Departamento de Computación
FCEFQyN, Universidad Nacional de Río Cuarto
Asignatura: Estructuras de Datos y Algoritmos - Algoritmos y Estructuras de Datos II
Primer Cuatrimestre de 2025

Práctica No. 2

1. Acceda al código de base del proyecto *my-clock-display* a través del siguiente enlace: https://classroom.github.com/a/bLEYnXpP

   1.1 Cree una nueva clase para representar al ClockDisplay, que permita representar un reloj de 12 horas, en lugar de uno de 24 horas. Tenga en cuenta que el nuevo reloj debe distinguir entre las horas previas al mediodía y posteriores al mediodía, diferenciando a éstas con *am* y *pm*, en la visualización del estado del reloj (método *getTime()*).

   1.2 Agregue test para el método *getTime* de la clase *ClockDisplay*. Realice diferentes pruebas usando los constructores y los diferentes métodos modificadores.

   1.3 Describa los cambios que se requerirían en la clase *ClockDisplay* para poder mostrar horas, minutos y segundos. ¿Cuántos objetos *NumberDisplay* necesitaría utilizar un objeto *ClockDisplay*?

   1.4 Escriba el código para el método *timeTick* en ClockDisplay que considere horas, minutos y segundos. Complete la implementación de la clase.

   1.5 Analice si el diseño actual de la clase *ClockDisplay* admitiría la visualización de *horas*, *minutos*, *segundos*, *décimas de segundo* y *centésimas de segundo*. Considere cómo se implementaría el método *timeTick* en este caso.

2. Acceda al código de base del proyecto *club*. La clase *Club* está diseñada para almacenar objetos de tipo *Membership* en una colección. Dentro de la clase *Club*, definir un atributo de tipo *LinkedList*, usar un import adecuada para este atributo y analizar cuidadosamente sobre el tipo de elemento que almacenará la lista. En el constructor, crear el objeto de la colección y asignarlo al atributo. Asegúrate de que todos los archivos en el proyecto compilen antes de pasar a los siguientes ejercicios:

   2.1 Completa el método *numberOfMembers* para que retorne el tamaño actual de la colección.

   2.2 La membresía de un club está representada por una instancia de la clase *Membership*. Una versión completa de *Membership* ya está implementada en el proyecto *club*. Una instancia de esta clase, contiene detalles del nombre de una persona y el mes y año en que se unió al club. Todos los detalles de la membresía se completan cuando se crea una instancia.

   Un nuevo objeto *Membership* se agrega a la colección de un objeto *Club* a través del método *join* del objeto *Club*, que tiene la siguiente especificación:

```
/**
* @post Add a new member to the club's list of members.
*/
public void join(Membership member)
```

   Completa el método *join*.

   2.3 Utilice la clase ClubTest para agregar nuevos objetos *Membership* al objeto *Club*. Al agregar nuevos objetos, utiliza el método *numberOfMembers* para verificar tanto que el método *join* está agregando correctamente a la colección, como que el método *numberOfMembers* está dando el resultado correcto.

   2.4 Define un método en la clase *Club* con la siguiente especificación:

```
/**
* @post Return the number of members who joined in the given month.
*/
public int joinedInMonth(int month)
```

El parámetro *month* debería ser un mes válido, dentro del rango de 1 a 12, modifica la especificación e implementa el método considerando esta restricción.

2.5 Define un método en la clase *Club* con la siguiente especificación:

```
/**
* @pre {@code 1 <= 'month' <=  12 && 'year' >= 1900}
* @post Remove from the club's collection all members who
* joined in the given month and year. And return a new collection
* with the members object who joined in the given month and year.
*/
public LinkedList<Membership> purge(int month, int year)
```

Considerar las restricciones posibles del parámetro *month*.

3. Acceder al modelo del proyecto *auction* que contiene las clases *Auction*, *Bid*, *Lot*, y *Person*, explorar el código de los métodos de las clases y resolver los siguientes ejercicios:

3.1 Cree clases de test para las diferentes clases del proyecto auction y diferentes métodos de test para probar los métodos implementados en las clases.

3.2 implemente getHighestBid que dado un número de lote retorne la mejor oferta. En caso que no haya oferta retorna null.

3.3 Agrega un método *close* a la clase *Auction*, que cierre la subasta, a partir de ese momento no se pueden hacer mas ofertas. Este método también debe mostrar los detalles de los lotes vendidos.