Resolução 1ª Freq. Programação II (10-04-2012)

1. Como certamente é do seu conhecimento, o SMS é um serviço de pequenas mensagens de texto (até 160 caracteres), disponível entre telemóveis. Implemente a classe SMS. Esta classe deve providenciar uma implementação para representar estas mensagens de texto, partilhadas entre um remetente (nº de telemóvel) e um destinatário(outro número de telemóvel).

```
public class SMS {

    private int remetente;
    private int destinatario;
```

Não esquecer de fechar a chaveta.

(a) Crie um construtor que dados dois números, remetente, destinatário, e um texto, cria o correspondente objecto do tipo SMS. Caso o texto tenha mais de 160 carateres deve considerar os primeiros 160.

```
private String texto; //Nova variável da classe
private final int TAMANHO_MAXIMO = 160; //Constante = 160 para o código ser mais perceptível

public SMS(int r, int d, String t) { //Construtor
    this.remetente = r;
    this.destinatario = d;
    this.texto = "";

    if (t.length() > TAMANHO_MAXIMO) { //Condição para ver quando o texto ultrapassa os 160
        this.texto = t.substring(0, TAMANHO_MAXIMO - 1); //O método substring para este caso devolve a String entre 0 e 159
    }

    else this.texto = t; //Caso o texto seja menor que 160, simplesmente atribui
}
```

(b) Apresente implementações para os métodos:

i. selectores para saber o destinatário e a mensagem;

```
public int getDestinatário() {
    return this.destinatario;
}

public String getTexto() {
    return this.texto;
}
```

ii. o método modificador que permita alterar a mensagem, com o conteúdo da String passada por parâmetro;

```
public void setTexto(String t) {
    if (t.length() > TAMANHO_MAXIMO) {
        this.texto = t.substring(0, TAMANHO_MAXIMO);
    }

    else this.texto = t;
}
```

iii. toString(), que retorne uma representação do tipo

DE: 916236532
PARA: 965297563
Tou a tua espera

```
public String toString() {
    String representacao = "DE: "+this.remetente+
            "PARA: "+this.destinatario+
            this.texto;
    return representacao;
}
```

iv. o método size() que retorna o número de carateres da mensagem

```
public int size() {
    return this.texto.length();
}
```

v. equals(Object x), que retorne true se dois SMS são iguais.

```
public boolean equals(SMS a) {
    return (this.remetente == a.remetente &&
            this.destinatario == a.destinatario &&
            this.texto == a.texto);
}
```

vi. clone() que retorna um objecto exatamente igual ao objecto original mas com outra referência.

```
public SMS clone() {
    return new SMS(this.remetente, this.destinatario, this.texto);
}
```

2. Implemente a classe Gestor Mensagens. Um gestor de mensagens é responsável pela gestão das mensagens enviadas e recebidas por um telemóvel. Usualmente, um gestor comtempla o número de telemóvel ao qual está associado e duas listas de mensagens, uma para as mensagens recebidas e outra para as mensagens enviadas. Implemente para esta classe:

```
import java.util.ArrayList;

public class Gestor_Mensagens {

    private int numero_telemovel;
    private ArrayList<SMS> enviadas;
    private ArrayList<SMS> recebidas;
```

Não esquecer mais uma vez de fechar a chaveta.

(a) Um construtor que recebe um inteiro correspondente ao número de telemóvel do gestor e inicializa as listas de mensagens: enviadas e recebidas

```
public Gestor_Mensagens(int numero, ArrayList<SMS> env, ArrayList<SMS> rec) {
    this.numero_telemovel = numero;
    this.enviadas = env;
    this.recebidas = rec;
}
```

(b) o método void receber(SMS msg) que adiciona a mensagem msg à lista das mensagens recebidas

```
public void receber(SMS msg) {
    this.recebidas.add(msg);
}
```

(c) o método void criar msg(int destino,String texto) que cria a mensagem correspondente e coloca-a na lista de mensagens enviadas

```
public void criar_msg(int destinatario, String texto) {
    SMS nova_mensagem = new SMS(this.numero_telemovel, destinatario, texto);
    this.enviadas.add(nova_mensagem);
}
```

(d) o método String ler(int n) que retorna a String correspondente à n-ésima mensagem da lista de recebidas. Caso não exista tal mensagem deve ser retornada a String vazia

```
public String ler(int n) {
    if (this.recebidas.size() > n) {
        return this.recebidas.get(n).getTexto();
    }
    else return "";
}
```

(e) o método void apagar(SMS msg) que apaga a mensagem passada por parâmetro, da lista de mensagens recebidas

```
public void apagar(SMS msg) {
    this.recebidas.remove(msg);
}
```

(f) o método void limpar_recebidas() que apaga toda a lista de mensagens recebidas

```
public void limpar_recebidas() {
    this.recebidas.clear();
}
```

3. Como sabe é possível, actualmente, adicionar uma imagem, um som, ou um vídeo às mensagens. Que alterações introduziria na definição da classe SMS para comtemplar estas novas mensagens com anexo? Considere que o anexo é uma String, correspondente ao nome do ficheiro que quer anexar.

Resposta: Para contemplar o anexo, teria de criar uma variável de classe (private String anexo) para o representar, adicioná-la ao construtor (ou criar outro construtor adicionando o anexo), adicionar os métodos get() e set() e alterar os métodos equals, clone e toString adicionando o anexo.

(a) Também as mensagens quando são recebidas são marcadas como “não lidas” na lista das mensagens recebidas, já as mensagens enviadas não têm este tipo de etiqueta, dado que não tem qualquer significado o “não lida” para este tipo de mensagens. Que alterações introduziria na classe/hierarquia de classes implementada para contemplar esta característica das mensagens recebidas?

Resposta: Poderia criar uma classe SMS_Recebida (public class SMS_Recebida extends SMS), adicionaria uma “flag” booleana para saber se a mensagem já foi lida ou não e iria fazer as alterações semelhantes ao exercício acima.