Universidad del Valle de Guatemala
CC2008 - Introducción a la Programación Orientada a Objetos
Semestre II, 2021
Docente:

Hoja de Ejercicios 3

Objetivos que se buscan del estudiante:

- Identifica los problemas de diseño en el código proporcionado y rediseña el sistema para que cumpla con las buenas prácticas de la Programación Orientada a Objetos.
- Aplica conocimientos sobre encapsulación, atributos estáticos, sobrecarga de métodos, relaciones entre clases, sentencias repetitivas y condicionales.
- Aplica arreglos dinámicos en el diseño de su solución.
- Implementa el sistema diseñado de manera que se puede ejecutar sin errores.
- Investigar implementación de clases nuevas, específicamente la clase "`File`"

Recordatorios

- La clase `Scanner` del paquete `java.util` permite crear objetos para leer de distintas fuentes.
- La clase `Scanner` debe ser importada.
- Para leer del teclado se necesita crear un objeto de la clase `Scanner`.
- Consulte la documentación del API de Java en línea.
- Hemos visto tres tipos de relaciones entre clases: dependencia, agregación y composición.

Problema a resolver

Un joven emprendedor amigo del colegio le informa que ha adquirido un terreno en una zona urbana de la ciudad el cual proporcionará servicios de parqueo.

A él le interesa mucho el tema de la tecnología y le gustaría tener más información acerca de los clientes que hacen uso de su servicio, debido a esto se pone en contacto con usted para proponerle una idea de negocio.

La primera fase de su emprendimiento tiene contemplado 5 espacios de parqueo, pero al capitalizarse y obtener mayor inversión, piensa adquirir parqueos aéreos para albergar el doble de automóviles, en sus planes a medio plazo se encuentra la construcción de una torre de parqueos y adquisición de más terrenos.

Cada espacio de parqueo debería tiene las siguientes características (puede agregar según sea su análisis):

- Largo
- Ancho
- Altura
- ¿Está techado?
- ¿Es áereo?

Sobre la ocupación del espacio, se debería almacenar la siguiente información (tomar en cuenta que este listado de características es específico para cada espacio):

- Vehículo que se estaciona.
- Cuantas horas a estado el vehículo estacionado
- Placa del vehículo.
- Marca del vehículo.
- Modelo del vehículo.

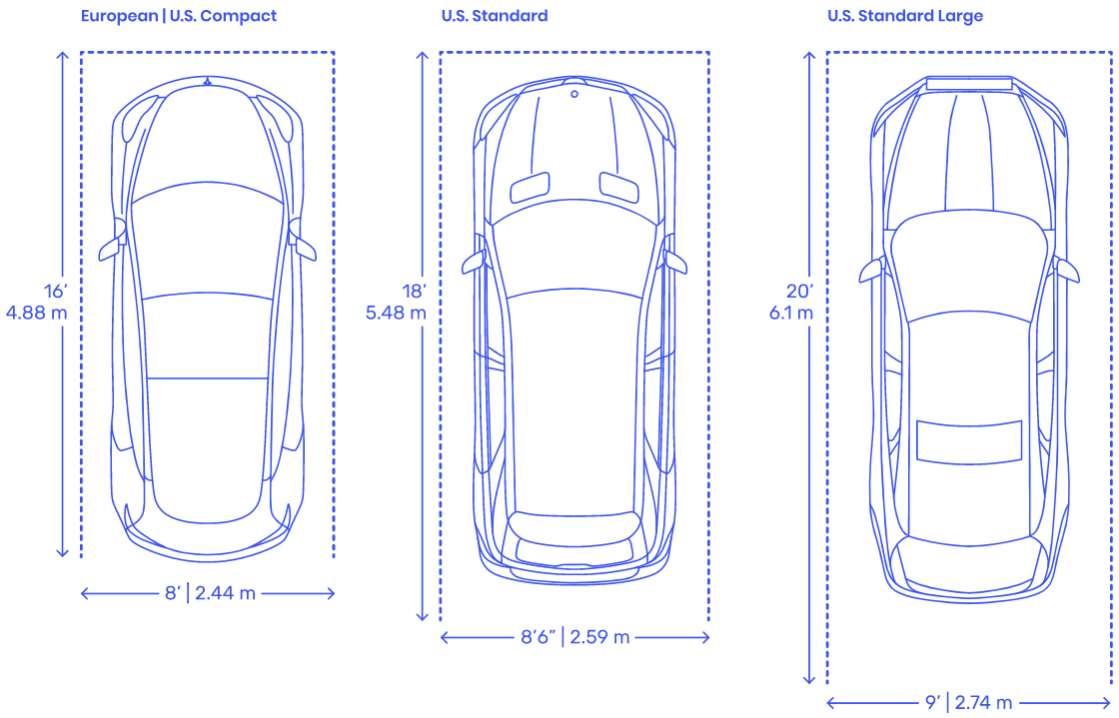

Imagen 1: Referencia consultada en: https://www.dimensions.com/element/parking-spaces

Para su posterior análisis, las estadísticas mínimas que su amigo necesita son:

- Horarios de mayor utilización.
- Tiempo promedio de uso del parqueo.
- Parqueo más utilizado.
- ¿Cuántos vehículos son rechazados cuando el parqueo ya está lleno?
- ¿Qué marca de automóviles son mayormente utilizados?
- ¿Qué características poseen los parqueos que son más utilizados?

Su amigo está muy entusiasmado y le solicita un último requerimiento, desea poder guardar todos los datos al cerrar el programa, y al ejecutarlo nuevamente poder hacer uso de la información almacenada, cada archivo con información pertenecerá a un determinado local de parqueo (como se planteó al inicio se podrán administrar diferentes ubicaciones).

Su amigo había empezado a trabajar él solo, pero debido a que desconocía temas relacionados a la programación orientada a objetos no estaba haciendo el uso correcto de este paradigma, por lo usted debe corregir y modificar el código proporcionado.

```java
/**
 * @author
 * @file Estacionamiento.java
 *
 */
public class Estacionamiento {
	public String nombre;
	public String dirección;
	public double precio;
	public EspacioParqueo espacio1;
	public EspacioParqueo espacio2;
	public EspacioParqueo espacio3;
	public EspacioParqueo espacio4;
	public EspacioParqueo espacio5;


	public void desplegar() {
	}
}
/**
 * @author
 * @file EspacioParqueo.java
 *
 */
class EspacioParqueo {
	public double ancho;
	public double largo;
	public double altura;
	public String caracteristicas; //Si está techado o aereo

	public EspacioParqueo ()  {


	}
}
```

Universidad del Valle de Guatemala
CC2008 - Introducción a la Programación Orientada a Objetos
Semestre II, 2021
Docente:

Instrucciones

Identificación de errores:

Identifique los errores que considera usted que su amigo ha cometido y proponga las soluciones correspondientes.

Análisis:

Especifique qué se espera que haga el programa y, si es necesario, cómo es que el código presentado falla en cumplir dichas expectativas. Describa cada una de las clases que creará o alterará para representar y resolver el problema. De cada una de las clases, debe especificar:

- Objetivo de la clase.
- Objetivo de cada atributo.
- Descripción breve de cada método.

Ponga especial atención al uso de arreglos dinámicos en su solución. ¿De qué necesitará almacenar múltiples instancias? ¿Cuándo se agregarán instancias? ¿Se podrán remover instancias? ¿Quién agregará las instancias, y cómo?

Diseño:

Elabore el diagrama UML de las clases diseñadas. Preste atención a las relaciones entre clases, detallando entre dependencia, composición y agregación. **NO** use asociación (línea continua, sin flecha). Explique las relaciones usadas entre clases por medio de una nota agregada a su(s) diagrama(s).

Implementación:

Programe el sistema diseñado usando Java. Incluya un *driver program* que demuestre las funcionalidades requeridas.
Se requiere aplicar el patrón de diseño MVC (*Model – View – Controller*).

Material a entregar en Canvas

- Archivo .pdf con la información del análisis y la identificación de los errores.
- Imágenes (.jpg, .png) con los diagramas de clases.
- Archivos **.java** escritos.

Evaluación

(10 puntos) Identificación de errores

- Se identifican correctamente los errores de diseño y malas prácticas.
- Se propone una solución correcta y lógica para corregir los errores de diseño que detectaron en el código.

`

Universidad del Valle de Guatemala
CC2008 - Introducción a la Programación Orientada a Objetos
Semestre II, 2021
Docente:

(20 puntos) Análisis.

- Se especifica correctamente qué debe hacer el programa.
- Describe correctamente cada una de las clases involucradas. La descripción es clara por lo que se comprende el propósito de la clase.
- Describe correctamente los atributos y métodos de cada clase, se comprende bien la necesidad de cada uno.
- Investiga el uso de la clase "File" para guardar datos en memoria persistente.
- Emplea el concepto de almacenamiento de tamaño dinámico para desarrollar su solución.

(30 puntos) Diseño.

- Elabora diagramas de clases correctos siguiendo el estándar UML y aplicando el patrón MVC.
- Incluye todas las clases concebidas en el análisis, con todos sus métodos y atributos.
- Todas las clases tienen correcta la visibilidad de métodos y atributos. Se aplican los principios de POO vistos hasta ahora (SRP, Ley de Demeter, Alta cohesión -> bajo acoplamiento).
- Se incluyeron todos los métodos y sobrecargas necesarias (*e.g.*, constructor, *setters*, *getters*, métodos funcionales).
- Las relaciones entre las clases son correctas, representan bien la interacción entre las mismas y están claramente explicadas.
- Define el formato (*e.g.,* CSV) en que los archivos deberán ser guardados para ser utilizados posteriormente.

(40 puntos) Implementación.

- Están codificadas todas las clases diseñadas (si se han realizado cambios al diseño, incluir un rediseño señalando los cambios y comentándolos).
- Se usan clases de almacenamiento dinámico (*e.g.,* `ArrayList`, `Stack`, `LinkedList`, etc.)
- Se pueden guardar y cargar archivos previamente guardados en memoria persistente.
- No tiene errores de sintaxis ni en tiempo de ejecución. Se aplica programación defensiva con bloques `try-catch` para manejo de errores en tiempo de ejecución.
- El programa cumple con las funcionalidades requeridas.
- Está documentado y organizado adecuadamente: cada archivo tiene encabezado y para cada método se describe el propósito, los parámetros y el valor de retorno. También se desarrolla una clase por archivo.