Instituto de Computação

Técnicas de Programação Avançada

TCC-00.174

Prof.: Anselmo Montenegro

www.ic.uff.br/~anselmo

anselmo@ic.uff.br

Conteúdo: Padrões Iterator & Composite

Documento baseado no material preparado pelo Prof. Luiz André (http://www.ic.uff.br/~lapaesleme/)

Padrões de Projeto

Introdução

É possível utilizar uma variedade enorme de estruturas de dados e contêineres para armazenar objetos

A questão é: como fazer o cliente iterar uniformemente sobre tais coleções sem expor as implementações internas?

Padrões de Projeto

Exemplo de Problema

Suponha duas organizações, uma confeitaria e um restaurante, que em um dado momento são unificadas e uma única empresa

Ambas contém seus próprios menus (cardápios) que agora precisam ser unificados e possivelmente customizados

Padrões de Projeto

Exemplo de Problema

Os itens dos cardápios são tratados igualmente pelas duas organizações originais:

```
public class MenuItem {
  String name;
  String description; boolean vegetarian; double price;
  public MenuItem(String name, String description, boolean vegetarian, double price){
    this.name = name;
    this.description = description;
    this.vegetarian = vegetarian;
    this.price = price;
  }

  public String getName() { return name;}

  public String getDescription() { return description;}

  public double getPrice() { return price;}

  public boolean isVegetarian() { return vegetarian;}
}
```

Padrões de Projeto

Exemplo de Problema

Entretanto os itens são estruturados de forma completamente distinta nos menus originais: a confeitaria utiliza um Array ...

```
public class DinerMenu{
static final int MAX_ITEMS = 6;
                int numberOfItems = 0;
                MenuItem[] menuItems;

                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”,true, 3.89);
                }
                public void addItem(String name, String description, boolean vegetarian, double price)
                {
                                MenuItem menuItem = new MenuItem(name, description, vegetarian, price);
                                if (numberOfItems >= MAX_ITEMS) {
                                                System.err.println("Sorry, menu is full!  Can't add item to menu");
                                } else {
                                                menuItems[numberOfItems] = menuItem;
                                                numberOfItems = numberOfItems + 1;
                                }
                }
                public String toString() {return "Objectville Diner Menu";
                }

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

Padrões de Projeto

Exemplo de Problema

... e o restaurante usa um ArrayList

```
public class PancakeHouseMenu{
                ArrayList menuItems;

                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)
                {               MenuItem menuItem = new MenuItem(name, description, vegetarian, price);
                                menuItems.add(menuItem);
                }

                public ArrayList getMenuItems() {return menuItems;
                }

                public String toString() {return "Objectville Pancake House Menu";
                }

                // other menu methods here
}
```

Padrões de Projeto

Exemplo de Problema

A fusão das empresas requer que o sistema que gerencia o negócio contenha uma classe cliente que conhece os menus originais e que disponibilize as seguintes funcionalidades:

- printMenu() – imprime todos os itens do menu unificado
- printBreakfastMenu() – imprime apenas itens da confeitaria
- printLunchMenu() – imprime itens do restaurante
- printVegetarianMenu() – imprime todos os itens vegetarianos
- isItemVegetarian(name) – verfica se um item é vegetariano

Padrões de Projeto

Exemplo de Problema

Vejamos como implementar a primeira funcionalidade: imprimir todos os itens do menu:

```
Public class Waitress{
  PancakeHouseMenu pancakeHouseMenu = new PancakeHouseMenu();
  ArrayList breakfastItems = pancakeHouseMenu.getMenuItems();
  DinerMenu dinerMenu = new DinerMenu();
  MenuItem[] lunchItems = dinerMenu.getMenuItems();

   …
  public void printMenu(){
    for (int i = 0; i < breakfastItems.size(); i++) {
      MenuItem menuItem = (MenuItem)breakfastItems.get(i);
      System.out.print(menuItem.getName() + " ");
      System.out.println(menuItem.getPrice() + " ");
      System.out.println(menuItem.getDescription());
    }

    for (int i = 0; i < lunchItems.length; i++) {
      MenuItem menuItem = lunchItems[i];
      System.out.print(menuItem.getName() + " ");
      System.out.println(menuItem.getPrice() + " ");
      System.out.println(menuItem.getDescription());
    }
}
```

Padrões de Projeto

Exemplo de Problema

Quais os problemas com essa solução?

```
public class Waitress{
  PancakeHouseMenu pancakeHouseMenu = new PancakeHouseMenu();
  ArrayList breakfastItems = pancakeHouseMenu.getMenuItems();
  DinerMenu dinerMenu = new DinerMenu();
  MenuItem[] lunchItems = dinerMenu.getMenuItems();

   …
  public void printMenu(){
    for (int i = 0; i < breakfastItems.size(); i++) {
      MenuItem menuItem = (MenuItem)breakfastItems.get(i);
      System.out.print(menuItem.getName() + " ");
      System.out.println(menuItem.getPrice() + " ");
      System.out.println(menuItem.getDescription());
    }

    for (int i = 0; i < lunchItems.length; i++) {
      MenuItem menuItem = lunchItems[i];
      System.out.print(menuItem.getName() + " ");
      System.out.println(menuItem.getPrice() + " ");
      System.out.println(menuItem.getDescription());
    }
}
```

Padrões de Projeto

Exemplo de Problema

Quais os problemas com essa solução???

Os menus originais não estão bem encapsulados: os detalhes de suas implementações estão expostos (Array e ArrayList)

São necessários dois loops para iterar pelos itens de menu

A classe Waitress depende de classes concretas (MenuItem[] e ArrayList)

A classe Waitress depende de classes concretas que possuem as mesmas interfaces (PancakeHouseMenu e DinerMenu)

Padrões de Projeto

Exemplo de Problema

Solução: verificar o que varia e encapsular (tema recorrente no curso)

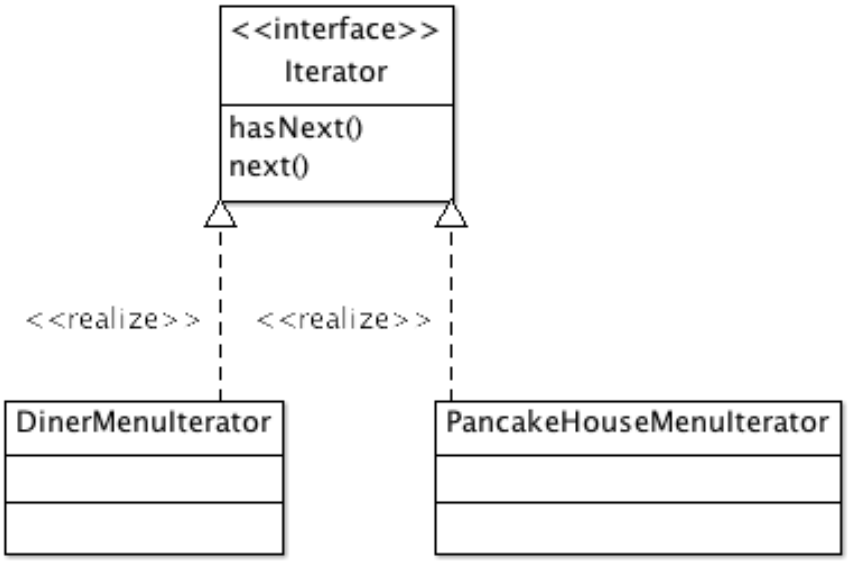
```
for (int i = 0; i < breakfastItems.size(); i++) {
      MenuItem menuItem = (MenuItem)breakfastItems.get(i);

for (int i = 0; i < lunchItems.length; i++) {
      MenuItem menuItem = lunchItems[i];
```

Obviamente, o que varia é a iteração...

Logo, vamos encapsular a iteração em uma interface Iterator

Padrões de Projeto

Exemplo de Problema



Padrões de Projeto

Exemplo de Problema

```
public interface Iterator {
  boolean hasNext();
  Object next();
}
```

```
public class DinerMenuIterator implements Iterator {
  MenuItem[] items; int position = 0;

  public DinerMenuIterator(MenuItem[] items) {
    this.items = items;
  }

  public Object next() {
    MenuItem menuItem = items[position];
    position = position + 1;
    return menuItem;
  }

  public boolean hasNext() {
    if (position >= items.length || items[position] == null) {
      return false;
    } else {
      return true;
    }
  }
}
```

Redefinindo os menus usando a interface iterator

```
public class DinerMenu{
static final int MAX_ITEMS = 6;
                int numberOfItems = 0;
                MenuItem[] menuItems;

                public DinerMenu() {
                …
                }
                public void addItem(String name, String description, boolean vegetarian, double price)
                {
                 …                      }
                public String toString() {return "Objectville Diner Menu";
                }

                public MenuItem[] getMenuItems() {return menuItems;};

                public Iterator createIterator(){
                  return new DinerMenuIterator(menuItems);
                }


                // other menu methods here
}
```

Redefinindo os menus usando a interface iterator

```
public class PancakeHouseMenu{
                ArrayList menuItems;

                public PancakeHouseMenu() {
                ...
                }

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

                public ArrayList getMenuItems() {
                …
                }

                public String toString() {return "Objectville Pancake House Menu";
                }

                public Iterator createIterator(){
                  return new DinerMenuIterator(menuItems);
                }
                // other menu methods here
}
```

Padrões de Projeto

Exemplo de Problema

```
public class Waitress {
  PancakeHouseMenu pancakeHouseMenu; DinerMenu dinerMenu;

   public Waitress(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("\nLUNCH");
     printMenu(dinerIterator);
   }

   private void printMenu(Iterator iterator) {
     while (iterator.hasNext()) {
       MenuItem menuItem = (MenuItem)iterator.next();
       System.out.print(menuItem.getName() + ", ");
       System.out.print(menuItem.getPrice() + " -- ");
       System.out.println(menuItem.getDescription());
   } // other methods here
}
```

Padrões de Projeto

Exemplo de Problema

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

    Waitress waitress = new Waitress(pancakeHouseMenu, dinerMenu);
    waitress.printMenu();
  }
}
```

Padrões de Projeto

Exemplo de Problema

Problemas:

A classe Waitress ainda continua dependendo de classes concretas...

Apesar delas possuirem interfaces idênticas...

```
public class Waitress {
  PancakeHouseMenu pancakeHouseMenu;
  DinerMenu dinerMenu;

  public Waitress(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("\nLUNCH");
    printMenu(dinerIterator);
  }

  private void printMenu(Iterator iterator) {
    while (iterator.hasNext()) {
      MenuItem menuItem = (MenuItem)iterator.next();
      System.out.print(menuItem.getName() + ", ");
      System.out.print(menuItem.getPrice() + " -- ");
      System.out.println(menuItem.getDescription());
  } // other methods here
}
```

Padrões de Projeto

Exemplo de Problema

Solução:

Criar uma abstração, isto é, uma interface Menu!

```
public class Waitress {
  Menu pancakeHouseMenu;
  Menu dinerMenu;

  public Waitress(Menu pancakeHouseMenu, Menu 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("\nLUNCH");
    printMenu(dinerIterator);
  }

  private void printMenu(Iterator iterator) {
    while (iterator.hasNext()) {
      MenuItem menuItem = (MenuItem)iterator.next();
      System.out.print(menuItem.getName() + ", ");
      System.out.print(menuItem.getPrice() + " -- ");
      System.out.println(menuItem.getDescription());
  } // other methods here
}
```

Padrões de Projeto

Exemplo de Problema

Podemos fazer melhor?

Sim, as três invocações de printMenu parecem inconvenientes…

E se adicionarmos mais menus… Teremos problemas com essa solução

```
public class Waitress {
  Menu pancakeHouseMenu;
  Menu dinerMenu;

  public Waitress(Menu pancakeHouseMenu, Menu 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("\nLUNCH");
    printMenu(dinerIterator);
  }

  private void printMenu(Iterator iterator) {
    while (iterator.hasNext()) {
      MenuItem menuItem = (MenuItem)iterator.next();
      System.out.print(menuItem.getName() + ", ");
      System.out.print(menuItem.getPrice() + " -- ");
      System.out.println(menuItem.getDescription());
  } // other methods here
}
```

Padrões de Projeto

Exemplo de Problema

Vamos armazenar um contêiner de menus e usar polimorfismo para resolver o problema.

```
public class Waitress {
   ArrayList menus;

   public Waitress(ArrayList menus) {
     this.menus = menus;
   }

   public void printMenu() {
     Iterator menuIterator = menus.iterator();
     while (menuIterator.hasNext()){
       Menu menu = (Menu)menuIterator.next();
       printMenu(menu.createIterator());
     }
   }

   private void printMenu(Iterator iterator) {
     while (iterator.hasNext()) {
       MenuItem menuItem = (MenuItem)iterator.next();
       System.out.print(menuItem.getName() + ", ");
       System.out.print(menuItem.getPrice() + " -- ");
       System.out.println(menuItem.getDescription());
   } // other methods here
}
```

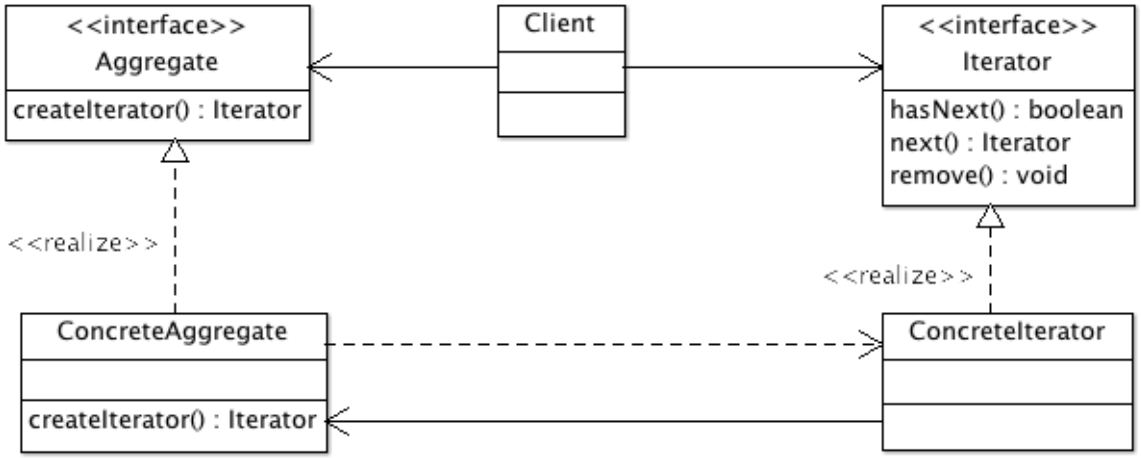
Padrões de Projeto

O Padrão Iterator

O Padrão Iterator provê um modo de acessar os elementos de um objeto agregado sequencialmente sem expor sua representação interna

Padrões de Projeto

O Padrão Iterator – Diagrama de Classes



Por que não colocar a responsabilidade de iteração dentro da classe que representa a coleção?

Porque isto fere um dos princípios mais importantes e mais difíceis de se aplicar em projeto O.O: o princípio da responsabilidade única

Padrões de Projeto

O Princípio da Resposabilidade Única

Princípio da Responsabilidade Única – uma classe deve ter apenas uma causa para mudanças. Toda responsabilidade é uma potencial fonte para mudanças e uma classe deve ter somente uma responsabilidade.

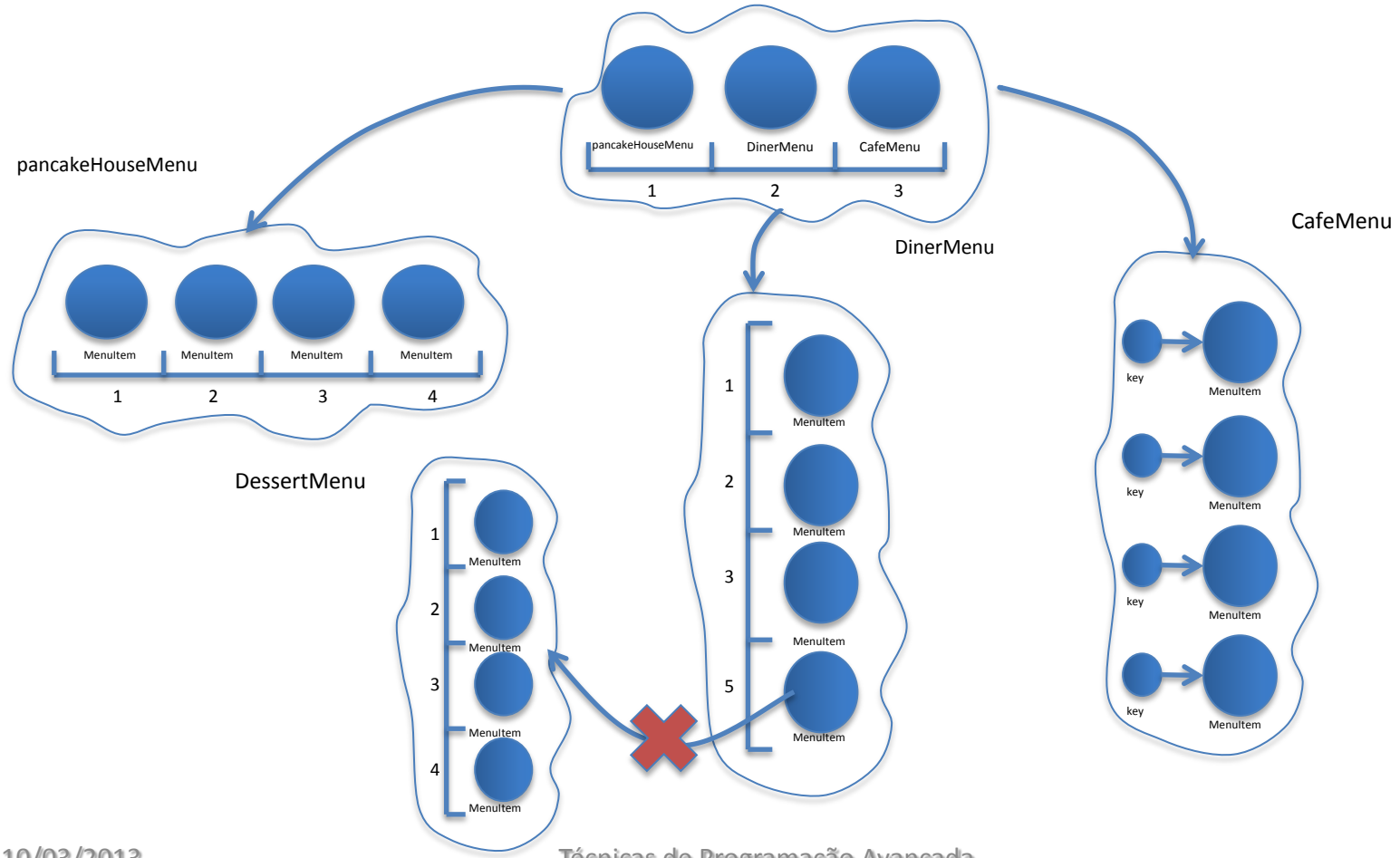
O Padrão Iterator obedece o princípio da responsabilidade única

Ele separa as operações que manipulam o agregado (collection type) da operação de iteração

Padrões de Projeto

Um novo problema

Precisamos agora criar submenus; como lidar com essa situação?



Padrões de Projeto

Um novo problema

Não podemos associar um menu a um item de menu.

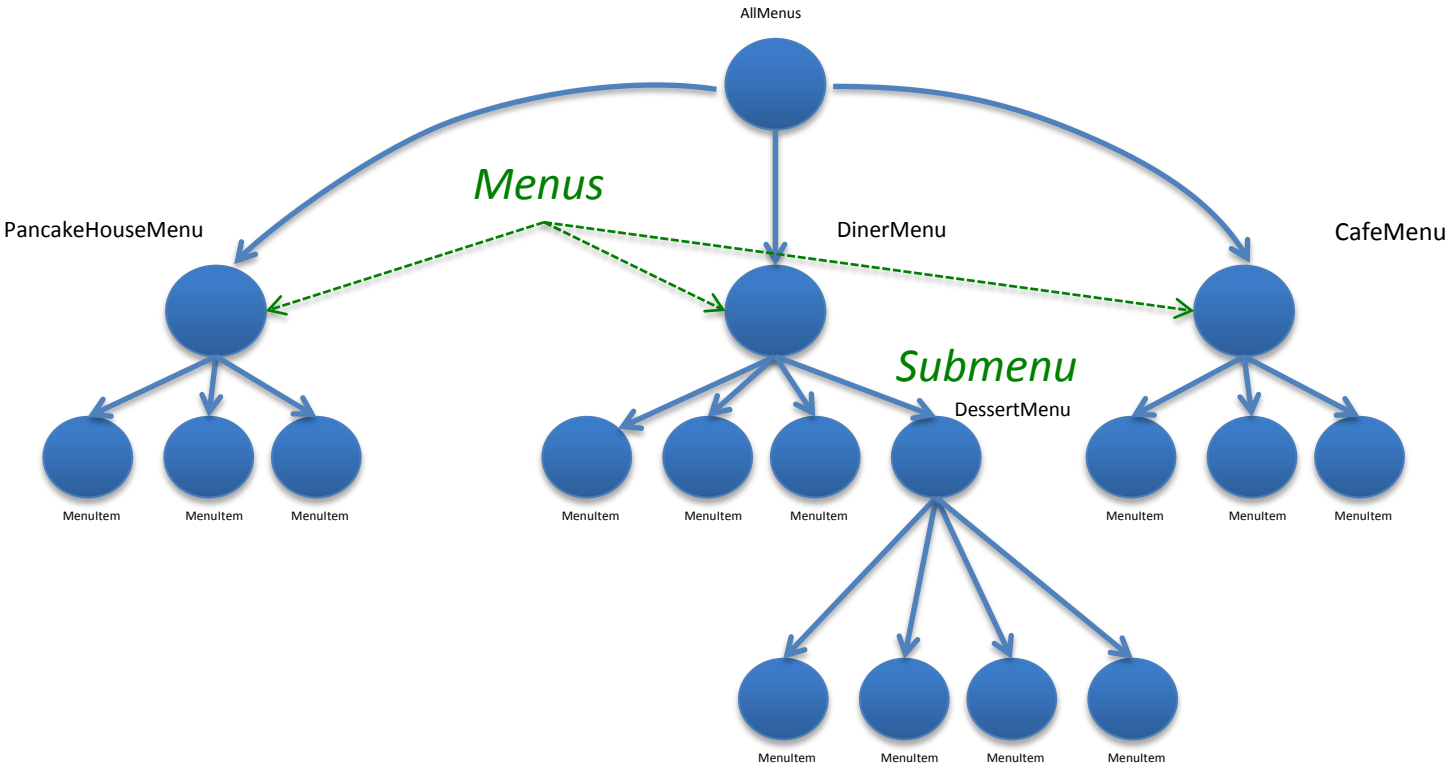
Não há muita escolha a não ser refatorar o código.

Precisamos de uma estrutura que represente a noção de hierarquia que emerge naturalmente do problema.

Iremos utilizar uma estrutura de árvore

Padrões de Projeto

Estruturas hierárquicas



Surge uma questão: como tratar uniformemente todos os elementos na estrutura, quer sejam individuais (itens de menu) ou agregações (menus e submenus)?

Introduziremos uma abstração de componente: a interface Component com as seguintes operações:

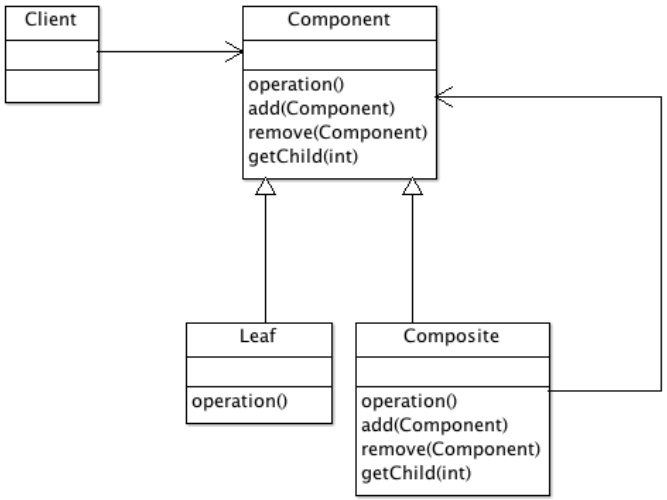
```
addComponent(Component c);
removeComponent(Component c);
getChild(int i) ;
```

Os elementos *Component* da hierarquia podem ser compostos (Composite) ou elementos individuais (Leafs)

Em outras palavras, tanto os elementos individuais (Leafs), quanto os elementos compostos (Composite) serão subtipos de Component.

Padrões de Projeto

Padrão Composite



Padrões de Projeto

O Padrão Composite

O Padrão Composite permite compor objetos através de uma estrutura de árvore que representa hierárquias que caracterizam relações todo-parte. Composite permite que objetos individuais e composições sejam tratados uniformemente.

Para tratamento uniforme da estrutura, todos os componentes devem implementar a interface em Component.

Entretanto, visto que as folhas e nós (Composite) têm diferentes papéis, nem sempre é possível definir uma implementação default para cada método.

Para elementos do tipo Leaf, as seguintes operações não fazem sentido:

addComponent(...)
removeComponent(...)
getChild(...)

Parece que estamos violando algum princípio de projeto importante...

Infelizmente, o Padrão Composite, sacrifica o princípio da responsabilidade única por transparência, o que torna o código menos seguro.

É impossível distinguir em tempo de compilação folhas de nós, entretanto é exatamente isso que buscamos.

Como vamos lidar com comportamentos diferentes de folhas e nós?

Uma solução é considerar *Component* como uma classe abstrata e fornecer métodos default que lançam exceções

Padrões de Projeto

O Padrão Iterator

```
public abstract class MenuComponent {

  public void add(MenuComponent menuComponent) {
    throw new UnsupportedOperationException();
  }
   public void remove(MenuComponent menuComponent) {
    throw new UnsupportedOperationException();
  }
  public MenuComponent getChild(int i) {
    throw new UnsupportedOperationException();
  }
  public String getName() {
     throw new UnsupportedOperationException();
  }
  public String getDescription() {
    throw new UnsupportedOperationException();
  }
   public double getPrice() {
    throw new UnsupportedOperationException();
  }
  public boolean isVegetarian() {
    throw new UnsupportedOperationException();
  }
  public void print() {
     throw new UnsupportedOperationException();
  }
}
```

Padrões de Projeto

O Padrão Iterator

```
public class Menu extends MenuComponent {
  ArrayList menuComponents = new ArrayList();
  String name; String description;

  public Menu(String name, String description) {
    this.name = name; this.description = description;
  }
  public void add(MenuComponent menuComponent{
    menuComponents.add(menuComponent);
  }
  public void remove(MenuComponent menuComponent) {
    menuComponents.remove(menuComponent);
  }
  public MenuComponent getChild(int i) {
    return (MenuComponent)menuComponents.get(i);
  }
  public String getName() { return name;}

  public String getDescription() { return description;}


   ....
```

```
public void print() {
    System.out.print("\n" + getName());
    System.out.println(", " + getDescription());
    System.out.println("----------------------");

    Iterator iterator = menuComponents.iterator();
    while (iterator.hasNext()) {
      MenuComponent menuComponent = (MenuComponent)
      iterator.next();
      menuComponent.print();
    }
  }
}
```

```
public class Waitress {
  MenuComponent allMenus;
  public Waitress(MenuComponent allMenus) {
    this.allMenus = allMenus;
  }

  public void printMenu() {
    allMenus.print();
  }
}
```

Padrões de Projeto

O Padrão Iterator

```
public class MenuTestDrive {
  public static void main(String args[]) {

  MenuComponent pancakeHouseMenu = new Menu("PANCAKE HOUSE MENU", "Breakfast");
  MenuComponent dinerMenu = new  Menu("DINER MENU", "Lunch");
  MenuComponent cafeMenu = new  Menu("CAFE MENU", "Dinner");
  MenuComponent dessertMenu = new Menu("DESSERT MENU", "Dessert of course!");
   MenuComponent allMenus = new Menu("ALL MENUS", "All menus combined");

   allMenus.add(pancakeHouseMenu);
   allMenus.add(dinerMenu);
   allMenus.add(cafeMenu);

   // add menu items here
   dinerMenu.add(new MenuItem( "Pasta","Spaghetti with Marinara Sauce, and a slice of sourdough bread",true, 3.89));

   dinerMenu.add(dessertMenu);

   dessertMenu.add(new MenuItem( "Apple Pie","Apple pie with a flakey crust, topped with vanilla icecream", true,1.59));
    // add more menu items here
   Waitress waitress = new Waitress(allMenus);
    waitress.printMenu();
}
```

Padrões de Projeto

Exercício

Proponha uma classe CompositeIterator que implemente iterador e que trabalhe sobre uma estrutura que segue o Padrão Composite.

Padrões de Projeto

Exercício

```
public class Menu extends MenuComponent {
  Iterator iterator = null;
 // other code here doesn't change
  public Iterator createIterator() {
     if (iterator == null) {
         iterator = new CompositeIterator(menuComponents.iterator());
     }
    return iterator;
}

public class MenuItem extends MenuComponent {
  // other code here doesn't change
   public Iterator createIterator() {
     return new NullIterator();
  }
}
```

```
public class CompositeIterator implements Iterator {
  Stack stack = new Stack();

  public CompositeIterator(Iterator iterator) {
    stack.push(iterator);
  }
  public Object next() {
    if (hasNext()) {
      Iterator iterator = (Iterator) stack.peek();
      MenuComponent component = (MenuComponent) iterator.next();
      if (component instanceof Menu) {
        stack.push(component.createIterator());
      }
     return component;
    } else {
      return null;
    }
  }
  public boolean hasNext() {
    if (stack.empty()) {
      return false;
    }else{
      Iterator iterator = (Iterator) stack.peek();
      if (!iterator.hasNext()) {
        stack.pop();
        return hasNext(); }
     else {return true;
      }
    }
  }
  public void remove() { throw new UnsupportedOperationException();
  }
}
```

Padrões de Projeto

Bibliografia

- Use a Cabeça ! Padrões de Projetos (design Patterns) - 2ª Ed. Elisabeth Freeman e Eric Freeman. Editora: Alta Books

- Padroes de Projeto – Soluções reutilizáveis de software orientado a objetos. Erich Gamma, Richard Helm, Ralph Johnson. Editora Bookman