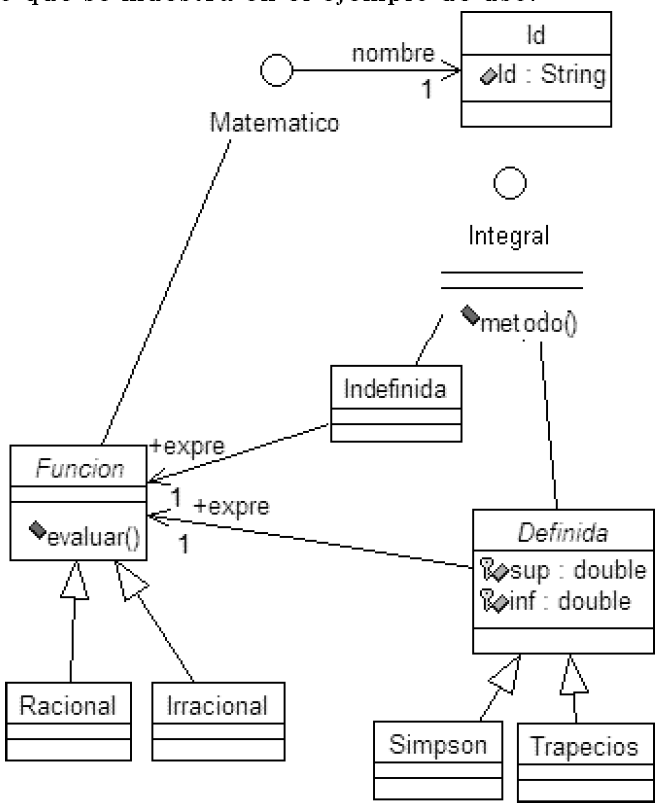
>H ola profe: estamos repasando de unos exámenes viejos nos topamos coneste ejercicio, y le

>pedimos si nos puede ayudar a comprender algunos

**Implementar el modelo de acuerdo a lo especificado. Debe funcionar para el modo que se muestra en el ejemplo de uso.**



**class Id:** es una especialización de la clase

Hastable<K, V> donde K = Integer, V = String.

**class Funcion:** es abstracta, al crear un objeto de

alguna subclase, inserta en el contenedor Id un

objeto<K, V> donde K = identificador generado con

hashcode() y V = expresión funcional.

**class Definida:** es abstracta.

**metodo():** regresa una cadena con la identidad del

objeto, los valores de los atributos del objeto

Integral según corresponda y la expresión

funcional asociada.

**evaluar():** recupera desde el objeto Id la

expresión funcional correspondiente y regresa

dicha cadena.

**Ejemplo de uso:**

Funcion x, y;

Integral a;

x = new Racional("x\*x");

y = new Irracional("sin(x)");

a = new Simpson(5.1, 2.6, x);

System.out.println(a.metodo());

a = new Trapecios(0.928, 0.9999, y);

System.out.println(a.metodo());

a = new Indefinida(x);

System.out.println(a.metodo());

**Salida:**

Simpson@c3c749 (5.1,2.6) x\*x

Trapecios@1690726 (0.928,0.9999) sin(x)

Indefinida@9931f5 x\*x

**Este ejercicio ya deberían estar en condiciones de hacerlo solos. Van algunos tips para ir encarándolo, pero les sugiero que lo realicen nuevamente desde cero.**

Resolución:

1. El orden para comenzar a crear clases e interfaces no tiene importancia, mientras esté todo correctamente definido.

El ejercicio pide trabajar en unos de sus puntos sobre un Hashtable. Recordar que es un tipo particular de contenedor, donde cada una de sus celdas guarda un par de datos, el objeto de interés, y una clave de búsqueda asociada a este. Se guardan objetos con el método hashtable.put(clave , objeto), y se leen brindando su clave de búsqueda, por ej: hashtable.get(“1”), donde en este caso “1” es la clave de búsqueda para acceder al objeto relacionado con esta.

El ejercicio pide crear **Id** que es una especialización (herencia) de Hashtable., como una clase parametrizada.

***class Id extends Hashtable<Integer, String>{***

***}***

1. Crear una clase Funcion (abstracta). Esta función abstracta también debe implementar la interface *Matematico* (ver UML). El método evaluar lo calificamos como abstracto, y luego las clases que hereden de función deberán implementar el código. Función tiene un atributo del tipo Id llamado *nombre*, que además es un contenedor (ver UML) Se accede a este a través de la herencia.

*interface Matematico {*

*Id nombre = new I d ( ) ; //esto es una subclase de Hashtable !*

*}*

abstractclass Funcion implements Matematico{

public Funcion(String obj ) {

nombre.put(thi s .hashCode( ) , obj ) ; //nombre es un subtimo de Hashtable

//luego, podemos agregar un par (clave, objeto) con put.

//obj es el objeto que ingresamos

//y la clave es el hashcode que generamos a partir del

//objeto con el que estamos trabajando

}

public abstractString evaluar(); //este método lo calificamos como abstracto

}

1. Crear la clase Definida como abstracta. Implementa la interfaz Integral (ver UML). Contiene dos atributos principales del tipo doublé y protegidos (pueden ser heredados).

En el UML se observa una relación de asociación (uso) función. Esto implica que declaramos en Definida un atributo publico que se llama expre y es del tipo Funcion.

Hay que incluir en la clase el método *método*(). Este método regresa una cadena con la identidad del objeto, y los valores que se especifican concatenados.

La identidad la podemos obtener llamando al método hashcode() (heredado de una la clase Object). Luego concatenamos los parámetros, y el resultado que devuelve el método *evaluar* del objeto expre.

abstractclass Definida implements Integral {

protected double sup;

protected double inf ;

public Funcion expre;

public Definida(double para1 , double para2 , Funcion f1) {

sup = para1 ;

inf = para2 ;

expre = f1 ;

}

public String metodo( ) {

return hashCode() + " " + inf + " " + sup + " " + expre.evaluar () ;

}

}