ЛАБОРАТОРНА РОБОТА №1

Розробка інтерактивного інтерфейсу користувача до бази даних

Метою лабораторної роботи є оволодіння навичками проектування архітектури та розробки програмних засобів, які надають інтерактивний інтерфейс користувача до бази даних на основі патерну “Модель-Подання-Поведінка”.

Основні теоретичні відомості

Сучасні інформаційні технології пропонують значну кількість різноманітних засобів організації баз даних та людино-машинного інтерфейсу. Програмні рішення, придатні підтримувати одразу декілька таких засобів, вимагають архітектурної гнучкості, яка дозволяла б замінювати певні компоненти без переробки іншого програмного коду. При чому, з огляду на розробку програмних засобів з інтерактивним інтерфейсом, основними компонентами, які можуть змінюватись, є засоби відображення інформації користувачу — одна й та сама інформація може бути подана як таблиця, як діаграма, може бути виведена на екран, записана в файл чи надрукована на принтері тощо; та алгоритми реагування на дії користувача — певна дія, наприклад, обрання пункту меню чи натиснення клавіші, може призводити до ряду операцій, послідовність яких залежить від налаштувань інтерфейсу. Вимоги до зменшення залежностей від компонентів, які можуть змінюватись, призвели на початку 80-х років XX-ого ст. до появи архітектурного патерну “Модель-Подання-Поведінка” (англ. MVC — Model-View-Controller), який і дотепер, з деякими модифікаціями, є основним стандартом організації програмних засобів з інтерактивним інтерфейсом користувача.

Підхід “Модель-Подання-Поведінка” вимагає від розробника виділити три основні компоненти:

1. Модель — описує інформацію про певні сутності предметної галузі та їх поведінку;
2. Подання — визначає, способи візуалізації інформації;
3. Поведінка — забезпечує зв'язок між користувачем та системою.

Подання та Поведінка залежать від Моделі, тобто зміни в Моделі призводитимуть і до змін в інших компонентах, а от Модель не залежить ні від Подання, ні від Поведінки. Такий підхід дозволяє відокремити розробку Моделі від розробки інших компонентів та зменшує залежність програмного коду від апаратної чи програмної платформи.

Функції модуля Поведінки полягають в забезпеченні правильного порядку взаємодії Моделі та Подання. Як правило, саме цей модуль має інформацію про відповідні об'єкти Моделі та Подання, та, отримавши інформацію з Моделі, повідомляє про неї Подання. Але, в багатьох випадках, доцільно буває функції повідомлення перекласти на саму Модель, оскільки їй краще відомо в який момент що змінюється. В такому випадку, для того щоб уникнути сильного зв'язку між Поданням та Моделлю, застосовується патерн Спостерегач.

Розглянемо приклад побудови програмного застосунку на основі патерну “Модель-Подання-Поведінка” мовою програмування Python. Розробки інтерактивного віконного інтерфейсу мовою Python може виконуватись за допомогою ряду бібліотек, зі списком яких можна ознайомитись за посиланням http://wiki.python.org/moin/GuiProgramming. Крім бібліотек зовнішніх розробників до складу стандартної бібліотеки мови входять міжплатформрмені засоби побудови інтерфейсу Tkinter, які будуть використані в подальшому.

Нехай, програмні засоби, які буде розроблено, будуть призначені для розрахунку вартості певного продукту з урахуванням податку. Почнемо розробку з Моделі. Програмний код Моделі може містити лише ту інформацію, яка пов'язана з предметною галуззю, тобто вартість продукту, величину податку та алгоритм розрахунку остаточної вартості. Оскільки буде застосовано патерн Спостерегач, реалізуємо допоміжний клас, який дозволить повідомляти Подання про зміни в Моделі.

```
class Observable(object):

    def __init__(self, value=None):
        self.value = value
        self.callbacks = []

    def addCallback(self, func):
        self.callbacks.append(func)

    def set(self, value):
        self.value = value
        for func in self.callbacks:
            func(self.value)
```

Цей клас призначено для обгортання будь-якого значення, зміни якого планується відстежувати. Крім самого значення value він містить ще список функцій, які викликатимуться при його зміні. Метод addCallback() дозволяє додати нову функцію до цього списку, а метод set() змінює значення value та проходить по списку функцій, викликаючи їх послідовно.

Тепер можна реалізувати Модель (в реальних проектах бажано обирати змістовніші імена ніж Model).

```
class Model(object):

    TAX = 0.2

    def __init__(self):
        self.price = Observable(0)

    def calcPrice(self, price):
        self.price.set(price * (1 + self.TAX))
```

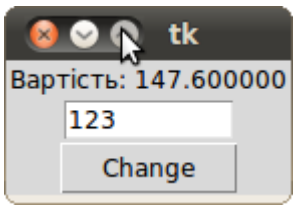
Значення price є об'єктом класу Observable, що дозволяє Моделі в подальшому інформувати про всі зміни вартості.

Подання залежить від Моделі, тобто при його розробці враховуються параметри, потрібні для розрахунку вартості, тому при зміні алгоритму функціонування Моделі Подання доведеться переробити. Але, з іншого боку, Моделі про Подання нічого невідомо, тому інформацію про вартість можна отримувати як за допомогою віконного інтерфейсу, так і з консолі або з файлу.

```
# coding: utf-8

import Tkinter as Tk

class View(Tk.Toplevel):

    def __init__(self):
        self.root = Tk.Tk()
        self.root.withdraw()
```

```
        Tk.Toplevel.__init__(self, self.root)
        self.protocol('WM_DELETE_WINDOW', self.master.destroy)
        self.priceValue = Tk.Label(self, text='Вартість: 0')
        self.priceValue.pack(side='top')
        self.priceEntry = Tk.Entry(self, text='0', width=10)
        self.priceEntry.pack(side='top')
        self.changeButton = Tk.Button(self, text='Change',width=8)
        self.changeButton.pack(side='top')

    def run(self):
        self.root.mainloop()

    def setPriceChanger(self, changer):
        self.changeButton.config(command=changer)

    def getPrice(self):
        return self.priceEntry.get()

    def setPrice(self, value):
        self.priceValue.config(text='Вартість: %f' % value)
```

В даному випадку за допомогою бібліотеки Tkinter створюється вікно, в якому можна ввести значення вартості без урахування податку, та, натиснувши відповідну кнопку, отримати перераховану вартість.



Нарешті, модуль Поведінки призначено для того, щоб зв'язати між собою Модель та Подання. Саме на цьому етапі відбувається визначення, який саме спосіб Подання буде обрано.

```
class Controller(object):

    def __init__(self):
        self.view = View()
        self.view.setPriceChanger(self.changePrice)
        self.model = Model()
        self.model.price.addCallback(self.priceChanged)
        self.view.run()

    def changePrice(self):
        self.model.calcPrice(int(self.view.getPrice()))

    def priceChanged(self, price):
```

```
        self.view.setPrice(price)


app = Controller()
```

Таким чином, використання патерну “Модель-Подання Поведінка” дозволяє вирішити наступні задачі:

– для однієї Моделі може бути розроблено декілька модулів Подання;

– не змінюючи Подання, можна змінити реакцію на дію користувача, змінивши модуль Поведінки;

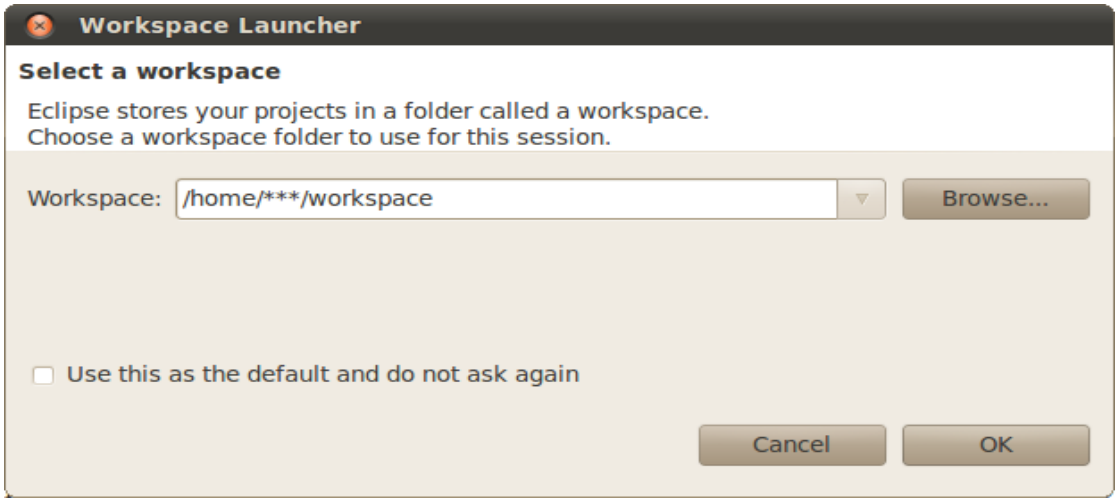
– кожний компонент системи може реалізовувати окремий розробник, який спеціалізується лише на відповідних технологіях.

Методичні вказівки

Для виконання лабораторної роботи необхідно встановити та налаштувати інтерпретатор мови програмування Python та інтегроване середовище розробки Eclipse. При використанні Linux-систем інтерпретатор Python, як правило, встановлюється під час інсталяції, в іншому випадку, для його встановлення можна скористатися менеджером пакунків. Для інших платформ інсталятор інтерпретатора може бути завантажено з http://python.org/download/. Зазвичай після встановлення інтерпретатор Python додаткових налаштувань не потребує.

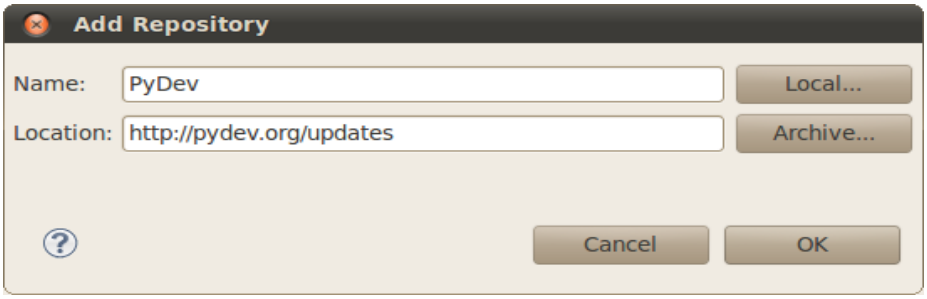
Інтегроване середовище розробки Eclipse можна знайти за адресою http://www.eclipse.org/downloads/. Оскільки для роботи з Python використовується додатковий пакунок PyDev, можна обрати довільний з запропонованих інсталятор, наприклад Eclipse IDE for C/C++ Developers.

Після завантаження архівного файлу, його необхідно розпакувати, отримавши при цьому каталог eclipse. Перший запуск інтегрованого середовища призводить до появи вікна, в якому запропоновано обрати шлях до робочого простору.

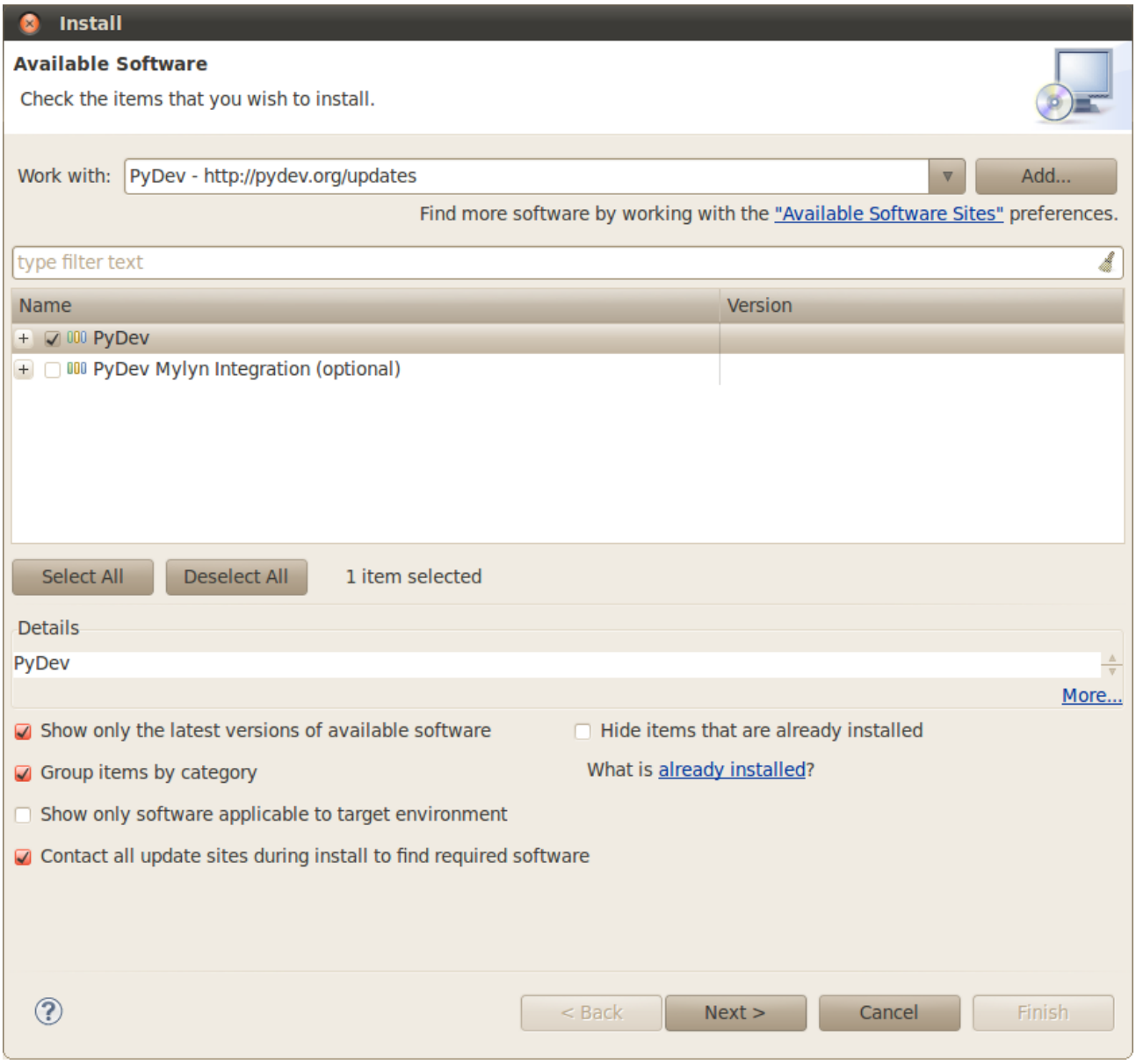


Робочій простір є каталогом, в якому користувач зберігає файли своїх проектів. В одному середовищі можна використовувати декілька робочих просторів для розділення проектів.

Після обрання робочого простору завантажується основне вікно системи і наступним кроком необхідно встановити пакунок PeDev. Для цього в головному меню програми потрібно обрати пункт Help/Install New Software. У вікні інсталяції необхідно натиснути кнопку “Add” встановити параметри сайту завантаження.



Після завантаження інформації про пакунки у вікні інсталяції з'явиться позиція PyDev, яку потрібно обрати та натиснути кнопку “Next”.

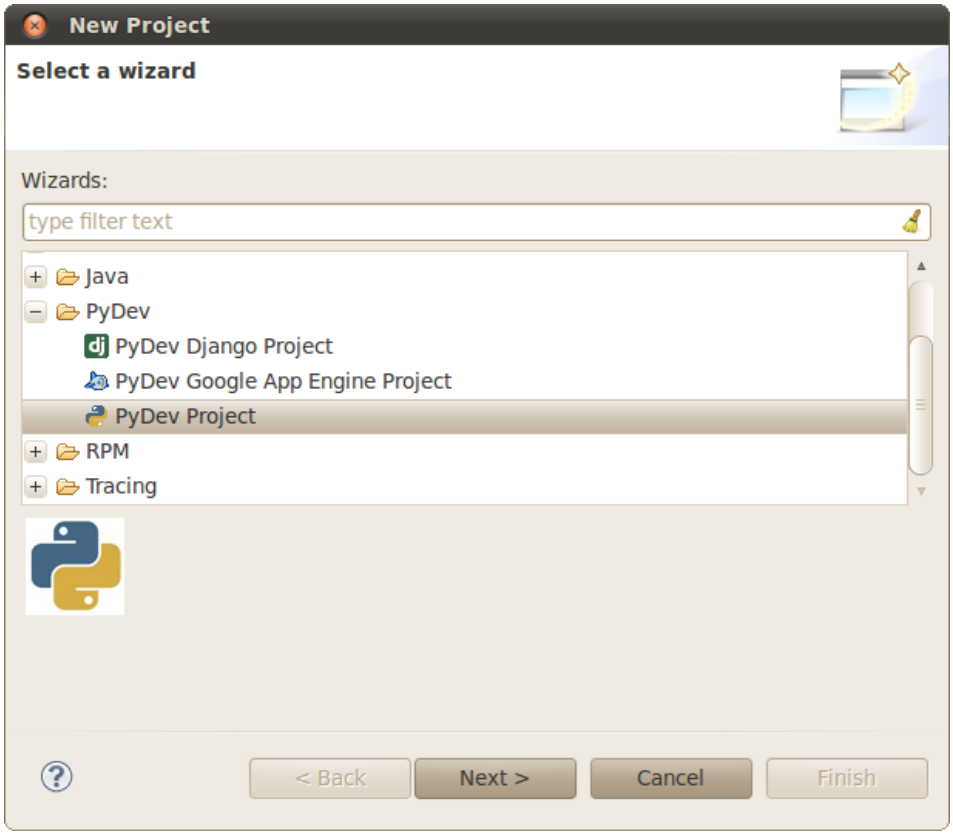


Всі подальші роботи з інсталяції пакунку Eclipse виконає самостійно і, після їх завершення, відбудеться перезавантаження середовища.

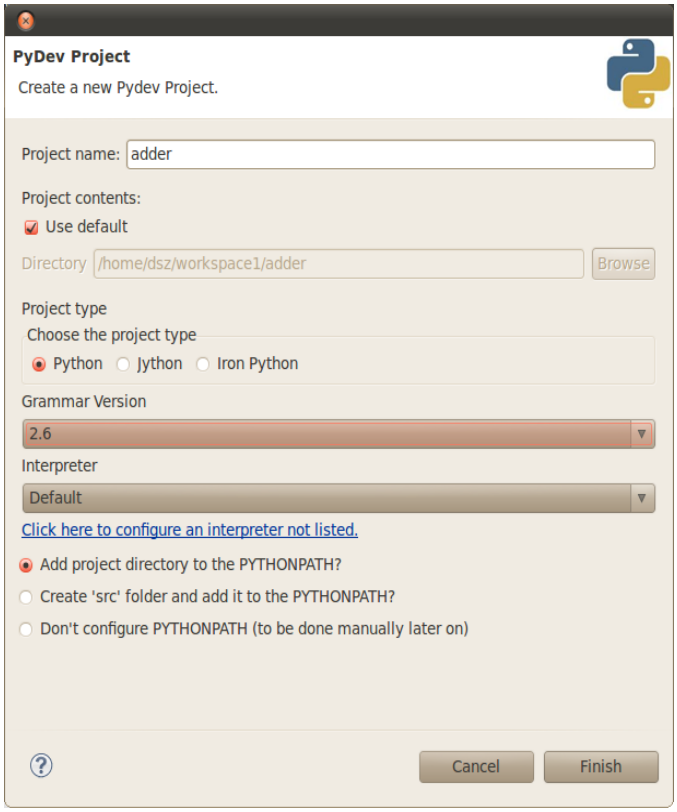
Розглянемо процес створення нового проекту в Eclipse з використанням методології розробки, орієнтованої на тестування (TDD). Для прикладу, розробимо просту програму додавання двох чисел.

Створення проекту відбувається шляхом вибору в головному меню програми пункту

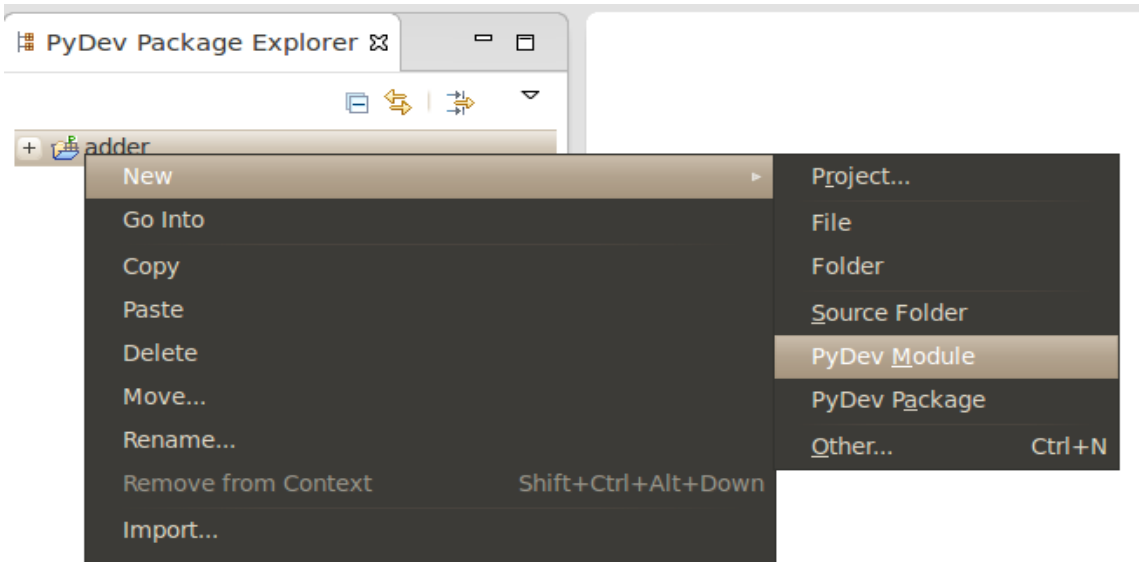
New/New Project. Після цього у вікні створення необхідно обрати відповідний тип проекту.



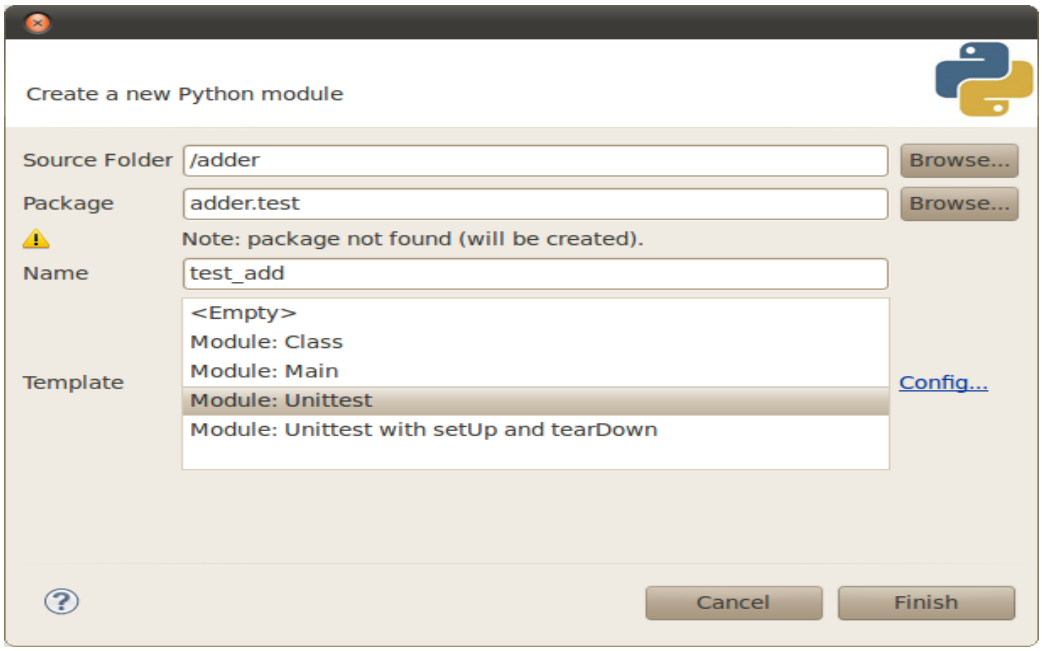
У наступному вікні буде запропоновано ввести ім'я проекту, вказати відповідну версію інтерпретатора для перевірки синтаксису. Оскільки запуск середовища відбувся перший раз, пакунок PyDev вимагатиме налаштувати параметри інтерпретатора.



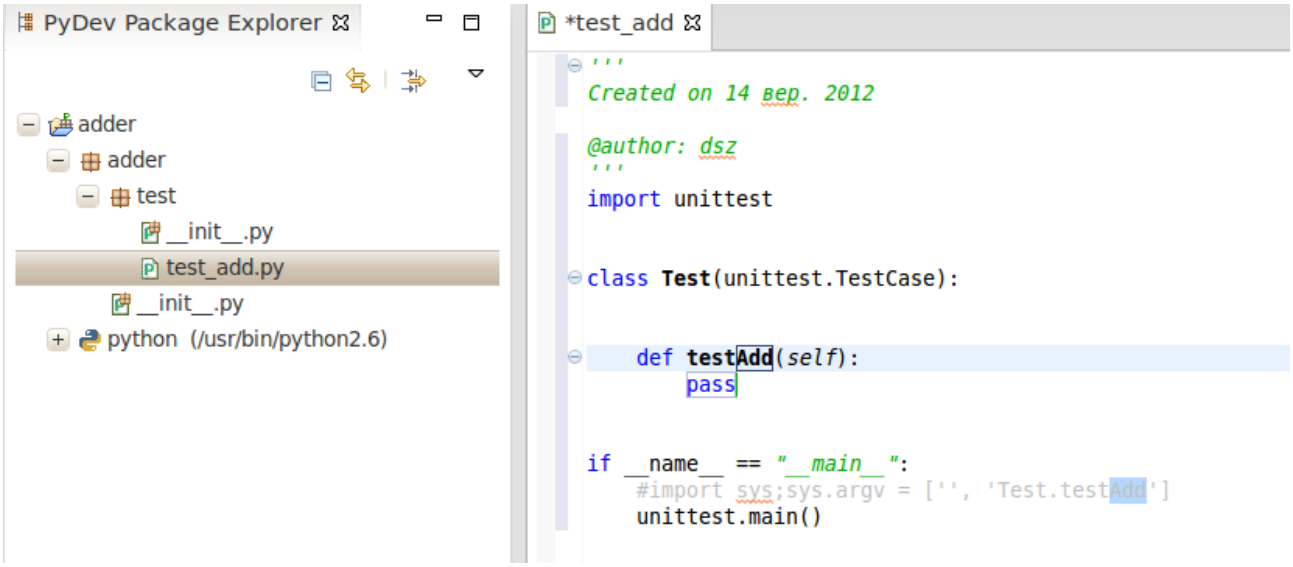
Для виконання налаштування потрібно натиснути на посилання “Please configure an iterpreter ...”. В більшості випадків у вікні конфігурування інтерпретатора достатньо натиснути кнопку “Auto Config”.



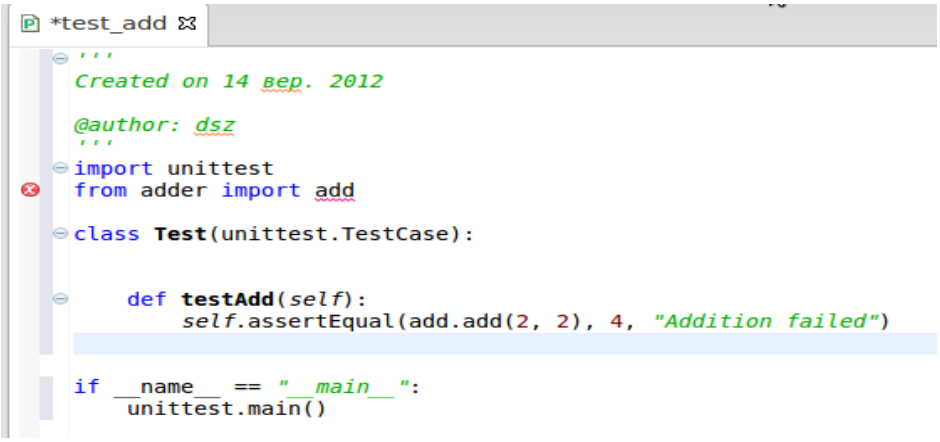
В результаті створення нового проекту в лівій частині екрану у закладці “PyDev Package Explorer” з'явиться рядок з назвою проекту. Почнемо розробку з реалізації тесту. Для цього створімо новий модуль, який буде вміщувати тестовий клас. Натиснувши правою кнопкою миші на назву проекту, у контекстному меню обираємо New/PyDev Module.



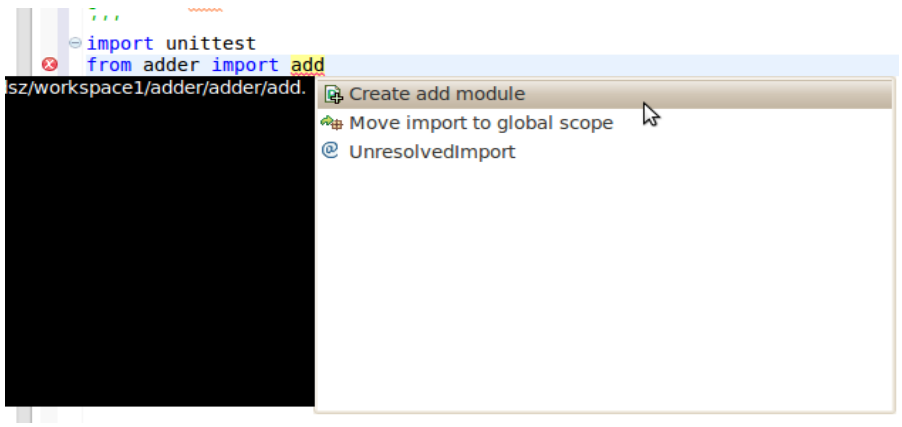
Тестовий клас найкраще всього розмітити в окремому пакеті. Для цього, у вікні створення нового модуля в полі “Package” вводимо “adder.test”, крім цього, встановлюємо ім'я модуля та обираємо шаблон Unittest для прискорення процесу написання тесту.



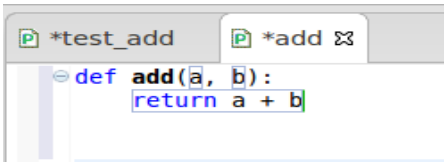
Виконання означених дій призводить до появи нового файлу у “PyDev Package Explorer”, вміст якого формується з вищевказаного шаблону. Напишемо тест для функції додавання двох чисел.



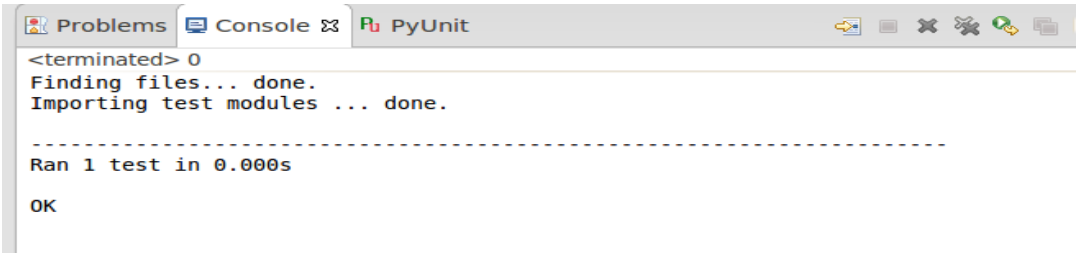
В тексті тестового модулю середовище розробки підкреслює назву модуля додавання add, оскільки ми його ще не створили. Для спрощення процесу написання коду, можна встановити курсор на підкреслене слово та натиснути <Ctrl + 1>.

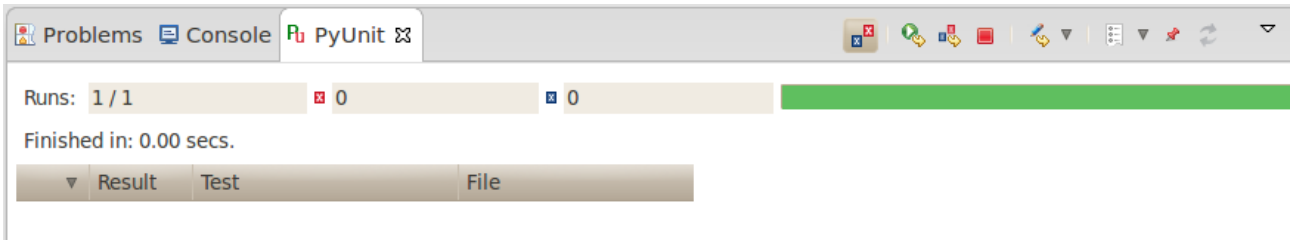


Створивши новий модуль, аналогічно за допомогою клавіш <Ctrl + 1> створюється нова функція add() у цьому модулі і можна написати її реалізацію.



Для покращення форматування вихідного коду можна скористатися функцією автоформатування <Shift + Ctrl + F>. Тепер код програми готовий для запуску. Для його виконання необхідно зберегти і модуль тесту, і модуль, який тестується. Далі у закладці “PyDev Package Explorer” потрібно правою кнопкою миші обрати модуль test і в контекстному меню натиснути Run As/Python Unit Test. Результат тестування буде відображено у нижній частині екрану у закладках “Console” та “PyUnit”. Зелена смуга у останній буде свідчити про правильний результат.





Після отримання зеленої смуги можна виконати рефакторинг написаного коду, відправити його у репозиторій системи контрою версій та почати реалізацію наступного тесту.

Завдання на лабораторну роботу

Лабораторна робота виконується бригадами по 4-5 осіб. Кожна бригада обирає предметну галузь та набір даних з цієї предметної галузі. Набір даних обирається з наявних у вільному доступі в мережі Internet, наприклад з ресурсів:

http://archive.ics.uci.edu/ml/

http://aws.amazon.com/datasets?_

http://crawdad.org/ або поібних. Розмір набору даних повинен бути не менше за 100 Мб.

В процесі виконання лабораторної роботи, кожна бригада повинна розробити програмні засоби з графічним інтерфейсом користувача на основі патерну MVC мовою програмування Python. Засоби повинні надавати наступні можливості:

1. Ідентифікація та автентифікація користувача шляхом введення логіну і паролю.
2. Посторінкове відображення вмісту набору даних.
3. Створення нових записів у наборі.
4. Редагування записів у наборі.
5. Вилучення записів з набору.
6. Прискорений пошук за певним ключем у наборі за допомогою попереднього індексування даних.

Лабораторна робота виконується без використання СУБД. Вихідні коди повинні відповідати конвенціям оформлення програмного коду PEP8

(http://www.python.org/dev/peps/pep-0008/), засоби статичного контролю pylint не повинні вказувати на помилки. Покриття коду тестами повинно складати не менше 80%, для всіх модулів, класів та функцій повинні бути створені відповідні документуючі коментарі. Всі інтерфейсні текстові повідомлення повинні виводитись за допомогою модулю gettext. Вихідні коди розміщуються на довільному репозиторії у мережі Internet.

Результатом виконання лабораторної роботи повинні бути працездатні програмні засоби та звіт, який складається з титульного аркуша; вступу, в якому коротко описується предметна галузь та містяться посилання на джерело набору даних та репозиторій; UML-діаграми класів розроблених програмних засобів; автоматично згенерованого на основі документуючих коментарів опису програмних одиниць. Лабораторна робота виконується до 1 жовтня, після чого вихідні коди передаються наступній бригаді.

Список літератури

1. Бек, К. Экстремальное программирование: разработка через тестирование [Текст] / Бек К. — СПб. : Питер, 2003. — 244 с. — ISBN 5-8046-0051-6.

2. Гамма, Э. Приемы объектно-ориентированного проектирования. Паттерны проектирования [Текст] / Гамма Э., Хелм Р., Джонсон Р. [та ін.] — СПб. : Питер, 2005. — 366 с. — ISBN 978-5-469-01136-1.

3. Рамбо, Дж. UML 2.0. Объектно-ориентированное моделирование и разработка [Текст] / Дж. Рамбо, М. Блаха. — СПб. : Питер, 2007. — (2-е изд.). — 544 с. : ил. — ISBN 5-469-00814-2.

4. Фаулер, М. Рефакторинг: улучшение существующего кода [Текст] / Фаулер М.. — СПб. : Символ-Плюс, 2003. — 432 с. — ISBN 5-93286-045-6.

5. Г. Россум, Ф.Л.Дж.Дрейк, Д.С. Откидач Язык программирования Python. http://python.ru.