Resuelto-ControlCheck2-G3-2425.p...

CVV__

Diseño y Pruebas i

3º Grado en Ingeniería Informática - Ingeniería del Software

Escuela Técnica Superior de Ingeniería Informática
Universidad de Sevilla

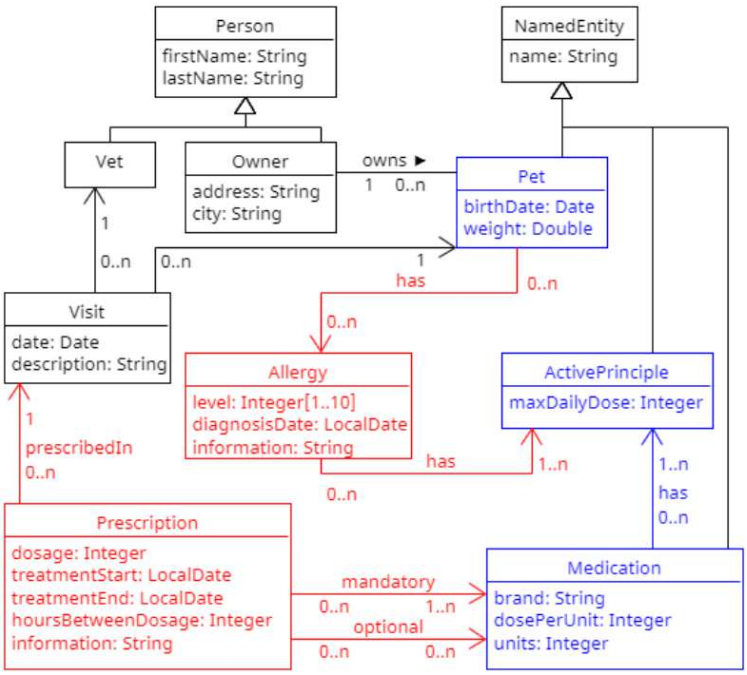
Accede al documento original

Control práctico de DP1 2024-2025 (Control-check 2)

Enunciado

En este ejercicio, añadiremos la funcionalidad de gestión de alergias de las mascotas a principios activos de los medicamentos prescritos en las visitas de las mascotas. Concretamente, se proporciona una clase que representa los medicamentos disponibles que está relacionada con otra clase que representa los principios activos que pueden contener. Como objetivo, tendremos que completar las clases "Allergy" que representa las alergias de las mascotas a los distintos principios activos, así como "Prescription" que representa la prescripción de medicamentos obligatorios y opcionales para una mascota en una visita y que obviamente tendrá que tener en cuenta las alergias de dicha mascota.

El diagrama UML que describe las clases y relaciones con las que vamos a trabajar es el siguiente:



Las clases para las que realizaremos el mapeo como entidades JPA se han señalado en rojo. Las clases azules son clases que se proporcionan ya mapeadas pero con las que se trabajará durante el control.

Realizaremos una serie de ejercicios basados en funcionalidades que implementaremos en el sistema, y validaremos mediante pruebas unitarias. Si desea ver el resultado que arrojarían las pruebas en backend, puede ejecutarlas (bien mediante su entorno de desarrollo favorito, bien mediante el comando "*mvnw test*" en la carpeta raíz del proyecto). Cada ejercicio correctamente resuelto valdrá dos puntos, el número de casos de prueba de cada ejercicio puede variar entre uno y otro y la nota se calculará en base al

porcentaje de casos de prueba que pasan. Por ejemplo, si pasan la mitad (50%) de los casos de prueba de un ejercicio, en lugar de dos puntos usted obtendrá un 1.

Para comenzar el control debe aceptar la tarea de este control práctico a través del siguiente enlace:

https://classroom.github.com/a/oSqbyiyk

Al aceptar dicha tarea, se creará un repositorio único individual para usted, debe usar dicho repositorio para realizar el control práctico. Debe entregar la actividad en EV asociada al control check proporcionando como texto la dirección url de su repositorio personal. Recuerde que además debe entregar su solución del control.

La entrega de su solución al control se realizará mediante un único comando "*git push*" a su repositorio individual. Recuerde que debe hacer push antes de cerrar sesión en la computadora y abandonar el aula, de lo contrario, su intento se evaluará como no presentado. Su primera tarea en este control será clonar (recuerde que si va a usar los equipos del aula para realizar el control necesitará usar un token de autenticación de GitHub como clave, tiene un documento de ayuda a la configuración en el propio repositorio del control). A continuación, deberá importar el proyecto en su entorno de desarrollo favorito y comenzar los ejercicios abajo listados. Al importar el proyecto, el mismo puede presentar errores de compilación. No se preocupe, si existen, dichos errores irán despareciendo conforme usted vaya implementando los distintos ejercicios del control.

Nota importante 1: No modifique los nombres de las clases ni la signatura (nombre, tipo de respuesta y parámetros) de los métodos proporcionados como material de base para el control. Las pruebas que se usan para la evaluación dependen de que las clases y los métodos tengan la estructura y nombres proporcionados. Si los modifica probablemente no pueda hacer que pasen las pruebas, y obtendrá una mala calificación.

Nota importante 2: No modifique las pruebas unitarias proporcionadas como parte del proyecto bajo ningún concepto. Aunque modifique las pruebas en su copia local del proyecto, éstas serán restituidas mediante un comando git previamente a la ejecución de las pruebas para la emisión de la nota final, por lo que sus modificaciones en las pruebas no serán tenidas en cuenta en ningún momento.

Nota importante 3: Mientras haya ejercicios no resueltos habrá tests que no funcionen y, por tanto, el comando "*mvnw install*" finalizará con error. Esto es normal debido a la forma en la que está planteado el control y no hay que preocuparse por ello. Si se quiere probar la aplicación se puede ejecutar de la forma habitual pese a que "mvnw install" finalice con error.

Nota importante 4: La descarga del material de la prueba usando git, y la entrega de su solución con git a través del repositorio GitHub creado a tal efecto forman parte de las competencias evaluadas durante el examen, por lo que no se aceptarán entregas que no hagan uso de este medio, y no se podrá solicitar ayuda a los profesores para realizar estas tareas.

Nota importante 5: No se aceptarán como soluciones válidas proyectos cuyo código fuente no compile correctamente o que provoquen fallos al arrancar la aplicación en la inicialización del contexto de Spring. Las soluciones cuyo código fuente no compile o sean incapaces de arrancar el contexto de Spring serán evaluadas con una nota de 0.

Test 1 – Creación de las entidades Prescription y Allergy y sus repositorios asociados

Parte 1A: Entidades Prescription y Allergy y sus repositorios (1punto):

Modificar las clases “Prescription” y “Allergy” para que sean entidades. Estas clases están alojadas en el paquete “org.springframework.samples.petclinic.medication”, y deben tener los siguientes atributos y restricciones:

Para ambas clases:

- El atributo de tipo entero (Integer) llamado “id” actuará como clave primaria en la tabla de la base de datos relacional asociada a la entidad.
- El atributo de tipo cadena caracteres (String) llamado “information” opcional.

Para la clase Prescription:

- El atributo obligatorio de tipo entero (Integer) llamado “dosage”, que representa la cantidad de medicación en miligramos que debe tomar el animal. El atributo será obligatorio y tendrá un valor mínimo de 1 y un valor máximo de 1000.
- El atributo de tipo fecha (LocalDate) llamado “treatmentStart”, que representa la fecha en que comienza la medicación pautada. La fecha seguirá el formato “dd/MM/yyyy” (puede usar como ejemplo la clase Pet y su fecha de nacimiento para ver cómo se especificar dicho formato, pero nótese que el patrón del formato es distinto). Este atributo es obligatorio.
- El atributo de tipo fecha (LocalDate) llamado “treatmentEnd”, que representa la fecha en que debe terminar la medicación pautada. Seguirá el formato “dd/MM/yyyy”. Este atributo debe ser obligatorio.
- El atributo de tipo entero (Integer) llamado “hoursBetweenDosage”, que representa el número de horas entre cada dosis a dar al animal. El atributo será obligatorio y tendrá un valor mínimo de 1 y un valor máximo de 24.

Para la clase Allergy:

- El atributo obligatorio de tipo entero (Integer) llamado “level”, que representa la severidad con la que se presenta la alergia en el animal. El atributo será obligatorio y tendrá un valor mínimo de 1 y un valor máximo de 10.
- El atributo de tipo fecha (LocalDate) llamado “diagnosisDate”, que representa la fecha en la que la alergia fue diagnosticada al animal. Seguirá el formato “dd/MM/yyyy”. Este atributo debe ser obligatorio.

No modifique por ahora las anotaciones @Transient de las clases. Modificar las interfaces “PrescriptionRepository” y “AllergyRepository” alojadas en el mismo paquete para que extiendan a CrudRepository. No olvide especificar sus parámetros de tipo.

Parte 1B: Relaciones entre las entidades (1 punto)

Elimine las anotaciones @Transient de los métodos y atributos que las tengan en las entidades creadas en el ejercicio anterior, así como del atributo allergies de la clase Pet. Se pide crear las siguientes relaciones entre las entidades:

Cree una relación unidireccional desde "Prescription" hacia "Visit" que exprese la que aparece en el diagrama UML (mostrado en la primera página de este enunciado) respetando sus cardinalidades, usando un atributo "visit" en la clase "Prescription".

Cree dos relaciones unidireccionales desde "Prescription" hacia "Medication" que representen las que aparecen en el diagrama UML, es decir la medicación prescrita tanto obligatoria como opcional. Tenga en cuenta la cardinalidad que tienen (*recuerde que en este caso, al tratarse de una doble relación n a n entre las mismas entidades se trata de unas relaciones bastante exóticas vistas en teoría)*, usando como nombre de los atributos "mandatoryMedications" y "optionalMedications" en la clase "Prescription". Debe asegurarse de que las relaciones expresan adecuadamente la cardinalidad que muestra el diagrama UML.

Finalmente, se piden crear dos relaciones para la clase Allergy. Una de ellas unidireccional desde "Pet" hacia "Allergy"; y otra desde "Allergy" hacia "ActivePrinciple" que representen las que aparecen en el diagrama. Debe asegurarse de que las relaciones expresan adecuadamente la cardinalidad que muestra el diagrama UML.

Test 2 – Modificación del script de inicialización de la base de datos

Parte 2A: Incluir cuatro alergias y dos prescripciones (1 punto)

Modificar el script de inicialización de la base de datos, para que se creen las siguientes alergias (Allergy) y prescripciones (Presciption):

Alergia 1:

- id: 1
- information: "potential Hair loss"
- level: 4
- diagnosisDate: 01/11/2023

Alergia 2:

- id: 2
- information: "Death danger if no medication is provided in time"
- Level: 10
- diagnosisDate: 15/10/2023

Alergia 3:

- id: 3
- information: "potential skin irritation"
- level: 2
- diagnosisDate: 05/08/2023

Alergia 4:

- id: 4
- information: "stomach inflammation danger if no medication is provided in time"
- Level: 8

- diagnosisDate: 10/03/2023

Prescription 1:

- id: 1
- information: “Rimadyl and Metacam optionally”
- dosage: 500
- treatmentStart: 17/12/2024
- treatmentEnd: 25/12/2024
- hoursBetweenDosage: 12

Prescription 2:

- id: 2
- information: “Heartgard and Advantage optionally”
- dosage: 1000
- treatmentStart: 17/12/2024
- treatmentEnd: 31/12/2024
- hoursBetweenDosage: 8

Tenga en cuenta que el orden en que aparecen los INSERT en el script de inicialización de la base de datos es relevante al definir las asociaciones. El formato para introducir fechas en SQL que debe usar es similar al que expresa esta cadena:'2024-12-30'. Es posible que necesite asociar las prescripciones con una visita (si anotó el atributo ‘visit’ de la clase Prescription como @NotNull), puede usar los valores especificados en el siguiente ejercicio (Parte 2B).

Parte 2B: Relacionar las prescripciones con visitas y medicación; así como las mascotas con alergias y estas con los principios activos que las causan (1 punto)

Modificar este script de inicialización de la base de datos para que:

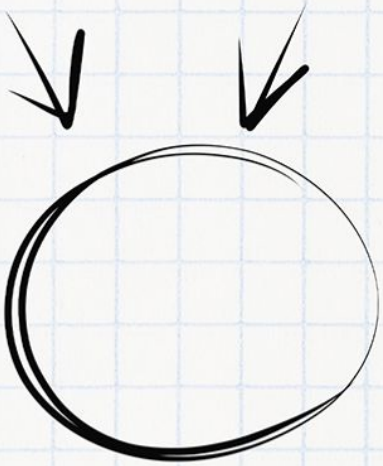
- La Mascota con id 1 se asocie con la Alergia cuyo id es 1 y esta se asocie con el Principio activo cuyo id es 1 (ivermectina).
- La Mascota con id 1 se asocie con la Alergia cuyo id es 2 y esta se asocie con el Principio activo cuyo id es 2 (imidacloprid).
- La Mascota con id 2 se asocie con la Alergia cuyo id es 3 y esta se asocie con el Principio activo cuyo id es 3 (carprofeno).
- La Mascota con id 3 se asocie con la Alergia cuyo id es 4 y esta se asocie con el Principio activo cuyo id es 4 (meloxicam).
- La Prescripción cuyo id es 1 se haga en la Visita con id 1 y tenga como medicación obligatoria (mandatory) la de id 1; así como medicación opcional la de id 2.
- La Prescripción cuyo id es 2 se haga en la Visita con id 2 y tenga como medicación obligatoria (mandatory) la de id 3; así como medicación opcional la de id 4.

Imagínate aprobando el examen

Necesitas tiempo y concentración

Planes	PLAN TURBO	PLAN PRO	PLAN PRO+
Descargas sin publi al mes	10	40	80
Elimina el video entre descargas	✓	✓	✓
Descarga carpetas	✗	✓	✓
Descarga archivos grandes	✗	✓	✓
Visualiza apuntes online sin publi	✗	✓	✓
Elimina toda la publi web	✗	✗	✓
Precios Anual	0,99 € / mes	3,99 € / mes	7,99 € / mes

Ahora que puedes conseguirlo, ¿Qué nota vas a sacar?



WUOLAH

Tenga en cuenta que el orden en que aparecen los INSERT en el script de inicialización de la base de datos es relevante al definir las asociaciones

Test 3 – Creación y modificación de un controlador y un componente frontend de visualización de un Historial de Prescripciones y alergias de las mascotas.

Parte 3A – Creación de controlador de historial de prescripciones y alergias de las mascotas (1 punto)

Modificar la clase "HistoryController" para que responda a peticiones tipo GET en la url: http://localhost:8080/api/v1/pets/{petID}/prescriptionhistory

Para ello, el controlador debe usar el servicio de gestión de prescripciones (PrescriptionService) así como el de gestión de mascotas (PetService).

Es importante que dicho controlador devuelva los datos de las prescripciones, medicación asociada, y las alergias de la mascota. Todo ello en el siguiente formato:

```
{
        petName: "Leo",
        prescriptions: [   {visitDate: "2024-12-10", dosage: 500, treatmentStart: "2024-12-17", treatmentEnd: "2024-12-25",
                            hoursBetweenDosage: 12, mandatoryMedication: ["Rimadyl"], optionalMedication: ["Metacam"] },
                           …,      ]
        allergies: [       {information: "potential Hair loss", level: 4},
                           {information: "Death danger if no medication is provided in time", level: 10}  ]
}
```

Si se pide el historial de una mascota cuyo id no está en la BD se deberá devolver un código de respuesta 404.

Parte 3B - Creación de un componente frontend para la visualización del historial de prescripciones y alergias de las mascotas (1 punto)

Modificar el componente React proporcionado en el fichero "frontend/src/disease/prescriptionhistory /index.js" para que muestre las prescripciones y alergias de una mascota. Para ello debe hacer uso de la API lanzando una petición tipo GET contra la URL api/v1/pets/{petID}/prescriptionhistory.

Este componente debe tomar como propiedad llamada id el identificador de la mascota para la que se debe mostrar el historial. Tras realizar la llamada a la API, el componente debe mostrar el nombre de la mascota como título de nivel 1 (h1) y la información de las prescripciones en sendas columnas de una tabla (se recomienda usar el componente Table de reactstrap). Además, deberá mostrar las alergias en una segunda tabla.

Para poder lanzar esta prueba y comprobar su resultado puede colocarse en la carpeta de frontend y ejecutar el comando npm test y pulsar 'a' en el menú de comandos de jest. Nótese que previamente debe haber lanzado al menos una vez el comando npm install para que todas las librerías de node estén instaladas

Test 4 – Anotar el repositorio de Visitas con una consulta compleja

Modificar la consulta personalizada que puede invocarse a través del método "getFrequentDiagnoses" del repositorio de visitas "VisitRepository" (alojado en el paquete org.springframework.samples.petclinic.visit) que reciba como parámetro un tipo de mascota, una fecha de inicio y una fecha de fin, y devuelva la lista ordenada de enfermedades (de más frecuentes a menos frecuentes expresadas como cadenas de texto) diagnosticadas en las visitas realizadas a las mascotas del tipo pasado en el periodo determinado por las fechas de inicio y fin pasadas. El diagnóstico realizado (enfermedad detectada) en cada visita se especifica en el campo description de la visita (como una cadena de texto).

A continuación, se muestra un ejemplo. Sea la siguiente tabla de visitas de mascotas con alergias a ciertos principios activos y por tanto a los medicamentos mostrados:

Visit id	PetType[1]	Visit Date	Visit description
1	{id: 1, name:"Cat"}	13/11/2024	"Toxoplasmosis"
2	{id: 2, name:"Dog"}	14/11/2024	"Rabies Shot"
3	{id: 6, name:"Hamster"}	17/11/2024	"Rabies Shot"
4	{id: 2, name:"Dog"}	18/11/2024	"Flu"
5	{id: 2, name:"Dog"}	29/11/2024	"Rabies shot"
6	{id: 2, name:"Dog"}	05/12/2024	"Flu"
7	{id: 2, name:"Dog"}	10/12/2024	"Rabies Shot"
8	{id: 1, name:"Cat"}	16/12/2024	"Rabies Shot"

Si invocamos al método con los siguientes valores de los parámetros: petType="dog", startDate=12/11/2024, endDate=15/12/2024; el resultado debería ser una lista con los diagnósticos en el siguiente orden: {"Rabies Shot", "Flu", "Lyme Disease"}.

Test 5 – Implementar una prueba para un algoritmo que muestra medicamentos con principios activos cuyo nombre empieza por una cadena de texto determinada

En la clínica se ha decidido facilitar la labor de búsqueda de medicamentos con ciertos principios activos. Para ello, se ha decidido crear un algoritmo que realizará una búsqueda dentro de un conjunto de medicamentos pasados como primer parámetro que contengan principios activos cuyo nombre empiece por una cadena de texto especificada como segundo parámetro.

Por ejemplo, si tenemos que el medicamento "Heartgard" contiene el principio activo "ivermectina" y el medicamento "Advantage" contiene el principio activo "imidacloprid", y ambos son los únicos con

[1] Obtenido a partir de la mascota asociada con la visita correspondiente

principios activos que empiezan por “i”, el algoritmo devolverá como resultado ambos medicamentos a la búsqueda de medicamentos con principios activos que empiecen por “i”.

Además, hay algún caso límite a considerar, cuando el parámetro indicando la cadena de texto está vacía, el algoritmo debe devolver una colección también vacía.

La interfaz del algoritmo está especificada en la interfaz “MedicationSearchAlgorithm” que se encuentra en el paquete “org.springframework.samples.petclinic.medication.searching”. Y se proporcionan tres implementaciones del algoritmo (una correcta y dos incorrectas).

Modifique la clase de pruebas llamada “MedicationSearchAlgorithmTest” que se encuentra en la carpeta “scr/main/test/…/medication” y especifique tantos métodos con casos de prueba como considere necesarios para validar el correcto funcionamiento del algoritmo. Nótese que la clase tiene un atributo de tipo MedicationSearchAlgorithm llamado algorithm, use dicho atributo como sujeto bajo prueba en todos sus métodos de prueba.

Su implementación del test **no debe usar mocks, ni anotaciones de pruebas de spring** (@DataJpaTest, @SpringBootTest,etc.), **ni tests parametrizados, y todos los métodos anotados con @Test deben ser sin parámetros**.

CONTROL CHECK 2 - GRUPO 3 - 24/25

TEST 1

```
@Getter
@Setter
@Entity
public class Prescription extends BaseEntity {

    @NotNull @Min(1) @Max(1000)
    Integer dosage; // cantidad de medicación

    @NotNull @DateTimeFormat(pattern = "dd/MM/yyyy")
    LocalDate treatmentStart; // fecha de inicio de la
medicación

    @NotNull @DateTimeFormat(pattern = "dd/MM/yyyy")
    LocalDate treatmentEnd;  //fecha de fin de la medicación

    @NotNull @Min(1) @Max(24)
    Integer hoursBetweenDosage;


    String information;


    @ManyToOne @NotNull
    Visit visit;

    @ManyToMany @NotNull
    Set<Medication> mandatoryMedications;

    @ManyToMany
    Set<Medication> optionalMedications;

}
```

```
@Getter
@Setter
@Entity
public class Allergy extends BaseEntity {

    @NotNull @Min(1) @Max(10)
    Integer level;
```


```
    @NotNull @DateTimeFormat(pattern = "dd/MM/yyyy")
    LocalDate diagnosisDate;
    String information;

    @ManyToMany @NotNull
    Set<ActivePrinciple> activePrinciples;

}
```

```
public interface PrescriptionRepository extends
CrudRepository<Prescription, Integer> {

    Optional<Prescription> findById(Integer i);

    List<Prescription> findAll();

    Prescription save(Prescription any);

}
```

```
public interface AllergyRepository extends
CrudRepository<Allergy, Integer> {

    Optional<Allergy> findById(Integer i);

    List<Allergy> findAll();

    Allergy save(Allergy any);

}
```

TEST 2

```
INSERT INTO allergy(id, information, level, diagnosis_date)
VALUES (1, 'potential Hair loss', 4, '2023-11-01'),
(2, 'Death danger if no medication is provided in time', 10,
'2023-10-15'),
(3, 'potential skin irritation', 2, '2023-08-05'),
(4, 'stomach inflammation danger if no medication is provided
in time', 8, '2023-03-10');

INSERT INTO prescription(id, information, dosage,
treatment_start, treatment_end, hours_between_dosage,
visit_id) VALUES
```

```
(1, 'Rimadyl and Metacam optionally', 500, '2024-12-17',
'2024-12-25', 12, 1),
(2, 'Heartgard and Advantage optionally', 1000,'2024-12-17',
'2024-12-31', 8, 2);

INSERT INTO pets_allergies(pet_id, allergies_id) VALUES (1,
1), (1, 2), (2, 3), (3, 4);
INSERT INTO allergy_active_principles(allergy_id,
active_principles_id) VALUES (1,1), (2, 2), (3, 3), (4, 4);

INSERT INTO
prescription_mandatory_medications(prescription_id,
mandatory_medications_id) VALUES (1, 1), (2, 3);
INSERT INTO prescription_optional_medications(prescription_id,
optional_medications_id) VALUES (1, 2) ,(2, 4);
```

TEST 3

```
@RestController
public class HistoryController {

   private final PetService petService;
   private final PrescriptionService prescriptionService;

   @Autowired
   public HistoryController(PetService petService,
PrescriptionService prescriptionService){
       this.petService = petService;
       this.prescriptionService =  prescriptionService;
   }

   @GetMapping("/api/v1/pets/{petId}/prescriptionhistory")
   @ResponseStatus(HttpStatus.OK)
   public ResponseEntity<HistoryDTO> getHistory(@PathVariable
Integer petId){
       Pet pet = petService.findPetById(petId);
       if (pet == null) throw new
ResourceNotFoundException("pet not found");

       HistoryDTO res= new HistoryDTO(pet.getName(),

prescriptionService.getPrescriptionsByPetId(petId));
```

```
        return new ResponseEntity<HistoryDTO>(res,
HttpStatus.OK);
    }
}
```

```
@Getter
@Setter
public class HistoryDTO {
    String petName;
    List<PrescriptionDTO> prescriptions;
    List<AllergyDTO> allergies;

    public HistoryDTO(String petName, List<Prescription> ps){
        this.petName = petName;
        this.prescriptions = ps.stream().map(p -> new
PrescriptionDTO(p)).toList();

        AllergyDTO a1 = new AllergyDTO("potential Hair loss",
4);
        AllergyDTO a2 = new AllergyDTO("Death danger if no
medication is provided in time", 10);

        this.allergies = List.of(a1, a2);
    }
```

```
@Getter
@Setter
public class PrescriptionDTO {
    LocalDate visitDate;
    Integer dosage;
    LocalDate treatmentStart;
    LocalDate treatmentEnd;
    Integer hoursBetweenDosage;
    List<String> mandatoryMedication;
    List<String> optionalMedication;

    public PrescriptionDTO(Prescription p) {
        this.visitDate =
p.getVisit().getDatetime().toLocalDate();
        this.dosage = p.getDosage();
```


```
        this.treatmentEnd = p.getTreatmentEnd();
        this.treatmentStart = p.getTreatmentStart();
        this.hoursBetweenDosage = p.getHoursBetweenDosage();
        this.mandatoryMedication = p.getMandatoryMedications()
                .stream().map(m-> m.getName()).toList();
        this.optionalMedication = p.getOptionalMedications()
                .stream().map(m->m.getName()).toList();
    }
}
```

TEST 4

```
@Query("""
   SELECT v.description
   FROM Visit v
   WHERE v.pet.type = :petType
   AND v.datetime BETWEEN :startDate AND :endDate
   GROUP BY v.description
   ORDER BY COUNT(v.description) DESC
      """)
public List<String> getFrequentDiagnoses(@Param("petType")
PetType petType, @Param("startDate") LocalDateTime startDate,
@Param("endDate") LocalDateTime endDate);
```

TEST 5

```
@Test
public void someTest(){
   // Arrangement / Configuration /Fixture
    Medication m1=createMedication("M1","Acme",500,30);
    m1.setActivePrinciples(Set.of(ap1));
    m1.setActivePrinciples(Set.of(ap2));

   Medication m4=createMedication("M2","Acme",500,30);
   m4.setActivePrinciples(Set.of(ap1));
   m4.setActivePrinciples(Set.of(ap2));

   Medication m2=createMedication("Rimadyl","Acme",500,30);
   m2.setActivePrinciples(Set.of(ap3));
   m2.setActivePrinciples(Set.of(ap4));


   Medication m3 =createMedication("Rimadyl","Acme",500,30);
```

```
   Assertions.assertEquals(2,
algorithm.searchMedicationWithActivePrinciple(Set.of(m1, m4),
"i").size());
   Assertions.assertNotEquals(2,
algorithm.searchMedicationWithActivePrinciple(Set.of(m2),
"i").size());

   Assertions.assertEquals(1,
algorithm.searchMedicationWithActivePrinciple(Set.of(m2),
"").size());

}
```