Universidad Tecnológica de La Habana José Antonio Echeverría

Trabajo de Curso

“Sistema de planificación de guardia”

Corte #2

Autores:

Alfredo Hernández Rodríguez

Ernesto Alejandro Carralero Conde

Tutor:

Isis Torres Pérez

Asignatura: Diseño y Programación Orientada a Objetos

Año académico: 1ro.

Carrera: Ing. Informática

La Habana 2022

Resumen

Con este proyecto se busca informatizar el proceso de planificación de la guardia de la Facultad de Ingeniería Informática para facilitar dicha gestión. Para su realización se analizan las clases que intervienen en la problemática mediante los diagramas de clases UML y las tarjetas CRC. Se crea un sistema en el lenguaje de programación Java donde se automatiza la planificación de la guardia a partir de la información disponible.

**Palabras clave:**

Estudiante, guardia, persona, trabajador.

Introducción

Se plantea la problemática de automatizar el proceso de planificación de la guardia de la facultad de Ingeniería Informática para garantizar un mayor control del personal en esta actividad. Para ello se desarrolla una aplicación para gestionar de manera eficiente este proceso.

**Plan de trabajo**

Se define el plan de trabajo para una organización de las tareas y así lograr una óptima eficiencia. En este se pueden observar las tareas y tiempos en las que se cumplen, así como los principales responsables y participantes en cada una de estas. A continuación se presenta el plan de trabajo a seguir:

**Fecha de inicio: 15/9/2022**

Tabla del plan de trabajo

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **No.** | **Tarea** | **Fecha** | **Responsable** | **Participantes** |
| 1 | Estudiar enunciado | Semana 2 | Todos | Todos |
| 2 | Ver softwares similares | Semana 3 | Todos | Todos |
| 3 | Hacer lista de  funcionalidades (Fase 1) | Semana 3 | Todos | Todos |
| 4 | Elaborar tarjetas CRC (Fase 2) | Semana 4 | Todos | Todos |
| 5 | Hacer diagrama de clases (Fase 2) | Semana 4 | Todos | Todos |
| 6 | Definir el núcleo del software (Fase 3) | Semana 4 | Todos | Todos |
| 7 | Distribuir funcionalidades entre miembros del equipo (priorizando el núcleo) | Semana 4 | Todos | Todos |
| 8 | Para cada funcionalidad:  Elaborar pantallas asociadas | Semana 5 | Ernesto A. | Alfredo H. |
| 9 | Programar prototipo del software (Fase 4) | Semana 6 | Alfredo H. | Ernesto A. |
| 10 | Implementar clases | 5-10 | Todos | Todos |
|  | Realizar pruebas (Fase 4) | 11 | Alfredo H. | Alfredo H. |
|  | Integrar (Fase 4) | 12 | Alfredo H. | Ernesto A. |
|  | Tareas propias de la Fase 5\* |  | Todos | Todos |
|  | Redactar versión final documento | Semana 13 | Todos | Todos |
|  | Entregar documento | Semana 14 | Todos | Todos |
|  | Preparar defensa | Semana 15 | Todos | Todos |
|  | Defensa | Semana 16 | Todos | Todos |

Desarrollo

En esta sección se documentan las tarjetas CRC, el diagrama de clases, el diagrama UML y los reportes.

A continuación se aprecia la distribución de responsabilidades y las colaboraciones entre clases a través de las tarjetas CRC.

Tarjetas CRC

|  |  |  |
| --- | --- | --- |
| Student | | |
| Responsabilidades | Colaboraciones |

|  |  |  |
| --- | --- | --- |
| Person | | |
| Responsabilidades  Dar sus datos | Colaboraciones |

|  |  |  |
| --- | --- | --- |
| Worker | | |
| Responsabilidades | Colaboraciones |

|  |  |  |
| --- | --- | --- |
| Holiday | | |
| Responsabilidades  Dar sus datos | Colaboraciones |

|  |  |  |
| --- | --- | --- |
| Asignment | | |
| Responsabilidades  Guardar las asignaciones | Colaboraciones |

|  |  |  |
| --- | --- | --- |
| Planification | | |
| Responsabilidades  Crear las asignaciones  Distribuir las guardias del periodo lectivo | Colaboraciones  Person  Asignment  Holiday |

|  |  |  |
| --- | --- | --- |
| Vacation | | |
| Responsabilidades  Crear las asignaciones  Distribuir la guardia en el periodo vacacional | Colaboraciones  Asignment  Person |

|  |  |  |
| --- | --- | --- |
| Faculty | | |
| Responsabilidades  Crear las planificaciones | Colaboraciones  Planification  Vacation  Person |

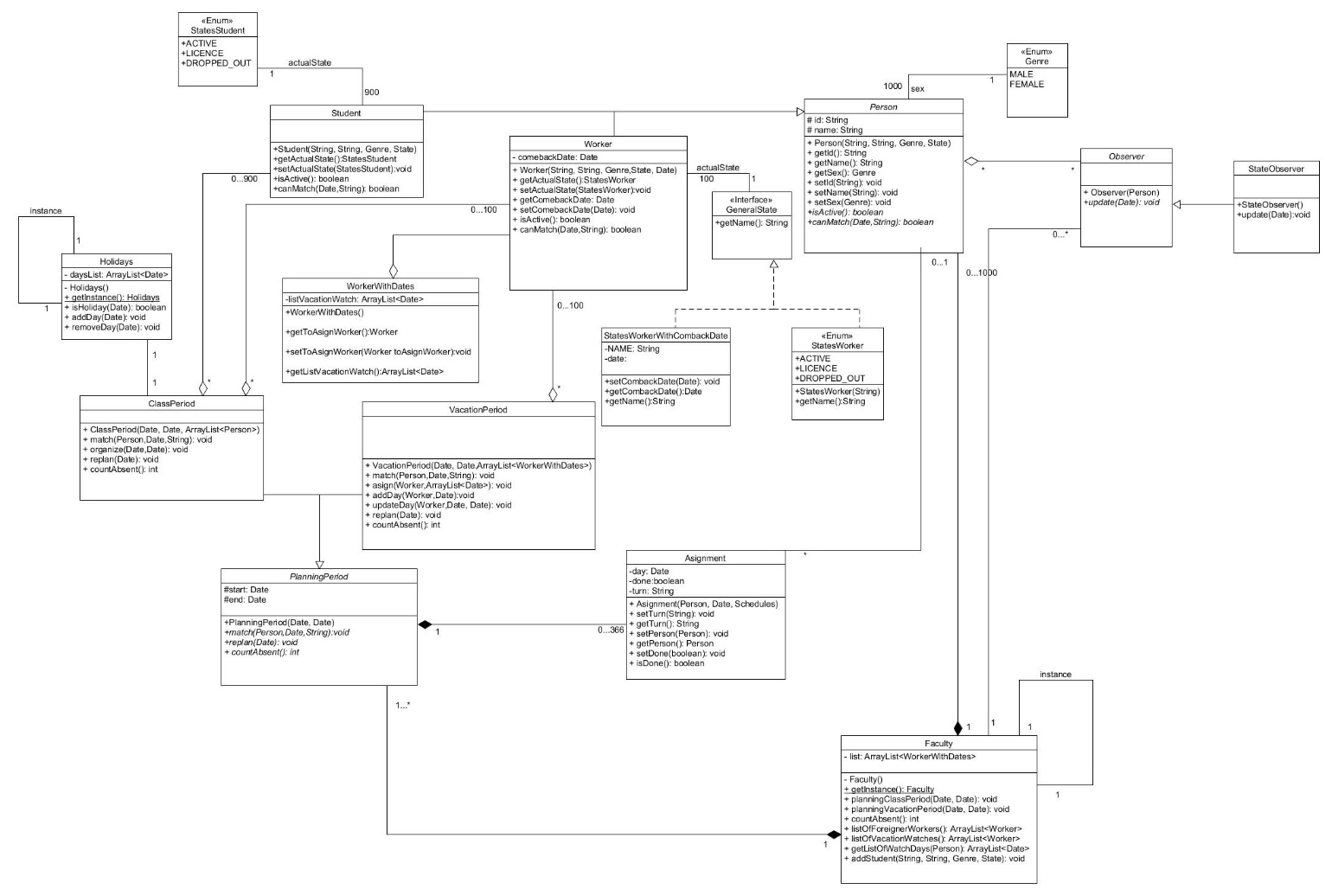
|  |  |
| --- | --- |
| Turn | |
| Responsabilidades  Decir si ese turno se realizó | Colaboraciones |

El proyecto se divide en los siguientes paquetes de contenido:

* *classes*: en este paquete se encuentran las principales clases para la resolución de la problemática, donde se almacena la información y se procesa para determinar los turnos de guardia. Las clases son:
  + - *Asignment*
    - *ClassPeriod*
    - *Faculty*
    - *Holiday*
    - *Person*
    - *PlanningPeriod*
    - *Student*
    - *VacationPeriod*
    - *Worker*
    - *WorkerWithDates*
* *Interfaces*: se implementan dos interfaces, *IOrganize* y *GeneralState*.
* *run*: se encuentra la clase con el *main,* desde aquí inicia la ejecución del programa.
* *utils*: aquí están todas las clases auxiliares, como modelos de tablas,enumeradores, entre otras como Schedule, para facilitar y generalizar la implementación lo más posible.
* *visual*: contiene todas las clases relacionadas con la interfaz gráfica visual, cada una de las pantallas accesibles por el usuario, para así este tenga una interacción con el sistema.

Este modelo tiene como objetivo que la planificación de guardias se pueda hacer de forma dinámica. Se han realizado cambios con respecto al modelo anterior. A continuación, se dará una explicación de cada clase y la función que cumple:

* *Faculty*: Esta es la clase controladora. Los cambios más notables han sido que se le ha aplicado el patrón *Singleton* ya que solo necesitamos una instancia de esa clase. Y una nueva relación la cual no se pudo representar gráficamente en el diagrama y se lleva como atributo, la relación sería con la clase *WorkerWithDates* que a continuación se explicará con más detalle.
* *Person*: Esta clase es la abstracción de la persona que realiza guardia. El único cambio significativo fue dividir el enumerador para sus clases hijas, además de agregar un método abstracto para determinar si puede realizar guardia o no.
* *Student*: Este sería el estudiante que realiza la guardia. Los cambios realizados fueron consecuencias del cambio a persona, como por ejemplo la implementación del método abstracto
* *Worker*: Esta fue la clase con más cambios debido a errores que tenía en el modelo pasado. En un principio se pensó en utilizar herencia para resolver esos problemas, debido a algunas funcionalidades que se tienen pensadas hacer se ha tenido que pensar de otra manera el modelado. La liberó la responsabilidad de que el trabajador lleve la fecha de regreso, dato que solo utilizaba cuando tenía el estado de “Viaje”, la solución que se encontró fue que ese estado en específico cargara con ese dato, la explicación concreta se encuentra en las clases donde se utiliza. El otro cambio fue la necesidad de guardar el listado de fechas, esto se modeló utiliza una clase que solo guarda: un trabajador y un listado de días (que serían las fechas auto propuestas del trabajador)
* *WorkerWithDates*: Esta clase la función que cumple es que antes de crear un periodo de vacaciones se debe de crear esta clase la cual almacenará esas fechas auto propuestas y luego pasa esos datos al periodo de vacaciones que se va a planificar.
* *GeneralState*: Con esta interfaz se solucionó el problema del atributo *comebackDate* ya que, gracias al polimorfismo, puedo hacer que se comporte como un estado u otro. En un principio se pensó que dos *enums* implementaran esa interfaz, uno con los estados convencionales (Activo, De Baja, Licencia) y el otro sea el referente a cuando está de viaje, pero debido a que los atributos de los *enums* son estáticos, se pensó utilizarla como una clase normal y corriente, así es libre de modificarse sin tener que cambiar a todos los estados. Y también se hace que el Trabajador solo tenga fecha de regreso solo si su *GeneralState* se comporta como un estado que tiene fecha de regreso.
* *StatesWorkerWithComebackDate*: Esta es la clase que representaría el estado. La idea es que pueda usarse para llevar esa responsabilidad al propio estado en lugar de que sea responsabilidad del trabajador.
* *PlanningPeriod*: Esta junto a sus clases hijas son el eje del modelo. La idea de estos periodos es que no son fijos, el usuario pueda decidir que periodos van a ser de clase y cuáles de vacaciones. Además de poder planificar la guardia por intervalos de fecha. Dándole al usuario la libertad de poder crear tanto planificaciones de un mes como de varios meses. Ya que posee como atributos una fecha inicial y una fecha final.
* *ClassPeriod*: El periodo de clases sería un periodo que se planifica sin distinción y de manera automática, mientras una persona esté activa y pueda realizar guardia ese día la tomará en cuenta. Como ya se dijo anteriormente, planifica las guardias de manera automática y consecutiva, haciendo rotaciones en los listados de personas que posee.
* *VacationPeriod*: Este es un poco más manual, ya que este sería un periodo donde se da de entrada los días de guardia que la persona quiere hacer. El recibe una lista de la clase nueva que se creó la cual contiene al trabajador con su listado de fechas auto propuestas y a partir de ahí asigna mientras no haya colisión.
* *Asignment:* Esta es la clase creada por los periodos. La idea principal de estas clases que gestionan la planificación de la guardia es que terminen creando asignaciones las cuales representaría una persona asignada a un día y un horario
* *Holiday*: Esta clase es otro *Singleton* que usamos para tener una referencia a los días festivos, previo a cualquier asignación se tiene en cuenta si ese dia se encuentra en la lista de días que posee la clase Holiday
* *Observer*: Esto es aplicando el patrón de diseño *Observer*. Ya que pretendemos que la asignación de la guardia sea dinámica. La planificación puede variar. El marco de referencia que tenemos para que pueda cambiar una planificación es el estado de la persona. Suponiendo el caso de una persona que se consideraba activa deja de estarlo por diferentes motivos, se necesita de una clase que notifique a la clase controladora en este caso, para que los periodos se re-planifiquen. Esta re-planificación necesita de un punto de referencia que sería el día en que ocurrió ese cambio, a partir de ahí se hace una re-planificación a todos los periodos desde esa fecha en adelante. Es para la analogía de que si no se te va a contar para la guardia no tiene sentido tenerte en cuenta para las demás planificaciones, y el caso contrario también, si te reincorporas al sistema como activo ya se te puede contar como alguien más para la realización de la guardia. De momento se mantiene como abstracta debido a la posibilidad de agregar otro Observer concreto que siga una misma analogía.
* *StateObserver*: Esta sería la clase concreta la cual aplica lo explicado en el punto anterior

Diagrama UML

A continuación se muestran los reportes propuestos para facilitar al usuario la gestión de la información y la toma de decisiones del mismo.

Reportes

Alfredo Hernández Rodríguez:

* Cantidad de ausentes: permite conocer la cantidad de ausentes a la guardia en un periodo.
* Cantidad de trabajadores en el extranjero o provincia y la fecha de regreso: permite conocer aquellos trabajadores que se encuentran fuera por viaje y la fecha de incorporación.

Ernesto Alejandro Carralero Conde:

* Dado una persona conocer los días de guardia de esa persona: permite conocer los días de guardia de una persona en específico.
* Listado de personas en vacaciones: permite conocer aquellos trabajadores que realizan guardia en vacaciones.

Para el diseño de interfaces se tuvieron en cuenta las reglas de oro, reducción de la memoria del usuario, este puede consultar la información en todo momento en las correspondientes pantallas, control del sistema, es capaz de entrar los datos y decidir la planificación de los periodos de guardia, en este punto aún quedan funcionalidades por implementar, pero el grueso del problema está completo, además de la consistencia en la interfaz.

Conclusiones

Con este proyecto se espera facilitar la planificación de guardias en la facultad de Ingeniería Informática a través de la realización de tareas como son:

* La confección del diagrama UML.
* La implementación de las clases y reportes en lenguaje java.
* Diseñar una interfaz intuitiva y fácil de manejar por el usuario para el cumplimiento de la planificación de la guardia.
* Realizar las pruebas de software pertinentes.