USO DE UN LENGUAJE DE PROGRAMACIÓN DE ALTO NIVEL PARA LA IMPLEMENTACIÓN DE SOLUCIONES COMPUTACIONALES EMPLEANDO PROGRAMACIÓN ORIENTADA A OBJETOS

Maria C. Torres Madroñero
Profesora asociada
Departamento de Ciencias de la Computación y la Información

Objetivos

- Comprender el proceso de diseño basado en objetos para la solución de problemas
- Identificar los procesos para crear y usar objetos en un lenguaje de programación de alto nivel

Recursos requeridos

- PC
- IDE para JAVA o Python – el estudiante deberá seleccionar un lenguaje de programación para el desarrollo de las prácticas de laboratorio

Actividades preliminares al laboratorio

- Lectura de la guía

Marco teórico

Programación orientada a objetos: La programación orientada a objetos (POO) es un paradigma de programación, es decir, un estilo o forma de estructurar un programa. En POO, los objetos manipulan los datos de entrada para la obtención de los datos de salida, cada objeto tiene una funcionalidad especifica.

¿Qué es un objeto?

Los objetos representan cosas, que pueden ser simples o complejas, pueden ser reales o imaginarios. Cada objeto tiene unas características o valores denominados atributos, los cuales permiten definir el estado del objeto. Adicionalmente, cada objeto tiene asociadas las acciones que puede realizar definidas a través de métodos, los cuales se ejecutan de acuerdo con los parámetros o argumentos de entrada y pueden devolver un valor al acabar su ejecución.

Ejemplo:

• Atributos contantes: Marca, color, potencia, velocidad máxima.
• Atributos variables: velocidad, aceleración
• Métodos: arrancar, parar, acelerar, frenar, girar a la derecha, girar a la izquierda

¿Qué es una clase?

Las clases representan un tipo particular de objetos, agrupando objetos con características y comportamiento similares. Ejemplo clase carro:

Cada clase tiene asociado una definición que determina:
• Los atributos que tienen los objetos de la clase
• Los métodos que pueden ejecutar los objetos de la clase y como lo hacen

La POO consiste en diseñar e implementar las clases de objetos que permiten la solución de un problema. Un programa en POO consta de un conjunto de instancias de una clase (objetos) y un flujo de control principal (main). Durante la ejecución del programa los objetos se crean y se destruyen, y se solicita a los objetos que ejecuten métodos.

Metas de POO

• Robustez: un programa se considera robusto cuando es capaz de manejar una entrada inesperada. En general, un programa funciona correctamente, produciendo la salida correcta dada una entrada esperada. Sin embargo, en el caso de entradas no esperadas (por ejemplo, ingresan caracteres en vez de enteros), el programa debe dar una respuesta.
• Adaptabilidad: un programa debe ser capaz de adaptarse en el tiempo al cambio de condiciones. Un concepto relacionado a la adaptabilidad es la portabilidad, la cual es la habilidad de un programa para correr con cambios mínimos en diferentes hardware o sistemas operativos.
• Reusabilidad: se espera que le mismo código pueda ser usando como un componente de diferentes sistemas en diversas aplicaciones.

Principios de POO

• Abstracción: se refiere a dividir un sistema complejo en sus partes más fundamentales. Definir las partes de un sistema incluye nombrarlas y explicar su funcionalidad.
• Encapsulación: los componentes de un sistema no deben revelar detalles internos de sus respectivas implementaciones. Una de las principales ventajas de la encapsulación es que un programados tiene la libertad de implementar un componente, solo debe mantener la interfaz (como se relaciona con otros componentes). Esto permite la robustez y adaptabilidad de los programas.
• Modularidad: principio de organización en el cual los diferentes componentes de un programa se dividen en diferentes unidades funcionales.

Diseño orientado a objetos

En el diseño se especifica la estructura de como un programa o sistema sera escrito y funcionara, sin escribir la implementación completa. El diseño es una transición de “que” debe hacer un sistema a “como” lo va a hacer, respondiendo las siguientes preguntas:

• ¿Qué clases se requieren?
• ¿Cuáles son los campos y métodos de cada clase?
• ¿Cómo cada clase interactúa con otras clases?

Caso de estudio: Sistemas de información para una biblioteca

> Una biblioteca desea tener un sistema de información para administrar su colección bibliográfica. Cada item de la biblioteca (ejemplo: libros, revistas, cds, etc.) tiene un identificador y pueden existir varios ejemplares de un mismo item. El staff de la biblioteca puede prestar hasta 6 items por 15 días, y un usuario registrado puede prestar hasta 3 items por 10 días. Tanto del staff como de los usuarios registrados se almacena en el sistema su información de contacto (nombre, id, teléfono, email).

Para identificar las clases requeridas para el diseño del sistema información se puede seguir los siguientes pasos:

1. Identificar los sustantivos en la descripción
2. Identificar los verbos
3. A partir de los sustantivos definir las clases y los atributos de las clases

4. A partir de los verbos definir los métodos de las clases

Clase	Atributos	Métodos
Biblioteca	• Colección	• Prestar • Devolver
Item	• Id • Num_ejemplares	• Prestar • Devolver
Usuario	• Nombre • Id • Tel • Email • Limite_items • Limite_dias	• Prestar • Devolver

Construcción de diagrama de clase UML

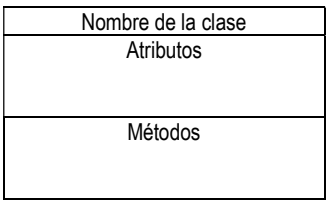
UML es un lenguaje de modelamiento unificado (Unified modeling lenguage). Un esfuerzo de mediados de los 90 para promover la programación orientada objetos. UML integra el uso de diagramas de caso, diagramas de clase, diagramas de objetos, diagramas de secuencias, diagramas de colaboración entre otros.

En este curso haremos uso de los diagramas de clase UML para la representación gráfica de las clases.

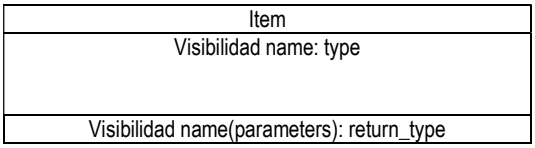
Los diagramas de clase UML incluyen:
• Los atributos y los métodos
• Conexiones entre clases: interacción y herencia

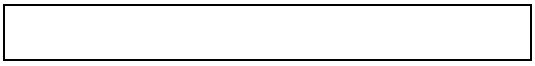
Los diagramas de clase UML no incluyen:
• Detalles de cómo interactúan las clases
• Detalles de los algoritmos que constituyen cada método

Ejemplo de diagrama de clase:

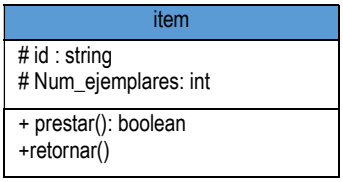


• El nombre de la clase sigue las mismas recomendaciones para nombrar variables y métodos.
• Los atributos deben incluir la visibilidad, el nombre y el tipo
  - Visibilidad:
    + public: accesible a todos los objetos en su sistema
    # protected: accesible a la clase y sus subclases
    - private: accesible únicamente a la clase
• Operaciones/métodos: Los métodos deben incluir la visibilidad, el nombre del método, los parámetros de entrada y el tipo de rato que retorna.





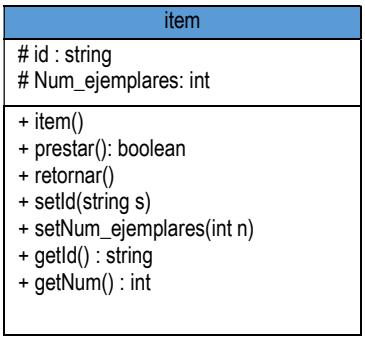
Ejemplo diagrama de clases para item del Sistemas de información para una biblioteca:



Settings y Gettings

• Para seguir el principio de encapsulamiento, usualmente los atributos de una clase se declaran privados o protegidos. Por tanto, para modificarlos y conocer su estado es necesario incluir los métodos que permitan hacer esta tarea, estos métodos se denominan settings y gettings.
• Los settings permiten modificar el estado de los atributos, es decir cambiar su valor.
• Los gettings permiten conocer el estado de los atributos, es decir retornar el valor actual.
• En el momento de diseñar la clase es necesario determinar la visibilidad de los atributos, ya que esto define la inclusión de los setting y gettings o su omisión.

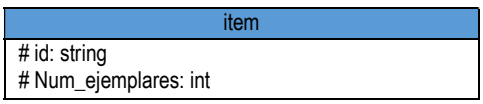
Ejemplo settings y gettings para clase item del Sistemas de información para una biblioteca



Constructores

Los constructores son una clase especial de método usado para crear e inicializar una nueva instancia de una clase, es decir, crear un nuevo objeto. Este método típicamente realiza la inicialización de cada atributo del objeto. Para definir el constructor de una clase se debe tener en cuenta:
• El nombre del constructor debe ser idéntico al nombre de la clase
• No se especifica un tipo de dato de retorno
• Una clase puede tener más de un constructor, pero estos deben variar en tipo y número de parámetros

Ejemplo de constructores para la clase item del Sistemas de información para una biblioteca



+ item() + item(string s) + item(string s, int n) + prestar(): boolean + retornar() + setId(string s) + setNum_ejemplares(int n) + getId() : string + getNum() : int

Actividades

Registro de usuarios: En diferentes aplicaciones requerimos administrar información personal y de contacto de usuario. En este laboratorio vamos a diseñar una clase que permita administrar esta información, es decir, permita inicializar y editar los datos personales y de contacto de un usuario. El diagrama de clase para Usuario se presenta a continuación:

Usuario	
- nombre: String - id: long - fecha_nacimiento: Fecha - ciudad_nacimiento: String	- tel: long - email: String - dir: Direccion
+ Usuario() + Usuario(String n, long id) + setNombre(String n) + setId(long id) + setFecha_nacimiento(Fecha f) + setCiudad_nacimiento(String c) + setTel(long t) + setEmail(String e) + setDir(Direccion d)	+ getNombre():String + getId():long + getFecha_nacimiento():Fecha + getCiudad_nacimiento():String + getTel():long + getEmail():String + getDir():Direccion + toString(): String

La clase presentada en el diagrama anterior hace uso de dos clases: Fecha y Dirección; estas clases permiten administrar respectivamente: una fecha en el formato día-mes-año y la dirección de contacto. La dirección incluye datos de la calle, nomenclatura, barrio, ciudad y alternativa incluye el nombre del edificio/urbanización y el número de apartamento. Por ejemplo, la dirección de la sede Fraternidad del ITM es: Calle 54A No. 30-01 Barrios Boston. En el formato que estamos utilizando para la dirección, la sede Fraternidad correspondería a:

- calle: Calle 54A
- nomenclatura: 30-01
- barrio: Boston
- ciudad: Medellin
- edificio: Null
- apto: Null

Los diagramas de clase para Fecha y Direccion se presentan a continuación:

Fecha
- dd: short - mm: short - aa: short
+ Fecha() + Fecha(short dd, short mm, short aa) + setDia(short dd) + setMes(short mm) + setA (short aa) + getDia(): short + getMes(): short + getA(): short

Direccion	
- calle: String	- ciudad: String
- nomenclatura: String	- edificio: String
- barrio: String	- apto: String
+ Direccion()	+ getCalle():String
+ setCalle(String c)	+ getNomenclatura():String
+ setNomenclatura(String n)	+ getBarrio():String
+ setBarrio(String b)	+ getCiudad():String
+ setCiudad(String ci)	+ getEdificio():String
+ setEdificio(String e)	+ getApto():String
+ setApto(String a)	+ toString():String

Sobre método toString(): como podrá notar, tanto la clase Usuario como la clase Dirección presentada en los anteriores diagrama de clase, incluye un método denominado toString(). Este método lo incluiremos en algunas de nuestras clases para facilitar la impresión de la información en la pantalla (incluso más adelante para escribir archivos). Este método debe retornar una cadena con la información de la clase organizada. Por ejemplo, el método toString() de la clase Usuario retorna la información personal y de contacto de un usuario en especificio, en una sola línea de texto. En el caso de la clase Direccion, toString() retorna la información de la dirección concatenada.

1. Implementar la clase Fecha
2. Implementar la clase Dirección
3. Implementar la clase Usuario
4. En un programa principal:
    - Crear un objeto tipo fecha con la información de su fecha de nacimiento e imprimir en pantalla los datos en el formato dd – mm – aa.
    - Crear un objeto tipo dirección con su dirección de residencia e imprimir en pantalla los datos en una sola cadena.
    - Crear un objeto tipo Usuario con su información personal y de contacto, imprimir en pantalla todos los atributos del objeto creado.
5. En un programa principal:
    - Solicite por consola la información personal y de contacto requerida para construir un Usuario
    - Cree un objeto empleando la información ingresada por consola, e imprima en pantalla todos los atributos del objeto.

Instrucciones de entrega

- La solución de los problemas debe desarrollarse en JAVA o Python. Los estudiantes tendrán la libertad de seleccionar el lenguaje de programación y plataforma para presentar la solución de los problemas.
- La solución debe emplear librerías nativas y se invita a los estudiantes a no usar código descargado de internet. Los laboratorios están diseñados para practicar los fundamentos teóricos; entre más código escriba el estudiante más fácil será su comprensión de los temas de clase.
- La solución se puede presentar en grupos de hasta 3 estudiantes.
- La solución de los problemas debe entregarse y sustentarse en el aula de clase o en la hora de asesoría a estudiantes. No se reciben soluciones por correo electrónico.