FACULTAD DE INGENIERÍA

FORMATO
PRÁCTICAS DE LABORATORIO

UNIVERSIDAD AUTÓNOMA DE CAMPECHE

NOMBRE DEL PROFESOR: Ing. Héctor Manuel Quej Cosgaya
NOMBRE DE LA PRÁCTICA: Acceso a Miembros
PRÁCTICA NÚM. [3]

LABORATORIO:	Centro de Ingeniería Computacional
MATERIA:	Lenguaje de Programación II
UNIDAD:	Subcompetencia II
TIEMPO:	2 horas

OBJETIVO:

Comprender el principio del encapsulamiento, aprender la manera en que se implementa, y el mecanismo para proveer acceso a miembros que han sido encapsulados.

MARCO TEÓRICO:

El encapsulamiento es uno de los pilares sobre el cual se basa el paradigma de la Programación Orientada a Objetos. Permite hacer que una clase se preocupe únicamente de su propio funcionamiento, haciendo que una tarea compleja pueda dividirse fácilmente en tareas más sencillas. Dado que una clase es responsable únicamente de la forma en que ella misma funciona, otras clases pueden utilizarla sin preocuparse por *cómo* la clase realiza sus operaciones, sino que simplemente reciben el resultado y ya. En la práctica, se procede a ejemplificar el uso y funcionamiento de los diferentes tipos de modificadores de acceso, así como la manera de proveer acceso a miembros que han sido encapsulados.

LISTA DE MATERIALES:

- Java SDK
- Editor de texto SciTE
- Archivo de código fuente 'Motor.java'
- Archivo de código fuente 'AutomovilFord.java'
- Archivo de código fuente 'TesterFord.java'

EQUIPO DE LABORATORIO:

- Computadora Personal

DESARROLLO DE LA PRÁCTICA:

1. **Abre** el archivo con código fuente '`Motor.java`'.
2. **Examina** la clase. La clase cumple con algunos de los principios del encapsulamiento, pero

está incompleta.

3. **Compila** la clase '`Motor.java`'
4. **Abre** el archivo con código fuente '`AutomovilFord.java`'

`AutomovilFord` es una modificación de nuestra clásica clase `Automovil`, pero que ha sufrido algunas modificaciones: por ejemplo, la `marca` ahora solamente puede ser "`Ford`" (de ahí el nombre de la clase) y se sustituyó la variable booleana `motor` por una referencia a la clase '`Motor`'.

5. **Observa** que a diferencia de la clase '`Motor`', la clase '`AutomovilFord`' no cumple una sola de las reglas del encapsulamiento.
6. **Compila** la clase '`Automovil`'. Un montón de errores surgieron. No temas, ¡pronto los arreglaremos!
7. Dentro de la clase '`Motor`' **añade** métodos de acceso (***setters*** o ***getters***, o ambos) para las variables que se necesitan utilizar en la clase `AutomovilFord` (`encendido` y `revoluciones`).
8. De vuelta a la clase '`AutomovilFord`', **sustituye** el intento de acceso a las variables privadas de '`Motor`' por los métodos de acceso correspondientes. 4 de los 7 errores deben haber desaparecido.

'`AutomovilFord`' intenta utilizar métodos que '`Motor`' declaró como ***privados***, pues comprometen su funcionamiento. '`Motor`' debe ser el ***único*** responsable de su funcionamiento, y sin embargo, '`AutomovilFord`' intenta hacer funcionar el `motor`… ¡*en el orden incorrecto*! Hagamos pues, que el único responsable del funcionamiento del `motor` sea la propia clase '`Motor`'.

9. **Crea** el siguiente método en la clase '`Motor`'

```
public void revolucionar() {
   if(encendido) {
      prepararMezcla();
      comprimirMezcla();
      encenderMezcla();
   }
}
```

10. **Modifica** el método `acelerar()` de la clase '`AutomovilFord`'. **Borra** los tres métodos que intentaban hacer funcionar al motor y **sustitúyelas** por una llamada al método `revolucionar()` que creaste en el paso anterior.
11. **Modifica** el método `frenar(int)` de la clase '`AutomovilFord`' para que incluya una llamada al método de acceso para el atributo '`revoluciones`' que creaste en el paso 7, y que establezca su valor en la cantidad proporcionada como argumento.
12. **Compila** ambas clases: '`AutomovilFord`' y '`Motor`'.
13. **Abre** el archivo con código fuente '`TesterFord.java`'.

14. **Compila** y **ejecuta** la clase '`TesterFord`'. El `AutomovilFord` funciona correctamente, ya que el `Motor` se hizo responsable de su funcionamiento.

Fin de la práctica.

RETROALIMENTACIÓN:

- Investiga las buenas prácticas de desarrollo de software (entre ellas está el encapsulamiento)

RECOMENDACIONES ADICIONALES:

- Lee el capítulo 11 del Dean (Detalles de tipo y mecanismos de codificación alternativa)

BIBLIOGRAFÍA:

- Dean, J. S., & Dean, R. H. (2009). Introducción a la programación con Java. México: Mc Graw Hill.
- The Java™ Tutorials: Classes and Objects: http://docs.oracle.com/javase/tutorial/java/javaOO/index.html
- Apuntes del profesor.