D) Many numbers will be printed because of infinite recursion.
E) No numbers will be printed because of infinite recursion.

** There are always a few recursion problems on the Multiple Choice section.

6. A few picky details:

There are a few things guaranteed to be in there:

- De Morgan's law
  - This lets you simplify Boolean Expressions
  - Basically this is distribution :
    - o ! ((a < b) || (c == a)) → (a >= b) && (c != a)

Try it:

```
! ( y != x) || ((x ==3) && (y < 0)))
```

- **When chuck Norris does division, there are no remainders.**

  Remember that :

```
int x = 9/2;
System.out.println(x);          ← displays 4 NOT 4.5
```

- **Know your limit: Integer.MIN_VALUE and Integer.MAX_VALUE**

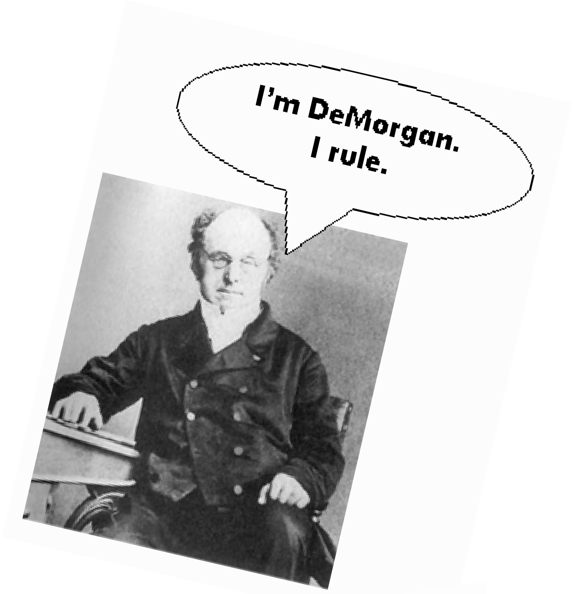
Integer.MIN_VALUE	= smallest possible integer
Integer.MAX_VALUE	= largest possible integer

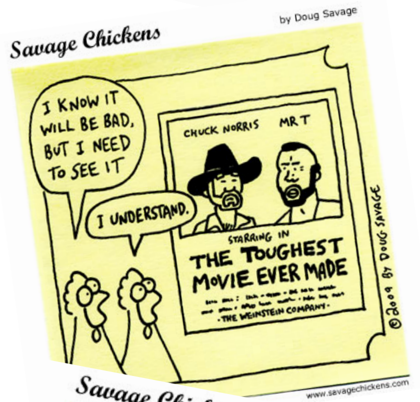
Use the array: int list [] ;

Complete the following method to return the smallest value less than n in the array **list.**

```
public int smallerThan( int n) {
   //Precondition
   //Postcondition: returns the smallest value in the array less
  //      than n, if no value is smaller it returns a 0
```

Why use MIN_VALUE here?





7. Interfaces (again)

Lets review: What *is* an interface?

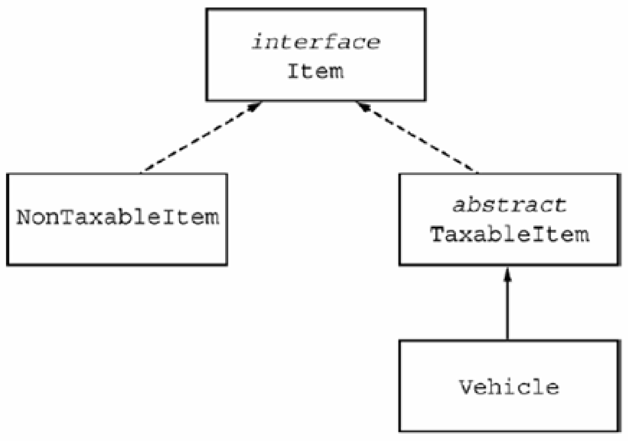
- Picky detail: All methods default to **public** (unless coded otherwise)

- Example: **Comparable**:
  - One of the Java standard interfaces
  - holds one abstract method: compareTo ()
  - Any class that implements the Comparable interface must hold this method.
  - You have used this → String
    - obj1.compareTo(obj2) → What will this return (see the Quick reference)

- Example: **List interfaces:**
  - List: look it up (it's in the Quick Reference) what does it do?

  - What class have we used that uses List?

2. A set of classes is used to represent various items that are available for purchase. Items are either taxable or nontaxable. The purchase price of a taxable item is computed from its list price and its tax rate. The purchase price of a nontaxable item is simply its list price. Part of the class hierarchy is shown in the diagram below.



The definitions of the `Item` interface and the `TaxableItem` class are shown below.

```
public interface Item
{
  double purchasePrice();
}


public abstract class TaxableItem implements Item
{
  private double taxRate;


  public abstract double getListPrice();


  public TaxableItem(double rate)
  {  taxRate = rate;  }


  // returns the price of the item including the tax
  public double purchasePrice()
  {  /* to be implemented in part (a) */  }

}
```

(a) Write the `TaxableItem` method `purchasePrice`. The purchase price of a `TaxableItem` is its list price plus the tax on the item. The tax is computed by multiplying the list price by the tax rate. For example, if the tax rate is 0.10 (representing 10%), the purchase price of an item with a list price of $6.50 would be $7.15.

Complete method `purchasePrice` below.

```
// returns the price of the item including the tax
public double purchasePrice()
```

(b) Create the `Vehicle` class, which extends the `TaxableItem` class. A vehicle has two parts to its list price: a dealer cost and dealer markup. The list price of a vehicle is the sum of the dealer cost and the dealer markup.

For example, if a vehicle has a dealer cost of $20,000.00, a dealer markup of $2,500.00, and a tax rate of 0.10, then the list price of the vehicle would be $22,500.00 and the purchase price (including tax) would be $24,750.00. If the dealer markup were changed to $1,000.00, then the list price of the vehicle would be $21,000.00 and the purchase price would be $23,100.00.

Your class should have a constructor that takes dealer cost, the dealer markup, and the tax rate as parameters. Provide any private instance variables needed and implement all necessary methods. Also provide a public method `changeMarkup`, which changes the dealer markup to the value of its parameter.

8. No problemo boss!

- When Java encounters an error-*throws* an error.

Types of exceptions: What do each of the following do?

- ArithmeticException:

- NullPointerExcetion

- ArrayIndexOutOfBounds

2-D Arrays: Drawing inside the lines.

Remember 2D Arrays are **row-major**. That means we look at the rows first, then the columns.

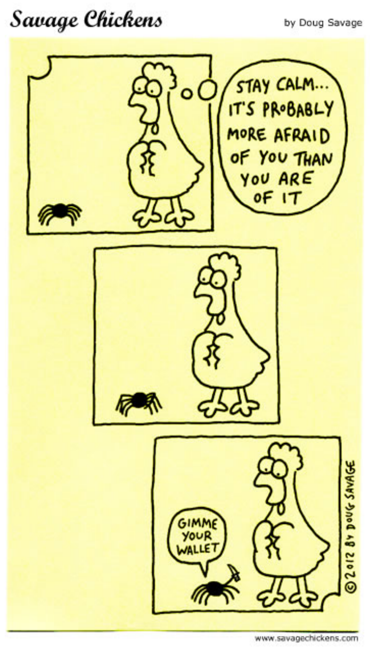
Use the 2D array:

int test [] [] → No, I'm not telling you how big the array is!

Print the sum of only the even values stored in **test**.

9. Grid World – Quit Bugging me!

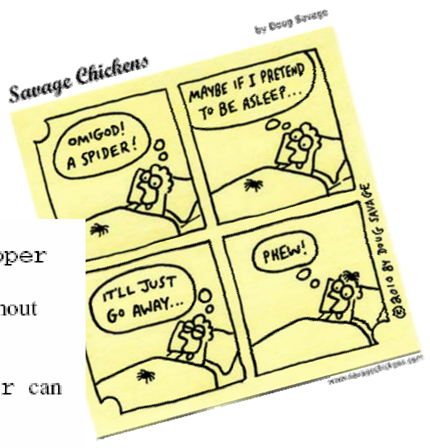
Grid world is 3/16 of the test. You WILL have one Free Response question on it, AND several multiple choice questions.



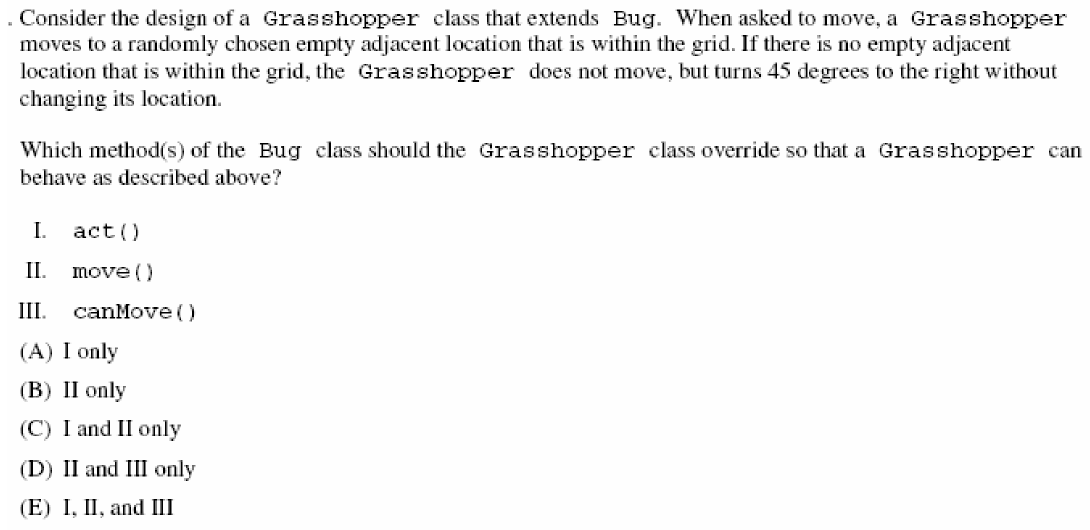
What do the following GridWorld methods do:

- move()
- moveTO()
- act()
- getGrid()
- setDirection(int newDirection)
- getActors() ** Hint → check page E2 of the Quick Reference Guide

Look at page G1 of your Quick Reference Guide → What does this page do?



(From the AP Practice exam)


. Consider the design of a `Grasshopper` class that extends `Bug`. When asked to move, a `Grasshopper` moves to a randomly chosen empty adjacent location that is within the grid. If there is no empty adjacent location that is within the grid, the `Grasshopper` does not move, but turns 45 degrees to the right without changing its location.

Which method(s) of the `Bug` class should the `Grasshopper` class override so that a `Grasshopper` can behave as described above?

I. `act()`
II. `move()`
III. `canMove()`

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

Savage Chickens
by Doug Savage
YOU SHOULDN'T BE AFRAID OF SPIDERS. THEY EAT BUGS!
WHAT SORT OF LOGIC IS THAT? I'VE GOT A COUSIN WHO EATS BUGS AND EVERYBODY'S AFRAID OF HIM
©2012 BY DOUG SAVAGE

Now try a free response question:

This question involves reasoning about the code from the GridWorld case study. A copy of the code is provided as part of this exam.

A `Grub` is a `Critter` that burrows from one location to another. A `Grub` knows how far away it can burrow and randomly chooses the direction in which to burrow. It burrows down from its current location and burrows up at some target location. If the target location is empty or contains a `Flower`, the `Grub` moves to this location. If the target location contains any other type of object, the `Grub` gets stuck underground and dies.

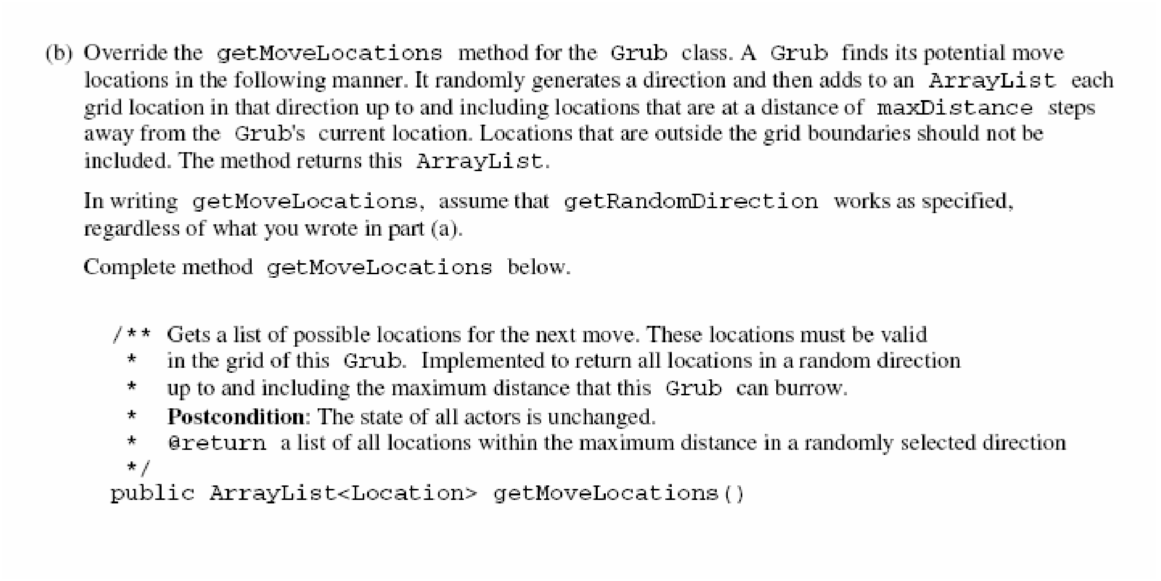
You will implement three of the methods in the following `Grub` class.

```
public class Grub extends Critter
{
  private int maxDistance;

  public Grub(int distance)
  {  maxDistance = distance;   }

  /** @return one of the eight direction constants from the Location class
   */
  public int getRandomDirection()
  {   /* to be implemented in part (a) */   }


  /** Gets a list of possible locations for the next move. These locations must be valid
   *  in the grid of this Grub. Implemented to return all locations in a random direction
   *  up to and including the maximum distance that this Grub can burrow.
   *  Postcondition: The state of all actors is unchanged.
   *  @return a list of all locations within the maximum distance in a randomly selected direction
   */
  public ArrayList<Location> getMoveLocations()
  {   /* to be implemented in part (b) */   }


  /** Selects the location for the next move.
   *  Postcondition: (1) The returned location is an element of locs, this critter's current location,
   *  or null. (2) The state of all actors is unchanged.
   *  @param locs the possible locations for the next move
   *  @return the location that was selected for the next move, or null to indicate
   *              that this Grub should be removed from the grid.
   */
  public Location selectMoveLocation(ArrayList<Location> locs)
  {   /* to be implemented in part (c) */   }

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

}
```

(a) Write the `Grub` method `getRandomDirection` that will randomly return one of the eight direction constants from the `Location` class.

Complete method `getRandomDirection` below.

```
  /** @return one of the eight direction constants from the Location class
   */
  public int getRandomDirection()
```



(b) Override the `getMoveLocations` method for the `Grub` class. A `Grub` finds its potential move locations in the following manner. It randomly generates a direction and then adds to an `ArrayList` each grid location in that direction up to and including locations that are at a distance of `maxDistance` steps away from the `Grub's` current location. Locations that are outside the grid boundaries should not be included. The method returns this `ArrayList`.

In writing `getMoveLocations`, assume that `getRandomDirection` works as specified, regardless of what you wrote in part (a).

Complete method `getMoveLocations` below.

```
/** Gets a list of possible locations for the next move. These locations must be valid
 *  in the grid of this Grub.  Implemented to return all locations in a random direction
 *  up to and including the maximum distance that this Grub can burrow.
 *  Postcondition: The state of all actors is unchanged.
 *  @return a list of all locations within the maximum distance in a randomly selected direction
 */
public ArrayList<Location> getMoveLocations()
```

10. And last but not least….covering all the bases

http://www.learn-programming.za.net/articles_decbinhexoct.html

Check out the above website

You should be able to change :

decimal	base ______
binary	base ______
octal	base ______
hexadecimal	base ______

Try writing the following in:			
decimal	binary	octal	hexadecimal
37			
		230	
	1011		
			33ff00