Colección de problemas de POO

Versión 19.05.30

Manel Guerrero Zapata

Índice

1 Programación Básica

1 Cálculo del área de un triangulo

Haz un programa que pida la base y la altura de un triangulo y te calcule su área.

```
run:
¿Como te llamas? Pepe
Hola Pepe! Vamos a calcular el area de un triangulo. :-)
- Base: 7
- Altura: 4
El area de un triangulo de base 7.0 y altura 4.0 es 14.0.
BUILD SUCCESSFUL (total time: 16 seconds)
```

2 Algoritmos recursivos versus iterativos

Las implementaciones iterativas (con bucles) de algoritmos que pueden ser pensados como recursivos (función que se invoca a sí misma) son más eficientes (una llamada a función tiene un coste en tiempo y en memoria no menospreciable) que sus implementaciones recursivas equivalentes.
Y es por esto que las preferiremos. No obstante, en algunos casos excepcionales, el código recursivo puede ser más simple.
Existen casos excepcionales donde el algoritmo recursivo es más eficiente que el iterativo. Por ejemplo, en el buscaminas, si destapamos una casilla que no tiene bombas en su casillas vecinas estas a su vez se destapan y así sucesivamente.

- Implementa las versiones recursivas e iterativas de cálculo de factorial de 'n' y cálculo del Máximo Común Divisor de dos números.
- Implementa también la versión iterativa de un programa que imprima los 'n' primeros elementos la serie de Fibonacci.
- Razona porque la versión recursiva de Fibonacci no tiene sentido.
Pista: ¿Cuantas invocaciones al método necesitas para, por ejemplo, Fibonacci(10)?
- Razona porque la versión recursiva del “destape en cascada” del buscaminas es más eficiente que la versión iterativa.
Pista: Una casilla es vecina de las casillas que son sus vecinas.
Pista: La implementación recursiva es “depth-first” y la iterativa es “breadth-first”.

- https://es.wikipedia.org/wiki/B%C3%BAsqueda_en_profundidad
- https://es.wikipedia.org/wiki/B%C3%BAsqueda_en_anchura

En el tema “Problemas complejos” uno de los problemas consiste en que implementéis todo el buscaminas. B-)

Recuerda:

- Factorial: 0! = 1, 1! = 1, n! = n * (n-1)!
- MCD(a,0) = a, MCD(a,b) = MCD (b, a modulo b).
- Fibonacci: 0, 1, y a partir de estos cada término es la suma de los dos anteriores.

```
run: (factorial)
6! = 720
BUILD SUCCESSFUL (total time: 0 seconds)

run: (MDC)
a = 654,    b = 2322
a = 2322,   b = 654
a = 654,    b = 360
a = 360,    b = 294
a = 294,    b = 66
a = 66,     b = 30
a = 30,     b = 6
a = 6,      b = 0
MCD(654, 2322) = 6
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
run: (Fibonacci (elementos separados entre “, “ y acabado en “.”))
Imprimiendo los 20 primeros elementos de la serie de Fibonacci:
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597, 2584,
4181.
BUILD SUCCESSFUL (total time: 0 seconds)
```

3 Arrays y algoritmos de ordenación

Implementad un programa que ordene un vector de enteros utilizando el algoritmo de ordenamiento por inserción. Implementad otro que lo ordene utilizando el algoritmo de ordenamiento de burbuja. Ver que en algoritmo de inserción el vector se va ordenando de principio a final y en de la burbuja de final al principio. Ver que el algoritmo de la burbuja es menos optimo que el de inserción.

En realidad nunca ordenaremos manualmente un vector (este ejercicio es sólo para que trabajar vuestra algorítmica). Si queremos ordenar un vector utilizaremos el método:
static void Arrays.sort(Object[])

Mirar:
https://es.wikipedia.org/wiki/Ordenamiento_por_inserci%C3%B3n
https://es.wikipedia.org/wiki/Ordenamiento_de_burbuja
https://es.wikipedia.org/wiki/Algoritmo_de_ordenamiento
https://docs.oracle.com/javase/9/docs/api/java/util/Arrays.html

```
run (Insercion):
[6, 5, 3, 1, 8, 7, 2, 4]
[5, 6, 3, 1, 8, 7, 2, 4]
[3, 5, 6, 1, 8, 7, 2, 4]
[1, 3, 5, 6, 8, 7, 2, 4]
[1, 3, 5, 6, 8, 7, 2, 4]
[1, 3, 5, 6, 7, 8, 2, 4]
[1, 2, 3, 5, 6, 7, 8, 4]
[1, 2, 3, 4, 5, 6, 7, 8]
BUILD   SUCCESSFUL   (total   time:   0
seconds)
```

```
run (burbuja):
[6, 5, 3, 1, 8, 7, 2, 4]
[5, 3, 1, 6, 7, 2, 4, 8]
[3, 1, 5, 6, 2, 4, 7, 8]
[1, 3, 5, 2, 4, 6, 7, 8]
[1, 3, 2, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8]
[1, 2, 3, 4, 5, 6, 7, 8]
BUILD   SUCCESSFUL   (total   time:   0
seconds)
```

2 Clases Básicas: Constructores, getters, setters y otros métodos

1 Invertir coordenadas de un punto bidimensional

Implementa una clase Point con sus atributos x e y. La clase debe contener: su constructor, los getters y setters, un invertirCoordinates() que invierta las coordenadas x e y del punto. Además la clase debe tener un toString() para poder imprimir los puntos en formato "(x,y)".

Implementa un clase tester PointTester con un método main() donde crees un punto, lo imprimas utilizando de manera implícita el toString(), imprimas su coordenada x, asignes 0 a su coordenada x, y vuelvas a imprimir el punto.

```
run:
Point 1: (3.0,2.0)
I invert coordinates
The X coordinate of point 1 is: 2.0
I set X coordinate to 0
Point 1: (0.0,3.0)
BUILD SUCCESSFUL (total time: 0 seconds)
```

2 Coordenadas polares y cartesianas. Atributos de la clase versus propiedades de la clase

Las propiedades de una clase son todo aquello que tiene un getter. Algunas de las propiedades se corresponden a un atributo, otras pueden ser calculadas.

A) Haz una clase punto bidimensional con atributos x e y, con su constructor Point(double x, double y), y con getters para x, y, r ("coordenada radial") y angulo ("coordenada angular"). Los getters de las propiedades x e y se obtienen las propiedades directamente de los atributos. Y los getters de r y angulo calculan estas propiedades a través de los atributos x e y.

Mirad el siguiente enlace para ver como se calculan r y angulo:
https://es.wikipedia.org/wiki/Coordenadas_polares

Recordad que para calcular la arctangente podéis usar Math.atan() y que la constante Pi en Java está en Math.PI.

Añadid los métodos toString() "(x=1.0,y=1.0)"
y toStringPolar() "(r=1.4142135623730951,angulo=45.0)".

Cread una clase tester que cree un punto y los imprima de las dos formas.

```
run:
p1: (x=1.0,y=0.0)
p1: (r=1.0,angulo=0.0)

p1: (x=1.0,y=1.0)
```

```
p1: (r=1.4142135623730951,angulo=45.0)

p1: (x=0.0,y=1.0)
p1: (r=1.0,angulo=90.0)

p1: (x=-1.0,y=1.0)
p1: (r=1.4142135623730951,angulo=135.0)

p1: (x=-1.0,y=0.0)
p1: (r=1.0,angulo=180.0)

p1: (x=-1.0,y=-1.0)
p1: (r=1.4142135623730951,angulo=225.0)

p1: (x=0.0,y=-1.0)
p1: (r=1.0,angulo=270.0)

p1: (x=1.0,y=-1.0)
p1: (r=1.4142135623730951,angulo=315.0)

BUILD SUCCESSFUL (total time: 0 seconds)
```

B) Haz otra versión de la clase punto donde las cabeceras de los métodos (constructor incluido) sean las mismas, pero ahora los atributos son r y a (“coordenada radial” y “coordenada angular”).
El tester no requiere ninguna modificación porque en Java usamos encapsulación (cada clase esconde a las otras clases como está implementada por dentro y se puede reimplementar de forma diferente siempre y cuando se mantengan las cabeceras de los métodos).

```
run:
p1: (x=1.0,y=0.0)
p1: (r=1.0,angulo=0.0)

p1: (x=1.0000000000000002,y=1.0)
p1: (r=1.4142135623730951,angulo=45.0)

p1: (x=6.123233995736766E-17,y=1.0)
p1: (r=1.0,angulo=90.0)

p1: (x=-1.0,y=1.0000000000000002)
p1: (r=1.4142135623730951,angulo=135.0)

p1: (x=-1.0,y=1.2246467991473532E-16)
p1: (r=1.0,angulo=180.0)

p1: (x=-1.0000000000000002,y=-1.0)
p1: (r=1.4142135623730951,angulo=225.0)

p1: (x=-1.8369701987210297E-16,y=-1.0)
p1: (r=1.0,angulo=270.0)

p1: (x=0.9999999999999998,y=-1.0000000000000002)
p1: (r=1.4142135623730951,angulo=315.0)

BUILD SUCCESSFUL (total time: 0 seconds)
```

Fijaros que ahora las componente x e y salen con un pequeño error de cálculo porqué han sido calculadas a partir de r y a.

¿Cual de las dos implementaciones es mejor? Depende de si me van ha usar más los getters de x e y, o los getters de r y a.

C) Haz una última versión de la clase punto con las mismas cabeceras de métodos donde ahora los cuatro (x, y, a y r) son atributos. El programa debería imprimir lo mismo que el del apartado A.

3 Cálculo de área y perímetro de un rectángulo

A) Implementa la clase Punto (con x e y) y la clase Rectangulo (determinado por dos puntos) (las dos con sus constructores, getters, setters y toString()). La clase rectángulo tendrá además dos métodos para calcular su área y su perímetro. Implementa también una clase tester que cree dos puntos y un rectángulo a partir de estos dos puntos y que muestre el área y e perímetro de dicho rectángulo.

```
run:
Point 1: (3.0,2.0)
Point 2: (0.0,0.0)
Rectangle: ((3.0,2.0),(0.0,0.0))
Area: 6.0
Perimeter: 10.0
BUILD SUCCESSFUL (total time: 1 second)
```

B) Haz una segunda versión de Rectangulo donde las propiedades área y perímetro se implementen como atributos.

4 Dado

Implementa la clase Dado. Por defecto el dado tendrá 6 caras. Tendremos tres constructores (uno al que no se le pasa nada e inicializa el dado al azar, otro al que sólo se le pasa que número tiene el dado en la cara superior y otro con el número del dado en la cara superior y el número de caras del dado). Implementa los getters, el método tirarDado() que tirará el dado al azar y el toString() de acuerdo con el ejemplo de ejecución. Implementa un tester que tenga un vector de 4 dados y los lance una serie de veces.

```
run:
Lanzo 4 dados:
---
  o
 o
o
---
---
o o
 o
o o
---
---
o o
 o
o o
---
---
o o
```

```
Lanzo 4 dados:
---
o o
 o
o o
---
---

 o

---
---
  o
 o
o
---
---
o o
o o
```

```
o o
o o
---
```

```
o o
---
BUILD SUCCESSFUL (total time: 0 seconds)
```

5 Fracciones

Implementa la clase Fracción. Con constructores, getters, setters, toString() que sume, reste, multiplique y divida fracciones y de su resultado en su versión reducida. Implementa un tester para probarla.

```
run:
Numerador de la primera fraccion: 2
Denominador de la primera fraccion: 4
Numerador de la segunda fraccion: 7
Denominador de la segunda fraccion: 2
2/4 + 7/2 = 4/1
2/4 - 7/2 = -3/1
2/4 * 7/2 = 7/4
2/4 / 7/2 = 1/7
BUILD SUCCESSFUL (total time: 15 seconds)
```

6 Un comentario sobre la manipulación de Strings

La clase String es inmutable en Java. Cada vez que hacemos alguna operación (como concatenación, o obtener una parte del String) se genera un nuevo String con el resultado de dicha operación y el viejo se descarta (y el recogedor de basura la eliminará). Esto puede llegar a ser bastante ineficiente. Para solucionar este problema tenemos la clase StringBuilder.

La clase StringBuilder es más adecuada para la manipulación de Strings porque, StringBuilder es mutable e irá mucho más rápido. Existe también la clase StringBuffer pero es más lenta que StringBuilder y sólo se usa en programas que usan varios hilos de ejecución (no es nuestro caso).

Nota: System.out.print() y println() pueden imprimir directamente StringBuilders ya que StringBuilder implementa el método toString().

Versión del método toString() de una clase Punto2D usando un String:

```
    @Override
    public String toString() {
        return "(" + x + "," + y + ")";
    }
```

Y versión que usa StringBuilder:

```
    @Override
    public String toString() {
        StringBuilder s = new StringBuilder("(");
        s.append(x).append(",").append(y).append(")");
        return s.toString();
    }
```

La versión que usa StringBuilder es más engorrosa y difícil de leer que la que usa String. Y, en cualquier caso es muy probable que la versión que usa String es cambiada por la que usa StringBuilder por el optimizador JIT (Just In Time). JIT compila en tiempo de ejecución el bytecode a código máquina nativo y realiza muchas optimizaciones. Casi todas las optimizaciones se realizan en tiempo de ejecución, porque es mejor hacerlas para una arquitectura específica. (El compilador de Java que compila el fichero Java a bytecode hace muy pocas optimizaciones).

Pero hay un caso donde JIT no se “atreve” a hacer esta optimización, y es cuando las concatenaciones se realizan en el cuerpo de un bucle.

Por lo tanto usaremos StringBuilder solo en bucles como en el método toString() de la clase Arrays (que invocábamos en los algoritmos de ordenación):

```
public class Arrays {

    public static String toString(long[] a) {
        if (a == null)
            return "null";
        int iMax = a.length - 1;
        if (iMax == -1)
            return "[]";

        StringBuilder b = new StringBuilder();
        b.append('[');
        for (int i = 0; ; i++) {
            b.append(a[i]);
            if (i == iMax)
                return b.append(']').toString();
            b.append(", ");
        }
    }

}
```

3 Colecciones (o contenedores)

1 Aclaración sobre la terminología

Las colecciones (collections en inglés) y menos frecuentemente referidas como contenedores (containers) no deben confundirse con la clase Collection o con la clase de utilidades Collections. Técnicamente, una List y un Set son Collections pero Map no (Map puede producir tres Collections: values (Collection de valores), key Set (Set de las llaves) y entry Set (Set de las entradas).

Si bien es posible que debido a esto el termino colecciones no sea el más acertado, es el nombre oficial. De hecho todas las clases de tipos de contenedores forman parte del “Java Collections Framework”.

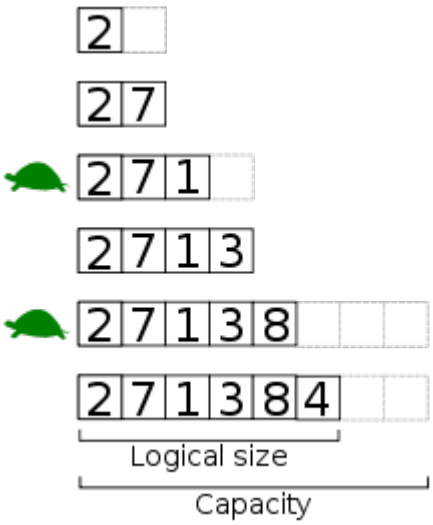
Y, el termino contenedores también podría llevar a confusiones porque existe la clase java.awt.Container (que no tiene nada que ver con las colecciones).

2 ¿Que colecciones conocemos?

ArrayList

Es un array dinámico de referencias a objetos. Cuando está lleno y queremos añadir otro objeto se crea un array de más tamaño y se copian las referencias. Estas inserciones son más lentas (las de la tortuga). Muchas veces si el array está muy vacío se copian las referencias a otro más pequeño.

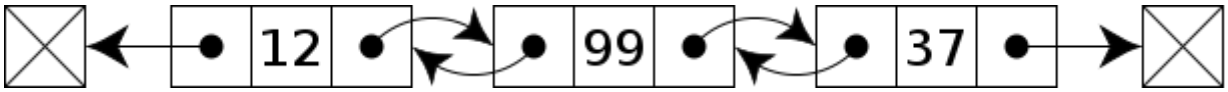
Al igual que en C, las operaciones de inserción e eliminación de un elemento son muy ineficientes. Por que hay que desplazar los elementos de detrás de la posición donde se realiza la inserción/eliminación.



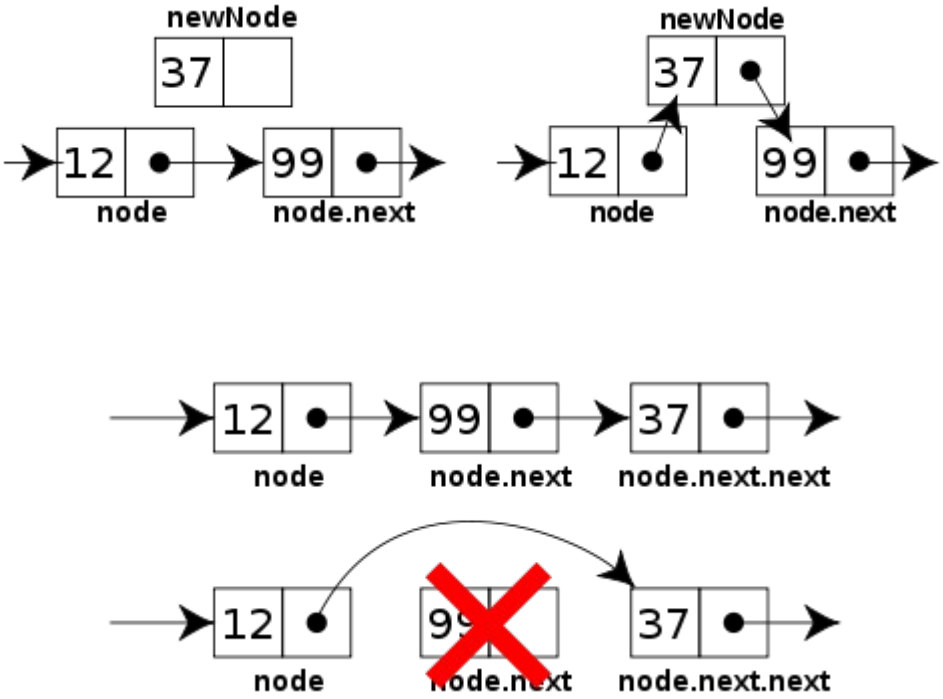
[https://en.wikipedia.org/wiki/Dynamic_array]

LinkedList

Es una doble linked list con referencias al primer y último elemento de la lista



Permite insertar e eliminar elementos de manera eficiente:



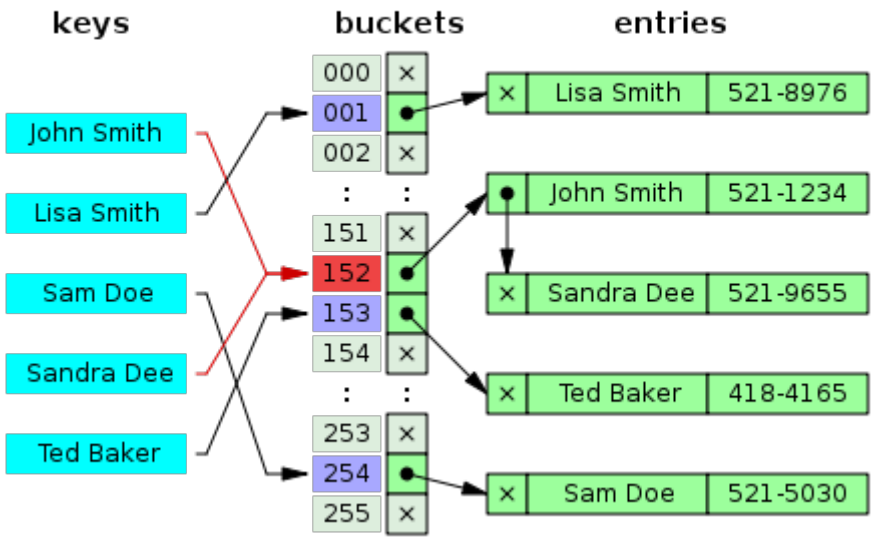
Pero acceder a un elemento en una posición 'n' es muy lento.

Incluye métodos como addFirst(), addLast(), getFirst(), getLast() que ArrayList no tiene.

[https://en.wikipedia.org/wiki/Linked_list]

HashSet

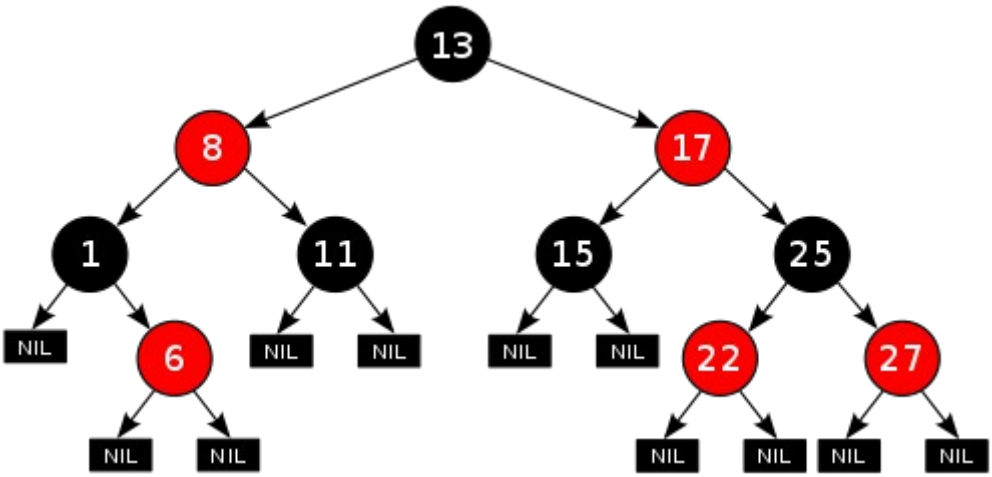
Es un conjunto de objetos donde no puede haber elementos repetidos que se implementa con una tabla de hash. Los accesos son muy rápidos.



[https://en.wikipedia.org/wiki/Hash_table]

TreeSet

Es un conjunto de objetos (no puede haber elementos repetidos) que se implementa con un árbol rojo-negro que mantiene los elementos siempre ordenados (los objetos tienen que ser comparables). Este tipo de árbol es un árbol binario de búsqueda equilibrado muy complejo pero muy eficiente.



[https://es.wikipedia.org/wiki/%C3%81rbol_rojo-negro]

HashMap

Es un mapa de keys (identificadores) a values (valores) donde las keys están en una tabla de hash.

TreeMap

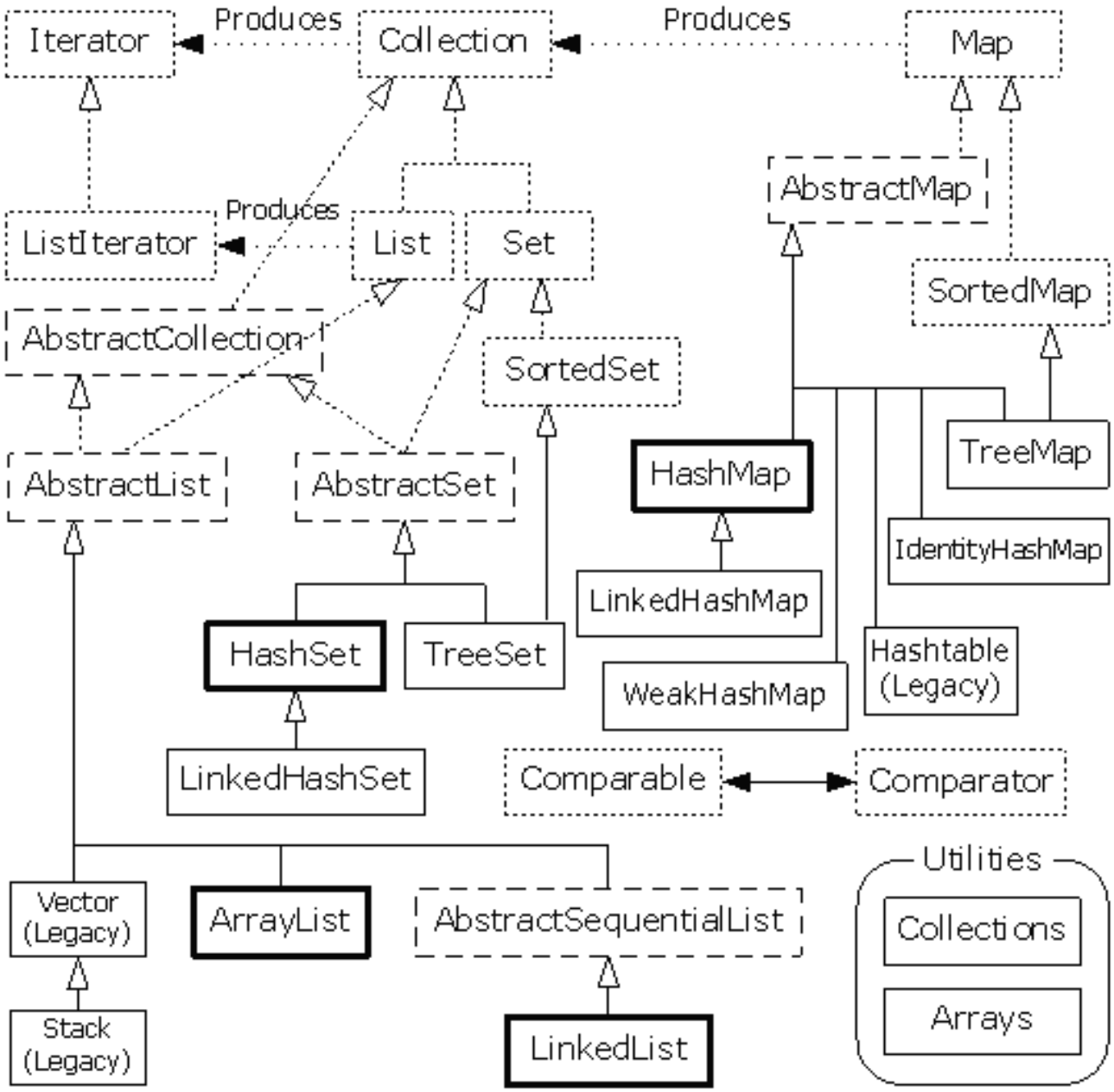
Es un mapa de keys (identificadores) a values (valores) donde las keys están en un árbol rojo-negro.

Las clases Collections y Arrays

Las clases Collections y Arrays son dos clases exclusivamente compuestas de atributos y métodos estáticos y que tienen un constructor privado vacío (para que no puedan ser instanciadas) con métodos para manipular colecciones y arrays respectivamente. De la misma manera cómo la clase Math nos permite operar con números.

Pero hay muchas más...

Taxonomía de colecciones (JDK 1.4):



[https://www.linuxtopia.org/online_books/programming_books/thinking_in_java/TIJ313_018.htm]

Como ahora Queue y Deque

Utilizar la interfaz Queue para colas (FIFO) y Deque para pilas (LIFO). La clase Stack es también para pilas pero se prefiere Deque. Queue y Deque son interfaces que se pueden implementar con LinkedList. Deque significa “double ended queue” is se suele pronunciar como “Deck”.

```
Queue<String> cola = new LinkedList<String>();
cola.add("Encolo");
String primero = cola.element();
int tamanyo = cola.size();
String desapilo = cola.remove();
```

Summary of Queue methods

	Throws exception	*Returns special value*
Insert	`add(e)`	`offer(e)`
Remove	`remove()`	`poll()`
Examine	`element()`	`peek()`

[https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/Queue.html]

```
Deque<String> pila = new LinkedList<String>();
pila.addFirst("Apilo");
String cima = pila.getFirst();
int tamanyo = pila.size();
String desapilo = pila.removeFirst();
```

Summary of Deque methods

	First Element (Head)		**Last Element (Tail)**	
	Throws exception	*Special value*	*Throws exception*	*Special value*
Insert	`addFirst(e)`	`offerFirst(e)`	`addLast(e)`	`offerLast(e)`
Remove	`removeFirst()`	`pollFirst()`	`removeLast()`	`pollLast()`
Examine	`getFirst()`	`peekFirst()`	`getLast()`	`peekLast()`

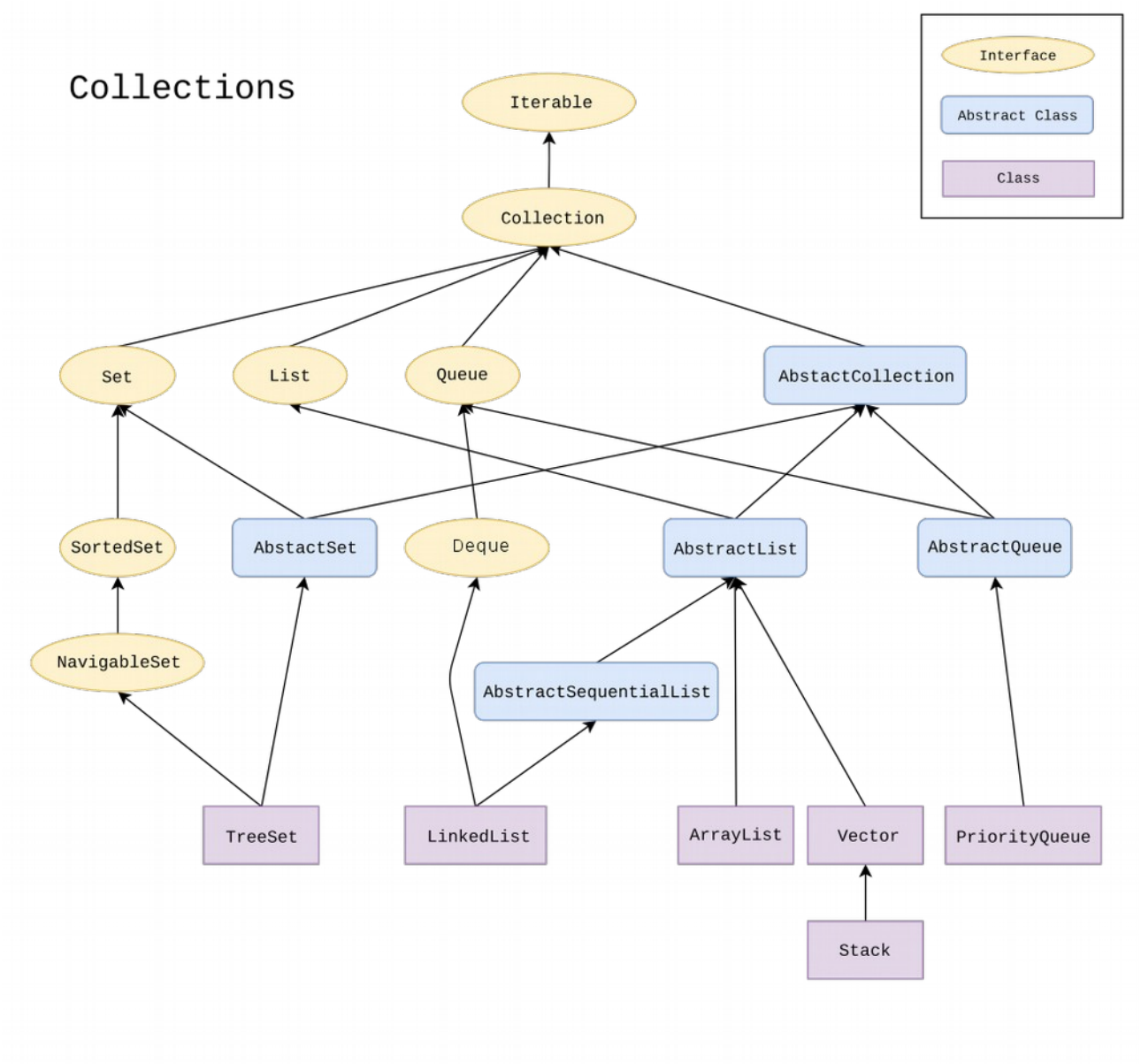
[https://docs.oracle.com/en/java/javase/12/docs/api/java.base/java/util/Deque.html]

Los métodos “examine” obtienen la cima de la pila o el primer/último elemento de la cola sin eliminarlo.

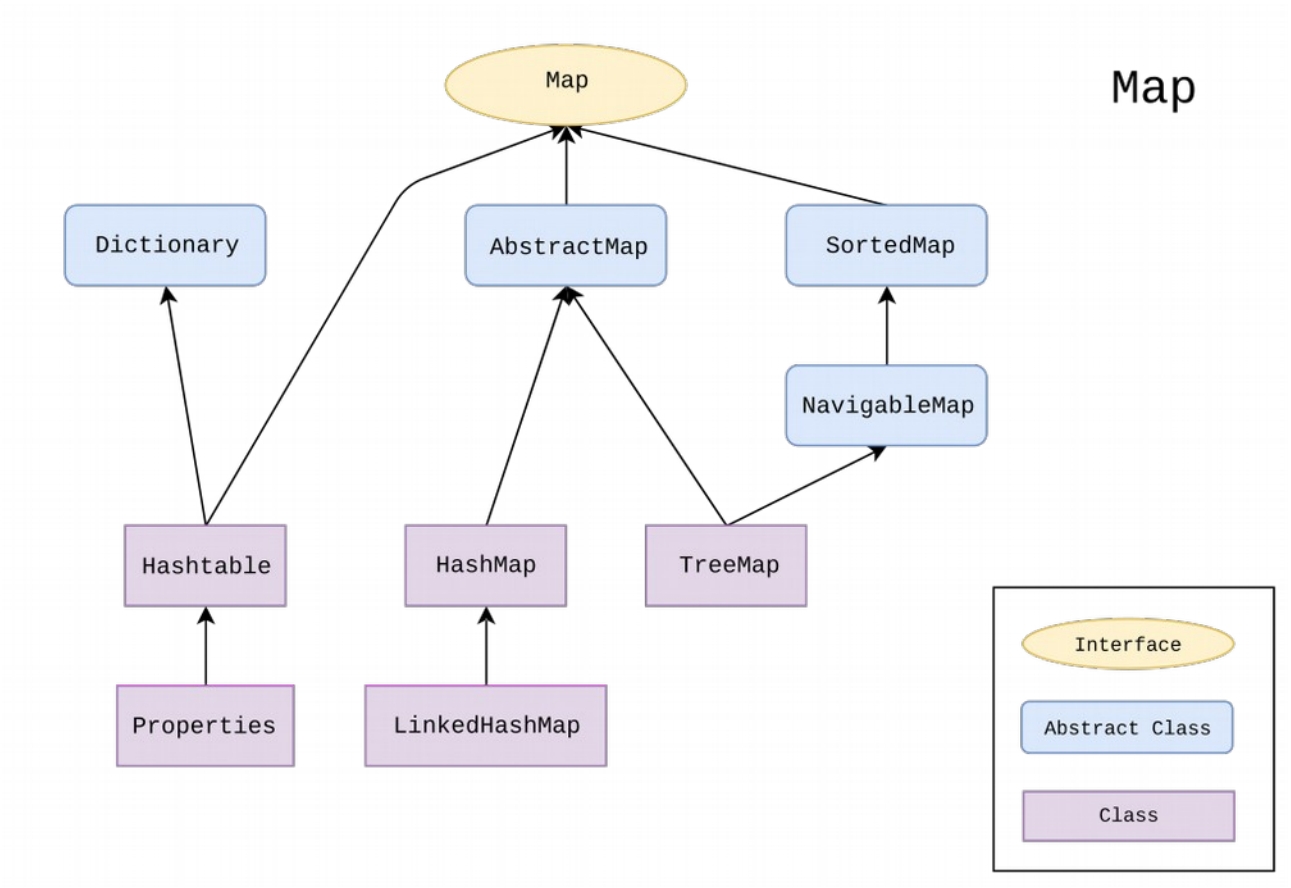
Los métodos para hacer “insert” que devuelven un valor especial devuelven false si no se pudo hacer el insert. El resto de métodos que devuelven un valor especial devuelven null si no se pudo hacer la operación.

Los métodos para hacer “insert” que lanzan una excepción no suelen fallar. El resto de métodos que lanzan una excepción lanzan NoSuchElementException si la colección estaba vacía.

Taxonomía actual java.util.Collection:



Taxonomía actual java.util.Map:



[https://en.wikipedia.org/wiki/Java_collections_framework]

3 ¿Cómo decidir que colección usar en cada caso?

1- Necesito que sean una secuencia de elementos (donde puedo hablar del elemento que hay en la posición 0, del que hay en la posición 1, etc): List

(List permite acceder a la posición 'i' de la lista con “get(i)”)

- Insertar / eliminar deben ser eficientes: LinkedList.
- Acceder a la posición 'n' debe ser eficiente: ArrayList.

2- Los elementos tienen key y (por tanto) no pueden haber keys repetidas: Map

- La key es comparable y querré tenerlos ordenados: TreeMap.
- De lo contrario: HashMap.

3- No puedo tener elementos repetidos: Set

- Los elementos son comparables y no quiero repetidos y querré tenerlos ordenados: TreeSet.
- De lo contrario: HashSet.
- Si tiene que permitirse elementos repetidos se tendrá que implementar como List aunque no nos importe en que posición se guarda cada elemento.

Noto que:

- Un Set no puede tener elementos repetidos (sea Tree o no).
- Un Map no puede tener keys repetidas (sea Tree o no).
- Un Set es preferible a una List si no va a contener repetidos y nos interesa ver si un elemento está en ese conjunto o no.
- La búsqueda de un elemento en una colección es más rápida en Set y Map (búsqueda en hash lists o trees) que en List (donde hay que recorrer la lista secuencialmente).

(Pueden haber casos donde dos colecciones o más sean igual de apropiadas, pero no suele suceder)

4 List: Robot jugando al solitario del dominó

Implementa un clase Ficha (de dominó) con su constructor y sus getters, con un toString() (que imprima el "seis-zero" como "`6:0|`" y con un método llamado "Ficha girarFicha()" que gire la ficha (el "`6:0|`" pasaría a ser "`0:6|`"). Implementa también el método "boolean esUnDoble()".

A) Ahora implementa con ArrayLists una clase que genere todas las fichas desde el doble 0 al doble MAX_NUM. Después, el programa, actuando como si fuera un robot jugando al solitario, empezará por tirar el doble más grande e irá tirando fichas (las jugadas tienen que ser legales) hasta que haya tirado todas sus fichas (de su "mano" a la "mesa) o ya no pueda tirar ninguna ficha más.

Tu ejecución puede ser diferente de las de los ejemplos dependiendo de cómo lo implementes.

```
run: (MAX_NUM = 2)
mano(5): 0:0|0:1|0:2|1:1|1:2|
mesa(1): 2:2|
mano(4): 0:0|0:1|1:1|1:2|
mesa(2): 0:2|2:2|
mano(3): 0:1|1:1|1:2|
mesa(3): 0:0|0:2|2:2|
mano(2): 1:1|1:2|
mesa(4): 1:0|0:0|0:2|2:2|
mano(1): 1:2|
mesa(5): 1:1|1:0|0:0|0:2|2:2|
mano(0):
mesa(6): 2:1|1:1|1:0|0:0|0:2|2:2|
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
run: (MAX_NUM = 5)
mano(20):  0:0|0:1|0:2|0:3|0:4|0:5|1:1|1:2|1:3|1:4|1:5|2:2|2:3|2:4|2:5|3:3|3:4|
3:5|4:4|4:5|
mesa(1): 5:5|
mano(19):  0:0|0:1|0:2|0:3|0:4|1:1|1:2|1:3|1:4|1:5|2:2|2:3|2:4|2:5|3:3|3:4|3:5|
4:4|4:5|
mesa(2): 0:5|5:5|
mano(18):  0:1|0:2|0:3|0:4|1:1|1:2|1:3|1:4|1:5|2:2|2:3|2:4|2:5|3:3|3:4|3:5|4:4|
4:5|
mesa(3): 0:0|0:5|5:5|
mano(17): 0:2|0:3|0:4|1:1|1:2|1:3|1:4|1:5|2:2|2:3|2:4|2:5|3:3|3:4|3:5|4:4|4:5|
mesa(4): 1:0|0:0|0:5|5:5|
mano(16): 0:2|0:3|0:4|1:2|1:3|1:4|1:5|2:2|2:3|2:4|2:5|3:3|3:4|3:5|4:4|4:5|
mesa(5): 1:1|1:0|0:0|0:5|5:5|
mano(15): 0:2|0:3|0:4|1:3|1:4|1:5|2:2|2:3|2:4|2:5|3:3|3:4|3:5|4:4|4:5|
mesa(6): 2:1|1:1|1:0|0:0|0:5|5:5|
mano(14): 0:3|0:4|1:3|1:4|1:5|2:2|2:3|2:4|2:5|3:3|3:4|3:5|4:4|4:5|
mesa(7): 0:2|2:1|1:1|1:0|0:0|0:5|5:5|
mano(13): 0:4|1:3|1:4|1:5|2:2|2:3|2:4|2:5|3:3|3:4|3:5|4:4|4:5|
mesa(8): 3:0|0:2|2:1|1:1|1:0|0:0|0:5|5:5|
mano(12): 0:4|1:4|1:5|2:2|2:3|2:4|2:5|3:3|3:4|3:5|4:4|4:5|
mesa(9): 1:3|3:0|0:2|2:1|1:1|1:0|0:0|0:5|5:5|
mano(11): 0:4|1:5|2:2|2:3|2:4|2:5|3:3|3:4|3:5|4:4|4:5|
mesa(10): 4:1|1:3|3:0|0:2|2:1|1:1|1:0|0:0|0:5|5:5|
mano(10): 1:5|2:2|2:3|2:4|2:5|3:3|3:4|3:5|4:4|4:5|
mesa(11): 0:4|4:1|1:3|3:0|0:2|2:1|1:1|1:0|0:0|0:5|5:5|
mano(9): 2:2|2:3|2:4|2:5|3:3|3:4|3:5|4:4|4:5|
mesa(12): 0:4|4:1|1:3|3:0|0:2|2:1|1:1|1:0|0:0|0:5|5:5|5:1|
BUILD SUCCESSFUL (total time: 0 seconds)
```

B) No obstante, como estamos todo el rato eliminando fichas de la “mano” y la mitad de las veces estás insertando fichas al principio de la “pila” sería más eficiente implementarlas con LinkedList.

Haz por tanto esta re-implementación con LinkedList.

Pista: Acuérdate que LinkedList incluye métodos como addFirst(), addLast(), getFirst(), getLast() que ArrayList no tiene.

Nota: Como en la “mano” no importa en que posición esté cada elemento, también podría ser un HashSet.

Pista: ¿Si te peta al eliminar una ficha de la “mano” con un java.util.ConcurrentModificationException que tienes que cambiar en como recorres la List y/o en como eliminas el elemento?

5 compareTo(), equals() y hashCode()

compareTo()

El método compareTo es el único miembro de la interfaz Comparable. Proporciona un medio para ordenar completamente los objetos.

Implementar Comparable permite:

- Invocar Collections.sort(), Collections.binarySearch(), Arrays.sort() y Arrays.binarySearch().
- Usar objetos de esa clase como claves en un TreeMap o como elementos en un TreeSet.

equals()

El operador “==” invoca de manera implícita el método equals() de la clase Object., que devuelve true si las dos referencias apuntan a la misma posición de memoria, y false en caso contrario. Si queremos modificar este comportamiento para una clase en concreto deberemos sobrescribir el método equals().

Los métodos equals y hashCode deben estar implementados de manera congruente:

- Si se sobrescribe uno se debe sobrescribir el otro.
- hashCode debe generar valores iguales para objetos iguales.
- equals y hashCode deben depender del mismo conjunto de campos. No es obligatorio utilizar todos los campos.

Los objetos colocados en una List, Set o Map deben tener una definición adecuada de equals.

hashCode()

El método hashCode es el utilizado en las collections HashSet y HashMap. Dos objetos para los cuales equals() retorna false podrían tener el mismo hash code. Pero, dos objetos para los cuales equals() retorna true tienen que tener el mismo hash code.

Una posible forma de implementar el método hashCode() es generar un String con la concatenación de todos los campos usados en equals() separados por tabulado y devolver el hashCode() de dicho String.

NetBeans

Recuerda que NetBeans puede generar automáticamente los métodos equals() y hashCode(). Y probablemente lo hará mejor que nosotros.

6 CompareTo: Ejercicios

A) Implementa una clase Ficha (ficha de dómino) con su constructor y sus métodos compareTo(), equals(), hashCode() y toString(). Después implementa una clase tester con su constructor y su toString() que genere una List de 6 fichas al azar y las imprima ordenadas ascendentemente y descendentemente de acuerdo a un "peso ponderado" (que consistirá en el número más alto de la ficha multiplicado por 6 más el otro número). Para eso implementa también el método pesoPonderado().

```
run:
[4:3|, 5:0|, 4:1|, 2:5|, 1:2|, 1:3|]
Sin ordenar:
fichas(6): 4:3|5:0|4:1|2:5|1:2|1:3|
sort():
fichas(6): 1:2|1:3|4:1|4:3|5:0|2:5|
reverseOrder:
fichas(6): 2:5|5:0|4:3|4:1|1:3|1:2|
BUILD SUCCESSFUL (total time: 0 seconds)
```

B) Implementa una clase Casa (o House) con atributos tipo (String) y tamaño (int) con su constructor y sus métodos compareTo(), equals(), hashCode() y toString(). Después implementa una clase tester que primero añada unas cuantas casas en una List y las imprima: sin ordenar, ordenadas y en orden inverso y luego cree una nueva casa y mire si está en la lista. Las casas cuando se imprimen, imprimen sólo su tipo y cuando se comparan se comparan sólo por su tamaño.

Pista: Mírate los métodos sort() y contains() de la clase Collections.

```
A la derecha, ejemplo de ejecución para una List con las siguientes House:

House("medium", 200)
House("small", 100)
House("large", 300)

y la casa:

House("nice house", 200)
```

```
run:
Sin ordenar:
[medium, small, large]
Ordenadas:
[small, medium, large]
Ordenadas por orden inverso:
[large, medium, small]
A house that is not on the list: nice house
"nice house" has the same size that one of the houses on the list.
BUILD SUCCESSFUL (total time: 0 seconds)
```

7 Sets: Conjuntos de enteros

Implementa un programa que cree dos conjuntos no ordenados de enteros a partir de dos arrays de ints y calcule su unión, intersección, diferencia (resta) y diferencia simétrica.

Después de que los ordene, los muestre ordenados, y muestre sus primeros y últimos elementos.

```
run:
A: [1, 2, 3, 6, 7]
B: [0, -1, 2, -3, 4]
Intersection A y B:   [2]
Union A y B       :   [0, -1, 1, 2, -3, 3, 4, 6, 7]
Substraction A - B:   [1, 3, 6, 7]
Symmetric difference: [0, -1, 1, -3, 3, 4, 6, 7]
The sorted sets are:
A: [1, 2, 3, 6, 7]
B: [-3, -1, 0, 2, 4]
The first element of the sorted set A is: 1
The last element of the sorted set A is: 7
The first element of the sorted set B is: -3
The last element of the sorted set B is: 4
BUILD SUCCESSFUL (total time: 0 seconds)
```

8 Maps: Mapas de personas

Implementa una clase Persona (o Person) con atributos nombre y apellido con su constructor, sus getters y su toString. Después implementa una clase tester que añada unas cuantas personas en un HashMap<String, Person> (donde la key es el NIF de la persona).

Esta clase tester deberá tener métodos que impriman: las keys, los values y las entries del mapa.

Y comprobar que:

- Si intentas obtener un valor con una llave que no existe devuelve null.
- Si intentas añadir una entrada <llave, valor> con una llave ya existente, esta entrada sobrescribe la pre-existente.
- Si generamos un TreeMap a partir del HashMap sus entradas estarán ordenadas por key.

```
run:
El mapa está vacio
Intentando obtener Person 12345678Z...
      Person 12345678Z: null
Insertando 3 personas
Intentando obtener Person 12345678Z...
      Person 12345678Z: Guerrero, Manel
Keys:
      Key : 12345678Z
      Key : 12341234B
      Key : 12341234A
Values:
      Value : Guerrero, Manel
      Value : Garcia, Pere
      Value : Guerrero, Pere
HashMap:
      Key : 12345678Z Value : Guerrero, Manel
      Key : 12341234B Value : Garcia, Pere
      Key : 12341234A Value : Guerrero, Pere
TreeMap (ordenat per Key):
      Key : 12341234A Value : Guerrero, Pere
      Key : 12341234B Value : Garcia, Pere
      Key : 12345678Z Value : Guerrero, Manel
BUILD SUCCESSFUL (total time: 0 seconds)
```

9 Pilas: Las Torres de Hanói

Implementa el juego de las Torres de Hanói con la interfaz Deque implementada con una LinkedList. El programa debe controlar que no introduces un número de torre equivocado y que no intentas desapilar de una torre vacía. También hay que controlar que las torres siempre están ordenadas (puedes tener esta torre [3, 1], pero no esta [1, 2]).

Nota: Tendrás dos clases: la clase Torre y la clase TorresDeHanoi (con el main que se construirá a si mismo).

Nota: El toString() de la clase Torre devuelve String del tipo “[3,2,1]” y no del tipo “[3,2,1,]”. Y, si está vacía retorna “[]” y no “[”.

Pista: Fíjate que la pila la imprimimos “del revés” (lo último que imprimimos es la cima de la pila que es el primer elemento de la List). Esto se consigue iterando de fin a principio con el método descendingIterator() de la clase Iterator.

Pista: El método para determinar si el juego se ha resuelto es más fácil de lo que crees.

run: Torres de Hanoi Con cuantos discos [3-8]? **3** 0: [3, 2, 1] 1: [] 2: [] De torre a torre (ej:"0 1") **2 3** De torre a torre (ej:"0 1") **0 2** 0: [3, 2] 1: [] 2: [1] De torre a torre (ej:"0 1") **0 1** 0: [3] 1: [2] 2: [1]	De torre a torre (ej:"0 1") **2 1** 0: [3] 1: [2, 1] 2: [] De torre a torre (ej:"0 1") **2 1** No puedes desapilar de una torre vacía. 0: [3] 1: [2, 1] 2: [] De torre a torre (ej:"0 1") **0 2** 0: [] 1: [2, 1] 2: [3]	De torre a torre (ej:"0 1") **1 0** 0: [1] 1: [2] 2: [3] De torre a torre (ej:"0 1") **1 2** 0: [1] 1: [] 2: [3, 2] De torre a torre (ej:"0 1") **0 2** 0: [] 1: [] 2: [3, 2, 1] SOLUCIONADO!!! :-) BUILD SUCCESSFUL (total time: 56 seconds)

10 Colas: A la cola

Copia las clases de las Torres de Hanói a otro proyecto y modifícalas para que ahora sean 3 colas y desencoles un elemento de una de las colas y lo encoles a otra cola (o a la misma). En este caso el programa no finaliza nunca.

Las colas seguirán siendo implementadas como LinkedList. Pero ahora la interfaz será Queue.

```
run:
A la cola

Con cuantos nums [3-8]? 4
0: [1, 2, 3, 4]
1: []
2: []

De que cola a que cola (ej:"0 1")
0 0
0: [2, 3, 4, 1]
1: []
2: []

De que cola a que cola (ej:"0 1")
0 1
0: [3, 4, 1]
1: [2]
2: []
```

```
De que cola a que cola (ej:"0 1")
0 2
0: [4, 1]
1: [2]
2: [3]

De que cola a que cola (ej:"0 1")
1 2
0: [4, 1]
1: []
2: [3, 2]

De que cola a que cola (ej:"0 1")
2 0
0: [4, 1, 3]
1: []
2: [2]

De que cola a que cola (ej:"0 1")
```

4 Herencia

1 Personas, profesores y alumnos

Crea una clase Persona (con nombre y apellidos como atributos) y dos clases que hereden de Persona: Profesor (con atributo adicional String despacho) y Alumno (con atributo adicional String cuatrimestre). (Por simplicidad, para nuestro ejercicio asumimos que un alumno sólo está matriculado de asignaturas de un mismo cuatrimestre.)

Las tres clases sobrescribirán el método toString() de tal forma que las clases hijas invoquen el método de la clase madre. El toString() de Persona mostrará “apellido, nombre”, el de Alumno lo mismo seguido del cuatrimestre (por ejemplo “ (Semester: Q1B)” y el de Profesor seguido de su despacho (por ejemplo “ (Office: D6-212)”.

Las tres clases tendrán como mínimo el constructor y el toString(), pero podéis añadir todos los métodos que queráis.

Finalmente, implementad un clase tester que cree e imprima objetos de las tres clases, para que se vea que en cada caso se invoca el toString que corresponde.

```
run:
Garcia Perez, Pepe
Garcia, Pere (Semester: Q1B)
Manel Guerrero (Office: D6-212)
Manel Guerrero (Office: D6-212)
BUILD SUCCESSFUL (total time: 0 seconds)
```

2 Puntos

Cread la clase Punto (con atributos x e y) con los siguientes métodos (a parte de constructor, getters y setters):

- public double distanceToZero() // distancia entre this y (0, 0)
- public double distance(Point other) // distancia entre this y other
- public double squaredDistanceToZero() // sirve para comparar distancias
- public String toString() // “(x,y)

Cread la clase Punto3D que herede de Punto y tenga como atributo la z con los mismos métodos (sobrescribiendo los que tenga que sobrescribir y invocando los métodos de la clase madre cuando sea posible para que los métodos de la clase hija re-aprovechen código).

Implementa un tester que las pruebe.

```
run:
Point #1: (3.0,2.0)
Point #2: (-1.0,-1.0)
Distancia entre p1 i p2: 5.0
Point3D #1: (3.0,2.0,0.0)
Point3D #2: (-1.0,-1.0,-1.0)
Distancia entre p3D1 i p3D2: 5.0990195135927845
Distancia entre p3D1 i p2: 5.0
(3.0,2.0,0.0) esta mas lejos de zero que (-1.0,-1.0)
BUILD SUCCESSFUL (total time: 0 seconds)
```

5 Polimorfismo

1 Figuras

A) Vamos a crear una clase abstracta Figura con un atributo final de tipo String llamado “tipo” (que podrá ser “Triangulo”, “Circulo”, “Rectangulo”, etcétera. A parte del constructor y del toString() tendrá dos métodos abstractos: double calculateArea() y double calculatePerimeter() que devolverán el área y el perímetro de la figura. Crearemos también la clase punto bidimensional.

Crearemos tres clases hijas de Figura: Triangulo (construido a través de 3 puntos), Circulo (construido a través de 1 punto y 1 radio) y Rectangulo (construido a través de 2 puntos).

Crearemos una clase tester que compruebe que funcionan correctamente.

```
run:
Point 1: (3.0,2.0)
Point 2: (-1.0,-1.0)
Point 3: (0.0,0.0)
Radius: 4
Rectangle (p1,p2): Rectangle: ((3.0,2.0),(-1.0,-1.0))
Area: 12.0
Perimeter: 14.0
Circle (p1, radius): Circle: ((3.0,2.0),4)
Area: 50.26548245743669
Perimeter: 25.132741228718345
Triangle (p1, p2, p3): Triangle: ((3.0,2.0),(-1.0,-1.0),(0.0,0.0))
Area: 0.5
Perimeter: 10.019764837837084
BUILD SUCCESSFUL (total time: 0 seconds)
```

B) Crearemos un segundo tester con un array de figuras. En cada posición del array crearemos un nuevo Triangulo, Circulo o Rectangulo. Luego tendremos un bucle que iterará por las posiciones del array. Nos imprimirá la figura, su área, su perímetro, si es una instancia de Circulo, si es una instancia de Figura, el nombre de la clase y (en el caso que sea un Triangulo) las distancias entre sus puntos (la medida de sus tres lados).

Fijaros que getClass() devuelve nombreDelPackage.NombreClase e instanceof sólo el NombreClase.

```
run:
Figure 0: Rectangle: ((3.0,2.0),(-1.0,-1.0))
Area: 12.0
Perimeter: 14.0
instanceof Circle? false
instanceof Figure? true
getClass() returns: "class W07H02_Polymorphism.Rectangle"
=====================================
Figure 1: Circle: ((3.0,2.0),4)
Area: 50.26548245743669
Perimeter: 25.132741228718345
instanceof Circle? true
instanceof Figure? true
```

```
getClass() returns: "class W07H02_Polymorphism.Circle"
=======================================
Figure 2: Triangle: ((3.0,2.0),(-1.0,-1.0),(0.0,0.0))
Area: 0.5
Perimeter: 10.019764837837084
instanceof Circle? false
instanceof Figure? true
getClass() returns: "class W07H02_Polymorphism.Triangle"
Distance from p1 to p2: 5.0
Distance from p2 to p3: 1.4142135623730951
Distance from p3 to p1: 3.605551275463989
=======================================
BUILD SUCCESSFUL (total time: 0 seconds)
```

2 Piedra, papel, tijeras

Implementad el juego piedra, papel, tijeras donde la clase abstracta jugador tiene dos clases hijas: humano y máquina.

```
run:
Human player's name?
Manel
How many points to get the game?
3
Form? [r]ock, [p]aper or [s]cissors
r
Hand ready!
Hand ready!
He plays paper
You loose! :-(
Manel 0 points.
Robot 1 points.

Form? [r]ock, [p]aper or [s]cissors
r
Hand ready!
Hand ready!
He plays rock
It's a tie! :-/
Manel 0 points.
Robot 1 points.

Form? [r]ock, [p]aper or [s]cissors
r
Hand ready!
Hand ready!
He plays paper
You loose! :-(
Manel 0 points.
Robot 2 points.
```

```
Form? [r]ock, [p]aper or [s]cissors
s
Hand ready!
Hand ready!
He plays scissors
It's a tie! :-/
Manel 0 points.
Robot 2 points.

Form? [r]ock, [p]aper or [s]cissors
s
Hand ready!
Hand ready!
He plays scissors
It's a tie! :-/
Manel 0 points.
Robot 2 points.

Form? [r]ock, [p]aper or [s]cissors
s
Hand ready!
Hand ready!
He plays rock
You loose! :-(
Manel 0 points.
Robot 3 points.

Robot WINS THE GAME!!!
BUILD SUCCESSFUL (total time: 26
seconds)
```

6 Excepciones

1 Cuenta corriente del banco

A) Crearemos la clase InsufficientFundsException (no tienes suficiente dinero en la cuenta), la clase CheckingAccount (cuenta corriente) y una clase tester. En la clase CheckingAccount tendremos

- Un método toString() que mostrará el número de cuenta y el saldo.
- Un constructor al que le pasaremos el número de cuenta y nos pondrá el saldo a cero.
- Un método para ingresar dinero y otro para extraer dinero.

Lanzaremos la excepción InsufficientFundsException cuando sea debido. Y, la clase tester la capturará e imprimirá “Sorry, but you don't have that much money”.

```
run:
Account number: 101, Balance: 0.0

Depositing $500...
Account number: 101, Balance: 500.0

Withdrawing $100...
Account number: 101, Balance: 400.0

Withdrawing $600...
Sorry, but you don't have that much money
BUILD SUCCESSFUL (total time: 0 seconds)
```

B) Haced una nueva versión donde la clase InsufficientFundsException tenga un atributo que sea cuantos Euros le falta para poder retirar el dinero que ha pedido.

Para ello, al constructor se le tendrá que pasar dicho valor y tendrá que tener un setter que devuelva el valor de dicho atributo.

Esto es para que interioricemos que la Exceptions son clases y, como tales, pueden tener los atributos y métodos que consideremos necesarios.

```
run:
Account number: 101, Balance: 0.0

Depositing $500...
Account number: 101, Balance: 500.0

Withdrawing $100...
Account number: 101, Balance: 400.0

Withdrawing $600...
Sorry, but you are short $200.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

7 Ficheros

1 BufferedReader y PrintWriter

Para escribir a un fichero construiremos un FileWriter con el String del nombre del fichero y, con ese Writer, construiremos un PrintWriter.

FileWriter:
Constructor: FileWriter(String fileName, boolean append)
Los métodos de FileWriter throws IOException

PrintWriter:
Constructor: PrintWriter(Writer out, boolean autoFlush)
Los métodos de PrintWriter throws FileNotFoundException

Una vez creado el PrintWriter podemos escribir en el fichero con los métodos print() y println() sobre el PrintWriter. Y para cerrar el fichero podemos invocar el método close() (también sobre el PrintWriter).

Para leer de fichero construiremos un FileReader con el String del nombre del fichero y, con ese Reader, construiremos un BufferedReader.

FileReader:
Constructor: FileReader(String fileName)
Los métodos de FileReader throws FileNotFoundException

BufferedReader:
Constructor: BufferedReader(Reader in)
Los métodos de BufferedReader throws IOException

Una vez creado el BufferedReader podemos leer del fichero con los siguientes métodos:

int	read()	Reads a single character.
String	readLine()	Reads a line of text.

Y, para cerrar el fichero:

void	close()	Closes the stream and releases any system resources associated with it.

2 Scanner y PrintWriter

Pero hay una manera más fácil de leer y escribir ficheros. Construyendo un Scanner sobre un new File construido con el String del nombre de fichero a leer y construyendo un PrintWriter directamente con el String del nombre del fichero a escribir.

Aquí tienes un ejemplo que lee las líneas de un fichero y las imprime por pantalla y luego escribe en otro fichero 10 números al azar:

Files.java:

```
package p2_Scanner_y_PrintWriter;

import java.io.File;
import java.util.Scanner;
import java.io.PrintWriter;

public class Files {

    public static void main(String[] args) throws Exception {
        Scanner in = new Scanner(new File("Files_entrada.txt"));
        PrintWriter out = new PrintWriter("Files_salida.txt");
        for (int line = 1; in.hasNext(); line++) {
            System.out.println(line + ": " + in.nextLine());
        }
        in.close();
        for (int line = 1; line <= 10; line++) {
            out.println(line + ": " + Math.random());
        }
        out.close();
    }
}
```

3 wc

Haz un programa que dado un fichero de entrada lo lea y escriba en un fichero lo que retornaría el comando “wc” de Linux si lo invocáramos sobre ese fichero de entrada (es decir: el número de líneas, número de palabras, número de caracteres y el nombre de dicho fichero).

Ejemplo: 3 8 60 fichero_entrada.txt

(Puedes hacer dos versiones utilizando las dos maneras que sabes de leer y escribir en un fichero.)

8 Interfaces

Vosotros ya habéis usado interfaces de la API de Java:

- Comparable: Permite comparar dos objetos con el método compareTo().
- List: Interfaz que implementan ArrayList y LinkedList.
- Set: Interfaz que implementan HashSet y TreeSet.
- Map: Interfaz que implementan HashMap y TreeMap.

1 Comparable

Comparable es una interfaz que sólo obliga a implementar el método compareTo() a todas las clases que lo implementen:

```
public interface Comparable<T> {

    /**
     [...]
     * @param   o the object to be compared.
     * @return  a negative integer, zero, or a positive integer as this object
     *          is less than, equal to, or greater than the specified object.
     *
     * @throws NullPointerException if the specified object is null
     * @throws ClassCastException if the specified object's type prevents it
     *         from being compared to this object.
     */
    public int compareTo(T o);

}
```

2 List, Set y Map

El hecho de que List, Set y Map sean interfaces y no clases de las que heredan las seis colecciones que vemos en la asignatura puede ser anti-intuitivo.

Si cogemos como ejemplo ArrayList, esta clase implementa la interfaz List (que dice que métodos todos las clases que implementen List deben implementar) pero hereda de AbstractList (que da un principio de implementación esquelética con métodos que definen el comportamiento por defecto de ciertos métodos).

Por ejemplo, el método add() de ArrayList:

Se declara en la interfaz List.java:

```
public interface List<E> extends Collection<E> {
    [...]
    boolean add(E e);
    [...]
}
```

Se le da una implementación por defecto en AbstractList.java:

```
public abstract class AbstractList<E> extends AbstractCollection<E> implements
List<E> {
{
    [...]
    public boolean add(E e) {
        add(size(), e);
        return true;
    }
    [...]
}
```

Y se sobrescribe en ArrayList.java (con una implementación probablemente más eficiente y especifica para este tipo de List):

```
public class ArrayList<E> extends AbstractList<E>
        implements List<E>, [...]
{
    [...]
    public boolean add(E e) {
        modCount++;
        add(e, elementData, size);
        return true;
    }
    [...]
}
```

Nota: Estos códigos fuente son del openjdk-11. Para instalarlos en vuestro ordenador (si tenéis una distribución de Linux que use apt como sistema de paquetes) podéis hacer:

```
sudo apt install openjdk-11-source
cd /usr/lib/jvm/openjdk-11/lib
sudo unzip src.zip
cd java.base/java
```

Ver como están implementadas las clases que más utilizáis os puede dar muchas ideas. Especialmente porque los comentarios que hay en los códigos fuente explican el porque de como han sido implementadas.

Para bajaros la versión del openjdk-8 podéis hacer:

```
sudo apt install openjdk-8-source
cd /usr/lib/jvm/openjdk-8/lib
sudo unzip src.zip
cd java
```

Para ver la versión de Oracle-JDK-8 podéis ir a la siguiente URL:

```
https://zgrepcode.com/java/oracle/jdk-8u181/java
```

3 Diferencias entre clases abstractas e interfaces

Las diferencias principales entre clases abstractas e interfaces son:

- A pesar de que típicamente todos los métodos de una interfaz serán abstractos, a partir de la versión 8 del JDK es posible tener métodos no abstractos (a veces referidos como “default methods” o métodos por defecto) que podrán ser sobrescritos por las clases que implementen dicho interfaz.
- A pesar de que típicamente no tendremos métodos estáticos, a partir de la versión 8 del JDK también están permitidos.
- (Hay que ir con cuidado porque la mayoría de las páginas web siguen diciendo cosas que dejaron de ser ciertas en versiones posteriores a la 7.)
- Una clase puede implementar varios interfaces pero sólo heredar de una clase (sea abstracta o no).
- Una interfaz sólo puede tener atributos “static final” (no hace falta escribir).
- Una interfaz no tiene modificadores de acceso (public, friendly, protected o private). Todo (atributos y métodos) se asume que es public, (pero no se escribe la palabra “public” en el código). Aunque, a partir la versión 9 del JDK, una interfaz también puede incluir métodos privados.
- Una interfaz no puede declarar constructores.
- En una clase abstracta es obligatorio usar la palabra reservada “abstract” para los métodos abstractos. En una interfaz, no.
- Trabajar con una clase que hereda de una clase abstracta es más rápido que con una que implementa una interfaz.

Los interfaces tradicionales sólo tendrán atributos static final y métodos abstract. Pero esto se ha ido flexibilizando en sucesivas versiones de Java. Aquí podéis ver como ha ido evolucionando:

Java 7 permite sólo	• Atributos static final • Métodos abstract
Java 8 añade	• Métodos dinámicos (no abstract) (a veces referidos como “default methods”) • Métodos static
Java 9 añade	• Métodos private • Métodos private static

4 Tus propios interfaces

Tu puedes crear tus propios interfaces con códigos parecidos al de comparable, donde simplemente tienes una secuencia de métodos que todas las clases que implementen una interfaz deben implementar.

5 Invertible

Implementa en un proyecto de NetBeans las clases Vector (con las componentes double x e y) y Fraction (con los atributos enteros numerador y denominador) que implementen la interfaz Invertible con el método invert() (Interfaz que también deberás implementar).

Tanto en Vector cómo en Fraction, invert() intercambia los dos atributos de la clase.

Implementa también una clase tester con un main() que cree una fracción y un vector, los imprima por pantalla, los invierta y los imprima invertidos.

Haremos dos versiones en dos proyectos NetBeans diferentes:

A) invert() devuelve un nuevo objeto invertido (no modifica el original).

```
T invert();
```

B) invert() no devuelve nada (invierte el “this”).

```
void invert();
```

Nota: No podríamos hacer la versión:

```
static void invert(T o);
```

Porqué nos daría el siguiente error: “Non-static type variable T cannot be referenced from static context”. Pero a parte, un método no puede ser static y abstract a la vez. Por tanto tampoco podríamos hacer:

```
   static abstract void invert(Object o);
```

Porqué nos daría el siguiente error: “Illegal combinationof modifiers: abstract and static”.

Resultado de la ejecución:

```
run:
Vector: (3.0, 2.0)
Vector invertit: (-3.0, -2.0)
Fracció: 1/2
Fracció invertida: 2/1
BUILD SUCCESSFUL (total time: 0 seconds)
```

9 Grafos

A) Haz un programa que permita crear un grafo vacío o al azar y que una, vez creado, permita añadir o eliminar nodos y aristas (que después de crear o modificar imprima la lista de nodos adyacentes) e imprimir secuencia de nodos resultantes de recorrer el grafo en DFS o BFS separados por comas y con la distancia a la raíz entre paréntesis a partir de un nodo dado.

```
run:
Grafo vacío

help: Muestra la ayuda del programa.
comando: help
gv:         Genera grafo Vacío.
ga nn na:   Genera al Azar un grafo con nn nodos y na aristas.
an:         Añade Nodo.
en in:      Elimina Nodo con id in.
aa n1 n2:   Añade Arista que conecte n1 con n2.
ea n1 n2:   Elimina Arista que conecta n1 con n2.
id in:      Imprime el árbol DFS desde el nodo con id in.
ib in:      Imprime el árbol BFS desde el nodo con id in.
help:       Muestra esta ayuda.
quit:       Sale del programa.

Grafo vacío

help: Muestra la ayuda del programa.
comando: ga 20
Número de argumentos erróneo
Grafo vacío

help: Muestra la ayuda del programa.
comando: ga 20 60
1: 9
2: 1, 4, 11, 18
3: 6, 16
4: 3, 8, 9, 11, 18
5: 3, 8, 16
6: 3, 13, 18, 19, 20
7: 1, 5
8: 3, 9, 12, 14, 17
9: 14, 18
10: 4, 6, 8
11: 6
12: 2, 4, 15, 17, 19
13: 1, 3, 4, 11
14: 8, 19
15: 17, 20
16: 1, 11, 14
17: 7
18: 3, 7, 9, 10, 12, 14
19: 12, 13
20: 15, 18

help: Muestra la ayuda del programa.
comando: id 2
2(0),1(1),9(2),14(3),8(4),3(5),6(6),13(7),4(8),11(9),18(9),7(10),5(11),16(12),1
0(10),12(10),15(11),17(12),20(12),19(11).
```

```
help: Muestra la ayuda del programa.
comando: ib 2
2(0),1(1),9(2),4(1),3(2),16(3),8(2),17(3),11(1),6(2),13(3),19(3),20(3),18(1),7(
2),5(3),10(2),12(2),15(3),14(2).

help: Muestra la ayuda del programa.
comando: salir
Comando inexistente

help: Muestra la ayuda del programa.
comando: quit
BUILD SUCCESSFUL (total time: 11 minutes 47 seconds)
```

B) Una segunda versión podría imprimir el árbol generador correspondiente a recorrer el grafo en DFS y BFS. Aunque es relativamente complicado:

```
comando: id 2
    2--> 1--> 9
     \-> 4--> 3-->16
     ·    \-> 8-->17
     \->11--> 6-->13
     ·    ·    \->19
     ·    ·    \->20
     \->18--> 7--> 5
     ·    \->10
     ·    \->12-->15
     ·    \->14

help: Muestra la ayuda del programa.
comando: ib 2
    1--> 9-->14--> 8-->17
     ·    ·    \->19-->13-->11
     ·    \->18--> 3--> 6-->20
     ·    ·    ·    \->16
     ·    ·    \-> 7--> 5
     ·    ·    \->10--> 4
     ·    ·    \->12--> 2
     ·    ·    ·    \->15
```

C) Y si queréis, podríais hacer una versión que primero guardara en una List<StringBuilder> los árboles tal y como se muestran en el apartado anterior y los modificara para poder imprimir los árboles así:

```
comando: id 2
    2--> 1--> 9
     \-> 4--> 3-->16
     |    \-> 8-->17
     \->11--> 6-->13
     |         \->19
     |         \->20
     \->18--> 7--> 5
          \->10
          \->12-->15
          \->14

help: Muestra la ayuda del programa.
comando: ib 2
    1--> 9-->14--> 8-->17
          |    \->19-->13-->11
          \->18--> 3--> 6-->20
               |    \->16
               \-> 7--> 5
               \->10--> 4
               \->12--> 2
                    \->15
```

D) Podéis ampliar el programa para pueda leer y escribir en ficheros los grafos que generéis con los comandos (utilizando el mismo formato con el que imprime la lista de nodos adyacentes por pantalla):

```
lg nombreFichero.txt:        Lee el grafo del fichero.
eg nombreFichero.txt:        Escribe el grafo en el fichero.
```

E) Y, a partir de aquí, podríais ir añadiendo métodos que hagan todo lo que sabéis hacer con grafos:

- Ver si dos nodos son adyacentes.
- Calcular la distancia entre dos nodos.
- Ver si un grafo es un ciclo.
- Ver si un grafo es un árbol (o un bosque).
- Ver si un grafo es completo.
- Ver si dos caminos (cada uno de ellos determinado por el camino más corto entre dos nodos) son independientes (es decir, si no tienen ningún vértice en común excepto el primero y el último).

  Nivel avanzado:
- Ver si un grafo es Euleriano o semi-Euleriano y cual es el ciclo o camino Euleriano (pasa por todas las aristas una y sólo una vez).
- Ver si un grafo tiene un ciclo o camino Hamiltoniano (que pasa por todos los vertices una y sólo una vez) y mostrarlo.

https://es.wikipedia.org/wiki/Teor%C3%Ada_de_grafos
https://es.wikipedia.org/wiki/Grafo_ciclo
https://es.wikipedia.org/wiki/Grafo_completo
https://es.wikipedia.org/wiki/Ciclo_euleriano
https://es.wikipedia.org/wiki/Camino_hamiltoniano

10 Poniéndolo todo a la práctica

En este apartado tenéis lo que fueron controles de laboratorio de cuatrimestres anteriores.

Sirve para practicar como se traducen los diagramas UML a código Java (atributos, métodos, relaciones que se convierten en referencias a un objeto, a un array o a una colección de objetos).

Cada enunciado tiene el diagrama de clases, una pequeña explicación y la salida que tiene que generar por pantalla.

Junto con el enunciado tenéis el fichero de PlantUML (que es la herramienta que usamos para generar el diagrama UML). Y así, si queréis podéis aprender a hacer los vuestros.

Si hacéis copiar y pegar del texto de PlantUML en la web de PlantUML y le dais al botón de [Submit] os los generará.

Web para generar los diagramas:

http://www.plantuml.com/plantuml/uml

Web que explica la sintaxis de PlantUML:

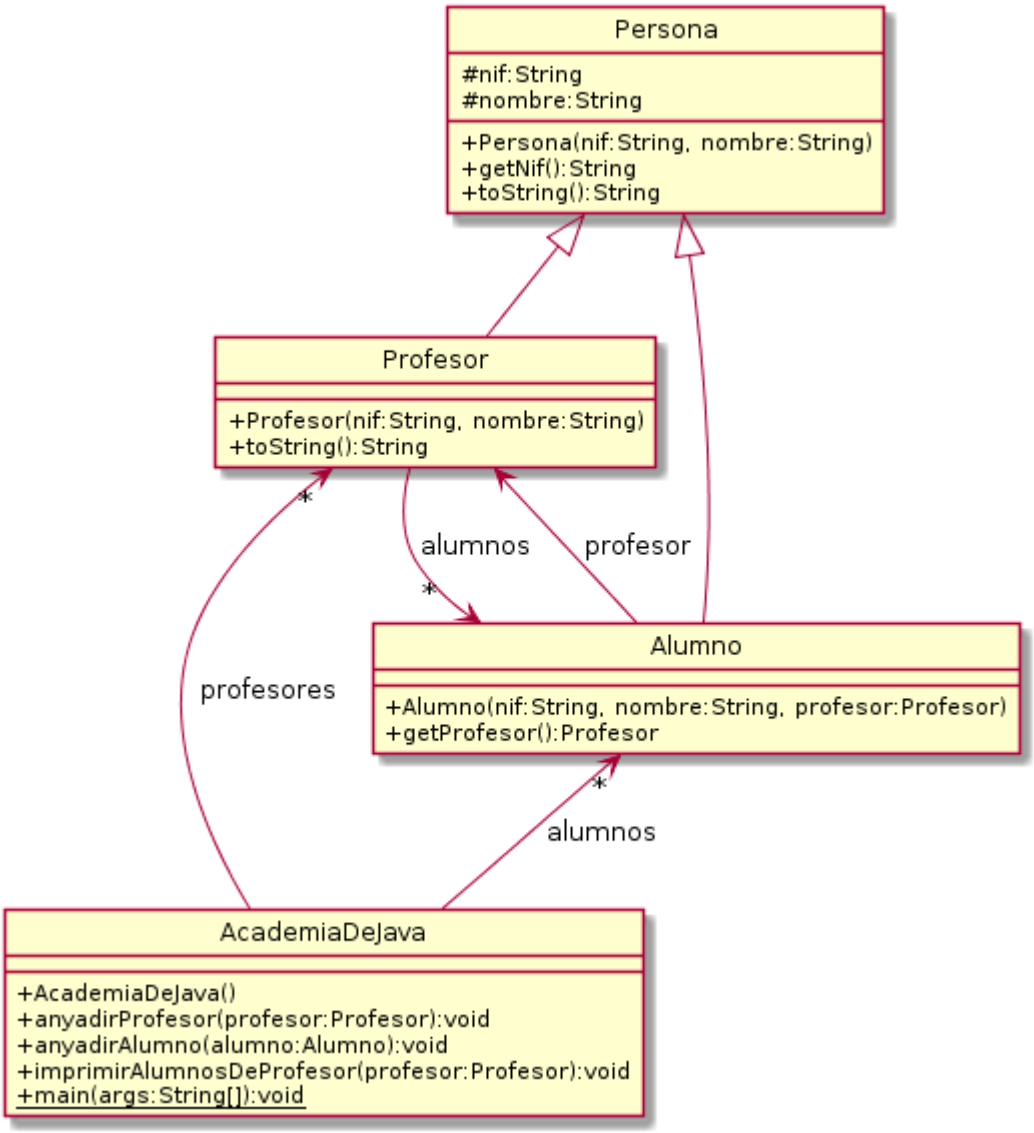
http://plantuml.com/class-diagram

1 Academia de Java

(Fue control de Laboratorio 2018-2019 Q2)

Implementad un proyecto de programación para gestionar una academia.
Aquí tenéis el diagrama UML y la salida que ha de generar.

El método main() crea un profesor y dos alumnos suyos e imprime los alumnos de dicho profesor.




```
run:
[nif=12344321Z, nombre=Joan Martinez
, nif=87654321W, nombre=Enric Perez
]
BUILD SUCCESSFUL (total time: 0 seconds)
```

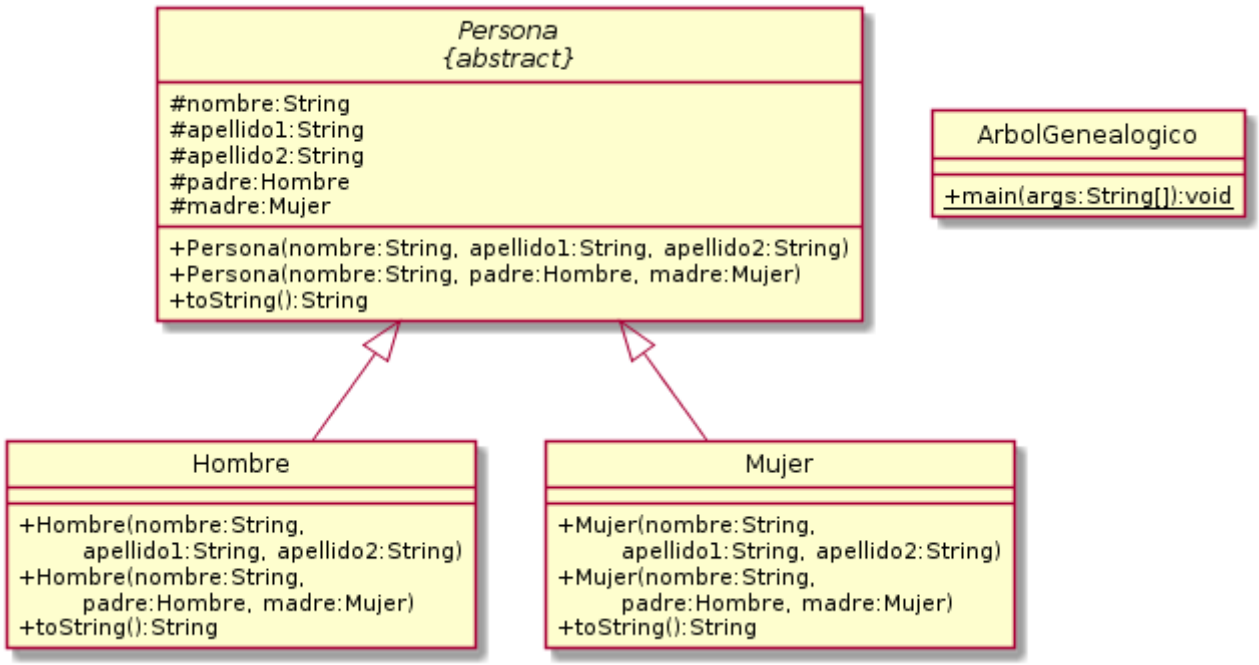
```
@startuml
/'
http://www.plantuml.com/plantuml/uml
http://plantuml.com/class-diagram
'/

hide circle
skinparam classAttributeIconSize 0

class Persona {
      #nif:String
      #nombre:String
      +Persona(nif:String, nombre:String)
      +getNif():String
      +toString():String
}

class Profesor {
      +Profesor(nif:String, nombre:String)
      +toString():String
}

class Alumno {
      +Alumno(nif:String, nombre:String, profesor:Profesor)
      +getProfesor():Profesor
}

class AcademiaDeJava {
      +AcademiaDeJava()
      +anyadirProfesor(profesor:Profesor):void
      +anyadirAlumno(alumno:Alumno):void
      +imprimirAlumnosDeProfesor(profesor:Profesor):void
      {static} +main(args:String[]):void
}

Persona <|-- Alumno
Persona <|-- Profesor

Alumno "*" <-- Profesor: alumnos
Profesor <-- Alumno: profesor
Alumno "*" <-- AcademiaDeJava: alumnos
Profesor "*" <-- AcademiaDeJava: profesores

@enduml
```

2 Árbol genealógico

(Fue control de Laboratorio 2018-2019 Q2)

Implementad un proyecto de programación que trabaje con árboles genealógicos.
Aquí tenéis el diagrama UML y la salida que ha de generar.

Si creamos una persona sin decir quienes son sus padres, lo haremos con su nombre y apellidos.
Y, si la creamos con sus padres ya no hace falta que le pasemos sus apellidos porque son los primeros apellidos de sus padres.
El método main() crea un padre y una madre (a partir de sus nombres y apellidos) y un hijo de ellos dos (a partir del nombre del hijo y del padre y la madre) y los imprime por pantalla.




```
run:
Padre: Hombre{Persona{nombre=Joan, apellido1=Martí, apellido2=Martinez,
        padre=null,
        madre=null}}
Madre: Mujer{Persona{nombre=Joana, apellido1=Garcia, apellido2=Perez,
        padre=null,
        madre=null}}
Hijo: Hombre{Persona{nombre=Joanet, apellido1=Martí, apellido2=Garcia,
        padre=Hombre{Persona{nombre=Joan, apellido1=Martí, apellido2=Martinez,
        padre=null,
        madre=null}},
        madre=Mujer{Persona{nombre=Joana, apellido1=Garcia, apellido2=Perez,
        padre=null,
        madre=null}}}}
BUILD SUCCESSFUL (total time: 0 seconds)
```

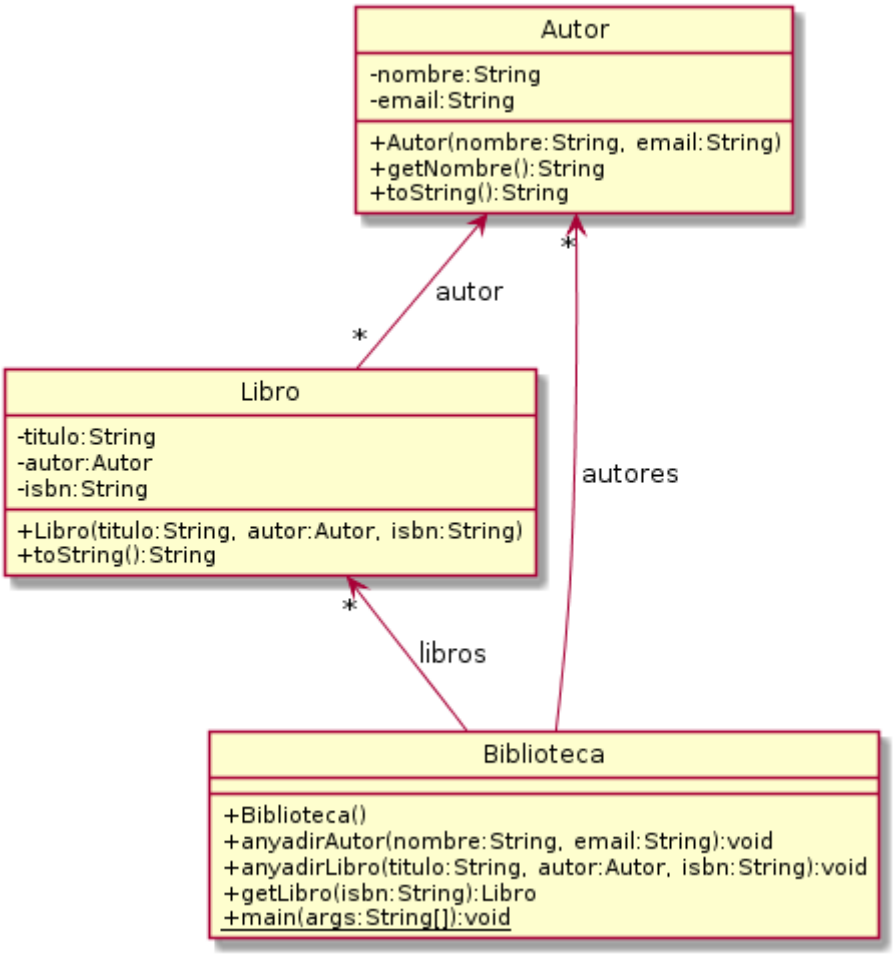
```
@startuml
/'
http://www.plantuml.com/plantuml/uml
http://plantuml.com/class-diagram
'/

hide circle
skinparam classAttributeIconSize 0

class Hombre {
      +Hombre(nombre:String, \n\tapellido1:String, apellido2:String)
      +Hombre(nombre:String, \n\tpadre:Hombre, madre:Mujer)
      +toString():String
}

class Mujer {
      +Mujer(nombre:String, \n\tapellido1:String, apellido2:String)
      +Mujer(nombre:String, \n\tpadre:Hombre, madre:Mujer)
      +toString():String
}

abstract class "Persona\n{abstract}" as Persona {
      #nombre:String
      #apellido1:String
      #apellido2:String
      #padre:Hombre
      #madre:Mujer
      +Persona(nombre:String, apellido1:String, apellido2:String)
      +Persona(nombre:String, padre:Hombre, madre:Mujer)
      +toString():String
}

Persona <|-- Hombre
Persona <|-- Mujer

class ArbolGenealogico {
      {static} +main(args:String[]):void
}

@enduml
```

3 La biblioteca

(Fue control de Laboratorio 2018-2019 Q2)

Implementad un proyecto de programación para gestionar una biblioteca.
Aquí tenéis el diagrama UML y la salida que ha de generar.

El método main() añade un autor y un libro de dicho autor a la biblioteca, lo busca por ISBN y lo imprime por pantalla. Los métodos anyadeLibro() y anyadeAutor() crean y añaden el libro o autor. El ISBN identificada un Libro y se asume que el nombre identifica un Autor.




```
run:
Buscar libro con ISBN: 978-0596007126
Libro{
titulo=Head First Design Patterns,
autor=Autor{
nombre=Eric Freeman,
email=eric.freeman@gmail.com},
isbn=978-0596007126}
BUILD SUCCESSFUL (total time: 0 seconds)
```

```
@startuml
/'
http://www.plantuml.com/plantuml/uml
http://plantuml.com/class-diagram
'/

hide circle
skinparam classAttributeIconSize 0

class Autor {
      -nombre:String
      -email:String
      +Autor(nombre:String, email:String)
      +getNombre():String
      +toString():String
}
class Libro {
      -titulo:String
      -autor:Autor
      -isbn:String
      +Libro(titulo:String, autor:Autor, isbn:String)
      +toString():String
}

class Biblioteca {
      +Biblioteca()
      +anyadirAutor(nombre:String, email:String):void
      +anyadirLibro(titulo:String, autor:Autor, isbn:String):void
      +getLibro(isbn:String):Libro
      {static} +main(args:String[]):void
}

Autor <-- "*" Libro: autor
Libro "*" <-- Biblioteca: libros
Autor "*" <-- Biblioteca: autores

@enduml
```

11 Problemas complejos

1 El juego de la vida

Implementar el “juego de la vida” en una matriz de tamaño DIMxDIM donde DIM es una constante y donde la matriz se inicializa al azar con un PORCENTAGE_INICIAL de casillas “vivas”.

Mirad la wikipedia para ver como va:

https://es.wikipedia.org/wiki/Juego_de_la_vida

“Todas las células se actualizan simultáneamente en cada turno, siguiendo estas reglas:
- Una célula muerta con exactamente 3 células vecinas vivas "nace"
(es decir, al turno siguiente estará viva).
- Una célula viva con 2 o 3 células vecinas vivas sigue viva, en otro caso muere
(por "soledad" o "superpoblación").”

En una primera versión hay que pulsar retorno de carro para generar la siguiente generación.
En una segunda versión se pasa a la siguiente generación cada ciertos milisegundos y el programa termina cuando la matriz se ha estabilizado o se han realizado un numero máximo de generaciones.
Podéis hacer que cada vez que imprima el “tablero” también imprima el número de generación y la población actual.

```
run:
··#·#·#···
##··#·##··
··#·····##
··········
··········
#·····#···
#··##····#
····##····
··#·#···#·
·····#··##

·#·#··##··
·##···###·
·#·····##·
··········
··········
··········
···##·····
·····#····
···##···##
········##
```

```
·#····#·#·
##········
·##···#·#·
··········
··········
··········
····#·····
·····#····
····#···##
········##
[…]
```

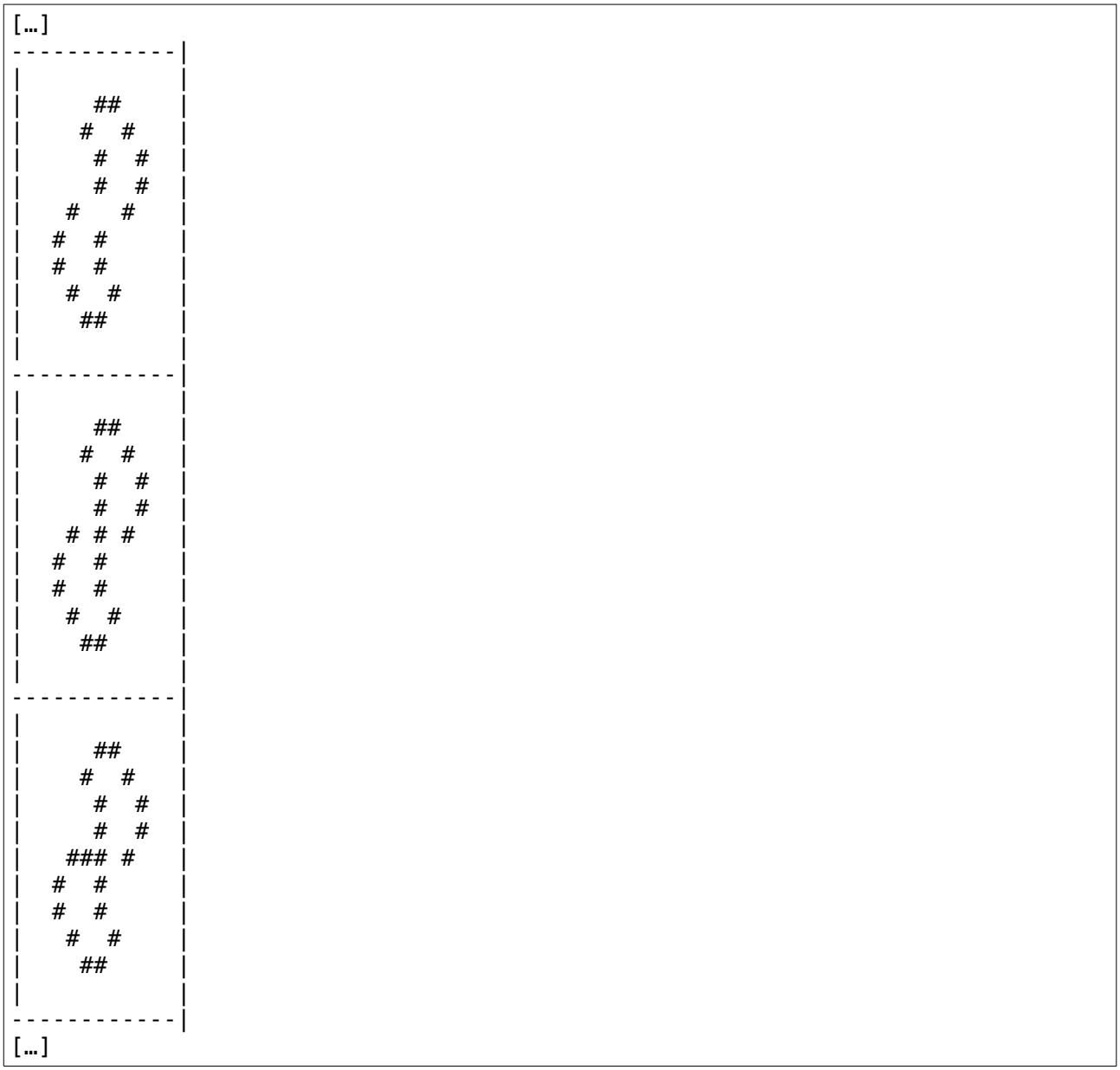
2 La hormiga de Langton

Utilizando la descripción de la wikipedia implementar la versión clásica de la hormiga de Langton.

https://es.wikipedia.org/wiki/Hormiga_de_Langton

“Comenzando con una rejilla totalmente blanca:
- Si está sobre un cuadrado blanco, cambia el color del cuadrado, gira noventa grados a la izquierda y avanza un cuadrado.
- Si está sobre un cuadrado negro, cambia el color del cuadrado, gira noventa grados a la derecha y avanza un cuadrado.”

Hay que definir la dirección y posición inicial. Por ejemplo: centro del tablero y empieza moviendo hacía arriba.



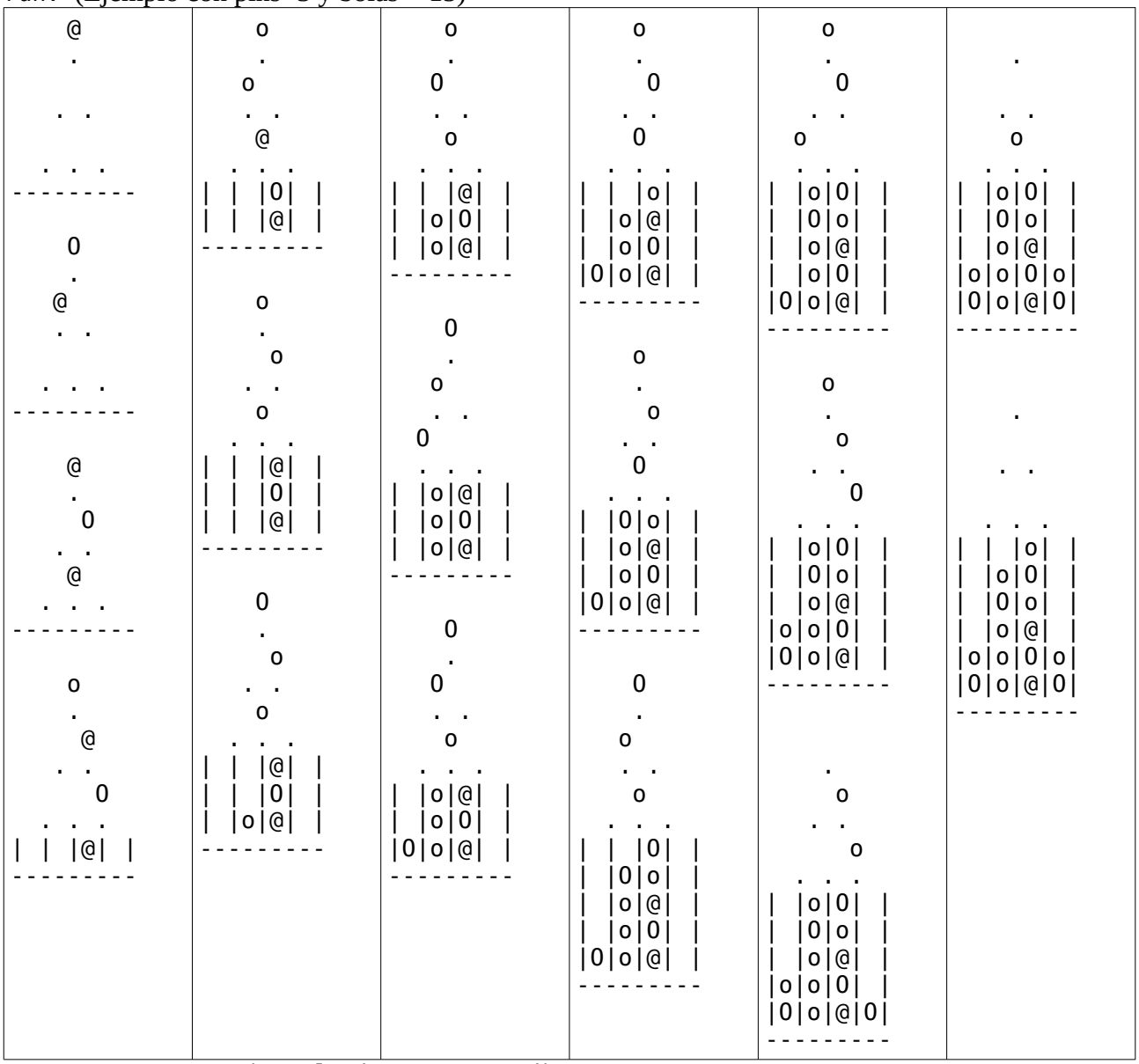
3 La máquina de Galton

Utilizando la descripción de la wikipedia implementar la máquina de Galton

https://es.wikipedia.org/wiki/M%C3%A1quina_de_Galton

“La máquina de Galton, o caja de galton, es un dispositivo inventado por Francis Galton para demostrar el teorema del límite central, en particular que la distribución binomial es una aproximación a la distribución normal.
La máquina consta de un tablero vertical con varias filas de clavos. Las bolillas caen desde la parte superior, botando aleatoriamente y van depositándose, a medida que caen, en los casilleros de la parte inferior. Formando una superficie de campana.”

`run:` (Ejemplo con pins=3 y bolas = 15)



```
BUILD SUCCESSFUL (total time: 1 second)
```

Como extra podéis hacer que la bola (de manera aleatoria) sea 'o', 'O' o '@', (tal y cómo está en el ejemplo).

4 El juego del oso

Implementar el juego del oso como una matriz de casillas:

https://es.wikipedia.org/wiki/Oso_(juego)

Vuestra implementación tiene que hacer que las casillas donde no hay ninguna letra sean “null”. Como extra podéis hacer que se pueda jugar contra el ordenador. En el ejemplo no se ve pero cada letra tiene que imprimirse en el color del jugador (por ejemplo: jugador #1 en rojo, #2 en azul, etc).

```
Run:
Numero de jugadores (2-6): 2
Numero de jugadores humanos (0-2): 0
Mapa:
      Filas (3-15): 4
      Columnas (3-35): 4

  |0|0|0|0|
  |0|1|2|3|
00| | | | |
01| | | | |
02| | | | |
03| | | | |

OSOs:  [#0]: 0 | [#1]: 0 |
Jugador [#0] 0 osos

  |0|0|0|0|
  |0|1|2|3|
00|S| | | |
01| | | | |
02| | | | |
03| | | | |

OSOs:  [#0]: 0 | [#1]: 0 |
Jugador [#1] 0 osos

  |0|0|0|0|
  |0|1|2|3|
00|S| | | |
01| | |O| |
02| | | | |
03| | | | |

OSOs:  [#0]: 0 | [#1]: 0 |
Jugador [#0] 0 osos

  |0|0|0|0|
  |0|1|2|3|
00|S| | | |
01| | |O| |
02| | |S| |
03| | | | |

OSOs:  [#0]: 0 | [#1]: 0 |
Jugador [#1] 1 osos

[…]
```

```
  |0|0|0|0|
  |0|1|2|3|
00|S| | |S|
01|O|O|O| |
02|S|S|S| |
03|O|O|O|O|

OSOs:  [#0]: 4 | [#1]: 2 |
Jugador [#0] 1 osos

  |0|0|0|0|
  |0|1|2|3|
00|S| | |S|
01|O|O|O|O|
02|S|S|S| |
03|O|O|O|O|

OSOs:  [#0]: 5 | [#1]: 2 |
Jugador [#0] 1 osos

  |0|0|0|0|
  |0|1|2|3|
00|S| | |S|
01|O|O|O|O|
02|S|S|S|S|
03|O|O|O|O|

OSOs:  [#0]: 6 | [#1]: 2 |
Jugador [#0] 0 osos

  |0|0|0|0|
  |0|1|2|3|
00|S| |O|S|
01|O|O|O|O|
02|S|S|S|S|
03|O|O|O|O|

OSOs:  [#0]: 6 | [#1]: 2 |
Jugador [#1] 0 osos

  |0|0|0|0|
  |0|1|2|3|
00|S|S|O|S|
01|O|O|O|O|
02|S|S|S|S|
03|O|O|O|O|

OSOs:  [#0]: 6 | [#1]: 2 |
JUGADOR/ES GANADOR/ES:[#0]!
```

Para imprimir las 'O's y 'S's en color se pueden usar los “ANSI escape codes” tal y como se explica en:

https://stackoverflow.com/questions/5762491/how-to-print-color-in-console-using-system-out-println

“
If your terminal supports it, you can use ANSI escape codes to use color in your output. It generally works for Unix shell prompts; however, it doesn't work for Windows Command Prompt (Although, it does work for Cygwin). For example, you could define constants like these for the colors:

```
public static final String ANSI_RESET = "\u001B[0m";
public static final String ANSI_BLACK = "\u001B[30m";
public static final String ANSI_RED = "\u001B[31m";
public static final String ANSI_GREEN = "\u001B[32m";
public static final String ANSI_YELLOW = "\u001B[33m";
public static final String ANSI_BLUE = "\u001B[34m";
public static final String ANSI_PURPLE = "\u001B[35m";
public static final String ANSI_CYAN = "\u001B[36m";
public static final String ANSI_WHITE = "\u001B[37m";
```

Then, you could reference those as necessary.

For example, using the above constants, you could make the following red text output on supported terminals:

```
System.out.println(ANSI_RED + "This text is red!" + ANSI_RESET);
```
“

5 Tetravex

Implementar el juego del Tetravex:

http://smart-games.org/en/tetravex/game/
https://gamegix.com/tetravex/game

Donde al principio las piezas ya están en el tablero, pero están mezcladas.

```
Tamano tablero entre 2 y 6: 3

    A   B   C
  +---+---+---+
  |\4/|\4/|\8/|
 0|4x1|5x1|1x0|
  |/5\|/1\|/1\|
  +---+---+---+
  |\7/|\1/|\1/|
 1|2x0|1x6|4x5|
  |/2\|/7\|/4\|
  +---+---+---+
  |\5/|\1/|\3/|
 2|5x2|6x0|0x5|
  |/7\|/3\|/4\|
  +---+---+---+

[00h:00m:00s]

Casillas a intercambiar: A0A1

    A   B   C
  +---+---+---+
  |\7/|\4/|\8/|
 0|2x0|5x1|1x0|
  |/2\|/1\|/1\|
  +---+---+---+
  |\4/|\1/|\1/|
 1|4x1|1x6|4x5|
  |/5\|/7\|/4\|
  +---+---+---+
  |\5/|\1/|\3/|
 2|5x2|6x0|0x5|
  |/7\|/3\|/4\|
  +---+---+---+
```

```
[00h:00m:03s]

Casillas a intercambiar: A0B2

    A   B   C
  +---+---+---+
  |\1/|\4/|\8/|
 0|6x0|5x1|1x0|
  |/3\|/1\|/1\|
  +---+---+---+
  |\4/|\1/|\1/|
 1|4x1|1x6|4x5|
  |/5\|/7\|/4\|
  +---+---+---+
  |\5/|\7/|\3/|
 2|5x2|2x0|0x5|
  |/7\|/2\|/4\|
  +---+---+---+

[00h:00m:06s]

Casillas a intercambiar: A0C1

    A   B   C
  +---+---+---+
  |\1/|\4/|\8/|
 0|4x5|5x1|1x0|
  |/4\|/1\|/1\|
  +---+---+---+
  |\4/|\1/|\1/|
 1|4x1|1x6|6x0|
  |/5\|/7\|/3\|
  +---+---+---+
  |\5/|\7/|\3/|
 2|5x2|2x0|0x5|
  |/7\|/2\|/4\|
  +---+---+---+

[00h:00m:09s]
Puzzle resuelto!!!
```

6 El dominó

Implementar el dominó:

https://es.wikipedia.org/wiki/Domin%C3%B3

Como mucho un jugador puede ser humano. El resto los debe llevar el ordenador (“robots”). Los jugadores “robots” pueden ser tan simples como realizar una jugada legal al azar o “ir a la pila” si no pueden tirar ninguna ficha. Los diferentes jugadores (Humano, T800 y T1000) serán clases hijas de la clase Jugador e implementarán el método polimórfico que, dada una colección de posibles tiradas, le pregunta al jugador cuan escoge.

Si juega un humano el programa le pedirá que jugada de las posibles quiere realizar.

```
run:
¿Numero de jugadores? [2-4]: 2
¿Un jugador humano?[s/n]: s
¿Quieres ver las fichas que estan boca a bajo?[s/n]: n
Como te llamas? Manel
Hola Manel! :-)
Pila: ?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|
Mesa: 6:6|
J0:   0:2|1:5|2:4|1:4|0:0|3:6|Manel(Humano)
J1:   ?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|T-800 N1(Robot)
Turno: J1
Tira: 6:5|d

Pila: ?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|
Mesa: 6:6|6:5|
J0:   0:2|1:5|2:4|1:4|0:0|3:6|Manel(Humano)
J1:   ?:?|?:?|?:?|?:?|?:?|?:?|?:?|T-800 N1(Robot)
Turno: J0
[0]1:5|d [1]3:6|i ¿Que tirada juegas? [0-1]: 1

Pila: ?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|?:?|
Mesa: 3:6|6:6|6:5|
J0:   0:2|1:5|2:4|1:4|0:0|Manel(Humano)
J1:   ?:?|?:?|?:?|?:?|?:?|?:?|?:?|T-800 N1(Robot)
Turno: J1
Tira: 5:5|d
```

En el siguiente ejemplo de ejecución hay dos tipos de robots: El T-800 que realiza una jugada al azar y el T-1000 que tiene una heurística simple para escoger una jugada relativamente buena.

```
run:
¿Numero de jugadores? [2-4]: 4
¿Un jugador humano?[s/n]: n

Pila:
Mesa: 6:6|
J0:   0:2|3:5|0:1|4:5|1:6|3:4|2:2|T-1000 N1(Robot)
J1:   2:6|1:1|0:4|0:5|1:5|4:4|T-800 N1(Robot)
J2:   2:4|2:3|5:5|4:6|3:3|3:6|1:4|T-1000 N2(Robot)
J3:   5:6|2:5|0:0|0:6|0:3|1:3|1:2|T-1000 N3(Robot)
Turno: J2
 | h(4:6|i) = 6 | h(4:6|d) = 6 | h(3:6|i) = 6 | h(3:6|d) = 6
```

```
Tira: 4:6|i

Pila:
Mesa: 4:6|6:6|
J0:   0:2|3:5|0:1|4:5|1:6|3:4|2:2|T-1000 N1(Robot)
J1:   2:6|1:1|0:4|0:5|1:5|4:4|T-800 N1(Robot)
J2:   2:4|2:3|5:5|3:3|3:6|1:4|T-1000 N2(Robot)
J3:   5:6|2:5|0:0|0:6|0:3|1:3|1:2|T-1000 N3(Robot)
Turno: J3
 | h(5:6|d) = 6 | h(0:6|d) = 7
Tira: 6:0|d

Pila:
Mesa: 4:6|6:6|6:0|
J0:   0:2|3:5|0:1|4:5|1:6|3:4|2:2|T-1000 N1(Robot)
J1:   2:6|1:1|0:4|0:5|1:5|4:4|T-800 N1(Robot)
J2:   2:4|2:3|5:5|3:3|3:6|1:4|T-1000 N2(Robot)
J3:   5:6|2:5|0:0|0:3|1:3|1:2|T-1000 N3(Robot)
Turno: J0
 | h(0:2|d) = 5 | h(0:1|d) = 5 | h(4:5|i) = 5 | h(3:4|i) = 5
Tira: 0:2|d

Pila:
Mesa: 4:6|6:6|6:0|0:2|
J0:   3:5|0:1|4:5|1:6|3:4|2:2|T-1000 N1(Robot)
J1:   2:6|1:1|0:4|0:5|1:5|4:4|T-800 N1(Robot)
J2:   2:4|2:3|5:5|3:3|3:6|1:4|T-1000 N2(Robot)
J3:   5:6|2:5|0:0|0:3|1:3|1:2|T-1000 N3(Robot)
Turno: J1
Tiro al azar. Me programaron asi. :-/
Tira: 0:4|i

Pila:
Mesa: 0:4|4:6|6:6|6:0|0:2|
J0:   3:5|0:1|4:5|1:6|3:4|2:2|T-1000 N1(Robot)
J1:   2:6|1:1|0:5|1:5|4:4|T-800 N1(Robot)
J2:   2:4|2:3|5:5|3:3|3:6|1:4|T-1000 N2(Robot)
J3:   5:6|2:5|0:0|0:3|1:3|1:2|T-1000 N3(Robot)
Turno: J2
 | h(2:4|d) = 7 | h(2:3|d) = 6
Tira: 2:4|d

Pila:
Mesa: 0:4|4:6|6:6|6:0|0:2|2:4|
J0:   3:5|0:1|4:5|1:6|3:4|2:2|T-1000 N1(Robot)
J1:   2:6|1:1|0:5|1:5|4:4|T-800 N1(Robot)
J2:   2:3|5:5|3:3|3:6|1:4|T-1000 N2(Robot)
J3:   5:6|2:5|0:0|0:3|1:3|1:2|T-1000 N3(Robot)
Turno: J3
 | h(0:0|i) = 20 | h(0:3|i) = 7
Tira: 0:0|i

Pila:
Mesa: 0:0|0:4|4:6|6:6|6:0|0:2|2:4|
J0:   3:5|0:1|4:5|1:6|3:4|2:2|T-1000 N1(Robot)
J1:   2:6|1:1|0:5|1:5|4:4|T-800 N1(Robot)
J2:   2:3|5:5|3:3|3:6|1:4|T-1000 N2(Robot)
J3:   5:6|2:5|0:3|1:3|1:2|T-1000 N3(Robot)
Turno: J0
 | h(0:1|i) = 7 | h(4:5|d) = 7 | h(3:4|d) = 7
Tira: 1:0|i
[…]
```

7 El buscaminas

Implementad el buscaminas. Recordad que el “destape en cascada” es más simple y eficiente en su implementación recursiva.

```
run:
Menu:
1) Tablero  8x 8  10 minas
2) Tablero 16x16  40 minas
3) Tablero 16x30  99 minas
4) Tablero personalizado
Que opcion eliges? [1-4]: 1

[00m:00s] | :-) | Minas marcadas: 0/10

 |0|1|2|3|4|5|6|7|
-+-+-+-+-+-+-+-+-+
0|x|x|x|x|x|x|x|x|
1|x|x|x|x|x|x|x|x|
2|x|x|x|x|x|x|x|x|
3|x|x|x|x|x|x|x|x|
4|x|x|x|x|x|x|x|x|
5|x|x|x|x|x|x|x|x|
6|x|x|x|x|x|x|x|x|
7|x|x|x|x|x|x|x|x|
-+-+-+-+-+-+-+-+-+

Acciones: '!' marcar, '?' marcar como dudosa, 'x' desmarcar, ' ' levantar.
Introduce jugada [fca] (fila, columna, accion [!?x ]): 44

[00m:07s] | :-) | Minas marcadas: 0/10

 |0|1|2|3|4|5|6|7|
-+-+-+-+-+-+-+-+-+
0|x|x|x|x|x|x|x|x|
1|x|x|1|1|1|1|2|x|
2|x|2|1| | | |2|x|
3|x|2| | | | |1|1|
4|x|3| | | | | | |
5|x|3|1| | |1|1|1|
6|x|x|2| | |1|x|x|
7|x|x|2| | |1|x|x|
-+-+-+-+-+-+-+-+-+

Acciones: '!' marcar, '?' marcar como dudosa, 'x' desmarcar, ' ' levantar.
Introduce jugada [fca] (fila, columna, accion [!?x ]): 66!

[51m:29s] | :-) | Minas marcadas: 1/10

 |0|1|2|3|4|5|6|7|
-+-+-+-+-+-+-+-+-+
0|x|x|x|x|x|x|x|x|
1|x|x|1|1|1|1|2|x|
2|x|2|1| | | |2|x|
3|x|2| | | | |1|1|
4|x|3| | | | | | |
5|x|3|1| | |1|1|1|
6|x|x|2| | |1|!|x|
```

```
7|x|x|2| | |1|x|x|
-+-+-+-+-+-+-+-+-+

Acciones: '!' marcar, '?' marcar como dudosa, 'x' desmarcar, ' ' levantar.
Introduce jugada [fca] (fila, columna, accion [!?x ]): 77

[51m:37s] | :-) | Minas marcadas: 1/10

 |0|1|2|3|4|5|6|7|
-+-+-+-+-+-+-+-+-+
0|x|x|x|x|x|x|x|x|
1|x|x|1|1|1|1|2|x|
2|x|2|1| | | |2|x|
3|x|2| | | | |1|1|
4|x|3| | | | | | |
5|x|3|1| | |1|1|1|
6|x|x|2| | |1|!|x|
7|x|x|2| | |1|x|1|
-+-+-+-+-+-+-+-+-+

Acciones: '!' marcar, '?' marcar como dudosa, 'x' desmarcar, ' ' levantar.
Introduce jugada [fca] (fila, columna, accion [!?x ]): 11

[51m:50s] | X-( | Minas marcadas: 1/10

 |0|1|2|3|4|5|6|7|
-+-+-+-+-+-+-+-+-+
0|x|x|x|x|*|x|x|x|
1|x|*|1|1|1|1|2|*|
2|x|2|1| | | |2|*|
3|*|2| | | | |1|1|
4|*|3| | | | | | |
5|*|3|1| | |1|1|1|
6|x|*|2| | |1|*|x|
7|x|*|2| | |1|x|1|
-+-+-+-+-+-+-+-+-+

 ---> BOOOOOOOOMMMMM!!!!!! <---
[…]
```