Terceira Prova de MAB 240 2014.1 — Computação II

Fabio Mascarenhas

30 de Maio de 2014

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	Total
Pontos:	1	3	3	3	10
Nota:					

Uma *regra de negócio* é um predicado que diz se um objeto qualquer se aplica àquela regra, e pode ser combinada com outras regras usando as operações booleanas `e` e `ou`, ou transformada na sua negação. Podemos representar regras de negócio usando a classe abstrata `Regra`, parametrizada pelo tipo de objeto sobre o qual a regra se aplica:

```
public abstract class Regra<T> {
  public abstract boolean seAplica(T obj);

  public Regra<T> e(Regra<T> outra) {
    return new RegraE<T>(this, outra);
  }

  public Regra<T> ou(Regra<T> outra) {
    return new RegraOu<T>(this, outra);
  }

  public Regra<T> nao() {
    return new RegraNao<T>(this);
  }
}
```

Um *motor de regras* associa regras a ações e, dado um objeto, testa cada regra em sequência, executando a ação da primeira regra que se aplica. Motores de regras e ações seguem as interfaces abaixo:

```
public interface Motor<T> {
  void adiciona(Regra<? super T> r, Acao<? super T> a);
  void remove(Regra<? super T> r);
```

```
  void executa(T obj);
}

public interface Acao<T> {
  void executa(T obj);
}
```

1. (1 ponto) O tipo do parâmetro `outra` dos métodos `e` e `ou` de `Regra` possui um parâmetro de tipo muito restritivo. Dê ao menos uma maneira de melhorar a assinatura desses métodos para serem menos restritivos.

2. (3 pontos) Dê uma implementação concreta para a interface `Motor`. Use um `HashMap` para armazenar as regras e ações.

3. (3 pontos) Implemente as classes concretas `RegraE`, `RegraOu` e `RegraNao`, usadas na definição de `Regra`.

4. (3 pontos) Considere a classe `Cliente` abaixo, que representa um cliente em um sistema comercial:

   ```
   public class Cliente {
     public int idade; // idade em anos
     public boolean vip; // é cliente VIP?
     public String aniv; // data de aniversário (ex: "15/06")
   }
   ```

   Implemente as classes `Vip`, `Aniversariante`, e `Idoso`, que são regras de negócio que respectivamente se aplicam a clientes vip, clientes que fazem aniversário em determinado mês (passado no construtor), e clientes com mais de 60 anos. Por exemplo, uma regra de negócio que se aplica a clientes VIP que não são idosos e são aniversariantes de Maio poderia ser construída da seguinte forma:

   ```
   Regra<Cliente> vip = new Vip();
   Regra<Cliente> anivMaio = new Aniversariante("05");
   Regra<Cliente> idoso = new Idoso();
   Regra<Cliente> regra = vip.e(anivMaio).e(idoso.nao());
   ```

BOA SORTE!