Programación 4

EXAMEN JULIO 2022 - SOLUCIÓN

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 parcial junto al parcial
- Está prohibido el uso de computadoras, tabletas o teléfonos durante el parcial

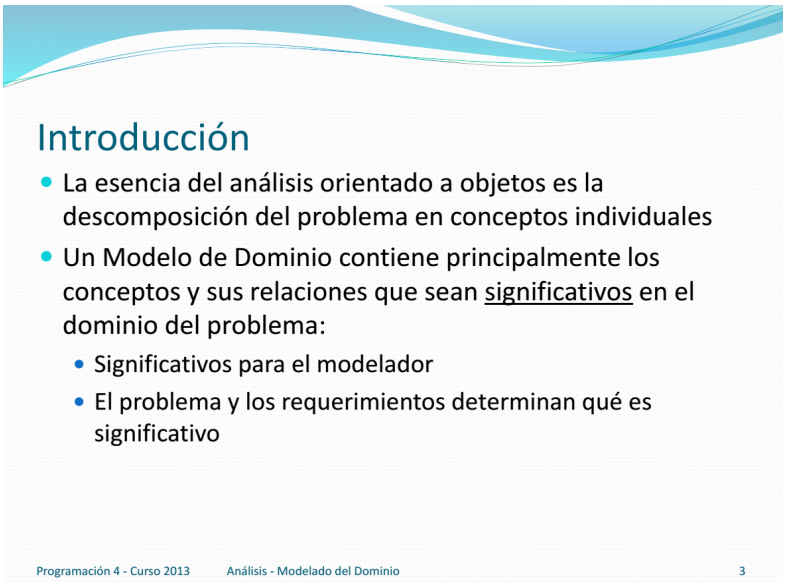
Problema 1 (30 puntos)

Se desea modelar un sistema para gestionar la toma de exámenes.

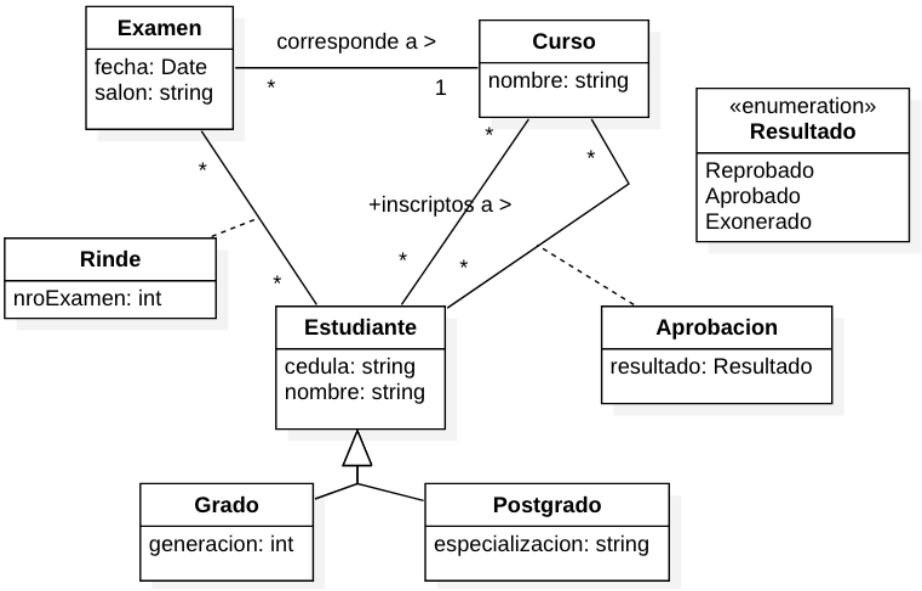
Se pide:

a) ¿Cuál es el propósito de realizar modelos de dominio?

La idea es que reflexionen sobre para qué sirve el modelado del dominio. Podrían basarse en la siguiente transparencia del curso, o decir directamente que sirve para comprender mejor el problema y el contexto en el cual se quiere crear o entender un sistema/software.



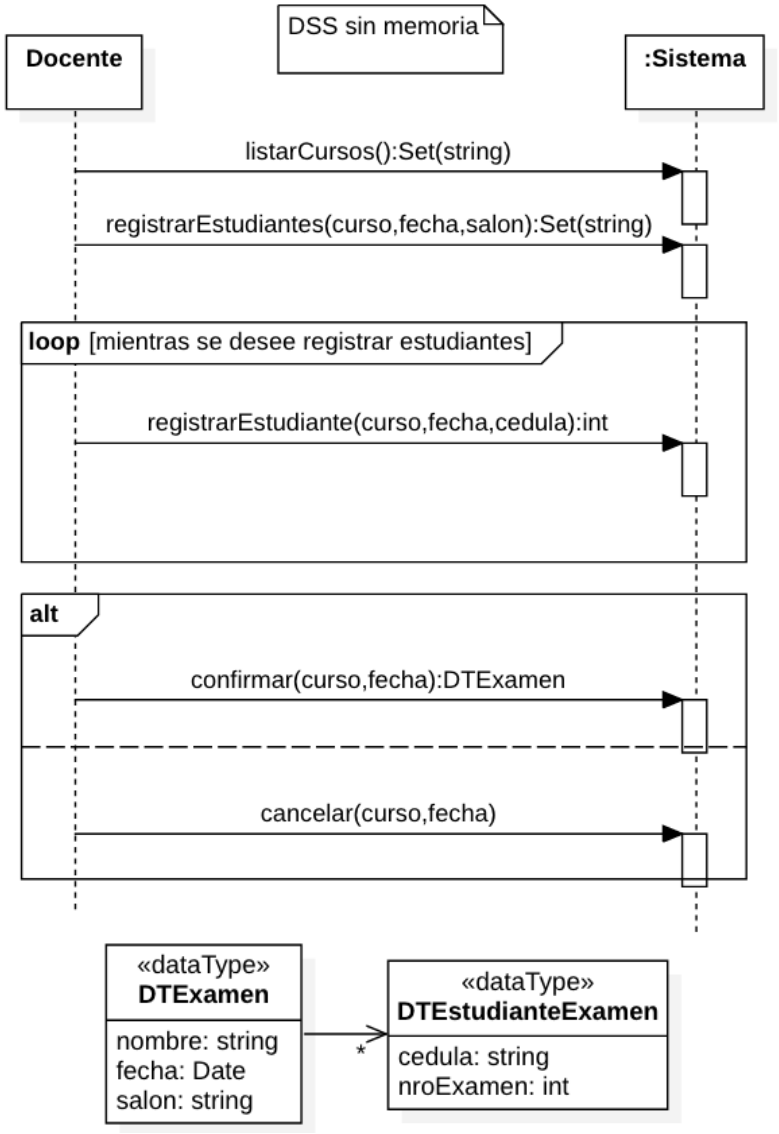
b) Realizar el Modelo de Dominio de la realidad planteada, incluyendo solamente restricciones de integridad circular en lenguaje natural.



Restricciones:

- Una instancia de Estudiante tiene un link Aprobación con una instancia de Curso, solo si también tiene un link “inscripto a” con la misma instancia de Curso.
- Una instancia de Estudiante tiene un link Rinde con una instancia de Examen solamente si esa instancia de Estudiante tiene un link Aprobación con la instancia Curso asociada a la instancia de Examen y además el atributo Resultado del objeto Aprobación es igual a “Aprobado”.

c) Realizar un Diagrama de Secuencia del Sistema (DSS) para el Caso de Uso “Registrar asistentes a un examen” indicando el uso de memoria del Sistema y de datatypes, si corresponde.



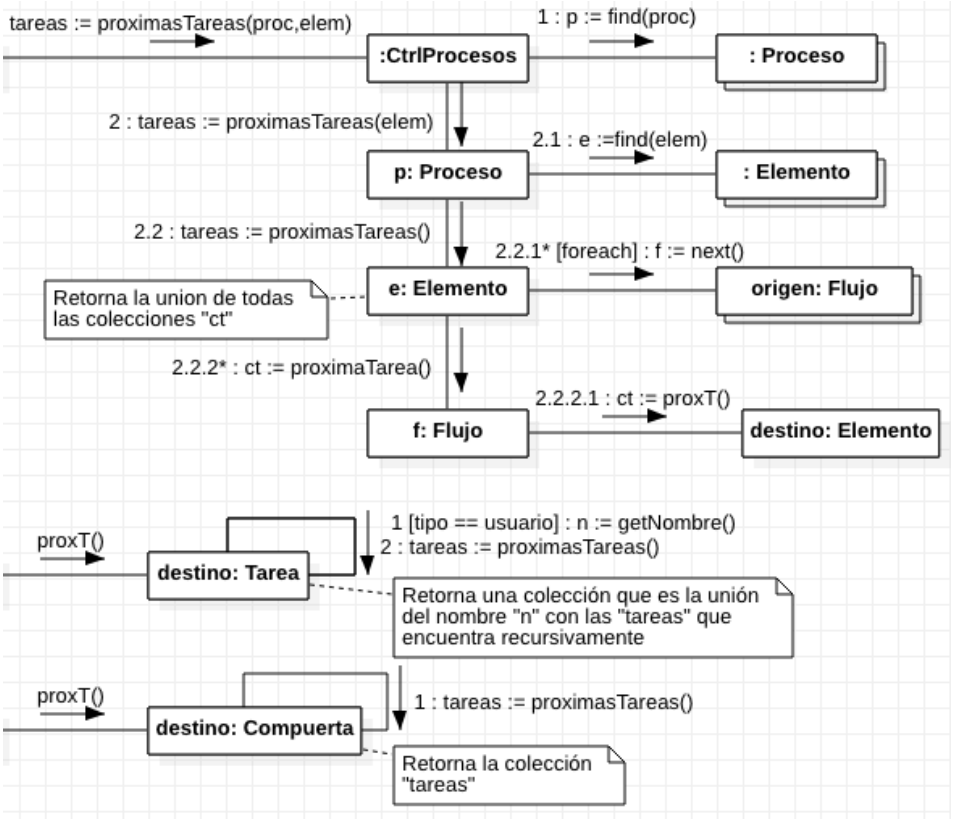
Nota: el actor aparece representado como un objeto, pero debería utilizarse el ícono correspondiente.

Problema 2 (35 puntos)

Un motor de procesos permite automatizar un conjunto de pasos que se realizan en un orden determinado para lograr un objetivo.

Se pide:

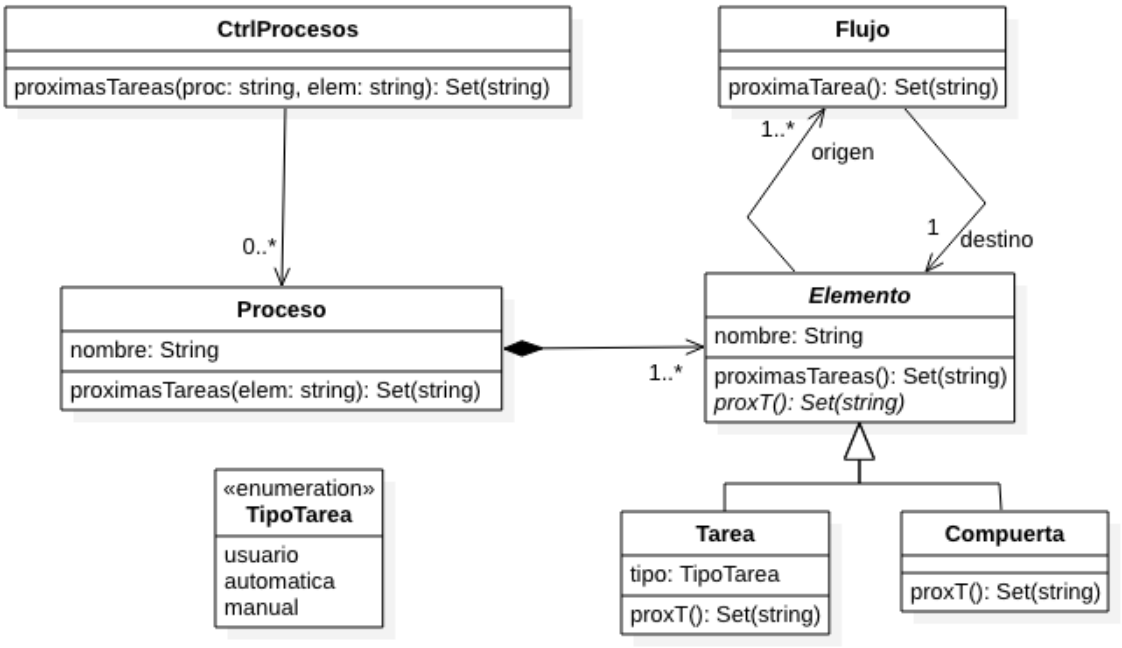
a) Realizar el Diagrama de Comunicación correspondiente a la operación especificada.



La operación puede dar lugar a dos interpretaciones, ambas válidas: la búsqueda para cuando se encuentran las primeras tareas de usuario a través de todos los caminos de un elemento, o se devuelven todas las tareas de usuario a partir de un punto hasta que no haya más en el proceso. Ambas opciones son consideradas válidas. En la solución anterior, se toma la segunda opción.

Una vez que se encuentra el elemento dentro del proceso, para encontrar las próximas tareas de usuario, es necesario ver las secuencias que salen de ese elemento. Por eso hay que analizar todas las secuencias de las cuales el elemento es origen (relación desde) y a partir de allí buscar el próximo elemento. Dado que el elemento puede ser una compuerta, podría haber varias secuencias de salida desde el elemento (mensaje 2.2.1). Se usa polimorfismo para buscar el nombre del elemento. Si el siguiente elemento es una tarea de tipo usuario, entonces se retorna su nombre. En cualquier caso, hay que llamar recursivamente para analizar todas las secuencias de salida de ese elemento (varias si es una compuerta).

b) Realizar el Diagrama de Clases de Diseño (DCD) resultante del diagrama anterior.



Problema 3 (35 puntos)

Se desea implementar un sistema que gestione archivos distribuidos dentro de una red.

Se pide:

a. Implementar en C++ los .h de las clases `ICtrlArchivos` y `CtrlArchivos`.

```
class ICtrlArchivos {
 virtual set<DtArchivo> listarArchivos(string nombreDir) = 0;
 virtual void crearArchivoLocal(string nombreDir, DTArchivo data) = 0;
 virtual ~ICtrlArchivos() = 0;
};

class CtrlArchivos: public ICtrlArchivos {
private:
      static CtrlArchivos* instance;
      map<string, Directorio*> directorios;
      CtrlArchivos();
public:
      set<DtArchivo> listarArchivos(string nombreDir);
      void crearArchivoLocal(string nombreDir, DTArchivo data);
      static CtrlArchivos* getInstance();
      ~CtrlArchivos();
};
```

b. Implementar en C++ los .h del enumerado `TipoPermiso` y de las clases `Archivo` y `Local`.

```
Class Archivo{
      private :
            String nombre;
            Int tamanio;
      public:
            DtArchivo getDatos();
            virtual bool esRemoto() = 0;
}

class Local: public Archivo {
      public:
            Local(string nombre, int tamanio);
            bool esRemoto();
            ~Local();
};
```

c. Implementar en C++ el .cpp de la clase `CtrlArchivos`. No incluya destructor. Incluya código de manejo de excepciones en la operación `crearArchivoLocal()` de la clase `CtrlArchivos` para el caso en que no exista el directorio en donde se crea el archivo.

```
CtrlArchivos * CtrlArchivos::instance = NULL;

CtrlArchivos::CtrlArchivos() {
      this->directorios = {};
      this->usuarios = {};
}
```

```
CtrlArchivos * CtrlArchivos::getInstance(){
      if (instance == NULL)
            instance = new CtrlArchivos();
      return instance;
}

set <DTArchivo> CtrlArchivos::listarArchivos() {

      set <DTArchivo> dts;
      map<string, Directorio*>::iterator it;

      for (it = directorios.begin(); it != directorios.end(); it++) {
            Directorio* d = it->second;
            set <DTArchivo> dtaux = d->listarArchivos();
            dts.insert(dtaux.begin(), dtaux.end());
      }
      return dts;
}


Void CtrlArchivos::crearArchivoLocal(string nombreDir,DTArchivo data){

      if (directorios.find(nombreDir) == directorios.end())
            throw invalid_argument("El directorio no existe");

      Directorio* d = directorios[nombreDir];
      d->crearArchivoLocal(DTArchivo data);
      SyncServer *s = SyncServer::getInstance();
      s->sincronizarl(string nombreDir, DTArchivo data);
}
```