Terceira Prova de MAB 240 — Computação II

Fabio Mascarenhas

9 de Dezembro de 2016

A prova é individual e sem consulta. Responda as questões na folha de respostas, a lápis ou a caneta. Se tiver qualquer dúvida consulte o professor.

Nome: _____

DRE: _____

Questão:	1	2	3	4	5	6	Total
Pontos:	1	1	3	1	2	2	10
Nota:							

Considere as classes a seguir, que representam expressões aritméticas simples, com as quatro operações aritméticas, numerais e variáveis:

```
public interface Exp {}

public class Numeral implements Exp {
  public String valor;
}

public class Mais implements Exp {
  public Exp esq, dir;
}

public class Menos implements Exp {
  public Exp esq, dir;
}

public class Variavel implements Exp {
  public String nome;
}

public class Vezes implements Exp {
  public Exp esq, dir;
}

public class Div implements Exp {
  public Exp esq, dir;
}
```

1. (1 ponto) Escreva construtores triviais, que recebem os valores a serem atribuídos para cada campo e inicializam os campos, para as seis classes que implementam `Exp`.

O padrão *visitor* representa uma operação a ser realizada sobre os objetos que formam uma estrutura de dados heterogênea, em que cada parte da estrutura pode ser uma instância de uma classe diferente. Ele torna fácil criar novas operações em cima da mesma estrutura, deixando o código de cada operação localizado ao invés de pulverizado em diferentes classes. A interface a seguir é um exemplo do padrão `visitor` para as expressões dadas acima:

```
public interface VisitorExp<A,B> {
  B visita(A contexto, Numeral exp);
  B visita(A contexto, Variavel exp);
  B visita(A contexto, Mais exp);
  B visita(A contexto, Menos exp);
  B visita(A contexto, Vezes exp);
  B visita(A contexto, Div exp);
}
```

```
public interface Exp {
  <A,B> B executa(A contexto,
                  VisitorExp<A,B> visitor);
}
```

2. (1 ponto) A implementação do método `executa` para todas as classes que implementam `Exp` é a mesma:

   ```
   public <A,B> B executa(A contexto, VisitorExp<A,B> visitor) {
     return visitor.visita(contexto, this);
   }
   ```

   Essa implementação poderia ser uma implementação default na própria interface `Exp`, ou poderia estar em `Exp` caso `Exp` fosse uma classe abstrata ao invés de uma interface? Por quê?

3. (3 pontos) Escreva uma implementação de `VisitorExp<Map<String,Double>,Double>` chamada `CalculaExp` que calcula o valor da expressão, dado um contexto que mapeia nomes de variáveis para seus valores. Lembre-se que a interface `Map<K,V>` possui um método `V get(K chave)` para obter o valor associado a uma chave que está no mapa. Use a função `double Double.parseDouble(String s)` para converter o valor `String` de um `Numeral` no seu valor numérico.

A interface `Seq<T>` dada abaixo representa um iterador em uma sequência genérica de items. Seu método `proximo` retorna o próximo elemento e avança para o elemento seguinte enquanto ainda há elementos, caso contrário lança a exceção *checada* assinalada. A interface `Predicado<T>`, também dada abaixo, representa uma função predicado em cima de um item:

```
public interface Seq<T> {
  T proximo() throws AcabouSeq;
}
```

```
public interface Predicado<T>
  boolean testa(T item);
}
```

4. (1 ponto) Implemente a classe de exceção `AcabouSeq`.

5. (2 pontos) Escreva a classe `PA` que implementa `Seq<Integer>` e representa um iterador em uma progressão aritmética dada pelos três parâmetros de seu construtor: item inicial, o incremento (razão) e a quantidade de itens.

6. (2 pontos) Escreva a classe `FiltroSeq<T>` que implementa `Seq<T>` e filtra uma outra sequência (instância de `Seq`) de acordo com um predicado (instância de `Predicado`), ambos passados no construtor. Chamadas a `proximo` na sequência filtrada só produzem itens para o qual o predicado retorna `true`. Tanto a outra sequência quanto o predicado podem ter coringas em seus tipos para tornar a classe `FiltroSeq` mais genérica. Use esses coringas em sua implementação.

BOA SORTE!