Objetivos:

- Clase abstracta, composición vs herencia. Uso de métodos template.
- Pseudos variables self/super/this.
- Diseño OO

Ejercicio 1.

Introducción Pseudos variables: *Las pseudo-variables son identificadores reservados. Dos de las pseudos-variables más utilizadas son* ***self (o this) y super****. Tanto self (o this) como super varían de forma dinámica al ejecutar el código.*

self o this**:** *El objeto actual, esto es, el receptor del mensaje actual.*

super**:** *Como receptor de un mensaje, super se refiere al mismo objeto que self. Sin embargo, cuando se envía un mensaje a super, la búsqueda para un método adecuado comienza en una superclase de la clase cuya definición de método contiene la palabra super.*

a) Dado los siguientes métodos y clases indique que retornan las expresiones:

ClaseA	ClaseB	ClaseC

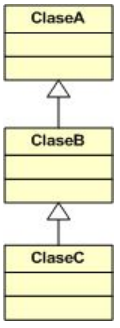
public int m1(){
return 3;
}
	public int m1(){
	return 6;
	}
		public int m1(){
		return this.m4();
		}

public int m2(){
return 5;
}
	public int m4(){
	return this.m2() + super.m3();
	}
		public int m2(){
		return 9;
		}

public int m3(){
return this.m6()+this.m2();
}
	public int m3(){
	return 4;
	}
		public int m7(){
		return super.m6();
		}

public int m6(){
return 0;
}
	public int m6(){
	return this.m2();
	}	

¿Qué retornan las siguientes expresiones?



ClaseC unObjeto = new ClaseC();
a) unObjeto.m2();

b) unObjeto.m7();
c) unObjeto.m1();
d) unObjeto.m4();
e) unObjeto.m6();

b) Dado los siguientes métodos y clases indique que retornan las expresiones:

ClaseA	ClaseB	ClaseC

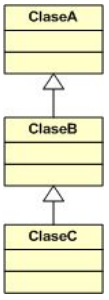
public int m1(){
return 2;
}
	public int m1(){
	return 8;
	}
		public int m1(){
		return this.m4();
		}

public int m2(){
return 10;
}
	public int m4(){
	return this.m7() + super.m3();
	}
		public int m2(){
		return 19;
		}

public int m3
return this.m2();
	public int m3(){
	return 1;
	}
		public int m7(){
		return super.m2();
		}

public int m7(){
return 0;
}
	public int m6(){
	return this.m2();
	}	

¿Qué retornan las siguientes expresiones?



ClaseC unObjeto = new ClaseC();
a) unObjeto.m2();
b) unObjeto.m7();
c) unObjeto.m1();

c) Dado los siguientes métodos y clases indique que retornan las expresiones:

ClaseA	ClaseB	ClaseC

public int m1(){
return 1;
}
	public int m1(){
	return this.m2;
	}
		public int m1(){
		return this.m7();
		}

public int m2(){
return 90;
}
	public int m4(){
	return this.m2() + super.m1();
	}
		public int m2(){
		return 5+super.m3();
		}

public int m3(){
return this.m6();
}
	public int m3(){
	return 30;
	}
		public int m7(){
		return this.m6();
		}

public int m7(){ return 5+this.m1();	public int m6(){ return this.m4();	

}	}	

public int m6(){
return 0;
}		

ClaseC unObjeto =new **ClaseC**();
1) unObjeto. m1()
2) unObjeto.m2()
3) unObjeto.m7()

ClaseB unObjeto :=new **ClaseB**();
4) unObjeto.m2()
5) unObjeto.m4()

Ejercicio 2.
Se desea modelar Cajas de Ahorro y Cuentas Corrientes. En ambos casos se tienen como atributos un titular, un saldo, un máximo de extracciones y una fecha de creación.

En ambos casos las cuentas pueden:

- extraer(double unMonto);
- depositar(double unMonto);
- saldo();
- nombreTitular(); //Retorna el nombre y apellido del titular.

Pero hay ciertas restricciones según la cuenta que sea:

- Las cajas de ahorro no pueden extraer más dinero que el saldo que poseen.
- Las cuentas corrientes pueden extraer más dinero que el saldo, pero tienen un límite en la cantidad de extracciones.

Por otro lado, se desea llevar un control de las transacciones realizada, es por ellos que cada vez que se realice una transacción, la misma se debe guardar en la cuenta. De cada transacción se conoce la fecha, el monto y el tipo de transacción (Un String simple)

a) Realice el diagrama de clases en UML
b) Implemente en JAVA-LIKE

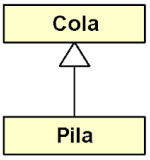
Nota: Tenga en cuenta que el *extraer* de la caja de ahorro y de la cuenta corriente deben utilizar el extraer de *cuenta bancari*a.

Ejercicio 3.
Se desea extender el ejercicio 2 para que ahora cada cuenta tenga en cuenta la moneda, las cuales pueden ser USD, Pesos o Yenes. Cada moneda tiene un valor de conversión determinado el cual puede variar. Cuando se pide el saldo el mismo se retorna siempre en pesos, con lo cual si la cuenta está en otra moneda se deberá hacer la conversión y luego retornar el saldo.

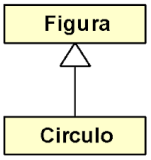
Ejemplo: Tenemos una cuenta donde el saldo es 1000, la moneda USD y el valor $65.00.- Cuando a esa cuenta le pedimos el saldo (dado que SIEMPRE se retorna en pesos) el valor de retorno es 6500.-

Ejemplo 2: Tenemos otra cuenta donde el saldo es 200, la moneda Yen y el valor $1.64.- Cuando a esa cuenta le pedimos el saldo (dado que SIEMPRE se retorna en pesos) el valor de retorno es 327.67.-

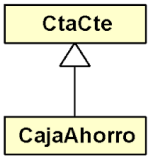
Ejercicio 4.
Diga si las siguientes jerarquías son correctas o no:



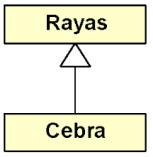
- ☐ Incorrecto
- ☐ Correcto



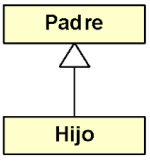
- ☐ Incorrecto
- ☐ Correcto



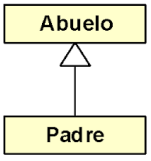
- ☐ Incorrecto
- ☐ Correcto



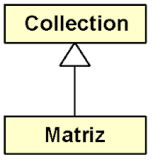
- ☐ Incorrecto
- ☐ Correcto



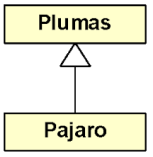
- ☐ Incorrecto
- ☐ Correcto



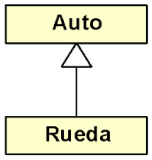
- ☐ Incorrecto
- ☐ Correcto



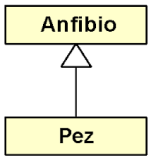
- ☐ Incorrecto
- ☐ Correcto



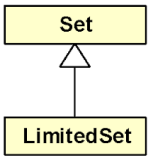
- ☐ Incorrecto
- ☐ Correcto



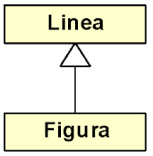
☐ Incorrecto
☐ Correcto



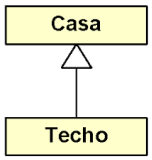
☐ Incorrecto
☐ Correcto



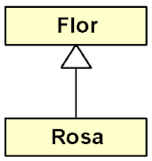
☐ Incorrecto
☐ Correcto



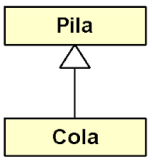
☐ Incorrecto
☐ Correcto



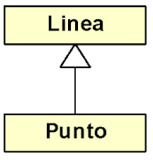
☐ Incorrecto
☐ Correcto



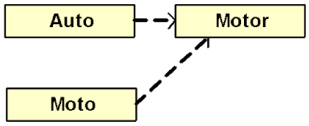
☐ Incorrecto
☐ Correcto



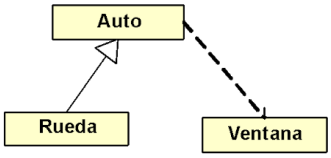
☐ Incorrecto
☐ Correcto



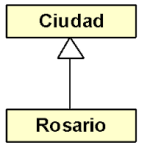
☐ Incorrecto
☐ Correcto



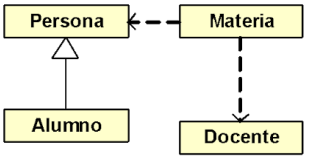
☐ Incorrecto
☐ Correcto



☐ Incorrecto
☐ Correcto



☐ Incorrecto
☐ Correcto



☐ Incorrecto
☐ Correcto

Ejercicio 5.
Se desea modelar un *traductor ortográfico*. Un ***Traductor*** es un sistema donde se le pide la traducción de una palabra de un idioma a otro.
El mismo debe soportar al menos el **Español** y el **Inglés**. Esto es traducir de español a ingles y de inglés al español. Una vez instanciado el traductor, este debe estar listo para traducir palabras simples o una colección de palabras.

El traductor debe tener la siguiente funcionalidad:

- **Instanciar e inicializar el Traductor.**

'instancia e inicializa el traductor'

- **traducirDeInglesAEspañol(String *unaPalabra*)**

'Traduce un palabra del inglés al español'

- **traducirDeEspañolAIngles(String *unaPalabra*)**

'Traduce un palabra del español al inglés'

- **cantidadDePalabrasSoportadas**()

'retorna la cantidad de palabras soportadas'

- **contienePalabra(String *unaPalabra*)**

'retorna true o false si la palabra que se envía por parametro se puede traducir'

- **traducirTodoDeInglesAEspañol(Vector<String> *unaColeccion)***

'dada una colección de palabras las traduce a todas'

- **traducirTodoDeEspañolAIngles(Vector<String> *unaColeccion)***

'idem anterior'

Nota: la traducción es de palabra a palabra, no hay que tener en cuenta la gramática.

Ejemplo de uso:

Traductor trad:= new Traductor();
trad.traducirDeInglesAEspañol("Hi") -> "Hola"
trad traducirDeEspañolAIngles("Hola") -> 'Hi'
Vector<String> v=new Vector<String> ();
v.add("Hola");
v.add("Buen");
v.add("Dia");
trad.traducirTodoDeEspañolAIngles: (v) -> "Hi" "Good" "Day"

Ejercicio 6.
Se desea modelar las siguientes estructuras:

a) Implementar en JAVA-LIKE la clase **Pila**
   a. Debe implementar al menos:
      i. apilar
      ii. desapilar
      iii. top
      iv. pop
      v. size
      vi. isEmpty

b) Implementar en JAVA-LIKE la clase **Cola**
   a. Debe implementar al menos:
      i. encolar

ii. desencolar
iii. isEmpty
iv. size
v. first //El primer objeto de la cola
vi. last //El ultimo objeto de la cola

c) Implementar en JAVA-LIKE la clase **ColaDoble**. Una **ColaDoble** es un objeto que maneja dos colas a la vez. Cuando se le dice encolar, este encola en la cola que menos objetos tenga. Al desencolar debe desencolar de la cola que más Objetos tenga. Cuando se le pide el tamaño (size) es la suma de las dos colas.
   a. Debe implementar al menos:
      i. encolar
      ii. desencolar
      iii. isEmpty
      iv. size
      v. first //El primer objeto de la cola más larga o en caso de empate el de la cola1.
      vi. Last //El último objeto de la cola más larga o en caso de empate de la cola2

Ejercicio 7. (SUBIR a https://plataformaed.unnoba.edu.ar)

Un tren está compuesto por una locomotora y un conjunto de vagones. Las locomotoras que pueden ser parte del tren son Eléctricas, Diesel o Fuel Oil. Cada locomotora tiene una velocidad máxima y una potencia máxima, las cuales pueden variar de una locomotora a otra. La potencia máxima indica el peso máximo que puede mover.

Además, cada locomotora tiene un consumo (en litros de combustible) que se mide en km recorridos. La siguiente tabla indica el consumo:

Tipo	Consumo
Eléctrica	0 litros por km
Diesel	10 litros por km
Fueloil	15 litros por km

Por otro lado, los vagones pueden ser de pasajeros o de carga. Los vagones de carga no pueden llevar pasajeros, en cambio los vagones de pasajeros pueden llevar X pasajeros.

En ambos casos, los vagones tienen un peso. En el caso de los vagones de carga es fijo (en el mismo se incluye el peso específico del vagón y el de la carga que pueden llevar) y en el caso de los vagones de pasajeros se calcula según el peso específico del vagón más 80kilos por cada una de las personas que puede llevar.

Las locomotoras tienen un peso específico que también se debe tener en cuenta. Y su velocidad disminuye según la cantidad de vagones que tenga enganchados. Por cada vagón de pasajeros la velocidad disminuye en un 3% y en el caso de los de carga en un 5%.

a) Modele en UML el diagrama de clases
b) Implemente en JAVA-LIKE todas las clases y métodos que se piden a continuación. Como así todos los métodos necesarios que crea convenientes:

I. **cantidadDeLitrosNecesarios(int km);** *//Este método retorna la cantidad de litros, según cada locomotora, que son necesarios para recorrer los kilómetros que se pasan como parámetro.*
II. **pesoTotalDelTren();** *//Este método retorna el peso total del tren teniendo en cuenta que los trenes pueden tener vagones de carga y vagones de pasajeros en la misma formación. Además se debe sumar el peso de la locomotora.*
III. **puedeMover(Vector<Vagon> vagones);** *//El tren retorna T/F en el caso de pueda mover o no la formación de vagones. Se tiene en cuenta la potencia de la locomotora en cuestión.*
IV. **cantidadDePasajeros();** *//Es la suma de todos los pasajeros que contienen los vagones. Recuerde que la formación puede contener vagones de carga que NO tienen pasajeros.*
V. **velocidadMaxima();** *//Es la velocidad máxima que puede levantar el tren según los vagones que tiene enganchados.*

Ejercicio 8. (SUBIR a https://plataformaed.unnoba.edu.ar)

ARBA (**A**gencia de **R**ecaudación de la Provincia de **B**uenos **A**ires) nos está pidiendo un sub-módulo para gestionar el cobro de impuestos de diferentes bienes personales.

La agencia cobra impuestos sobre todos los bienes de las personas, los cuales pueden ser autos, motos, embarcaciones e inmuebles. De cada bien se conoce un número de identificación, un valor fiscal (valor que le da ARBA) y el dueño de dicho bien.

Cada vez que se realiza un cobro, este se guarda en el sistema. Del cobro se conoce la fecha y el bien por el cual se realizó el mismo. El cobro además puede ser anual o en cuotas (la cantidad de cuotas es variable). El cálculo del monto a pagar de cada impuesto se realiza según la siguiente tabla:

	Moto	**Auto**	**Inmueble**	**Embarcación**
Cuotas	No Aplica	5% del valor fiscal	2% del valor fiscal	3% del valor fiscal
Anual	Un % variable del valor fiscal - 10% de descuento por pago anual	5% del valor fiscal - 10% de descuento por pago anual	2% del valor fiscal - 10% de descuento por pago anual	3% del valor fiscal - 10% de descuento por pago anual

Se pide que:

a) Realice el diagrama de clases en UML
b) Desarrolle en Java-Like todas las clases y métodos necesarios teniendo en cuenta que el sistema debe tener como mínimo la siguiente funcionalidad:

I) **public void agregarCobroAnual(Bien bien)**
/*Agrega un cobro a los cobros del sistema*/

II) **public float valorCobro(Cobro cobro)**
/* Retorna el valor de cobro teniendo en cuenta los descuentos pertinentes y porcentajes*/

III) **public float sumaCobros()**
/* Retorna la suma de todos los cobros*/

IV) **public Cobro cobroMayor()**
/* Retorna el cobro de mayor importe de la colección de cobros */

V) **public Vector<Cobro> cobroMayoresA(float valor)**
/* Retorna una colección con los cobros mayores al valor que se pasa por parámetro */

VI) **public float valorFiscal()**
/* Retorna la suma de todos los valores fiscales */

Ejercicio 9.

a) ¿Qué es Ruby?
b) ¿Qué es C# (See Sharp)?
c) ¿Qué es Squeak?
d) ¿Qué tipos de herencia soporta Ruby, C# y Squeak?
e) Brinde ejemplos de cómo hacer la clase persona en Ruby, en C# y en Squeak