Sección 1 – Requerimientos funcionales, asignación de responsabilidades y documentación de contratos (Realizar en hojas blancas)

Etapa 1: Documentación de contratos

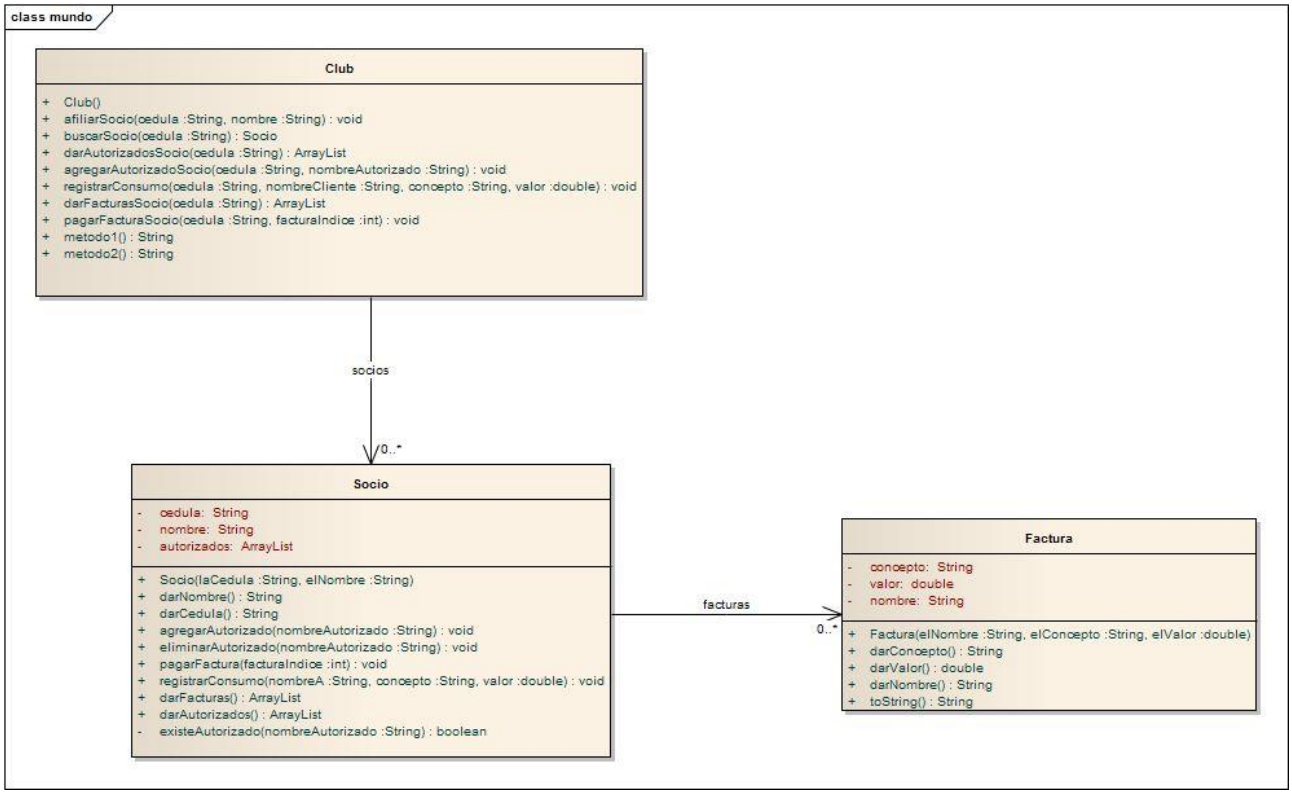
Escenario:

Considere el siguiente método dentro de la clase Club, perteneciente al ejemplo del club de cupi2, para contestar las siguientes preguntas:

```
public void afiliarSocio( String cedula, String nombre ) throws Exception
{

    Socio s = buscarSocio( cedula );
    if( s == null )
    {
        Socio nuevoSocio = new Socio( cedula, nombre );
        socios.add( nuevoSocio );
    }
    else
    {
        throw new Exception( "El socio ya existe" );
    }
}
```

El diagrama UML del club social es el siguiente:



1. ¿Para qué sirve este método? Dé una breve descripción

2. ¿Qué condiciones son necesarias para que el método pueda funcionar de forma correcta? Tenga en cuenta la relación con los atributos de esa clase, especialmente en lo que respecta a inicialización de arreglos.

3. ¿Qué se espera que ocurra dentro de los datos del programa después de la ejecución de este método?

4. ¿Qué parámetros necesita el método para funcionar? Descríbalos y diga las condiciones en las que deben llegar estos parámetros

5. ¿El método tiene excepciones? ¿En qué casos debe lanzarse estas excepciones?

6. Complete la documentación del con la información que proveyó en los puntos anteriores.

```
/**
*                   <br>
* <b>pre: </b>      <br>
* <b>post: </b>     <br>
* @param
* @param
* @throws
*/
```

Etapa 2: Distribución de responsabilidades

Escenario:

Suponga la sucursal de un banco. En esta se tienen cajeros, asesores de servicios (atienden a los clientes que no tienen productos con el banco y quieren adquirirlos), asesores de productos (atienden a los clientes del banco) y el gerente de la sucursal.

1. Asocie a cada cargo a una responsabilidad

Cargo	Responsabilidad
1. Cajero	a) Generación de chequeras a un cliente del banco
2. Asesor de servicio	b) Firma de documentos de alta importancia
3. Asesor de productos	c) Apertura de nueva cuenta en el banco
4. Gerente de sucursal	d) Recibir dinero de transacciones (Sumas menores a 10 millones)

2. De acuerdo a las responsabilidades asignadas y suponiendo que sólo las personas en ese cargo pueden realizar esas acciones, responda las siguientes preguntas:
    a. ¿Qué debe hacer el gerente de sucursal si está atendiendo una persona que quiere abrir una nueva cuenta en el banco?
    b. Si después de hacer un depósito en el banco, una persona quiere una certificación de la cantidad de dinero que tiene su empresa en dicho banco (la persona tiene una empresa grande). ¿Qué debe hacer el cajero?
    c. Si después de abrir una cuenta, una persona quiere hacer un depósito de $1'000.000 y luego obtener una chequera. ¿Qué debe hacer el asesor de servicio?

Etapa 3: Generación de Excepciones

Escenario:

Suponga que usted trabaja en una compañía de paracaidismo ayudando a las personas a realizar el salto desde el avión. Usted es la última persona con la que tiene contacto la persona antes de realizar el salto al vacío.

1. ¿Qué acciones de verificación realizaría antes de que la persona realizara el salto?

2. ¿Qué acción realizaría si alguna de estas verificaciones falla? ¿Permitiría que se realizara el salto o cómo procedería?

3. Si tuviera un método empujarParaSaltar() en la clase Persona, el cual empuja a esa persona fuera del avión, y un método que se llama realizarVerificaciones() el cual retorna true en caso de que las verificaciones que usted definió anteriormente fueron correctas y false en caso contrario. Realice el método ayudarEnSalto() de acuerdo al contrato (El parámetro persona es de la clase Persona).

```
/**
 * Empuja a una persona para realizar el salto desde el avión. <br>
 * <b>post: </b> Se actualiza el estado de la persona a en el aire. <br>
 * @param persona Persona que se va a ayudar en el salto. persona != null
 * @throws Exception En caso que las verificaciones arrojen un resultado
negativo
 */
```

Sección 2 – Requerimientos funcionales, asignación de responsabilidades y documentación de contratos (Realizar en el ejemplo Club Social)

Para la parte práctica, vamos a agregar una funcionalidad al club social que nos permita agregar nuevos usuarios de tipo beneficiario. De esta manera, será posible que un socio afilie en calidad de beneficiarios a sus hijos menores de 18 años. Para ello, es necesario seguir los siguientes pasos (se recomienda primero crear y documentar el encabezado de los métodos y posteriormente implementarlos):

Nueva clase Beneficiario:

1. Crear una nueva clase Beneficiario que nos permita modelar e instanciar beneficiarios del club.

2. Documentar el encabezado de la clase Beneficiario, indicando su función dentro del sistema Club Social y siguiendo el formato requerido para generar la documentación mediante la herramienta javadoc.

3. Agregar los siguientes atributos a la clase Beneficiario: documento de identidad, nombre y edad.

4. Documentar los atributos siguiendo el formato requerido para generar la documentación mediante la herramienta javadoc.

5. Agregar los métodos correspondientes que nos permitan retornar y establecer el documento de identidad, el nombre y la edad de un beneficiario.

6. Documentar los métodos siguiendo el formato requerido para generar la documentación mediante la herramienta javadoc.

En la clase Socio:

1. Agregar un nuevo arreglo de tamaño variable que nos permita guardar los beneficiarios de un socio, e inicializarlo en el método constructor.

2. Crear y documentar un método que nos permita retornar los beneficiarios de un socio.

3. Crear y documentar un método que nos permita determinar si existe o no un beneficiario dado. El método debe recibir el documento de identidad del beneficiario a buscar.

4. Crear y documentar un método que nos permita agregar un nuevo beneficiario. El método debe recibir el documento de identidad, el nombre y la edad del nuevo beneficiario y retornar una excepción con un mensaje de acuerdo con el error encontrado, de la siguiente manera:

   "El beneficiario no cumple el requisito de edad" – Si la edad no está en el rango [1, 17].
   "El beneficiario ya existe" – Si ya existe un beneficiario con el mismo número de identidad.

"El socio ha llegado al límite de beneficiarios" – Si el socio ya tiene 5 beneficiarios, que corresponde al límite de beneficiarios impuesto por el club.

5. Crear y documentar un método que nos permita eliminar un beneficiario. El método debe recibir el documento de identidad del beneficiario y retornar una excepción con el mensaje "El beneficiario no existe" si el beneficiario no hace parte de la lista de beneficiarios del socio.

En la clase Club:

1. Crear y documentar un método que nos permita retornar el vector de beneficiarios de un socio dado. El método debe recibir la cédula de un socio y retornar una excepción con el mensaje "El socio no existe" si el socio indicado no se encuentra registrado en el sistema. En caso de que el socio no tenga beneficiarios, debe lanzarse una excepción con el mensaje "El socio de cédula _____ no tiene beneficiarios" (En el espacio en blanco debe ir la cédula del socio).

2. Crear y documentar un método que nos permita agregar un nuevo beneficiario a un socio. El método debe recibir la cédula del socio, el documento de identidad, el nombre y la edad del beneficiario. El método debe retornar una excepción si el socio indicado no se encuentra registrado en el sistema.

3. Crear y documentar un método que permita eliminar un socio. El método debe recibir la cédula del socio. Se lanza excepción en caso de que el socio indicado no se encuentre en el sistema, con el mensaje "El socio indicado no existe". También tenga en cuenta que el socio no puede eliminarse si tiene por lo menos un beneficiario, por política del club; si se da este caso debe lanzarse una excepción con el mensaje "El socio tiene ___ beneficiarios, y por lo tanto no puede eliminarse" (En el espacio en blanco debe ir el número de beneficiarios del socio).

Javadoc:

1. Generar la documentación del sistema del Club Social mediante la herramienta javadoc y verificar que ahora se incluya la clase Beneficiario y los nuevos métodos agregados a las clases Socio y Club.

UML:

1. Agregar al modelo UML la nueva clase Beneficiario.