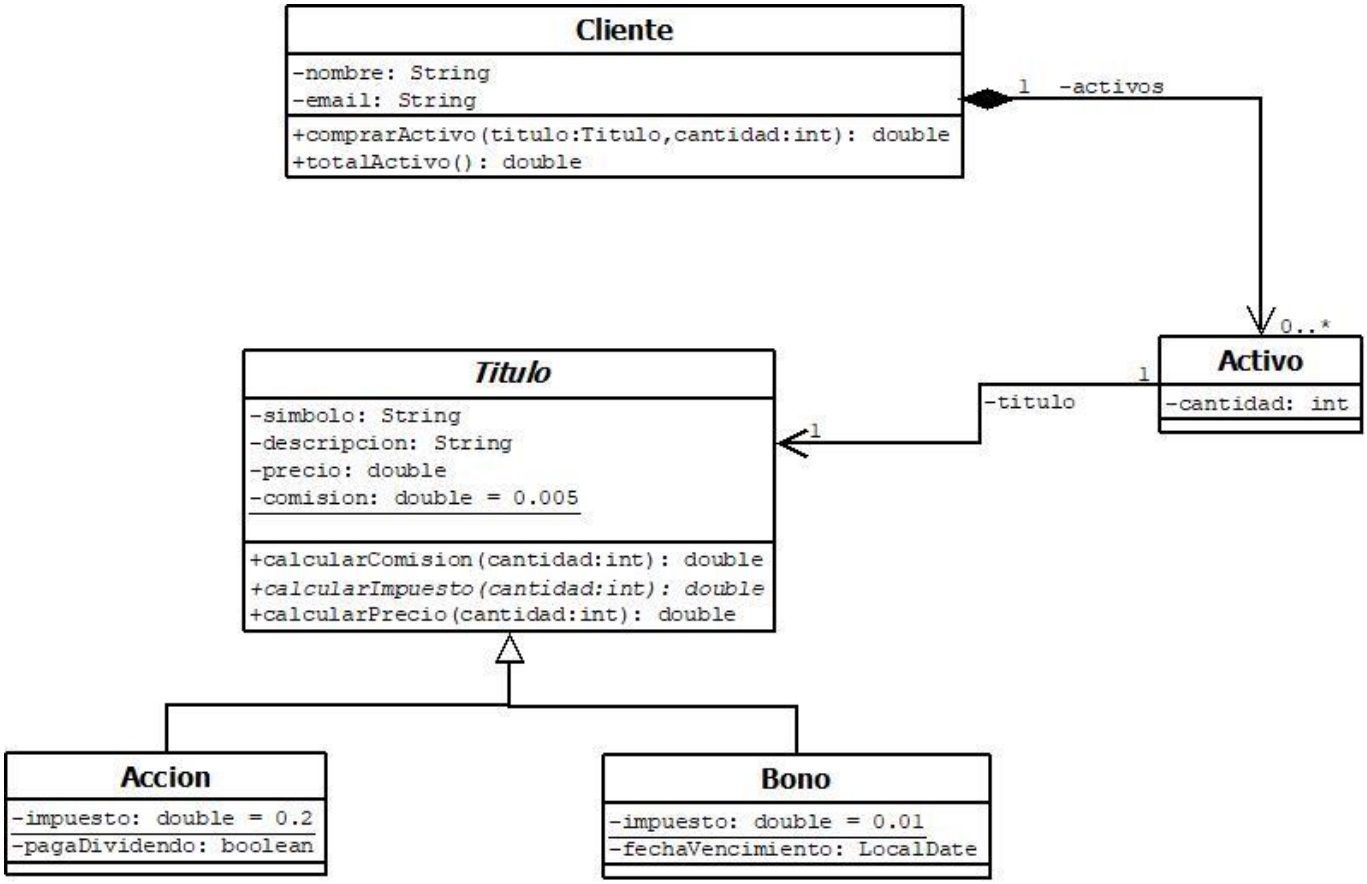
1° PARCIAL DE ALGORITMICA Y PROGRAMACION II - 2025

1. Un banco necesita un sistema para gestionar la compra de distintos tipos de activos a sus clientes.

Diagrama de Clases



Crear todas las clases con sus atributos y constructores, gets y sets, equals y toString.

2. Implementar los siguientes métodos:

```
/**
 * Calcula el impuesto por la compra realizada (cantidad * precio *
 * impuesto)
 *
 * @param cantidad: cantidad de títulos comprados
 * @return impuesto
 */
public abstract double calcularImpuesto(int cantidad);


/**
 * Calcula la comisión del banco por la compra realizada (cantidad * precio *
 * comision)
 *
 * @param cantidad: cantidad de títulos comprados
 * @return comisión del banco
 */
public double calcularComision(int cantidad)


/**
 * Calcula el precio total de la compra incluida la comisión del banco y los
```

```
 * impuestos (precio * cantidad + comisión + impuestos)
 *
 * @param cantidad: cantidad de títulos comprados
 * @return precio total de compra
 */
public double calcularPrecio(int cantidad)


/**
 * Agrega nuevos activos al cliente. Si el título no está en sus activos, lo
 * agrega. Si ya está en sus activos, incrementa su cantidad.
 *
 * @param titulo:   título comprado
 * @param cantidad: cantidad de títulos comprados
 * @throws ArrayIndexOutOfBoundsException: supera la cantidad máxima de títulos
 *                                         distintos que puede tener un cliente
 */
public void comprarActivo(Titulo titulo, int cantidad)


/**
 * Realiza la sumatoria de todos los activos del cliente. Multiplica el precio
 * de cada título por la cantidad que tiene comprada.
 *
 * @return valuación de la activos del cliente
 */
public double totalActivo()
```

3. Realizar un programa de prueba donde se cargan los siguientes datos:

```
Accion a1 = new Accion("ALUA", "Aluar", 750, false);
Accion a2 = new Accion("YPF", "YPF", 36100, true);
Bono b1 = new Bono("AL30", "Bono AL 30", 73500, LocalDate.of(2030, 6, 30));
Bono b2 = new Bono("GD35", "Bono GD 35", 74100, LocalDate.of(2035, 9, 30));

Cliente c1 = new Cliente("Juan", "juan@gmail.com");
Cliente c2 = new Cliente("Ana", "ana@gmail.com");

c1.comprarActivo(a1, 1200);
c1.comprarActivo(b1, 12);

c2.comprarActivo(a2, 150);
c2.comprarActivo(b2, 45);

c1.comprarActivo(a1, 1500);
c2.comprarActivo(b2, 35);
```

3.1 Llamar al método totalActivo para cada cliente y mostrar su resultado.

3.2 Colocar en un arreglo los objetos a1, a2, b1 y b2. Recorrerlo y llamar a los métodos calcularComision, calcularImpuesto y calcularPrecio para una cantidad de 100 títulos.

4. Dada una lista simplemente enlazada, implementar el siguiente método:

```
/**
 * Elimina todos los elementos que están en una posición impar.
 * Retorna una lista con los elementos eliminados
 *
 * Por ejemplo:
 *
 * {A, B, C, D} => {A, C} retorna la lista {B, D}
 *
```

```
 * {A, B, C} => {A, C} retorna la lista {B}
 *
 * {A, B} => {A} retorna la lista {B}
 *
 * {A} => {A} retorna la lista {}
 *
 * {} => {} retorna la lista {}
 */
public SinglyLinkedList<E> removeOdd()
```

5) Dada una lista simplemente enlazada, implementar el siguiente método:

```
/**
 * Retorna una nueva lista con los n elementos comenzado desde la derecha
 *
 * Por ejemplo:
 *
 * Dada la lista {A, B, C, D}
 *
 * right(4) retorna la lista {A, B, C, D}
 *
 * right(3) retorna la lista {B, C, D}
 *
 * right(2) retorna la lista {C, D}
 *
 * right(1) retorna la lista {D}
 *
 * right(0) retorna la lista {}
 *
 * right(5) lanza la excepción IndexOutOfBoundsException
 *
 * right(-1) lanza la excepción IndexOutOfBoundsException
 *
 *
 * @param n: número de elementos a retornar comenzando desde la derecha
 * @return nueva lista con los elementos de la derecha de la lista original
 * @throws IndexOutOfBoundsException n: supera el tamaño de la lista o es
 *                                   negativo
 */
public SinglyLinkedList<E> right(int n) throws IndexOutOfBoundsException
```

IMPORTANTE:

Subir al Campus solamente los archivos con extensión .java.