Apuntes desorganizados

Page • 1 enlace entrante • Tag

- **Dominio de aplicación** → Considerar a la aplicación como algo aislado de todo lo demás. Unidad lógica
- **Programación estructurada** → Los sistemas contienen datos y programas; Asignación, secuencia, iteración, condicionales
    - Flujo central → "main"
- **Programación Orientada a Objetos** → Los sistemas contienen objetos que se comunican entre sí con mensajes.
    - La clave del éxito es poder agregar nueva funcionalidad, reemplazar o modificar objetos, etc, y que el sistema "no se entere" ni se rompa → encapsulamiento, reusabilidad, herencia, composición, asociación...
    - No hay un objeto "main"
    - Cuando codificamos, describimos clases
    - Una jerarquía de clases no indica lo mismo que la jerarquía Top-Down
    - Cuando se ejecuta el programa, lo que tenemos son objetos que cooperan y que se crean dinámicamente durante la ejecución del programa
    - Este modelo no asume objetos localizados en el mismo espacio de memoria
- **Objeto** → abstracción de una entidad del ***dominio*** del problema → *ámbito de una actividad, orden determinado*
    - Puede representar también conceptos del espacio de la solución
    - **Identidad** → le permite distinguirse de otros objetos
    - **Conocimiento** → en base a sus relaciones con otros objetos y su ***estado interno*** → *se mantiene en variables de instancia* → en general son referencias
    - **Comportamiento** → idioma / vocabulario / conjunto de mensajes (métodos)
        - Indica sus responsabilidades
        - La realización de cada mensaje se especifica a través de un ***método*** → *expresa la forma de llevar a cabo la semántica propia de un mensaje particular* → **ámbito de actuación** → modificar

estado interno de un objeto; retornar y terminar; colaborar con otros objetos

        - Par poder enviarle un mensaje a un objeto, hay que conocerlo (tener una referencia)
    - **Instanciación** → mecanismo de creación de objetos
    - **Clase** → descripción abstracta de un conjunto de objetos (molde)
- **Conocimiento**
    - **Interno** → Variables de instancia
    - **Externo** → Parámetros, ***protocolo*** → *el objeto establece su protocolo para controlar la comunicación de otros objetos con él*
    - **Temporal**
    - **Pseudo-variables** → this, super → toma valor cuando un objeto comienza a ejecutar un método
- **Cualidades/propiedades de los objetos**
    - **Encapsulamiento** → ocultar los detalles de implementación y el estado interno
    - **Herencia** → mecanismo que le permite a una clase "heredar" la estructura de otra
    - **Polimorfismo**
    - Composición
- **Tipo** → Conjunto de firmas de métodos
- **Interfaz** → Declara tipos sin tener que ofrecer ***implementación*** → *la implementación que haga cada clase de una interfaz la chequea el compilador*
- Método lookup()
- **Tipos comunes de colecciones**
    - **List** → Admite duplicados; elementos indexados por enteros de 0 en adelante
    - **Set** → No admite duplicados; sus elementos no están indexados
    - **Map** → Asocia objetos que actuan como claves a otros que actuan como valores
    - **Queue** → Maneja el orden en el que se devuelven los objetos
- **Iterador** → proporciona una manera de recorrer sobre los elementos de una colección de forma secuencial
- **For-each** → no permite modificar la colección

- **Optional** → se utiliza para representar un valor que podría estar presente o ausente en un resultado
- **UML** → Lenguae de modelado visual que permite especificar, visualizar, construir y documentar artefactos de un sistema de Software
    - Fusión del método Booch, el OMT, el OOSE, y otros métodos mas
    - Permite capturar decisiones y comportamientos
    - (Estructura) Diagramas de clases; Diagramas de componentes; Diagramas de despliegue; Diagramas de paquetes; Diagramas de objetos
    - (Comportamiento) Diagramas de caos de uso; Diagramas de interacción (secuencia; colaboración); Diagramas de máquina de estado; Diagramas de actividades
- **Análisis y Diseño Orientado a Objetos**
    - **Modelo del Dominio** (se encuenra adentro de la capa lógica de negocio) → representación visual de las clases conceptuales del mundo real en un dominio de interés
    - Idenificación de clases conceptuales → estrategias → usar una lista de categorías; frases nominales
    - **Construcción de un modelo de dominio**
        1. Listar los conceptos candidatos
        2. Graficarlos en Modelo de Dominio
        3. Agregar atributos a los conceptos
        4. Agregar asociaciones entre los conceptos → centrarse en aquellas para las que se necesita conservar el conocimiento de la relación durante algún tiempo ("need-to-know"). Evitar aquellas asociaciones redundantes o derivadas
    - **Contrato** → Forma de describir el comportamiento en un sistema en forma de tallada → son antesalas a los test de unidad
        - Describen pre- y post-condiciones
- **Heurísticas para Asignación de Responsabilidades**
    - La asignación de responsabilidades (qué debe conocer y hacer cada objeto) por lo general sucede durante la confección de diagramas de secuencia
    - **Experto** → Asignar una responsabilidad al experto en información
    - **Creador** → A una clase B se le debe asignar la responsabilidad de crear instancias de la clase A si:
        - B usa objetos A de manera exclusiva

        - B contiene objetos A
        - B tiene los datos para inicializar objetos A (no necesita que se los pasen por parámetro)
    - **Bajo Acoplamiento** → Medida de la dependencia de un objeto con otros. Es bajo si mantiene pocas relaciones con otros
    - **Alta Cohesión** → Medida de la fuerza con las que se relacionan las responsabilidades de un objeto, y la cantidad de ellas
    - **Entity vs Value Object** → Comparables por su contenido; No viven por si mismos, necesitan una entidad base; Inmutables
    - "No hables con extraños"
    - **Principios SOLID**
        - SRP → Cada clase tiene una tarea
        - OCP → Abierta para la extensión, cerrada para la modificación
        - LSP → Si se crea un objeto de la superclase y se le pone una subclase como tipo, debería andar igual
        - ISP → Numerosas interfaces pequeñas antes que una sola interfaz voluminosa
        - DIP → "Retrasar" la implementación concreta lo mas posible
    - **Herencia vs Composición** → Si cambio algo en la superclase, se cambia para todas las subclases; Se delegan responsabilidades, acoplamiento débil
- **Testing** → Asegurarse de que el programa haga lo que se espera de él bien y de la forma solicitada
    - Tipos → Funcionales; No funcionales; De unidad; De integración; De regresión; Punta a punta; Automatizados; De carga; De performance; De aceptación; De UI; De accesibilidad
    - **Test de Unidad**
        - Asegura que la unidad mínima de nuestro programa funciona correctamente, y aislada de otras unidades
        - Testear un método es confirmar que el mismo acepta el rango esperado de entradas, y que retorna el valor esperado en cada caso
    - **Test automatizados** → Se utiliza software para guiar la ejecución de los tests y controlar los resultados
    - **Cobertura** → Cuan completos/integrales son nuestros tests → clases/métodos/líneas cubiertas; branches...

        - Fixture
        - **Estrategias**
            - Particiones equivalentes
            - **Valores de borde** → los errores ocurren con frecuencia en los límites
- **Smalltalk**
    - Lenguaje de programación orientado a objetos y con tipado dinámico, en el que no se indica explícitamente el tipo de las variables
    - Todo se implementa con objetos y está abierto a modificación, incluendo a las estructuras de control
    - Propone una estrategia exploratoria al desarrollo de Software
    - Hot repair
    - Metamodelo de Smalltalk
- **Javascript (ECMAScript)**
    - Es un lenguaje dinámico, en el que no se indica explícitamente el tipo de la variables, basado en prototipos
    - No hay clases
    - Cada objeto puede tener su propio comportamiento, y heredan comportamiento y estado de sus prototipos
    - Cualquier objeto puede servir como prototipo de otro. Puede cambiar el prototipo de un objeto
- Objetos y persistencia
    - Diferencia de Impedancia → diferencia existente entre el paradigma relacional y el paradigma orientado a objetos
        - acceso a la información
        - datos y lógica separados
        - objeto que representa al sistema
        - tipos de datos
        - colecciones en objetos y en relacional
        - jerarquías en objetos y en relacional
        - identificación de los objetos y de las tuplas
        -
    - Persistencia por Alcance → todo objeto al cual se pueda navegar a partir de un objeto persistente, debe ser necesariamente persistente a su vez

- ORM → Mapeo relacional de objetos