ESCUELA
POLITÉCNICA
NACIONAL

Escuela Politécnica Nacional

Facultad de Ingeniería de Sistemas

Sección teórica

Trabajo Final

Materia:
Programación II
ICCD244

Autor:
Mathias Mogrovejo

Carrera:
Ingeniería en Software

Profesor:
Dr. Jaime Sayago.

Quito, Ecuador
21 de noviembre de 2025

1. **FOUNDATIONAL JAVA - PEARSON**

Ejercicio 6.3.

¿Cómo sería un diseño más completo para la clase CourseDelivery?

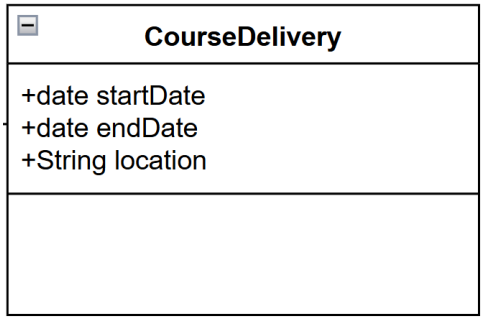


Figura 1 (Autoría). Diseño completo para la clase CourseDeilvery

• ¿Qué otros atributos podría tener?

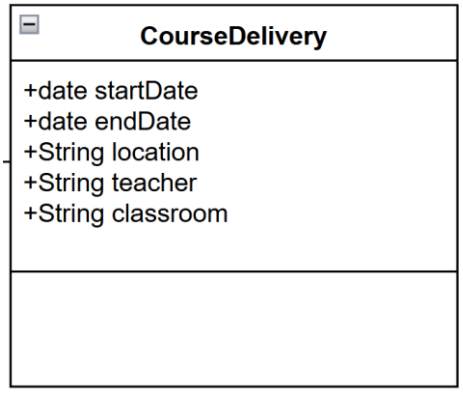


Figura 2 (Autoría). Atributos extra para la clase CourseDeilvery

• Actualmente, el diagrama de la figura 6.10 no muestra ningún método. ¿Qué métodos, aparte de los getters y setters, podrían ser adecuados para un CourseDelivery?

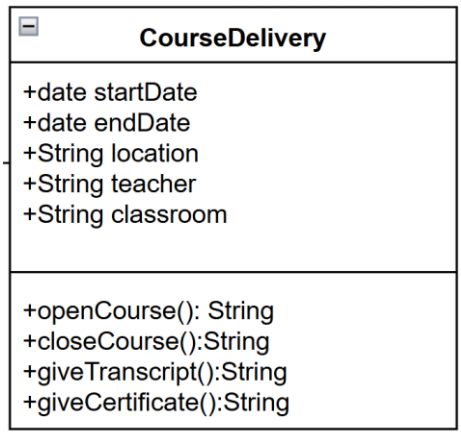


Figura 3 (Autoría). Métodos para la clase CourseDeilvery

• Hemos sugerido que las clases Course y CourseDelivery podrían estar asociadas. ¿Puedes sugerir otras clases que puedan ser relevantes para este dominio y que podrían estar asociadas con las clases Course o CourseDelivery?

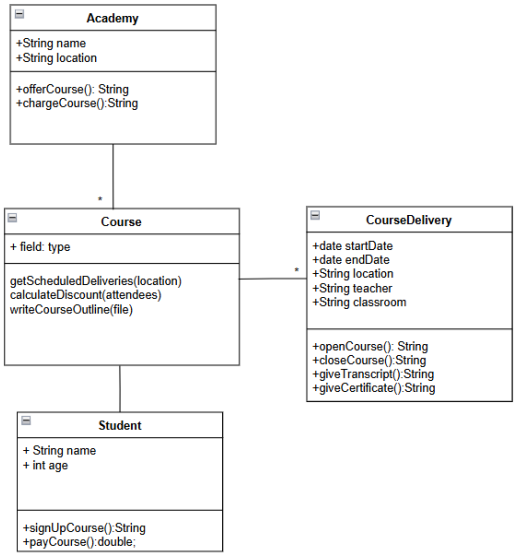


Figura 5 (Autoría). Clases extra asociadas

2. COMO PROGRAMAR EN JAVA - DEITEL

8.1 Indique si el siguiente enunciado es verdadero o falso, y si es falso, explique por qué:

Si un atributo de una clase se marca con un signo menos (-) en un diagrama de clases, el atributo no es directamente accesible fuera de la clase.

Es verdadero, el signo menos (-) se usa para definir que una variable clase o método son privadas, es decir, de uso explicito para la clase donde se aplicó.

8.2 En la figura 8.25, la asociación entre los objetos ATM y Pantalla indica:

a) que podemos navegar de la Pantalla al ATM.

b) que podemos navegar del ATM a la Pantalla.

c) (a) y (b); la asociación es bidireccional.

d) Ninguna de las anteriores.

8.3 Escriba código de Java para empezar a implementar el diseño para la clase Teclado.

```java
package Capitulo8_10.sistemaATMExample;
import java.util.*;
public class Teclado {
    Scanner scanner = new Scanner(System.in);

     // constructor sin argumentos
     public Teclado()
     {
     } // fin del constructor de Teclado sin argumentos

      // operaciones
      public int obtenerEntrada()
      {
          return scanner.nextInt();
      } // fin del método obtenerEntrada

}
```

Figura 6 (Autoría). Código clase Teclado.

EJERCICIO DE AUTOEVALUACIÓN 8.1

Complete los siguientes enunciados:

a) Al compilar una clase en un paquete, la opción ___-d____ de línea de comandos de javac especifica en dónde se debe almacenar el paquete, y hace que el compilador cree los directorios, en caso de que no existan.

b) El método static _____format___ de la clase String es similar al método System.out.printf, pero devuelve un objeto String con formato en vez de mostrar un objeto String en una ventana de comandos.

c) Si un método contiene una variable local con el mismo nombre que uno de los campos de su clase, la variable local ____oculta_____ al campo en el alcance de ese método.

d) El recolector de basura llama al método ____finalize_____ antes de reclamar la memoria de un objeto.

e) Una declaración _____import_____ especifica una clase a importar.

f) Si una clase declara constructores, el compilador no creará un(a) _____constructor_____.

g) El método ____toString_____ de un objeto se llama en forma implícita cuando aparece un objeto en el código, en donde se necesita un String.

h) A los métodos establecer se les llama comúnmente _____get____ o _____metodo de acceso_____.

i) Un método ____predicado_____ evalúa si una condición es verdadera o falsa.

j) Para cada enum, el compilador genera un método static llamado _____values_____, que devuelve un arreglo de las constantes de la enum en el orden en el que se declararon.

k) A la composición se le conoce algunas veces como relación _____tiene un_____.

l) Una declaración ____enum_____ contiene una lista separada por comas de constantes.

m) Una variable ____static_____ representa información a nivel de clase, que comparten todos los objetos de la clase.

n) Una declaración _____static import_____ importa un miembro static.

o) El _____principio de privilegio menor____ establece que al código se le debe otorgar sólo el nivel de privilegio y de acceso que necesita para realizar su tarea designada.

p) La palabra clave _____final____ especifica que una variable no se puede modificar.

q) Un(a) ____dato abstracto_____ consiste en una representación de datos y las operaciones que pueden realizarse sobre esos datos.

r) Sólo puede haber un(a) _____declaracion de paquete_____ en un archivo de código fuente de Java, y debe ir antes de todas las demás declaraciones e instrucciones en el archivo.

s) Un(a) declaración _____import de demanda_____ sólo importa las clases que utiliza el programa de un paquete específico.

t) El compilador utiliza un(a) ____cargador de clases_____ para localizar las clases que necesita en la ruta de clases.

u) La ruta de clases para el compilador y la JVM se puede especificar mediante la opción _____-classpath___ para el comando javac o java, o estableciendo la variable de entorno ____CLASSPATH_____.

v) A los métodos establecer se les conoce comúnmente como _____setters___, ya que, por lo general, modifican un valor.

w) Un(a) _____import *_____ importa a todos los miembros static de una clase.

x) Los métodos public de una clase se conocen también como los _____interfaces public____ o ____servicios public_____ de la clase.

y) El método static _____gc.____ de la clase System indica que el recolector de basura debe realizar su mejor esfuerzo para tratar de reclamar los objetos que sean candidatos para la recolección de basura.

z) Un objeto que contiene ____datos consistentes____ tiene valores de datos que siempre se mantienen dentro del rango.

Ejercicios

8.2 Explique la noción del acceso a nivel de paquete en Java. Explique los aspectos negativos del acceso a nivel de paquete.

El acceso a nivel de paquete es el modificador de acceso por defecto que se aplica cuando no se especifica ningún modificador (public, private o protected) en una clase, método o variable, permitiendo que estos elementos sean visibles y accesibles únicamente dentro del mismo paquete pero no desde clases ubicadas en paquetes diferentes. Los aspectos negativos de este tipo de acceso pueden ser la falta de claridad intencional ya que la ausencia de modificador puede interpretarse como un olvido del programador en lugar de una decisión deliberada; debilita el encapsulamiento porque cualquier clase dentro del paquete puede acceder a estos miembros sin restricciones, violando potencialmente los principios de ocultamiento de información; dificulta el mantenimiento del código al crear dependencias implícitas entre clases del mismo paquete que no son evidentes; y puede generar problemas de seguridad en aplicaciones grandes donde múltiples desarrolladores trabajan en el mismo paquete.

8.3 ¿Qué ocurre cuando un tipo de valor de retorno, incluso void, se especifica para un constructor?

Si se especifica un tipo de retorno para un constructor, incluyendo void, Java no lo reconocerá como un constructor sino como un método común con el mismo nombre que

la clase. Esto significa que el supuesto "constructor" no será invocado automáticamente durante la creación de objetos con la palabra clave new, el compilador buscará el constructor real sin tipo de retorno y si no existe generará un error o usará el constructor por defecto.

8.4-8.19 ver en repositorio

EJERCICIOS DE AUTOEVALUACIÓN 9.1

Complete las siguientes oraciones:

a) _Herencia_ es una forma de reutilización de software, en la que nuevas clases adquieren los miembros de las clases existentes, y se mejoran con nuevas capacidades.

b) Los miembros ___public y protected_ de una superclase pueden utilizarse en la declaración de la superclase y en las declaraciones de las subclases.

c) En una relación _Herencia__, un objeto de una subclase puede ser tratado también como un objeto de su superclase.

d) En una relación ___Composición___, el objeto de una clase tiene referencias a objetos de otras clases como miembros.

e) En la herencia simple, una clase existe en una relación __jerárquica_ con sus subclases.

f) Los miembros ____públicos_____ de una superclase son accesibles en cualquier parte en donde el programa tenga una referencia a un objeto de esa superclase, o a un objeto de una de sus subclases.

g) Cuando se crea la instancia de un objeto de una subclase, el ____constructor____ de una superclase se llama en forma implícita o explícita.

h) Los constructores de una subclase pueden llamar a los constructores de la superclase mediante la palabra clave __super____.

9.2 Conteste con verdadero o falso a cada una de las siguientes proposiciones; en caso de ser falso, explique por qué.

a) Los constructores de la superclase no son heredados por las subclases.

Verdadero

b) Una relación “tiene un” se implementa mediante la herencia.

Falso, la relación de composición es la que expresa un “tiene un”

c) Una clase Auto tiene una relación “es un” con las clases VolanteDireccion y Frenos.

Falso, la relación de “es un” se aplica solo en herencia

d) La herencia fomenta la reutilización de software comprobado, de alta calidad.

Verdadero

e) Cuando una subclase redefine al método de una superclase utilizando la misma firma, se dice que la subclase sobrecarga a ese método de la superclase.

Falso, cuando se redefine al método se llama sobreescritura

EJERCICIOS

9.3 Muchos programas escritos con herencia podrían escribirse mediante la composición, y viceversa. Vuelva a escribir las clases EmpleadoBaseMasComision4 (figura 9.13) de la jerarquía EmpleadoPorComision3-EmpleadoBaseMasComision4 para usar la composición en vez de la herencia. Una vez que haga esto, valore los méritos relativos de las dos metodologías para los problemas de EmpleadoPorComision3 y EmpleadoBaseMasComision4, así como también para los programas orientados a objetos en general. ¿Cuál metodología es más natural? ¿Por qué?

Para el método de la herencia, se puede obtener un beneficio porque es más entendible al poder relacionar en la vida real los métodos y características que ambos tienen en común, mientras que la composición nos sirve como referencia a largo plazo cuando deseemos hacer mantenimiento o comprender las funciones que ejecuta tanto el empleado 3 y el 4, dándonos una perspectiva de antemano sobre sus roles.

9.4 Describa las formas en las que la herencia fomenta la reutilización de software, ahorra tiempo durante el desarrollo de los programas y ayuda a prevenir errores.

La herencia tiene varias aplicaciones que fomenta la reutilización de código, como lo son el paso de parámetros a variables generales de una sola clase y métodos aplicables en cada subclase. Por otro lado, ahorra tiempo en el desarrollo de programas porque nos permite definir claramente cuales son los métodos y variables que debe tener cada objeto con el fin de generar estas características en una superclase y así, automáticamente, se generen los métodos en las otras clases para su sobreescritura. Esto al mismo tiempo ayuda a prevenir errores de ejecución, cuando queramos por ejemplo efectivizar la codificación podremos asumir de manera segura que podemos obtener una característica “x” de todos los elementos.

9.5 Dibuje una jerarquía de herencia para los estudiantes en una universidad, de manera similar a la jerarquía que se muestra en la figura 9.2. Use a Estudiante como la superclase de la jerarquía, y después extienda Estudiante con las clases EstudianteNoGraduado y EstudianteGraduado. Siga extendiendo la jerarquía con el mayor número de niveles que sea posible. Por ejemplo, EstudiantePrimerAnio, EstudianteSegundoAnio, EstudianteTercerAnio y EstudianteCuartoAnio podrían extender a EstudianteNoGraduado, y EstudianteDoctorado y EstudianteMaestria

podrían ser subclases de EstudianteGraduado. Después de dibujar la jerarquía, hable sobre las relaciones que existen entre las clases. [Nota: no necesita escribir código para este ejercicio].

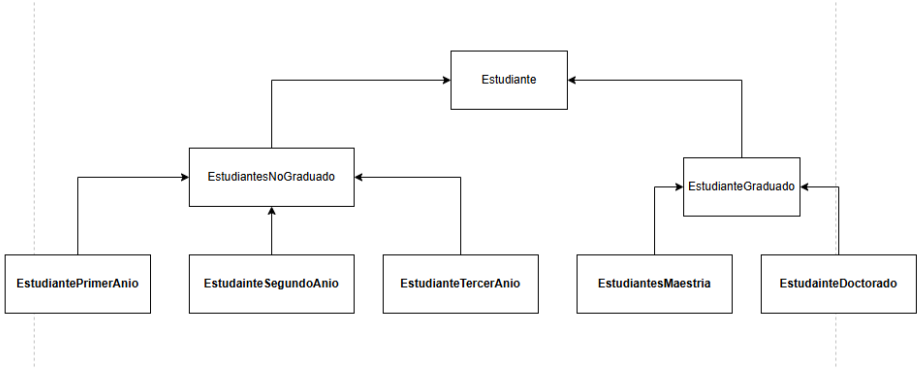


Figura 7 (Autoría). jerarquía clases de estudiantes.

9.6 El mundo de las figuras es más extenso que las figuras incluidas en la jerarquía de herencia de la figura 9.3. Anote todas las figuras en las que pueda pensar (tanto bidimensionales como tridimensionales) e intégrelas en una jerarquía Figura más completa, con todos los niveles que sea posible. Su jerarquía debe tener la clase Figura en la parte superior. Las clases FiguraBidimensional y FiguraTridimensional deben extender a Figura. Agregue subclases adicionales, como Cuadrilatero y Esfera, en sus ubicaciones correctas en la jerarquía, según sea necesario.

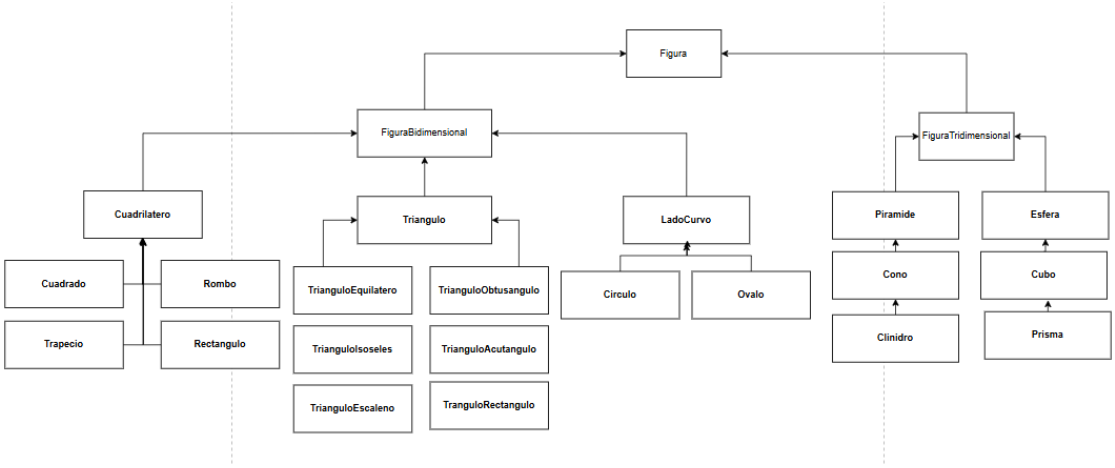


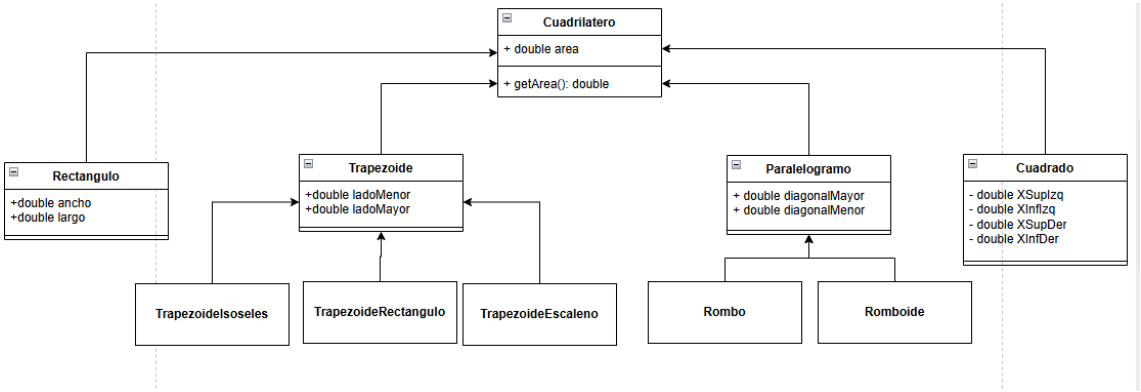
Figura 8 (Autoría). Jerarquia clase figura.

9.7 Algunos programadores prefieren no utilizar el acceso protected, pues piensan que quebranta el encapsulamiento de la superclase. Hable sobre los méritos

relativos de utilizar el acceso protected, en comparación con el acceso private en las superclases.

Al usar el acceso *protected*, se puede aplicar de una mejor manera la reutilización de código, lo cual a largo plazo reduce el tiempo de producción al dejar usar métodos y atributos ya programados en diferentes subclases. Esta ventaja no es posible si usamos el nivel *private*, pues todos estos bloques de programación se limitarían a la misma clase.

9.8 Escriba una jerarquía de herencia para las clases Cuadrilátero, Trapezoide, Paralelogramo, Rectángulo y Cuadrado. Use Cuadrilátero como la superclase de la jerarquía. Agregue todos los niveles que sea posible a la jerarquía. Especifique las variables de instancia y los métodos para cada clase. Las variables de instancia private de Cuadrilatero deben ser los pares de coordenadas x-y para los cuatro puntos finales del Cuadrilátero. Escriba un programa que cree instancias de objetos de sus clases, y que imprima el área de cada objeto (excepto Cuadrilatero).



EJERCICIOS DE AUTOEVALUACION 10.1

10.1 Complete las siguientes oraciones:

a) El polimorfismo ayuda a eliminar la lógica de ___switch___.

b) Si una clase contiene al menos un método abstracto, es una clase ____abstracta____.

c) Las clases a partir de las cuales pueden instanciarse objetos se llaman clases ____concretas___.

d) El ____polimorfismo___ implica el uso de una variable de superclase para invocar métodos en objetos de superclase y subclase, lo cual nos permite "programar en general".

e) Los métodos que no son métodos de interfaz y que no proporcionan implementaciones deben declararse utilizando la palabra clave _____abstract___.

g) Al proceso de convertir una referencia almacenada en una variable de una superclase a un tipo de una subclase se le conoce como ____conversión descendente___.

10.2 Conteste con verdadero o falso a cada una de las siguientes proposiciones; en caso de ser falso, explique por qué.

a) Es posible tratar a los objetos de superclase y a los objetos de subclase de manera similar.

VERDADERO

b) Todos los métodos en una clase abstract deben declararse como métodos abstract.

FALSO, existen métodos que no necesitan ser abstract porque son generalizados

c) Es peligroso tratar de invocar a un método que sólo pertenece a una subclase, a través de una variable de subclase.

FALSO, puede ser peligroso si se lo invoca de una variable de la superclase

d) Si una superclase declara a un método como abstract, una subclase debe implementar a ese método.

FALSO, solo una de las subclases debe implementarlo

e) Un objeto de una clase que implementa a una interfaz puede considerarse como un objeto de ese tipo de interfaz.

VERDADERO

Ejercicios

10.3 ¿Cómo es que el polimorfismo le permite programar "en forma general", en lugar de hacerlo "en forma específica"? Hable sobre las ventajas clave de la programación "en forma general".

El programar en forma general hace referencia a poder tener atributos y métodos que son aplicables a varios objetos, por lo que en lugar de ir creando una variable y una función en cada objeto independiente, puedes hacerlo en una superclase que herede dichas funciones a la clases que se desprendan de la misma.

10.4 Una subclase puede heredar la "interfaz" o "implementación" de una superclase. ¿En qué difieren las jerarquías de herencia diseñadas para heredar la interfaz, de las jerarquías diseñadas para heredar la implementación?

La principal diferencia es que la interfaz se hereda para métodos vacíos generalmente, que deben de manera obligatoria ser usadas por las subclases, mientras que el heredar la implementación no necesariamente conlleva la sobreescritura de métodos, pues pueden estar generalizados

10.5 ¿Qué son los métodos abstractos? Describa las circunstancias en las que un método abstracto sería apropiado.

Un método abstracto es un bloque de código que debe realizar una función especifica creado en una superclase, por lo que debe ser aplicado a una subclase en concreto para la reescritura del mismo. Es apropiado usar métodos abstractos cuando queremos definir que una cierta subclase debe tener obligatoriamente un método para que el programa funcione, por lo que generamos dicho bloque de código para tenerlo presente en un momento futuro del desarrollo.

10.6 ¿Cómo es que el polimorfismo fomenta la extensibilidad?

Al permitirnos reutilizar código de manera efectiva e intuitiva, la extensibilidad de un programa se ve mejorada porque se mejora la eficiencia la momento del desarrollo, evitando la duplicación innecesaria de código.

10.7 Describa cuatro formas en las que podemos asignar referencias de superclases y subclases a variables de los tipos de las superclases y las subclases.

1. **Super s = new Super();**
Se accede solo a lo definido en Super

2. **Super s = new Sub();**

Solo se pueden usar métodos visibles en Super, pero la implementación ejecutada es la de la subclase.
No permite acceder a atributos o métodos exclusivos de Sub.

3. Sub sub = new Sub();
Acceso a todo lo de Sub y lo heredado de Super

4. Super s = new Sub();

Sub sub = (Sub) s;

Permite acceder a métodos de la subclase

10.8 Compare y contraste las clases abstractas y las interfaces. ¿Para qué podría usar una clase abstracta? ¿Para qué podría usar una interfaz?

Las clases abstractas y las interfaces son formas de definir herencias en POO. Una clase abstracta puede contener tanto métodos abstractos como métodos con implementación, además de atributos y estado interno, por lo que es útil cuando varias clases comparten comportamiento común o una base parcial de implementación; en cambio, una interfaz define únicamente el conjunto de métodos que una clase debe ofrecer, sin imponer detalles de como se ejecuta.