Mais sobre Padrões de Projeto

Fernando Magno Quintão Pereira

6 de Outubro de 2010

Questão 1

O que será impresso?

```
public class Avatar {
  public void buy(Knife k) {
    System.out.println("Avatar bought a knife");
  }
  public void buy(Sword s) {
    System.out.println("Avatar bought a sword");
  }
}
```

```
public class Knife {}
```

```
public class Sword extends Knife {}
```

```
public class SpiderAv extends Avatar {
  public void buy(Knife k) {
    System.out.println("Spider bought a knife");
  }
  public void buy(Sword s) {
    System.out.println("Spider bought a sword");
  }
}
```

```
public static void main(String args[]) {
  Avatar a1 = new Avatar();
  Avatar a2 = new SpiderAv();
  SpiderAv sa = new SpiderAv();
  Knife k = new Knife();
  Sword s = new Sword();
  Knife ks = new Sword();
  System.out.println("a1");
  a1.buy(k);
  a1.buy(s);
  System.out.println("a2");
  a2.buy(k);
  a2.buy(s);
  System.out.println("sa");
  sa.buy(k);
  sa.buy(s);
  System.out.println("???");
  a1.buy(ks);
  a2.buy(ks);
  sa.buy(ks);
}
```

Questão 2

- Quais critérios a máquina virtual usa para resolver uma chamada?
- Será que *double dispatch* faz falta?
- Como simular *double dispatch*?

Questão 3 – O que será impresso?

Qual o tipo estático de *this*?

```
public class Avatar {
  public void buy(Knife k) {
    System.out.println("Avatar bought a knife");
  }
  public void buy(Sword s) {
    System.out.println("Avatar bought a sword");
  }
}
```

```
public class SpiderAv extends Avatar {
  public void buy(Knife k) {
    System.out.println("Spider bought a knife");
  }
  public void buy(Sword s) {
    System.out.println("Spider bought a sword");
  }
}
```

```
public static void main(String args[]) {
  Avatar a1 = new Avatar();
  Avatar a2 = new SpiderAv();
  SpiderAv sa = new SpiderAv();
  Knife ks = new Sword();
  ks.isBoughtBy(a1);
  ks.isBoughtBy(a2);
  ks.isBoughtBy(sa);
}
```

```
public class Knife {
  public void isBoughtBy(Avatar a) {
    a.buy(this);
  }
}
```

```
public class Sword extends Knife {
  public void isBoughtBy(Avatar a) {
    a.buy(this);
  }
}
```

Questão 4

Desenhe o diagrama UML para as classes abaixo:

```
public class Chain extends SimpleItem {
        Chain(String desc) {
                super(500, "Chain", desc);
        }
}
```

```
public class Diamond extends SimpleItem {
        Diamond(String desc) {
                super(1000, "Diamond", desc);
        }
}
```

```
public class Ruby extends SimpleItem {
        Ruby(String desc) {
                super(800, "Ruby", desc);
        }
}
```

```
public interface Item {
  double getValue();
  int getAge();
  double getWeight();
  int getHitPoints();
  void setDamage(int damage);
  String getName();
  String getDescription();
}
```

```
public class SimpleItem implements Item {
  private int age, hitPoints;
  private double weight;
  private String name, description;
  public SimpleItem
  (int initialHP, String name, String desc){...}
  public SimpleItem() {...}
  public int getAge() {...}
  public int getHitPoints() {...}
  public double getValue() {...}
  public double getWeight() {...}
  public void setDamage(int damage) {...}
  public String getDescription() {...}
  public String getName() {...}
}
```

```
public class CompositeItem extends SimpleItem {
  private List<Item> components;
  public CompositeItem() {...}
  public CompositeItem
  (String name, String description) {...}
  @Override
  public double getWeight() {...}
  @Override
  public int getHitPoints() {...}
  @Override
  public int getAge() { ....}
  @Override
  public double getValue() {...}
  @Override
  public void setDamage(int damage) {...}
}
```

Questão 5

1. Que objeto está sendo criado abaixo?
2. Projete e implemente uma forma de imprimir a descrição de todos os items simples que fazem parte de um item composto.

```
public class Driver {
  public static void main(String[] args) {
    CompositeItem t = new CompositeItem("Treasure", "A box of jewels");
    t.addItem(new Diamond("A glittering diamond"));
    CompositeItem c1 = new CompositeItem("Necklace", "The princess's necklace");
    c1.addItem(new Chain("Chain of gold"));
    c1.addItem(new Ruby("The Tear of Blood"));
    t.addItem(c1);
    t.addItem(new Diamond("The Sparkling Star"));
    System.out.println(t);
  }
}
```

Questão 6

Em nosso jogo, *As Trilhas de Alucubaca*, existe o conceito de venda de items. Em determinadas regiões do mundo virtual vivem *compradores*. Estes são personagens que avaliam um item, e o compram pelo valor avaliado.
Implemente um programa para avaliar o valor de um item composto. Note que neste caso, não estamos chamando o método `getValue()` do item, afinal, nós temos nossa própria tabela, por sinal com preços menores para a compra, e preço maiores para a venda!

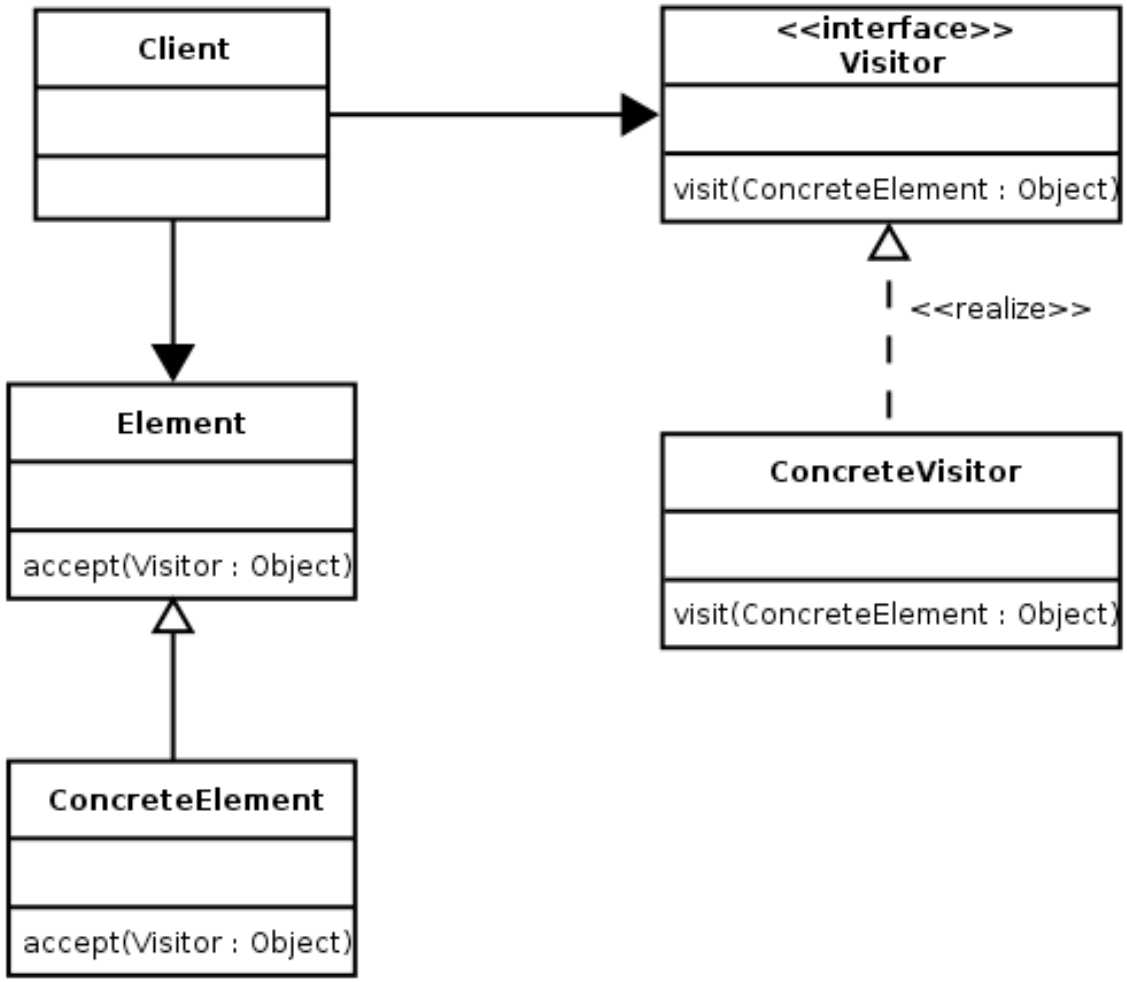
Questão 7

Você deve ter notado um padrão se formando:

- Para imprimir um item composto é necessário visitar todos os itens.
- Para computar o valor de um item composto é necessário visitar todos os itens.
- Algum outro exemplo?

Como implementar um "visitante"?

Behold "El Visitante"



Questão 8

O padrão `Visitor` separa o ato de percorrer a estrutura de dados do ato de aplicar alguma ação sobre esta estrutura de dados.

- Quem percorre a estrutura de dados?
- Quem aplica alguma aạão sobre a estrutura de dados?

Logo, há algumas perguntas fundamentais que devem ser respondidas:

1. Como implementar o código que percorre a estrutura de dados?
2. Este código deverá ser implementado de forma diferente para cada visitante diferente?
3. Como passar o visitante para o dado sobre o qual ele atuará?

Questão 9

Implemente um programa para encontrar o item mais frágil que é parte de um item composto. Este item é aquele cujo método `getHitPoints()` retorna o menor valor.

Questão 10

Como o padrão `Visitor` lida com mudanças na especificação do programa?

1. Quais partes do programa estão fechadas para o uso? Quais não estão?
2. O que é preciso ser feito se for necessário implementar um outro visitante?
3. E se eu criar um item novo, o que acontecerá?
4. Qual a alternativa para quem não quer usar `visitors`?

Questão 11

`Visitors` são muito utilizados no projeto de compiladores escritos em linguagens orientadas por objetos.

1. Onde este padrão entra, neste caso?
2. E se não quisermos usar visitors, como poderíamos proceder?
3. O que é melhor neste caso, os visitantes, ou os métodos nos itens?

Questão 12

Vamos voltar ao projeto de nosso bom e velho `Builder`.
Normalmente, nosso código conteria uma chamada assim:

```
Item treasure = (new TreasureBuilder()).getItem();
```

Note que estamos instanciando um `Builder` sempre que precisamos obter um item.

- Seria possível mover o código de instanciação de `Builder` para um objeto comum?
- Qual a vantagem disto?

Questão 13

Como garantir que o construtor de `Builder` não possa ser chamado no escopo da classe `Driver`?

Questão 14

Todo o código de instanciação de `Builder` está contido na mesma classe, porém, agora corremos o risco de termos várias instâncias de `BuilderFactory` espalhadas pelo nosso programa:

```
treasure = (new BuilderFactory()).
getBuilder("treasure").getItem();
```

Como refatorar o projeto de `BuilderFactory` para que, em qualquer momento da vida da aplicação, exista somente uma instância desta classe?