Лекція з навчальної дисципліни: **«Крос-платформне програмування»**

Тема 2. Компонентна ідеологія Заняття 2.1. Компоненти та інтерфейси

КОВАЛЕНКО Олексій Єпифанович,

к.т.н., доцент СК №5

Навчальні питання:

1. Визначення та властивості компонентів.
2. Модель посилань.
3. Стратегії інтеграції програмного забезпечення.

Література:

1. Кулямин В. В.. Технологии программирования. Компонентный подход. М. Интернет-университет информационных технологий — БИНОМ. Лаборатория знаний, 2007. – 316 с. – Режим доступу: http://panda.ispras.ru/~kuliamin/lectures-sdt/sdt-book-2006.pdf.
2. Степанов Е.О. Кросс-платформенные и многозвенные технологи. / Интернет-университет информационных технологий - ИНТУИТ.ру, 2010 – Режим доступа: http://www.intuit.ru/department/se/crosspl/
3. Лавріщева К.М. Програмна інженерія. – К.: Академперіодика, 2008. – 319 с.
4. Таненбаум Э., ван Стеен М. Распределенные системы. Принципы и парадигмы. СПб.: Питер, 2003. – 877 с.

1. Визначення та властивості компонентів

Компонент – самостійний продукт, що підтримує об'єктну парадигму, реалізує окрему предметну область і може взаємодіяти з іншими компонентами через інтерфейси.

Компоненти програм:
дані і їх структури (прості і складні),
функції і **композиції**,
базові об’єкти (модуль, компонент, каркас, контейнер, компонент повторного використання (КПВ) тощо)
цільові об’єкти (програмне забезпечення, програмна система, сімейство систем, програмний проект, складні програмні застосування тощо).

Об'єкти розглядаються на логічному рівні проектування ПС, а **компоненти** – це безпосередня фізична, тобто програмна реалізація об'єктів. Один компонент може бути реалізацією декількох об'єктів або навіть деякої частини системи, отриманої при проектуванні.

Схема еволюції повторних елементів компонентного типу

Елемент композиції	Опис елемента	Схема взаємодії	Форма збереження	Результат композиції
Процедура, підпрограма, функція	Ідентифікатор	Безпосереднє звертання, оператор виклику	Бібліотеки підпрограм і функцій	Програма на мові програмування
Модуль	Паспорт модуля, зв'язки	Виклик модулів, інтеграція модулів	Банк, бібліотеки модулів	Програма з модульною структурою
Об'єкт	Опис класу	Створення екземплярів класів, методів	Бібліотеки класів	Об`єктно-орієнтована програма
Компонент	Опис логіки, інтерфейсів (API, IDL), розгорнення	Вилучений виклик у компонентних моделях систем (COM, CORBA, OSF, JAVA, ...)	Репозитарій компонентів, сервери, контейнери компонентів	Розподілена компонентно-орієнтована програмна система
Сервіс	Опис логіки і інтерфейсів (XML, WSDL...)	Віддалений виклик (RPC, HTTP,INVOKE, SOAP, ...)	Індексація і каталогізація сервісів (XML, UDDI, ...)	Розподілене сервісо-орієнтоване застосування

Компоненти конструюються самостійно, як деяка абстракція, що містить у собі ***інформаційну частину*** й ***артефакт*** (специфікація, код, каркас тощо).

- В **інформаційній частині** задаються відомості: призначення, дата виготовлення, умови застосування (ОС, середовище, платформа тощо).
- **Артефакт** – це реалізація (implementation), інтерфейс (interface) і схема розгортання (deployment) компонента.
- ***Реалізація*** – це код, що буде виконуватися при зверненні до операцій, визначених в інтерфейсах компонента.

Інтерфейс відображає операції звертання до реалізації компонента, описується в мовах IDL або API, містить у собі опис типів і операції передачі аргументів і результатів для взаємодії компонентів. Компонент, як фізична сутність, може мати множину інтерфейсів.

Розгортання – це виконання фізичного файлу відповідно до конфігурації (версії), з урахуванням параметрів запуску системи.

Типи компонентних структур

Розширенням поняття компонента є **шаблон (зразок, паттерн)** – абстракція, що містить у собі опис взаємодії сукупності об'єктів у загальній кооперативній діяльності, для якої визначені ролі учасників і їхня відповідальність.
Шаблон є повторюваною частиною програмного елемента або схемою вирішення проблеми

Компонентна модель – відбиває проектні рішення щодо композиції компонентів, визначає типи шаблонів компонентів і припустимі взаємодії між ними, а також джерело формування файлу розгортання ПС у середовищі функціонування.

Каркас являє собою високорівневу абстракцію проекту ПС, у якій функції компонентів відділені від задач керування ними.

- ***Компонентне середовище*** – розширення класичної моделі клієнт–сервер з урахуванням специфіки побудови і функціонування програмних компонентів, а також результатів практичних реалізацій і їхньої апробації.
- ***Контейнер*** – це оболонка, усередині якої реалізується функціональність компонента. Взаємодія контейнера із сервером строго регламентована і здійснюється через стандартизовані інтерфейси.

Складові компонентного середовища

– сервери компонентів;
– контейнери компонентів;
– реалізації функцій, подані як екземпляри усередині контейнерів;
– реалізація компонентних моделей, об'єктів, що задовольняють установку і конфігурування окремих компонентів для деякої комп'ютерної платформи;
– клієнтські компоненти і інтерфейси, що забезпечують кінцевого користувача, реалізовані у вигляді різних типів клієнтів (веб-клієнти, повноцінні реалізації графічного інтерфейсу і т.д.);
– компонентний додаток, як сукупність компонентів.

Інтерфейс компонентів

Інтерфейс відображає перелік сервісів, вхідні та вихідні параметри сервісів та їхні типи, переду- і постумови функціонування компонента, а також перелік інших компонентів, сервіси яких потрібні для здійснення своїх сервісів

Домашній (Home interface) забезпечує керування екземплярами компонента з обов'язковими реалізаціями методів пошуку, створення і видалення окремих екземплярів.

Функціональні інтерфейси (function interface), що забезпечують доступ до реалізації компонента.

Інтерфейс як контракт

Семантика інтерфейсу компонента може бути представлена за допомогою ***контрактів***, що визначають зовнішні обмеження і підтримують інваріант, який містить у собі правила встановлення взаємозв’язків властивостей компонента або умови його життєздатності.

Для кожної операції компонента *контракт* може визначати обмеження, що повинні бути враховані клієнтом перед викликом операції (***передумова***), і ***постумови*** перевірки правильності функціонування компонента після завершення операції.

Перед- і постумова визначають специфікацію поведінки компонента і залежать від стану компонента, а також інтерфейсу і зв'язаним з ним набором інваріантів.

Контракти й інтерфейс зв'язані між собою, але їхні сутності різні.

Контракт задає опис поведінки компонента, націлений на взаємодію з іншими компонентами і відбиває семантику функціональних властивостей компонента.

Інтерфейс являє собою колекцію операцій або функціональних властивостей специфікації сервісів, що підтримує компонент.

Модель специфікації семантики компонента визначає його інтерфейс і обмеження. Кожен інтерфейс складається з набору операцій (сервісів, що він пропонує або потребує). З кожною операцією зв'язаний набір перед- і пост-умов.

Типи композицій компонентів

- компонент з компонентом зв’язані через інтерфейс на рівні застосування;
- каркас з компонентом зв’язані через інтерфейси на системному рівні;
- компонент з каркасом взаємодіють через інтерфейси на сервісному рівні;
- каркас з каркасом взаємодіють через інтерфейси на мережному рівні.

Компоненти повторного використання (КПВ)

Повторне використання в компонентному програмуванні – це застосування готових порцій формалізованих знань, здобутих під час попередніх реалізацій ПС, у нових розробках систем

Компоненти повторного використання (КПВ) – це готові компоненти, елементи оформлених знань (проектні рішення, функції, шаблони й ін.), що використовуються у ході розроблення не тільки самими розробниками, а й іншими користувачами шляхом адаптації їх до нової ПС, що спрощує і скорочує терміни її розробки

Етапи життєвого циклу компонентної ПС

1. *Пошук, вибір КПВ* і розроблення нових компонентів, виход ячи із системи класифікації компонентів і їхньої каталогізації, формалізоване визначення специфікацій інтерфейсів, поводження і функціональності компонентів, а також їхнього анотування і розміщення в репозитарії системи або в Інтернеті.
2. *Розроблення вимог* (Requirements) до ПС – це формування й опис функціональних, нефункціональних і інших властивостей ПС.
3. *Аналіз поведінки* (Behavioral Analysis) ПС полягає у визначенні функцій системи, деталей проектування і методів їхнього виконання.
4. *Специфікація інтерфейсів і взаємодій компонентів* (Interface and Interaction Specification) віддзеркалює розподіл ролей компонентів, інтерфейсів, їхню ідентифікацію і взаємодію компонентів через потоки дій або робіт (workflow).
5. *Інтеграція набору компонентів і КПВ* (Application Assembly and Component Reuse) у єдине середовище ґрунтується на підборі й адаптації КПВ, визначенні сукупності правил, умов інтеграції і побудові конфігурації каркаса системи.
6. *Тестування компонентів і середовища* (Component Testing) ґрунтується на методах верифікації і тестування для перевірки правильності як окремих компонентів і КПВ, так і інтегрованої з компонентів програмної системи.
7. *Розгортання (System Deployment)* містить у собі оптимізацію плану компонентної конфігурації з урахуванням середовища, розгортання окремих компонентів і створення цільової компонентної конфігурації для функціонування ПС.
8. *Супровід ПС* (System Support and Maintenance) складається з аналізу помилок і відмов при функціонуванні ПС, пошуку і виправлення помилок, повторного її тестування й адаптації нових компонентів до вимог і умов інтегрованого середовища.

2. Модель посилань зберігання об’єктів

Модель посилань - означає, що змінна типу **клас** зберігає в дійсності вказівник на об'єкт.

Кожен об'єкт містить копії всіх полів, визначених у його класі, і може користуватися всіма його методами.

Однак, при зверненні до полів, методів або властивостей об'єкта розіменування такого вказівника не вимагається;

вказується ім'я об'єкта і потім, після роздільника-точки, вказується ім'я поля, методу або властивості.

```
var p: Person := new Person(‘Іванчук',20);
p.Print;
```

змінна типу клас може зберігати значення **nil**:

```
p := nil;
...
if p = nil then ...
```

Кілька змінних типу клас можуть посилатися на один об'єкт і спільно модифікувати його:

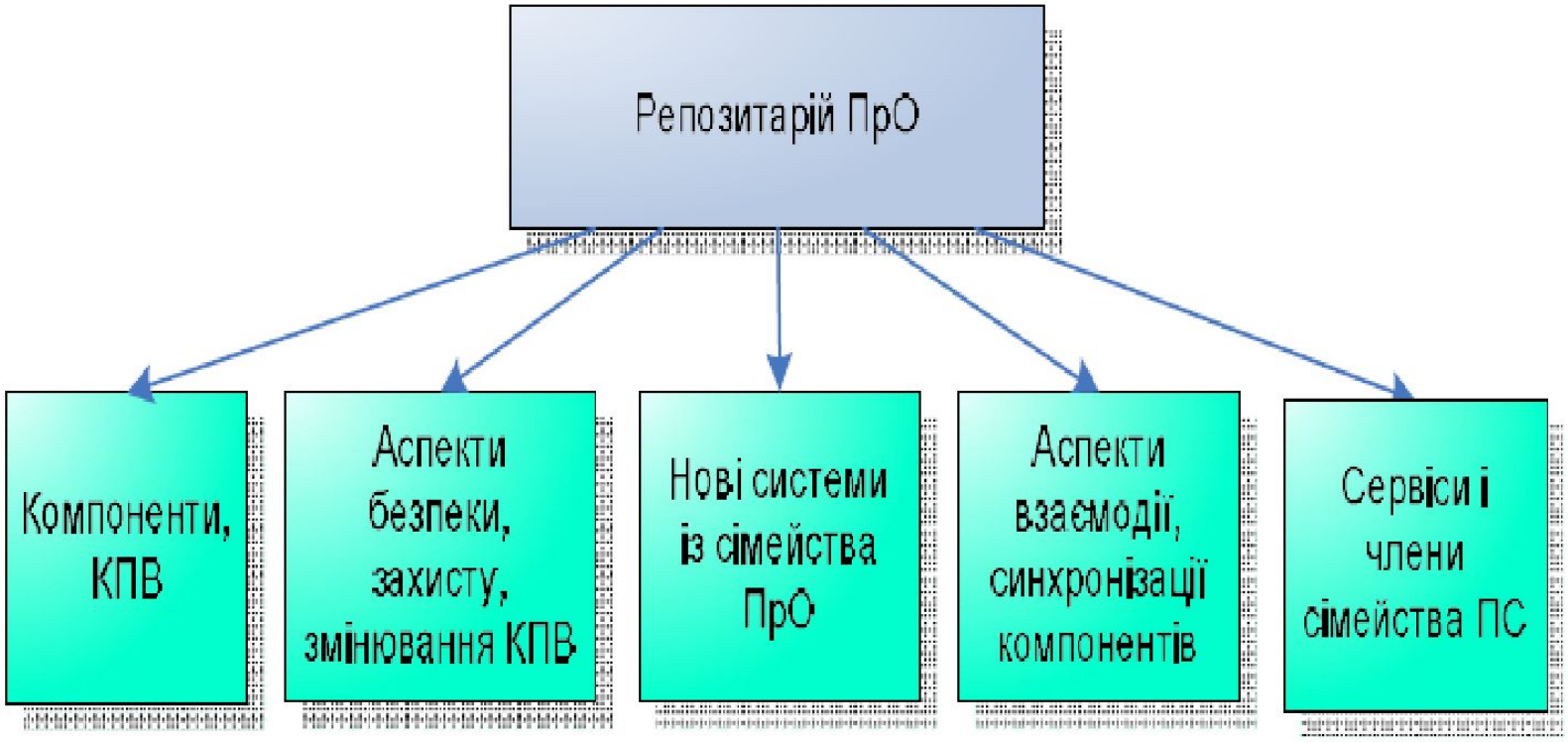
```
var p1,p2: Person;
...
p1 := new Person('Петренко',20);
p2 := p1;
p1.IncAge;
p2.Print; // Ім’я: Петренко Вік: 21
```

3. Стратегії інтеграції програмного забезпечення

Репозітарій компонентів

репозитарій – це система засобів для зберігання, поповнення напрацьованих КПВ, що містить у собі інфраструктуру розробки ПС з компонентів, організацію доступу до КПВ, які розташовані в ньому, для подальшого їхнього застосування в нових проектах

Структура репозитарію для ПрО



Інформаційну потребу щодо КПВ формулює користувач у вигляді ***пошукового запиту***, який зіставляється з описом ***пошукового образу*** КПВ, що зберігається у репозитарію.

Пошуковий образ готових компонентів в репозитарії може містити у собі:

– список ключових слів, що найчастіше згадуються в тексті КПВ;

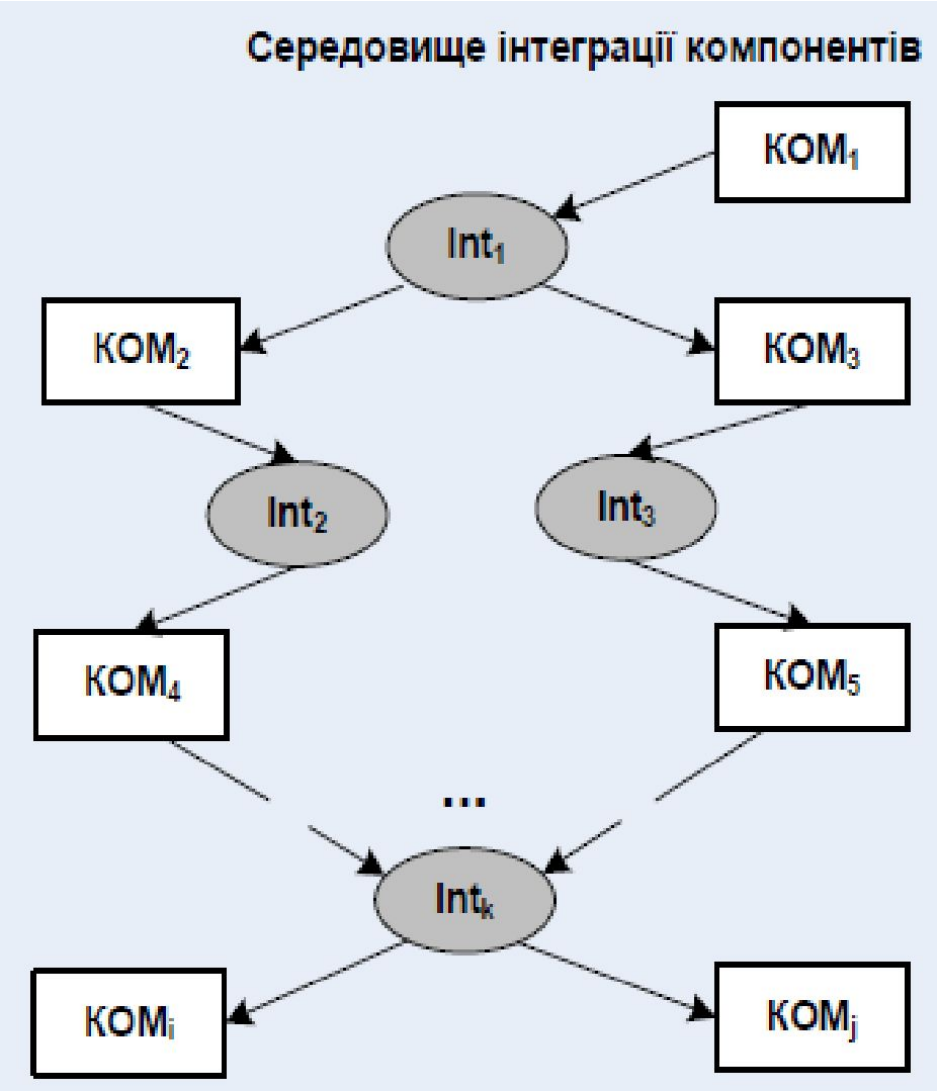
– посилання на заздалегідь побудовану онтологію домену проблемної області, до якої цей КПВ належить.

Ознаки класифікації КПВ

– приналежність до ПрО;
– тип компонента (модуль, клас та ін.);
– наявність інтерфейсу в КПВ;
– готовність КПВ до застосування;
– складність КПВ (каркас, патерн, контейнер тощо).

Методологія компонентної розробки ПС

Інтеграція компонентів на різних МП



Шляхи інтеграції компонентів

1. Розроблення компонентів (КОМ) мовою програмування.
2. Відбір готових компонентів – КПВ.
3. Розроблення інтерфейсів – Int для КОМ.
4. Генерація інтерфейсів пари КОМ на МП.
5. Розроблення середовища та репозитарію КОМ.
6. Типізація компонентів.
7. Тестування КОМ, інтерфейсів, КОМ-систем.
8. Інтеграція компонентів.
9. Розгортання КОМ-системи у середовищі .
10. Супровід компонентної системи.

Для об'єднання компонентів у ПС необхідною умовою є наявність для них формально визначених інтерфейсів у сучасних мовах IDL або API, а також механізмів динамічного контролю зв'язків між компонентами.

Синтаксис типового інтерфейсного модуля у формі Бекуса–Наура:

<інтерфейс об’єкта> ::= **object** *<ім’я_Об’єкта>* :{<множина початкових інтерфейсів>};{<множина вхідних інтерфейсів>} **end**;

<множина вхідних інтерфейсів> ::= <множина інтерфейсів>;

<множина вихідних інтерфейсів> ::= <множина інтерфейсів>;

< множина інтерфейсів > ::= ∅ | <інтерфейс>; < множина інтерфейсів >;

<інтерфейс> ::= **interface** *<ім’я _інтерфейса>***:**{<множина функцій>} **end**;

<множина функцій > ::= ∅ | <функція>; <множина функцій>;

<функція> ::= **function <** *ім’я_функції>:* <множина параметрів> **end;**

<множина параметрів> ::= <параметр> | <параметр>, <множина параметрів >;

<параметр> ::= <тип> (<вид параметра>);

<вид параметра> ::= **in | out | inout** (вхідний, вихідний, разом вхідний і вихідний).