IPS Instituto Politécnico de Setúbal
Escola Superior de Tecnologia de Setúbal

Programação Avançada

Ano Letivo de 2019/20

28/01/2019

Exame época normal

GRUPO 1

1) Considere o código da Figura 1, indique quais os padrões de desenho que estão envolvidos.

A. Data Access Object & Factory Method

B. Data Access Object & SimpleFactory (DocumentAccess – DAO Interface;

C. Strategy & Singleton

D. Data Access Object & Singleton

---

2) A classe DocumentTree foi desenhada para conter a estrutura hierárquica de um documento. A classe DocumentTreeView é responsável por desenhar a estrutura hierárquica do documento. Foram identificadas três diferentes formas de desenhar a estrutura da árvore (condensada, horizontalmente, verticalmente), mas espera-se que no futuro outras alternativas sejam disponibilizadas. Pertende-se que seja permitido ao utilizador alternar entre os vários tipos de visualização. **Qual o padrão mais indicado, para implementar a variação do desenho da hierarquia do documento?**

A. AbstractFactory

B. MVC

C. Strategy (porque se pretende fazer variar o comportamento em tempo de execução)

D. Template Method

---

3) O professor Bruno disponibilizou uma classe TreeSpecialView para desenhar árvores genéricas. Mas o projeto solicita o desenho de árvores binárias de pesquisa. Para tal decidiu-se aplicar um padrão de forma a simplificar o trabalho e aproveitar o trabalho desenvolvido pelo professor Bruno. **Qual a opção lhe parece mais adequada aplicar?**

A. Usar o padrão Adapter, criando a classe BSTViewAdapter. (Usa-se a TreeSpecialView como adaptee)

B. Usar o polimorfismo, criando uma subclasse de TreeSpecialView.

C. Usar o padrão Strategy criando uma estratégia concreta para desenho de Binary SearchTree.

D. Usar o padrão TemplateMethod definindo uma classe Abstrata com os métodos comuns a classe dada TreeSpecialView e a nova classe BSTSpecialView.

---

4) Considere o seguinte excerto de código da figura 3, que se refere à utilização do padrão Factory Method. Qual a afirmação correta?

A. A classe Loja assume o papel de Creator e a classe Pizza o papel de ConcreteProduct.

B. A classe Loja assume o papel de Concrete Creator e a classe Pizza o papel de Concrete Product.

C. A classe LojaAmerica assume o papel de Creator e a classe Pizza o papel de Product.

D. Nenhuma das anteriores. (Loja-Creator, LojaAmerica-ConcreteCreator,Pizza-Product)

---

5) Considere o pseudocódigo do algoritmo da Figura 4, qual o propósito deste?

A. Escreve todos os elementos da árvore usando a estratégia pos-order

B. Escreve todos os elementos usando que não são folhas da árvore, usando a estratégia in-order.

C. Escreve todos os elementos da árvore começando na raiz e terminado no nó mais à direita.

D. Nenhuma das anteriores (Imprime usando o algoritmo in-order)

---

6) Considere o Código em JAVA da implementação da classe privada treeNode da Figura 5. Qual o conjunto de instruções que completa corretamente o código.

A. **A** – TreeNode(element,null,null); **B**- return(left==null || right==null);
B. **A** – this(element,null,null); **B**- return(left==null && right==null && element!=root);
C. **A** – this(element,null,null); **B**- return(left!=null && right!=null && this!=root);
D. **A** – ***Nenhuma das anteriores***. *((return(left!=null || right!=null) && this!=root);*

---

7) Considere o excerto de código da Figura 7. Qual a técnica de refatoring que deve ser aplicada para corrigir o BAD SMELL apresentado ?

A. Form Template Method
B. Replace Conditional with Polymorphsism (Criar subclasses Cicle, Square e Rectangle e definir em cada um delas o método getArea)
C. Replace Type Code with Strategy
D. Replace Delegation with Inheritance.

---

8) Considere o excerto de código apresentado na Figura 8. Qual a técnica de refatoring que deve ser aplicada para corrigir o BAD SMELL Data Class apresentado ?

A. Remover a classe Product
B. Não fazer nada, porque é um BAD SMELL que não representa perigo.
C. Aplicar a técnica Inline Class da classe Product para a Classe Item.
D. Adicionar o método ToString à classe Produto.

---

9) Considera a tabela abaixo que indique o resultado do método Disjktra aplicada a um grafo com 6 vértices, a partir do vértice B. Selecione a opção correta.

A. A é um vértice isolado e o menor caminho para ir de D a E tem um custo de 4.
B. A é um vértice indeterminado e o menor caminho para ir de B a F tem um custo de 20.
C. A é um vértice isolado e o menor caminho para ir de B a D é {B,E,D} (A é isolado porque não existe caminho de B para A e existe caminho de B para os restantes 5 vertices; Para chegar a D partindo de B, o melhor caminho é vindo de E. Do vértice Para ir de B a E o melhor caminho é sair de B. Logo o melhor caminho é B, E,D)
D. Nenhuma das anteriores

A	**∞**	**null**
B	**0**	**null**
C	**8**	**D**
D	**4**	**E**
E	**2**	**B**
F	**20**	**B**

:

---

10) Considera que na classe UniversityNetwork, é definido o atributo **network** que é do tipo **Graph<Person, Relationship>**. Indique qual dos conjuntos de instruções completa corretamente o código, para o metodoX (ver figura 6) retornar a pessoa mais popular da rede.

A. for (Vertex<Person> v : network.vertices()){
int n = ((Collection)) network.incidentEdges(v).size();

B. while(((Collection) (network.vertices()).isEmpty()){
int n = ((Collection) network.incidentEdges(v)).size();

C. for (Edge<Relationship,Person> edge : edges(){
int n = edge.degree();

D. Nenhuma das anteriores

GRUPO 2

1. Complete a implementação do método (ver figura 2 para especificação da TAD Graph) **public boolean isPathBetween(Graph<V,E> graph, Vertex<V> start, Vertex<V> end, List<Edge<E,V>> path)** que verifica se a lista de arestas em path liga os vértices **start** e **end** no grafo graph.

```
public boolean isPathBetween(Graph<V,E> graph,
                             Vertex<V> start, Vertex<V> end,
                             List<Edge<E,V>> path) {
  /* Validations are given */
  if(start == null || end == null || start == end)
    throw IllegalArgumentException("invalid vertices.");
  if(path == null || path.size() == 0)
    throw IllegalArgumentException("invalid path.");



  if( path.isEmpty() ) return false;

  Vertex<V> current = start;
  for(int i=0; i<path.size(); i++) {
    current = graph.opposite(current, path.get(i));

    if(current == null) return false; //no path possible

    if( current == end && path.size() == (i+1) ) return true;
  }

  return false;

}
```

2. Na classe BSTImplementation (ver figura 9) implemente o método **public double mean()** que devolve a média dos elementos presentes da árvore; assuma que possui o método **public int size()** implementado que devolve o número de elementos presentes na árvore.

```
public double mean() {

            if(isEmpty()) throw new EmptyContainerException();

      return sum(root)/size();

    }


    private int sum(TreeNode root){

       if(root==null) return 0;

       return root.element.value() +sum(root.left)+sum(root.right);

    }
```

3. Considere o código Bag of Integers Problem da figura 10. Complete o código por forma a obter uma implementação do padrão Memento com o objetivo de uma funcionalidade multi-undo sobre o estado de uma instância de BagOfIntegers.

```
public Memento createMemento() {
    return new Memento(bag,name);




  }
public void setMemento(Memento state) {


            bag=((MyMemento)state).bag;

           name==((MyMemento)state).name;

  }


  private class MyMemento implements Memento {
       private ArrayList<String> bag;

        private String name;

   public MyMemento(ArrayList<String> bag, String name){

          this.bag= new ArrayList(bag);

          this.name=name;

}

  }

}


public class CareTaker {
  private final BagOfIntegers bag;

  private final Stack<Memento> stack;

  public CareTaker(BagOfIntegers bag) { this.bag = bag;
    stack= new Stack();

 }

  public void saveState() {

      stack.push(bag.createMemento());

}
```

```
  public void restoreState(){

  if(!stack.empty())

      bag.setMemento(stack.pop());



}

}
}
```

4. Considere o código **Roll the Dice App** da figura 11. Reescreva o código das classes Dice, DiceView e MiddleLayer por forma a aplicar corretamente o padrão **MVC** e a obter uma aplicação funcional.

```
public class Dice extends Observable{

  private int currentValue = 1;

  public void roll() {
    currentValue = (int)(Math.random() * 6)+1;
    setChanged();
    notifyObservers();
  }

    public int getCurrentValue() {
        return currentValue;
    }

}
public class DiceView  extends VBox  implements Observer{
  private Label diceValue;
  private Dice model;
  private Button btn;

  public DiceView(Dice model) {
    this.model=model;
    initComponents();
  }

  private void initComponents() {
    Label text = new Label("Dice Value:");
    diceValue = new Label(model.getCurrentValue()+"");
    btn= new Button("roll");
    this.getChildren().addAll(text, diceValue, btn);
  }

  public void setTriggers(MiddleLayer c) {
    btn.setOnAction(e->c.doRoll() );
  }

    @Override
    public void update(Observable o, Object arg) {
     diceValue.setText(model.getCurrentValue()+"");
    }

}

public class MiddleLayer {

    private final Dice model;
    private final DiceView view;
```

```
    public MiddleLayer(Dice model, DiceView view) {
        this.model = model;
        this.view = view;
        this.model.addObserver(view);
        view.setTriggers(this);

    }

    public void doRoll() {
      model.roll();
    }

}
```

5. Para o exemplo Inventory (figura 12) apresente a solução completa final para o refactoring que julgue necessário.
   Apresente posteriormente uma listagem (**ver pag 8**) de todos os *code smells* que identificou durante esse processo (nome e snippet de código correspondente) e qual a técnica de refactoring que utilizou.

```
public class Product {{ // Extract Class

  private String name;
  private double price;

    public Product(String name, double price) {
        this.name = name;
        this.price = price;
    }

    public String getName() {
        return name;
    }

    public double getPrice() {
        return price;
    }

    public boolean compareName(String name){
            return name.equals(this.name);
    }

    void setPrice(double price) {
     this.price=price;
    }
}
public class Inventory {

    private ArrayList<Product> products; { // Extract class e relplace array to ArrayList

    public Inventory() {
        products = new ArrayList();
    }

    public boolean addProduct(String name, double price) {
        if (exists(name)) {
            return false;
        }

        products.add(new Product(name, price));
        return true;
```

```
    }

    public boolean updatePrice(String name, double price) {
        if (exists(name)) {
            return false;
        }
        for (Product p : products) {//Substitute algorithm
            if (p.compareName(name)) {{ // Extract method compareName e seguido de MoveMethod to Product
                p.setPrice(price);
                return true;
            }
        }
        return false;
    }

    public String getCheapestProduct() {
        if (products.isEmpty()) {{//Substitute algorithm
            return "None";
        }
        double min = products.get(0).getPrice();
        int cheapestIndex = 0; //define local variable
        for (int i = 0; i < products.size; i++) {
            if (products.get(i).getPrice() < min) {
                min = products.get(i).getPrice();
                cheapestIndex = i;
            }
        }
        return products.get(cheapestIndex).getName();
    }

    private boolean exists(String name) {
        for (Product p : products) {
            if (p.compareName(name)) { // Extract method compareName e seguido de MoveMethod to Product
                return true;
            }
        }
        return false;
    }

}
```

BED SMELL	Codigo	Tecnica Refatoring Aplicada
DataClumps	`String[] productNames;` `double[] productPrices;`	**Extract Class (product)**
Primitive Obsession	**Product[] products;**	**Replace Data Type**
Temporary Field	**private** `int cheapestIndex;`	**Remove Field;**
Duplicate Code	**if**`(name.compareToIgnoreCase(productNames[i] == 0))`	**Extract Method seguido de Move Method para a classe Product**

(fim do enunciado)