

4.Object, Class, and Profile Diagrams. Class Stereotypes and Associations. Резюме

Обект: Представлява реален или концептуален елемент с **идентичност, състояние и поведение.** **Идентичността** прави всеки обект уникатен, **състоянието** - свойствата на обекта в даден момент, а **поведението** - как обектът реагира на заявки от др.обекти.

Клас: Описание на множество обекти с общи свойства (атрибути), поведение (операции), отговорности, връзки и семантика. Класът е шаблон за създаване на обекти.

Стереотипи: Разширяват модела: «entity», «boundary», «control».

Асоциации: **двуосочни връзки между обекти** от различни класове. Включват **роля и множественост**, като например 1..1 (точно един), 0..1 (нула или един), 1..* (един или повече).

Обектно-ориентиран анализ (ООА): Процес на **дефиниране на проблема** чрез обекти, включващ реални обекти, с които системата трябва да взаимодейства, и кандидат софтуерни обекти за изследване на различни решения.

Обектно-ориентиран дизайн (OOD): Процес на **дефиниране на решението** чрез компоненти, интерфейси, обекти, класове, атрибути и операции, които ще удовлетворят изискванията. Два вида: **архитектурен** (дефиниране на компонентите) и **компонентен** (дефиниране на класовете и интерфейсите в рамките на компонент).

Диаграми на класове: Показват колекция от декларативни (статични) елементи като класове, пакети и техните връзки.

Границни класове- взаимодействие м/у системата и нейната среда; напр. потребителски интерфейси

Контролни класове- контролно поведение за един/повече use cases.

Entity класове- информация и свързаното с нея поведение, което трябва да се съхранява (например данни за събития, лица и т.н.).

Abstraction - показване същественото, скриване несъщественото

Delegation - reusing classes

Encapsulation - затваряне на всички части на абстракцията в **клас**;

Information Hiding - скриване на части от абстракцията в **обект**.

Наследяване (Inheritance) в обектно-ориентираното програмиране

Наследяването е един от основните принципи на обектно-ориентираното програмиране (ООП), който позволява един клас (наречен подklass или дъщерен клас) да наследява характеристики (свойства и методи) от друг клас (наречен суперklass или родителски клас).

Sub-typing (Субтиปизация или логическа класификация)

- **Същност:** Това наследяване се фокусира върху **логическата връзка** между класовете. Подklassът се разглежда като специализация на суперklassа.
- **Ключова идея:** „е вид на“ (is-a relationship).
- **Пример:**
 - **Клас:** Animal (животно)
 - **Подklassове:** Dog (куче), Cat (котка)
 - **Обяснение:** Dog е вид Animal. Тук кучето наследява общите характеристики на животните, но добавя свои специфични.

- **Основна цел:** Подкласовете да се използват там, където се очаква обект от супер класа, без да се нарушава логиката на програмата (Принцип на Лисков за заместване).

Implementation Inheritance (Наследяване на имплементация)

- **Същност:** Този тип наследяване се използва с цел **повторно използване на кода**. Подкласът наследява **конкретната реализация** (методи и функционалности) от супер класа, за да не се пише същият код отново.
- **Ключова идея:** „използвай повторно“ (reuse implementation).
- **Пример:**
 - **Клас:** Logger (клас за логване на съобщения)
 - **Подклас:** FileLogger (логва съобщения във файл)
 - **Обяснение:** FileLogger използва готовата логика от Logger и добавя функционалност за запис във файл.
- **Основна цел:** Намаляване на дублирането на код и улесняване на поддръжката. Подкласът не винаги е „вид“ на супер класа в логическия смисъл, но използва неговата имплементация за удобство.

Съществуват два основни типа наследяване:

Class-Responsibility-Collaborations (CRC) card - класове - карти: описва класа, атрибути му и отговорности

Диаграми на обекти- моментна снимка на инстанции на класове в опр. момент, помагайки за изясняване на класовете и наследяването.

Профилните диаграми са структурни диаграми, предоставящи общ механизъм за разширяване и персонализиране на UML-модели за конкретни домейни или платформи. Вкл.: **stereotypes; tagged value**

definitions- keyword-value pairs of attributes; **constraints** - as a string enclosed in brackets near associated element; OCL boolean expression

visibility ::= { + | - | # | ~ }

Derived Attributes- / - attribute may not be strictly necessary

Optional Property Modifiers { id}, {readOnly}, {unique}

Direction (optional) - in, inout, out, or return

Self-Associations - един екземпляр от класа има асоциации към други екземпляри от същия клас