Prática de Programação J2ME (17)

Especialização em Desenvolvimento Web com Interfaces Ricas
www.especializacao.info

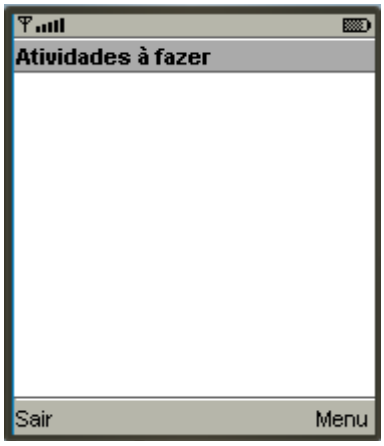
Crie a aplicação correspondente aos itens seguintes. Execute o programa e observe os resultados.

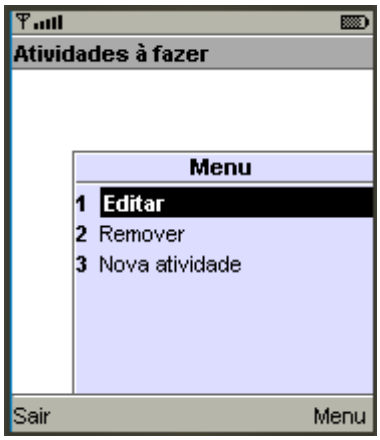
---

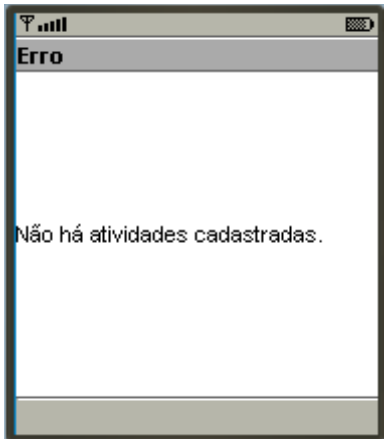
1. Crie um MIDlet simples, chamado `ActivitiesListMidlet.java`, que ao ser executado, mostra uma lista de atividades a serem feitas;
2. Cada atividade será representada pela classe `Atividade.java`, que possui dois atributos: descrição (String) e prioridade(int – de 1 a 5);
3. A lista de atividades, `AtividadesList.java`, deverá ser do tipo List.IMPLICIT, contendo as seguintes características adicionais:
   - Título deve ser “Atividades à fazer”;
   - Vector (java.util) de armazenamento de atividades existentes;
   - Quatro comandos: “Nova Atividade”, “Editar”, “Remover”(ambos do tipo Command.OK) e “Saída” (do tipo Command.BACK);
   - Ao ser iniciado, deverá carregar a lista de atividades já presentes localmente no dispositivo, mais precisamente no espaço para dados RMS (Record Management Storage) reservado à aplicação ;
4. Ao selecionar o botão “Sair”, a aplicação é encerrada;
5. Ao selecionar “Editar” ou “Remover” e não haver atividades, um alerta de erro deve ser mostrado, indicando que não há itens na lista;
6. Ao selecionar “Excluir”, a atividade é removida do vetor de AtividadesList.java e seu RecordStore também é apagado. Após essas ações, a lista de atividades é atualizada na tela;
7. Ao selecionar o botão “Editar”, abre-se um novo formulário `EditaAtividadeForm.java`, que contém um TextField para edição da descrição da atividade, um ChoiceGroup (do tipo POPUP) para edição da prioridade e dois comandos: Voltar (Command.BACK) e Salvar (Command.OK).
8. Ao selecionar o botão “Nova Atividade”, abre-se um novo formulário `NovaAtividadeForm.java`, que contém um TextField para entrada de descrição da atividade, um ChoiceGroup (do tipo POPUP) para entrada da prioridade e dois comandos: Voltar (Command.BACK) e Salvar (Command.OK).
9. Ao selecionar o comando “Voltar” nas telas de entrada e edição de nova atividade, volta-se à tela da lista de atividades existentes;
10. Ao selecionado o comando “Salvar” nas telas de entrada e edição de nova atividade, é chamada sua gravação no vetor de `AtividadesList.java e` em um RecordStore (javax.microedition.rms) privativo da aplicação, cujo nome equivalerá à sua posição (última) no vetor de atividades existentes,no caso de nova atividade, ou sua antiga posição, no caso de edição de atividade;
11. Os dados a serem armazenados no RecordStore deverão obedecer ao seguinte padrão:
    - Primeiro registro: descrição da atividade;
    - Segundo registro: prioridade da atividade.

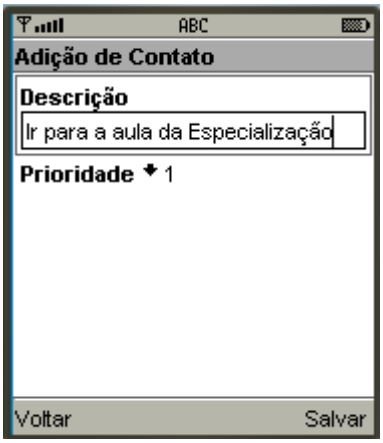
Resposta da Prática de Programação J2ME (17)

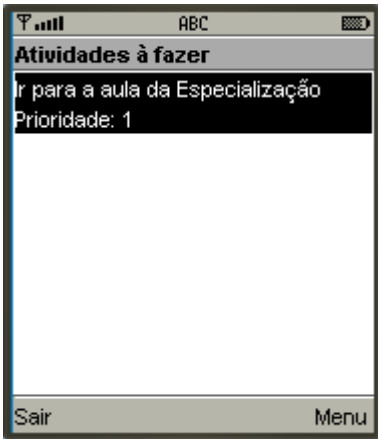
TELA(S)

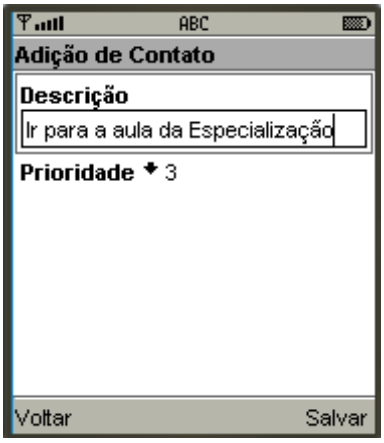


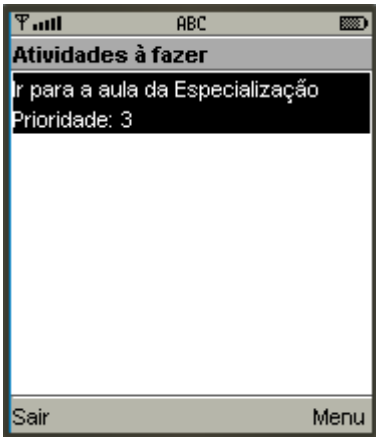












CÓDIGO EM JAVA

AtividadesStorage.java

```
package PP17.store;

import PP17.Atividade;
import java.io.*;
import java.util.Vector;
import javax.microedition.rms.*;

public class AtividadesStorage {

    public static boolean salvar(Atividade a, int num) {
        try {
            RecordStore rs = criaRecordStore(Integer.toString(num));
            add(a.getDescricao(), rs);
            add(Integer.toString(a.getPrioridade()), rs);
            rs.closeRecordStore();
            return true;
        } catch (Exception e) {
            return false;
        }
    }

    public static Vector getAtividades(){
        Vector atividades = new Vector();
        String[] numeroAtividades = RecordStore.listRecordStores();
        if(numeroAtividades != null){
            for(int i=0; i < numeroAtividades.length;i++){
                RecordStore rs;
                try {
                    rs = abreRecordStore(numeroAtividades[i]);
                    Atividade a = leituraAtividade(rs);
                    atividades.addElement(a);
                    rs.closeRecordStore();
                } catch (Exception ex) {
                    return null;
                }
            }
        }
        return atividades;
    }

    public static boolean removeAtividade(int num){
        String nomeAtividade = Integer.toString(num);
        try {
            RecordStore.deleteRecordStore(nomeAtividade);
            return true;
        } catch (Exception e){
            return false;
        }
```

```
}


private static Atividade leituraAtividade(RecordStore rs) throws
 Exception {
    String descricao = getRecord(1, rs);
    int prioridade = Integer.parseInt(getRecord(2, rs));
    Atividade a = new Atividade(descricao, prioridade);
    return a;
}

private static RecordStore criaRecordStore(String nomeRS) throws
 Exception {
    RecordStore rs = null;
    try {
        RecordStore.deleteRecordStore(nomeRS);
    } catch (RecordStoreNotFoundException e) { }
        rs = RecordStore.openRecordStore(nomeRS, true,
                RecordStore.AUTHMODE_PRIVATE, true);
    return rs;
}

private static RecordStore abreRecordStore(String nomeRS)
        throws RecordStoreFullException,
                RecordStoreNotFoundException,RecordStoreException {
    RecordStore rs = null;
    rs = RecordStore.openRecordStore(nomeRS, false);
    return rs;
}

private static void add(String texto, RecordStore rs) throws
 Exception {
    ByteArrayOutputStream baos = new ByteArrayOutputStream();
    DataOutputStream dos = new DataOutputStream(baos);
    dos.writeUTF(texto);
    dos.flush();
    byte[] data = baos.toByteArray();
    rs.addRecord(data, 0, data.length);
    baos.close();
    dos.close();
}

private static String getRecord(int id, RecordStore rs) throws
 Exception {
    String toReturn = "";
    int recordSize = rs.getRecordSize(id);
    byte[] data = new byte[recordSize];
    ByteArrayInputStream bais = new ByteArrayInputStream(data);
    DataInputStream dis = new DataInputStream(bais);
    rs.getRecord(id, data, 0);
    toReturn = dis.readUTF();
    bais.close();
    dis.close();
```

```
        return toReturn;
    }
}
```

Atividade.java

```
package PP17;

public class Atividade {

    private String descricao;
    private int prioridade;

    public Atividade(String descricao, int prioridade) {
        this.descricao = descricao;
        this.prioridade = prioridade;
    }

    public String getDescricao() {
        return descricao;
    }

    public void setDescricao(String descricao) {
        this.descricao = descricao;
    }

    public int getPrioridade() {
        return prioridade;
    }

    public void setPrioridade(int prioridade) {
        this.prioridade = prioridade;
    }
}
```

AtividadesList.java

```
package PP17;

import PP17.store.AtividadesStorage;
import java.util.Vector;
import javax.microedition.lcdui.*;

public class AtividadesList extends List implements CommandListener{

    private ActivitiesMidlet controlador;
    private Vector atividades;
    private Command exit, add, remove, edit;

    public AtividadesList(ActivitiesMidlet controlador) {
        super("Atividades à fazer", List.IMPLICIT);
        this.controlador = controlador;
```

```
        setFitPolicy(List.TEXT_WRAP_ON);
        carregaLista();
        exit = new Command("Sair", Command.EXIT, 0);
        add = new Command("Nova atividade", Command.OK, 2);
        edit = new Command("Editar", Command.OK, 0);
        remove = new Command("Remover", Command.OK, 1);
        addCommand(exit);
        addCommand(add);
        addCommand(edit);
        addCommand(remove);
        setCommandListener(this);
    }

    public void commandAction(Command arg0, Displayable arg1) {
        if (arg0 == exit) {
            controlador.destroyApp(true);
        } else {
            if (arg0 == add) {
                NovaAtividadeForm n = new NovaAtividadeForm(this);
                controlador.mudarTelaPara(n);
            }else{
                if(arg0 == remove){
                    int posicaoEscolhida = getSelectedIndex();
                    removerAtividade(posicaoEscolhida);
                }else{
                    editarAtividade();
                }
            }
        }
    }

    public ActivitiesMidlet getControlador() {
        return controlador;
    }

    public void carregaLista() {
        deleteAll();
        this.atividades = AtividadesStorage.getAtividades();
        for (int i = 0; i < atividades.size(); i++) {
            Atividade a = (Atividade) atividades.elementAt(i);
            append(a.getDescricao() + "\nPrioridade: " +
                      a.getPrioridade(), null);
        }
    }

    public Vector getAtividades() {
        return atividades;
    }


    private void editarAtividade() {
        if(getSelectedIndex() == -1){
            Alert alerta = new Alert("Erro", "Não há atividades
```

```
                                cadastradas.", null, AlertType.ERROR);
            controlador.mudarTelaPara(alerta);
        }else{
            EditaAtividadeForm m = new EditaAtividadeForm(this,
                    (Atividade)atividades.elementAt(getSelectedIndex()));
            controlador.mudarTelaPara(m);
        }
    }

    private void removerAtividade(int posicaoEscolhida) {
         if(getSelectedIndex() == -1){
            Alert alerta = new Alert("Erro", "Não há atividades"
                                    + "cadastradas.", null,
                                    AlertType.ERROR);
            controlador.mudarTelaPara(alerta);
        }else{
            AtividadesStorage.removeAtividade(posicaoEscolhida);
            atividades.removeElementAt(posicaoEscolhida);
            delete(posicaoEscolhida);
        }
    }
}
```

NovaAtividadeForm.java

```
package PP17;

import PP17.store.AtividadesStorage;
import java.util.Vector;
import javax.microedition.lcdui.*;

public class NovaAtividadeForm extends Form implements CommandListener,
Runnable {

    TextField descricao;
    ChoiceGroup prioridades;
    Command salvar, voltar;
    ActivitiesMidlet controlador;
    AtividadesList telaAnterior;

    public NovaAtividadeForm(AtividadesList anterior) {
        super("Adição de Contato");
        this.telaAnterior = anterior;
        controlador = anterior.getControlador();
        descricao = new TextField("Descrição", "", 50, TextField.ANY);
        String[] p = {"1", "2", "3", "4", "5"};
        prioridades = new ChoiceGroup("Prioridade", ChoiceGroup.POPUP, p,
                                      null);
        salvar = new Command("Salvar", Command.OK, 0);
        voltar = new Command("Voltar", Command.BACK, 0);
        addCommand(salvar);
        addCommand(voltar);
```

```
        append(descricao);
        append(prioridades);
        this.setCommandListener(this);
    }


    public void commandAction(Command arg0, Displayable arg1) {
        if (arg0 == voltar) {
            controlador.mudarTelaPara(telaAnterior);
        } else {
            new Thread(this).start();
        }
    }

    public void run() {
        String d = descricao.getString();
        int p = Integer.parseInt(prioridades.getString(
                prioridades.getSelectedIndex()));
        Atividade a = new Atividade(d, p);
        if (AtividadesStorage.salvar(a, atividades.size() - 1)) {
            telaAnterior.carregaLista();
            controlador.mudarTelaPara(telaAnterior);
        } else {
            Alert alerta = new Alert("Erro",
                    "Não foi possível salvar a atividade", null,
                    AlertType.ERROR);
            controlador.mudarTelaPara(alerta);
        }
    }
}
```

EditaAtividadeForm.java

```
package PP17;

import PP17.store.AtividadesStorage;
import java.util.Vector;
import javax.microedition.lcdui.*;

public class EditaAtividadeForm extends Form implements CommandListener,
        Runnable {

    TextField descricao;
    ChoiceGroup prioridades;
    Command salvar, voltar;
    ActivitiesMidlet controlador;
    AtividadesList telaAnterior;
    Atividade atividade;


    public EditaAtividadeForm(AtividadesList anterior, Atividade a) {
        super("Adição de Contato");
```

```
        this.telaAnterior = anterior;
        this.atividade = a;
        controlador = anterior.getControlador();
        descricao = new TextField("Descrição", a.getDescricao(), 50,
                                  TextField.ANY);
        String[] p = {"1", "2", "3", "4", "5"};
        prioridades = new ChoiceGroup("Prioridade", ChoiceGroup.POPUP, p,
                                  null);
        prioridades.setSelectedIndex(atividade.getPrioridade() - 1,
                                  true);
        salvar = new Command("Salvar", Command.OK, 0);
        voltar = new Command("Voltar", Command.BACK, 0);
        addCommand(salvar);
        addCommand(voltar);
        append(descricao);
        append(prioridades);
        this.setCommandListener(this);
    }

    public void commandAction(Command arg0, Displayable arg1) {
        if (arg0 == voltar) {
            controlador.mudarTelaPara(telaAnterior);
        } else {
            new Thread(this).start();
        }
    }

    private int getPosicaoAtividade() {
        Vector atividades = telaAnterior.getAtividades();
        for (int i = 0; i < atividades.size(); i++) {
            Atividade a = (Atividade) atividades.elementAt(i);
            if (a.getDescricao().equals(atividade.getDescricao()) &&
                    a.getPrioridade() == atividade.getPrioridade()) {
                return i;
            }
        }
        return -1;
    }

    public void run() {
        String d = descricao.getString();
        int p = Integer.parseInt(prioridades.getString(
                prioridades.getSelectedIndex()));
        Atividade a = new Atividade(d, p);

        int posicaoAtividade = getPosicaoAtividade();

        if (AtividadesStorage.salvar(a, posicaoAtividade)) {
            telaAnterior.carregaLista();
            controlador.mudarTelaPara(telaAnterior);
        } else {
            Alert alerta = new Alert("Erro",
                    "Não foi possível salvar a atividade", null,
```

```
                    AlertType.ERROR);
            controlador.mudarTelaPara(alerta);
        }
    }
}
```

ActivitiesMidlet.java

```
package PP17;

import javax.microedition.midlet.*;
import javax.microedition.lcdui.*;

public class ActivitiesMidlet extends MIDlet {

    Display display;

    public void startApp() {
        display = Display.getDisplay(this);
        AtividadesList lista = new AtividadesList(this);
        display.setCurrent(lista);
    }

    public void pauseApp() {}

    public void destroyApp(boolean unconditional) {
        notifyDestroyed();
    }

    public void mudarTelaPara(Displayable d){
        display.setCurrent(d);
    }
}
```