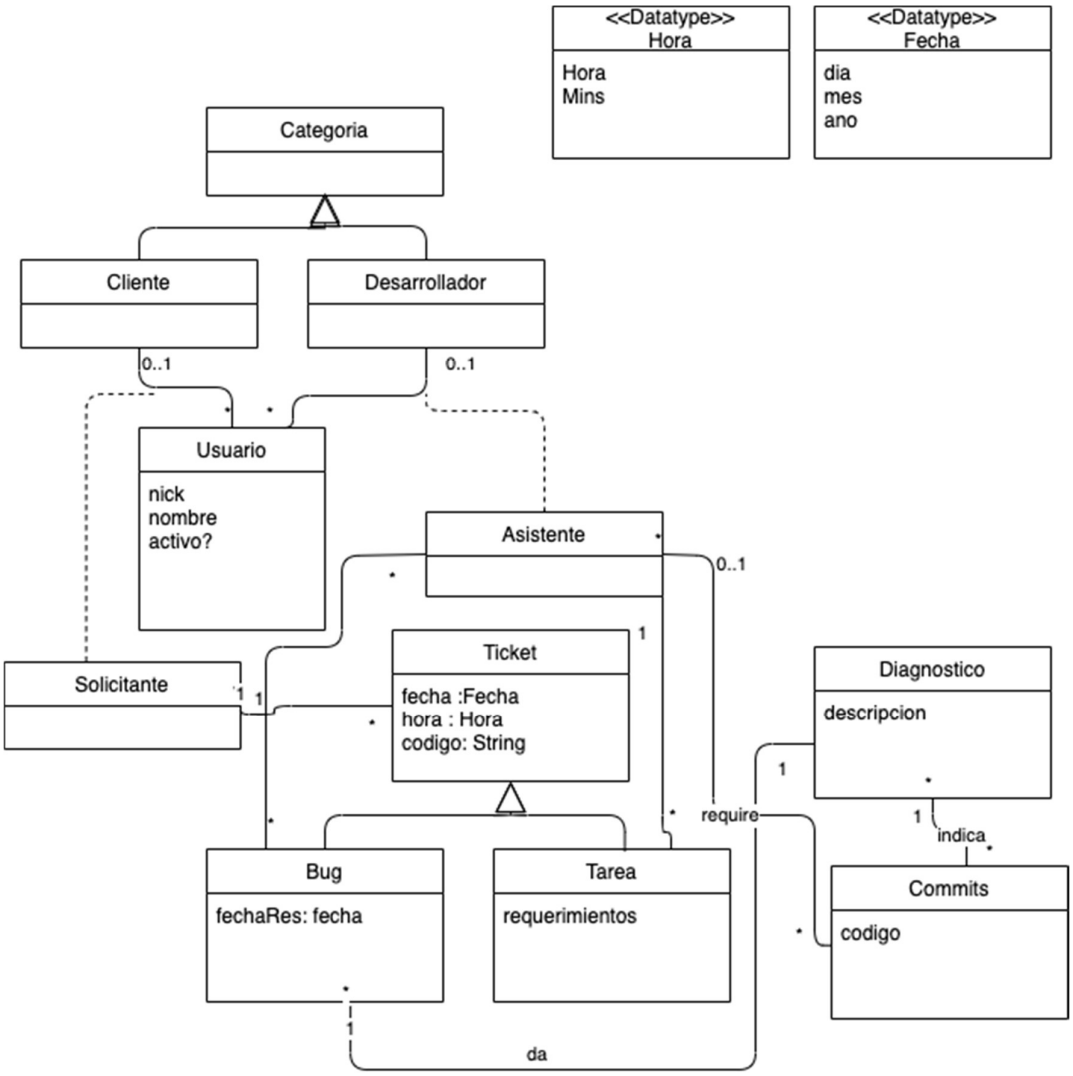
Programación 4

SOLUCIÓN EXAMEN JULIO 2021

Problema 1 (30 puntos)

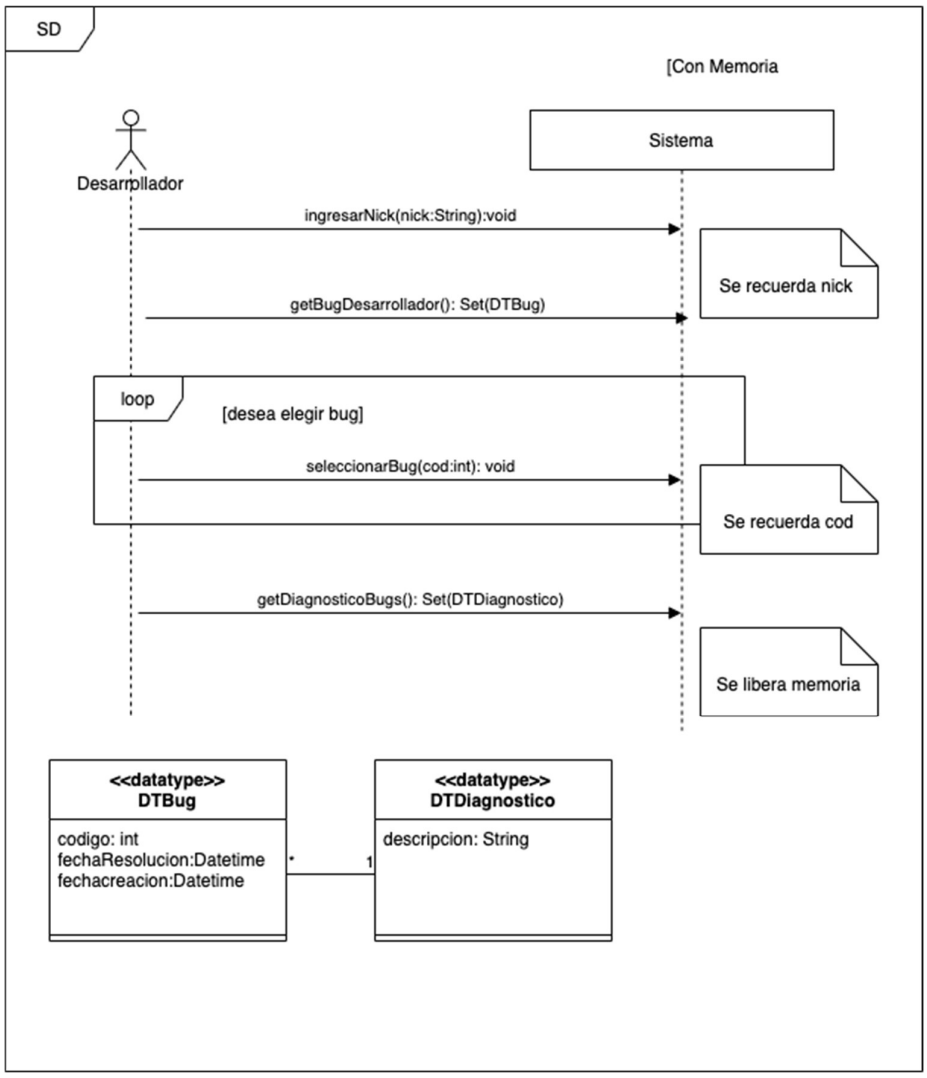
a)



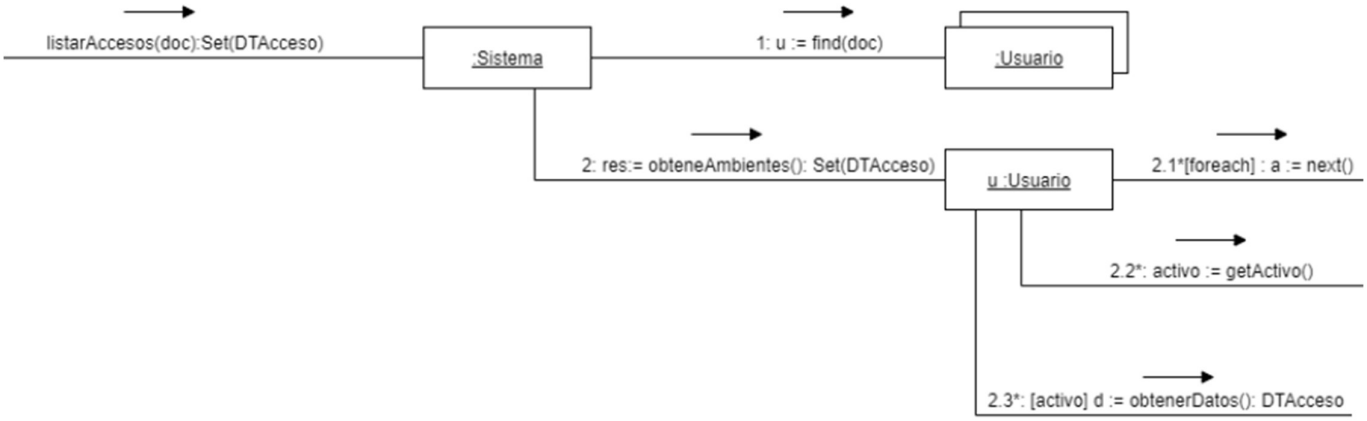
Restricciones:

Nick de Usuario debe ser único
Código de Ticket debe ser único
La fecha de resolución de un Bug debe ser posterior a la fecha del Ticket.
Asistente-Commits solo existe si hay un Bug asignado al Asistente.
Diagnostico-Commits está ordenado cronológicamente ascendente.

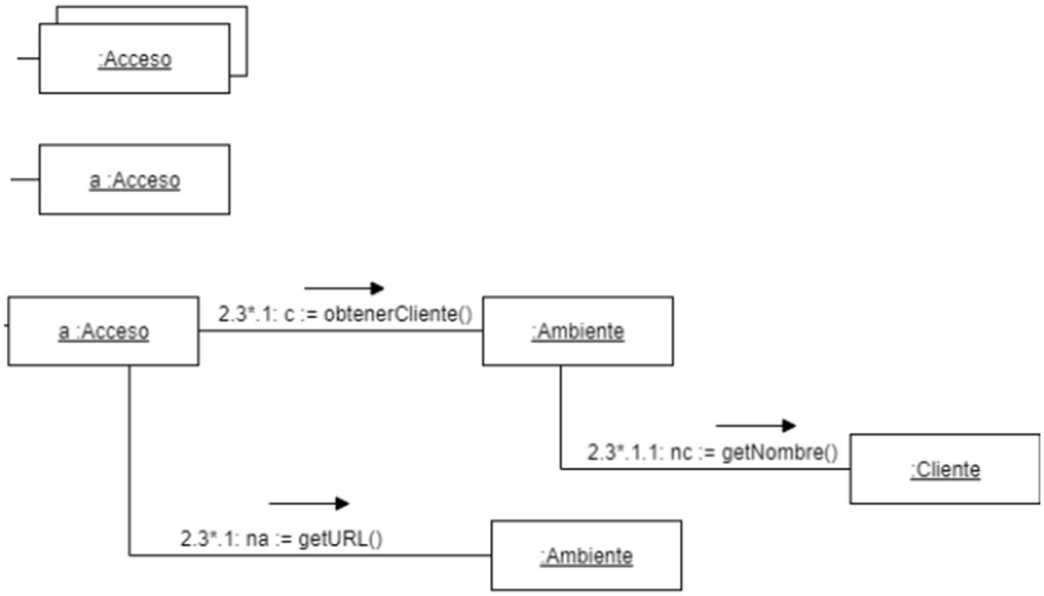
b)


SD
[Con Memoria
Sistema
Desarrollador
ingresarNick(nick:String):void
Se recuerda nick
getBugDesarrollador(): Set(DTBug)
loop
[desea elegir bug]
seleccionarBug(cod:int): void
Se recuerda cod
getDiagnosticoBugs(): Set(DTDiagnostico)
Se libera memoria
<<datatype>>
DTBug
codigo: int
fechaResolucion:Datetime
fechacreacion:Datetime
*
1
<<datatype>>
DTDiagnostico
descripcion: String

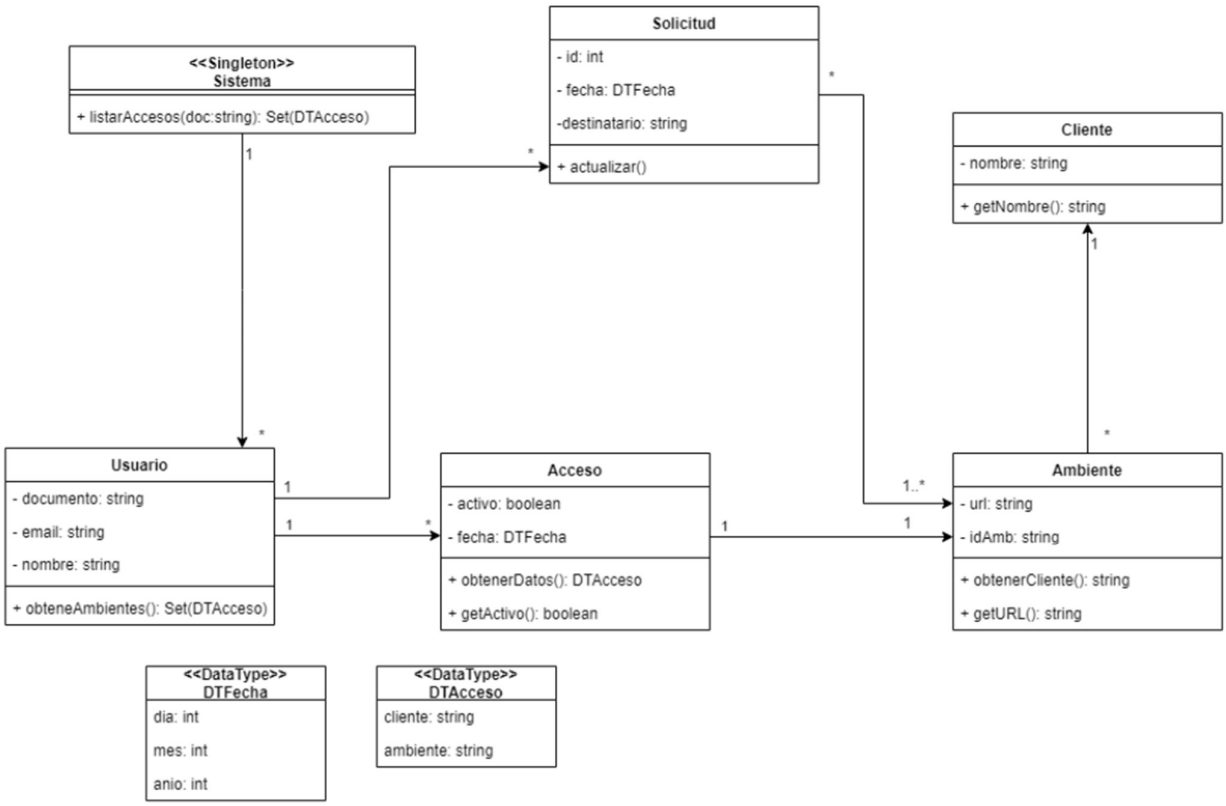
Problema 2 (35 puntos)



a)



b)



c)

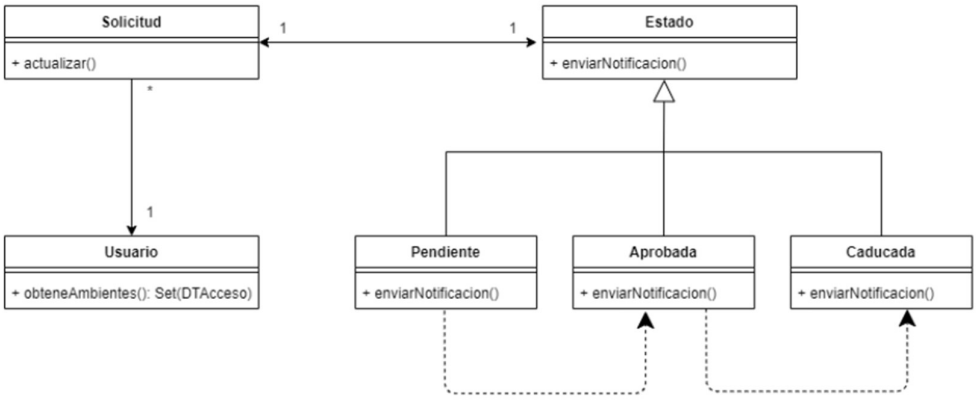
Se aplica el patrón State, con los siguientes roles:

- Contexto: Solicitud
- Estado: Estado
- Estados concretos: Pendiente, Aprobada, Caducada

```
Pendiente::enviarNotificacion() {
  enviarMail(solicitud.getDestinatario(),
             "Tiene una solicitud pendiente");
}

Aprobada::enviarNotificacion() {
  enviarMail(solicitud.getUsuario().getEmail(),
             "Su solicitud fue aprobada");
}

Cancelada::enviarNotificacion() {
}
```



<u>Problema 3</u> (35 puntos)

a)

En las respuestas de esta parte se esperaba un razonamiento a partir de la definición del problema tipo resuelto por Singleton y no que únicamente se lo enunciara. Las respuestas que están a continuación eran las esperadas, pero dependiendo del contexto y la justificación otras opciones puede ser correctas.

GestorOperaciones: No tiene por qué ser ser Singleton. Como el gestor no tiene atributos, no guarda memoria ni posee colecciones, las ejecuciones de las operaciones son independientes de la instancia que las ejecute. Por tanto, no hay ningún motivo para requerir que exista una única instancia en todo el sistema.
GestorColecciones: Necesariamente debe ser Singleton puesto que mantiene colecciones de objetos. Si hubiese más de una instancia de estas colecciones podría darse la situación de que un objeto esté en la colección de una de las instancias, pero no en la de la otra, dejando el sistema inconsistente. Además, en el diseño de la operación nuevaSolicitud(…) se accede con visibilidad global.

<u>Nota</u>: Solo se requería uno de los dos argumentos para GestorColecciones.

b)

En las respuestas de esta parte se espera no solo que la implementación dada sea correcta, sino que siga el diagrama dado. La única operación de la clase Cliente en dicho diagrama es agregarSolicitud(…) y era la única que debía implementarse.

```
void cliente::agregarSolicitud(int idP, TipoPago tp){
  GestorColecciones* gc = GestorColecciones::getInstance(); //global
  Proyecto *p = gc→getProy(idP);                            //2.1
  Solicita *s = new Solicita(this, p, tp);                  //2.2
  ultimo = p;
  solicitudes.insert(s);                                    //2.3
}
```

c)

GestorOperaciones.h

```
class GestorOperaciones{
  public:
    void nuevaSolicitud(string, int, tipoPago);
    int calcularPagoPersona(int, int);
    static GestorOperaciones* getInstance();
  private:
    GestorOperaciones();
    static GestorOperaciones* instancia
    // Lo escrito en rojo era necesario si se había especificado como
    // Singleton
}
```

Persona.h

```
class Persona.h{
  public:
    Persona(int);
    virtual calcularPago(int) = 0;
  private:
    int id;
}
```

Trabajador.h

```
class Trabajador: Public Persona{
  public:
    Trabajador(int, int);
    virtual calcularPago(int);
  private:
    int sueldo;
}
```

Trabajador.cpp

```
int Trabajador::calcularPago(int plazo){
  return sueldo*plazo;
}
```

Cliente.cpp

```
int Trabajador::calcularPago(int plazo){
  int ret = 0;
  set<Solicita>::iterator iter;
  for(iter = solicitudes.begin(); iter < solicitudes.end, iter++){
    ret += (*it)→getMontoMensual*plazo;
  }
  return ret;
}
```