Guion de Capítulo 4 Clases

Capitulo 4.1 Estructuras de datos

¡Saludos y bienvenidos!

En este capítulo extenso no vamos a explorar el uso de cada una de las estructuras de datos, **sino a introducirte a estas.**

Objetivos:

- **Introducir diversas estructuras de datos**
- **Comentar sobre pom.xml**
- **Comentario sobre representaciones computacionales**

Para empezar, quisiera que sepas que hasta este punto hemos estando trabajando con instanciaciones de clases.

```
int x = new Integer(5);
double y = new Double(3.14);
```

Cada uno de nosotros podemos tener nuestra propia definición de una clase. Formalmente, una clase es un templete de la cual empleamos para crear objetos. Nosotros instanciamos una clase, creando un objeto de la siguiente manera tanto en Python como en Java:

Hoy vamos a explorar diversas clases que son útiles al momento de programar.

Ojo, debido a las funcionalidades principales de estas clases, estas también reciben el nombre de estructuras de datos, por ende uso estos términos de manera intercambiada.

Nosotros vamos a explorar estructuras de datos principalmente en Python. En este capítulo vamos a explorar estructuras sencillas que vienen tanto preinstaladas o a través de librerías importadas.

Tupla (tuples)

Estas estructuras de datos son bastante simplistas.

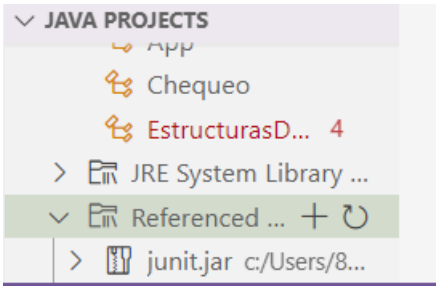
Aparecen de manera predeterminada en Python, pero no en Java.

Las tuplas son inmutables. Almacenan dato de cualquier clase.

Usualmente usado para almacenar datos en pares.

En java, es necesario agregar la dependencia en el archivo pom.xml, el cual es un archivo creado por el entorno Maven que sirve para administrar mi programa Java.

Comentario: podemos ver la dependencia de Junit la cual provee con métodos útiles para escribir archivos de depuración. Si no, también podríamos haberlo descargado e importado manualmente, lo cual es tedioso




```
<dependency>
        <groupId>org.javatuples</groupId>
        <artifactId>javatuples</artifactId>
        <version>1.2</version>
        <scope>compile</scope>
    </dependency>
```

```
Pair <Integer, String> miTupla = new Pair <Integer, String> (12, "Shirley");
```

Métodos comunes

```
.getSize()
.getValue(indice)
```

https://www.javatpoint.com/java-tuple

Arreglos-Listas

Los arreglos-listas proveen de más funcionalidades que los arreglos, por ejemplo no es obligatorio especificar el tamaño previsto del arreglo-lista o lista.

Puedo llenarlo con objetos diversos, como un arreglo de tuplas.

```
ArrayList<Integer> miArregloLista2 = new ArrayList<Integer>(Arrays.asList(1,2,3,4,5));

ArrayList<Pair<Character, Double>> miArregloListaDiverso = new ArrayList<Pair<Character, Double>>();
```

Métodos comunes

```
miArregloLista2.size();

miArregloLista2.add(10);

miArregloLista2.remove(0);

miArregloLista2.contains(19);

miArregloLista2.toString();
```

Pilas y colas (stack and queue)

Nosotros podemos emplear estas estructuras de datos importando la clase *collections.*

Son preferidos cuando tienes definida el tamaño de una lista (imagina los arreglos en Java). Las pilas siguen la política LIFO

```
Queue<Integer> miCola = new ArrayDeque<Integer>();
```

Métodos comunes

```
miCola.poll();

miCola.add(12);
```

Las colas siguen la política FIFO

```
Stack<Double> miPila = new Stack<Double>();
```

Métodos comunes

```
miPila.push(2.78);

miPila.push(12.3);

miPila.pop();

miPila.empty();
```

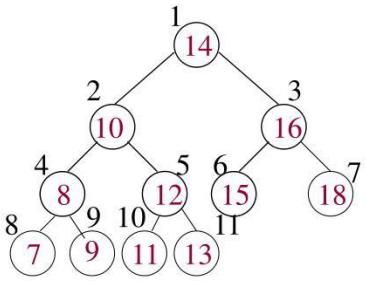
Son útiles cuando requieres trabajar de manera iterada y ordenada, unidireccionalmente con data. Ahorras memoria.

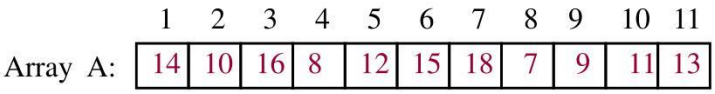
Cabe recalcar que esta pila y montículo en este capítulo son estructuras de datos, no las esquematizaciones conceptuales de cómo la RAM aloca memoria durante la ejecución de un programa

Montículos (heap),

Los montículos son estructuras de datos que ordenan los elementos siguiendo la estructura de un grafo o árbol con la propiedad especial de que el nodo central contiene el valor mínimo del arreglo.

Binary Trees: Array Representation

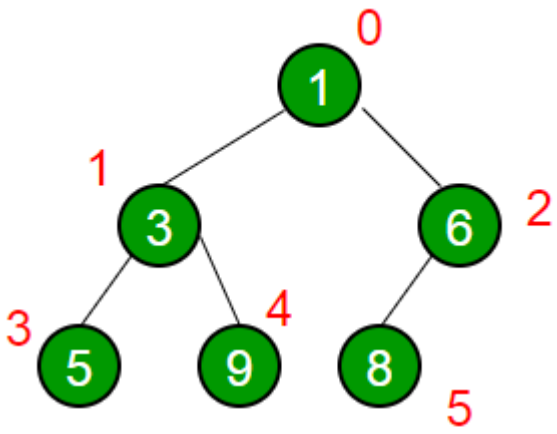


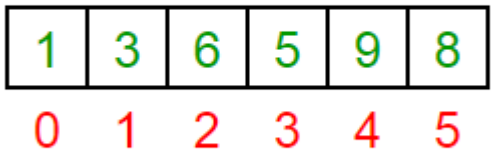


7

https://www.slideserve.com/whitney-golden/trees

Importante: La esquematización es realizada por cuestiones didácticas. No te confundas, esto no es lo que esta almacenada en nuestra computadora.





https://www.geeksforgeeks.org/binary-heap/

Nosotros podemos emplear estas estructuras de datos importando la clase *heapq* en Python.

En java, podemos instanciar la clase PriorityQueue la cual conceptualmente puede hacer lo que un montículo fue diseñado a hacer.

Tiempo esperado de operación> devuelve el mínimo

```
Queue<Integer> miMonticulo = new PriorityQueue<Integer>();
```

Métodos comunes

```
miMonticulo.poll();

miMonticulo.add(12);

int cabeza = miMonticulo.peek();
```

Diccionario

Estas estructuras de datos aparecen de manera predeterminada en Python.

En java, usamos el la clase HashMap.

Los diccionarios son útiles cuando quiero almacenar valores en pares de llave-valor, en la cual yo no uso la llave, sino como un acceso para el valor que desee.

En general, podemos suponer que podemos personalizar el índice.

https://pixels.com/featured/tabla-periodica-de-los-elementos-alejo-miranda.html

```
Map<String, String> tablaPeriodica = new HashMap<String, Integer>();
```

Métodos comunes

```
tablaPeriodica.put("Helio", 2);

tablaPeriodica.put("Carbono", 12);

int miValor = tablaPeriodica.get("Helio");

tablaPeriodica.entrySet();
```

Comentarios

1. No es necesario memorizar
    a. https://docs.oracle.com/en/java/javase/18/docs/api/
2. Podemos también visualizar estructuras de datos
    a. https://www.cs.usfca.edu/%7Egalles/visualization/Algorithms.html

Por ahora, es suficiente de que sepas lo siguiente:

1) existen diversas estructuras de datos.
2) Algunas estructuras de datos son mejores que otras en el sentido de que proveen las funciones necesarias. Pueden ser más sencillas, o eficientes en términos de memoria y tiempo de ejecución.
    a. Rasuradora de Ocam: elige el que es necesario, no el más usado.
3) El estudio más avanzado de ciencias computaciones incluye el análisis de estas estructuras de datos:
    a. ¿Cuáles son más convenientes en qué situación?
    b. ¿Cuáles permiten un tiempo más rápido de ejecución? Análisis asintótico.

Quisiera comentar sobre el chiste sobre la representación de números romanos en número arábico.

Hablemos un poco de filosofía: límites de una representación:

No es que en serio hay circulitos coloridos conectados: todo son arreglos de números

Cormen, T. H., Charles Eric Leiserson, Rivest, R. L., Stein, C., & Al, E. (2009). Introduction to algorithms. MIT Press.

Concluyamos no con un final cerrado, sino con un final abierto

Visión computacional: ¿es o no una matriz de números suficiente?

Pierde alguna esencia de la realidad

Si algo dentro de ti, llamado curiosidad, se ha prendido

Capítulo 3.2 (Matemática Avanzada):

Árboles

```
conda install binarytree -c conda-forge
```

Nosotros podemos emplear estas estructuras de datos importando la clase *binarytree*

Vectores y matrices

Nosotros podemos emplear estas estructuras de datos importando la clase *numpy*

Para grafos

Nosotros podemos emplear estas estructuras de datos importando la clase *networkx*

https://networkx.org/documentation/stable/_downloads/networkx_reference.pdf

Razuradora de Ocam

Capítulo 4.2 Algoritmos

Algoritmo Encontrar Min-Max

Algorithm 5 Maximum-Minimum algorithm

Input: Array a
Output: Maximum/Minimum value on the array

```
1: max ← a[0]
2: min ← a[0]
3: for i ← 0, a.length − 1 do
4:    if (a[i] > max) then
5:       max ← a[i]
6:    end if
7:    if (a[i] < min) then
8:       min ← a[i]
9:    end if
10: end for
11: return max, min
```

https://computinglearner.com/the-5-basic-algorithms-in-programming-for-beginners/

usar métodos predefinidos daña el orgullo de un programador. `Collections.max; Collections.min; Collections.sort`

BubbleSort

Ingreso*: *Arreglo arr

Egreso*: *arreglo ordenado de manera ascendiente

1. $n \leftarrow tamaño(arr)$
2. **for** $i \leftarrow 0, n$ **do**
3. **for** j $\leftarrow 0, n-1$ **do**
4. **if** $arr[j] > arr[j+1]$ **then**
5. $intercambiar(arr[j], arr[j+1])$
6. **end if**
7. **end for**
8. **end for**
9. **return** arr

https://www.cs.usfca.edu/%7Egalles/visualization/ComparisonSort.html

Un algoritmo es universal: aplica para todo lenguaje de programación

Busca un algoritmo que te guste e impleméntalo

InsertionSort, QuickSort, MergeSort

Ingreso*: *Montículo mon

Egreso*: *valor mínimo

1. $min \leftarrow mon.get(0)$
2. **return** min

Capítulo 4.3: Definiendo nuestras propias clases

Objetivos de hoy: Sneak Peak: Hoy vamos a estudiar sobre la definición de nuestras propias clases:

- Campos, constructor y métodos
- Modificadores públicos vs privados
- Métodos sobrescritos

Recapitulemos:

Un objeto es definido como un instante de una clase, y una clase es un templete que especifica las propiedades y funciones de un objeto.

Hay distintas clases predefinidas (estructuras de datos) que podemos emplear a nuestra conveniencia para resolver diversos problemas computacionales y matemáticos.

¿Qué pasa si nos encontramos con un problema que requiere de funciones y propiedades que no hemos visto anteriormente? Por ejemplo, simular un ascensor:

Fuiste encargad@ a simular cómo funciona un ascensor. Este ascensor puede cambiar de pisos solo cuando la electricidad que sobra es mayor que cero. Por cada piso que se desplaza, consume la mitad de energía. Por ejemplo, si se desplaza por 3 pisos, entonces consume 3/2 = 1.5 unidades (Joules) de electricidad. El ascensor comienza teniendo 50 unidades de energía y puede ser recargado cuando desees.

¿Con qué estructura de dato puedo simular todo este comportamiento? Arreglo, diccionario, cadena…

Una clase es definida:

1. empezando con creando un nuevo archivo Java.
2. Especificando campos (fields) que serían las propiedades de nuestro objeto, o sea, sus características (decidir cuáles son privados y públicos)
3. Luego defines un *constructor* de la clase, el cual es un método que especifica cuáles son los valores iniciales de algunos campos
4. Los métodos que especifican cuáles son las funciones del objeto son definidos (decidir cuáles son públicos y privados).

5. Hay alguno que otro método que se puede sobrescribir

Podemos observar todo este proceso en nuestro EDI.

En lo que concierne la decisión de sí o no definir un campo o método como público o privado depende de la lógica del programa. Por ejemplo: no tiene sentido definir tanto la electricidad como el piso actual como campos públicos ya que no puedo cambiarlos cuando yo desee. La misma idea aplica para métodos. El método de consumir energía debe ser privado porque no puedo ejecutarlo de manera arbitraria, solo cuando este cambiando de piso.

Puedes pausar el video y completar los métodos de rellenar energía y chequear energía.

Procedemos a discutir sobre métodos sobrescritos.

Trata de ejecutar

```
System.out.println(miAscensor); //
com.example.Ascensor@1f32e575
```

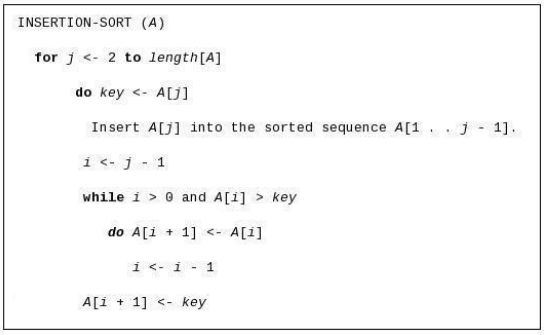
¿Qué ves en la consola? Algo similar a cuando imprimimos arreglos, observamos su ubicación en la memoria/montículo. Esta no es una representación fiel para el usuari@. Es mejor sobrescribir la función asociada a la representación en cadena de un objeto. Tú puedes decidir cómo lo deseas. Yo elegí lo siguiente:

```
@Override
    public String toString() {
        // TODO Auto-generated method stub
        String mensaje = "Actualmente estás en el piso
" + this.pisoActual +
        "y tienes " + this.electricidad + " unidades de
electricidad";
        return mensaje;
    }
```

```
// Campos - fields
    private int pisoActual;           // definido al
momento de instanciacion
    private double electricidad = 50; // predefinido
    //private double CAPACIDAD;


    // Constructor - como es inicializado
    public Ascensor (int piso){
        this.pisoActual = piso;
    }
```

```
    // private double PesoAscensor(){
    //     return this.capacidad * this.electricidad;
    // }

    // public boolean ExcederLimite(double pesoActual){
    //     double costoTotal = PesoAscensor();
    //     if (pesoActual > costoTotal){
    //         System.out.println("El ascensor necesita
recargar!");
    //         return true;
    //     }
    //     return false;
    // }
        private boolean chequearElectricidad(){
        return this.electricidad > 0;
    }
    private void ConsumirElectricidad (int
pisoSiguiente) {
        // no debe ser publico porque no tiene sentido
emplear esta funcion de
        // manera arbitraria. E.g. no estoy cambiando
de piso.
        double electricidadConsumida =
Math.abs(pisoSiguiente - this.pisoActual) / 2.0;
        this.electricidad -= electricidadConsumida;
        System.out.println("Se ha consumido " +
electricidadConsumida + " unidades de electricidad");
    }
    public void CambiarPiso (int pisoSiguiente){
        ConsumirElectricidad (pisoSiguiente);
        this.pisoActual = pisoSiguiente;
        System.out.println("Ascensor ha cambiado al
piso: " + this.pisoActual + " y le sobra " +
        this.electricidad + " unidades de
electricidad");
    }
```

Podemos extender el problema con el fin de aplicar lo que aprendimos en capítulos previos. Imaginamos que el ascensor puede tomar una lista de pisos, solo que de manera desbaratada y debemos primero organizarla, calcular la electricidad consumida e imprimir a qué pisos el ascensor se ha desplazado por. Puedes también tomar entradas del usuario.



```
INSERTION-SORT (A)
  for j <- 2 to length[A]
       do key <- A[j]
         Insert A[j] into the sorted sequence A[1 . . j - 1].
       i <- j - 1
       while i > 0 and A[i] > key
          do A[i + 1] <- A[i]
             i <- i - 1
       A[i + 1] <- key
```

```
InsertionSort(arr[])
      for j = 1 to arr.length
         key = arr[j]
         i = j - 1
         while i > 0 and arr[i] > key
            arr[i+1] = arr[i]
            i = i - 1
         arr[i+1] = key
```

fuente: https://www.freecodecamp.org/news/insertion-sort-what-it-is-and-how-it-works/

- Public vs Private vs Protected vs Extends vs Implements…

	Campos	Métodos/Funciones
Públicos	propiedad cuyo valor que puede ser cambiado de manera arbitraria	un proceso puede ser ejecutado de manera deliberada
Privados	Propiedad cuyo valor no puede ser cambiado cuando uno desea	Un proceso que debe ocurrir “detrás de cámaras”
Finales	Propiedad cuyo valor es una constante inmutable	
Sobrescritas		Métodos predefinidos de Java que TODA clase debe tener y que pueden ser redefinidos
Estático	Campos que pueden ser accedidos aunque un objeto no haya sido instanciado.	Métodos definidos que pueden tomar entradas que no tienen relación alguna con la clase definida. Métodos sin este modificador solo puede ser ejecutados cuando un objeto instanciado de la clase

Capítulo 4.4 Implementando clases abstractas

Objetivos:

- Hoy vamos a introducirnos a la definición de clases abstractas y a resolver problemas a través de estas.

Problema ejemplar:

- Simular el comportamiento de animales de una reserva de Galápagos
  - Todo animal es inicializado con un nombre
  - Una tortuga camina consumiendo energía. Por cada paso que toma, consumen media energía. Solo pueden moverse cuando no tienen hambre, en donde hambre es denotado si ya no tienen energía.
  - Un pez nada, y las condiciones de energía son iguales que la de la tortuga, con la excepción de que la conversión de ´paso a energía es por cada paso, una ave consume 2 unidades de energía (es por eso que desde un punto de vista evolucionario tienen menos peso)
  - Un ave vuela y las condiciones de energía son iguales que la de la tortuga, con la excepción de que la conversión de ´paso a energía es por cada paso, una ave consume 4 unidades de energía (es por eso que desde un punto de vista evolucionario tienen menos peso)
  -

Podemos implementar 3 clases distintas, o extender 3 clases de una clase abstracta.

Los beneficios de implementar una clase abstracta son principales estéticos. No es que escribes “menos código”, sino que tu programa o código aparecerá más limpio y conciso para alguna audiencia.

¿Qué pasa si quiero comparar dos animales y chequear de que son iguales para evitar redundancia?

Tengo que sobreescribir equals

Copiar y pegar de geeks for geeks y decidir cómo comparar. Por ejemplo, podemos en nuestro juicio personal, decidir que dos animales que tienen el mismo nombre son iguales.

Comentarios

Las clases abstractas son útiles cuando estimas que hay distintas clases que comparten similares métodos y propiedades

Si un campo no cambia, entonces usar el modificador final en adición a privado

Sobreescribir el método equals es un proceso mecánico, y al mismo tiempo hay que sobrescribir hashcode. Hashcode debe ser sobreescrito debido a la estructura de dato HashSet que emplea este código único como llave. No puedes inherir métodos de múltiples clases abstractas. Necesitas interfaces.

De nuevo, de manera inadvertida estamos practicando el pensamiento computacional de divide y vencerás.

Por cuestiones de tiempo, hemos omitido Enums, Interfaces, pero de manera breve

- En Interfaces solo pueden haber métodos abstractos y permite herencia múltiples
- Los Enums (enumeradores) permiten la definición de constantes.

Esto es porque te estaría dando la idea errónea si te fuese a sugerir de que aprenderás todo sobre programación a través de estos capítulos. Esto no es verdad. Programación es principal una rama auto-didacta, pero estate segur@ de que hay una comunidad (incluyéndome) que estará disponible a ayudarte cuando lo desees.

Capítulo 4.5: Resolviendo problemas varios

En el presente capítulo pondremos en práctica todo lo que hemos aprendido.

Objetivos:

- Completar ejercicios en donde los temas de 1) estructuras de datos, 2) entendimiento de algoritmos y la 3) definición de clases y 4) clases abstractas son puestas en prácticas.
- Compartir consejos sobre cómo mejorar habilidad de programación
  - Transferir aprendizaje a Python
  - Aprender a través de la comunidad
  - Practicar con Leetcode o HackerRank

Problemas varios

1. Aperitivo: Tomar entradas del usuario para un arreglo de decimales y ordenarlo usando sorteo de burbuja.

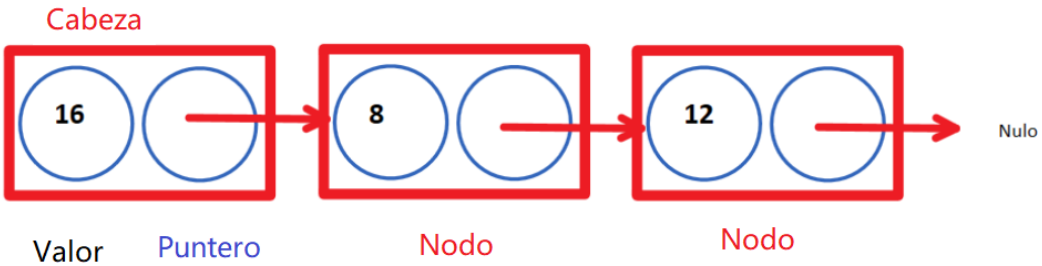
   Ingreso: ***Arreglo arr***

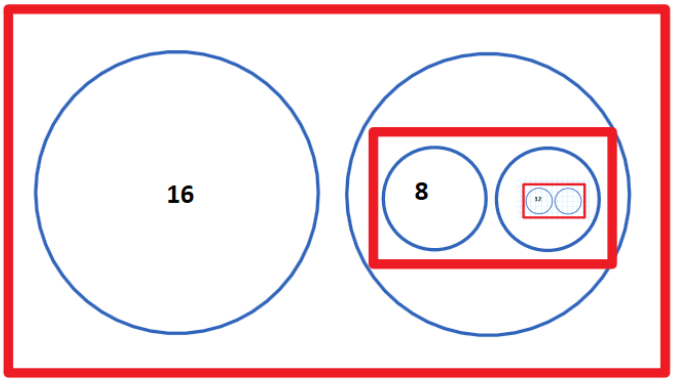
   Egreso: ***arreglo ordenado de manera ascendiente***

   1. $n \leftarrow tamaño(arr)$
   2. **for** $i \leftarrow 0, n$ **do**
   3. **for** j $\leftarrow 0, n-1$ **do**
   4. **if** $arr[j] > arr[j+1]$ **then**
   5. $intercambiar(arr[j], arr[j+1])$
   6. **end if**
   7. **end for**
   8. **end for**
   9. **return** arr

   a.
   b. Pensar sobre cómo mejorar el algoritmo. Sneak Peak de análisis asintótico.
   c. Es crucial que entiendas cómo funciona: https://www.cs.usfca.edu/%7Egalles/visualization/ComparisonSort.html
2. Implementar una lista vincular singular con las funciones respectivas de insertar un nodo, eliminar un nodo, encontrar el tamaño de la lista, intercambiar la posición de nodos, sortear la lista, entre otros métodos no-estáticos que consideres útil. Sobrescribir los métodos toString, equals y hashcode.
   a. Una lista vinculada singular es aquella que tiene nodos, la cual cada uno tiene un valor y un puntero que contiene el siguiente nodo. La lista vinculada

empieza con un nodo cabaza y termina con un puntero hacia nulo.





b. Una lista vinculada es "útil" en el ámbito autodidáctico ya que un programador@ puede poner en práctica varios temas de programación simultáneamente.
c. Una representación en cadena aceptable es:
   0->1->2->3->4->5->6->7->8->9->10->nulo
d. Pista sobre cómo acceder a cierto elemento en una lista vinculada:
   i. Para el primer valor: `cabeza.valor`
   ii. Para el segundo valor:
   `cabeza.punteroSiguiente.valor`
   iii. Para el tercer valor:

```
cabeza.punteroSiguiente.punteroSiguiente.valor
```

   iv. ¿Cómo expresarlo en código?

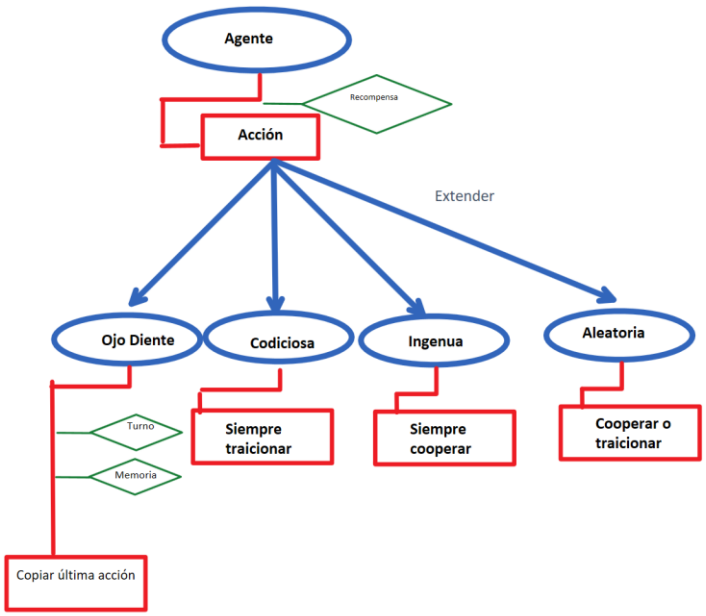
```
while (nodo.punteroSiguiente != null) {
        nodo = nodo.punteroSiguiente;}
```

3. Implementar un juego filosófico llamado dilema del prisionero iterado:
   a. El dilema del prisionero iterado es un juego filosófico/económica la cual plantea el siguiente escenario hipotético: tú y tu colega roban un banco, pero son capturados por la policía.

Recompensas desde *mi* punto de vista (TC > CC > TT > CT)		
	Si mi colega decide:	
Si yo decido:	Cooperar	Traicionar
Cooperar	3	0
Traicionar	5	1

b. Para observar una simulación interactiva, por favor visitar: https://ncase.me/trust . No vamos a recrear la simulación, sino a implementar la lógica básica del juego.

c. Yo propongo la siguiente esquematización del programa, tú critícalo y mejora:



Consejos de programación

1. Transferir lo que aprendiste a Python

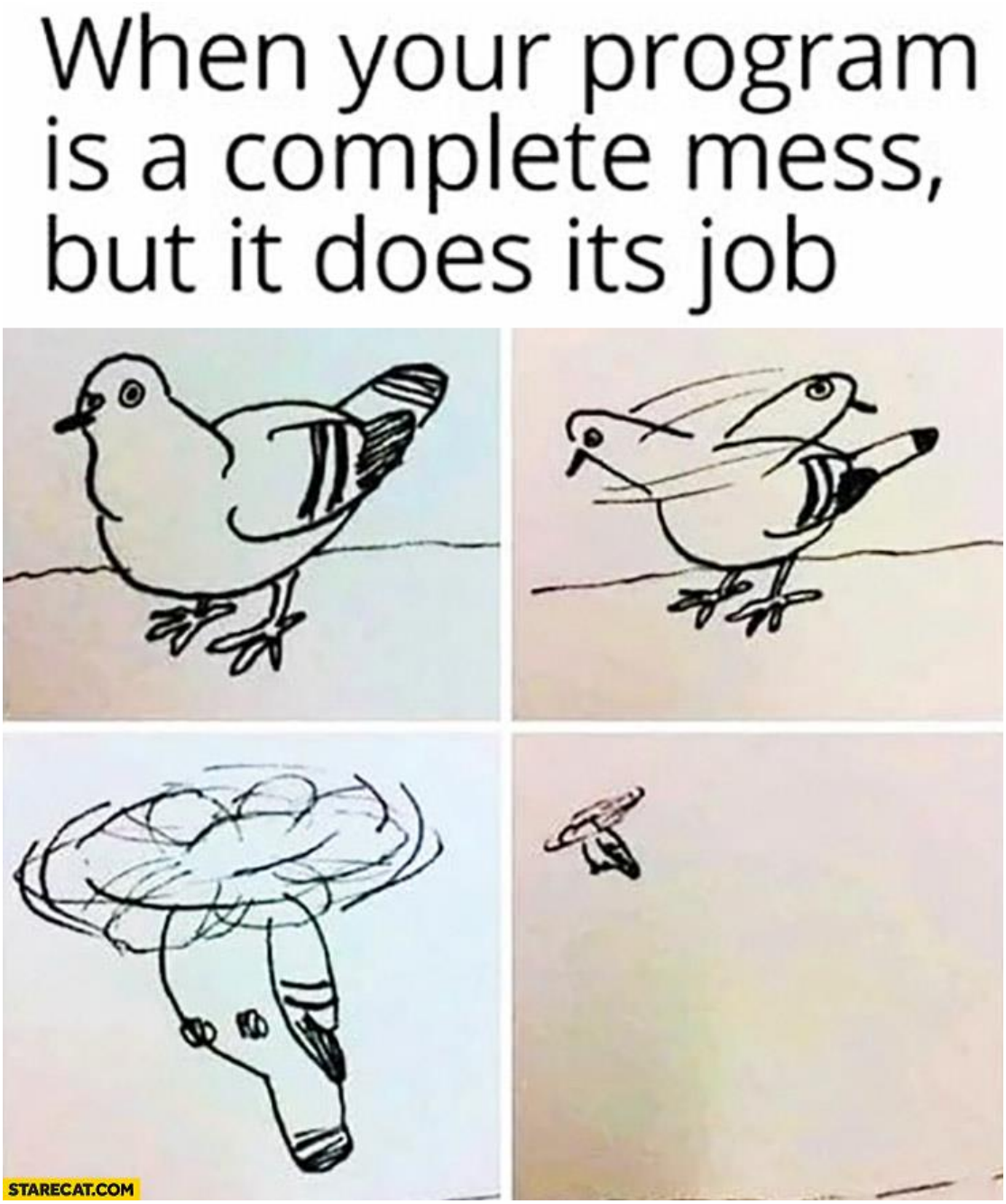
```
class miClase():
    def __init__(self, campo1, campo2):
        self.campo1 = campo1
        self.campo2 = campo2
        self.campo3 = 0
        self.campo4 = “a”;

    def __funcionPrivada(self, ingreso):
        raise NotImplemented

    def funcionPublica (self, ingreso):
        raise NotImplemented
```

```
    def funcionStatica (ingreso):
        raise NotImplemented
```

2. **Comprometerte a ti mismo a practicar frecuentemente a programar resolviendo problemas computacionales ya sea en LeetCode (**https://*leetcode*.com) o **Hackerrank (**https://www.*hackerrank*.com)
    - **a.** 1 ejercicio fácil por día, 5 ejercicios medios al fin de semana, 2 ejercicios difíciles al mes y mejorar
    - **b.** Aprender cómo usar un nuevo método, algoritmo, estrategia de programación o estructura de dato, por ejemplo, stream(), *Hashset*
    - **c.** Monitorear el tiempo utilizado
    - **d.** Entre otros.
3. **Aprender de los errores: ayudar a otros miembros de la comunidad computacional a depurar**
    a. Visitar Github, StackOverflow, o comunidades digitales de programación para familiarizarte con errores comunes y cómo resolverlos
    b. Por ejemplo, hay varios foros y comunidades de programación en donde comparten errores y soluciones:
    https://community.oracle.com/tech/developers/discussion/1261175/using-scanner-class-to-create-arraylist
    https://community.oracle.com/tech/developers/discussion/1262366/java-lang-numberformatexception

Conclusión

- Con este capítulo concluimos el tema de clases, esto no significa que hemos visto todo sobre objetos y clases, pero que ya tienes los conocimientos básicos para indagar por tu cuenta
- Programación es aproximadamente 30% teoría, 70% práctica, así que visita Leetcode o Hackerrank y define un horario de entrenamiento en tu calendario y visualízate en qué te quieres convertir.
- Ayudar a los demás permite el crecimiento personal



Más ideas

Practicar en Python

Problema del Tranvía

Conclusión

You are not supposed to create the higher order function before. You are supposed to use it.

So you need to change your function signature from something like

```
def insert_sort(mylist):
    # -- code --
```

run code snippet

to

```
def insert_sort(mylist, before):
    # -- code --
```

run code snippet

Then you need to replace all occurrences of the `<` operator with uses of your function `before`. (e.g. `x < y` would be come `before(x,y)`)

If you want to test your function, you can use a simple lambda:

```
mylist = [-x for x in range(10000)]
insert_sort(mylist, lambda x, y: x < y)
```

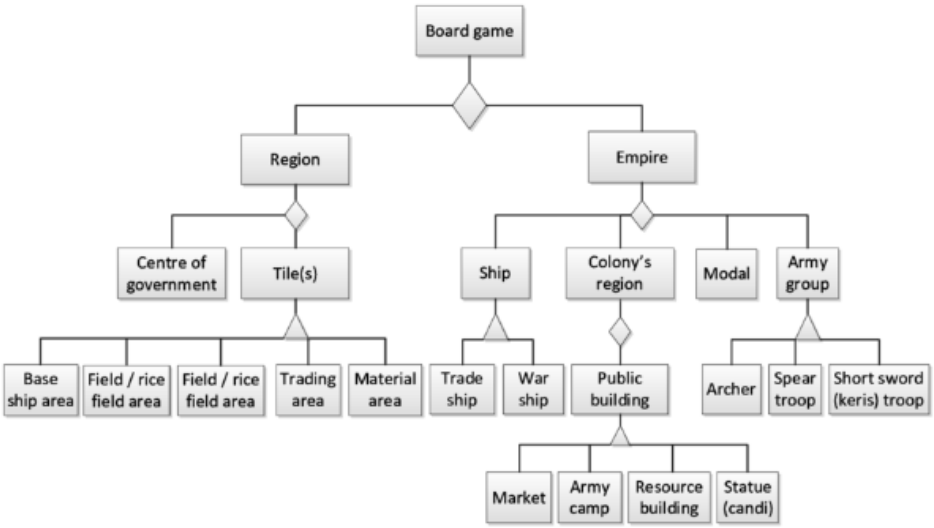
Observe that we could also do the following

```
insert_sort(mylist, lambda x, y: x > y)
```

That might give you an idea why this might be a useful thing to do.

https://www.daniweb.com/programming/software-development/threads/434728/using-a-scanner-in-an-arraylist

Divide and Conquer using Classes



Búsqueda binaria

Modifier	Class	Package	Global
Public	Yes	Yes	Yes
Default	Yes	Yes	No
Private	Yes	No	No

Representación de lo que quiero: recuerdo que neuronas en realidad son vectores, y vectores son listas de números.

Vector: producto punto y magnitud o estructura concatenada

Ascensor:

Árbol binario

Ascensor, tabla de Go, tic-tac-toe

Sort