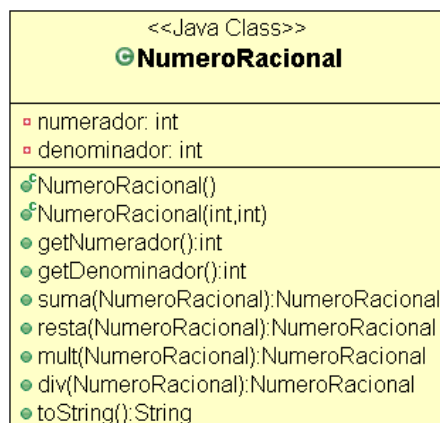


Programación Orientada a Objetos.

Ejercicios adicionales del Tema 2

- ✓1. Diseña una clase `NumeroRacional` que represente una fracción con numerador y denominador enteros. Por tanto tendrá dos variables de instancia (`numerador` y `denominador`) para almacenar dichos valores (ambas declaradas como `private`). Además tendrá dos constructores: uno sin argumentos (`NumeroRacional()`), que construye la fracción 0/1 y otro con dos argumentos (`NumeroRacional(int num, int den)`) que construye la fracción `num/den`. El denominador no puede ser 0, así que se lanzará una excepción (`RuntimeException`) si se pasa como segundo argumento un 0.



La clase ofrecerá métodos públicos para:

- Consultar el numerador (`int getNumerador()`)
- Consultar el denominador (`int getDenominador()`)
- Representar con una cadena de caracteres el número racional (`String toString()`). Si el denominador es 1, en la representación sólo aparecerá el numerador. Nota: para este caso puedes utilizar el método `toString()` de la clase `Integer` para convertir un entero a una cadena de caracteres (`String Integer.toString(int)`).
- Sumar dos fracciones (`NumeroRacional suma(NumeroRacional num)`)
- Restar dos fracciones (`NumeroRacional resta(NumeroRacional num)`)
- Multiplicar dos fracciones (`NumeroRacional mult(NumeroRacional num)`)
- Dividir dos fracciones (`NumeroRacional div(NumeroRacional num)`)

Los objetos de esta clase se deben mantener en su forma reducida. Por ejemplo la fracción 3/6 debería almacenarse como 1/2. También se debe evitar tener un denominador negativo. Así, la fracción 2/-3 se almacenará como -2/3, y la fracción -2/-3 se almacenará como 2/3.

Para probar esta clase, construye otra clase `PruebaRacionales` que contenga el método `main` para realizar diversas operaciones con números racionales.

Nota: si necesitas calcular el máximo común divisor de dos números (enteros positivos), puedes consultar los ejercicios resueltos de la relación 2 de la asignatura Fundamentos de la Programación. Y si necesitas calcular el mínimo común múltiplo, debes saber que: $mcd(a, b) \times mcm(a, b) = a \times b$. Si necesitas calcular el valor absoluto de un número entero puedes usar el método `Math.abs(int)`.

- ✓2. Un número complejo de coordenadas cartesianas es un número de la forma $a + bi$, donde a y b son números reales e i es la raíz cuadrada de -1. Las operaciones básicas con números complejos son las siguientes:

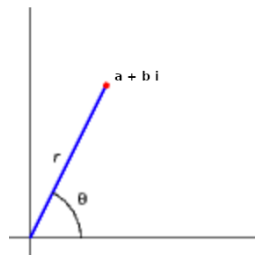
- *Suma:* $(a + bi) + (c + di) = (a + c) + (b + d)i$

- *Multiplicación:* $(a + bi) \times (c + di) = (ac - bd) + (ad + bc)i$
- *Magnitud:* $|a + bi| = \sqrt{a^2 + b^2}$
- *Parte real:* $\text{real}(a + bi) = a$
- *Parte imaginaria:* $\text{imaginaria}(a + bi) = b$

- a) Diseñe la clase Java `ComplejoCartesiano`, que represente este tipo de datos e implemente las operaciones descritas.
- b) La clase deberá contener también un método `toString` que devuelva la representación del número en la forma $a + bi$. Si b es 0, se mostrará sólo la parte real, y si la parte imaginaria es negativa se mostrará el signo $-$.
- c) Implemente la clase distinguida `ComplejoTest`, que cree los números complejos $1 + 3i$ y $1 - 2i$ y muestre siguiente salida en la consola:

```
x = 1.0 + 3.0i
y = 1.0 - 2.0i
|x| = 3.1622776601683795
|y| = 2.23606797749979
real(x) = 1.0
imaginaria(x) = 3.0
x + y = 2.0 + 1.0i
y + x = 2.0 + 1.0i
x * y = 7.0 + 1.0i
y * x = 7.0 + 1.0i
```

3. Un número complejo $z = a + bi$ en coordenadas cartesianas, se puede también representar en coordenadas polares r_θ , donde r es la magnitud del número, es decir, $r = \sqrt{a^2 + b^2}$ y θ es el ángulo del vector determinado por los puntos (0,0) y (a,b) con respecto al eje abscisas (positivo), que se calcula como $\theta = \arctan\left(\frac{a}{b}\right)$.



La relación entre ambas representaciones viene dada por la igualdad:

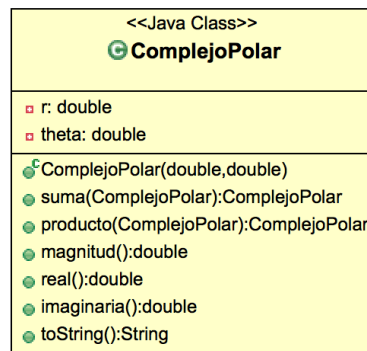
$$a + bi = r \cos \theta + r \sin \theta i$$

Las operaciones básicas se definen como sigue:

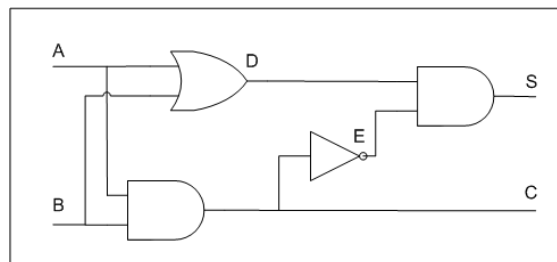
- *Suma:* $r_\alpha + r'_\beta = (r \cos \alpha + r' \cos \beta) + (r \sin \alpha + r' \sin \beta) i$
- *Multiplicación:* $r_\alpha \cdot r'_\beta = (r \cdot r')_{(\alpha+\beta)}$
- *Magnitud:* $|r_\theta| = r$
- *Parte real:* $\text{real}(r_\theta) = r \cos \theta$
- *Parte imaginaria:* $\text{imaginaria}(r_\theta) = r \sin \theta$

- a) Implemente la clase `ComplejoPolar` que internamente almacene los valores r y θ (*theta*) pero que conserve la misma interfaz que la clase del ejercicio anterior. Esto es, debe tener los mismos métodos y los objetos se construirán de la misma forma de modo que, si en

la clase `ComplejoTest` se sustituye la clase `ComplejoCartesiano` por `ComplejoPolar`, el usuario debe obtener los mismos resultados.



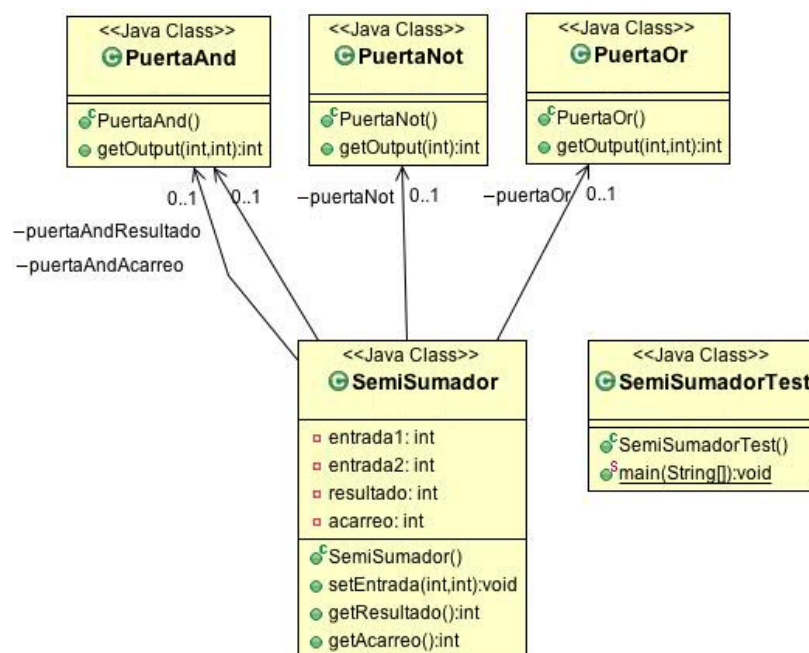
4. Un *semisumador* está compuesto por dos puertas *AND*, una puerta *OR* y una puerta *NOT* conectados como indica la figura, donde *A* y *B* son las entradas, *S* es el resultado y *C* el acarreo.



La siguiente tabla muestra las salidas *S* y *C* para todas las combinaciones de las entradas *A* y *B*:

A	B	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

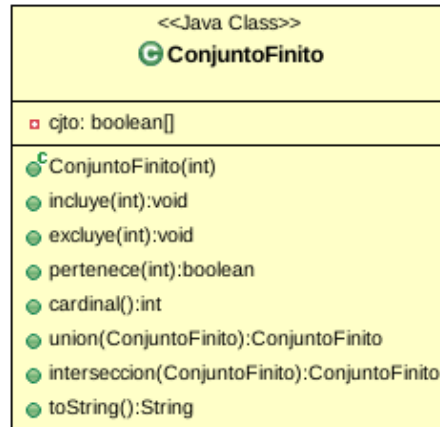
Se pide implementar un *semisumador* usando las clases que se muestran en el siguiente diagrama.



En método `main` de la clase `SemiSumadorTest` creará un objeto de la clase `SemiSumador` mostrará por pantalla los acarreos y sumas resultantes para todas las combinaciones de *A* y *B*.

5. Un modo de representar conjuntos finitos de números naturales menores a un determinado número (n), es mediante un array de booleanos de tamaño n . Si la casilla i -ésima del array vale verdadero, se considera que el número entero i está incluido en el conjunto finito. En caso contrario, se considera no incluido. Por ejemplo, el array {true, false, false, true, true} representaría al conjunto finito {0, 3, 4}.

Complete la siguiente clase que implementa los conjuntos finitos de números naturales usando esta representación:



Utilice la siguiente clase para probar la clase conjunto finito:

```

public class ConjuntoFinitoTest {
    public static void main (String[] args) {
        try {
            ConjuntoFinito c1 = new ConjuntoFinito(100);
            c1.incluye(1);
            c1.incluye(50);
            c1.incluye(60);
            System.out.println("c1 = " + c1 + " cardinal = " + c1.cardinal());
            ConjuntoFinito c2 = new ConjuntoFinito(70);
            c2.incluye(3);
            c2.incluye(60);
            System.out.println("c2 = " + c2 + " cardinal = " + c2.cardinal());
            System.out.println(c1 + " unión " + c2 + " es " + c1.union(c2));
            System.out.println(c1 + " intersección " + c2 + " es " + c1.interseccion(c2));
            c1.excluye(50);
            System.out.println("Se elimina el 50 de c1: " + c1);
            if (c1.pertenece(10)) {
                System.out.println("10 pertence a " + c1);
            } else {
                System.out.println("10 no pertence a " + c1);
            }
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
  
```







La salida debe ser:

```

c1 = {1,50,60} cardinal = 3
c2 = {3,60} cardinal = 2
{1,50,60} unión {3,60} es {1,3,50,60}
{1,50,60} intersección {3,60} es {60}
Se elimina el 50 de c1: {1,60}
10 no pertence a {1,60}
  
```

6. Defina la clase **Carga** para representar partículas cargadas, que contendrá los siguientes elementos:

- *Atributos o variables de instancia:* la posición de la carga, x e y , y la cantidad de la carga eléctrica, q .
- *Constructor:* inicializa las variables de instancia con los valores proporcionados.
- *Métodos de instancia:*
 - *Métodos de consulta de las variables de instancia:* `carga()`, `getx()` y `gety()`, devuelven respectivamente los valores de q , rx y ry .
 - Se definirá el método `potencialEn(...)` que calcula y devuelve el potencial eléctrico en un punto (x, y) debido a una partícula cargada. Según la ley de *Coulomb* se calcula como $V = kq/r$, donde $k = 8.99 \times 10^9$ es la constante electrostática, q es el valor de la carga y r es la distancia desde el punto (x, y) a la carga. La distancia entre los puntos $(x1, y1)$ y $(x2, y2)$ se calcula como $\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$

<<Java Class>>	
 Carga	
▣	<code>rx: double</code>
▣	<code>ry: double</code>
▣	<code>q: double</code>
	<code>Carga(double,double,double)</code>
	<code>carga():double</code>
	<code>getx():double</code>
	<code>gety():double</code>
	<code>potencialEn(double,double):double</code>

Escriba la clase distinguida `CargaTest` que contenga un método `main` que:

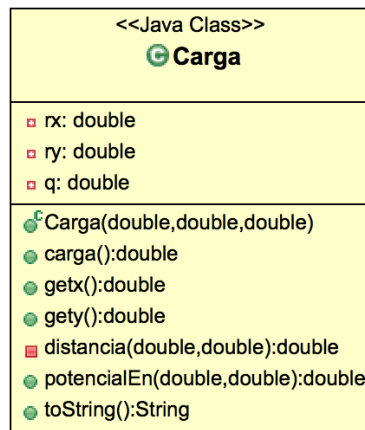
- Cree dos cargas, una con una cantidad 21.3 situada en (0.51, 0.63) y otra con una cantidad de carga 81.9 en (0.13, 0.94).
- Calcule el potencial de cada una de las cargas en el punto (1, 2).
- Muestre por pantalla el estado de cada una de las cargas y la suma de los potenciales calculados.

El resultado de la ejecución sería:

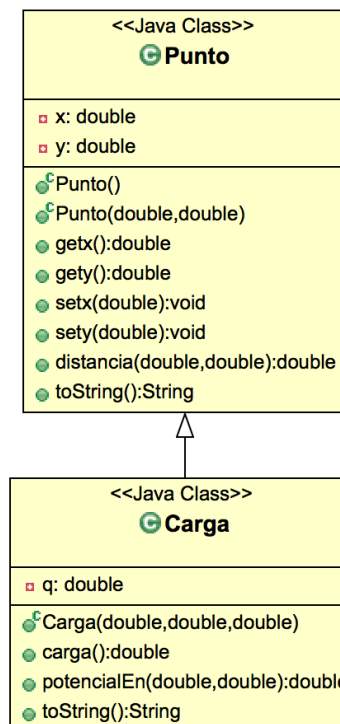
```
21.3 en (0.51, 0.63)
81.9 en (0.13, 0.94)
6.685236779167693E11
```

✓ Añada a la clase `Carga` del ejercicio anterior:

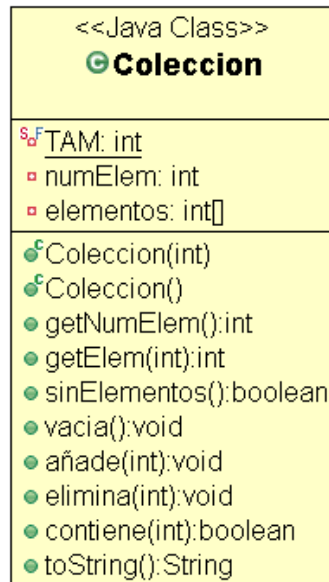
- Un método privado `distancia(double, double)` que devuelva la distancia de la carga a las coordenadas (x, y) recibidas como parámetros. Modifique `potencialEn(...)` para usar este nuevo método.
- El método `toString()` que devuelva una cadena de caracteres que represente el estado de una carga con el formato " q en (x, y) ".
- Modifique la clase `CargaTest` para obtener la misma salida que en el ejercicio anterior usando el método `toString()`.



8. Añada a su proyecto la clase Punto de los apuntes y haga que la clase Carga herede de ella. Realice las modificaciones necesarias para que la ejecución del programa no varíe.



9. Diseña una clase Coleccion que represente una colección de números enteros. Tendrá dos variables de instancia (ambas declaradas como **private**): un array de enteros denominado **elementos** (que almacenará la colección de números enteros de forma consecutiva a partir de la casilla 0) y un contador denominado **numElem** (que almacenará cuántos números enteros hay en la colección en cada momento, es decir, el número de casillas ocupadas del array). Además tendrá dos constructores: uno sin argumentos (**Coleccion()**), que crea una colección con un array vacío de tamaño **TAM** (una constante de clase privada definida e inicializada a 10) y otro con un argumento (**Coleccion(int tam)**) que crea una colección con un array vacío de tamaño **tam**. Se lanzará una excepción (**RuntimeException**) si el tamaño no es mayor que cero.

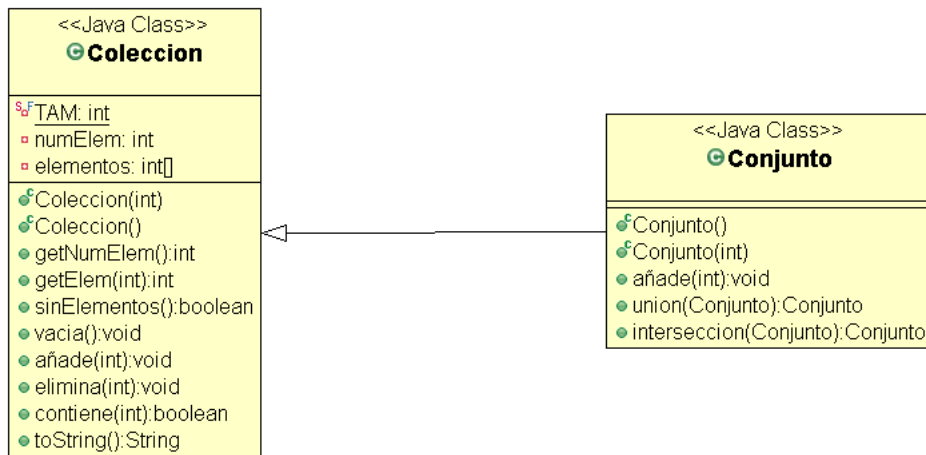


La clase ofrecerá métodos públicos para:

- Consultar el número de elementos de la colección (`int getNumElem()`)
- Consultar el elemento que ocupa la posición `i` de la colección, teniendo en cuenta que el primer elemento de la colección está en la posición 0 y el último ocupa la posición `numElem-1` (`int getElem(int i)`). Si el parámetro `i` no es correcto, se lanzará una excepción (`RuntimeException`).
- Comprobar si la colección está vacía (`boolean sinElementos()`)
- Vaciar una colección (`void vacia()`)
- Añadir un elemento a la colección (`void añade(int elemen)`). Si el array que almacena la colección está lleno, se doblará su capacidad y se meterá el nuevo elemento.
- Eliminar un elemento de la colección (`void elimina(int elem)`). Si el elemento no está, no se hace nada.
- Comprobar si la colección contiene un elemento (`boolean contiene(int elem)`)
- Representar con una cadena de caracteres la colección (`String toString()`). La representación consistirá en la secuencia de números que forman la colección, separados por comas y encerrados entre corchetes.

Para probar esta clase, construye otra clase `PruebaColeccion` que contenga el método `main` para realizar diversas operaciones con una colección de enteros.

10. Diseña una clase `Conjunto` que represente un conjunto de números enteros. Esta clase debe heredar de la clase `Coleccion` diseñada en un problema anterior. No añade ninguna variable de instancia más. Tendrá dos constructores: uno sin argumentos (`Conjunto()`), que crea un conjunto con un array vacío de tamaño `TAM` (una constante de clase definida en `Coleccion`) y otro con un argumento (`Conjunto(int tam)`) que crea un conjunto con un array vacío de tamaño `tam`. Se lanzará una excepción (`RuntimeException`) si el tamaño no es mayor que cero.



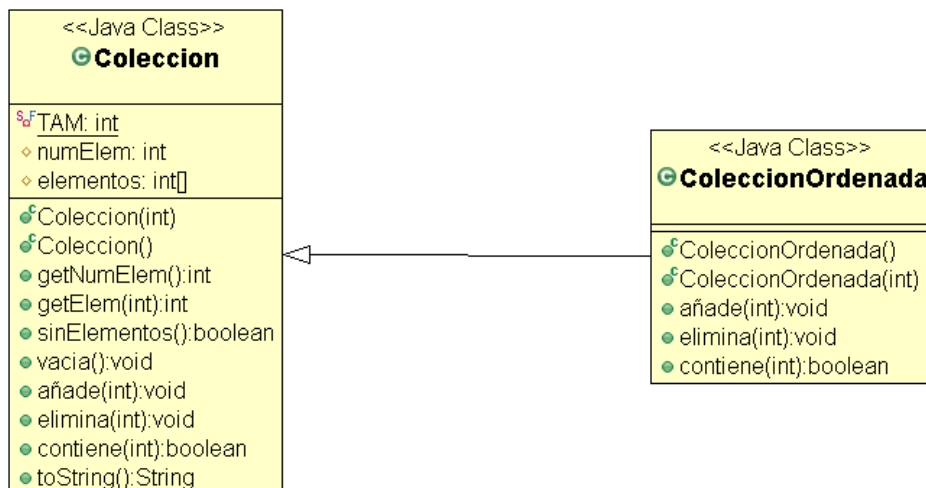
La clase ofrecerá métodos públicos para:

- Realizar la unión de dos conjuntos (`Conjunto union(Conjunto c)`). Devuelve un nuevo conjunto que es la unión del conjunto `c` y del conjunto receptor del mensaje `union`.
- Realizar la intersección de dos conjuntos (`Conjunto interseccion(Conjunto c)`). Devuelve un nuevo conjunto que es la intersección del conjunto `c` y del conjunto receptor del mensaje `interseccion`.

Además se deberá redefinir el método `añade` de `Coleccion` para evitar almacenar elementos repetidos.

Para probar esta clase, construye otra clase `PruebaConjunto` que contenga el método `main` para realizar diversas operaciones con un conjunto de enteros.

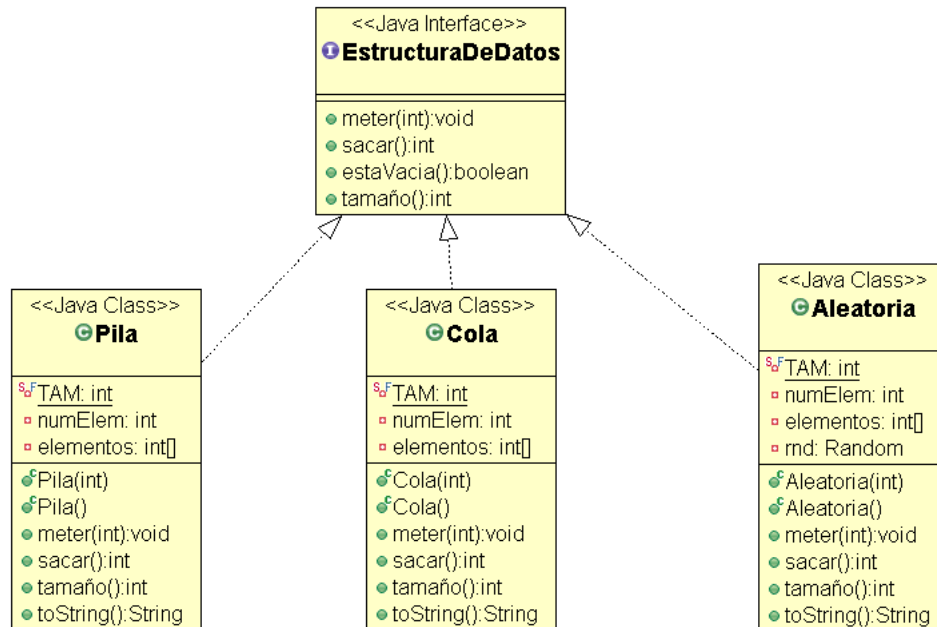
11. Diseña una clase `ColeccionOrdenada` que represente una colección ordenada (de menor a mayor) de números enteros. Esta clase debe heredar de la clase `Coleccion` diseñada en un problema anterior (la declaración de las variables de instancia de la clase `Coleccion` se debe modificar para hacer que su visibilidad sea `protected` en lugar de `private`). Tendrá dos constructores equivalentes a los de la clase `Coleccion`. Además habrá que redefinir los métodos `añade`, `elimina` y `contiene` para tener en cuenta la ordenación de los elementos.



Para probar esta clase, construye otra clase `PruebaColeccionOrdenada` que contenga el método `main` para realizar diversas operaciones con una colección ordenada de enteros.

12. Diseña una *Interfaz EstructuraDeDatos* para manipular estructuras de datos lineales que almacenan números enteros. Especifica las siguientes operaciones:
- Meter un elemento en la estructura (`void meter(int elem)`). Si la estructura está llena, se doblará su capacidad y se meterá el nuevo elemento.

- Sacar un elemento de la estructura (`int sacar()`). Si la estructura está vacía, se lanzará una excepción (`RuntimeException`).
- Consultar el número de elementos almacenados en la estructura (`int tamaño()`).
- Comprobar si la estructura está vacía (`boolean estaVacia()`). Esta operación será un *método por defecto* implementado en la propia Interfaz.



Diseña también 3 clases que implementen dicha Interfaz:

- Clase **Pila**. Una pila es una estructura de datos lineal en la que los elementos se meten y sacan por el mismo extremo (la cima de la pila).
- Clase **Cola**. Una cola es una estructura de datos lineal en la que los elementos se meten por un extremo (final de la cola) y se sacan por el otro extremo (principio de la cola).
- Clase **Aleatoria**. Esta clase representará una estructura de datos lineal en la que los elementos se meten y sacan por cualquier posición de la misma. Para determinar dicha posición se utilizará un objeto de la clase `java.util.Random`.

Estas 3 clases, además de diseñar los métodos ofrecidos por la Interfaz que implementan, también dispondrán del método `String toString()` para ofrecer una representación de la estructura de datos.

Por último, diseña una clase **Prueba** para probar el funcionamiento de la interfaz y clases anteriores. Esta clase tendrá 3 métodos *estáticos* (los dos primeros privados y el último público):

- Método `void rellenar(EstructuraDeDatos d)`. Almacena en la estructura de datos recibida como parámetro los números enteros del 0 al 9.
- Método `void vaciar(EstructuraDeDatos d)`. Saca todos los elementos de la estructura de datos recibida como parámetro, quedando ésta vacía.
- Método `void main(String[] args)`. Recibirá como argumento una cadena de caracteres, en función de la cual creará una estructura de datos u otra. Así, si la cadena recibida es "pila" se creará un objeto de la clase `Pila`; si la cadena es "cola" se creará un objeto de la clase `Cola`; en otro caso se creará un objeto de la clase `Aleatoria`. Después, utilizando los métodos anteriores, se realizará el siguiente proceso: se rellena la estructura y se imprime su contenido, se saca un elemento de la estructura y se imprime tanto el valor obtenido como el contenido de la estructura, se vacía la estructura y se imprime el contenido de la misma.