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 receta 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.

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.

1. El paquete data

El paquete data contendrá tres 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*), también un String, y DigitalSignature (la firma digital del médico), a representar como un byte[].

Clase HealthCardID

Representa el código de identificación asignado al paciente, como persona registrada en el HNS[1]. 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 sanitaria.

[1] La clase conceptual HealthNationalService en el modelo del dominio pasado como referencia (el SNS en inglés).

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 y DigitalSignature.

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

Podéis añadir las excepciones que consideréis oportunas. 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).

2. 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 escenario en que el médico considera oportuno añadir un número determinado de líneas de prescripción al tratamiento (correspondiente al diagrama de casos de uso y al diagrama de secuencia del sistema de eReceta utilizados como referencia durante el análisis).
El caso de uso incluido para este escenario es: **Crear línea de prescripción**.

Definiremos una clase llamada ConsultationTerminal[3] para implementar el caso de uso. Será la clase responsable de manejar los eventos de entrada (*controlador de fachada*).

Además, se implementarán los servicios involucrados. Estos se presentan a continuación.

Servicios involucrados

Agruparemos los servicios involucrados en un paquete denominado services.

La parte central del caso de uso **Supervisar tratamiento** requiere de un servicio externo. Por supuesto, estamos hablando del HNS, cuya interacción con el módulo de consulta médica se relaciona a continuación:

- HealthNationalService. Interviene en cuatro ocasiones en el transcurso de la revisión médica (consultar el *DSS-SupervTrat-SolRef.jpg*):
  - getePrescription(HealthCardID HC-ID): a partir del número de tarjeta sanitaria (un HealthCardID), verifica la existencia de una e-receta asociada a ese paciente, y en ese caso procede a descargarla.
  - getProductsByKW(String keyWord): obtiene una lista de medicamentos que contienen la palabra clave keyWord, tras una operación de búsqueda en el catálogo de productos, almacenado remotamente en el HNS.
  - getProductSpecific(int opt): accede de nuevo al catálogo de productos, a fin de obtener la especificación del producto escogido por el médico de entre los resultados de la búsqueda anterior (opción opt de la lista).
  - sendePrescription(ePrescription ePresc): transmite al HNS la versión actualizada de la e-receta, de acuerdo a los cambios aplicados a la prescripción médica, al finalizar la visita. Ello supone generar por parte del HNS un código de tratamiento, así como calcular la temporización de las dispensaciones para el periodo de tratamiento establecido por el médico. En consecuencia, la copia local de la e-receta (en particular, la información relativa a la prescripción médica) se actualiza.

---

[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).

Por simplicidad, manejaremos tan sólo la parte de la prescripción médica, obviando la parte correspondiente a las dispensaciones en farmacia. Eso significa que tanto el método getePrescription(HC-ID), como el método sendePrescription(…) retornan una instancia de la clase MedicalPrescription[4]. Adicionalmente, sendePrescription(…) recibe una instancia de MedicalPrescription como argumento.
Por ese motivo se obvia la definición de las clases ePrescription y Dispensing.

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

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

/**
 * External service for managing and storing ePrescriptions from population
 */

public interface HealthNationalService {

    MedicalPrescription[5] getePrescription(HealthCardID hcID)
        throws HealthCardException, NotValidePrescriptionException,
               ConnectException;

    List<ProductSpecification> getProductsByKW(String keyWord)
        throws AnyKeyWordMedicineException, ConnectException;

    ProductSpecification getProductSpecific(int opt)
       throws AnyMedicineSearchException, ConnectException;

     MedicalPrescription[6] sendePrescription(MedicalPrescription[7] ePresc)
        throws ConnectException, NotValidePrescription, eSignatureException,
                NotCompletedMedicalPrescription;
}
```

Este servicio se inyectará a la clase pertinente mediante un setter.

Las situaciones excepcionales a las que hacen referencia cada una de las excepciones son las mismas que para los eventos de entrada. La mayoría las podéis consultar en los contratos de referencia (documento *ModeloCasosUsoeReceta-ParteContratos.pdf*). En otro caso (AnyKeyWordMedicineException y AnyMedicineSearchException), se presentan más adelante, junto con los eventos de entrada.

---

[4] A efectos estrictamente de llevar a cabo la revisión médica por parte del médico, consideraremos que la información asociada a esta clase es suficiente.

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

[6] Igual que para el método getePrescription(…), se simplifica considerando que se retorna tan sólo la parte de la prescripción médica (una isntancia de MedicalPrescription).

[7] Idem que la nota anterior por lo que respecta al argumento que recibe.

Por otro lado, para simular el paso de obtener la identificación del paciente, registrada en la agenda de visitas concertadas que opera en el centro de salud, utilizaremos la siguiente componente:

- ScheduledVisitAgenda, que interviene al inicio, mediante el siguiente método:
  - getHealthCardID(): retorna un HealthCardID, el código identificativo del paciente. Esta es la cabecera del método:

```
HealthCardID getHealthCardID() throws HealthCardException;
```

  Como veis, se considera la excepción HealthCardException.

Queda para vosotros la definición de esta componente, a incorporar también en el paquete services.

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

3. El paquete medicalconsultation

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. 24 horas -frecuencia: 24 y unidad; hora). Para ello destinamos la clase Polology.

Sus operaciones serán los constructores, getters y setters (los cuales permitirán crear y modificar una determinada línea de prescripción).

Las clases Posology y TakingGuideline

Como sabéis, 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 of a medicine

    private float dayMoment;
    private float duration;
    private String instructions;
    private Posology posology;

  public TakingGuideline(float dM, float du, String i, float d, float f, FqUnit u)
                                        { . . . }  //  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,
    BEFOEMEALS, DURINGMEALS, AFTERMEALS;
}
```

Ambas incluidas en el paquete medicalConsultation.

Las clases ProductSpecification, MedicalPrescription y MedicalPrescriptionLine

Las clases ProductSpecification, MedicalPrescription, y su componente MedicinePrescriptionLine, corresponden a las clases con ese mismo nombre del modelo del dominio de referencia (documento *DCl-eReceta-ConsultaMedica-SolRef.jpg*).

Implementar y realizar test para la clase ProductSpecification[8].

[8] Con sus respectivos atributos que, por simplicidad consideraremos tan sólo: UPCcode (un objecto ProductID), description (un String) y price (un BigDecimal), con sus respectivos getters y setters (getProductID(), getDescription() y getPrice()). Pensad qué argumentos recibirá el constructor.

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

```
package medicalconsultation;

/**
* Package for the classes involved in the use case Suply next dispensing
*/

public class MedicalPrescription {// A class that represents medical prescription
    private int prescCode;
    private Date prescDate;
    private Date endDate;
    private HealthCardID hcID;  // the healthcard ID of the patient
    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 IncorrectTakingGuidelinesException;
  public void modifyLine(ProductID prodID, String[] instruc) { . . . }
        throws ProductNotInPrescription, IncorrectTakingGuidelinesException;
  public void removeLine(ProductID prodID) { . . . }
                      throws ProductNotInPrescription;
  ???   // the getters and setters
}
```

- 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 contendrá los valores introducidos por el médico para cada uno de los ítems asociados a las pautas y posología de ese medicamento, respectivamente).
- modifyLine(ProductID prodID, String[] instruc): modifica 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 contendrá o la cadena vacía –se mantiene el valor actual de la línea de prescripción-, o bien la variación introducida por el médico para ese medicamento durante la revisión, para cada uno de los ítems de pauta y posología, respectivamente, a fin de modificar la línea de prescripción de ese medicamento).
  También se pide su implementación, aunque no forme parte del escenario tratado.
- removeLine(ProductID prodID): elimina una línea de prescripción médica. Recibe un solo argumento (prodID), el código del producto a eliminar del tratamiento.
  También se pide su implementación, aunque no forme parte del escenario tratado.
  La excepción ProductNotInPrescription se produce al intentar modificar o eliminar de la prescripción médica un producto no incluido en la misma. IncorrectTakingGuidelinesException cuando el formato de la pauta o la posología son incorrectos, o bien la información es incompleta.

Completad la implementación más adecuada para esta clase, teniendo en cuenta todos los requisitos, inclusive los que caen fuera del escenario tratado aquí. Para ser más precisos, debería facilitar la búsqueda de líneas de prescripción dentro de la instancia

MedicalPrescription, a fin de modificar y eliminar las líneas de prescripción médica de forma efectiva.

¿Qué ocurre con la clase MedicalPrescriptionLine? ¿Qué implementación has escogido para ella?

Implementar y realizar test para todas 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 ConsultationTerminal

A continuación, se presenta la estructura, aunque incompleta, de la clase ConsultationTerminal, la cual define los eventos de entrada para resolver este caso de uso de forma completa, bajo el escenario que nos ocupa.

```
public class ConsultationTerminal {
    ???

  public void initRevision() { . . . } throws HealthCardException,
                  NotValidePrescriptionException, ConnectException;

  public void initPrescriptionEdition() { . . . } throws
                  AnyCurrentPrescriptionException, NotFinishedTreatmentException;

  public void searchForProducts(String keyWord) { . . . } throws
                  AnyKeyWordMedicineException, ConnectException;

  public void selectProduct(int option) { . . . } throws
                  AnyMedicineSearchException, ConnectException;

  public void enterMedicineGuidelines(String[] instruc) { . . . } throws
    AnySelectedMedicineException, IncorrectTakingGuidelinesException;

  public void enterTreatmentEndingDate(Date date) { . . . } throws
                  IncorrectEndingDateException;

  public void sendePrescription() { . . . } throws ConnectException,
    NotValidePrescription, eSignatureException, NotCompletedMedicalPrescription;

  public void printePresc() { . . . } throws printingException;

  ???    // Other methods, apart from the input events
}
```

A continuación se presentan, a grandes rasgos, dichos métodos (*algunos de sobras conocidos*). Los contratos de referencia detallados se encuentran en el documento *ModeloCasosUsoeReceta-ParteContratos.pdf*.

- initRevision(): Se procede a obtener la información médica del paciente. Para ello, primero es necesario recuperar su identificación, que se encuentra registrada en el servicio externo ScheduledVisitAgenda. A continuación, mediante conexión con el servicio externo HealthNationalService, se procede a la descarga de la e-receta del paciente. Por simplicidad, tan sólo se maneja la prescripción médica (prescindimos de la colección de dispensaciones).

Excepciones: ConnectException, NotValidePrescriptionException y HealthCardException, para las situaciones reflejadas en los contratos de operación pasados como referencia.

- initPrescriptionEdition(ProductID pID): El médico indica el inicio del proceso de edición de la prescripción médica.
  Excepciones: NotFinishedTreatmentException para la situación en que el tractamiento aún no ha finalizado, y AnyCurrentPrescriptionException si no hay una prescripción en curso.
- searchForProducts(String keyWord): El médico pone en marcha el motor de búsqueda, vía conexión con el HealthNationalService, para consultar los medicamentos del catálogo de productos que contengan la palabra clave keyWord. Obtiene una lista de todos ellos.
  Excepciones: AnyKeyWordMedicineException, en el caso en que no se encuentre ningún medicamento en el catálogo con la palabra clave introducida, y ConnectException (*ya conocida*).
- selectProduct(int option): El médico selecciona un medicamento de entre todos los medicamentos de la lista de resultados obtenida previamente, a fin de obtener su especificación.
  Excepciones: AnyMedicineSearchException, en el caso en que no se haya realizado ninguna búsqueda en el catálogo de productos, y ConnectException.
- enterMedicineGuidelines(String[] instruc): El médico introduce todos los ítems de información correspondientes a las pautas y la posología del medicamento que está siendo incluido en la prescripción médica. El argumento instruc (un array de String) proporciona toda esa información, tal y como se ha detallado anteriormente en el método addLine(…) de MedicalPrescription. Se crea una línea de prescripción médica asociada a ese medicamento (la línea de prescripción médica queda asociada a la especificación del producto y a la nueva instancia de TakingGuideline).
  Excepciones: AnySelectedMedicineException, en el caso en que no se haya seleccionado un medicamento para su inclusión en la prescripción médica en curso. IncorrectTakingGuidelinesException cuando el formato de la pauta o la posología son incorrectos, o bien la información es incompleta.
- 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 sea incorrecta o situada en el pasado.
- sendePrescription(): El médico da paso al proceso de transmitir la e-receta al HealthNationalService para su registro remoto. Previo a la conexión con el HNS, el sistema estampa la firma electrónica del médico (un DigitalSignature). Si todo es correcto, el HNS genera un código de tratamiento. La e-receta queda almacenada remotamente, actualizada y vigente para el siguiente periodo de tratamiento, y la retorna de nuevo al sistema.
  Por simplicidad, tan sólo se maneja la prescripción médica (prescindimos de la colección de dispensaciones).
  Excepciones: NotCompletedMedicalPrescription, ConnectException, NotValidePrescription y eSignatureException para las situaciones reflejadas en los contratos de operación pasados como referencia.
- printePresc(): El médico lanza la orden de imprimir la hoja de tratamiento. *No se pide su implementación*.

Consideraciones:

- No hace falta contemplar la parte correspondiente a los servicios de impresión, a fin de imprimir la hoja de tratamiento. Es por ello que no se pide la implementación del método relacionado, ni de las excepciones relacionadas.
- Definiréis las excepciones como clases propias (subclases de Exception – excepciones *checables*) y, adicionalmente, la excepción proporcionada por la API: ConnectException[9], tal y como aparece en las cabeceras de los métodos.
  Por lo que respecta a las excepciones correspondientes a las clases del paquete data se dejan para vosotros.

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

4. Consideraciones generales

- La práctica se puede realizar **en grupos de tres personas** (preferiblemente los mismos grupos que hasta ahora).
- 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 coordinaros con vuestros compañeros (e. g. GitHub, Bitbucket 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 definiros otros constructores además de los sugeridos aquí, simplificando la inicialización de las clases.
  - Es recomendable ir trabajando cada miembro del grupo en ramas distintas para así lograr una mejor colaboración y sincronización de vuestro trabajo (e.g. desarrollo de distintos requisitos/funcionalidades y/o unidades funcionales por separado).
- Como entregaréis un ZIP con el directorio del proyecto, entregaréis también el repositorio git (subdirectorio .git), por lo que podré comprobar los commits (*no os lo dejéis para el último día !*).
- Por lo que respecta al SUT (System Under Test), los eventos de entrada **deben satisfacer los contratos** de referencia facilitados en detalle (documento *ModeloCasosUsoeReceta-ParteContratos.pdf*), a excepción de las simplificaciones introducidas en este documento.
- 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 pasar los test.
- Los test dobles los podéis definir internamente en las clases de test, o bien en un paquete separado.

[9] 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. ¿Qué se ha de entregar?

Un **ZIP** que contenga:

- El **proyecto desarrollado** (podéis utilizar IntelliJ IDEA o cualquier otro entorno).
- Un pequeño **informe** en el que expliquéis con vuestras palabras las situaciones que habéis probado en cada uno de los test realizados, así como el/los criterio/s empleado/s para tomar vuestra/s decisión/es de diseño (principios SOLID, patrones GRASP, etc.), y justificación de los mismos, si es el caso (*la implementación se os da ya masticada*).
  Podéis añadir tambe cualquier otro 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 grupo**, indicando el nombre de vuestros compañeros.