Programación 4

EXAMEN DICIEMBRE 2016

Por favor siga las siguientes indicaciones:

- Escriba con lápiz y de un solo lado de las hojas.
- Escriba su nombre y número de documento en todas las hojas que entregue.
- Numere las hojas e indique el total de hojas en la primera de ellas.
- Recuerde entregar su número de examen junto al examen.
- Está prohibido el uso de computadoras, tabletas o teléfonos durante el parcial.

Problema 1 (30 puntos)

Parte a)

i) ¿Cuáles son las dos actividades vistas en el curso dentro de la etapa de Análisis?

ii) La especificación del comportamiento de una operación en un Contrato de Software, ¿es declarativa o imperativa? Justifique muy brevemente.

Parte b)

> *Mindfulness o "conciencia plena" consiste en prestar atención, momento a momento, a pensamientos, emociones, sensaciones corporales y al ambiente circundante, de una forma principalmente caracterizada por la "aceptación", es decir, una atención a pensamientos y emociones sin juzgar si son correctos o no. El cerebro se enfoca en lo que es percibido a cada momento, en lugar de proceder con la normal "rumiación" acerca del pasado o el futuro.* [Wikipedia]

Esta definición es una útil introducción a esta nueva práctica llamada mindfulness, la cual está tomando mucha fuerza en Occidente durante los últimos años, y el Uruguay no es la excepción.

En este contexto, asuma que se le ha solicitado a Ud. una aplicación móvil (app) que ayude a los usuarios en la práctica de mindfulness, mediante la realización de sesiones y el registro de éstas en la app.

Existen tres tipos de sesiones: Meditación, Pausa y Respiración. Todas las sesiones cuentan con un nombre que las identifica dentro de la app, así como la duración (en minutos) y un texto descriptivo. La única diferencia entre ellas es que las meditaciones y las respiraciones contienen una guía auditiva, mientras que las pausas no.

La app debe permitir al usuario realizar cualquier sesión en cualquier momento del día y la cantidad de veces que el usuario desee. Cada vez, se registrará la sesión que el usuario realizó, la fecha y hora y el tiempo que le llevó (pues podría cortar la sesión antes de su finalización o bien permanecer mas tiempo de lo sugerido por la sesión).

Asimismo, la app también debe permitir ingresar recordatorios de forma de ayudar al usuario a realizar sesiones a lo largo del día. Cada recordatorio únicamente le recordará al usuario realizar una sola sesión, a una hora determinada y mostrando un mensaje (ej: "¡Hola María Paz! Son las 8am, y debes hacer tu meditación matinal.")

Como se mencionó, el usuario es libre de realizar sesiones por fuera de los recordatorios, así como también es libre de no hacer caso a los recordatorios (lo cual no interesa registrar en la app).

Por último, y dado que el modelo de negocio de la app es del tipo Freemium, la app tendrá templates que podrán ser comprados y descargados (in-app purchases). Los

templates constan de una combinación de sesiones y recordatorios, de forma tal que al descargar un template, el usuario ya cuente con las sesiones y recordatorios de sesiones de ese template. Cada template lleva además un nombre (que lo identifica), un precio y un nivel de dificultad ("Principiante", "Medio" y "Avanzado"). A modo de ejemplo, el usuario puede descargar el template llamado "Mindfulness Running" por USD 0,99 para principiantes, el cual consta de una serie de sesiones especialmente confeccionadas para mejorar la performance de los corredores, así como sus recordatorios asociados.

Caso de Uso:	Crear recordatorio
Actor:	Usuario
Descripción:	Este caso de uso comienza cuando el actor ingresa el nombre (identificatorio), la hora y el mensaje del nuevo recordatorio. El Sistema devolverá todas las sesiones existentes y el actor deberá elegir aquella para la cual desea crear el recordatorio. En caso que el actor no desee elegir una sesión, podrá entonces cancelar el caso de uso.

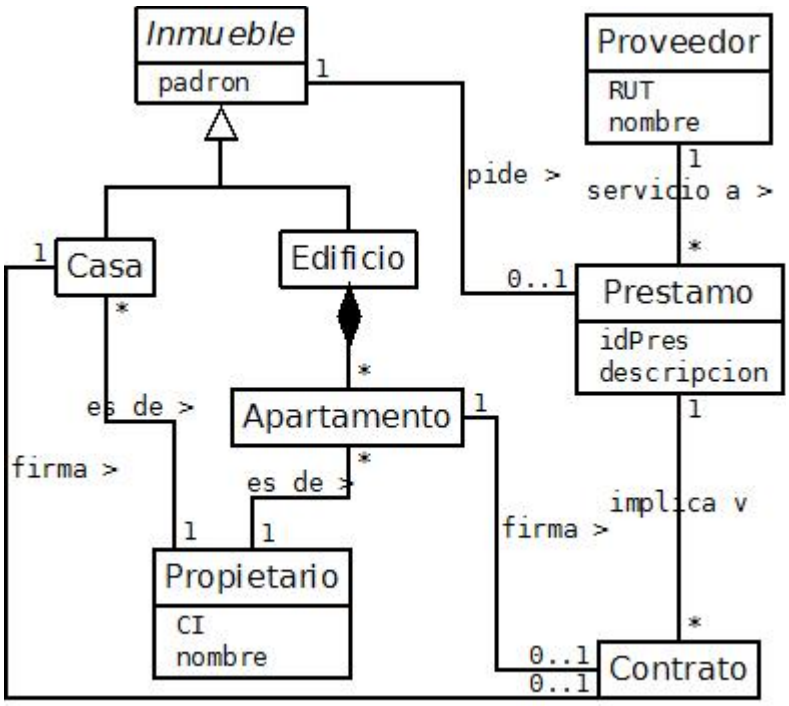
Se pide:

i) Modelo de Dominio de la realidad anterior (considerando el Caso de Uso) con restricciones en lenguaje natural.

ii) Diagrama de Secuencia del Sistema (DSS) del Caso de Uso, incluyendo el uso de Datatypes y de manejo de memoria del Sistema, en caso de ser necesario.

Problema 2 (35 puntos)

La municipalidad de la ciudad "Perla del Plata" organiza un sistema de préstamos para reparación de inmuebles patrimoniales. Estos préstamos están dirigidos a propietarios de casas o edificios, para realizar reparaciones de fachadas, estructura o áreas comunes, por ejemplo. Un préstamo está asociado a un inmueble y a un proveedor, que es la empresa que realiza la reparación. Además tiene asociado un contrato que se firma con el propietario del inmueble. En caso de ser un edificio, se firma un contrato con cada uno de los propietarios de los apartamentos del edificio (si un propietario tiene varios apartamentos en un edificio, debe firmar un contrato por cada uno). El modelo de dominio de la figura muestra el análisis de la estructura de la realidad.



Restricciones:

- Los Inmuebles están identificados por su padron.
- Los Proveedores están identificados por su RUT.
- Los Prestamos están identificados por su idPres generado por el Sistema.
- No puede existir un Contrato de un Prestamo, firmado por un Inmueble que no está asociado al Prestamo.

Además se consideran las siguientes operaciones del sistema:

altaPrestamo(padronInm, RUTProv: **Integer**, descrip: **String)**	
Descripción	Se ingresa un nuevo préstamo al sitema
Parámetros	- Existe en el Sistema una instancia de Inmueble con padron - padronInm: padrón del inmueble. - RUTProv: RUT del proveedor. - descrip: descripción de la reparación.
Precondiciones	- Existe en el Sistema una instancia de Inmueble con padron igual a padronInm. - Existe en el Sistema una instancia de Proveedor con RUT igual a RUTProv. - No existe en el Sistema una instancia de Inmueble con padron

	igual a padronInm asociada a alguna instancia de Prestamo.
Postcondiciones	- Existe en el Sistema una nueva instancia de Prestamo con idPres igual a un número autogenerado por el Sistema y descripcion igual a descrip. - Se crea un link entre la instancia de Prestamo recién creada y el Proveedor con RUT igual a RUTProv. - Se crea un link entre la instancia de Prestamo recién creada y el Inmueble con padron igual a padronInm. - Si el Inmueble con padron igual a padronInm es una Casa, se crea una instancia de Contrato, la cual se asocia a dicha Casa y al Prestamo. - Si el inmueble con padron igual a padronInm es un Edificio, se crea una instancia de Contrato para cada Apartamento del Edificio, así como los correspondientes links entre Contrato y Apartamento, y entre Contrato y el Prestamo.

darPropietarios (RUTProv: **Integer**) : **Set (DataPropietario)**	
Descripción	Devuelve los datos completos de todos los Propietarios que tienen Prestamos asociados a un Proveedor.
Parámetros	- RUTProv: RUT del Proveedor.
Precondiciones	- Existe en el Sistema una instancia de Proveedor con RUT igual a RUTProv.
Postcondiciones	- Se devuelve una colección de datavalues de DataPropietario para cada instancia de Propietario que tiene un Inmueble con un Prestamo asociado al Proveedor con RUT igual a RUTProv.

Se pide:

a) Realizar el Diagrama de Comunicación de la operación altaPrestamo. Asignar la responsabilidad de creación de las instancias de Contrato al tipo de Inmueble concreto.

b) Realizar el Diagrama de Comunicación de la operación darPropietarios.

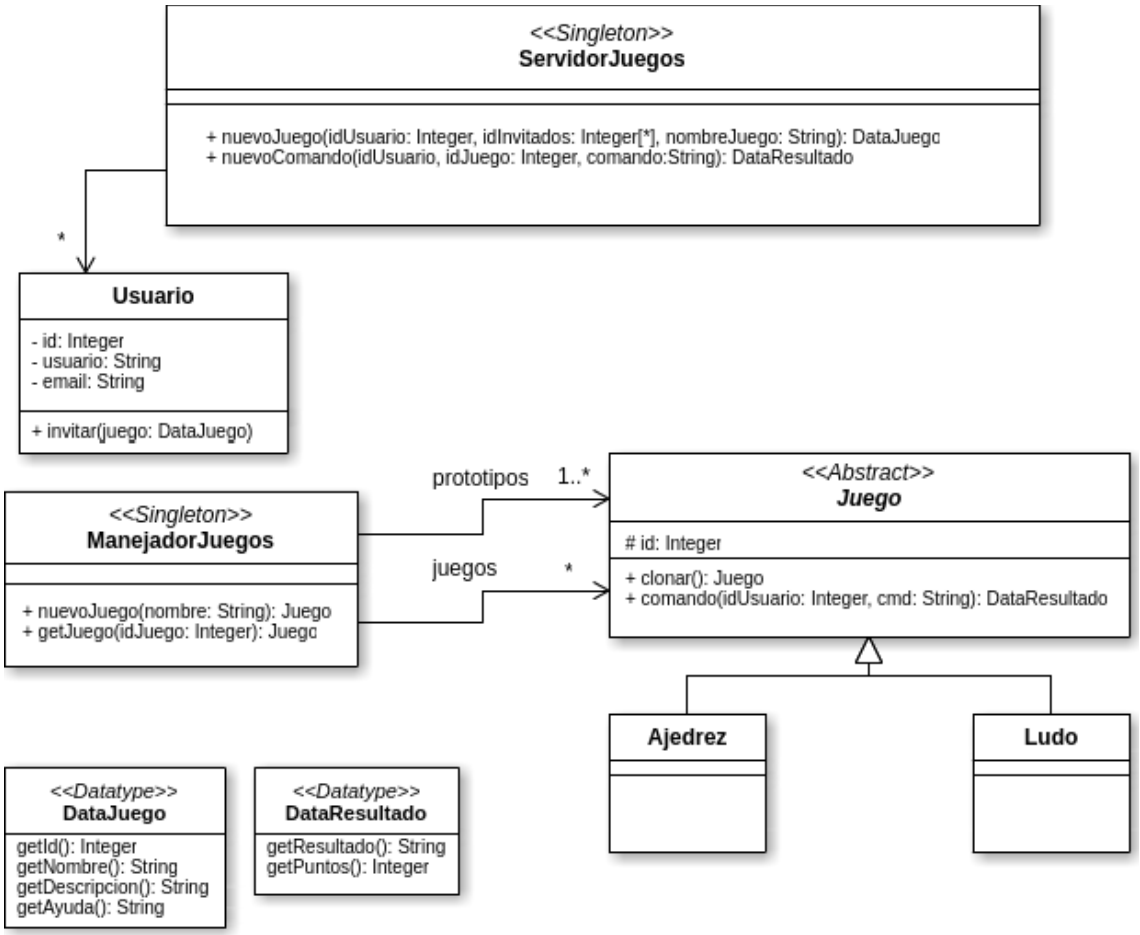
c) Realizar el Diagrama de Clases de Diseño Resultante.

Problema 3 (35 puntos)

En la implementación del patrón *Factory* es necesario modificar tanto el *Consumidor* como la *Fábrica* si se desea agregar un *Proveedor* nuevo, para ciertos lenguajes de programación (entre ellos C++). El objetivo de este ejercicio es introducir al patrón de diseño *Prototype*, que mitiga esta dificultad, logrando en este sentido un menor grado de acoplamiento. Los objetos a crear se clonan de prototipos, que son instancias existentes. Todos ellos deben implementar una interfaz que especifica la forma en que deben clonarse los mismos.

Se pretende implementar un servidor de juegos en línea, donde un usuario registrado puede invitar a uno o más a jugar una partida de los juegos existentes.

Se diseñó para tal fin un prototipo del sistema, el cual debe implementar. La siguiente figura muestra un diagrama de clases de diseño:



Nota: No se incluyen dependencias

El controlador *ServidorJuegos* cuenta con las siguientes operaciones:

- *nuevoJuego(idUsuario: Integer, idInvitados: Integer[*], nombreJuego: String): DataJuego* Genera una nueva instancia del *Juego* de nombre *nombreJuego*, y retorna sus datos en un datatype *DataJuego*. El usuario con ID *idUsuario* puede invitar a uno o más usuarios registrados en el sistema, cuyos IDs se deben incluir en el arreglo *idInvitados*.
- *nuevoComando(idUsuario, idJuego: Integer, comando: String): DataResultado* Mediante esta operación del sistema el usuario identificado por el ID *idUsuario* envía un comando al Juego de ID *idJuego*.

Mediante la operación *Usuario::invitar* se envía una invitación a un determinado *Juego* a un *Usuario*.

Para la creación de juegos se decidió utilizar el patrón *Prototype*: el *singleton ManejadorJuegos* tendrá una colección de instancias de juegos que utilizará como prototipos para crear otros nuevos. Cuando se invoque su operación *nuevoJuego*, deberá devolver una instancia nueva del *Juego* del nombre dado por parámetro, y a su vez agregarla a la colección de juegos en curso. Cada *Juego* será clonado invocando la operación *clonar* del prototipo pertinente.

Considerar:

1. Puede suponer la existencia de la interface *ICollectible* e implementaciones de *ICollection* (clase *List*), *Ikey* (clase *OrderedKey*), *IDictionary* (clase *OrderedDictionary*) e *Iiterator*, según sea necesario.
2. Es posible utilizar las clases *set<T>*, *vector<T>* o *map<K,T>* de la STL.
3. Las implementaciones deben incluir constructores y destructores.
4. Implementar los getters solamente para datatypes.
5. No implementar setters y getters, salvo la operación getJuego.
6. No incluir directivas al precompilador.

Se pide:

1. Implementar .h y .cpp del data type *DataJuego*.
2. Implementar .h y .cpp de la clase *ServidorJuegos*.
3. Implementar .h y .cpp de la clase *Usuario*.
4. Implementar .h y .cpp de la clase *ManejadorJuegos*.
5. Implementar .h y .cpp de las clases *Juego* y *Ajedrez*.
6. Las operaciones invitar y comando deben tener método vacío.