Segunda Prova de MAB 240 2014.1 — Computação II

Fabio Mascarenhas

23 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	Total
Pontos:	6	4	10
Nota:			

1. Um *filtro* de linha de comando é um programa que lê linhas da entrada padrão, faz alguma espécie de processamento com elas, e escreve sua saída na saída padrão. Muitos dos aplicativos em um sistema *Linux* podem ser usados como filtros, alguns bastante complexos.

   A classe a seguir modela filtros de linha de comando simples. Uma linha da entrada pode gerar zero ou mais linhas na saída, e o tipo de processamento fica a cargo da subclasse de `Filtro`:

```
public abstract class Filtro {
  private BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
  public void filtra() {
    String linha = br.readLine();
    while(linha != null) {
      for(String s: processa(linha)) {
        System.out.println(s);
      }
      linha = br.readLine();
    }
  }
  protected abstract List<String> processa(String linha);
}
```

   (a) (2 pontos) O método `readLine` de `BufferedReader` pode lançar uma exceção de tipo `IOException`, que é uma exceção checada. Qual problema isso causa na implementação do método `filtra`, e como resolve-lo?

   (b) (2 pontos) Crie uma subclasse de `Filtro` chamada `FiltroUnico`, que elimina da saída linhas que já apareceram antes. Dica: use um `HashSet` para guardar as linhas que já foram vistas pelo filtro.

(c) (2 pontos) Crie uma subclasse abstrata de `Filtro` chamada `FiltroRecupera` que redefine `filtra` para capturar exceções que aconteçam durante a filtragem. Se a exceção for do tipo `IOException` ela deve ser passada adiante. Qualquer outra exceção deve ser passada para um novo método abstrato `erro`, que recebe a exceção e não retornada nada, e a filtragem continua. O novo método deve ser visível apenas para a classe `FiltroRecupera` e suas subclasses.

2. (4 pontos) Em aplicações com uma interface gráfica, qualquer computação que seja demorada não deve ser executada na mesma linha de execução da interface gráfica, quando a aplicação está respondendo a eventos do usuário, para não travar a interface. No sistema Android podemos usar *tarefas assíncronas* para computações demoradas. Uma tarefa assíncrona define a sua computação em um método `doInBackground`, que executa em uma linha de execução separada da interface gráfica, e também define um método `onPostExecute` que é chamado quando a tarefa terminou, na linha de execução da interface gráfica, e atualiza a interface.

```
public class TarefaDownloadFoto extends AsyncTask<String, Void, Bitmap> {
    private ImageView foto;
    public TarefaDownloadFoto(ImageView foto) {
        this.foto = foto;
    }
    protected Bitmap doInBackground(String... urls) {
      ... // baixa imagem e retorna o Bitmap
    }
    protected void onPostExecute(Bitmap bm) {
      ... // atualiza interface para mostrar imagem
    }
}
```

Um *callback* é uma forma alternativa de uma tarefa assíncrona avisar a aplicação que ela terminou. Um objeto callback tem um método que é chamado se a tarefa terminou com sucesso, e recebe o resultado, e um método chamado se ocorreu uma exceção na execução da tarefa, e é chamado com a exceção que ocorreu. A interface parametrizada abaixo descreve callbacks:

```
public interface Callback<T> {
  void sucesso(T res);
  void falha(Exception ex);
}
```

Implemente uma subclasse de `TarefaDownloadFoto` que, além de receber uma `ImageView` no construtor, recebe um callback. Se não acontecerem exceções no download ela atualiza a interface para mostrar a imagem e chama o método `sucesso` do callback com a imagem. Se ocorrer uma exceção ela chama o método `falha` do callback com a exceção. Lembre-se que o callback deve sempre ser chamado na linha de execução da interface gráfica.

BOA SORTE!