**Sistema de Gestión de Citas para Clínica Veterinaria**

**Contexto**

Una clínica veterinaria necesita implementar un sistema informático para la gestión de citas. El sistema debe permitir crear citas para las mascotas, asignarles una fecha y hora, un propietario, y gestionar el estado de las mismas. La clínica dispone de un horario de atención de 9:00 a 20:00 de lunes a viernes, excepto los miércoles que permanece cerrada por descanso semanal.

**Requisitos del Sistema**

**1. Clases Principales**

**Clase EstadoCita (Enum)**

Debe implementar un enum que represente los posibles estados de una cita:

* ACTIVA: La cita está programada y pendiente de realizarse.
* ANULADA: La cita ha sido cancelada.
* REALIZADA: El propietario ha acudido con su mascota a la cita programada.
* NOREALIZADA: La cita no fue cancelada, pero el propietario no acudió.

Cada estado debe contener:

* Una descripción textual del estado
* Un código numérico (1 para ACTIVA, 2 para ANULADA, etc.)
* Un método para comprobar si un cambio de estado es válido (por ejemplo, una cita REALIZADA no puede pasar a ACTIVA)

**Clase Cita**

Debe contener:

* Fecha y hora de creación de la cita (automáticamente la del sistema)
* Fecha y hora programada para la cita
* Nombre del propietario
* Nombre de la mascota
* Estado de la cita (usando el enum EstadoCita)
* Duración estimada en minutos (por defecto 30 minutos)
* Motivo de la consulta (texto breve)
* Un identificador único para cada cita (puede ser un número secuencial)

Implementaciones:

* Dos constructores:
  + Constructor sin parámetros: Crea una cita con valores por defecto y fecha 15 días después.
  + Constructor parametrizado: Crea una cita con los datos proporcionados.
* Métodos para cambiar el estado de la cita
* Método toString() para mostrar toda la información de la cita
* Debe implementar Comparable<Cita> para ordenar naturalmente por fecha y hora de la cita

**Clase UtilidadesCita**

Clase que contendrá métodos estáticos para operaciones con citas:

* Método para solicitar datos de una cita por consola
* Método para validar fechas (no permitir citas en miércoles, no permitir citas en fechas pasadas)
* Método para formatear fechas en diferentes formatos (corto, largo, solo hora, etc.)

**Clase GestorCitas**

Clase principal que gestionará una lista de citas:

* Debe mantener una lista de citas (ArrayList o LinkedList)
* Métodos para agregar, modificar y eliminar citas
* Métodos para buscar citas por diferentes criterios

**2. Operaciones con Listas**

Se deben implementar las siguientes operaciones con listas:

1. **Operaciones Básicas/Intermedias:**
   * Añadir y eliminar citas de la lista
   * Filtrar citas por estado (mostrar solo las activas, anuladas, etc.)
   * Filtrar citas por fecha (mostrar citas de un día específico)
   * Filtrar citas por nombre de propietario o mascota
2. **Operaciones Avanzadas:**
   * Comprobar si una nueva cita se solapa con alguna existente
   * Mostrar los huecos disponibles en un día específico
   * Obtener estadísticas básicas: número de citas por estado, por día de la semana, etc.
   * Reorganizar citas en caso de cancelación (proponer nuevas fechas)

**3. Operaciones con Enum**

1. **Operaciones Básicas/Intermedias:**
   * Implementar un método que devuelva todos los estados posibles como un array o lista
   * Crear un método para verificar si una transición de estado es válida
   * Obtener la descripción textual de un estado
2. **Operaciones Avanzadas:**
   * Implementar un método que proponga el siguiente estado lógico de una cita
   * Crear un historial simple de cambios de estado para cada cita
   * Validar operaciones disponibles según el estado actual

**4. Comparable, Comparator y Búsqueda Binaria**

1. **Implementación de Comparable:**
   * La clase Cita debe implementar la interfaz Comparable<Cita> para ordenación natural por fecha y hora de la cita.
2. **Implementación de Comparator:**
   * Crear al menos tres comparadores específicos para ordenar citas por:
     + Nombre del propietario (alfabéticamente)
     + Duración estimada (de menor a mayor)
     + Estado de la cita (según el orden de los valores del enum)
3. **Búsqueda Binaria:**
   * Implementar búsqueda binaria para encontrar:
     + Una cita en una fecha y hora específica (o la más cercana)
     + Citas de un propietario específico (previa ordenación)

**5. Métodos Adicionales de Gestión de Citas**

1. **Operaciones Intermedias:**
   * Verificar si una hora está dentro del horario de atención
   * Calcular el tiempo entre citas consecutivas
   * Mostrar un resumen de citas por día o semana
2. **Operaciones Avanzadas:**
   * Identificar días con más o menos carga de trabajo
   * Sugerir el mejor momento para una nueva cita (menor tiempo de espera)
   * Implementar un sistema de prioridad simple para citas urgentes

**Programa Principal**

Debe implementarse una clase Programa con un menú interactivo que permita:

1. Crear cita por defecto (no parametrizada)
2. Solicitar datos y guardar cita
3. Modificar datos de una cita existente
4. Buscar citas por diferentes criterios
5. Mostrar listados ordenados según diferentes criterios
6. Mostrar estadísticas básicas
7. Gestionar cancelaciones y cambios de estado
8. Exportar listado de citas a texto
9. Salir del programa

**Restricciones y Consideraciones Técnicas**

1. No se permite el uso de herencia ni abstracción para la solución.
2. Las fechas deben gestionarse usando las clases de la API de tiempo de Java (LocalDate, LocalDateTime, etc.).
3. La clínica no atiende los miércoles (día de descanso).
4. No se pueden programar citas para fechas anteriores a la actual.
5. El horario de atención es de 9:00 a 20:00, con citas de duración variable (mínimo 15 minutos).
6. Debe implementarse validación de datos para todos los campos introducidos.
7. Las búsquedas por fecha/hora deben implementarse de manera eficiente.

**Entregables**

1. Código fuente comentado de todas las clases implementadas
2. Breve documentación explicando la estructura de clases y métodos principales
3. Pruebas que demuestren el correcto funcionamiento del programa

**Implementación Recomendada**

Se sugiere implementar las clases en el siguiente orden:

1. Enum EstadoCita
2. Clase Cita
3. Clase UtilidadesCita
4. Clase GestorCitas
5. Clase Programa

**Consejos de Implementación**

* Comienza con las funcionalidades básicas y ve añadiendo las más complejas gradualmente
* Prueba cada método por separado antes de integrarlo en el sistema
* Utiliza la clase Scanner para la entrada de datos por consola
* Maneja adecuadamente las excepciones, especialmente en la entrada de fechas