Análisis y Diseño con Patrones

Práctica 1: Principios de diseño, patrones de análisis y arquitectónicos

(evaluada sobre 50 puntos)

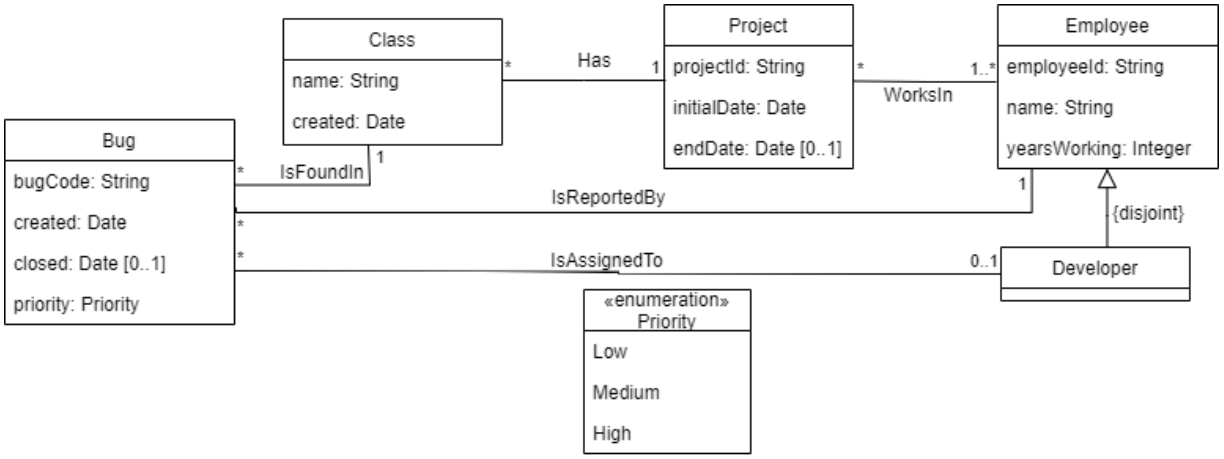
Una empresa quiere crear un software para gestionar los bugs que se producen en el desarrollo de proyectos de software orientados a objetos. Por eso, nos pide que diseñemos una parte del software para gestionar la asignación a desarrolladores, de los bugs que se detectan en los proyectos. Cada proyecto que desarrolla una empresa tiene un identificador de proyecto, una fecha de inicio, una fecha de fin (que puede no tener valor si el proyecto todavía está en curso), el conjunto de clases de objetos definidas en el proyecto y el conjunto de empleados que trabajan en el proyecto.

De las clases de objetos conocemos su nombre (que las identifica dentro del proyecto), su fecha de creación y los bugs hallados en la clase. Los empleados tienen un identificador de empleado, nombre y antigüedad. Existe un subconjunto de empleados que son los que tienen el rol de programadores.

Los bugs se identifican por el identificador del proyecto, el nombre de la clase de objetos donde se encuentra el bug y por el código del bug. Además, de los bugs sabemos la fecha de apertura, la fecha de cierre del bug (que puede no tener valor, si el bug todavía no se ha resuelto), el empleado que reportó el bug y el empleado al que se asigna la resolución del bug (que puede ser diferente al empleado que le reportó).

Disponemos ya de un análisis previo que podemos utilizar como punto de partida, pero que habrá que corregir y mejorar. A continuación disponéis del diagrama estático de análisis para esta parte del software.

Diagrama estático de análisis:



Claves de las clases:

- Project: projectId
- Class: projectId, name
- Bug: projectId, name, bugCode
- Employee: employeeId

Restricciones de integridad:

- La fecha final de un proyecto, si existe, debe ser posterior a la fecha de inicio del proyecto.
- La fecha de creación de una clase de un proyecto es posterior o igual a la fecha de inicio del proyecto y anterior a la fecha de finalización (si existe).
- La fecha de apertura de un bug debe ser posterior a la fecha de creación de la clase en la que se ha encontrado el bug.
- La fecha de cierre de un bug debe ser posterior a la fecha de su apertura.
- El bug de un proyecto debe ser reportado por un empleado del mismo proyecto.
- El bug de un proyecto debe ser asignado a un desarrollador del propio proyecto.

Pregunta 1 (8 puntos)

Enunciado

Después de realizar el análisis previo, nos damos cuenta de que además de grabar las clases de los proyectos, también queremos grabar los paquetes donde están definidas estas clases. Por tanto, los proyectos tendrán componentes que serán de dos tipos diferentes: los paquetes y las clases de objetos. Los paquetes, que tienen un nombre que los identifica y una fecha de creación, sólo pueden contener otros paquetes, paquetes y clases de objetos o sólo clases de objetos. Evidentemente, un paquete no puede contenerse a sí mismo ni directa, ni indirectamente.
Asumamos que una clase de objetos o un paquete puede moverse a un paquete creado posteriormente.

Se pide:

a) (2 puntos) Explica cuál de los patrones de análisis explicados en los materiales utilizarías para representar esta información. Razona la respuesta.
b) (6 puntos) Propón una solución detallada en forma de diagrama estático de análisis para representar esta información (incluyendo los atributos, multiplicidades de las asociaciones, y restricciones de integridad). Muestra sólo las clases que cambien respecto al diagrama del enunciado pero incluye en el diagrama también la clase proyecto y la asociación Has.

Pregunta 2 (12 puntos)

Enunciado

En la versión actual del software que gestiona los bugs que se producen en el desarrollo de proyectos software se graba el desarrollador encargado de resolver un bug (asociación IsAssignedTo). Nos damos cuenta de que, en ocasiones, es necesario reasignar un bug a otro desarrollador ya que el asignado previamente no sabe resolverlo, e incluso, si no hay desarrolladores que se puedan hacer cargo, se puede dejar temporalmente sin desarrollador asignado. Así, queremos almacenar todos los desarrolladores que han sido asignados a los bugs y queremos saber el período en el que estuvieron asignados.

Nos proponen buscar una solución que nos permita representar esa información en nuestro sistema. En concreto, se pide:

a) (2 puntos) Explica cuál de los patrones de análisis explicados en los materiales utilizarías para representar esta información. Razona la respuesta.
b) (5 puntos) Propón una solución detallada en forma de diagrama estático de análisis (incluyendo los nuevos atributos que aparezcan, las restricciones de integridad y las multiplicidades de las asociaciones). Muestra sólo las clases que cambien respecto al diagrama del enunciado.
c) (5 puntos) Propón el pseudocódigo de la operación *priorityBugs*(): *Set(Tuple(projId: String, className: String, bugCode: String))* de la clase *Developer* que devuelve el identificador de los bugs que tiene asignados para resolver y que tiene una prioridad Alta. Podéis obviar el pseudocódigo de los getters de las clases. La operación debe generar el acoplamiento más bajo posible.

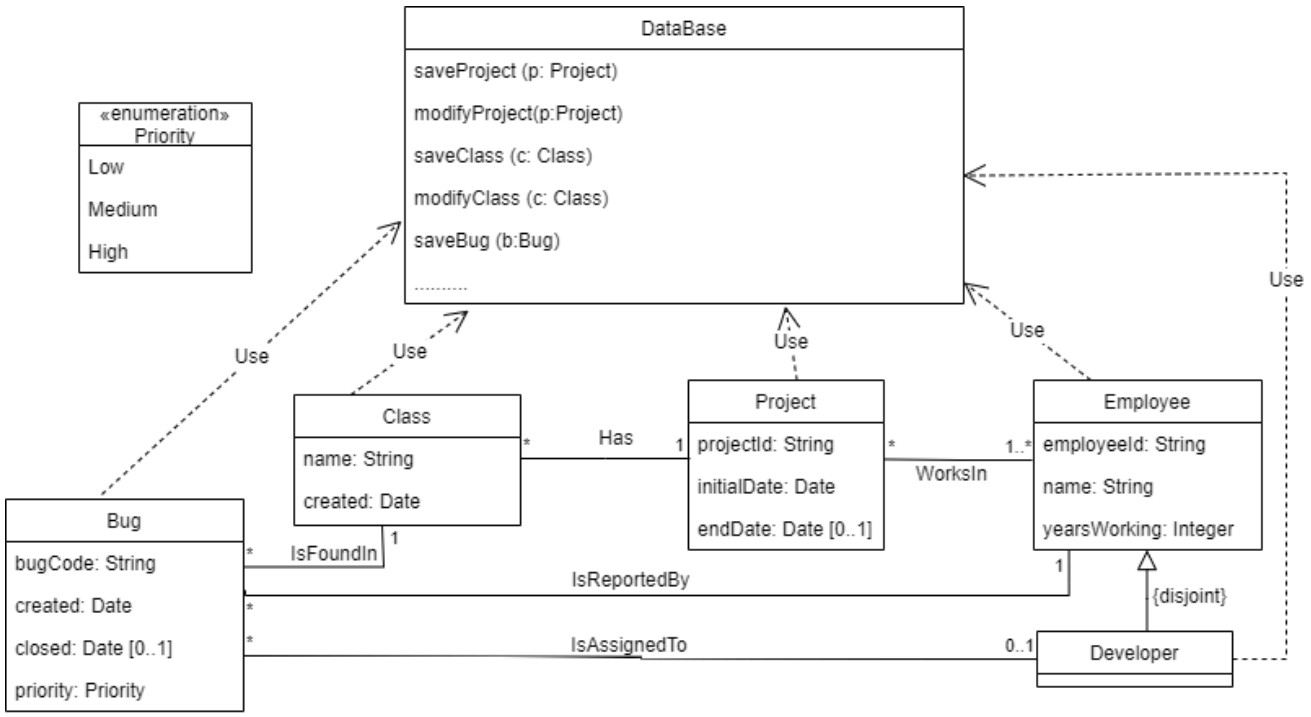
Nota: Para hacer esta pregunta podéis partir del diagrama estático de análisis del enunciado.

Pregunta 3 (4 puntos)

Enunciado

Con el fin de hacer persistente la información de nuestro software, se decide añadir una nueva clase de objetos DataBase, con las operaciones necesarias para obtener y almacenar la información en su base de datos. La clase DataBase tiene operaciones por cada una de las clases que hay en el diagrama estático de análisis. Por ejemplo, para la clase Project, entre otras, están las operaciones: saveProject (p: Project), para almacenar en la base de datos los datos del proyecto p cuando éste no existe y modifyProject (p:Project) que modifica en la base de datos los datos del proyecto existente p. Las operaciones saveProject y modifyProject están invocadas desde operaciones de la clase Project. A continuación dispone de un fragmento del diagrama de clases con esta nueva clase.

Nota: Para hacer esta pregunta podéis partir del diagrama estático de análisis del enunciado.



Se pide:

a) (2 puntos) Indica si se satisface el principio de segregación de interfaces y razona tu respuesta.
b) (2 puntos) Indica si se satisface el principio de inversión de dependencias y razona tu respuesta.

Pregunta 4 (8 puntos)

Enunciado

Si has detectado violaciones de los principios de diseño mencionados en la pregunta anterior, se pide:

a) (4 puntos) Explica cómo solucionarías las violaciones.
b) (4 puntos) Proporciona un diagrama de clases que satisfaga los principios de diseño violados.

Pregunta 5 (12 puntos)

Enunciado

Queremos utilizar el patrón de arquitectura Modelo, Vista y Controlador (MVC) para definir la capa de presentación del software que estamos desarrollando. Empezamos el diseño por el caso de uso "Asignar un desarrollador a un bug". Este caso de uso debe permitir al project manager de un proyecto asignar los bugs a los desarrolladores del proyecto. El caso de uso comienza cuando el project manager selecciona en un desplegable los bugs que no están asignados, el bug que desea

asignar (podéis asumir que la vista ya dispone de los bugs que no están asignados para poder ser mostrados). Una vez seleccionado, el sistema muestra un desplegable con todos los desarrolladores que pueden ser asignados a ese bug. Cuando se selecciona el desarrollador, el sistema registra la asignación y devuelve el mensaje “Successfully assigned”.

Se pide:

a) (8 puntos) Diagrama de clases del MVC para este caso de uso incluyendo todas las operaciones que sean necesarias (las operaciones deben contener los parámetros que necesiten y los tipos de retorno, si las tienen) para desarrollar el caso de uso. Además, debéis proporcionar una pequeña descripción de cada una de las operaciones. El diagrama de clases y las operaciones deben ser específicas para este caso de uso.
b) (4 puntos) Diagrama de secuencia o pseudocódigo del caso de uso, desde que se muestra por pantalla con todos los bugs que no están asignados, hasta que el sistema muestra el mensaje “Successfully assigned”.

Nota: Para hacer esta pregunta podéis partir del diagrama estático de análisis del enunciado.

Pregunta 6 (6 puntos)

Enunciado

Para acabar de diseñar la arquitectura en 3 capas del software que estamos desarrollando, utilizando las tres capas clásicas de muchos sistemas de información:

1. Presentación
2. Dominio
3. Servicios técnicos (Datos)

Queremos reflexionar sobre a qué capa pertenece cada una de las clases que han ido surgiendo en el enunciado (y en la solución que has elaborado hasta ahora):

- Clases del diagrama de clases del ejercicio 1.
- Clases del diagrama de clases del ejercicio 2.
- Clases del diagrama de clases del ejercicio 4.
- Clases del diagrama de clases del ejercicio 5.

a) (2 puntos) Indica qué clases (e interfaces, si las hay) pertenecen a la capa de servicios técnicos (en nuestro caso, capa de datos). Razona tu respuesta.
b) (2 puntos) Indica qué clases (e interfaces, si las hay) pertenecen a la capa de dominio. Razona tu respuesta.
c) (2 puntos) Indica qué clases (e interfaces, si las hay) pertenecen a la capa de presentación. Razona tu respuesta.