Capitulo 5

Pautas de diseño de clase

- Modelando sistemas del mundo real
  - Es uno de los objetivos principales de la programación orientada a objetos.
  - Los datos y el comportamiento están separados lógicamente entre sí.
    - La OOP encapsula los datos y el comportamiento en objetos que se relacionan entre sí
      - Estas interacciones ocurren de manera similar a las interacciones en los objetos del mundo real.
- Clasificación de las interfaces públicas
  - Mantener la interfaz pública al mínimo.
    - El objetivo es proporcionar al usuario una interfaz para hacer el trabajo correctamente.
  - Al propósito de construir una clase es proporcionar algo útil y concreto.
  - Usterfaz de extensión
    - La nueva clase utiliza y crea una nueva clase con una interfaz extendida
- Diseño de contrucciones robustas (y eficaces) interfaces
  - Los objetos deben ser constructos correctamente en lo constructor por defecto.
    - El constructor puso un objeto en un estado seguro inicial.
  - Los destructores incluyen funciones de limpieza adecuadas.
- Diseño con mantenibilidad en mente
  - El diseño de clases coiles y concisas promueve un alto nivel de mantenibilidad
    - Promueve la mantenibilidad ya reduce el código de interdependencia.Esto significa que los cambios en una clase tiene un impacto mínimo o nulo en otras clases
  - Las clases de diseño obligan a organizar el código en muchas piezas manejables.
  - Usando la iteración en el proceso de desarrollo
    - Se recomienda utilizar un proceso iterativo.
    - Cree el código en pequeños incrementos y luego pruébelo en cada paso.
  - Probando la interfaz
    - Las implementaciones internas de la interfaz se denominan stubs.El uso de aplicación se usa para probar las interfaces sin escribir ningún código real.
- Diseñar teniendo en cuenta la extensibilidad
  - Añadir nuevas características a una clase
    - Extender de una clase existente
      - Métodos nuevos
      - Modifica el comportamiento de los demás.
  - Hacer nombres descriptivos.
    - Crea clases, atributos y nombres de métodos.
      - Nombres descriptivos
      - Crear convenciones
        - Leer el nombre debería ser capaz de decir qué cosa representa el objeto.
  - Restrear el código no portátil
    - Al diseñar un sistema que debe usar código no portátil (lo subespecifiqlo se ejecutará en una plataforma de hardware específica) debe aislarse el código de la clase.
  - Proporcionar una manera de copiar y comparar objetos
    - Es importante entender cómo se copian y comparan los objetos.
- Diseñando con la reutilización en mente
  - Para hacer una clase reutilizable en varios sistemas, debe diseñarse con una reutilización en mente
- Diseño de manejo de errores en una clase
  - No es: Suemo ignorar el error potencial
  - Un buen desarrollador anticipa los posibles errores y les condiciones de manejo reales cuando se encuentran.
  - Documentar una clase y usar comentarios
    - Uno de los aspectos más cruciales de un buen diseño, es documentar cuidadosamente el proceso
      - El exceso de comentarios puede ser un problema
        - Demasiada documentación se convierte en ruido de fondo y puede anular el propósito de la documentación.
    - Al diseñar una clase, asegúrese de evitar el lío en cómo otros objetos interactuarán con ella
  - Construyendo objetos con la intención de cooperar
- Uso de la persistencia de objetos
  - Sistema de archivo plano
    - Puede almacenar un objeto en un archivo plano serializando el objeto.
  - Base de datos relacional
    - Es necesario algún tipo de middleware para convertir un objeto a un modelo relacional
  - Base de datos OOD
    - Esta puede ser una forma más eficiente de hacer que los objetos sean persistentes.

UML

- Examinar los beneficios del software de modelado
  - El equipo de desarrollo debe crear una serie de modelos conceptuales que transformen la idea en un sistema de producción.
  - De la arquitectura se puede aprender que:
    - Un modelo es una conceptualización de una idea, un proceso, un sistema político o un sistemas como un software.
    - Los modelos es abstracto por su naturaleza. Los modelos pueden representar muchas cosas diferentes, tanto físicas como estéticas (reales o formales), y corrompuestas como software
    - Diferentes ustes muestran el modelo desde diferentes perspectivas.
      - Un modelo de software se puede construir mediante diferentes vistas, y puede crear diagramas para todas estas vistas.
    - ¿Por qué modelar software?
      - Construimos modelos para poder entender mejor el sistema que se está desarrollando.
    - Específicamente, el modelado le permite
      - Visualizar sistemas nuevos o existentes.
      - Los modelos nos ayudan a entender, conceptualizar y visualizar cualquier tipo de sistema.
    - Comunicar decisiones a las partes interesadas del proyecto.
      - Algunos propietarios de negocios, expertos en dominios y otras partes interesadas del lado del cliente pueden entender fácilmente algunos modelos.
    - Documentar las decisiones tomadas en cada flujo de trabajo OOSD.
      - Dependiendo del tamaño del equipo de desarrollo, no todos los desarrolladores participan en cada flujo de trabajo.
    - Especifique los elementos de estructura (estático) y comportamiento (dinámico) de un sistema.
      - Durante los flujos de trabajo de Diseño e implementación, los aspectos estructurales y dinámicos del sistema de software deben especificarse a partir del modelo de Requisitos.
      - El modelo de solución representa la conceptualización completa del sistema de software.
    - Use una plantilla para construir la solución de software
      - El modelo de solución se utiliza como plantilla para implementar los módulos de código que conforman el sistema de software.
- Explorando el flujo de trabajo de diseño
  - ¿Qué es un modelo?
    - "Un modelo es una simplificación de la realidad"
    - Una descripción de las características reales(o) y/o dinámicas de un área temática, representada a través de varias vistas.
- OOSD como modelo de transformaciones
  - El proyecto comienza como los modelos conceptuales que se transforman en el modelo del cliente. Durante los flujos de trabajo de recopilación de requisitos y análisis, los modelos mencionados se combinan y/o transforman en el modelo de requisitos.
  - Los requisitos no funcionales del modelo de Requisitos se transforman en el modelo de Arquitectura, que define la estructura de alto nivel de la solución de software.
  - Los requisitos funcionales del modelo de Requisitos se transforman en el modelo de diseño, que define los componentes abstractos de la solución de software.
  - El modelo de diseño se combina con el modelo de arquitectura para crear el modelo de solución, que define la estructura detallada de la solución de software. El modelo de solución se utiliza para guiar la construcción del código para la solución de software.
  - Algunos de estos modelos son importantes para capturar y grabar en un artefacto.
    - Algunos artefactos son documentos.
    - Algunos artefactos son diagramas de modelos visuales del sistema.
    - Algunos artefactos son los módulos del programa que conforman la implementación.
  - Definiendo el UML
    - Es un lenguaje gráfico para visualizar, especificar, construir y documentar los artefactos de un sistema de software intensivo.
  - Componentes del modelo
    - Elementos
      - (cosas y relaciones)
    - Diagramas
      - (construidos a partir de elementos)
    - Vistas
      - (diagramas que muestran diferentes perspectivas de un modelo)
  - Elementos UMA
    - Los diagramas UML se construyen a partir de elementos o primitivas de modelado.
    - Hay dos categorías amplias de elementos:
      - Cosas (también llamadas nodos)
      - Relaciones (también llamadas enlaces)
  - Diagramas UML
    - Los diagramas UML le permiten crear visualizaciones de sus modelos mencionados en un sistema de software.
    - Hay muchos diagramas se utilizan para construir muchos de los artefactos en los flujos de trabajo descritos en este curso
    - Tipos de Diagramas
      - Diagramas de Caso de Uso
        - Representa el conjunto de comportamientos de alto nivel que el sistema debe realizar para un actor determinado.
      - Diagramas de Clase
        - Representa una colección de clases de software y sus interrelaciones.
      - Diagrama de Comunicacion (Colaboracion)
        - Representa una colección de objetos que trabajan juntos para admitir algún comportamiento del sistema.
      - Diagrama de Objetos
        - Representa una instantánea de tiempo de ejecución de los objetos de software y sus interrelaciones.
      - Diagrama de Secuencia
        - Representa una perspectiva orientada en el tiempo de una comunicación de objeto.
      - Diagrama de Actividad
        - Representa un flujo de actividades que puede realizar un sistema o un actor.
      - Diagrama de Máquina de Estados
        - Representa el conjunto de estados que un objeto puede ocupar y las transiciones parciales/aleatorias que hacen que el objeto pase de un estado a otro.
      - Diagrama de Componentes
        - Representa una colección de componentes físicos de software y sus interrelaciones.
      - Diagrama de Implementacion
        - Representa una colección de componentes y muestra cómo se distribuyen en uno o más nodos de hardware.
      - Diagrama de Paquete
        - Representa una colección de otros elementos y diagramas de modelado.
      - Diagrama de Informacion General de Interacion
        - Representa una forma de diagrama de actividad que los nodos pueden representar fragmentos de diagrama de interacción.
      - Diagrama de Tiempo
        - Representa cambios en el estado, o valor de un objeto o entidad a lo largo del tiempo, en tiempo y duración, así como las interacciones entre eventos programados.
      - Diagrama de Estructura Compuesta
        - Representa la estructura interna de un clasificador (clase, componente o caso de prueba), y puede incluir los puertos e interfaces de interacción.
      - Diagrama de Perfil
        - Podría definir tipos de diagramas adicionales y extender diagramas existentes con información adicional.
    - Clasificacion de Diagramas
      - Estructural
        - Muestra la estructura estática de los objetos en un sistema
      - Comportamiento
        - Muestra el comportamiento dinámico de los objetos en un sistema
    - Elementos y Conectores
      - Paquete
        - "Se utiliza para agrupar cualquier elemento y/o diagrama.
        - Este paquetes los elementos, y pueden ser anidados.
        - un paquete es una vista lógica.
        - "Los paquetes de tecnología de java son un subconjunto de paquetes UML.
        - Ejemplo
          - Las notas pueden anotar clases, métodos, componentes, actores, asociaciones, etc.
      - Dependencia
        - La notación de dependencia muestra que un elemento UML dependa de otro elemento UML.
        - Grupo de dependencia se puede adjuntar a la línea con un estereotipo.
      - Estereotipos
        - Se utilizan para declarar un tipo de elemento o conector más específico. NOTACIÓN ==
    - Que es y que no es UML
      - Herramientas de modelado UMA
        - Proporcionar dibujos asistidos por computadora de diagramas UML
          - Ejemplo
            - Las herramientas de modelado UML pueden probablemente colocar un nuevo actor en un diagrama de Objeto.
        - Apoyar (o hacer cumplir) la verificación semántica de diagramas.
          - Ejemplo
            - Las herramientas de modelado UML mantendrán una conexión semántica entre una clase llamada Ryo en un diagrama de Clase y la misma clase Ryo en un diagrama de Componentes.
        - Proporcionar soporte para una metodología específica.
          - Ejemplo
            - Rational Software Modeler (RSM) y Rational Software Architect (RSA)
        - Generar esqueletos de código a partir de los diagramas UML.
          - Ingeniería Directa
            - La mayoría de las herramientas UML pueden generar esqueletos de código para uno o más lenguajes de programación, como C ++ y el lenguaje de programación Java.
        - Generación automática de elementos de modelado para patrones de diseño.
          - Java ™ Platform, Enterprise Edition (plataforma Java ™ EE): componentes, etc.
          - Algunas herramientas UML proporcionan mecanismos avanzados que comprenden plantillas de patrones de diseño y otros marcos, como los componentes de la plataforma Java EE.
    - Diagramas de Clase
      - Identificar los elementos de un diagrama de clase
        - Asociaciones
          - Agregacion
            - Implica una relación donde el niño puede existir independientemente del padre.
              - Ejemplo
                - Clase (padre) y Estudiante (niño): Eliminar la clase y los estudiantes cuidado existen.
          - Composicion
            - Implica una relación donde el niño no puede existir independientemente del padre.
              - Ejemplo
                - Casa (padre) y habitación (niño). Las habitaciones no existen separadas de una casa.
            - *Las asociaciones representan relaciones entre clases. Las asociaciones se manifiestan en el tiempo de ejecución en el que dos objetos se asocian entre sí, generalmente con una referencia de objeto.*
        - Multiplicidad
          - La multiplicidad determina cuántos objetos pueden participar en la relación.
        - Navegacion
          - Las flechas de navegación en la asociación direccional que dirección puede recorrer una asociación en tiempo de ejecución.

XMind: ZEN | Trial Mode