Programación 4

PARCIAL FINAL 2021

Problema 1 (25 puntos)

Se está construyendo un sistema de software para la gestión de la operativa de una empresa de transporte de personas, que tiene una flota de vehículos que se pueden contratar para realizar viajes particulares.

Se cuenta con la siguiente descripción de la realidad:

Para contratar un viaje se debe tener un usuario registrado en el sistema, identificado por su número de documento de identidad. En un viaje pueden participar varios usuarios, incluso si no están registrados. Estos últimos, también se ingresan en el sistema y existen mientras no finalizó el viaje. Todos los usuarios deben tener un seudónimo que es único en el sistema. Cuando un usuario registrado contrata un viaje, debe especificar, además de la fecha, una lista ordenada de paradas donde el vehículo se debe detener a levantar un usuario (en cada parada sube solamente un usuario). Para cada parada, se conoce su ubicación, el usuario que asciende, a qué hora se debe pasar y cuánto tiempo se espera como máximo a que se presente el usuario. Todos los usuarios descienden en la última parada del viaje. Una vez finalizado el viaje, los usuarios registrados que participaron del mismo pueden calificarlo en un rango de uno a cinco. Los viajes se pueden contratar de tipo normal o especial. Los vehículos de la empresa se identifican por su matrícula y además se conoce su marca y modelo. Pueden ser de tipo estándar o premium. Para estos últimos se conoce una lista de servicios extra, por ejemplo, internet a bordo, frigobar o calefacción en los asientos. Solamente los vehículos premium pueden hacer viajes especiales.

Además, se cuenta con la descripción del siguiente Caso de Uso:

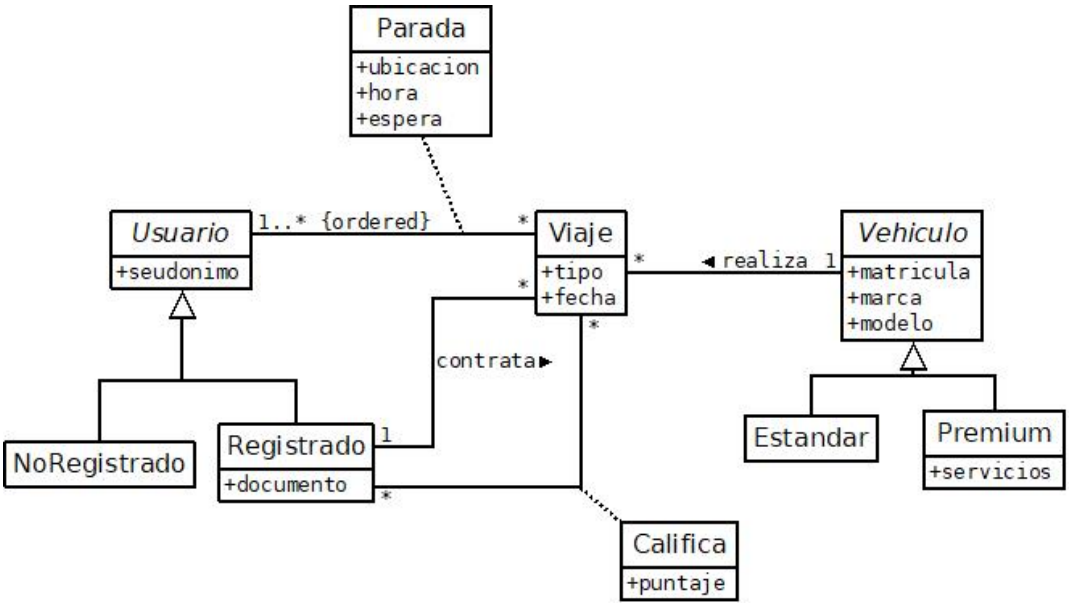
Caso de Uso	Contratar un viaje
Actor	Usuario
Descripción	El caso de uso comienza cuando un usuario registrado que tiene sesión iniciada desea contratar un viaje. Para eso indica el tipo de viaje, la cantidad de usuarios, la fecha y las horas estimadas de inicio y de fin. Si no hay un vehículo disponible para realizar un viaje con esas características, el sistema lo indica y se cancela el caso de uso. En caso contrario, el usuario ingresa una a una las paradas, indicando para cada una, su ubicación, la hora a estar en la parada y el tiempo máximo a esperar. Además, para cada parada se indica el usuario; si es registrado (incluido quien contrata el viaje), se debe seleccionar de una lista proporcionada previamente por el sistema, si no, se ingresa el seudónimo del usuario no registrado. Luego, el sistema muestra los datos completos del viaje que se va a contratar (incluyendo datos del viaje, usuarios y paradas). Finalmente, se puede cancelar o confirmar, en cuyo caso el sistema devuelve un código correspondiente al viaje contratado.

Se pide:

a) Realizar en UML el Modelo de Dominio de la realidad planteada**.**
b) Expresar en lenguaje natural y de la forma más precisa posible, todas las restricciones que apliquen al modelo de la parte anterior, clasificándolas según su tipo.
c) Realizar un Diagrama de Secuencia del Sistema (DSS) para el Caso de Uso "Contratar un viaje", indicando el uso de memoria del Sistema y de DataTypes si corresponde.

Solución

a)



b)

Unicidad de atributos:

- No existen dos instancias diferentes de `Registrado` con el mismo valor en su atributo `documento`.
- No existen dos instancias diferentes de `Vehiculo` con el mismo valor en su atributo `matricula`.

Dominio de atributos:

- El atributo `puntaje` de la clase `Califica` está en el rango `1..5`.
- El atributo `servicios` de la clase `Premium` es un conjunto de nombres.
- El atributo `tipo` de la clase `Viaje` toma valores en el conjunto `{normal, especial}`.

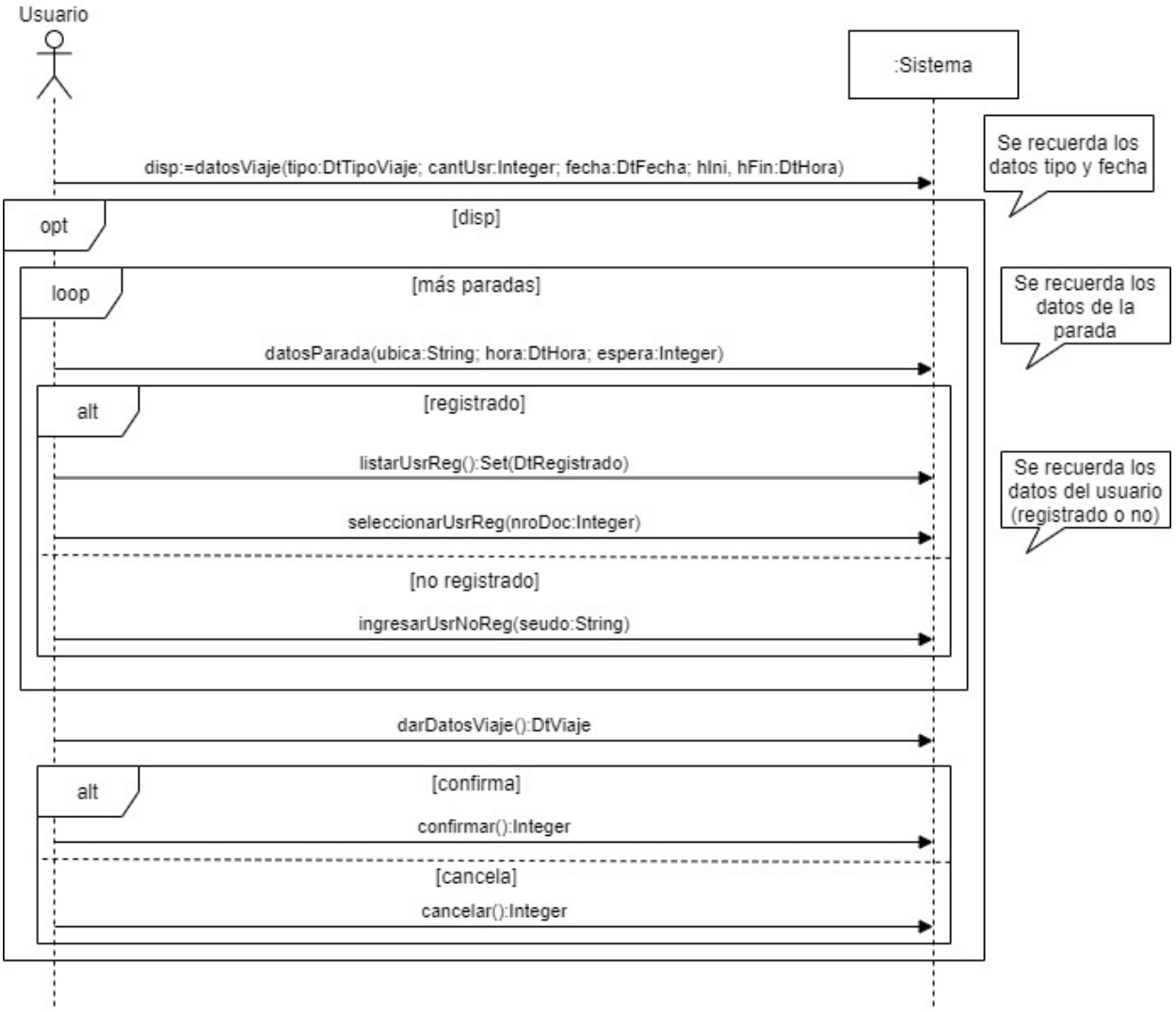
Integridad circular:

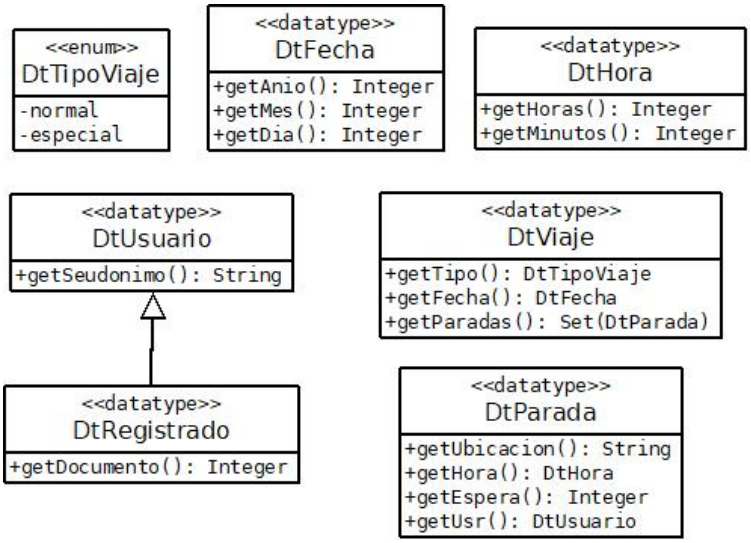
- Si una instancia de `Registrado` está asociada mediante `contrata` a un `Viaje v`, entonces también está asociada a `v` mediante `Parada`.
- Si una instancia de `Registrado` está asociada mediante `Califica` a un `Viaje v`, entonces también está asociada a `v` mediante `Parada`.

Reglas de negocio:

- Si una instancia de `Viaje` tiene el valor `especial` en su atributo `viaje`, debe estar asociada a una instancia de `Premium`.
- Todas las instancias de `Usuario` asociadas a una misma instancia de `Viaje`, tienen valores diferentes en su atributo `seudonimo`.

c)


Usuario
:Sistema
Se recuerda los datos tipo y fecha
disp:=datosViaje(tipo:DtTipoViaje; cantUsr:Integer; fecha:DtFecha; hIni, hFin:DtHora)
opt
[disp]
loop
[más paradas]
Se recuerda los datos de la parada
datosParada(ubica:String; hora:DtHora; espera:Integer)
alt
[registrado]
listarUsrReg():Set(DtRegistrado)
Se recuerda los datos del usuario (registrado o no)
seleccionarUsrReg(nroDoc:Integer)
[no registrado]
ingresarUsrNoReg(seudo:String)
darDatosViaje():DtViaje
alt
[confirma]
confirmar():Integer
[cancela]
cancelar():Integer

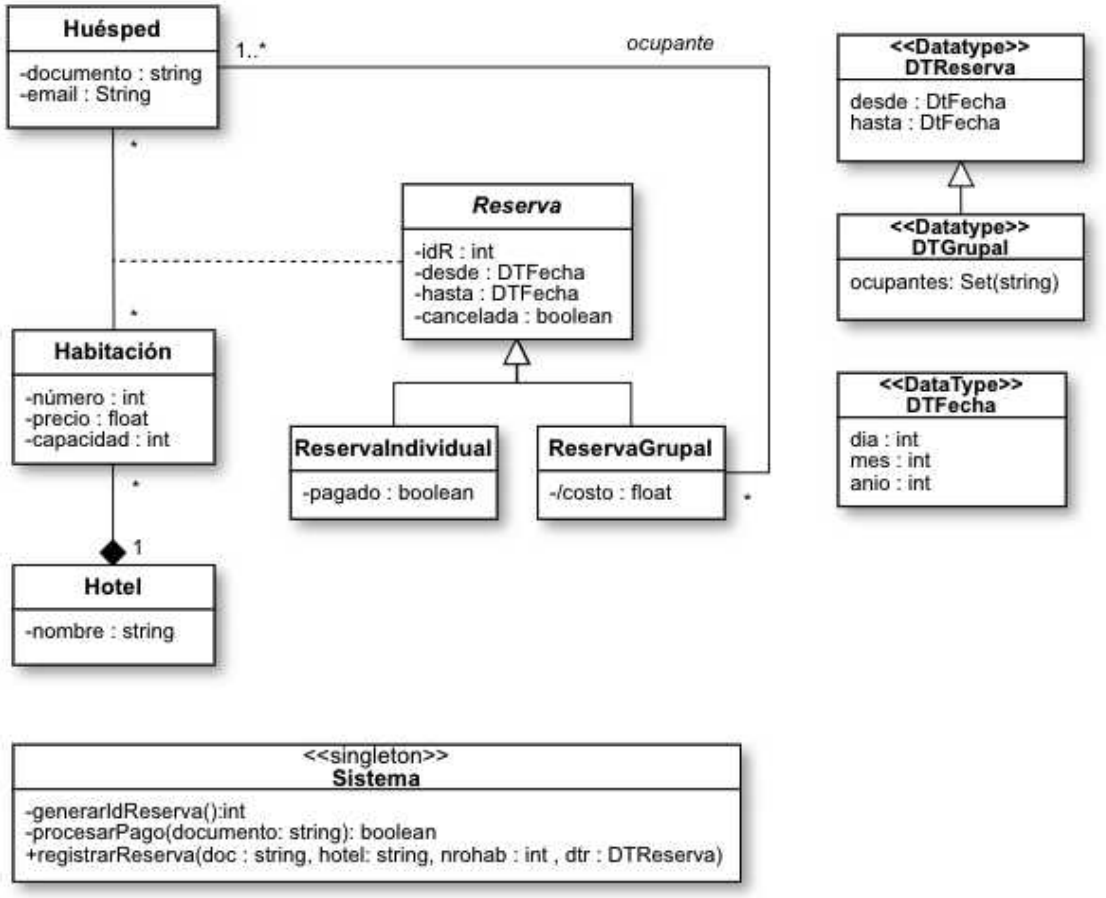

<<enum>>
DtTipoViaje
-normal
-especial
<<datatype>>
DtFecha
+getAnio(): Integer
+getMes(): Integer
+getDia(): Integer
<<datatype>>
DtHora
+getHoras(): Integer
+getMinutos(): Integer
<<datatype>>
DtUsuario
+getSeudonimo(): String
<<datatype>>
DtRegistrado
+getDocumento(): Integer
<<datatype>>
DtViaje
+getTipo(): DtTipoViaje
+getFecha(): DtFecha
+getParadas(): Set(DtParada)
<<datatype>>
DtParada
+getUbicacion(): String
+getHora(): DtHora
+getEspera(): Integer
+getUsr(): DtUsuario

Programación 4

PARCIAL FINAL 2021

Problema 2 (25 puntos)

Considere el siguiente modelo de dominio y diseño parcial para un sistema de reservas de habitaciones en una cadena de hoteles.



Aclaraciones:

- El atributo documento identifica al Huésped.
- El atributo nombre identifica al Hotel.
- El atributo idR identifica a la Reserva.
- La fecha hasta debe ser mayor que desde.
- En una reserva grupal:
  - El huésped que realiza la reserva no es parte de los otros ocupantes de la estadía.
  - Los ocupantes de una reserva grupal NO están asociados a la habitación mediante la asociación Reserva.
  - El atributo costo es el producto de la cantidad de huéspedes por el precio de la habitación.

Se cuenta con el contrato de la siguiente operación del sistema:

`registrarReserva(doc: String, hotel: String, nrohab: Integer, dtr: DTReserva)`	
Descripción:	Se registra una nueva reserva en el sistema.
Parámetros:	`doc`: Identificador del huésped que realiza la reserva. `hotel`: Identificador del hotel en el cual se desea realizar la reserva. `nrohab`: Identificador de la habitación a reservar. `dtr`: Contiene los datos de la reserva.
Precondiciones:	Existe en el sistema un `Huésped` con documento `doc`. Existe en el sistema un `Hotel` con nombre `hotel`. Existe una `Habitación` con número `nrohab` en el `Hotel` con nombre `hotel`. Si `dtr` es de tipo `DTGrupal`, existen instancias de `Huésped` identificadas por los documentos contenidos en el set ocupantes de `dtr`. No existe una instancia de `Reserva` relacionada con la instancia de `Habitación nrohab`, con el atributo `cancelada` en `false` y que el rango de fecha determinado por sus atributos `desde` y `hasta` se superponga con las fechas solicitadas en `dtr`. La `Habitación nrohab` tiene capacidad suficiente para la cantidad de huéspedes de la reserva `dtr`.
Poscondiciones:	Se crea una instancia de `ReservaIndividual` o `ReservaGrupal`, dependiendo del tipo de reserva especificado en `dtr`, con un id generado por la operación `generarIdReserva` y los datos indicados en `dtr`. Se crean links entre la nueva reserva y las instancias de `Huésped documento` y `Habitación nrohab`. En caso de ser una instancia de `ReservaGrupal` se calcula el `costo` y se le asocian las instancias de `Huésped` correspondientes a los `ocupantes`. En caso de ser una instancia de `ReservaIndividual` se procesa el pago invocando la operación `procesarPago` y se setea el atributo `pagado` en base al resultado de dicha operación.

Se pide:

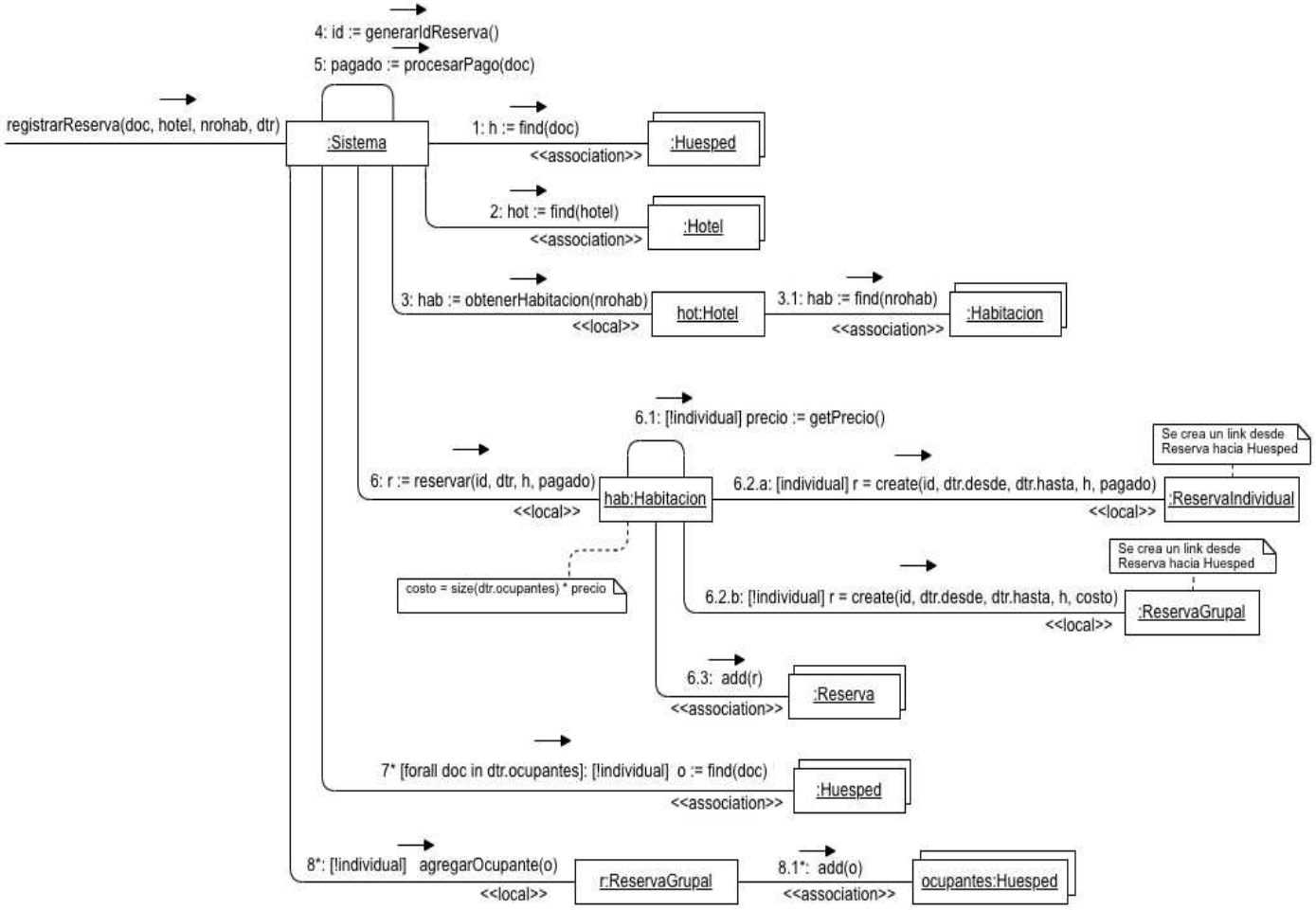
a) Realizar el Diagrama de Comunicación correspondiente a la operación `registrarReserva`, **incluyendo visibilidades**.

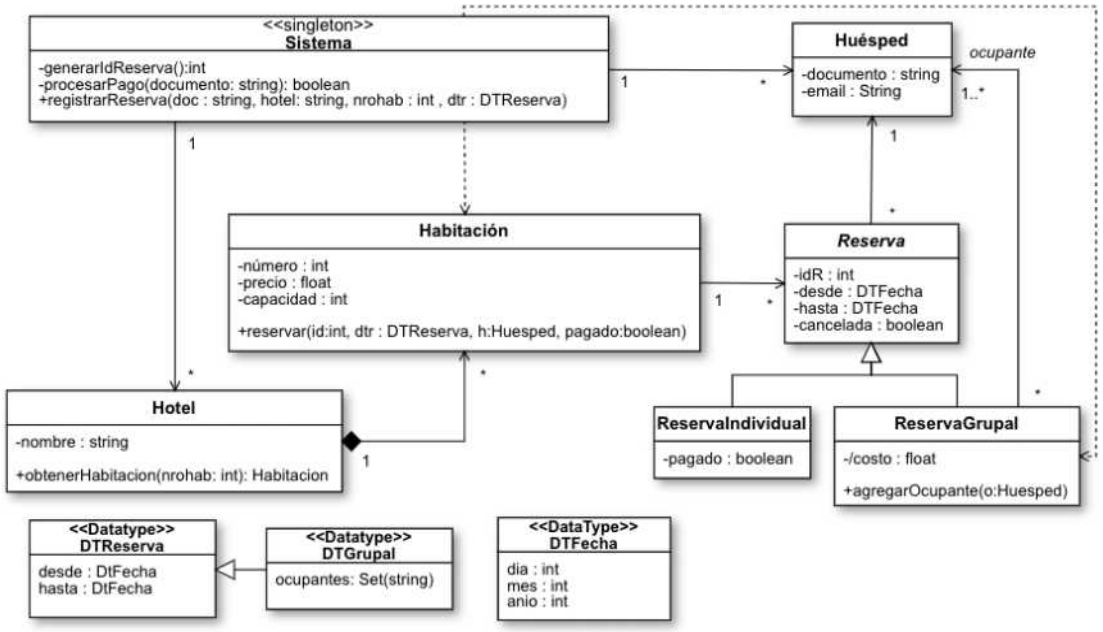
b) Realizar el Diagrama de Clases de Diseño (DCD) resultante.

Nota:

- No incorporar setters ni getters al DCD.
- Incorporar en el DCD sólo los constructores o destructores que se utilicen en los diagramas de comunicación.

Solución:


4: id := generarIdReserva()
5: pagado := procesarPago(doc)
registrarReserva(doc, hotel, nrohab, dtr)
:Sistema
1: h := find(doc)
<<association>>
:Huesped
2: hot := find(hotel)
<<association>>
:Hotel
3: hab := obtenerHabitacion(nrohab)
<<local>>
hot:Hotel
3.1: hab := find(nrohab)
<<association>>
:Habitacion
6.1: [!individual] precio := getPrecio()
6: r := reservar(id, dtr, h, pagado)
<<local>>
hab:Habitacion
6.2.a: [individual] r = create(id, dtr.desde, dtr.hasta, h, pagado)
<<local>>
:ReservaIndividual
Se crea un link desde Reserva hacia Huesped
costo = size(dtr.ocupantes) * precio
6.2.b: [!individual] r = create(id, dtr.desde, dtr.hasta, h, costo)
<<local>>
:ReservaGrupal
Se crea un link desde Reserva hacia Huesped
6.3: add(r)
<<association>>
:Reserva
7* [forall doc in dtr.ocupantes]: [!individual] o := find(doc)
<<association>>
:Huesped
8*: [!individual] agregarOcupante(o)
<<local>>
r:ReservaGrupal
8.1*: add(o)
<<association>>
ocupantes:Huesped


<<singleton>>
Sistema
-generarIdReserva():int
-procesarPago(documento: string): boolean
+registrarReserva(doc : string, hotel: string, nrohab : int , dtr : DTReserva)
Huésped
-documento : string
-email : String
ocupante
Habitación
-número : int
-precio : float
-capacidad : int
+reservar(id:int, dtr : DTReserva, h:Huesped, pagado:boolean)
Reserva
-idR : int
-desde : DTFecha
-hasta : DTFecha
-cancelada : boolean
Hotel
-nombre : string
+obtenerHabitacion(nrohab: int): Habitacion
ReservaIndividual
-pagado : boolean
ReservaGrupal
-/costo : float
+agregarOcupante(o:Huesped)
<<Datatype>>
DTReserva
desde : DtFecha
hasta : DtFecha
<<Datatype>>
DTGrupal
ocupantes: Set(string)
<<DataType>>
DTFecha
dia : int
mes : int
anio : int

Programación 4

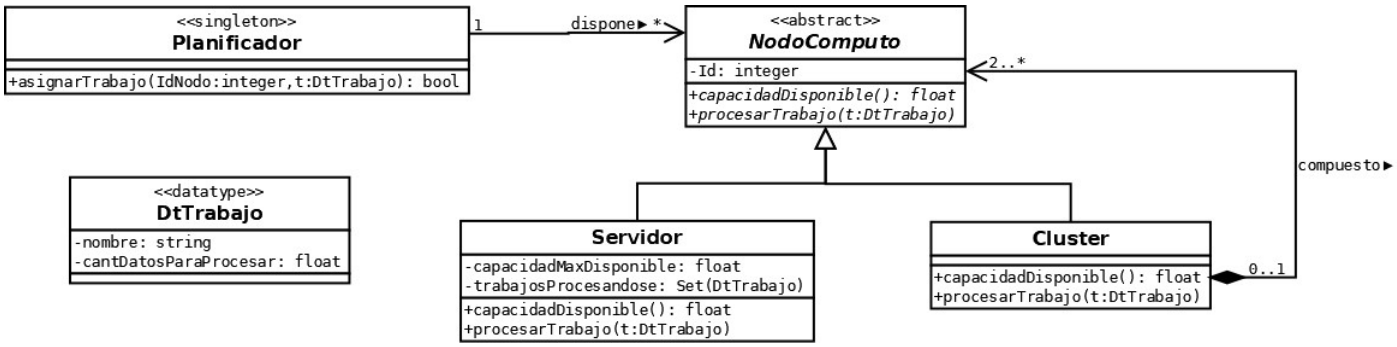
PARCIAL FINAL 2021

Problema 3 (25 puntos)

La figura muestra el Diagrama de Clases de Diseño (DCD) parcial de un sistema que permite distribuir el procesamiento de datos sobre un conjunto de nodos de cómputo interconectados, los cuáles se componen de servidores o clústeres (grupos) de otros nodos de cómputo.

El planificador del sistema se encarga de recibir los trabajos y asignarlos a nodos de cómputo con capacidad suficiente para poder procesarlos. Cada trabajo se identifica por un nombre y contiene una cantidad de datos para ser procesada.

Los servidores son los nodos finales en donde se realiza el procesamiento de los datos, y cada uno puede procesar más de un trabajo a la vez mientras no se sobrepase su capacidad máxima de procesamiento. Un trabajo no puede fraccionarse, es decir, es procesado en su totalidad en el mismo servidor. Por otro lado, los clústeres solamente delegan el procesamiento de los trabajos que reciben entre los nodos que lo componen.



A continuación, se describe el comportamiento de las operaciones principales.

Planificador::asignarTrabajo(idNodo: integer, t:DtTrabajo): bool
Pre-condición: existe un nodo de cómputo con identificador igual a *idNodo.*
Asigna el procesamiento del trabajo *t* al nodo de cómputo *idNodo* si su capacidad disponible es mayor o igual a la cantidad de datos de dicho trabajo. Retorna *true* si dicha asignación es exitosa.

NodoComputo::capacidadDisponible():float
Retorna la capacidad disponible del nodo de cómputo. Si el nodo es un servidor, dicho valor es igual su capacidad máxima menos la suma total de datos de los trabajos que está procesando. Por otra parte, si se trata de un clúster, dicho valor es igual a la máxima capacidad disponible entre los nodos que lo componen.

NodoComputo::procesarTrabajo(t:DtTrabajo)
Pre-condición: la capacidad disponible del nodo es mayor o igual a la cantidad de datos del trabajo *t.*
Procesa los datos del trabajo *t*. Si el nodo es un servidor, dicho trabajo es procesado por el propio nodo agregándolo a la colección de trabajos que están siendo procesados por él. Por otra parte, si se trata de un clúster, su procesamiento es delegado a cualquier nodo entre los que lo componen que tenga capacidad suficiente para hacerlo.

Se pide:

a) Implementar en C++ los .h de las clases *Planificador*, *NodoComputo* y *Cluster*.
b) Implementar en C++ el .cpp de la clase *Planificador*.
c) Implementar en C++ los métodos de las operaciones *capacidadDisponible()* y *procesarTrabajo()* para ambas clases *Servidor* y *Cluster*.

Observaciones:
- Puede utilizar colecciones genéricas (realizaciones de IDictionary e ICollection) o paramétricas (contenedores STL).
- Implementar exclusivamente las operaciones del DCD pedidas. Incluir la definición e implementación de constructores y destructores cuando sea necesario.
- No incluir directivas al precompilador (#include, etc).

Solución

```
// Planificador.h
class Planificador {
private:
    static Planificador* instancia;
    map<int, NodoComputo*> nodos;
    Planificador();
public:
    bool asignarTrabajo(int IdNodo, DtTrabajo t);
    static Planificador* darInstancia();
    ~Planificador();
};

// NodoComputo.h
class NodoComputo {
private:
    int Id;
public:
    NodoComputo(int Id);
    virtual float capacidadDisponible() = 0;
    virtual void procesarTrabajo(DtTrabajo t) = 0;
    virtual ~NodoComputo();
};

// Cluster.h
class Cluster: public NodoComputo {
private:
    set<NodoComputo*> nodos;
public:
    Cluster(int Id, set<NodoComputo*>& nodos);
    float capacidadDisponible();
    void procesarTrabajo(DtTrabajo t);
    virtual ~Cluster();
};
```

```cpp
// Planificador.cpp
Planificador* Planificador::instancia = nullptr;

Planificador::Planificador() {}

Planificador* Planificador::darInstancia() {
    if (instancia == nullptr) {
        instancia = new Planificador();
    }
    return instancia;
}

bool Planificador::asignarTrabajo(int IdNodo, DtTrabajo t) {
    NodoComputo* nodo = nodos[IdNodo];
    bool exito = nodo->capacidadDisponible() >= t.cantDatosParaProcesar;
    if (exito) {
        nodo->procesarTrabajo(t);
    }
    return exito;
}

Planificador::~Planificador() {
    map<int, NodoComputo*>::iterator it;
    for (it = nodos.begin(); it != nodos.end(); ++it) {
        NodoComputo* nodo = it->second;
        delete nodo;
    }
}

// Cluster.cpp
float Cluster::capacidadDisponible() {
    float capacidadDisp = 0;
    set<NodoComputo*>::iterator it;
    for (it = nodos.begin(); it != nodos.end(); ++it) {
        NodoComputo* nodo = *it;
        capacidadDisp = max(capacidadDisp, nodo->capacidadDisponible());
    }
    return capacidadDisp;
}

void Cluster::procesarTrabajo(DtTrabajo t) {
    set<NodoComputo*>::iterator it;
    for (it = nodos.begin(); it != nodos.end(); ++it) {
        NodoComputo* nodo = *it;
        if (nodo->capacidadDisponible() >= t.cantDatosParaProcesar) {
            nodo->procesarTrabajo(t);
            break;
        }
    }
```

```
}

// Servidor.cpp
float Servidor::capacidadDisponible() {
    float capacidadDisp = capacidadMaximaDisponible;
    set<DtTrabajo>::iterator it;
    for (it = trabajosProcesandose.begin(); it != trabajosProcesandose.en
d(); ++it) {
        DtTrabajo t = *it;
        capacidadDisp -= t.cantDatosParaProcesar;
    }
    return capacidad;
}

void Servidor::procesarTrabajo(DtTrabajo t) {
    trabajosProcesandose.insert(t);
}
```