Asignatura: **DISEÑO Y PROGRAMACION ORIENTADO A AOBJETOS**

Carrera: **Ingeniería Informática**

Tipo de Curso: **CRD** Año: **1er.** Semestre: **2do.**

**TEMA # 1:** Conceptos Básicos de la POO

**CLASE PRACTICA # 1:** Identificación y definición de clases.

#### Contenido

* Identificación de clases
* Definición de sus responsabilidades
* Definición de sus miembros de datos y de funciones miembros
* Modelación en UML
* Propiedades de la POO. Encapsulamiento.

#### Bibliografía

* [Eck]: Eckel, Bruce. “Thinking in Java”
* [Bud]: Bud, Timothy. “Introducción a la Programación orientada a Objeto”.
* [Zuk]: Zukowski, John. Programación en Java 2 J2SE 1.4
* [Die]: Dietel, H. M. ”Como programar en C/C++”.
* [Stro]: Stroustrup, Bjarne “The C++ Programming Language”.
* Ayuda en línea del Eclipse 3.2

#### Objetivos

* Identificar clases definiendo sus datos y responsabilidades.
* Modelar clases en ML.
* Implementar clases en Java.

#### Introducción

Revisar a través de preguntas el dominio de los alumnos sobre:

* ¿Qué es una clase y qué es un objeto?
* ¿Qué son los atributos de la clase?
* ¿Qué es un mensaje?
* ¿Qué es encapsulamiento y cómo se logra implementar?
* ¿Cómo se implementa una clase en C++?

Utilizar para ello la discusión de las preguntas de evaluación de la conferencia

#### Desarrollo

Ejercicio 1:

1) Considere que en un problema se necesita manipular rectángulos y círculos. De cada rectángulo se conocen las longitudes de sus lados y se sabe que la longitud mínima que puede tener un lado cualquiera es 1. De los círculos se conocen las coordenadas del centro y el radio, del cual se sabe que la longitud mínima es 1. Identificar las clases necesarias y sus atributos.

1. Modelar las clases identificadas en UML.
2. Definir las clases identificadas en Java garantizando los métodos que permitan:
3. hallar el perímetro/longitud de la circunferencia.
4. hallar el área.
5. no violar el encapsulamiento.
6. Declare y construya una instancia de cada clase.
7. Solicitar a las instancias un servicio a través del paso de mensajes.
8. Construir un nuevo rectángulo cuyos lados tengan igual longitud que el radio del círculo construido en c).

Circle

**float** x;

**float** y;

float radius;

**float** area()

**float** longitud()

**float** getX()

**void** setX (**float** pX)

**float** getY()

**void** setSideB(**float** pY)

**float** getRadius()

**void** setRadius(**float** pRadius)



Rectangle

**float** sideA;

**float** sideB;

**float** area()

**float** perimeter()

**float** getSideA()

**void** setSideA(**float** pSideA)

**float** getSideB()

**void** setSideB(**float** pSideB)

**b)**

**public** **class** Rectangle {

**private** **float** sideA;

**private** **float** sideB;

**public** Rectangle(){ // todavia no se ha dado el tema de constructores}

**public** **float** area(){

**return** sideA \* sideB;

}

**public** **float** perimeter(){

**return** 2\*(sideA + sideB);

}

**public** **float** getSideA(){

**return** sideA;

}

**public** **void** setSideA(**float** pSideA) {

**if** (pSideA >= 1)

sideA= pSideA;

}

**public** **float** getSideB(){

**return** sideA;

}

**public** **void** setSideB(**float** pSideB) {

**if** (pSideB >= 1)

sideB= pSideB;

}

}

**c)** Rectangle rectangle; //declarando una referencia a instancia

rectangle= **new** Rectangle(); //construyendo una instancia referenciada por *rectangle*

Circle circle = new Circle();

**d)** rectangle.setSideA(20); //actualizando el lado A con valor 20

rectangle.setSideB(10); //actualizando el lado B con valor 10

**float** p= rectangle.perimeter(); //solicitándole al objeto su perímetro

circle.setX(5);

circle.setY(1);

circle.setRadius(3);

**e)** Rectangle nuevoRect ;

nuevoRect = new Rectangle();

float radio = circle.getRadius();

nuevoRect.setSideA(radio); // nuevoRect.setSideA(circle.getRadius());

nuevoRect.setSideB(radio); // nuevoRect.setSideB(circle.getRadius());

**Ejercicio # 2**

En una cierta competencia de atletismo participan un grupo de deportistas y se necesita manipular la información correspondiente a un deportista, del cual se conoce: nombre, edad, el deporte en que compite, así como, su mejor tiempo en cada uno de los últimos 5 años. Identificar las clases necesarias y sus atributos.

1. Modele las clases identificadas en UML
2. Defínalas en Java garantizando métodos que permitan:
   1. No violar el encapsulamiento.
   2. Verificar si el promedio de sus tiempos anuales está por debajo de un valor record dado.
   3. Conocer su mejor registro
3. Declare y construya una instancia de cada clase.
4. Solicitar a las instancias un servicio a través del paso de mensajes.

*Respuesta:*

**int** Id;

**int** age;

String name;

**float**[] records; //diferente a C++

String sport;

**boolean** bestRecordMax;

SportMan

**int** getAge()

**void** setAge(**int** pAge)

**float** average()

**bool** compareRecord(**float** value)

**int** getId()

String getName()

**void** setName(String pName)

**float** getRecord(**int** year)

**void** setRecord(**int** year, **float** record )

String getSport()

**void** setSport(String pSport)

**void** setbestRecordMax()

**void** setbestRecordMin()

**float** maxRecord()

**float** minRecord()

**float** BestRecord()

b)

**package** sporTMan;

**public** **class** SportMan {

// campos

**private** **int** Id;

**private** **int** age;

**private** String name;

**private** **float**[] records;

**private** String sport;

**private** **boolean** bestRecordMax; //true si el mejor record es el mayor

// métodos

**public** SportMan(){ // no ver el constructor porque no se ha dado

records= **new** **float**[5];

bestRecordMax= **false**; }

**public** **float** average(){

**float** sum= 0;

**for** (**int** i=0; i < records.length; i++)

sum += records[i];

**return** sum / 5;

}

**public** **boolean** compareRecord(**float** pValue) {

**return** pValue > average();

}

**public** **int** getId(){

**return** Id;

}

**public** **void** setId(**int** pId) {

Id= pId;

}

**public** **int** getAge(){

**return** age;

}

**public** **void** setAge(**int** pAge) {

age= pAge;

}

**public** String getName(){

**return** name;

}

**public** **void** setName(String pName) {

name= pName;

}

**public** **float** getRecord(**int** pYear) {

**return** (pYear >=1 && pYear <= 5)? records[pYear-1] : -1;

}

**public** **void** setRecord(**int** pYear, **float** pRecord) {

**if** (pYear >=1 && pYear <= 5)

records[pYear-1]= pRecord;

}

**public** String getSport() {

**return** sport;

}

**public** **void** setSport(String pSport) {

sport= pSport;

}

**public** **void** setBestRecordMax(){

bestRecordMax= **true**;

}

**public** **void** setBestRecordMin(){

bestRecordMax= **false**;

}

**private** **float** MaxRecord() {

**float** max= records[0];

**for**(**int** i= 1; i < records.length; i++)

**if** (records[i] > max)

max= records[i];

**return** max;

}

**private** **float** MinRecord() {

**float** min= records[0];

**for**(**int** i= 1; i < records.length; i++)

**if** (records[i] < min)

min= records[i];

**return** min;

}

**public** **float** BestRecord(){

**return** bestRecordMax? MaxRecord(): MinRecord();

}

}

c) Un ejemplo de clase que usa la clase SportMan

**public** **class** Main {

**public** **static** **void** main(String[] args) { //programa principal

SportMan sportMan = **new** SportMan();//creando un deportista

sportMan.setName("juan"); //asignándole nombre al deportista

sportMan.setRecord(1,1); /\*poniendo el mejor record del primero de

los últimos 5 años\*/

sportMan.setRecord(2,2); /\*poniendo el mejor record del segundo de

los últimos 5 años \*/

sportMan.setRecord(3,3);

sportMan.setRecord(4,4);

sportMan.setRecord(5,5);

sportMan.setBestRecordMax();//el mejor record es el mayor valor

System.*out*.println(sportMan.average()); /\*mostrando su mejor record

Promedio\*/

System.*out*.println(sportMan.getName()); //mostrando su nombre

System.*out*.println(sportMan.getAge()); //mostrando su edad

System.*out*.println(sportMan.BestRecord()); /\*mostrando su mejor

record en los últimos 5 años\*/

}

}

Explicar los siguientes aspectos:

* No se hace un método que permita cambiar el Id de un deportista, pues esto es lo que lo identifica, por lo que debe crearse con un Id que se mantiene fijo durante toda su existencia. Como el tema de constructores se imparte en la segunda conferencia, quedará pendiente la creación con inicialización de miembros. Ahora estamos suponiendo que el constructor inicializa los campos en los valores por defecto.
* Explicar las diferencias de la declaración y uso de arreglos en C++ y Java.
* Insistir en que no siempre los métodos son públicos, por ejemplo: MinRecord() y MaxRecord()
* Insistir en que los campos no se pasan por parámetros a los métodos.
* Reforzar la idea de que el encapsulamiento se logra por una adecuada especificación de visibilidad más la incorporación de métodos de lectura (get) y escritura (set) de todos los campos cuyos valores puedan ser requeridos o modificados por los usuarios de la clase.
* Explicar la clase Main. Este es un contenido no estudiado en la conferencia. Los modificadores de durabilidad no se explicaron

**Ejercicio # 4**

Cree una clase para manipular números fraccionarios y ejecutar aritmética con fracciones. Garantice que:

- el denominador nunca sea cero

- se almacene la fracción en forma simplificada, es decir, 2/4 debe almacenarse como 1/2

- proporcione funciones para: sumar dos fracciones, restar dos fracciones, multiplicar dos fracciones, dividir dos fracciones, sumar fracciones y enteros, restar fracciones y enteros, multiplicar fracciones por enteros y dividir fracciones y enteros.

*Respuesta:*

RationalNumber

**int** num**;**

**int** den**;**

**int** minComunMultiply**(int** num1, **int** num2**)**

**int** MaximoComunDivisor**(int** m,**int** n**)**

RationalNumbersimplificated**(int** numerator**, int** denominator**)**

**int** getNum()

**void** setNum(**int** pNum)

**int** getDen()

**void** setDen(**int** pDen)

RationalNumber addition(RationalNumber ratNumb)

RationalNumber addition(**int** numb)

**void** add(RationalNumber ratNumb)

**void** add(**int** numb)

RationalNumber subtraction(RationalNumber ratNumb)

RationalNumber subtraction(**int** numb)

**void** subtract (RationalNumber ratNumb)

**void** subtract (**int** numb)

RationalNumber multiplication(RationalNumber ratNumb)

RationalNumber multiplication(**int** nNumb)

**void** multiply(RationalNumber ratNumb)

**void** multiply(**int** numb)

RationalNumber division(RationalNumber ratNumb)

RationalNumber division(**int** numb)

**void** divide(RationalNumber ratNumb)

**void** divide(**int** numb)

**package** Fraccion;

**public** **class** RationalNumber {

**private** **int** num;

**private** **int** den;

**private** **int** minComunMultiply(**int** num1, **int** num2){

//devuelve el mínimo común múltiplo entre dos números

**if**(num1 == num2)

**return** num1;

**int** menor, mayor;

**if**(num1 > num2){

menor= num2;

mayor= num1;

}

**else**{

menor= num1;

mayor= num2;

}

**int** i= mayor;

**while**(i < menor \* mayor && (i%menor != 0 || i%mayor != 0))

i++;

**return** i < menor \* mayor? i: menor \* mayor;

}

**private** **int** MaximoComunDivisor(**int** m,**int** n){

//Algoritmo de Euclides

//Devuelve el máximo común divisor o -1 si no es posible encontrarlo

**if** (m >= 0 && n >= 0){

**if** (n > m){

**int** temp= m;

m= n;

n= temp;

}

**int** R;

**while** (n != 0){

R = m % n;

m = n;

n = R;

}

**return** m;

}

**else**

**return** -1;

}

**private** RationalNumber simplificated(**int** numerator, **int** denominator){

//obtiene la fracción simplificada a su mínima expresión

RationalNumber result= **new** RationalNumber();

**int** maxComDiv= MaximoComunDivisor(numerator,denominator);

**if**(maxComDiv == -1 || maxComDiv == 1){

result.setNum(numerator);

result.setDen(denominator);

}

**else**{

result.setNum(numerator/maxComDiv);

result.setDen(denominator/maxComDiv);

}

**return** result;

}

**public** **int** getNum(){

**return** num;

}

**public** **void** setNum(**int** pNum){

num= pNum;

}

**public** **int** getDen(){

**return** den;

}

**public** **void** setDen(**int** pDen){

**if** (pDen != 0)

den= pDen;

}

**public** String fraction(){

**return** ((Integer)num).toString()+'/'+((Integer)den).toString();

}

**public** RationalNumber addtion(RationalNumber ratNumb){

**int** minComunDenominator= minComunMultiply(den,ratNumb.getDen());

**int** numerator= minComunDenominator/den\*num +

minComunDenominator/ratNumb.getDen()\*ratNumb.getNum();

**return** simplificated(numerator,minComunDenominator);

}

**public** RationalNumber addition(**int** number){

**int** numerator= num + number\* den;

**return** simplificated(numerator,den);

}

**public** **void** add(RationalNumber ratNumb){

**int** minComunDenominator= minComunMultiply(den,ratNumb.getDen());

**int** numerator= minComunDenominator/den\*num +

minComunDenominator/ratNumb.getDen()\*ratNumb.getNum();

RationalNumber temp= simplificated(numerator,minComunDenominator);

num= temp.getNum();

den= temp.getDen();

}

**public** **void** add(**int** number){

**int** numerator= num + number\* den;

RationalNumber temp= simplificated(numerator,den);

num= temp.getNum();

den= temp.getDen();

}

**public** RationalNumber subtraction(RationalNumber ratNumb){

**return** **null**;

}

**public** RationalNumber subtraction(**int** numb){

**return** **null**;

}

**public** **void** subtract(RationalNumber ratNumb){

}

**public** **void** subtract(**int** numb){

}

**public** RationalNumber multiplication(RationalNumber ratNumb){

**return** **null**;

}

**public** RationalNumber multiplication(**int** nNumb){

**return** **null**;

}

**public** **void** multiply(RationalNumber ratNumb){

}

**public** **void** multiply(**int** numb){

}

**public** RationalNumber division(RationalNumber ratNumb){

**return** **null**;

}

**public** RationalNumber division(**int** numb){

**return** **null**;

}

**public** **void** divide(RationalNumber ratNumb){

}

**public** **void** divide(**int** numb){

}

}

Observaciones:

* No es necesario implementar la clase completa. Se sugiere implementar sólo los métodos privados y los de adición y dejar propuesto como estudio independiente la implementación del resto.
* Este ejercicio ofrece buenas posibilidades para ejercitar la lógica de programación.
* Permite trabajar con más de una instancia, específicamente en los métodos que esperan una instancia del mismo tipo. Esto refuerza el conocimiento sobre el acceso a los miembros de instancia a través de sus métodos y permite desarrollar habilidades en ello.
* Permite apreciar la necesidad de crear instancias en un método.
* El algoritmo de Euclide aparece explicado e implementado en el documento del taller de repeticiones en IP. Es muy posible que no se haya trabajado en clase, pero estaba en el documento con las soluciones.

#### Conclusiones

Insistir en que la POO no consiste simplemente en unas cuantas características nuevas añadidas a los lenguajes de programación, es más que eso: es una nueva forma de pensar.

La POO considera un programa como a una colección de objetos, viéndose éstos como una cápsula donde coexisten sus datos y comportamiento.

#### Estudio independiente

Analice las siguientes situaciones y encuentre las categorías de objetos o clases que deben modelarse para la solución de las mismas. Defina según su criterio cuales podrían ser los datos y las operaciones o responsabilidades de cada una de ellas relacionadas con la solución del problema. Implemente las clases en Java.

1.- El centro recreativo “La Giraldilla” necesita automatizar su almacén de vinos. A partir de las entrevistas realizadas a sus trabajadores se obtuvieron las siguientes informaciones: cada vino tiene un nombre, una cosecha, un tipo (tinto, blanco y rosado) y una disponibilidad. Además, se conoce el valor de las ventas de los últimos 10 años y el suministrador del cual se conoce, nombre que lo identifica, país y tiempo que demora la entrega. A petición del administrador del complejo hay que decidir hacer nuevos pedidos a los suministradores del vino dependiendo de si las ventas de los últimos 5 años superan el promedio de las ventas realizadas durante los primeros 5 años y el plazo de entrega es menor que 30 días.

a) Identificar las clases y sus atributos que permitan resolver la situación anterior

b) Defínalas en Java

Implementar los métodos que permitan:

1. ayudar al administrador a decidir si hace un nuevo pedido de un vino en particular.

2.- Una estación territorial del Instituto de Meteorología necesita automatizar algunos cálculos semestrales que realiza relacionados con la lluvia caída en ese territorio. Se dispone de la información siguiente: cantidad de lluvia caída en cada uno de los meses del semestre así como el valor de la lluvia histórica en el semestre estudiado. La estación necesita poder determinar el promedio de lluvia caída en el semestre así como el mayor valor de lluvia caída mensualmente en el semestre y la cantidad de meses en los cuales la cantidad de lluvia caída supera el valor de lluvia histórica.

a) Identificar las clases y sus atributos que permitan resolver la situación anterior

b) Defínalas en Java.

Implemente los métodos que permitan:

1. los métodos que permitan darle solución a las necesidades de la estación.