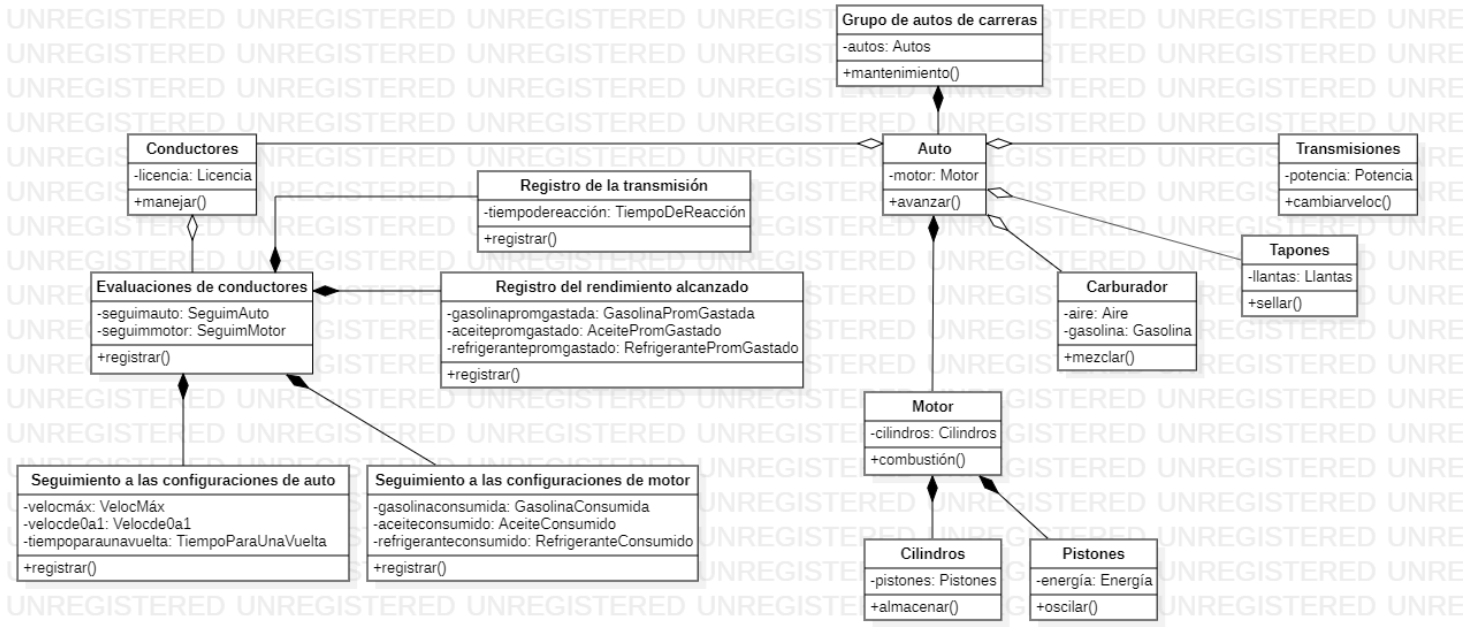
UNIVERSIDAD AUTÓNOMA METROPOLITANA
Casa abierta al tiempo
Unidad Iztapalapa

Práctica semana 8
Trimestre 21-O

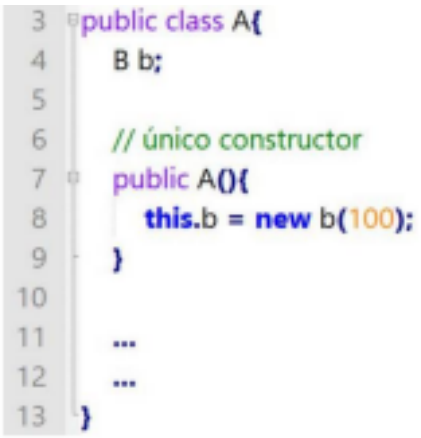
Análisis y diseño de sistemas de cómputo

Prof. Orlando Muñoz Texzocotetla

1. **Una compañía de mantenimiento a un grupo de autos de carreras. Estos autos utilizan algunos de los nuevos motores de 8 cilindros y las nuevas transmisiones. Una vez que los motores son ensamblados, los pistones, el carburador y los tapones no pueden cambiarse con otros motores debido a los cambios que causan altas temperaturas. Se desea mantener el registro del rendimiento alcanzado por cada motor en cada auto, y de cada transmisión en combinación con cada motor. Los conductores dan su evaluación después de manejar su correspondiente auto. Para ello se necesita de un sistema que lleve el seguimiento a las configuraciones de cada auto (y de cada motor), así como de las evaluaciones dadas por los conductores. Crear un diagrama de clases UML que representa la información dada en el párrafo anterior. El diagrama debe tener al menos 5 clases, con sus correspondientes atributos y métodos, así como sus correspondientes relaciones entre ellas: asociación, todo-parte y herencia.**



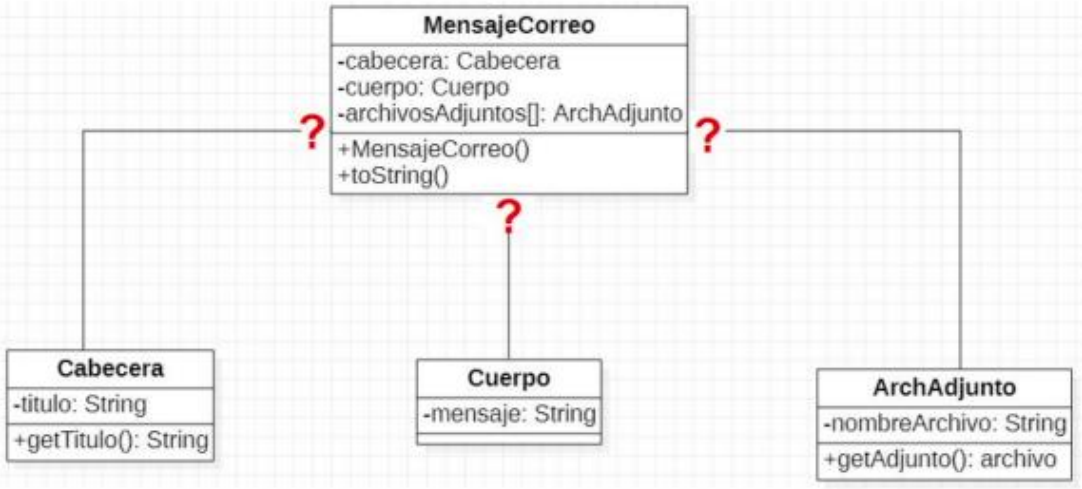
2. **Indique si el siguiente código en Java representa una relación de agregación o de composición, y explique por qué.**

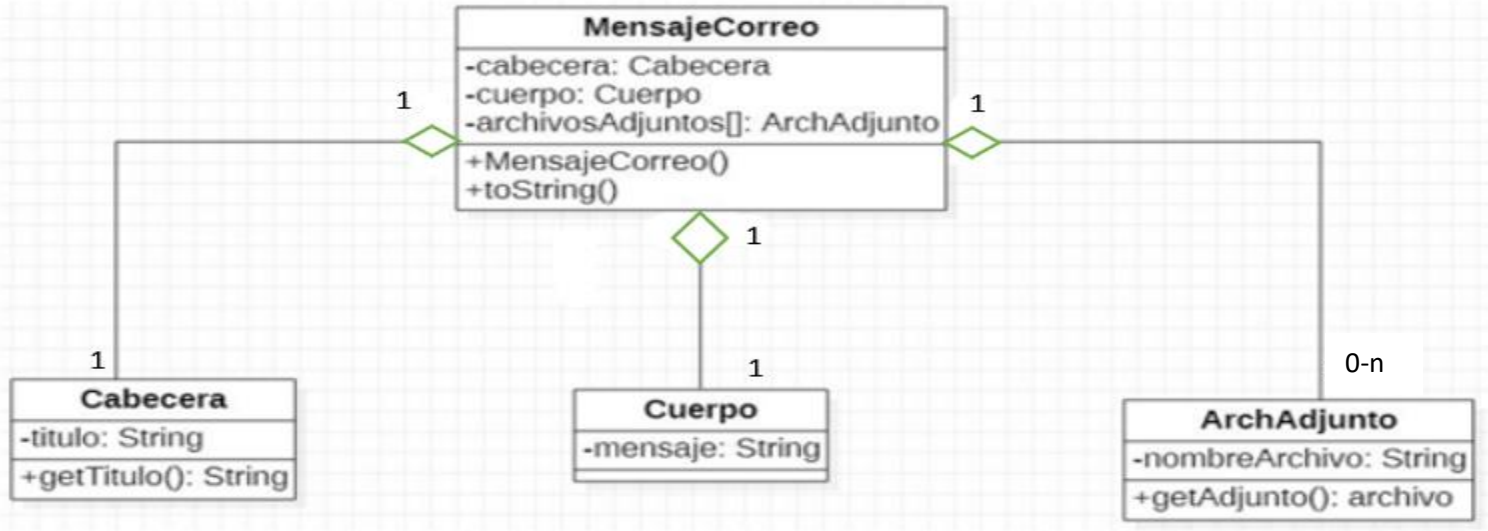


```
3  public class A{
4      B b;
5
6      // único constructor
7      public A(){
8          this.b = new b(100);
9      }
10
11     ...
12     ...
13 }
```

Representa una relación de composición, ya que para que sea agregación, el objeto en cuestión debe de ser agregado mediante un método siendo de otra clase, pero, en este caso es agregado mediante un único constructor. Por lo tanto, el objeto en cuestión es de dentro de la misma clase lo que hace que la relación sea de composición.

3. **Basados en el siguiente diagrama de clases, indique cuáles son las relaciones que se indican con un signo de interrogación, y escriba un programa en Java que implemente dichas relaciones.**





```
public class Encabezado
  {
  private String titulo;
  public Encabezado()
  {
                titulo = "Practica-Relaciones entre clases";
  }

  public final String getTitulo()
  {
                return titulo;
  }
}
public class Cuerpo
{
  private String mensaje;
  public Cuerpo()
  {
                mensaje = "Esta es la parte 3 de la practica";
  }
```

```
  public final String getMensaje()
  {
                        return mensaje;
  }
}

public class ArchAdjunto
{
  private String nombreArchivo;
  public ArchAdjunto ()
  {
                        nombreArchivo = "JavaApplication147.java Es un archivo .java";
  }

  public final String getArchAdjunto ()
  {
                        return nombreArchivo;
  }
}
```

```
public class CorreoMensaje
{
  private Encabezado encabezado = new Encabezado();
  private Cuerpo bd = new Cuerpo();
  private ArchAdjunto atc = new ArchAdjunto();
  public CorreoMensaje ()
  {
                        encabezado = new Encabezado();
                        bd = new Cuerpo();
                        atc = new ArchAdjunto ();
  }

  public final String toString()
  {
                        String result = " ";
                        result += "Encabezado : " + encabezado.getTitulo() + "\n";
                        result += "Cuerpo : " + bd.getMensaje() + "\n";
                        result += " ArchAdjunto: " + atc. getArchAdjunto() + "\n";
                        return result;
```

```
  }
}
package <missing>;
public class principal
{
                        public static void main(String[] args)
                        {
                          CorreoMensaje mail = new CorreoMensaje ();
                          System.out.print(mail.toString());
                          System.out.print("\n");
                          system("Pausa");
                        }
}
```

4. El código de las siguientes clases contiene errores, indique cuáles son e indique cómo corregirlos.

```
public class EjercicioHerencia{
    public static void main(String[] args){
        A oA = new A();
        System.out.println("En el main(): ");
        System.out.println("Valor double = "+oA.getB());

        B oB = new A();
        oB.setB(23.5);
    }
} // Clase principal
```

Primer punto: En la línea "`System.out.println("Valor double ="+oA.getB());`" se está obteniendo un valor de tipo double del método getB(), éste método getB() está dentro de la clase B, por lo tanto el error está en que se está llamando a partir del objeto oA pero esta clase no tiene dicho método por lo que va a salir un error. Para solucionar este error debemos de mandar a llamar el método con el objeto oB quedando el código de la siguiente manera:

```
System.out.println("Valor double ="+oB.getB());
```

Segundo punto: Solucionando el primer punto saldrá otro error, este error se debe a que se está utilizando el objeto oB mucho antes de su creación.Para su solución deberemos de subir la línea de código "`B oB=new A();`" para un lugar más arriba que la línea donde se manda a llamar dicho objeto.

Tercer Punto: Viendo la línea de código "`B oB=new A();`" se puede notar que se crea el objeto de la clase B pero se le relaciona con el constructor de la clase A, esto resultará en un error. Para solucionar este error se le deberá relacionar con el constructor de la clase B de la siguiente manera:

```
B oB=new B();
```

5. **Se desea crear una clase llamada vector3D, y se decide que esta clase derive de otra clase llamada vector2D. La explicación de esa decisión es que un vector 3D es un vector 2D pero con una dimensión más (añadida). Expliqué por qué esa decisión (y explicación) es un error conceptual de herencia en programación orientada a objetos.**

Para este problema debemos saber con claridad que es la **herencia** en **OOP** (programación orientada a objetos).

La ***herencia*** es el mecanismo en Java por el cual una clase permite heredar las características (atributos y métodos) de otra clase.

En el lenguaje de Java, una clase que se hereda se denomina superclase. La clase que hereda se llama subclase. Por lo tanto, una ***subclase*** es una versión especializada de una ***superclase***. Hereda todas las variables y métodos definidos por la superclase y agrega sus propios elementos únicos.

Dada la información anterior entendemos que la clase vector2D es nuestra superclase y el texto nos dice que se desea crear una clase llamada vector3D la cual deriva de la clase vector2D por lo que la podemos entender como nuestra subclase.

Un vector 3D está formado por 3 variables (x, y, z) y el vector 2D por 2 variables (x, y), pero el argumento nos indica que *“un vector 3D es un vector 2D, pero con una dimensión más (añadida)”*.

Esto anterior es incorrecto dado que la superclase debe contener atributos y métodos que sean utilizados por la subclase y esta subclase no puede añadirse con más dimensiones o atributos dado que se utilizara un extends que llame a nuestra superclase la cual nos dará los atributos (x, y) y no nos dará la variable (z) para tener un vector3D por lo que nuestra clase vector3D estaría incorrecta.

6. **El siguiente diagrama de clases contiene tres clases que fabrican un tipo particular de dulce: galleta, chocolate y mazapán. Se desea implementar UN SOLO método llamado fabricarDulce(¿Tipo? x). Este método recibirá un objeto x (ya sea una fábrica de galleta, de chocolate o de mazapán), el cual, creará el dulce correspondiente.**

a. **Complete el diagrama UML para que por medio de la herencia y la asociación sea posible que el método fabricarDulce reciba un solo objeto x, el cual, pueda fabricar cualquiera de los tres tipos de dulce dependiendo del objeto x recibido.**

b. **Implemente el método fabricarDulce(…).**