Class is just a way of organizing code into a representation of things

Class = factory of objects

constructor is the first thing it does to create an instance of this object.

Main class public static void is the thing to start.

Fields are the data types in a class and methods are a function in a class

```
Player x = new Player(5,"Taylor","password");
1. Sees the new keyword, calls the Player constructor
2. Create a new object ({})
3. Assign fields -> {health:undefined, username: undefined, password:
undefined}
4. Run the constructor on the new object
```

Polymorphism

Void - returning nothing

Sub Class - It's a more specific thing of that class

```
class Dragon extends Enemy {
 String dragonPower;
 public Dragon(String xdragonPower){
   dragonPower = xdragonPower;
   super(10,20);
 }

 public void dragonRoar(){
   System.out.println("Roar!")
 }
}

class ant extends Enemy {
 String color;
   public ant(String Ycolor){
     color = Ycolor;
     super(.2,.1);
 }
```

```
 public void specialAnt(){
   if(color == "White"){
     System.out.println("Nothing, it's still a ant")
   }else{
     System.out,println("No ants have anything special and this one isn't
even a cool color");
   }
 }
```

Extends

Inheritance -

```
public void feelingGood(){
   super.feelingGood();
   System.out.println("..But im an ant so Im never feeling that good...")
 }
```

Super = stand in for class

```
private static class Philosopher {

        private String name;

        private String favoriteSubject;

        public Philosopher(String n, String f) {

                name = n;

                favoriteSubject = f;

        }

        public String getName() {

                return name;

        }
```

```
	public String getFavoriteSubject() {
		return favoriteSubject;
	}
	public void speak() {
		System.out.println("Hello, World!  My name is "+name + ". 
My favorite subject is "+favoriteSubject);
	}
}
private static class Nominalist extends Philosopher {
	boolean franciscan;
	public Nominalist(String n,boolean frank) {
		super(n,"logic");
		franciscan = frank;
	}
	public void speak() {
		super.speak();
		if(franciscan) {
			System.out.println("I am a Franciscan");
		} else {
			System.out.println("I am not a Franciscan");
		}
```

```
        }
        public String whoMightHaveTaughtMe() {
                if(franciscan) {
                        return "Perhaps William of Ockham?....";
                } else {
                        return "Perhaps it was Durandus of St. Pourçain —
scandalous, a Dominican nominalist!";
                }
        }
}
public static void main(String[] args) {
    Philosopher[] phils = {
                new Philosopher("Petrus","Ethics"),
                new Nominalist("Minimus Maximus",false),
                new Nominalist("Theodoric",true)};
    for(int i = 0; i < phils.length; i++) {
          phils[i].speak();
    }
}
```

1.

What is the output for the code above?

Output -

creates an array of philosphers called phils

"Hello, World! My name is Petrus. My favorite subject is Ethics"

"Hello, World! My name is Minimus Maximus. My favorite subject is Logic"

"I am not a Franciscan"

"Hello, World! My name is Theodoric. My favorite subject is logic"

"I am a Franciscan"

```
class Animal {
 String name;
 String vocalization;

 public Animal(xname,xvocalization){
   name = xname;
   vocalization = xvocalization;
 }
```

6 7 8 on
http://compscimadison.weebly.com/uploads/5/8/7/4/58741529/ap-computer-science-a-2014-practice-exam.pdf

6. E
7. E
8. A

```
class Animal{
 int numLegs;
```

```
 Animal(int xnumLegs){
   numLegs = xnumLegs;
 }

 Animal(){
   numLegs = 0;
 }




}
```

Recursion

```
public static int sum(int k) {
   if (k > 0) {
     return k + sum(k - 1);
   } else {
     return 0;
   }
 }
```

1. Has to call itself inside it self
2. Base case - where to stop
3. Worth practicing one

5+ 4+3+2+1+0

```
public static int sum(int k) {
   if (k > 0) {
     return k + sum(k - 1);
   } else {
     return 0;
   }
 }

 //return a string that has the appropriate number of numStars
 // 2 - > **
```

```
 /// k - > ***
 public static String numStars(int k){
   if(k > 0){
     return "*" + numStars(k - 1);
   } else {
     return " ";
   }
 }
```

Homework

Write a factorial recursive function

```
public int[] merge_sort(int[] numbers){
   // sorted smalled to largest
   if(numbers.length == 0){
     return numbers
   }

   if(numbers.length === 1){
     return numbers;
   }

   return [join (merg_sort(merge ) merge_sort( second half of numbers)


 }

public static int sum(int k) {
   if (k > 0) {
     return k + sum(k - 1);
   } else {
     return 0;
   }
 }

 //return a string that has the appropriate number of numStars
 // 2 - > **
 /// k - > ***
```

```
 public static String numStars(int k){
   if(k > 0){
     return "*" + numStars(k - 1);
   } else {
     return " ";
   }
 }
```

```
public static int factorial(int K){
   if(K == 0){
     return 0;
   }
   if(K == 1){
     return 1;
   }else{
   return factorial(K) * factorial(K - 1);
 }
 }
```

```
 if(str == "hi"){
   return 1;
 }else{
   return 1 + countHi(str.remove(2));
 }
 }
 }
}
```

```
  int[] myArray;
```

Declaring is all done first above the constructor or main function it just tells the computer to store some data in a spot.
Initializing is just putting some data in the memory spot of the declaration.

```
int[] myArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };
```

```
 static String myString;
int[] myOtherArray = new int[20];

Or instialization it in the constructor.


for(int i = 0; i < myOtherArray.length; i++){
     System.out.println(myOtherArray[i]);
   }
//If you don't instilizae it

class Main {

  int[] myArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };
 static int[] myOtherArray = new int[20];
 static String myString;


 public static void main(String[] args) {
   System.out.println("Hello world!");

   for(int i = 0; i < myOtherArray.length; i++){
     myOtherArray[i] = i + 1;
     System.out.println(myOtherArray[i]);
   }

   // for(int i = 0; i; i++)
   //
 //
//

 }
}



 ArrayList<String> myOtherOtherArray = new ArrayList<String>();

/* List of Arraylist methods
 add -> adds an item to the end of the ArrayList
```

```
 get -> gets an item from the ArrayList, by index
 set -> set a value by index (example myArrayList.set(0,'Im the first
item')  )
 remove -> removes an item based on index
 clear - > empties the list
 size -> gives you the current length of the list

 */
```

Static means that all version of this class will share this field.

Private means only can be interacted with inside the class
You can make a method that can change it later on.

Public means anyone can use it

Static means every instance will share a spot in the memory.

```
import java.util.ArrayList;

class Main {

 /* List of Arraylist methods
 add -> adds an item to the end of the ArrayList
 get -> gets an item from the ArrayList, by index
 set -> set a value by index (example myArrayList.set(0,'Im the first
item')  )
 remove -> removes an item based on index
 clear - > empties the list
 size -> gives you the current length of the list

 */

 static ArrayList<Integer> myOtherOtherArray = new ArrayList<Integer>();
 int[] myArray = new int[]{ 1,2,3,4,5,6,7,8,9,10 };
 static int[] myOtherArray = new int[20];
 static String myString;


 public static void main(String[] args) {
   System.out.println("Hello world!");
```

```
   for(int i = 0; i < myOtherArray.length; i++){
     myOtherOtherArray.add(myOtherArray[i]);
     System.out.println(myOtherOtherArray.get(i));
   }

   // for(int i = 0; i; i++)
   //
 //
//

 }
}


  for(int i = 0; i < myOtherOtherArray.size(); i++){
     System.out.println(myOtherOtherArray.get(i));
   }



import java.util.ArrayList;

class Dog{
public String name;
public ArrayList<String> meals = new ArrayList<meals>();

Dog(String yname, ArrayList<String> ymeals){
 ArrayList<String> meals = ArrayList<String> ymeals;
 name = yname;
}

public int howManyMealsToday(){
 System.out.println(meals.size());
 return meals.size();
}

public void addMeal(String nameOfMeal){
meals.add(nameOfMeal);
}
```

```
public void clearMeals(){
meals.clear();
}

}
```

```
//class Dog should
// track the number of meals each instance has eaten today
// have methods for adding a meal, returning the number of meals, and
clearing the meals at the end of the day

class Dog{
 String name;

 Dog(String yname){
   name = yname;
 }

 public void printName(){
   System.out.println(this.name);
 }



}
```

This is just the instance of the object

```
Dog(String name){
   this.name = name;
 }

Reference another classes function

printArea(){
   location.play(this);
```

```
 }
```

Big

```
 //returns int[] example, sorted
  // basevalue = smallestnumber = the first in the array aka 0 index
   //for loop runs through the array
   //in each loop it will compare the i index to the smallest baseline

   //[5,1,3]

   //Big
   // O(1)
   //O(n) ->
   //O(n^2) -> selection sort
   // O (log n)

   //
   //Selection Sort:
   //loop through the array
   // at each position, save the number in that position to a variable
   //find the smallest number in the rest of the array
   //put the current element in its place



class SortingPractice{
 int[] example = {21,2,30,220,150,60,7771,88,90,998};

 sortSimple(){
   //returns int[] example, sorted
  // basevalue = smallestnumber = the first in the array aka 0 index
   //for loop runs through the array
   //in each loop it will compare the i index to the smallest baseline

   //[5,1,3]

   //Big
```

```
  // O(1)
  //O(n) ->
  //O(n^2) -> selection sort
  // O (log n)

  [5,1,3,4]

  //selection sort
  // [1,5,3,4]
  // [1,3,5,4]

  //insertion sort
  //[1,2,3,5,4]
  //


//sort Merge
//duplicate something to merge back together
// [1,5,3,6,-1,7,0,-5]
// [1,5,3,6] [-1,7,0,-5]
// [1] [2,3] [3,6] [-1, 7] [0,-5]
// [1] [5] [3] [6] [-1] [7] [0] [-5]
//[1,5] [3,6] [-1,7] [0,5]
// [1,3,5,6] [-1,0,5,7]




  //function merge(array)
  //return merge(array.())  merge(array.()


//


  //simplest
  //Selection Sort:
```

```
    //loop through the array
    // at each position, save the number in that position to a variable
    //find the smallest number in the rest of the array
    //put the current element in its place

    //Insertion Sort
    //loop through each element in the array
    //when

    //Merge sort



  }



}
```

https://www.hackerearth.com/practice/algorithms/sorting/merge-sort/tutorial/
https://www.google.com/search?q=selection+sort&oq=selection&aqs=chrome.0.0i67i433j46i433j0i67i131i433j0i67j0i131i433l2j0i433j69i60.1291j0j4&sourceid=chrome&ie=UTF-8
https://www.geeksforgeeks.org/selection-sort/
https://www.geeksforgeeks.org/insertion-sort/

https://www.hackerearth.com/practice/algorithms/sorting/merge-sort/tutorial/
https://www.google.com/search?q=selection+sort&oq=selection&aqs=chrome.0.0i67i433j46i433j0i67i131i433j0i67j0i131i433l2j0i433j69i60.1291j0j4&sourceid=chrome&ie=UTF-8
https://www.geeksforgeeks.org/selection-sort/
https://www.geeksforgeeks.org/insertion-sort/

```
int[][] test  = {{2,3,3},{1,2,3},{1,2,1}};
      for(int i = 0; i < test.length; i++){
        for(int k = 0; k < test[i].length; k++){
          System.out.println(test[i][k]);
```

```
        }
    }

//if you know initial values, use curly braces to define the array, and dont use the new keyword
    int[][] arr = {{1, 2}, {3, 4}};

    //if you dont know initial values, dont use curly braces to define the array, and use the new keyword, like this - >
    int[][] arr = new int[10][20];
```

```
//programing: there are different parts of speech
 // data types -  different types of data you can refference with human words
 //key words used to acess specials things inside a coding language
 // operators are math related/ logic based

Two main types of data types


Mutable vs immutable

Substring creates a substring from the first to the second value and includes the second value

https://www.w3schools.com/java/java_data_types.asp

Two statements inside loop functions
Break means end loop

Static

List of scopes:
private
```

Public
Protected
default

Encapsulation:
Style of coding called object oriented programming, where most things are objects in classes. Everything is done between classes. Java is well set up for object oriented programming, another popular one is you write in functions.

Inheritance:
Class Inherit is about passing things about a class to another smaller class
Polymorphism:
Core concept: the idea that objects of different types can be accessed with standard styles

For example, anytime different pieces of code can satisfy, having a subclass is a form of polymorphism.
Write code that calls a function called add - given arguments

Scaliable code

Abstraction:
Handel compexility by hiding unnecessary data from the user
In object oriented programming, I make a math class, only pass the information that the user needs to see, they don't need to see the backend.

Encapsulation:
Bunch all the access together and only pass the necessary information to other classes. Getter and setters to pass information between classes instead of making them all public

Methods in array list (memorize signatures)
How to write subclasses/inheritance in general
Tree traversals?
Classes: extends vs implements

```
class Main {
 public static void main(String[] args) {
   System.out.println("Hello world!");
   Animal george = new Dog(50,"george","German shepard");
   Animal bill = new Dog(50,"bill","German shepard");
   Animal apollo = new Dog(50,"apollo","German shepard");
   Animal princess = new bengalCat(2000, "princess", 100, 15);
   AnimalPen theGermanShepards = new AnimalPen();
   theGermanShepards.addAnimal(george);
   theGermanShepards.addAnimal(bill);
   theGermanShepards.addAnimal(apollo);
   theGermanShepards.addAnimal(princess);
   theGermanShepards.sayAllNames();

   Animal Ben = new bengalCat(2000, "princess", 100, 15);
 }
}


class Animal {
 int health;
 String name;

 Animal(int yhealth, String yname){
   health = yhealth;
   name = yname;
 }

 Animal(String yname){
   health = 50;
   name = yname;
 }

 void sayName(){
   System.out.println("My name is " + name);
 }

 }
```

```
import java.util.ArrayList;
class AnimalPen {
 ArrayList<Animal> animalsInPen;

 AnimalPen(){
   animalsInPen = new ArrayList<Animal>();
 }

 void addAnimal(Animal animal){
   animalsInPen.add(animal);
 }

 void sayAllNames(){
   for(int i = 0; i < animalsInPen.size(); i++){
     animalsInPen.get(i).sayName();
   }
 }


}
class bengalCat extends Cat{
 int spotsCount;

 bengalCat(int health, String name, int cutenessScale,int yspotsCount){
   super(health,name,cutenessScale);
   spotsCount = yspotsCount;
 }

 void findSpots(){
   System.out.println(spotsCount);
 }
}
class Cat extends Animal{
 int cutnessScale;

 Cat(int health, String name,int ycutnessScale){
   super(health, name);
   cutnessScale = ycutnessScale;
 }
```

```
 void findCuteness(){
   System.out.println(cutnessScale);
 }
}
class Dog extends Animal{
 String breed;

 Dog(int health, String name, String ybreed){
   super(health, name);
   breed = ybreed;
 }

 void findBreed(){
   System.out.print("breed =" + breed + "and health =" + health);
 }

}
```

Classes are the factories

Interfaces

A interface is a promise, a contract that a class will do certain things, a blueprint for any object

If you are creating a program and you need to inte

```
interface canStateNameAndNumber {
    // public, static and final
   final String name;
   final int number;

    // public and abstract
   int returnNumber();

   String returnName();

}

class OurCar implements canStateNameAndNumber {
```

```
 String name;
 int number;

 int returnNumber(){
   return number + 50;
 }

 String returnName(){
   return name;
 }
}
```

```
// Interface
interface Animal {
 public void animalSound(); // interface method (does not have a body)
 public void sleep(); // interface method (does not have a body)
}

// Pig "implements" the Animal interface
class Pig implements Animal {
 public void animalSound() {
   // The body of animalSound() is provided here
   System.out.println("The pig says: wee wee");
 }
 public void sleep() {
   // The body of sleep() is provided here
   System.out.println("Zzz");
 }
}

class Main {
 public static void main(String[] args) {
   Pig myPig = new Pig();  // Create a Pig object
   myPig.animalSound();
   myPig.sleep();
```

```
  }
}
```

The other things that interfaces can do is connected to abstracts - can rely on different things to do something but

Abstracts is when you only need the input and output;

Abstracts is all about

Java provides two feature for abstractions:
Interfaces
and
Abstract class
A class with a built in interface, but the big difference is that they can have interfaces and constructors/full methods

All interfaces need to be static

- **int size() returns the number of elements in the list**
- **boolean add(E obj) appends obj to the end of the list and returns true**
- **E remove(int index) removes the item at the index and shifts remaining items to the left (to a lower index)**
- **void add(int index, E obj) moves any current objects at index or beyond to the right (to a higher index) and inserts obj at the index**
- **E get(int index) returns the item in the list at the index**
- **E set(int index, E obj) replaces the item at index with obj**

https://www.w3schools.com/java/java_ref_string.asp

charAt()
compareTo()
contains()
concat()
contentEquals()
equals()
endsWith()
copyValueOf()
format()
getChars()
indexOf()
length()
isEmpty()
matches()
replace()
replaceAll()
replaceFirst()
toString()
trim()

https://apcentral.collegeboard.org/courses/ap-computer-science-a/exam/past-exam-questions

Do all theses:
https://secure-media.collegeboard.org/digitalServices/pdf/ap/ap16_frq_computer_science_a.pdf

Trees are structures that organize data in a tree-like fashion, with nodes connected to other nodes

```
Class Node {
        Int node_value;
        Public Node childA;
        Public Node childB;

        Node(value){
                Node_value = value;
```

```
        }

        Public Void setChild(Node newChild){
                if(childA == null){
                        childA = newChild;
                } else {
                        childB = newChild
                }
        }
}

Class Tree {
        parentNode Node;

        addNode(){
                Traverse down tree, add node at the bbottom
        }
}


getProblem() → “firstInteger Times initialSecondInteger”

class MultPractice implements StudyPractice {
int firstInteger;
int initialSecondInteger;

        MultPratice(Int yfirstInteger, int yinitialSecondInteger){
                firstInteger = yfirstInteger;
                initialSecondInteger = yinitialSecondInteger;
        }

        String getProblem(){
                return  “firstInteger Times initialSecondInteger”;
        }
        void nextProblem(){
                initialSecondInteger++
        }

}



Public void replaceNthOccurrence(String str, int n, String repl){
```

}

size() returns the number of elements in the list

- **int size() returns the number of elements in the list**
- **boolean add(E obj) appends obj to the end of the list and returns true**
- **E remove(int index) removes the item at the index and shifts remaining items to the left (to a lower index)**
- **void add(int index, E obj) moves any current objects at index or beyond to the right (to a higher index) and inserts obj at the index**
- **E get(int index) returns the item in the list at the index**
- **E set(int index, E obj) replaces the item at index with obj**

Answer:

```
class Main {
 public static void main(String[] args) {
   System.out.println("Hello world!");
   MultPratice test = new MultPratice(1, 2);
   System.out.println(test.getProblem());
   test.nextProblem();
   System.out.println(test.getProblem());
   test.nextProblem();
   test.nextProblem();
   System.out.println(test.getProblem());
 }
}
```

```
class MultPratice  implements StudyPractice  {
 int firstInteger;
```

```
int initialSecondInteger;

 MultPratice(int yfirstInteger, int yinitialSecondInteger){
   firstInteger = yfirstInteger;
   initialSecondInteger = yinitialSecondInteger;
 }

 String getProblem(){
   Integer aFirstInteger = firstInteger;
   Integer ainitialSecondInteger = initialSecondInteger;
   return aFirstInteger.toString() + " Times " + ainitialSecondInteger.toString();
 }

 void nextProblem(){
   initialSecondInteger += 1;
 }
}
```

Class RandomStringChooser{

Private ArrayList <String> array = new ArrayList<String>();

RandomStringChooser(Private[] xarray;){

For(int a = 0; a > xarray.length; a++){

array.add(xarray(a));

}

String getNext(){

If(0 == array.size()){

Return “NONE”

) else {

Int x = Int(array.size() * math.random()) //function on class math that simplifies to integers ;

array.remove(x);

Return array.get(x);

```
                    }
        }

}


Public RandomLetterChooser(String str)(
        Super(str.getChars());
)
        For(Int y = 0; y > Array.size(); y++){
        Int u = Int(Array.size() * math.random);
        Int s = Int(Array.size() * math.random);
        Array.replace(Array.get(u), s);
        }

 )

)


Public LogMessage(String Message){
        String[] splitted = Message.split(“:”); machineId = splitted(0);
description = splitted(1);


Public boolean containsWord(String keyword){

                If(description.Matches(“keyword(.*)”) |
        description.startsWith(“keyword”){
                        if(description.Matches(“(.*)keyword”) |
        description.endsWith(“keyword”){
                                Return True;
                        } else {
```

```
                Return False;
            }
        }

}


Public List<LogMessage> removeMessage(String keyword){
        List<logMessage> ends;
        ArrayList<logMessage> arraylist = new ArrayList<logMessage>();
        For(int c = 0; c > messageList.length(); c++{
                arraylist.add(messageList(c));
        }
        For(int z = 0; z > arraylist.size(); z++{
                if(arraylist(z).containsWord(keyword) == true){
                        ends.add(arraylist(z);
                }
        }
        Return ends;
}

Private boolean toBeLabeled(int r, int c, boolean[][] blackSquares){
        if(boolean[r][c] blackSquares == true){
                Return false;
        }
        if(boolean[r + 1][c] blackSquares == true){
                Return false;
        }else {
                if(boolean[r][c+1] blackSquares == true){
                        Return false;
                }else{
                        Return true;
                }
        }
}
```

```
Public Crossword(boolean[][] blacksquares){
        For(int x = 0; x > blackSquares.length; x++{
                For(int c = 0; c > blackSquares(x); c++{
                        Private Square[x][c] puzzle = ;


}


Public boolean simulate()
{
int frogDistance = 0;
        for(int x = 0; maxHops > x; x++){
                frogDistance += hopDistance();
                        if(goalDistance < frogDistance){
                                return true;
                } else {
                        if( 0 > frogDistance){
                                return false;
                        }
                }
        }
return false;
}

Public double runSimulations(int num){
Int totalWins = 0;
        for(int x = 0; num > x; x++){
                if(simulate() == true){
                        totalWins += 1;
                }
        }
        return  totalWins / num;
}
```

```
Public WordPairList(String[] words){
for(int x = 0; words.length > 0; x++){
        for(int c = 0; words.length > 0; c++){
                if(x < c){
                        WordPair version = new WordPair(words[x],words[c]);
                        WordPairList arraylist = new WordPairList (wordNums);
                        .add()    words[x] words[c]
                }



Public static int numberOfLeapYears(int year1, int year2){
int c = 0;
        For(year1 < year2; year1++){
                if(isLeapYear(year1) == true){
                        c++
                }
        }
}


Public static int dayOfWeek(int month, int day, int year){
        int x = dayOfYear(month, day, year) / 7;
        x = x * 7;
        x = dayOfYear(month, day, year) - x;
        x += firstDayOfYear(year) + 1;
        x -= 1;
        if(x > 6){
                return x - 7;
        }
}

Class StepTracker
{
int minSteps;
ArrayList<int> arraylist = new ArrayList<int>();
        StepTracker(int yminSteps){
```

```
            yminSteps = minSteps;
        }
        Public void addDailySteps(int num){
        arraylist.add(num);
        }
        Public int activeDays(){
        int v = 0;
                for(int c = 0; arraylist.size() > c; c++){
                        if(arraylist.get(c) >= minSteps){
                                v++;
                        }
            }
        return v;
        }
        Public float averageSteps(){
        int v = 0.0;
                for(int x = 0; arraylist.size() > x; x++){
                                v += arraylist.get(x);
            }
        return v / arraylist.size();
        }

}
```

Multiple choice:

0
10
3
0

1
10
4
9

2
4
11
2

3
11
5
8

4 3

```
// .equals for strings --- pass by value
// == pass by reference

public DownloadInfo getDownloadInfo(String title){
        for(int c = 0; c < downloadlist.size(); c++){
                if(downloadlist.get(c).getTitle().equals(title)){
                        return downloadlist.get(c);
                }
        }
        return null;
}

public  int[][]  k

public void updateDownloads (List<Strings> titles){
        for(int c = 0; titles.size() > c; c++){
                if(MusicDownloads.getDownloadInfo(titles.get(c)) == null){
                        DownloadInfo titles.get(c) = new
DownloadInfo(titles.get(c))
```

```
                    downloadList.add(titles.get(c));
                } else {
MusicDownloads.getDownloadInfo(titles.get(c)).incrementTimesDownloaded();
                }
            }
}
```

equal()

!()

SEARCHING:
Sequential search
Grows with the array it's speed is n
Binary search:
Goes to the middle breaks it up in the part where it is, and than do that again until you find the thing
Log n

SORTING:

Insertion sort
5,2,4,1
1,5,2,4
1,2,5,4
N^2

Selection sort
5,2,4,1
2,5,41
2,4,5,1
2,4,1,5
2,1,4,5

1,2,4,5
N^2

https://secure-media.collegeboard.org/apc/ap_frq_computerscience_12.pdf
https://secure-media.collegeboard.org/apc/ap12_computer_science_a_q1.pdf
https://secure-media.collegeboard.org/apc/ap12_computer_science_a_q3.pdf
https://secure-media.collegeboard.org/apc/ap12_computer_science_a_q3.pdf

```
public void addClimb(String peakName, int climbTime){
climbList.add(new ClimbInfo(peakName, climbTime));
}
```

```
public void addClimb(String peakName, int climbTime){
ClimbInfo peakName = new ClimbInfo(peakName, climbTime);
for(int i = 0; i < climbList.size(); i++){
if(peakName.getName().comparesTo(climbList.get(i).getName()) == 0){
        climbList.add(i,peakName);
} else if(peakName.getName().comparesTo(climbList.get(i)) > 0) {
        climbList.add(i, peakName);
}
}
if(peakName.getName().comparesTo(climbList.get(climbList.size() -= 1) <
0){
        climbList.add(peakName);
}
}
```

NO

YES

```
Class RetroBug{
spacialLocation = new Int []{0,0,0};
previousSpacialLocation = new Int[]{0,0,0};

/* precondition angle is between 0 and 360
*/
RetroBug(int angle, int x, int y){
SpacialLocation[0] = angle;
SpacialLocation[1] = x;
SpacialLocation[2] = y;
place(angle, x, y);
}

/* places the RetroBug to the x and y coordinates on the graph
* Rotates the Retro bug according the the degree of angle
*/
Public void place(int angle, int x, int y){

}
/* if the direction of the bug is 0, 90, 180, 270, 360 move it according to
*these directions else find what the angle int is closest to numerically, and
*move it that way
* if the RetroBug is moved up or down, change SpacialLocation[2] by 1 or
*-1, and if it moves * left or right, change SpacialLocation[1] by -1 or 1
* before changing any values save all of SpacialLocation to
*previousSpacialLocation
* if( the direction that is trying to be moved too is blocked than rotate the
bug by -25)
*/
Public void move(){
}
```

```
/* assign the previousSpacialLocations values to the corresponding ones
*/of the spacialLocation

Public void Restore(){
}


public int getTotalBoxes(){
int cookieBoxCount = 0;
        for(int c = 0; orders.size() > c; c++){
        cookieBoxCount += orders.get(c).getNumBoxes();
        }
return cookieBoxCount
}


public int removeVariety(String cookieVar){
int cookieBoxCount = 0;
        for(int c = 0; orders.size() > c; c++){
                if(orders.get(c).getVariety().equals(cookieVar)){
                cookieBoxCount += orders.get(c).getNumBoxes();
                orders.remove(c);
                }
        }
return cookieBoxCount;
}

Class APLine{
int aEquation;
int bEquation;
int cEquation;

public APLine(int a, int b, int c){
         aEquation = a
         bEquation = b
         cEquation = c
```

```
}

public double getSlope(){
        return (double) - aEquation / bEquation;
}

Public boolean isOnLine(int x, int y){
if(x * aEquation + y * bEquation + cEquation == 0){
        return true;
} else
        return false;
}




public boolean isLevelTrailSegment(int start, int end){
int max = markers[start];
int min = markers[start];
        for(int c = start + 1; c < end; c++){
                If(max < markers[c]){
                        max = markers[c];
                }
                if(min  > markers[c]){
                        min = markers[c];
                }
        }
        if(max - min <= 10){
        return true;
        } else {
        return false;
        }
}

public boolean isDifficult(){
int difficultCount = 0;
Int previousMarker = markers[0];
```

```
        for(int c = 1; c < markers.length(); c++){
            if(Math.abs(markers[c] - previousMarker) <= 30){
                difficultCount++;
            }
            if(difficultCount == 3){
                return true;
            }
    }
return false;
}
```

```
public LightBoard(int numRows, int numCols){
        for(int c = 0; numRows > c; c++){
            for(int v = 0; numCols > v; v++){
                if(100 * Math.random() <= 40){
                        lights[c][v] = true;
                } else {
                        lights[c][v] = false;
                }
        }
    }
}

Public boolean evaluateLight(int row, int col){
int numChecker;
if(lights[row][col] == true){
        for(int c = 0; lights[row].length() > c; c++){
            if(lights[row][c] == true){
                numChecker += 1;
            }
    }
```

```
        if(numChecker % 2 == 0){
                return false;
        } else {
                return true;
                }
} else {
        for(int c = 0; lights[row].length() > c; c++){
                if(lights[row][c] == true){
                        numChecker += 1;
        }
        if(numChecker % 3 == 0){
                return true;
        } else {
                return false;
        }
}
```