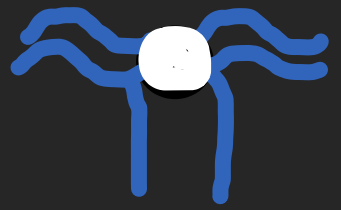
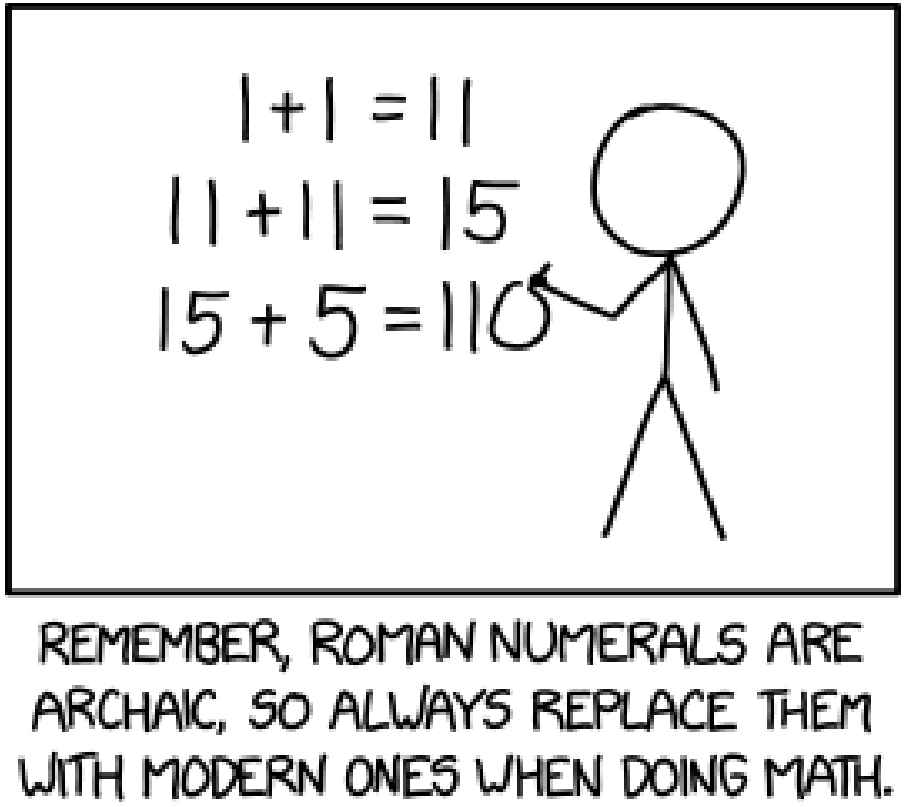
Capítulo IV.I - Clases

Estructuras de datos, dependencias y representaciones

Objetivos:



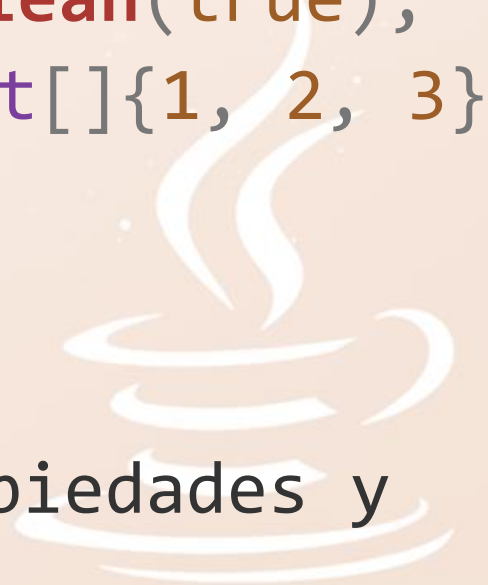
- Introducir diversas estructuras de datos
  - Breve comentario sobre dependencias
- Comentario sobre representaciones computacionales



- *“Recuerda, los números romanos son arcaicos, así que hay que reemplazarlos con números modernos cuando estés haciendo alguno que otro cálculo matemático”*
- Fuente: https://xkcd.com/2637

¿Con qué hemos trabajado hasta ahora?

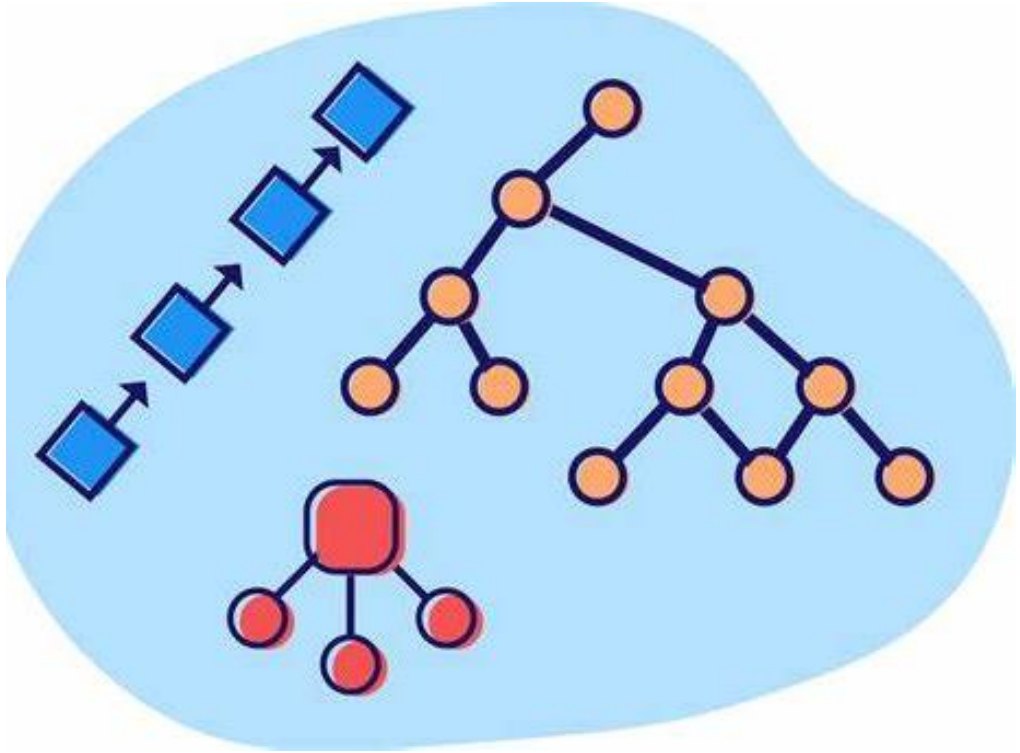
- `int nueve = new Integer(9);`
- `double pi = new Double(3.14);`
- `boolean verdadero = new Boolean(true);`
- `int[] arregloSencillo = new int[]{1, 2, 3};`



- Objetos: Instanciaciones de clases
- Clase: templete que especifica propiedades y funciones del objeto instanciado

Diversas clases predefinidas del paquete java.util

- Estructuras de datos
  - Tuplas (Tuples)
  - ArregloListas (ArrayLists)
  - Pilas y Colas (Stack and Queues)
  - Montículos (Heap)
  - Diccionario (Hashmap/Dictionary)
  - Vectores y Matrices
  - …



https://www.educative.io/blog/data-structures-algorithms

Tuplas (Tuples)

- En Java Maven, es necesario modificar pom.xml
- Estructura inmutable
- Usado usualmente para almacenar datos en pares

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

Métodos comunes

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

- En Python son fáciles de definir
  - `miTupla = ("Shirley", 42 )`

Arreglos-Listas (ArrayLists)

- En Java, estas tienen más métodos que los arreglos.

- Puede ser llenado con objetos de más diversas clases

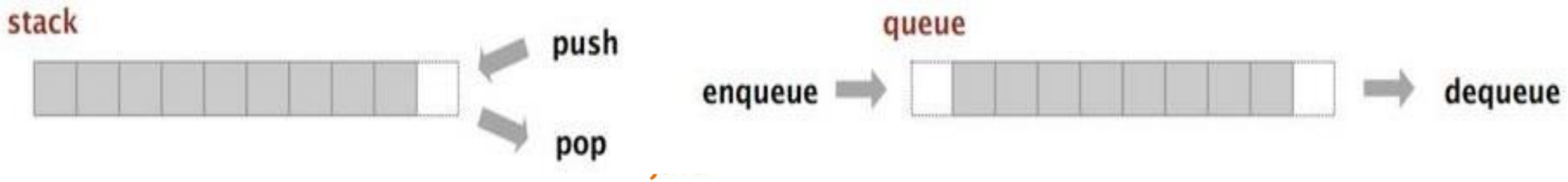

```
ArrayList<Integer> miArregloLista2 = new ArrayList<Integer>(Arrays.asList(1,2,3,4,5));

ArrayList<Pair<Character, Double>> miArregloListaDiverso = new ArrayList<Pair<Character, Doub
```

Métodos comunes

```
miArregloLista2.size();

miArregloLista2.add(10);

miArregloLista2.remove(0);

miArregloLista2.contains(19);

miArregloLista2.toString();
```

Pilas y Colas (Stack and Queue)

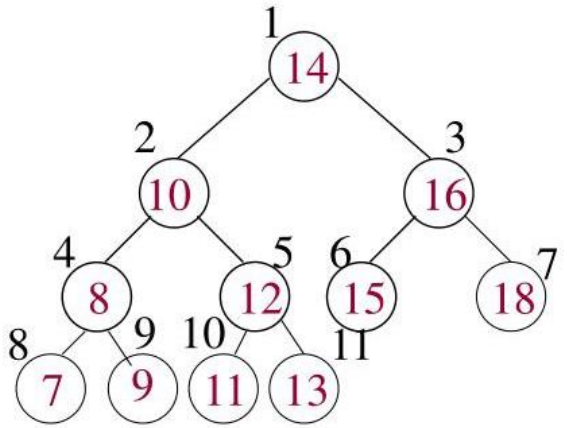
Las pilas siguen la norma de LIFO (el ultimo que entra es el primero que sale)
Las colas siguen la norma FIFO (el primero que entra es el primero que sale)
Útil en ciencia computacional



https://developpaper.com/stack-and-queue-algorithms-part-i-week-2-stacks-and-queues/

Árboles
(breve
presentación)

Binary Trees: Array Representation



	1	2	3	4	5	6	7	8	9	10	11
Array A:	14	10	16	8	12	15	18	7	9	11	13

7

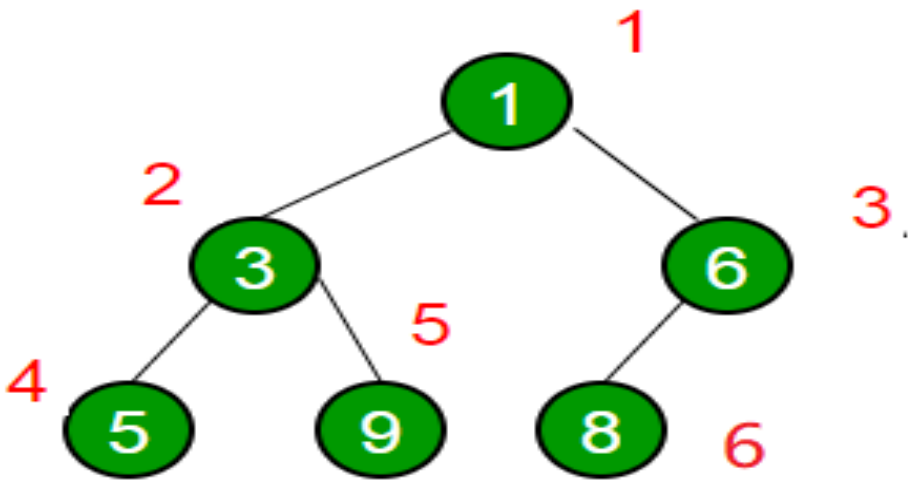
Importante: La esquematización es realizada por cuestiones didácticas.

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

Montículos (Heap)

Árbol en donde nodo central es el valor mínimo

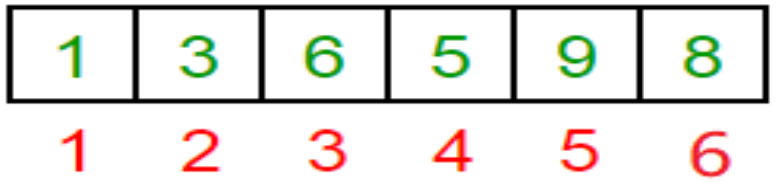
- En Java, podemos importar la clase PriorityQueue la cual los elementos de puede hacer lo que de un montículo diseñado a hacer.




```
Queue<Integer> miMonticulo = new PriorityQueue<
```

Métodos comunes

```
miMonticulo.poll();

miMonticulo.add(12);

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



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

Diccionario (Hashmap/Dictionary)

Útiles cuando quiero almacenar valores en pares de llave-valor, en donde la llave permite el acceso al valor.

- En Java, podemos instanciar la clase Hashmap

```
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();
```

- En Python:

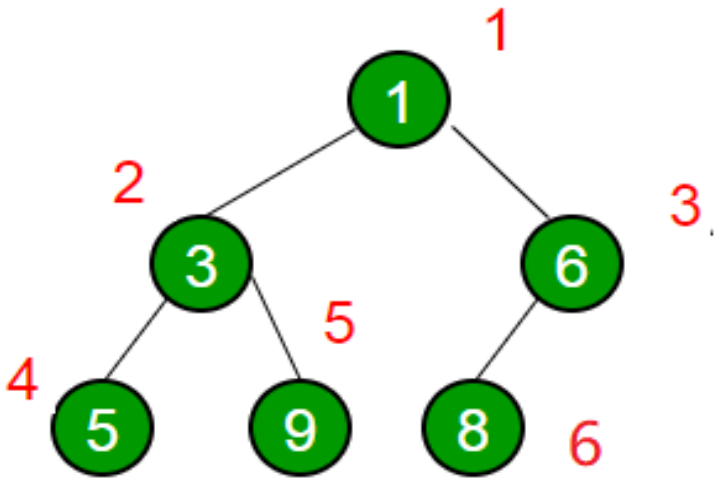
```
tablaPeriodica = {
    "Sodio": "Na",
    "Carbono": "C",
    "Helio": "He"
}
```

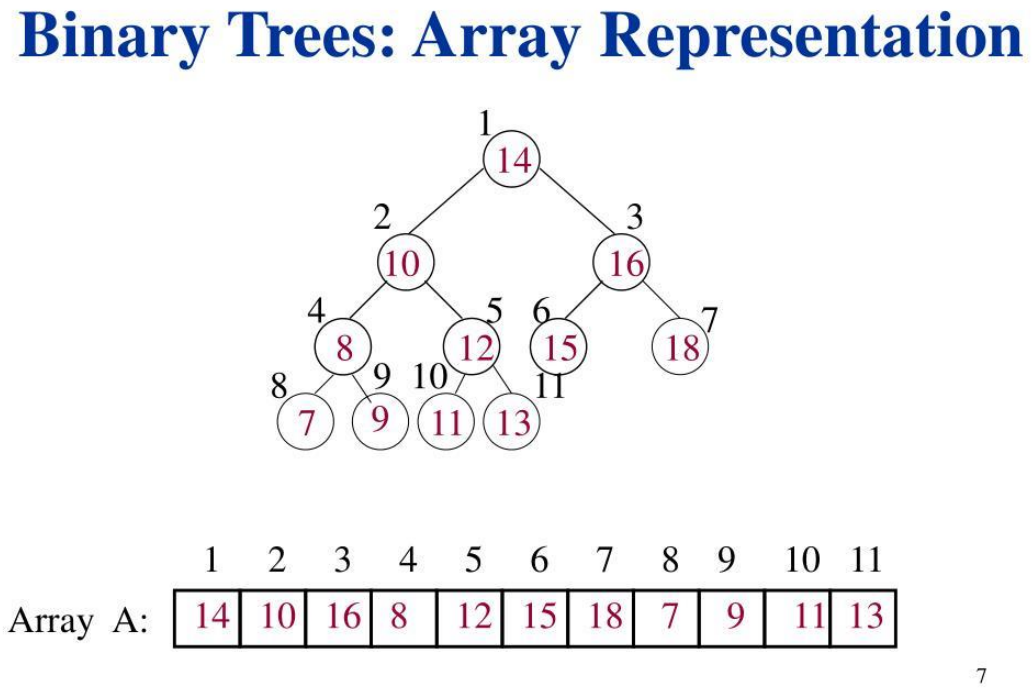
Comentarios:

- Estas son *algunas* estructuras de datos. Hemos omitido:
  - Estructures concatenadas (linked-lists)
  - Grafos
  - Árboles
  - Vectores, matrices
  - Entre otros.
- Desafío: qué estructuras de datos son las más adecuadas a usar para resolver cierto problema.
  - Rasuradora de Ocam: lo más sencillo es mejor
- No es necesario memorizarlas, sino practicar usándolas:
  - https://docs.oracle.com/en/java/javase/18/docs/api/
- Podemos también visualizar estructuras de datos
  - https://www.cs.usfca.edu/%7Egalles/visualization/Algorithms.html

Pensamiento computacional

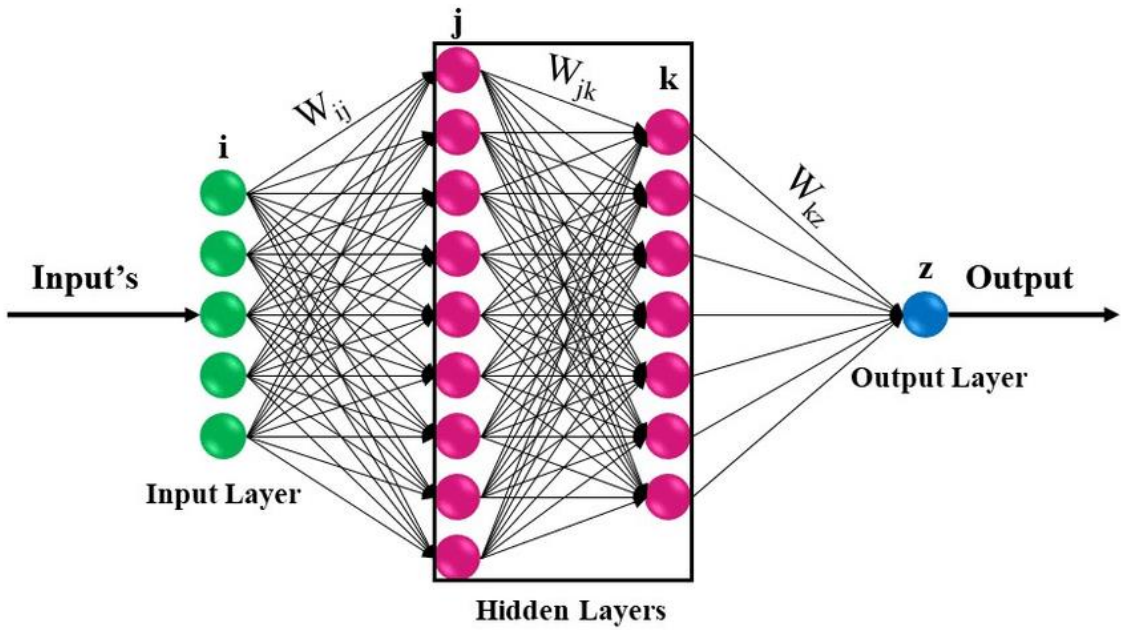
- Estructuras de datos tratan de representar la realidad

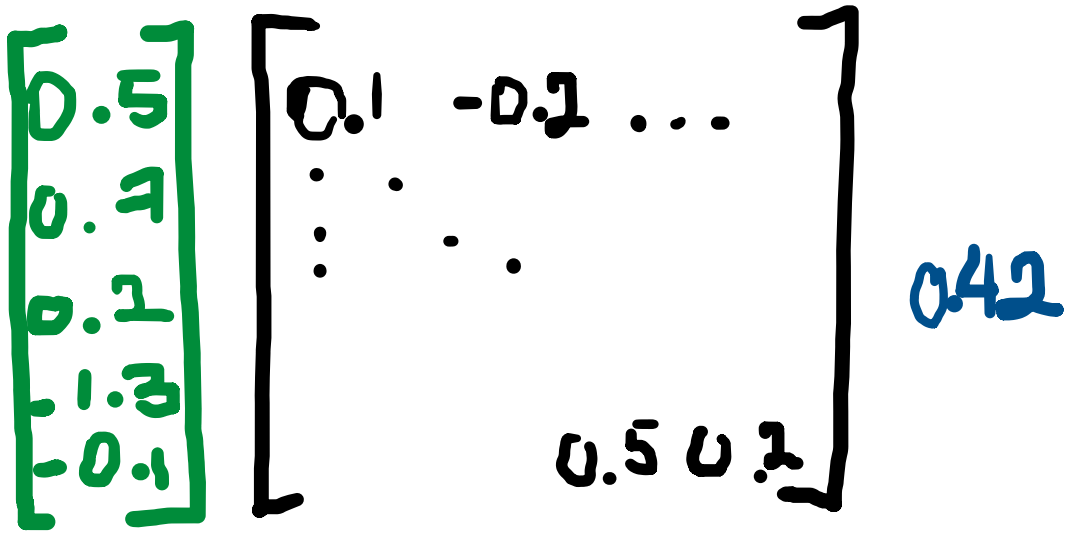




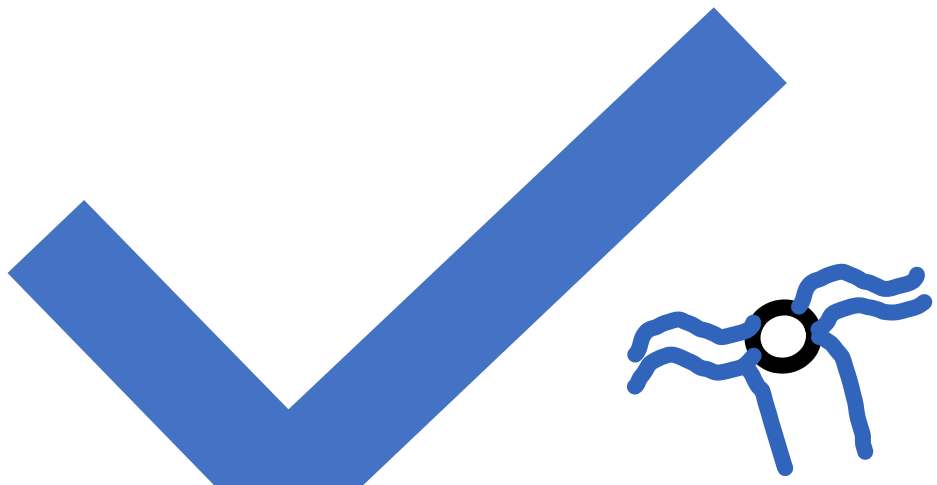
Visión computacional: ¿es o no suficiente una matriz de números para esta representación?

[33, 60, 43, ..., 47, 55, 37],
[45, 42, 24, ..., 41, 22, 19]]], dtype=uint8)





https://www.researchgate.net/figure/The-basic-form-of-multilayer-perceptron-artificial-neural-network-ANN-61_fig1_341626283



Conclusión:

- Java es un lenguaje de programación orientada a objetos: existen diversas estructuras de datos.
- El desafío de un programador incluye saber cómo descomponer un problema para resolverlo a través de objetos adecuados.
  - Rasuradora de Ocam: elige el que es necesario, no necesariamente el más usado.
- Pensamiento computacional: lo que la computadora procesa es diferente a lo que vemos en esquematizaciones: *todo se trata de arreglo de números* ;)

Futuras perspectivas:

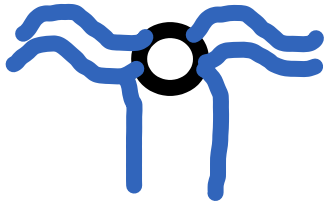
- El estudio más avanzado de ciencias computaciones incluye el análisis de estas estructuras de datos:
  a. ¿Cuáles son más convenientes en qué situación?
  b. ¿Cuáles permiten un tiempo más rápido de ejecución y más óptimo uso de memoria? Análisis asintótico.

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

Capítulo IV.II - Algoritmos

Interpretación e implementaciónón

Algoritmo



Una especificación (paso-a-paso) de cómo resolver un problema/completar una tarea

Escrito de manera “universal”: todo lenguaje de programación puede implementarlo.

Algoritmo para encontrar valor máximo y mínimo

- Dado $\left[\frac{1}{5}, -\frac{1}{2}, 3\pi, \frac{10\pi}{3}, \log 2\,, \log -2\pi\,, \sin 2\pi\right]$, ¿cuál es el valor mínimo y máximo del arreglo?

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/

Clásico: sortear un arreglo

- Dado $\left[\frac{1}{5}, -\frac{1}{2}, 3\pi, \frac{10\pi}{3}, \log 2\,, \log -2\pi\,, \sin 2\pi\right]$, sortéalo en orden ascendiente.

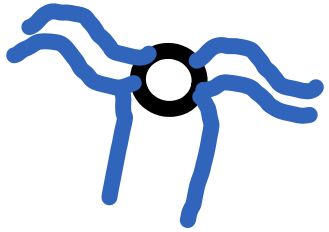
Sorteo de Burbuja

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*

Comentarios:



- Puedes practicar implementando los algoritmos InsertionSort, MergeSort, HeapSort, entre otros.
  - Intenta con algún algoritmo que te atraiga

- Puedes *editar* SorteoBurbuja para
  - Ordenar de manera descendiente
  - Evitar iteraciones redundantes

- Podemos también visualizar la ejecución de algoritmos
  - https://www.cs.usfca.edu/%7Egalles/visualization/Algorithms.html

- Hemos omitido: análisis de eficiencia de algoritmo en términos de tiempo y consumo de memoria – Análisis asintótico.

¿Qué pasaría si…

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
```

VS.

Ingreso*: *Montículo mon

Egreso*: *valor mínimo

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

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

Conclusión

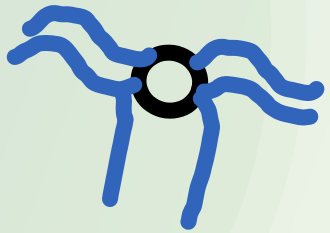
- Un algoritmo es una especificación para completar una tarea

- Retos personales:
  - Entender cómo funciona un algoritmo en adición a solo usarlos.
  - Analizar diferentes algoritmos es una rama de computación
  - Cormen, T. H., Charles Eric Leiserson, Rivest, R. L., Stein, C., & Al, E. (2009). Introduction to algorithms. MIT Press.

https://www.clker.com/clipart-java-coffee-drink.html

Capítulo IV.III – Definiendo nuestras propias clases

Clases, campos y métodos



http://www.clipartbest.com/pictures-of-animated-pythons

Objetivos:

¿Cómo definir clases?

Campos, constructor y métodos

Modificadores públicos vs privados

Métodos sobrescritos

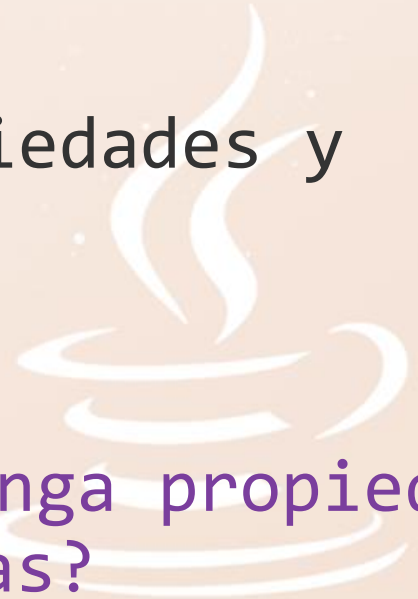
Introducirnos a resolver problemas a través de clases personalizadas

Recapitulando: Hemos estado trabajando con objetos

- `int[] arregloSencillo = new int[]{1, 2, 3};`

- Objetos: Instanciaciones de clases
- Clase: templete que especifica propiedades y funciones del objeto instanciado

- ¿Qué pasa si quiero un objeto que tenga propiedades y funciones más complejas?

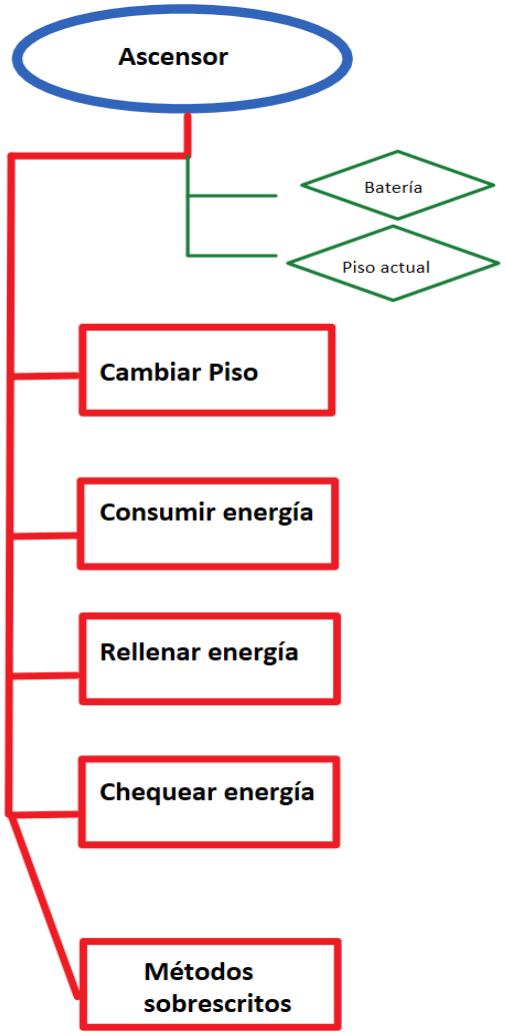


Problema ejemplar

- Simular un ascensor:
  - Tiene dos propiedades: Piso actual y electricidad que sobra
  - 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.

http://dept.swccd.edu/hlee/content/phys-170/lecture-web-04a/

Una combinación de todas



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, privados y sobreescritos).

Modificadores

	Campos	Métodos/Funciones
Públicos	propiedad cuyo valor 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”
Sobrescritas		Métodos predefinidos de Java que toda clase debe tener y que pueden ser redefinidos. Por ejemplo, toString()

Problema adicional

- Extiende las funcionalidades del ascensor:
  - 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.

```
1. InsertionSort(arr[])

2.       for j = 1 to arr.length

3.          key = arr[j]

4.          i = j - 1

5.          while i > 0 and arr[i] > key

6.             arr[i+1] = arr[i]

7.             i = i - 1

8.          arr[i+1] = key
```

¡O decide qué quieres simular!

Comentarios

- Una clase también puede ser definida como un **ensamblaje de estructuras de datos**
- Decisiones sobre qué modificadores usar dependen en parte del problema y tu juicio
- Pensamiento computacional: divide y vencerás

Capítulo IV.IV – Definiendo clases abstractas

Herencia de campos y métodos

Objetivos

- Hoy vamos a introducirnos a la definición de clases abstractas y a resolver problemas a través de estas.
  - Campos finales
  - Sobre escribir el método *equals*

Clase abstracta

- Si una clase es un templete, entonces una clase abstracta es un templete del templete

http://www.visitmysmokies.com/treasure-map-2/

Extensiones

http://riddles-for-kids.org/treasure-map-riddles/

https://friendlystock.com/product/treasure-map/

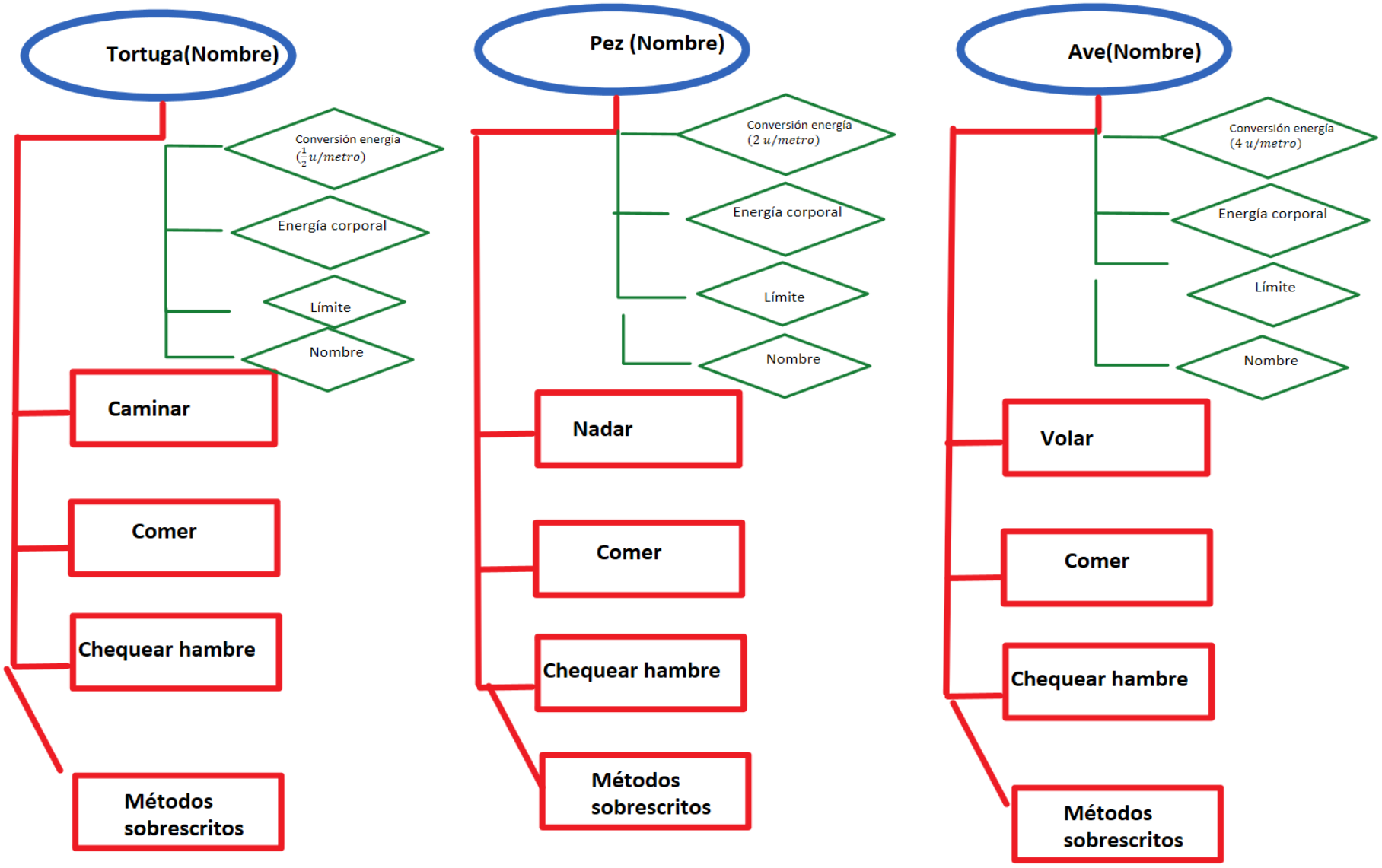
http://www.timvandevall.com/printable-treasure-maps-for-kids/

Mapa abstracto: todo mapa debe tener un marco de referencia al lado derecho, y el tesoro marcado con X

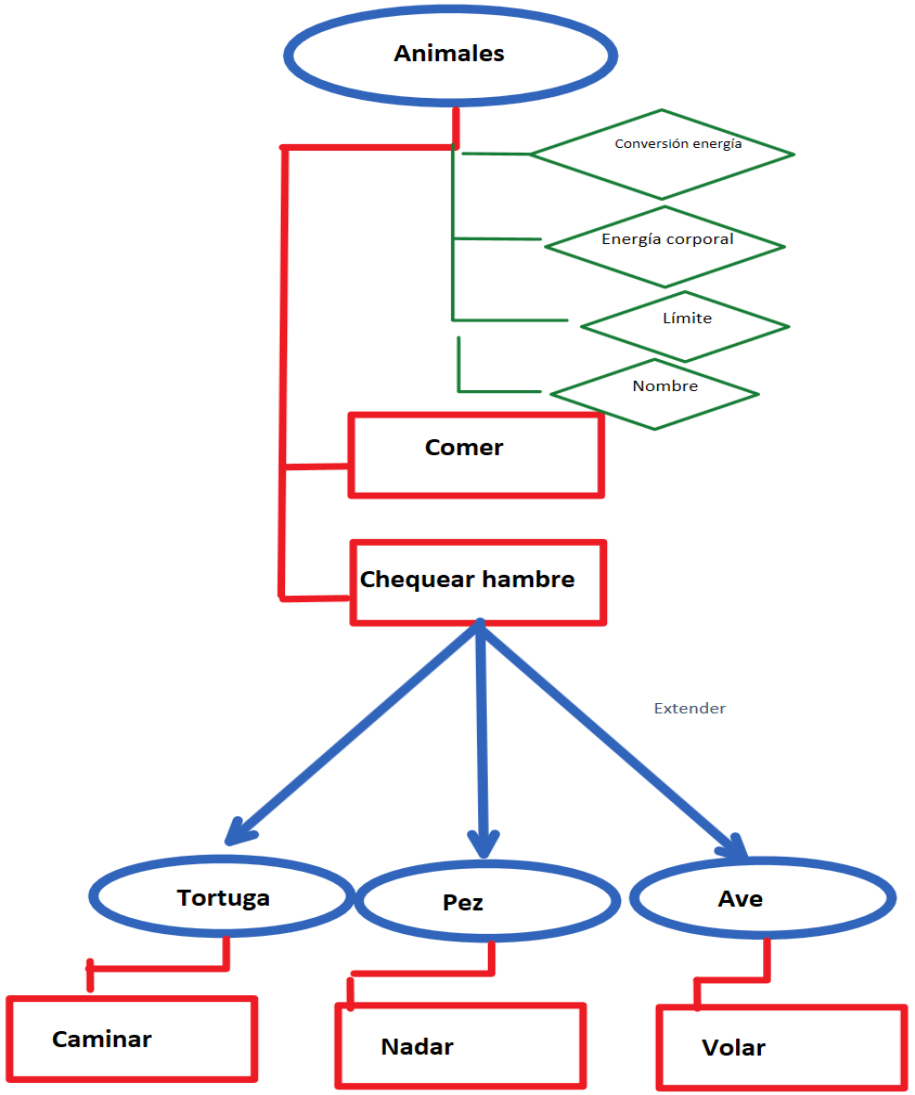
Problema ejemplar

- Simular el comportamiento de animales de una reserva de Galápagos
  - Todo animal es inicializado con un nombre y la energía corporal que tienen.
  - Una tortuga camina consumiendo energía. Por cada paso que toma, consumen medio Kilo Joule energía.
  - Un pez nada, y las condiciones de energía son iguales que la de la tortuga, con la excepción de que la conversión de es por cada paso, un pez consume 2 Kilo Joules de energía.
  - Un ave vuela y las condiciones de energía son iguales que la de la tortuga, con la excepción de que la conversión es por cada paso, una ave consume 4 Kilo Joules de energía
  - Todo animal puede moverse cuando no tienen hambre, en donde hambre es denotado si tienen menos de 1000 Kilo Joules de energía.
  - Todo animal puede comer con el fin de recargar la energía corporal.

Alternativa: implementar 3 clases distintas



Alternativa: extender una clase abstracta



- Código más limpio y conciso
- Clase abstracta puede generalizar hacia más clases

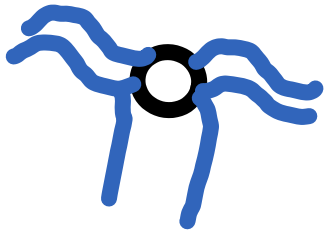
Comentarios

- Métodos abstractos vs. No abstractos
  - Métodos abstractos tienen que ser editados en las clases extendidas
  - Métodos abstractos son heredados por las clases extendidas “al pie de la letra”

- ¿Qué pasa si quiero chequear si dos objetos que son iguales?
  - Sobreescribir equals()

	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”
Protegido	Propiedad que es visible a través de clases de un mismo paquete	
Finales	Propiedad cuyo valor es una constante inmutable	
Sobrescritas		Métodos predefinidos de Java que TODA clase debe tener y que pueden ser redefinidos (toString(), equals(), hashCode())

Comentarios finales



- 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.
- **Hemos obviado Interfaces e Enums, pero en breve**:
  - En Interfaces solo pueden haber métodos abstractos y permite herencia múltiples
  - Los Enums (enumeradores) permiten la definición de constantes.

Capítulo IV.V: Práctica, consejos y conclusión del capítulo

Objetivos

Completar juntos 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

Python
LeetCode, HackerRank
Adentrarse a comunidad y resolver errores

Ejercicios

- 1. Tomar entradas del usuario para un arreglo de decimales y ordenarlo usando sorteo de burbuja *ascendente*.

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

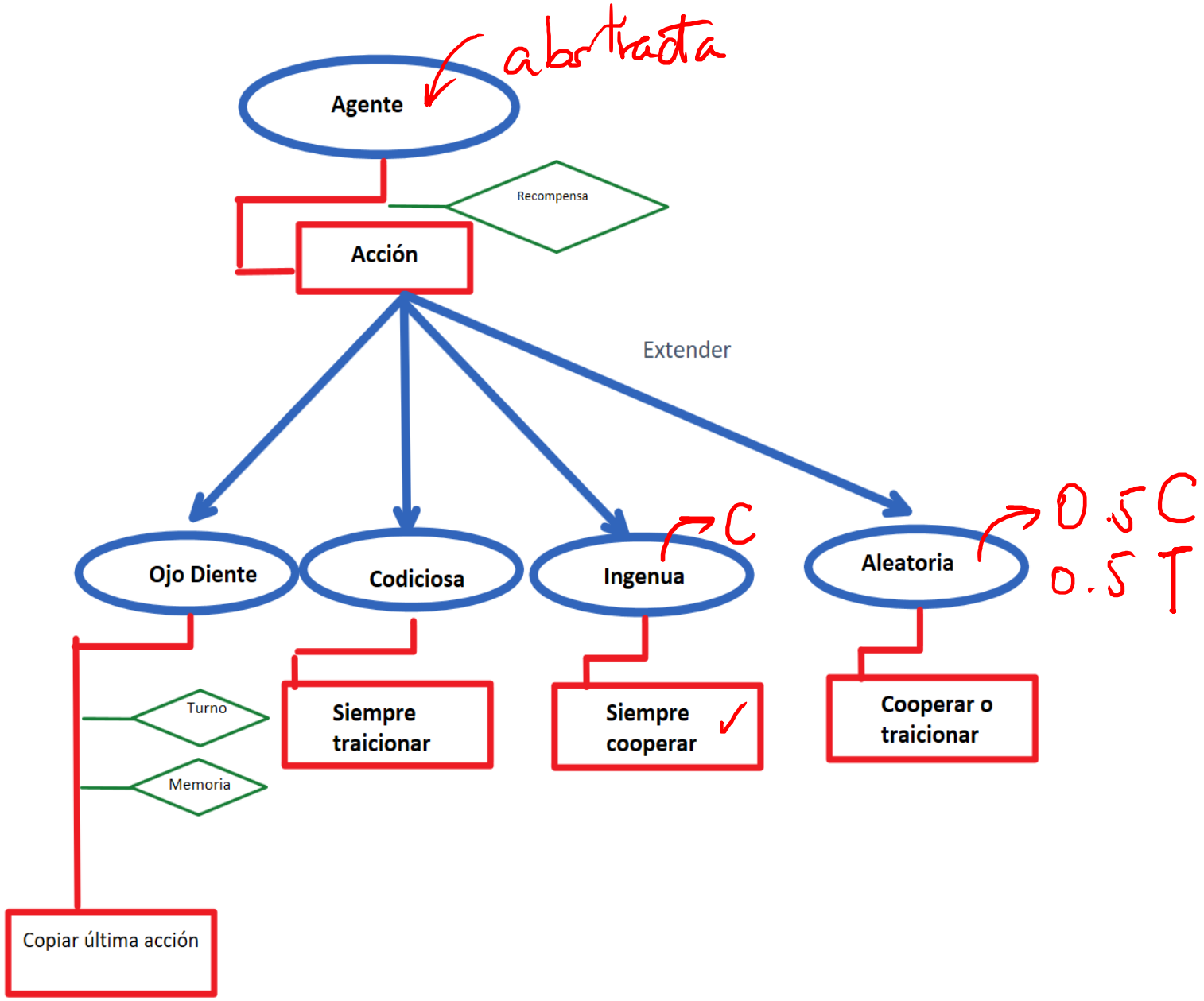
Es crucial que entiendas cómo funciona: https://www.cs.usfca.edu/%7Egalles/visualization/ComparisonSort.html

a. Pensar sobre cómo mejorar el algoritmo. ¿Qué pasa si te doy un arreglo que ya está ordenado ascendentemente?

- 2. Implementar la lógica de un juego filosófico llamado dilema del prisionero iterado:
  - 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.
  - Para observar una simulación interactiva, por favor visitar: https://ncase.me/trust

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

Diagrama

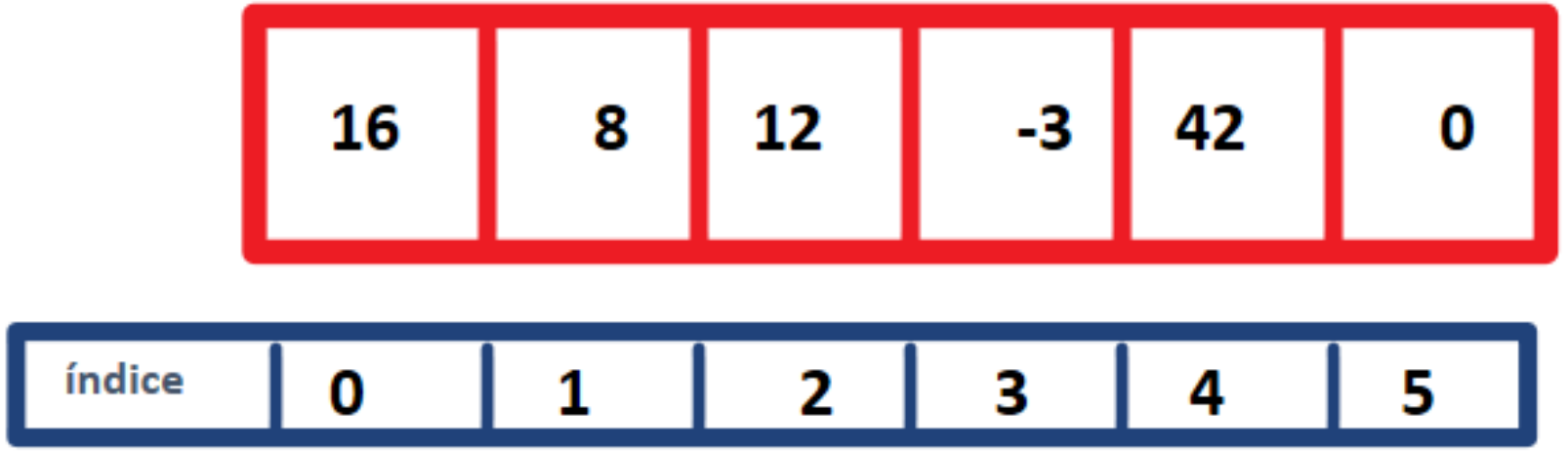

Agente
abstracta
Recompensa
Acción
Extender
Ojo Diente
Codiciosa
Ingenua
C
Aleatoria
0.5 C
0.5 T
Turno
Memoria
Siempre traicionar
Siempre cooperar
Cooperar o traicionar
Copiar última acción

- 3. Implementar una lista vincular singular con las funciones respectivas de:
  - insertar un nodo dado el valor,
  - eliminar un nodo dado el valor,
  - Acceder a un nodo de acuerdo al índice indicado,
  - encontrar el tamaño de la lista vinculada,
  - intercambiar la posición de nodos dada dos índices,
  - sortear la lista de manera descendente,
  - entre otros métodos no-estáticos que consideres útil.
  - Sobrescribir los métodos toString, equals y hashcode.
    - Una representación en cadena aceptable puede ser así:

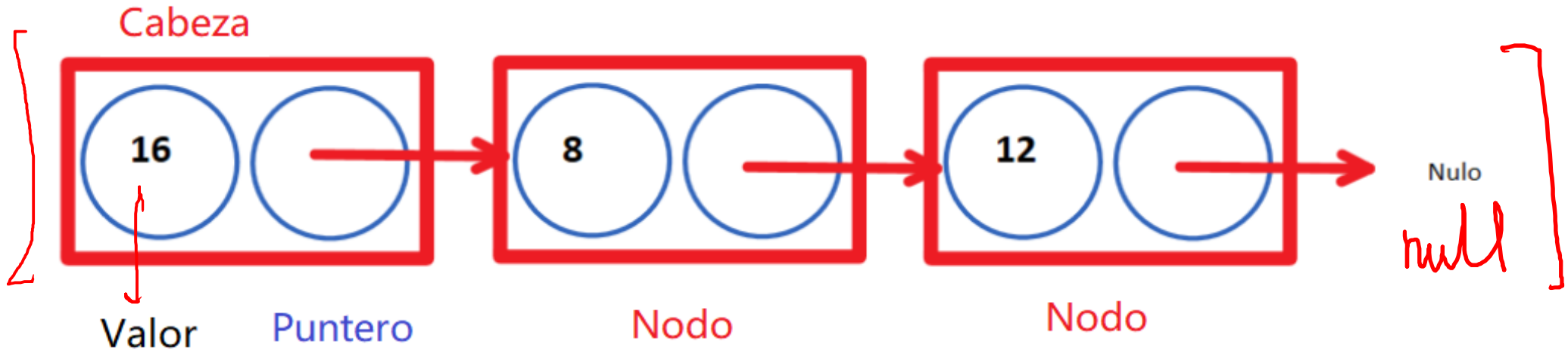
      0->1->2->3->4->5->6->7->8->9->10->nulo

```
1.InsertionSort(arr)
2.        for j = 1 to arr.length
3.            valorReferencial = arr.get(j)
4.            i = j - 1
5.            while i > 0 and arr.get(i) >valorReferencial
6.              intercambiar(arr.get(i + 1), arr.get(i))
7.                i = i - 1
```

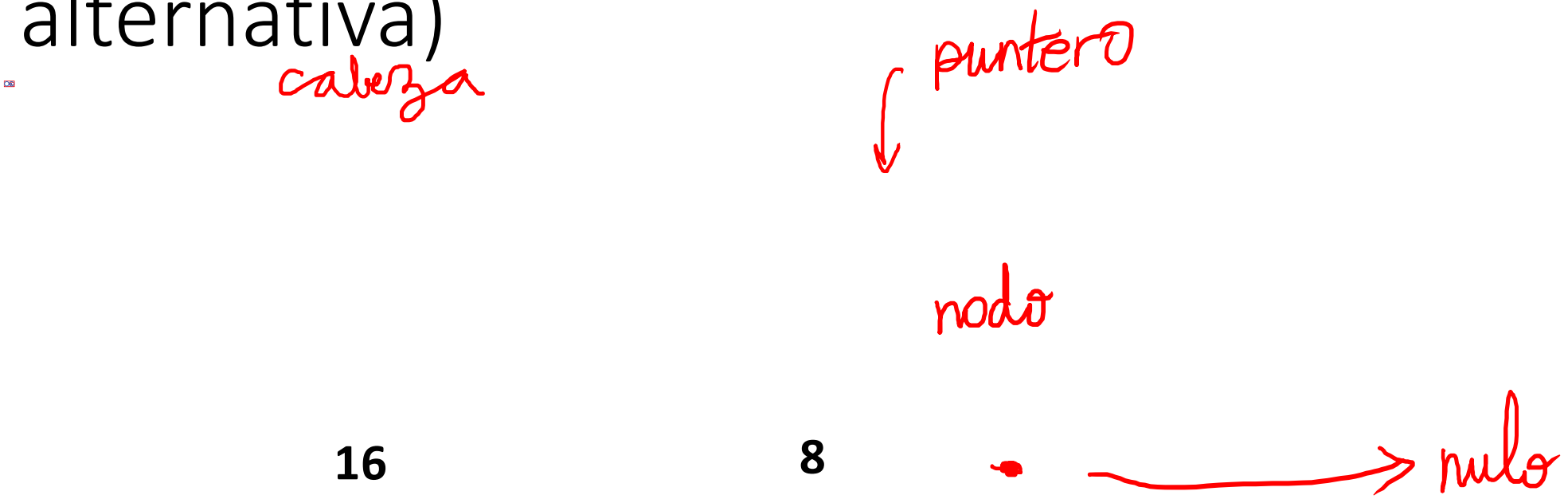
Arreglo-Lista (Ilustración)



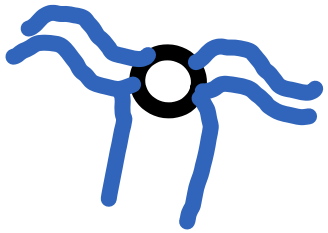
Lista vinculada singular I (Ilustración)



Lista vinculada singular II (ilustración alternativa)



Lista vinculada singular III - ¿cómo acceder al valor cada nodo?



- ¿cómo acceder a cierto elemento en una lista vinculada?
  1. Para el primer valor: `cabeza.valor`
  2. Para el segundo valor:

     nodo

     `cabeza.punteroSiguiente.valor`
  3. Para el tercer valor:

     `cabeza.punteroSiguiente.punteroSiguiente.valor`

¿Cómo generalizarlo en código?

```
ListaVincular nodo;
while (nodo.punteroSiguiente != null) {
        int valorActual =   nodo.valor;
        nodo = nodo.punteroSiguiente;}
```

Comentarios finales

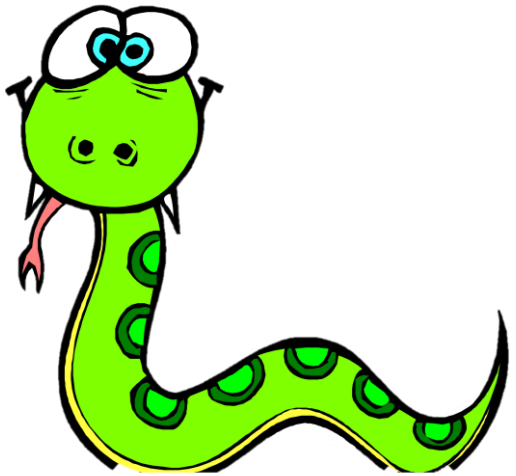
- Puedes analizar los algoritmos que hemos visto hasta ahora para decider cómo “mejorarlos”.
- Las listas vinculadas son flexibles para practicar uno que otro tema de computación

	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

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

- Visitar Github, StackOverflow, o comunidades digitales de programación para familiarizarte con errores comunes y cómo resolverlos, ejemplo: https://community.oracle.com/tech/developers/discussion/3635195/handling-an-input-mismatch-exception

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