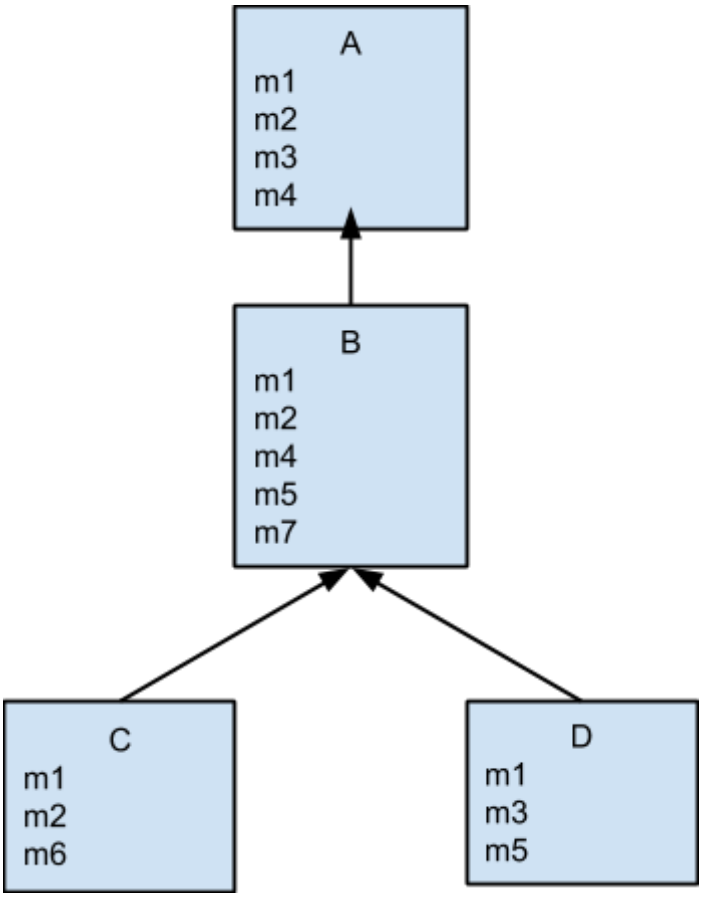
Ejercicio 1

Dado el siguiente diagrama de clases:



Cuya implementación es:

	m1()	m2()	m3()	m4()	m5()	m6()	m7()
A	this.m3()	10	3	this.m4()	-	-	Throw new RuntimeException()
B	8	super.m2()	-	20	this.m5()	-	super.m4()
C	super.m1()+ super.m2()	this.m5()	-	-	super.m5()	3	-
D	super.m3()	-	2	-	this.m4()	-	-

Complete la siguiente tabla, suponiendo que cada variable en minúscula es instancia de la clase en mayúscula

```
a = new A()
b = new B()
c = new C()
d = new D()
```

	m1	m2	m3	m4	m5	m6	m7
a							
b	8						
c							
d							

Luego verifiquelo en código.

Ejercicio 2

Diseñe e implemente las clases **Circle** y **Cylinder**, empleando composición y luego empleando herencia, de manera tal que pueda calcular el área y el volumen respectivamente.

Ejercicio 3

Implemente la clase **Figure,** y sus subclases **Rectangle** y **Triangle**. Luego agregue las clases **Circle** y **Ellipse** donde corresponda. Defina donde corresponda los métodos para obtener: perímetro, área, diagonal, altura y base. Pensar si debería haber una relación entre **Circle** y **Ellipse.** ¿Cuál debería ser?

Ejercicio 4

Realizar un programa que tenga un arreglo de **Figuras**, insertar círculos, rectángulos, triángulos, elipses, etc. Recorrer la estructura, invocando **perímetro** y **área.** Luego recorrer la estructura e imprimir la **diagonal** en aquellas que lo tengan definido. Controlar si funciona la equivalencia entre figuras. Investigue la palabra clave *instanceof.*

Ejercicio 5

Se quiere determinar si un punto se encuentra dentro de una figura, donde conviene definirlo? Implementarlo

Ejercicio 6

Diseñe una forma de definir polígonos, almacenando los vértices de estos. Haga una clase genérica, y luego restringir el funcionamiento mediante subclases. Por ejemplo, Mediante Triángulo, Cuadrado, Rectángulo, Pentágono, etc.

Ejercicio 7

La clase **Point2D** (ya implementada en la primera guía) permite representar un punto en un espacio cartesiano de dos dimensiones. Si se quisiera implementar la clase **Point3D** para representar puntos en un espacio en 3 dimensiones, ¿Cuál de las siguientes alternativas es más apropiada desde el punto de OOP?

- Definir la clase **Point3D** a partir de **Point2D** y agregar la variable **z**
- Definir la clase **Point3D** a partir de **Object**, sin relación alguna con **Point2D**
- Piense una tercera alternativa? Que ocurre con la redundancia de código? Critique esta implementación de **Point3D**.

Ejercicio 8

Crear las clases **SavingsAccount** y **CheckingAccount.** Evalúe cómo deberían ser las relaciones de herencia. Recuerde que una **CheckingAccount** tiene un descubierto autorizado disponible, mientras que la **SavingsAccount,** no. Además, la última tiene un máximo de operaciones gratuitas por mes. Alcanzado el límite de operaciones, se comienza a cobrar por los movimientos.

Las cuentas, tienen la posibilidad de emitir cheques, un funcionamiento de ejemplo sería:

```
CheckingAccount checkingAccount = new CheckingAccount("1234", 100) //Descubierto 100
checkingAccount.deposit(100) #Saldo 100
SavingsAccount savingsAccount = new SavingAccount("5678")
CheckingAccount otherAccount = new CheckingAccount("9012")
Check check1 = checkingAccount.issueCheck(100)
savinsAccount.depositCheck(check1) //Saldo 100
checkingAccount.getBalance() //100
otherAccount.depositCheck(check1)  //error! (check already performed)
Check check2 = checkingAccount.issueCheck(101)
otherAccount.performCheck(check2) //error! (insuficient founds)
```

Ejercicio 9

Modificar el punto anterior y modelar las cuentas a través de una clase abstracta.

Ejercicio 10

Agregue un nuevo tipo de cuenta, **SpecialAccount** que cuando se realice un depósito, o se deposite un cheque, debe sumar 5 puntos, que podrían ser utilizados para canjear por premios.