

Enunciado de Práctica

Diseño y Pruebas Unitarias

Temática

El dominio de esta práctica es el mismo que habéis estado trabajando en la parte de análisis, y que conocéis en detalle: el *Sistema integral personalizado de la Historia Clínica electrónica*.

Se pide implementar y probar una versión simplificada del caso de uso *Supervisar tratamiento*. En otras palabras, la creación de test y la implementación del código que pasa esos test (*o a la inversa, el orden lo decidís vosotros*). En cualquier caso, se recomienda escribir el código en orden creciente de complejidad, tal y como se propone en este documento, e ir probándolo progresivamente a lo largo del desarrollo.

Comenzaremos formalizando algunas clases consideradas básicas (igual que lo son String, BigDecimal, etc.), dado que su única responsabilidad es la de guardar ciertos valores. Todas ellas irán en un paquete denominado **data**.

El paquete **data**

El paquete **data** contendrá cuatro clases, la única responsabilidad de las cuales es la de guardar un valor de tipo primitivo o clase de java (concretamente String y byte[]). Pensad los diversos motivos que hacen que sea conveniente hacerlo así (aparte de que, como ya sabéis, hay algún que otro *code smell* que hace alusión a este aspecto).

Se trata de las clases HealthCardID (Código de Identificación Personal del paciente, o abreviadamente CIP), a guardar como un String, ProductID (código UPC -*Universal Product Code*), un String, ePrescripCode (el código de la prescripción médica), también un String y DigitalSignature (la firma digital del médico), a representar como un byte[].

A continuación se presenta la clase HealthCardID.

La clase **HealthCardID**

Representa el número de la tarjeta sanitaria, como persona registrada en el HNS¹. Permite vincular la información sanitaria de un individuo, generada en cualquier servicio o centro de salud a nivel estatal. Es el que se encuentra grabado en la banda magnética de la tarjeta, y consiste en una secuencia de 16 caracteres alfanuméricos.

¹ La clase conceptual **HealthNationalService** en el Modelo del Dominio pasado como referencia (SNS durante el análisis).

En nuestro caso se utiliza en el momento de concertar la visita al centro de salud por parte de los pacientes, quedando registrado en la agenda de visitas concertadas, y que posteriormente será recuperado (primer paso del caso de uso *Supervisar tratamiento*).

Esta es su implementación:

```
package data;

/**
 * The personal identifying code in the National Health Service.
 */

final public class HealthCardID {

    private final String personalID;

    public HealthCardID(String code) { this.personalID = code; }

    public String getPersonalID() { return personalID; }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        HealthCardID hcardID = (HealthCardID) o;
        return personalID.equals(hcardID.personalID);
    }

    @Override
    public int hashCode() { return personalID.hashCode(); }

    @Override
    public String toString() {
        return "HealthCardID{" + "personal code='" + personalID + '\'' + '}';
    }
}
```

Definid vosotros las clases ProductID, ePrescripCode y DigitalSignature. Para ePrescripCode y ProductID podéis escoger vosotros mismos el formato o patrón admitido para este identificador (longitud y caracteres admitidos).

Estas clases serán **inmutables** (por eso el **final** y la no existencia de setters), y tienen definido un **equals**, que comprueba si el valor de dos instancias coincide. Estas clases se denominan también **clases valor**, ya que de sus instancias nos interesa tan sólo el valor.

Es conveniente añadir las excepciones que consideréis oportunas. Por ejemplo, para el caso de la clase HealthCardID, podemos definir las dos situaciones siguientes: que al constructor le llegue null (objeto sin instanciar), y también un código de identificación mal formado.

¿Cuál es el caso de las otras clases básicas?

Implementar y realizar test para estas clases (es suficiente con comprobar las excepciones consideradas).

El caso de uso *Supervisar tratamiento*

En lo que queda de documento se presenta el caso de uso a desarrollar. Se trata de **Supervisar tratamiento**² y, concretamente para el siguiente escenario: el médico recurre directamente a la IA de apoyo a la toma de decisiones para obtener sugerencias sobre la adecuación del tratamiento.

Tras dialogar con la IA (no es necesario emular un bucle), se determina modificar la prescripción médica realizando distintos cambios que incluirán, como mínimo, los siguientes: 1) añadir un medicamento específico, con las pautas de administración recomendadas por la IA; 2) modificar la dosis de uno de los medicamentos ya existentes; y, por último, 3) suprimir una línea de prescripción.

Este escenario corresponde al *Diagrama de Casos de Uso* utilizado como referencia durante el análisis y al *Diagrama de Secuencia del Sistema* del anexo. Además, disponéis de los contratos de las operaciones para los eventos de entrada al sistema, los cuales deberán cumplirse.

Los casos de uso incluidos para este escenario son: **Dar soporte por IA en la toma de decisiones, Crear línea de prescripción y Registrar prescripción**. El caso de uso **Programar próxima revisión** se obviará.

Se utilizará la clase del dominio *ConsultationTerminal*³ como *controlador de fachada* para el caso de uso. Representa el terminal o dispositivo hardware, ubicado en la consulta médica, que permite acceder al sistema de la HCE. Será la clase responsable de manejar los eventos de entrada. Esta clase se explica más adelante.

Además, se implementarán los servicios involucrados, que incluiremos en un paquete denominado *services*. Este se presenta a continuación.

Servicios involucrados

Agruparemos los servicios involucrados en este caso de uso en un paquete denominado *services*.

El caso de uso **Supervisar tratamiento** requiere de un servicio externo. Por supuesto, estamos hablando del *SNS* (componente *HealthNationalService*), cuya interacción con el módulo de consulta médica se relaciona a continuación:

HealthNationalService. Interviene en tres ocasiones en el transcurso de la revisión médica (consultar el DSS del anexo):

- `getMedicalHistory(HealthCardID cip)`: a partir del número de tarjeta sanitaria (un *HealthCardID*), recupera la HCE del paciente.

² Como estamos realizando pruebas unitarias, no se presenta ninguna interfaz de usuario, sino que tendremos métodos que son los que usará la interfaz de usuario (los eventos de entrada). Tal y como se presenta en el tema de *Patrones GRASP*, implementaremos y probaremos un *controlador de fachada* para el caso de uso (es decir, un objeto del dominio escogido específicamente como controlador).

³ Corresponde a la clase del dominio *ConsultationTerminal* (Modelo del Dominio pasado como referencia).

Excepciones: ConnectException⁴ (la red falla), HealthCardIDException (el identificador del paciente no está registrado en el HNS).

- getMedicalPrescription(HealthCardID cip, String illness): recupera la prescripción médica del paciente correspondiente a la enfermedad indicada (illness).

Excepciones: ConnectException, HealthCardIDException y AnyCurrentPrescriptionException (no existe ninguna prescripción médica activa del paciente para esa enfermedad).

- sendHistoryAndPrescription(HealthCardID cip, MedicalHistory hce, String illness, MedicalPrescription mPresc): transmite al HNS la versión actualizada de la HCE y de la prescripción médica del paciente para la enfermedad involucrada en el caso de uso, conforme a los cambios aplicados. Ello supone generar por parte del HNS un código de tratamiento. En consecuencia, la copia local de la prescripción médica se actualiza.

Excepciones: ConnectException, HealthCardIDException, AnyCurrentPrescriptionException y NotCompletedMedicalPrescription (la prescripción médica está sin completar).

Por simplicidad, el método getMedicalPrescription(cip, illness) retorna una instancia de la clase MedicalPrescription (la prescripción médica de ese paciente para esa enfermedad, no toda la e-receta). Del mismo modo, sendMedicalHistoryAndPrescription(...) recibe como argumento y retorna una instancia de MedicalPrescription, que hace referencia a la prescripción médica en cuestión. Es por este motivo que obviaremos la implementación de la eReceta (la clase ePrescription en el Modelo del Dominio).

Operación interna:

- generateTreatmentCodeAndRegister(MedicalPrescription ePresc): si todo es correcto, se genera un nuevo código de prescripción para el tratamiento, quedando así actualizado en el sistema y vigente para el siguiente periodo de prescripción. Retorna la instancia de MedicalPrescription con el código asignado.

Excepción: ConnectException.

La definición del servicio HealthNationalService queda tal y como se muestra a continuación.

```
package services; // Package for involved services

/**
 * External services for managing and storing ePrescriptions from population and IA support
 */

public interface HealthNationalService {

    MedicalHistory getMedicalHistory (HealthCardID cip)
        throws ConnectException, HealthCardIDException;

    MedicalPrescription5 getMedicalPrescription(HealthCardID cip, String illness)
        throws ConnectException, HealthCardIDException,
               AnyCurrentPrescriptionException;
}
```

⁴ Excepción proporcionada por la API. Señala un error producido al intentar conectar un socket a una dirección y puerto remotos. Por lo general, es debido a que la conexión fue rechazada remotamente (e. g., ningún proceso fue escuchado en la dirección/puerto remoto).

⁵ Para simplificar, por lo que respecta a la clase ePrescription se manejará tan sólo la parte de la prescripción médica.

```

MedicalPrescription6 sendHistoryAndPrescription(HealthCardID cip,
    History hce, String illness, MedicalPrescription mPresc)
    throws ConnectException, HealthCardIDException,
    AnyCurrentPrescriptionException, NotCompletedMedicalPrescription;

// Internal operation

MedicalPrescription generateTreatmentCodeAndRegister(MedicalPrescription ePresc)
    throws ConnectException;
}

```

Adicionalmente, definiremos una interface para representar la funcionalidad relacionada con la IA involucrada en el escenario descrito aquí. Se trata de la IA de apoyo a la toma de decisiones que el médico invoca explícitamente en el paso de ajustar el tratamiento. La llamaremos **DecisionMakingAI**.

DecisionMakingAI. Interviene en tres ocasiones en el transcurso de la revisión médica (DSS del anexo):

- initDecisionMakingIA(): pone en marcha la IA por petición del médico.
Excepción: AIException indica algún problema de funcionamiento en la invocación de la IA.
- getSuggestions(String prompt): El sistema transmite el prompt a la IA. La respuesta proporcionada, conteniendo ciertas sugerencias, es retornada mediante un String que será mostrado por pantalla.
Excepción: BadPromptException se produce si el prompt introducido no es suficientemente claro para la IA o presenta alguna inconsistencia.
- parseSuggest(String aiAnswer): Una vez obtenida la respuesta, el sistema solicita a la IA que desglose el resultado en indicaciones específicas siguiendo un determinado patrón. Como resultado, el sistema presentará esas mismas sugerencias, pero esta vez en formato tabular, destinando una línea para cada una de ellas. Se trata de ofrecer el resultado de la IA de la manera más inteligible posible para el médico, quien decidirá para cada una de las sugerencias si la desecha o la acepta.
El tipo de retorno es un List<Suggestion> (una lista de sugerencias).

Este es el patrón ofrecido: cada entrada a la tabla corresponde a un ajuste en la medicación sugerido por la IA. Mediante una I, una E y una M se indica Inserción, Eliminación y Modificación de líneas de prescripción, respectivamente. A continuación, el identificador del producto (un ProductID) y, finalmente, las instrucciones, destinando una columna de la tabla a cada uno de los campos relativos a las pautas de medicación (uno a uno y en orden).

En el caso de E (Eliminación), no es necesario incorporar instrucciones.

Para el caso de M (Modificación), tan sólo rellena las columnas asociadas a aquellas partes de las pautas de medicación que la IA sugiere modificar (e. g. la dosis). El resto de columnas estarán vacías.

Ejemplos:

- <I, 243516578917, BEFORELUNCH, 15, 1, 1, DAY, Tomar con abundante agua> indica Inserción del producto con código 243516578917, acompañado de las pautas de medicación sugeridas (tomar antes de comer durante 15 días una dosis

⁶ Igual que para el método getMedicalPrescription(...), se simplifica considerando que se retorna tan sólo la parte de la prescripción médica (una instancia de MedicalPrescription).

de 1 unidad, con una frecuencia de 1 al día, y acompañado de la instrucción “Tomar con abundante agua” (véase sección siguiente).

- <R, 640557143200> indica Eliminación del producto con ProductID 640557143200.
- <M, 243516578917, , , 3, , ,> indica Modificación, en este caso tan sólo de la dosis, pasando ésta a ser de 3 unidades en la línea del producto con código 243516578917.

Inciso: Definid vosotros la clase Suggestion.

La definición de DecisionMakingAI queda tal y como se muestra a continuación.

```
public interface DecisionMakingAI {  
  
    void initDecisionMakingAI() throws AIException;  
  
    String getSuggestions(String prompt) throws BadPromptException;  
  
    List<Suggestion> parseSuggest(String aiAnswer);  
}
```

Los servicios se inyectarán a la/s clase/s pertinente/s, por ejemplo, mediante un *setter*.

A continuación, se presentan el resto de las clases relacionadas con la funcionalidad del módulo de consulta médica. Agruparemos todas ellas en el paquete medicalconsultation.

El paquete medicalconsultation

Este paquete contiene las clases directamente involucradas en el uso del sistema de historia clínica electrónica, en colaboración directa con los servicios descritos.

Comenzaremos con las clases que representan las pautas de administración de un medicamento y su posología. Como ya sabéis, las pautas incluyen la siguiente información: el momento del día en que debe administrarse ese medicamento, la duración (número de días), la posología y, opcionalmente, las instrucciones específicas para su administración.

Por su parte, la posología incluye la dosis (unidades del medicamento) y la frecuencia de las tomas, que representaremos mediante un número y una unidad temporal (e. g. para indicar 24 horas se puede hacer de dos formas: 1) frecuencia: 24 y unidad de frecuencia: HOUR; y 2) frecuencia: 1 y unidad de frecuencia: DAY). Para ello destinamos la clase Polology.

Los constructores, getters y setters nos permitirán consultar, crear y modificar una determinada línea de prescripción.

Las clases Posology y TakingGuideline

La clase Posology representa la información relativa a la posología de un medicamento. Esta es su estructura:

```
public class Posology { // A class that represents the posology of a medicine
    private float dose;
    private float freq;
    private FqUnit freqUnit;

    public Posology (float d, float f, FqUnit u) { . . . } // Initializes attributes
    ??? // the getters and setters
}
```

siendo la clase FqUnit el enumerado siguiente:

```
public enum FqUnit {
    HOUR, DAY, WEEK, MONTH;
}
```

La clase TakingGuideline representa la información relativa a las pautas de administración de un medicamento. Esta es su estructura:

```
public class TakingGuideline { // Represents the taking guidelines for a medicine
    private dayMoment dMoment;
    private float duration;
    private Posology posology;
    private String instructions;

    public TakingGuideline(dayMoment dM, float du, float d, float f,
                           FqUnit fu, String i){ . . . } // Initializes attributes
    ??? // the getters and setters
}
```

siendo la clase dayMoment el enumerado siguiente:

```
public enum dayMoment {
    BEFOREBREAKFAST, DURINGBREAKFAST, AFTERBREAKFAST, BEFORELUNCH,
    DURINGLUNCH, AFTERLUNCH, BEFOREDINNER, DURINGDINNER, AFTERDINNER,
    BEFOOMEALS, DURINGMEALS, AFTERMEALS;
}
```

Las operaciones de ambas clases son esenciales en este caso de uso para ajustar las pautas de medicación, de acuerdo con el criterio del médico.

Ambas clases incluidas en el paquete medicalConsultation.

Las clases MedicalHistory y MedicalPrescription

Las clases MedicalHistory y MedicalPrescription corresponden a las clases con ese mismo nombre del Modelo del Dominio de referencia (documento *DCI-HCE-SupervTrat-SolRef.jpg*)⁷.

Prescindimos de las clases ProductCatalog y ProductSpecification, dado que la consulta al catálogo oficial no se incluye en el escenario planteado aquí.

La estructura, aunque incompleta, de la clase MedicalHistory es la siguiente:

```
package medicalconsultation;

/*
 * Package for the classes involved in the use case Supervise treatment
 */

public class MedicalHistory // A class that represents a medical history

    private HealthCardID cip; // the CIP of the patient
    private int membShipNumb; // the membership number of the family doctor
    private String history; // the diverse annotations in the patient's HCE

    public MedicalHistory (HealthCardID cip, int memberShipNum) { . . . }
        throws IncorrectParametersException;
        // Makes its inicialization

    public void addMedicalHistoryAnnotations (String annot)
        // Adds new annotations to the patient history

    public void setNewDoctor (int mshN) // Modifies the family doctor for patient

    ??? // the getters
}
```

La excepción IncorrectParametersException indica algún tipo de error en los parámetros proporcionados como argumento.

Completad la implementación de esta clase, teniendo en cuenta los comentarios proporcionados.

La estructura, aunque incompleta, de la clase MedicalPrescription es la siguiente:

```
public class MedicalPrescription // A class that represents medical prescription

    private HealthCardID cip; // the healthcard ID of the patient
    private int membShipNumb; // the membership number of the family doctor
    private String illness; // illness associated
    private ePrescripCode prescCode; // the prescription code
    private Date prescDate; // the current date
```

⁷ Por simplicidad, obviamos la clase ePrescription, que aparece en el Modelo del Dominio, la cual representa la e-receta.

```

private Date endDate;      // the date when the new treatment ends
private DigitalSignature eSign; // the eSignature of the doctor
???
// Its components, that is, the set of medical prescription lines

public MedicalPrescription (. . .) { . . . } // Makes some inicialization
public void addLine(ProductID prodID, String[] instruc)
throws ProductAlreadyInPrescriptionException,
IncorrectTakingGuidelinesException { . . . }

public void modifyDoseInLine(ProductID prodID, float newDose)
throws ProductNotInPrescriptionException { . . . }

public void removeLine(ProductID prodID)
throws ProductNotInPrescriptionException { . . . }

???
// the getters and setters for some of the class members
}

```

- **addLine(ProductID prodID, String[] instruc):** añade una línea de prescripción médica. Recibe dos argumentos: prodID (el código del producto) y instruc (un array de String donde cada elemento del array, uno a uno y en orden, contiene los valores introducidos por el médico para cada uno de los ítems de información asociados a las pautas y posología de ese medicamento).
- **modifyDoseInLine(ProductID prodID, float newDose):** se aplica sobre la línea de prescripción del producto prodID, para reemplazar la dosis por un nuevo valor (newDose).
Habría un método específico para la modificación de cada uno de los datos relativos a las pautas de administración (no es necesario implementarlos).
- **removeLine(ProductID prodID):** elimina una línea de prescripción médica. Recibe un solo argumento (prodID), el código del producto a eliminar del tratamiento.

Excepciones: La excepción **ProductAlreadyInPrescriptionException** indica que el producto ya se encuentra en la prescripción médica. Por otro lado, **ProductNotInPrescriptionException** que el producto no forma parte de la prescripción, y **IncorrectTakingGuidelinesException** que el formato de la pauta o posología son incorrectos, o bien la información es incompleta.

Completad la implementación de esta clase teniendo en cuenta que debería resolver la búsqueda de líneas de prescripción de forma efectiva en las operaciones de modificación y eliminación.

¿Qué ocurre con las componentes de **MedicalPrescription**? ¿Qué implementación has escogido para ellas?

Implementar y realizar test para las clases del paquete medicalconsultation (se recomienda combinar ambas tareas en paralelo, con la finalidad de ir probando poco a poco el código, conforme se va desarrollando).

La clase controladora **ConsultationTerminal**

Se trata de la clase utilizada como *controlador de fachada* (patrón *Controlador*, dentro de los Patrones GRASP) del caso de uso **Supervisar tratamiento**. Gestiona, por tanto, los eventos del caso de uso, haciendo de intermediario entre la interfaz de usuario y el sistema.

La estructura, aunque incompleta, de la clase **ConsultationTerminal**, es la siguiente:

```
public class ConsultationTerminal {  
    ??? // Class members  
    ??? // The constructor/s  
    // Input events  
    public void initRevision(HealthCardID cip, String illness)  
        throws ConnectException, HealthCardIDException,  
        AnyCurrentPrescriptionException { . . . }  
  
    public void enterMedicalAssessmentInHistory(String assess) { . . . };  
  
    public void initMedicalPrescriptionEdition() { . . . };  
  
    public void enterMedicineWithGuidelines(ProductID prodID, String[] instruc)  
        throws ProductAlreadyInPrescriptionException,  
        IncorrectTakingGuidelinesException { . . . }  
  
    public void modifyDoseInLine(ProductID prodID, float newDose)  
        throws ProductNotInPrescriptionException { . . . }  
  
    public void removeLine(ProductID prodID) throws  
        ProductNotInPrescriptionException { . . . }  
  
    public void enterTreatmentEndingDate(Date date) throws  
        IncorrectEndingDateException { . . . }  
  
    public void finishMedicalPrescriptionEdition() { . . . }  
  
    public void stampeeSignature() throws eSignatureException { . . . }  
  
    public MedicalPrescription sendHistoryAndPrescription()  
        throws ConnectException, HealthCardIDException,  
        AnyCurrentPrescriptionException,  
        NotCompletedMedicalPrescription { . . . }  
  
    public void printMedicalPrescrip() throws printingException { . . . }  
  
    // Input events regarding the AI operation  
    public void callDecisionMakingAI() throws AIEception { . . . }  
    public void askAIForSuggest(String prompt) throws BadPromptException  
        { . . . }  
    public void extractGuidelinesFromSugg() { . . . };  
  
    // internal operations  
    private void createMedPrescriptionLine(ProductID prodID, String[] instruc)  
        { . . . };  
  
    private void setPrescDateAndEndDate(Date date) { . . . };  
    (. . .) // Setter methods for injecting dependences  
}
```

A continuación, se presentan a grandes rasgos dichos métodos. Los contratos de referencia de los eventos de entrada se pueden consultar en el documento *ModeloCasosUsoHCE-ParteContratos.pdf*.

Eventos de entrada:

- `initRevision(HealthCardID cip, String illness)`: Se inicia la revisión del paciente con código cip. Mediante conexión con el HealthNationalService, se procede a la descarga de la historia clínica y la prescripción médica del paciente para la enfermedad illness.
Excepciones: ConnectException, AnyCurrentPrescriptionException y HealthCardIDException, presentadas anteriormente.
- `enterMedicalAssessmentInHistory(String assess)`: El médico reporta las valoraciones observadas durante la visita de revisión en la historia clínica del paciente.
- `initMedicalPrescriptionEdition()`: El médico indica el inicio del proceso de edición de la prescripción médica correspondiente a una enfermedad.
- `enterMedicineWithGuidelines(ProductID prodID, String[] instruc)`: El médico introduce un medicamento en la prescripción médica con toda la información relativa a las pautas de medicación y posología. El argumento instruc (un array de String) proporciona toda esa información, tal y como se ha detallado anteriormente para el método `addLine(...)` de MedicalPrescription. Se crea así una línea de prescripción médica asociada a ese medicamento.
Excepciones: ProductAlreadyInPrescriptionException y IncorrectTakingGuidelinesException, presentadas anteriormente.
- `modifyDoseInLine(ProductID prodID, float newDose)`: se aplica sobre la línea de prescripción del producto prodID para reemplazar la dosis por un nuevo valor (newDose).
Habría un método específico para la modificación de cada uno de los datos relativos a las pautas de administración (no es necesario implementarlos).
Excepción: ProductNotInPrescriptionException, presentada anteriormente.
- `removeLine(ProductID prodID)`: elimina la línea de prescripción médica correspondiente a ese medicamento (prodID).
Excepción: ProductNotInPrescriptionException, presentada anteriormente.
- `enterTreatmentEndingDate(Date date)`: El médico introduce la fecha de finalización para el tratamiento. Como fecha de prescripción se establece la fecha actual.
Excepciones: IncorrectEndingDateException, en el caso que la fecha proporcionada no sea adecuada (fecha actual o demasiado cercana), o bien situada en el pasado.
- `finishMedicalPrescriptionEdition()`: El médico indica la finalización del proceso de edición.
- `stampeeSignature()`: El médico incorpora su firma electrónica en la prescripción médica.
Excepciones: eSignatureException, en el caso que se produzca algún problema al estampar la firma electrónica.
- `sendHistoryAndPrescription()`: El médico procede a transmitir la historia clínica y la prescripción médica al HealthNationalService para su registro remoto. Si todo es correcto, el HNS le asigna un código de tratamiento. La prescripción queda almacenada remotamente, actualizada y vigente para el siguiente periodo de tratamiento, y la retorna de nuevo al sistema.
Excepciones: ConnectException, HealthCardIDException, AnyCurrentPrescriptionException y NotCompletedMedicalPrescription para las situaciones reflejadas en los contratos de operación pasados como referencia.
- `printMedicalPrescript()`: El médico lanza la orden de imprimir la hoja de tratamiento. *No se pide su implementación, ni tampoco su excepción.*

Eventos de entrada relacionados con la IA:

- `callDecisionMakingAI()`: El médico invoca la IA de soporte a la toma de decisiones.
- `askAIForSuggest(String prompt)`: El médico dialoga con la IA mediante prompts.

- `extractGuidelinesFromSugg()`: El médico solicita desglosar la respuesta proporcionada por la IA en indicaciones específicas (una lista de sugerencias), para ajustar la prescripción médica.

Excepciones relacionadas: `AIException` y `BadPromptException`, presentadas anteriormente.

Operaciones internas:

- `createMedPrescriptionLine(ProductID prodID, String[] instruc)`: El sistema crea una línea de prescripción con todos los datos relativos a las pautas de medicación proporcionados.
- `setPrescDateAndEndDate(Date date)`: Una vez aplicada la firma electrónica, el sistema recopila la fecha actual e inserta la fecha de finalización del tratamiento (`date`), dejando la prescripción lista para transmitir al SNS.

Excepto `IncorrectEndingDateException` y `eSignatureException`, las excepciones de esta clase son las ya presentadas anteriormente para las clases y servicios relacionados. Además, también podéis consultarlas en los contratos de referencia (*ModeloCasosUsoHCE-ParteContratos.pdf*).

Consideraciones:

- No hace falta contemplar la parte correspondiente a los servicios de impresión. Es por ello por lo que no se pide la implementación del método ni de las excepciones relacionadas.
- **Deberán tratarse también las situaciones descritas en las precondiciones** (documento *ModeloCasosUsoHCE-Partedcl.pdf*), para detectar si se han completado con éxito los pasos que preceden a cada evento del caso de uso. Manejaréis una única excepción: `ProceduralException`, para representar todas las situaciones relacionadas con este aspecto, las cuales están detalladas específicamente para cada evento de entrada en el documento mencionado.
- Por lo que respecta a las excepciones correspondientes a las clases del paquete `data`, **quedan para vosotros** (deberán ser incorporadas en las cabeceras de los métodos).
- Definiréis las excepciones como clases propias (subclases de `Exception` –excepciones *cheatables*) y, adicionalmente, la excepción proporcionada por la API: `ConnectException`, tal y como aparece en las cabeceras de los métodos.

Implementar y realizar test para el caso de uso descrito aquí, utilizando dobles para los servicios colaboradores.

Consideraciones generales

- Deberíais resolver esta práctica **en grupos de entre dos y tres personas**
- Esta práctica tiene un valor de un **20%** sobre la nota final y **no es recuperable**.
- Utilizaréis el sistema de **control de versiones** git y un **repositorio remoto**, a fin de coordinarlos con vuestros compañeros (e. g. GitHub o GitLab –podrán ser repositorios privados). Algunas recomendaciones:
 - Cada vez que hagáis un test y el sistema lo pase, haced un commit. Nunca incorporar a la rama remota código que no ha pasado un test.
 - Cada vez que apliquéis un paso de refactoring en el código, haced un commit indicando el motivo que os ha llevado a hacerlo (¿quizás algún *code smell* o principio de diseño?), así como del refactoring aplicado.
 - Con el fin de facilitar los test, podéis definir otros constructores además de los sugeridos aquí, simplificando la inicialización de las clases.
 - Es recomendable ir trabajando cada miembro del equipo en ramas distintas, para así lograr una mejor colaboración y sincronización de vuestro trabajo (e. g. desarrollo de distintos requisitos/funcionalidades/ramas por separado).

- Como entregaréis un ZIP con el directorio del proyecto, entregareis también el repositorio git (subdirectorio .git), por lo que podré comprobar los commits (*¡ no os lo dejéis para el último día y colaborad todos los miembros del equipo !*).
- Por lo que respecta al SUT (System Under Test), los eventos de entrada **deben satisfacer los contratos** de referencia facilitados (documento *ModeloCasosUsoHCE-ParteContratos.pdf* –carpeta *Práctica-Proyecto/Soluciones/Dominio* del CV), los cuales cumplen con el planteamiento expuesto aquí.
- **Nivel de exhaustividad en los test:**
 - Para las clases del paquete data es suficiente con probar las excepciones.
 - **Donde las pruebas deben ser exhaustivas** es en el paquete medicalconsultation. **Deberán tratarse todas las situaciones posibles** de cada escenario, así como todas sus situaciones excepcionales. Para ello **es imprescindible planear una cierta estructura para el código de test**, tanto para las clases de test, como para los test dobles (e. g., podrían definirse por separado los distintos casos de test: los de éxito y los de fracaso).
 - Además, es recomendable recurrir a la definición de **interfaces de test**, así como la definición de **métodos default**, tal y como se ha sugerido en clase para la resolución de los problemas de la colección.
- Os indico, además, **cómo testear la clase controladora:**
 - Como ya se ha mencionado anteriormente, los eventos de entrada del caso de uso no se testeán individualmente. Cada evento es testeado precediéndolo de los eventos que van por delante en el caso de uso. De esta forma se va probando el progreso del caso de uso. En el caso de no seguir el orden establecido se lanza la excepción **ProceduralException**, asociada a las **precondiciones** de dichos eventos.
 - No hagáis test dobles complicados para poder aprovecharlos más de una vez. Se trata de definir **test dobles lo más simples posible**, con objeto de probar los distintos escenarios. En general, un buen enfoque consiste en definir dobles por separado para diferenciar entre distintas situaciones.
¡ Los test dobles no necesitan testearse !. Debéis enfocaros exclusivamente en el SUT. Estos serán testados indirectamente al testear las clases de producción.
 - Los test dobles pueden definirse internamente en las clases de test, o bien por separado, en paquetes separados.

Entrega

Un **ZIP** que contenga:

- El **proyecto desarrollado** (podéis utilizar IntelliJ IDEA o cualquier otro entorno).
- Un **informe** en el que expliquéis con vuestras palabras el/los criterios empleados para tomar vuestras decisiones de diseño (principios SOLID, patrones GRASP, etc.), y justificación de éstas, si es el caso.
Enumerad también los **métodos de refactoring** aplicados para resolver los posibles **code smell** detectados, ya sea en vuestro código o en el diseño que se propone en este enunciado.
En cuanto a las situaciones que habéis probado en cada uno de los test realizados, **por favor, NO explicarlas !**. Tan sólo si hay algún aspecto relevante que valga la pena mencionar, o bien cualquier detalle que pueda ayudar a valorar mejor vuestro trabajo.

Como siempre, haced la entrega a través del CV **tan sólo uno de los miembros del equipo**, indicando el nombre de vuestros compañeros.

Anexo. DSS adaptado al diseño aquí descrito

