CSU0020
程式語言結構
Final Project

Instructor
Chi–Jen Wu
Goal

• The student can gain the knowledge of the object operation for the Java programming language.
  – Class
  – Interface
• We will show the Java programming quickly
What you can learn from this project?

There are lots of ways to stuff objects into a collection. Array, Stack, List, Hashtable

Each has its own advantages and tradeoffs. But at some point your client is going to want to iterate over those objects, and when he does, are you going to show him your implementation?

We certainly hope not! That just wouldn’t be professional. Well, you don’t have to risk your career;

You’re going to see how you can allow your clients to iterate through the objects without ever getting a peek at how you store the objects.
Scenario

Breaking News: *Objectville Diner* and *Objectville Pancake House* Merge

That’s great news! Now we can get those delicious pancake breakfasts at the Pancake House and those yummy lunches at the Diner all in one place. But, there seems to be a slight problem …
They want to use my *Pancake House* menu as the breakfast menu and the *Diner’s menu* as the lunch menu. We’ve agreed on an implementation for the menu items…

… but we can’t agree on how to implement our menus. That joker over there used an *ArrayList* to hold his menu items, and I used an *Array*. Neither one of us is willing to change our implementations… We just have too much code written that depends on them.
Problems

• Suppose we are required to print every item on both menus
• Two loops will be needed instead of one
• If a third restaurant is included in the merger, three loops will be needed, right?
• Design principles that would be violated
  – Coding to implementation rather than interface
  – The program implementing the joint print_menu() needs to know the internal structure of the collection of each set of menu items
  – Duplication of code
Solution

• Encapsulate what varies, i.e. encapsulate the iteration
• An iterator is used for this purpose
• The DinerMenu class and the PancakeMenu class need to implement a method called `createIterator()`
• The Iterator is used to iterate through each collection without knowing its type
  – Array or ArrayList
Step 1

- Understand and compile the listed Java program
  - The `MenuItem` class is used to save the properties of a menu item.

- Then, write a Java program with a `public static void main(String[] args)` method to test the class.
Step 2

- Implement `PancakeHouseMenu` class and `DinerMenu` class
- There is a problem.
  - *Diner and Pancake House have different menu implementations.*
The Pancake House menu use a list to contain their menu items: (You can use the java.util.ArrayList class or other linked-list-like structure written by yourself)

```java
import java.util.ArrayList;

public class PancakeHouseMenu {
    private ArrayList menuItems;
    //... other fields

    public PancakeHouseMenu() {
        menuItems = new ArrayList();

        addItem("K&B's Pancake Breakfast", "Pancakes with scrambled eggs, and toast", true, 2.99);
        addItem("Regular Pancake Breakfast", "Pancakes with fried eggs, sausage", false, 2.99);
        addItem("Blueberry Pancakes", "Pancakes made with fresh blueberries", true, 3.49);
        addItem("Waffles", "Waffles with your choice of blueberries or strawberries", true, 3.59);
    }

    public void addItem(String name, String description, boolean vegetarian, double price) {
        //...
    }

    public ArrayList getMenuItems() {
        return menuItems;
    }
    //... other methods
}
```

Then, write a Java program with a `public static void main(String[] args)` method to test the class.
And the Dinner menu use an **array** to contain their menu items:

```java
public class DinerMenu {
    public static final int MAX_ITEMS = 6;
    private MenuItem[] menuItems;
    //... other fields

    public DinerMenu() {
        menuItems = new MenuItem[MAX_ITEMS];

        addItem("Vegetarian BLT", "(Fakin') Bacon with lettuce & tomato on whole wheat", true, 2.99);
        addItem("BLT", "Bacon with lettuce & tomato on whole wheat", false, 2.99);
        addItem("Soup of the day", "Soup of the day, with a side of potato salad", false, 3.29);
        addItem("Hotdog", "A hot dog, with saurkraut, relish, onions, topped with cheese", false, 3.05);
        addItem("Steamed Veggies and Brown Rice", "Steamed vegetables over brown rice", true, 3.99);
        addItem("Pasta", "Spaghetti with Marinara Sauce, and a slice of sourdough bread", false, 3.89);
    }

    public void addItem(String name, String description, boolean vegetarian, double price) {
        //...
    }

    public MenuItem[] getMenuItems() {
        return menuItems;
    }
    //... other methods
}
```

Then, write a Java program with a
```
public static void main(String[] args)
```
method to test the class.
Step 3

• Now the menus are ok. And it's time to provide a waitress for listing menu items.
• The waitress must implements our Waitress interface to be a real waitress:
Step 3(Cont’d)

- Using the following interface `Waitress` to implement the `WaitressVersion1` class:

```java
public interface Waitress {

    /** prints every item on the menu */
    public void printMenu();

    /** prints just breakfast items */
    public void printBreakfastMenu();

    /** prints just lunch items */
    public void printLunchMenu();

    /** prints all vegetarian menu items */
    public void printVegetarianMenu();

    /**
     * given the name of an item, returns true if the items is vegetarian,
     * otherwise, returns false
     */
    public boolean isItemVegetarian(String name);
}
```
The first waitress you hire will look like:

```java
public class WaitressVersion1 implements Waitress {
    private PancakeHouseMenu pancakeHouseMenu;
    private DinerMenu dinerMenu;
    // ... other fields

    public WaitressVersion1(PancakeHouseMenu pancakeHouseMenu, DinerMenu dinerMenu) {
        this.pancakeHouseMenu = pancakeHouseMenu;
        this.dinerMenu = dinerMenu;
    }

    public void printMenu() {
        // use getMenuItems() in each menu for printing
    }

    public void printBreakfastMenu() {
        // ...
    }

    public void printLunchMenu() {
        // ...
    }

    public void printVegetarianMenu() {
        // ...
    }

    public boolean isItemVegetarian(String name) {
        // ...
    }

    // ... other methods
}
```
You can use our `MenuTestDrive` to test your waitress:

```java
public class MenuTestDrive {
    public static void main(String[] args) {
        PancakeHouseMenu pancakeHouseMenu = new PancakeHouseMenu();
        DinerMenu dinerMenu = new DinerMenu();

        Waitress waitress = new WaitressVersion1(pancakeHouseMenu, dinerMenu);
        waitress.printMenu();
        // other testing
    }
}
```

And the output should be:

```

---

BREAKFAST

K&B's Pancake Breakfast(v), 2.99
    -- Pancakes with scrambled eggs, and toast
Regular Pancake Breakfast, 2.99
    -- Pancakes with fried eggs, sausage
Blueberry Pancakes(v), 3.49
    -- Pancakes made with fresh blueberries
Waffles(v), 3.59
    -- Waffles, with your choice of blueberries or strawberries

LUNCH

Vegetarian BLT(v), 2.99
    -- (Fakin') Bacon with lettuce & tomato on whole wheat
BLT, 2.99
    -- Bacon with lettuce & tomato on whole wheat
Soup of the day, 3.29
    -- Soup of the day, with a side of potato salad
Hotdog, 3.05
    -- A hot dog, with saurkraut, relish, onions, topped with cheese
Steamed Veggies and Brown Rice(v), 3.99
    -- Steamed vegetables over brown rice
Pasta, 3.89
    -- Spaghetti with Marinara Sauce, and a slice of sourdough bread
```
Step 4

• The implementation of *WaitressVersion1* have a disadvantage.
  – That is you must know the underlying data structure with each menu to access the menu items.

• Now, we can provides a way to access the menu items of each menu sequentially without exposing its underlying representation.
  – The solution is create an *iterator* for each menu, which implements the *Iterator* interface:
Step 4 (Cont’d)

- Now, we can provides a way to access the menu items of each menu sequentially without exposing its underlying representation.
  - The solution is create an iterator for each menu, which implements the Iterator interface:

```java
public interface Iterator {
    /** tells us if there are more elements in the aggregate to iterate through */
    public boolean hasNext();

    /** returns the next object in the aggregate */
    Object next();
}
```
Add iterators for each menu

Write the iterators for PancakeHouseMenu and DinerMenu class:

```java
public class PancakeHouseMenuIterator implements Iterator {
    // ... fields
    public boolean hasNext() {
        // ...
    }
    public Object next() {
        // ...
    }
}

public class DinerMenuIterator implements Iterator {
    // ... fields
    public boolean hasNext() {
        // ...
    }
    public Object next() {
        // ...
    }
}
```
And add the `createIterator()` method to `PancakeHouseMenu` and `DinerMenu` class:

```java
public class PancakeHouseMenu {
    //...

    public Iterator createIterator() {
        //...
    }
    //...
}

public class DinerMenu {
    //...

    public Iterator createIterator() {
        //...
    }
    //...
}
```
When you implement the iterators for each menu
You can hire a smarter waitress called `WaitressVersion2`:

```java
public class WaitressVersion2 implements Waitress {
    private PancakeHouseMenu pancakeHouseMenu;
    private DinerMenu dinerMenu;
    //... other fields

    public WaitressVersion2(PancakeHouseMenu pancakeHouseMenu, DinerMenu dinerMenu) {
        this.pancakeHouseMenu = pancakeHouseMenu;
        this.dinerMenu = dinerMenu;
    }

    public void printMenu() {
        Iterator pancakeIterator = pancakeHouseMenu.createIterator();
        Iterator dinerIterator = dinerMenu.createIterator();
        System.out.println("MENU\n----\nBREAKFAST");
        printMenu(pancakeIterator);
        System.out.println("LUNCH");
        printMenu(dinerIterator);
    }

    public void printMenu(Iterator iterator) {
        while (iterator.hasNext()) {
            MenuItem menuItem = (MenuItem) iterator.next();
            menuItem.print();
        }
    }

    //... other methods
}
```
Test your second waitress

Modify our MenuTestDrive to test your waitress:
And you will see the same output.

```java
public class MenuTestDrive {
    public static void main(String[] args) {
        PancakeHouseMenu pancakeHouseMenu = new PancakeHouseMenu();
        DinerMenu dinerMenu = new DinerMenu();

        Waitress waitress = new WaitressVersion2(pancakeHouseMenu, dinerMenu);
        waitress.printMenu();
        // other testing
    }
}
```
If you have any question about Java programming.

- You can refer the java.sun.com
- Ask the class TA at R107 of our department
Project Report

• The student should prepare a report which contains at least the follows:
  – The source codes and execution results in each step.
    • If you have your own test data, you can show it.
  – What you have learned and experienced during the implementation..
    • E.g. You could show your daily record of the implementation.
  – In case you implement more than the required specification, please itemize it.
  – Copyright Claim
    • Do you make the implementation yourself?
  – Any thing you would like to let CJ know.
    • E.g. Suggestion, …

• Who will be reading the report?
  – Not TAs but CJ
How to hand in your report?

• Please send a mail to TA Yi-Chan Kao (dogkevin@hotmail.com) with a zip file
  – Mail title: Programming Language final project + your student id
  – Attached filename: your_student_id.zip
  – It should have the at least the following items:
    • Electronic files of your report
      – MS word and (or) pdf
    • Source codes(src or .class)
    • Your test data and the corresponding execution results.