PRACTICO N° 17

EJERCICIO 1. Dadas las siguientes definiciones de clases y las declaraciones de variables

```
class Uno {
        int f (){ return 10; }
        int g (){ return 2; }
        int h () { return 100; }
        int m (int i) { return –i; } }
```

```
class Dos extends Uno {
        int f (){ return 11; }
        int g (){ return super.g() * 2; }
        int l () { return super.f() *-1 ;}
        int n () { return f() *-5 ;}
        int m(String s) { return s.length(); } }
```

```
class Tres extends Dos {
        int f (){ return 111; }
        int h (){ return 1000 ; }
        int p() {return 10000; }  }
```

```
Uno u1,u2,u3,u4;     Dos d1,d2,d3;     Tres t1,t2;
u1 = new Uno();   u2 = new Dos();   u3 = new Tres();
 t1 = new Tres();   t2 = (Tres) u3;
u4 = t1;
d1 = new Dos(); d2 = new Tres();  d3 = t2;
```

Indique qué instrucciones darán error de compilación y muestre la salida de las instrucciones válidas:

```
System.out.println("f "+u1.f()+" "+ u2.f()+" "+ u3.f()+" "+ u4.f());
System.out.println("g "+u1.g()+" "+ u2.g()+" "+ u3.g()+" "+ u4.g());
System.out.println("h "+u1.h()+"  "+ u2.h()+"  "+ u3.h()+"  "+ u4.h());
System.out.println("m  "+u1.m(8)+"  "+ u2.m(8)+"  "+ u3.m(8)+"  "+ u4.m(8));
System.out.println("f "+d1.f() +"  "+ d2.f()+"  "+ d3.f()    );
System.out.println("g "+d1.g()+"  "+ d2.g());
System.out.println(("h "+d1.h()+"  "+ d2.h());
System.out.println(("m "+d1.m(20)+"  "+ d2.m(20));
System.out.println(" f "+ t1.f()+" "+t2.f());
System.out.println(" g "+ t1.g()+" "+t2.g());
System.out.println(" h "+ t1.h()+" "+t2.h());
System.out.println(" m  "+ t1.m(3)+" "+t2.m(3));
System.out.println("l "+u2.l()+"  "+ u4.l());
System.out.println("l "+d1.l()+"  "+ d2.l());
System.out.println("n "+d1.n()+"  "+ d2.n());
```

EJERCICIO 2. Dadas las siguientes definiciones de clases y las declaraciones de variables`

```
class Alfa {
        int f (){ return 1; }
        int g (){ return 2; }}
class Beta extends Alfa {
        int f (){ return -1; }
        int h (){ return 10; }}
```

```
class Delta  {
    int v1 (Alfa a ){ return a.f()*a.f(); }
    int  v2 (Alfa a ){ return a.g()*a.g();}}
class Gama extends Delta  {
    int v1 (Beta a ){ return a.f()*a.h(); }
    int  v2 (Alfa a ){ return a.f()*a.g();}}
```

```
Alfa a1 = new Alfa(); Alfa a2 = new Beta();
Beta b = new Beta();
Delta d1 = new Delta(); Delta d2 = new Gama();
Gama g = new Gama();
```

Indique cuáles de las siguientes instrucciones provocarán errores de compilación. Siempre que sea posible, muestre cuál sería la salida por consola.

System.out.println(d1.v1(a1)); System.out.println(d1.v2(a1)); System.out.println(d1.v1(a2)); System.out.println(d1.v2(a2)); System.out.println(d1.v1(b)); System.out.println(d1.v2(b));	System.out.println(d2.v1(a1)); System.out.println(d2.v2(a1)); System.out.println(d2.v1(a2)); System.out.println(d2.v2(a2)); System.out.println(d2.v1(b)); System.out.println(d2.v2(b));	System.out.println(g.v1(a1)); System.out.println(g.v2(a1)); System.out.println(g.v1(a2)); System.out.println(g.v2(a2)); System.out.println(g.v1(b)); System.out.println(g.v2(b));

EJERCICIO 3: DADO EL SIGUIENTE DIAGRAMA

Coleccion
T [] TipoElemento cant:entero
<< Constructores>> Coleccion (max:entero) **<<Comandos>>** insertar(c:TipoElemento) eliminar(c:TipoElemento) **<<Consultas>>** estaLlena():boolean hayElementos():boolean cantElementos():entero existePosición(p:entero):boolean recuperarElemento(p:entero):TipoElemento estaElemento(c:TipoElemento):boolean

***TipoElemento**
*equals(t:TipoElemento):boolean *greater(t:TipoElemento):boolean

1. Implemente la clase genérica Colección para la siguiente especificación:
   Coleccion (max:entero) Crea una colección para max elementos
   insertar(c:TipoElemento) inserta el elemento c en la primera posición libre e incrementa cant. Asume que la estructura no está llena y c está ligada y no hay un elemento equivalente en la colección.
   eliminar(c:TipoElemento) busca un elemento equivalente a c. Si existe y ocupa la posición p, asigna cada elemento de la posición i, con p<i<cant a la posición i-1 y decrementa cant.
   estaLlena():boolean retorna true si cant es igual a la cantidad de elementos del arreglo
   hayElementos():boolean retorna true si cant es mayor a 0
   cantElementos():entero retorna cant
   existePosición(p:entero):boolean retorna true si 0<=p<cant
   recuperarElemento(p:entero):TipoElemento retorna el elemento asignado a la posición p, si p no es válida retorna nulo
   estaElemento(c:TipoElemento):boolean retorna true si algún elemento de la colección es equivalente a c, asume c ligada.

2. Implemente la clase ColeccionOrdenable que extiende a la clase genérica Coleccion.

ColeccionOrdenable
<< Constructores>> ColeccionOrdenable (max:entero) **<<Comandos>>** quickSort()

quickSort() ordena los elementos de la estructura aplicando el método recursivo quicksort

EJERCICIO 4: Implemente el siguiente diagrama de clases que modela una colección ordenada genérica de elementos:

ColeccionOrdenada
T [] Elemento cant:entero
<< Constructores>> ColeccionOrdenada (max:entero) **<<Comandos>>** insertar(c:Elemento) eliminar(c: Elemento) **<<Consultas>>** estaLlena():boolean hayElementos():boolean cantElementos():entero existePosición(p:entero):boolean recuperarElemento(p:entero):Elemento estaElemento(c: Elemento):boolean intercarlar(c:ColeccionOrdenada):ColeccionOrdenada

***Elemento**
*equals(t: Elemento):boolean *greater(t: Elemento):boolean

De acuerdo a la siguiente especificación:
Coleccion (max:entero) Crea una colección para max elementos

insertar(c:TipoElemento) inserta el elemento c manteniendo el orden la estructura e incrementa cant. Asume que la estructura no está llena y c está ligada y no hay un elemento equivalente en la colección.

eliminar(c:TipoElemento) busca un elemento equivalente a c. Si existe y ocupa la posición p, asigna cada elemento de la posición i, con p<i<cant a la posición i-1 y decrementa cant.

estaLlena():boolean retorna true si cant es igual a la cantidad de elementos del arreglo

hayElementos():boolean retorna true si cant es mayor a 0

cantElementos():entero retorna cant

existePosición(p:entero):boolean retorna true si 0<=p<cant

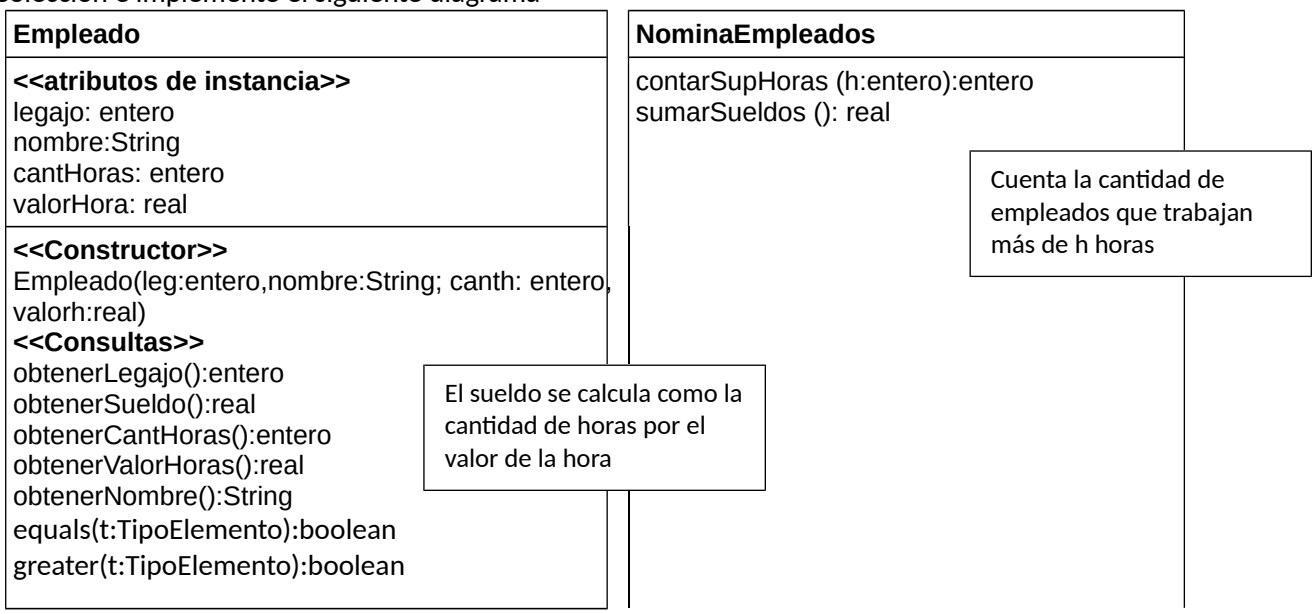
recuperarElemento(p:entero):TipoElemento retorna el elemento asignado a la posición p, si p no es válida retorna nulo

estaElemento(c:TipoElemento):boolean retorna true si algún elemento de la colección es equivalente a c, asume c ligada y usando búsqueda binaria.

intercarlar(c:ColeccionOrdenada):ColeccionOrdenada retorna una colección generada intercalando ordenadamente la colección que recibe el mensaje y la que pasa como parámetro.

EJERCICIO 5: Dadas las clases Coleccion y TipoElemento definidas antes

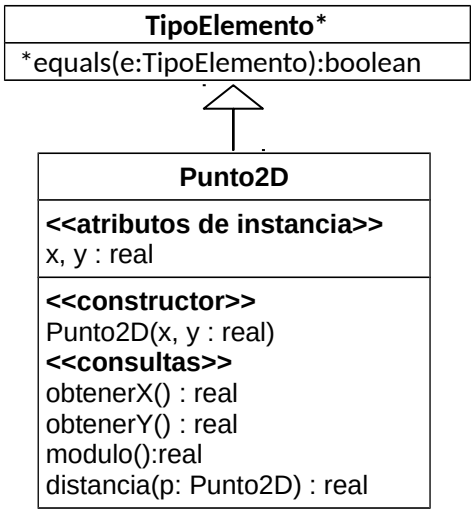
a) Implemente la clase Empleado extendiendo TipoElemento y la clase NominaEmpleados extendiendo a Coleccion e implemente el siguiente diagrama



LOS MÉTODOS EQUALS Y GREATER COMPARAN EL LEGAJO

b) Implemente la clase Técnico extendiendo Empleado y redefiniendo el método obtenerSueldo() computando el sueldo de cualquier empleado más un 15%
c) Defina una clase tester que cree una nómina de empleados, inserte 12 Empleados, algunos de clase Tecnico y compute la suma de sus sueldos.

EJERCICIO 6: Dadas las clases Coleccion y TipoElemento definidas antes, defina la clase Punto2D extendiendo TipoElemento y la clase Poligono extendiendo a Colección



Implemente servicios en la clase Poligono para

a. contar cuántos puntos del polígono tienen módulo mayor a un valor dado.
b. calcular la longitud de la poligonal determinada por la colección de puntos.
c. decidir todos los puntos están ubicados en el primer cuadrante, es decir x>0 e y>0.

EJERCICIO 7: Dado el siguiente diagrama de clase:

Matriz
<<atributos de instancia>>
M: TipoElemento[][]
<<constructor>>
Matriz(nf,nc: entero)
<<Comandos>>
establecer (f,c:entero,e: TipoElemento)
<<Consultas>>
obtener (f,c:entero):TipoElemento
pertenece (e: TipoElemento) : boolean
contarElem(e:TipoElemento):int
equals(c: Matriz) : boolean

TipoElemento*
*equals(e:TipoElemento):boolean
*mayor(e:TipoElemento):boolean

a. Implemente la clase genérica Matriz usando un arreglo.
b. Implemente una clase MatrizRacionales que extienda a Matriz y agregue un servicio esIdentidad(). El constructor de la clase recibe como parámetro el nombre de un archivo e inicializa la matriz de racionales con los valores leídos del archivo.
c. Implemente una clase MatrizPixels que extienda a la clase Matriz y agregue un servicio todosGrises() que decida si los pixels de la matriz representan algún matiz del color gris. El constructor de la clase recibe como parámetro el nombre de un archivo e inicializa la matriz de pixels con los valores leídos del archivo.
d. Implemente una clase tester para cada clase.

Ejercicio 8: Dado el siguiente diagrama

Conjunto
<<atributos de instancia>>
CE: TipoElemento[]
cantElementos: entero
Conjunto(max: entero)
<<Comandos>>
insertar(e: TipoElemento)
<<Consultas>>
pertenece (e: TipoElemento) : boolean
vacio():boolean
incluido(c: Conjunto) : boolean
equals(c: Conjunto) : boolean
disjunto(c :Conjunto) : boolean
union(c: Conjunto) : Conjunto
interseccion(c: Conjunto) : Conjunto
diferencia(c: Conjunto) : Conjunto
cardinalidad() : entero

Conjunto(max: entero)
Crea un conjunto vacío que puede tener hasta *max* elementos.

insertar(e:TipoElemento)
Inserta un elemento al final e incrementa cantElementos.
Si el elemento ya existe insertar no tiene ningún efecto

TipoElemento*
*equals(e:TipoElemento):boolean

Implemente la clase genérica Conjunto usando un arreglo.
Implemente una clase tester que verifique los servicios provistos por la clase Conjunto, a partir de dos conjuntos cuyas componentes sean de clase Punto2D. Para dos objetos A y B de clase Conjunto, el tester debe verificar que:

A.equals(B) es verdadero sí y solo sí A.incluido(B) y B.incluido(A) son verdadero.
A.equals(B) es verdadero sí y solo sí A.diferencia(B) y B.diferencia(A) son el conjunto vacío.
A.disjunto(B) es verdadero sí y solo sí A.interseccion(B) es el conjunto vacío.
A.disjunto(B) es verdadero sí y solo sí la cardinalidad de A.union(B) es igual a la cardinalidad de A más la cardinalidad de B.
A.union(B) es igual a la unión de A.diferencia(B), A.intersección(B) y B.diferencia(A).