FACULTAD DE INGENIERÍA

FORMATO
PRÁCTICAS DE LABORATORIO

UNIVERSIDAD AUTÓNOMA DE CAMPECHE

NOMBRE DEL PROFESOR: Ing. Héctor Manuel Quej Cosgaya
NOMBRE DE LA PRÁCTICA: Conceptos Esenciales de Programación Orientada a Objetos
PRÁCTICA NÚM. [2]

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

OBJETIVO:

Comprender algunos de los conceptos esenciales de la Programación Orientada a Objetos: constructores, métodos sobrecargados, paso de argumentos, Autorreferencias, clases anidadas y paquetes.

MARCO TEÓRICO:

Los conceptos básicos de la Programación Orientada a Objetos son suficientes para escribir programas pequeños pero funcionales. Sin embargo, conforme la funcionalidad (y por ende, la complejidad) de un programa aumenta, son necesarias más conocimientos acerca del paradigma Orientado a Objetos para mantener un buen diseño de clases, escribir un código estructurado y fácil de leer, y apegarse a los principios de la Programación Orientada a Objetos. Estos conocimientos son los que se pretende enseñar durante el transcurso de la práctica (y a lo largo del curso también). En la práctica, se exponen uno a uno dichos conceptos, para que el alumno aprenda como utilizarlos.

LISTA DE MATERIALES:

- Java SDK
- Editor SciTE de Scintilla
- Archivo de código fuente 'Automovil.java'
- Archivo de código fuente 'ControladorAutomovil.java'

EQUIPO DE LABORATORIO:

- Computadora Personal

DESARROLLO DE LA PRÁCTICA:

Primera parte: **Constructores y sobrecarga de métodos.**

1. **Abre** el editor de texto **SciTE**. En su defecto, puedes utilizar el **Bloc de Notas** de Windows

o cualquier editor de texto de tu preferencia (excepto **NetBeans**).

2. **Abre** el archivo con código fuente '`Automovil.java`' que se te proporcionó junto con la práctica.
3. **Añade** un nuevo constructor para la clase `Automovil`, como se indica a continuación:

```
public Automovil(String tono, String tipo) {
   color = tono;
   modelo = tipo;
}
```

4. **Compila** el programa. El compilador te marcará un error.

Observa que cambiar el nombre de los parámetros no es suficiente para sobrecargar un método (o constructor).

5. **Modifica** el constructor que acabas de crear para que reciba un **tercer argumento**, una cadena llamada '`fabricante`'. Asigna el valor de tu nuevo parámetro a la **variable de instancia** '`marca`'.
6. **Compila** el programa. El error habrá desaparecido.
7. **Añade** el siguiente método a la clase `Automovil`:

```
public void frenar(int cantidad) {
   velocidad -= cantidad;
   if (velocidad < 0) velocidad = 0;
   System.out.println("Vamos a " + velocidad + " km/h");
}
```

8. **Compila** el programa.

A pesar que el nuevo método tiene el mismo nombre que otro método, su *firma* es diferente, por lo que no hay problema para el compilador.

9. **Abre** el archivo con código fuente '`ControladorAutomovil.java`' que se te proporcionó junto con la práctica.
10. **Compila** y **ejecuta** la clase '`ControladorAutomovil`'.
11. Modifica la llamada al método `frenar()`, proporcionándole un valor entero como **argumento**.
12. Compila y ejecuta la clase '`ControladorAutomovil`' de nuevo.

Observa ambos resultados de la llamada al método `frenar()`. La máquina virtual sabe perfectamente cual método invocar de acuerdo con los parámetros que reciba.

Segunda parte: **Paso de argumentos.**

13. **Añade** las siguientes líneas de código después de la **última** instrucción del método main

de la clase ‘ControladorAutomovil’

```
auto.acelerar(50);
// Paso por valor
auto.duplicarVelocidad(auto.velocidad);
System.out.println(“El auto se mueve a “
   + auto.verVelocidad() + “ km/h”);

auto.color = “Rojo”;
System.out.println(“Mi auto es de color “ + auto.color);
// Paso por referencia
Automovil.pintar(auto);
System.out.println(“El color de mi auto ahora es “ + auto.color);
```

14. **Compila** y **ejecuta** la clase ‘ControladorAutomovil’.

¿Notas la diferencia entre el paso por valor y el paso por referencia?

Tercera parte: **Autorreferencias**

15. De regreso a la clase ‘Automovil’, **modifica** el método ‘duplicarVelocidad()’ de la siguiente manera:

```
public void duplicarVelocidad(int velocidad) {
   this.velocidad *= 2;
   System.out.println(“Ahora vas a “ + this.velocidad + “km/h!”);
}
```

16. **Compila** la clase ‘Automovil’.
17. **Compila** y **ejecuta** la clase ‘ControladorAutomovil’

La autorreferencia nos permite acceder a la variable que estaba siendo ocultada por el parámetro.

18. **Modifica** el constructor de la clase Automovil que recibe 2 parámetros de la siguiente manera:

```
public Automovil(String modelo, String marca) {
   this.modelo = modelo;
   this.marca = marca;
}
```

19. **Añade** un nuevo constructor para la clase Automovil:

```
public Automovil(int velocidad, boolean motor, String color,
   String marca, String modelo) {
   this.velocidad = velocidad;
   this.motor = motor;
   this.color = color;
   this.modelo = modelo;
   this.marca = marca;
   numeroDeAutos++;
}
```

20. **Añade** un último constructor para nuestra clase 'Automovil'. Ya es el último, ¡lo juro!

```
public Automovil() {
   this(0, false, "Rojo", "Ford", "Fiesta");
}
```

21. **Compila** la clase 'Automovil'.

Date cuenta que el compilador no tiene ninguna objeción de que existan cuatro constructores para nuestra clase.

22. **Modifica** los dos constructores originales para que utilicen la **autorreferencia**. Utiliza el constructor sin parámetros como guía: proporciona los **valores** que *recibas* al constructor que recibe los 5 **argumentos**, los que no, utiliza los valores *default* que utiliza el constructor sin parámetros.

Cuarta parte: **Clases anidadas.**

23. **Añade** las siguientes clases *dentro* de la clase 'Automovil'

```
static class Estereo {
   public void reproducirCancion(String cancion) {
      System.out.println("Now playing… " + cancion);
   }
}

class Radiador {
   public void enfriar() {
      if(motor) {
         System.out.println("Enfriando…");
      } else {
         System.out.println("No tengo nada que enfriar");
      }
   }
}
```

24. **Compila** la clase 'Automovil'.
25. **Observa** el directorio donde se encuentra nuestra clase 'Automovil'. ¡Las dos clases dentro de Automovil se compilaron también!.
26. **Regresa** a la clase 'ControladorAutomovil'.
27. **Instancia** un objeto de la **clase anidada estática** Estereo utilizando la siguiente sintaxis.

```
Automovil.Estereo radio = new Automovil.Estereo();
radio.reproducirCancion("Guns n' Roses – Nightrain");
```

28. **Instancia** un objeto de la **clase interna** Radiador utilizando la siguiente sintaxis:

```
Automovil.Radiador radiador = auto.new Radiador();
radiador.enfriar();
```

29. **Compila** y **ejecuta** la clase 'ControladorAutomovil'.

Nota que después de instanciarlos, ambos objetos se comportan como cualquier otro objeto.

Quinta parte: **Paquetes**

30. **Añade** la siguiente sentencia de empaquetado como la ***primera*** sentencia de ambas clases, 'Automovil' y 'ControladorAutomovil'

```
package mx.uacam.fi;
```

31. Intenta **compilar** ambas clases. El compilador te marcará un error para la clase 'ControladorAutomovil'
32. **Crea** una carpeta en el Escritorio llamada '*mx*'. **Dentro** crea otra carpeta llamada '*uacam*'. **Dentro** de '*uacam*' crea una carpeta llamada '*fi*'. Finalmente, **mueve** tus dos archivos fuente dentro de '*fi*'.
33. **Compila** la clase 'Automovil'.
34. Intenta **compilar** de nuevo la clase 'ControladorAutomovil' ¡El error sigue ahí!
35. **Modifica** la variable de entorno ***Classpath*** de tu sistema operativo para que incluya a tu escritorio. ¡*Consulta a tu asesor si tienes la menor duda sobre como hacerlo*!
36. Ya que hayas modificado la variable ***Classpath***, **compila** de nuevo la clase 'ControladorAutomovil'. ¡El error desapareció!
37. **Abre** el **Símbolo del sistema**.
38. **Cambia** la ruta del **Símbolo del sistema** al **Escritorio**.
39. **Escribe** el siguiente comando para ejecutar a la clase 'ControladorAutomovil'

```
java mx.uacam.fi.ControladorAutomovil
```

Fin de la práctica.

RETROALIMENTACIÓN:
• Investiga a fondo que es el classpath de Java, para qué sirve y cómo se configura.

RECOMENDACIONES ADICIONALES:
• Lee los capítulos 5 y 9 del Dean.

BIBLIOGRAFÍA:
• Dean, J. S., & Dean, R. H. (2009). Introducción a la programación con Java. México: Mc Graw Hill. • Apuntes del profesor.