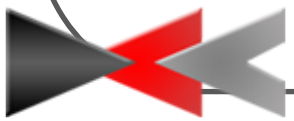
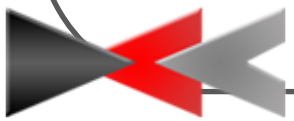
Operações Desconectadas



Conceitos

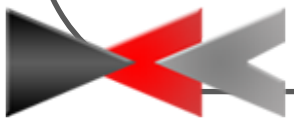
- Operações desconectadas
  - Banco de dados móveis
- Operações ocasionalmente conectadas
  - Transferências dinâmicas
  - Conexões fracas
- Operações autônomas



Desafios

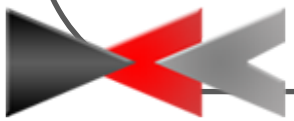
- Desconexões provocadas pela mobilidade
  - Previsíveis
  - Imprevisíveis

- Previsíveis
  - Economia de energia, custo, mudança de localização
- Imprevisíveis
  - Regiões indisponíveis, indisponibilidade de servidores, congestionamento de rede



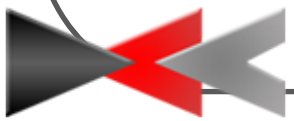
Metas

- Transparência para os usuários e aplicações
  - Ocultar as desconexões das aplicações (middleware)
  - Manter o funcionamento das aplicações sem interrupções
- Aplicação
  - Funcionalidade e dados disponíveis mesmo em momentos de desconexão
  - Preparar a disponibilidade durante os momentos de conexão



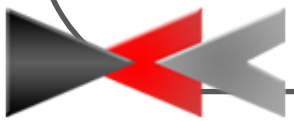
Fases

- Preparação
  - Acesso direto aos serviços
  - Cache do lado do cliente
  - Algoritmos de atualização de cache
- Desconexão
  - Acionamento dos serviços locais em detrimento aos remotos
  - Logs e erros para acesso a serviços fora do cache
  - Armazenamento
- Reintegração
  - Envio dos serviços armazenados para o servidor
  - Resolução de conflitos
  - Informações para algoritmos de atualização de cache



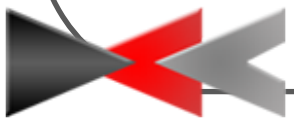
Banco de Dados

- Embarcados
  - Bancos locais de baixo consumo disponíveis no dispositivo
  - PointBase, SQLite, iAnywhere's UltraLite etc.
- Móveis
  - Bancos de baixo consumo sincronizados com o servidor
  - DB2 EveryPlace, Oracle 10g lite, Tamino Mobile etc.



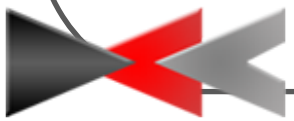
Persistência Local

Sistema de Arquivos x SQLite



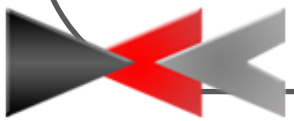
Persistência

- Dados que não desaparecem quando a aplicação termina a sua execução
  - Lista de compras
  - Lista de endereços
  - Posição das peças escolhidas em um jogo
  - Opções escolhidas em uma aplicação
  - Etc.



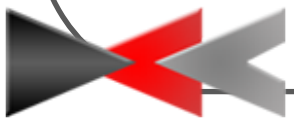
Formas de Armazenamento

- Diferentes formas, com os mais diversos objetivos:
  - API de preferências
  - Estados em “Bundles”
  - Sistema de Arquivos
  - Bases de dados
  - etc.



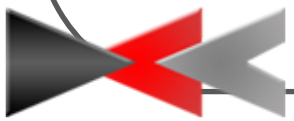
Sistema de Arquivos

- Android OS é Linux-like
  - Portanto, oferece acesso a um sistema de arquivos
- Java oferece uma biblioteca para esse objetivo
  - java.io
- Cada aplicação possui seu próprio espaço de armazenamento
  - Normalmente data/data/nome_pacote



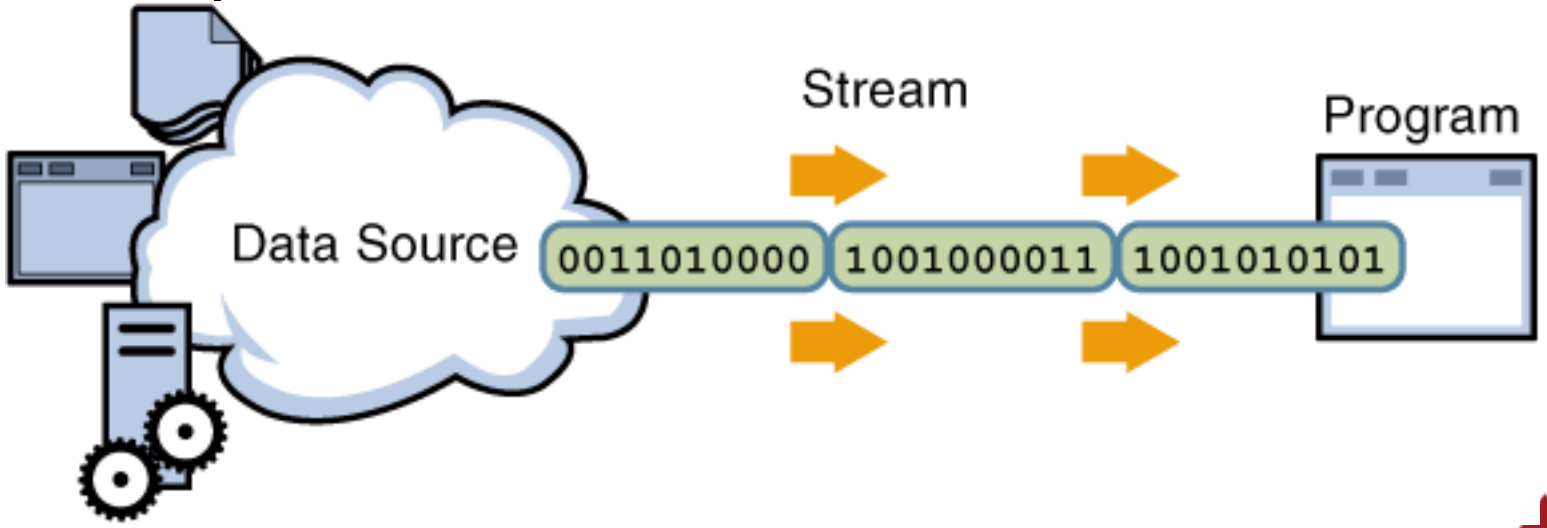
Android/Java

- Manipulação de Entrada/Saída pode usar os mesmos princípios de Java
- Unificação dos mecanismos de Entrada/ Saída
  - Fluxos/Streams



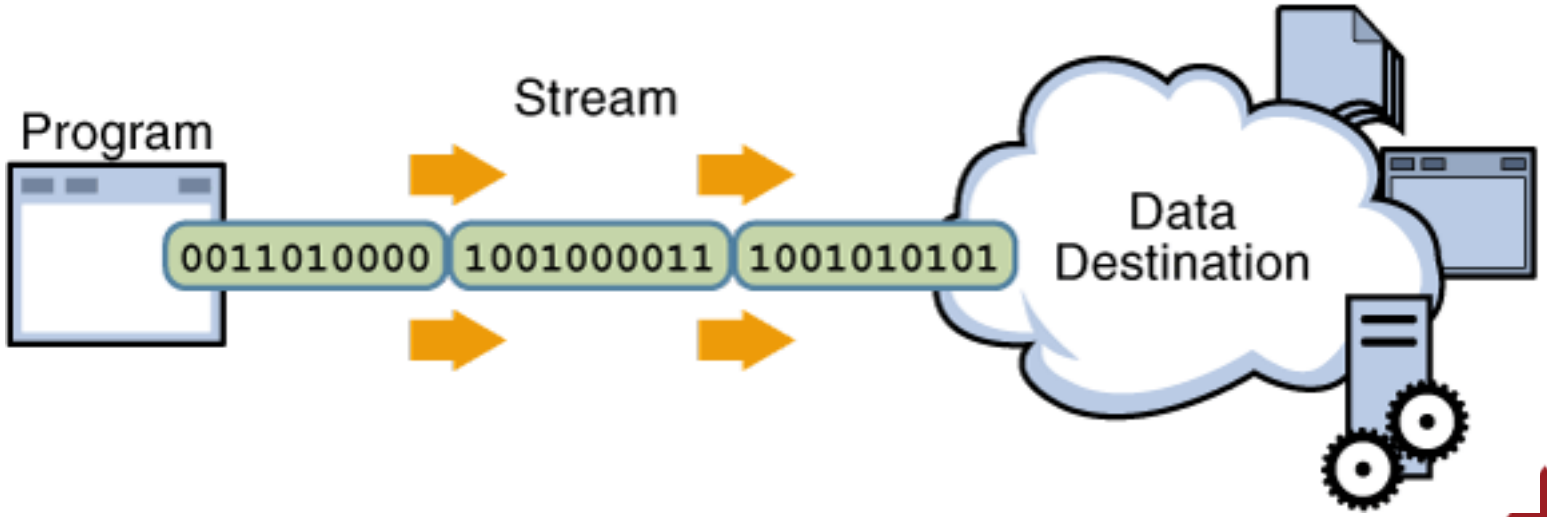
Stream de Entrada

- Para obter informações, uma aplicação abre um stream de uma fonte (arquivo, socket, memória, etc) e lê sequencialmente



Stream de Saída

- Para enviar informações, uma aplicação abre um stream para um destino (arquivo, socket, memória etc.) e escreve sequencialmente



Leitura e Escrita de Streams

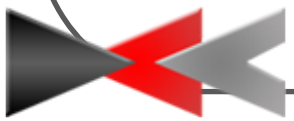
- Independente da fonte/destino e do tipo de informação, os algoritmos para leitura e escrita são basicamente os mesmos

Leitura

1. Abre um stream
2. Enquanto há informação:
   a. Lê informação
3. Fecha o stream

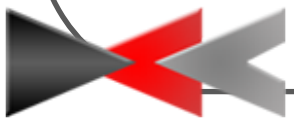
Escrita

1. Abre um stream
2. Enquanto há informação:
   a. Escreve informação
3. Fecha o stream



Pacote java.io

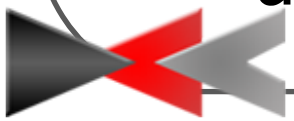
- Coleção de classes que suportam algoritmos de entrada e saída
- As classes são divididas em duas hierarquias, baseadas no tipo de dados (bytes ou caracteres) sobre as quais elas atuam
  - InputStream/OutputStream
  - Reader/Writer



java.io

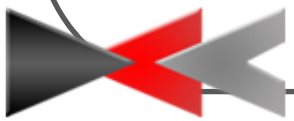
- São mais de 40 classes, divididas em:
  - Fluxos de entrada (input streams);
  - Fluxos de saída (output streams);
  - Leitores (readers);
  - Escritores (writers);
  - Arquivo de acesso aleatório (random access file).

- Classes podem indicar a mídia de I/O ou a forma de manipulação dos dados;



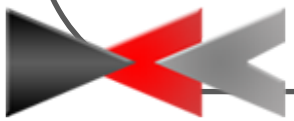
Streams de Bytes

- As classes InputStream e OutputStream são superclasses abstratas de todos os streams de bytes
  - InputStream define um método abstrado read para ler um byte de uma stream
  - OutputStream define um método abstrato write para escrever um byte em uma stream
- Suas subclasses provêem E/S especializada para cada tipo de fonte/ destino



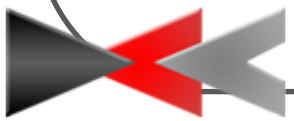
IOException

- É uma extensão da classe Exception
- Sinaliza a ocorrência de uma falha ou interrupção em uma operação de E/S
- Algumas subclasses:
  - EOFException, FileNotFoundException, InterruptedIOException, MalformedURLException, SocketException

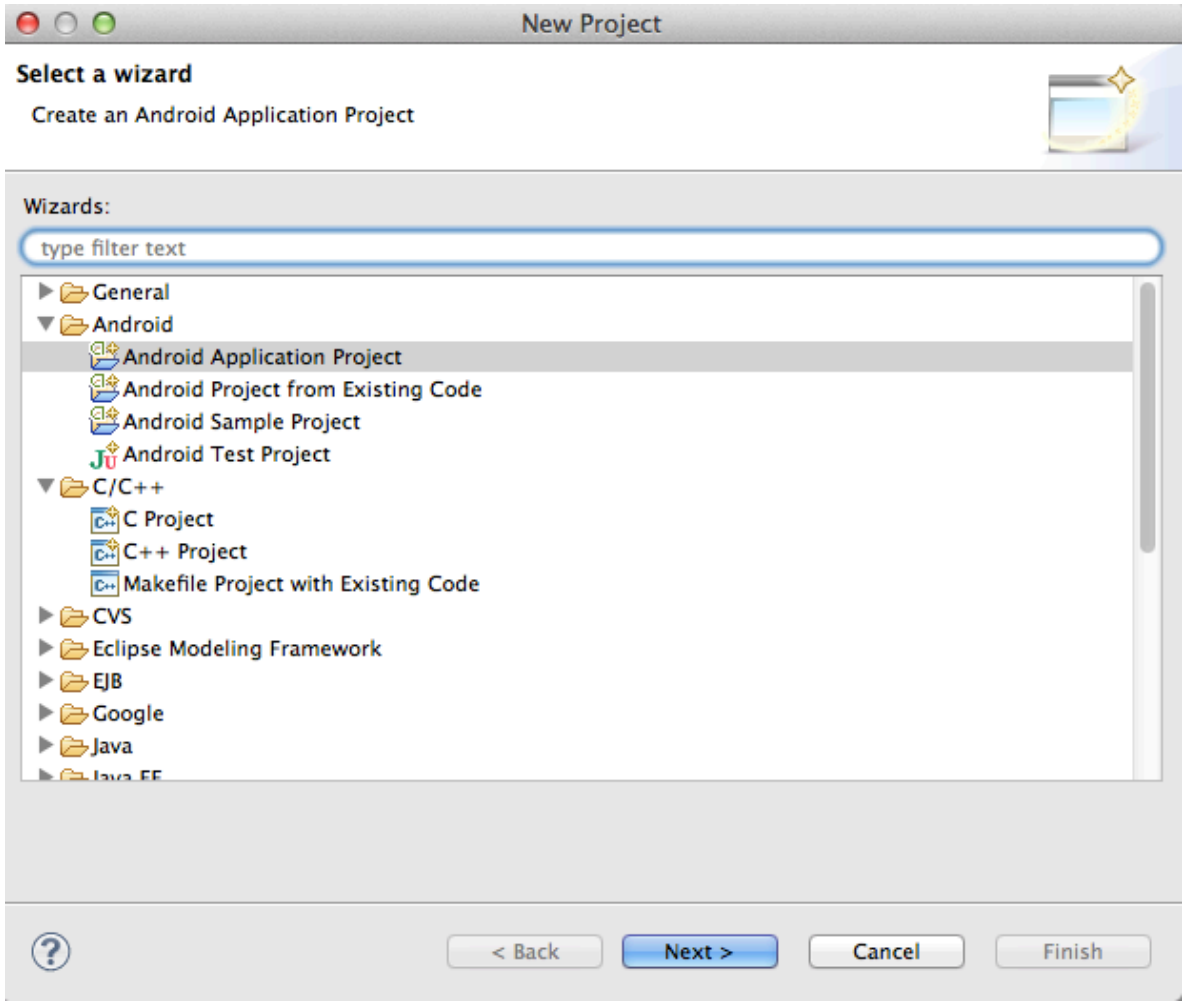


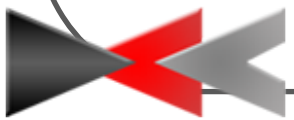
Buffered Streams

- Por default, os streams não são bufferizados
  - Essa funcionalidade pode ser obtida adicionando-se uma "camada" sobre o stream
- BufferedInputStream, BufferedOutputStream
  - Ex. public BufferedInputStream(InputStream in, int size)
- BufferedReader, BufferedWriter
  - Ex. public BufferedReader(Reader in, int size)



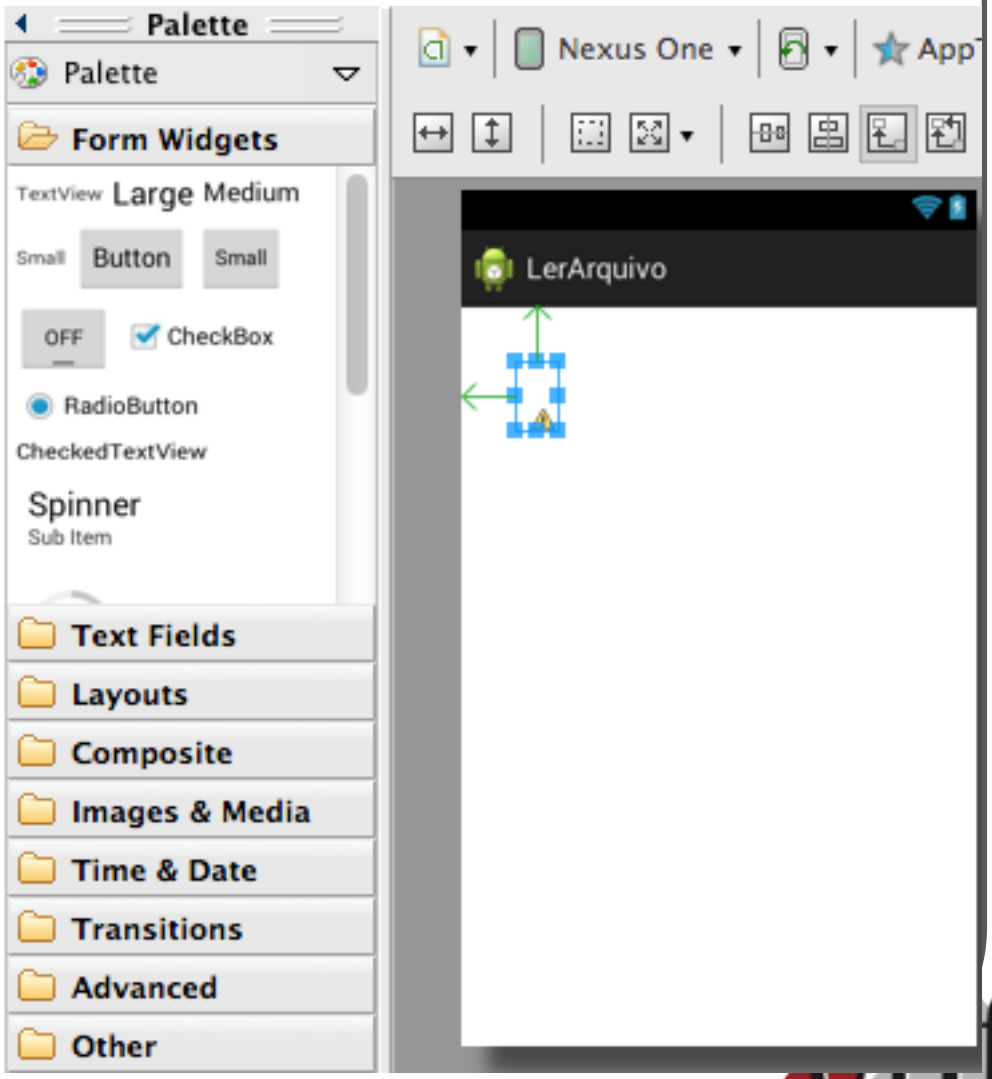
Lendo e Exibindo Arquivo





Continuando ...

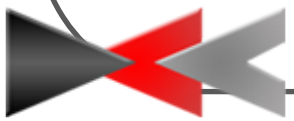
- Activity_main.xml
- Escolha um novo TextView
- Remova a propriedade text “deixe vazia”



Activity

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    TextView text = (TextView) findViewById(R.id.textView1);
    text.setText(getTextFromFile());
}
```

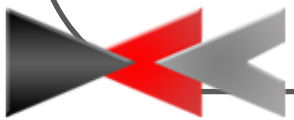
- Lembre-se de importar o pacote java.io



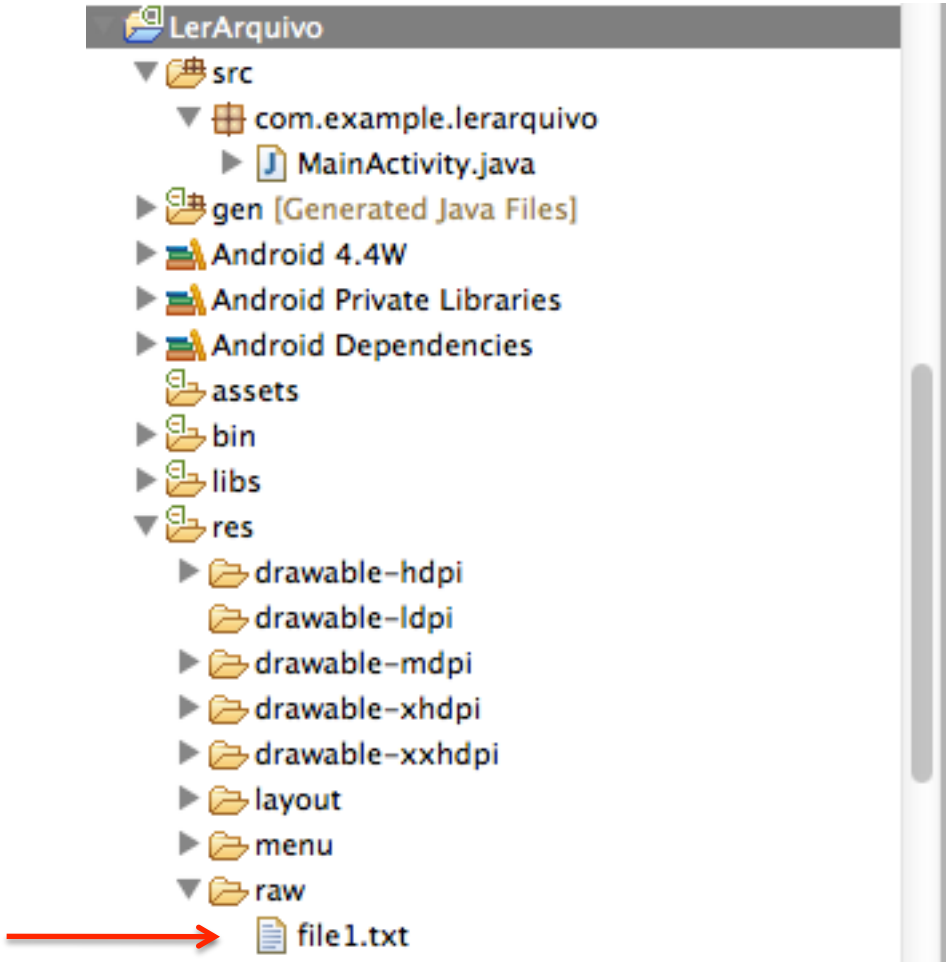
Bufferização

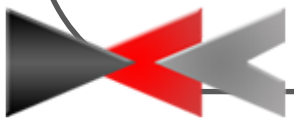
```
private String getTextFromFile() {
  StringBuffer contents = new StringBuffer();
  try {
    InputStream rawRes = getResources().openRawResource(R.raw.file1);
    BufferedReader input = new BufferedReader(new InputStreamReader(rawRes));
    String line = null;
    while ((line = input.readLine()) != null) {
      contents.append(line + '\n');
    }
  } catch (IOException ex) {
    ex.printStackTrace();
  }
  return contents.toString();
}
```

- Onde está o arquivo file1?



file1.txt

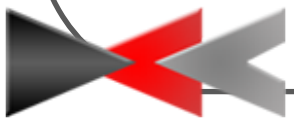

LerArquivo
src
com.example.lerarquivo
MainActivity.java
gen [Generated Java Files]
Android 4.4W
Android Private Libraries
Android Dependencies
assets
bin
libs
res
drawable-hdpi
drawable-ldpi
drawable-mdpi
drawable-xhdpi
drawable-xxhdpi
layout
menu
raw
file1.txt




ufjf

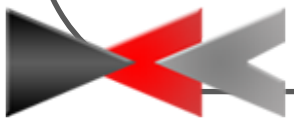
Persistência

- Quase todos os sistemas necessitam armazenar dados em memória secundária
  - Persistência
- Possibilidades
  - Diretamente em arquivos
  - Sistemas Gerenciadores de Banco de Dados



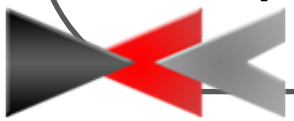
SGBD - Vantagens

- A redundância pode ser reduzida
- A inconsistência pode ser evitada
- Padrões podem ser impostos
- Restrições de segurança podem ser aplicadas
- A integridade pode ser mantida
- E os recursos?



SQLite

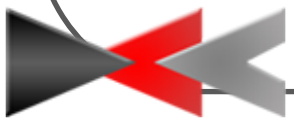
- Android disponibiliza um banco de dados chamado SQLite
- É o banco de dados padrão em várias outras aplicações: Firefox, iPhone, Symbia, Skype etc.
- Não possui licença, sendo de domínio público



SQL

- Comando para construção de tabelas:

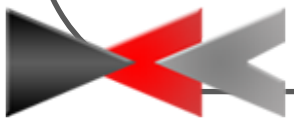
```
Create table mytable (
    id integer primary key autoincrement,
    name text,
    phone text
    );
```



SQL

- Comando para inserir dados na base:

```
insert into mytable values (null, 'Jose da Silva',
    '000-0000');
insert into mytable values (null, 'Maria das Dores',
    '111-1111');
insert into mytable values (null, 'Joaquim Jose',
    '222-2222');
```

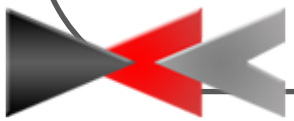


SQL

- Comando para consultar dados da base:

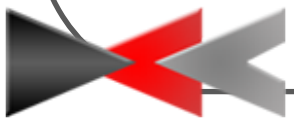
  select * from mytable where(id=3);

  select name, phone from mytable where(name like “%jose%”);

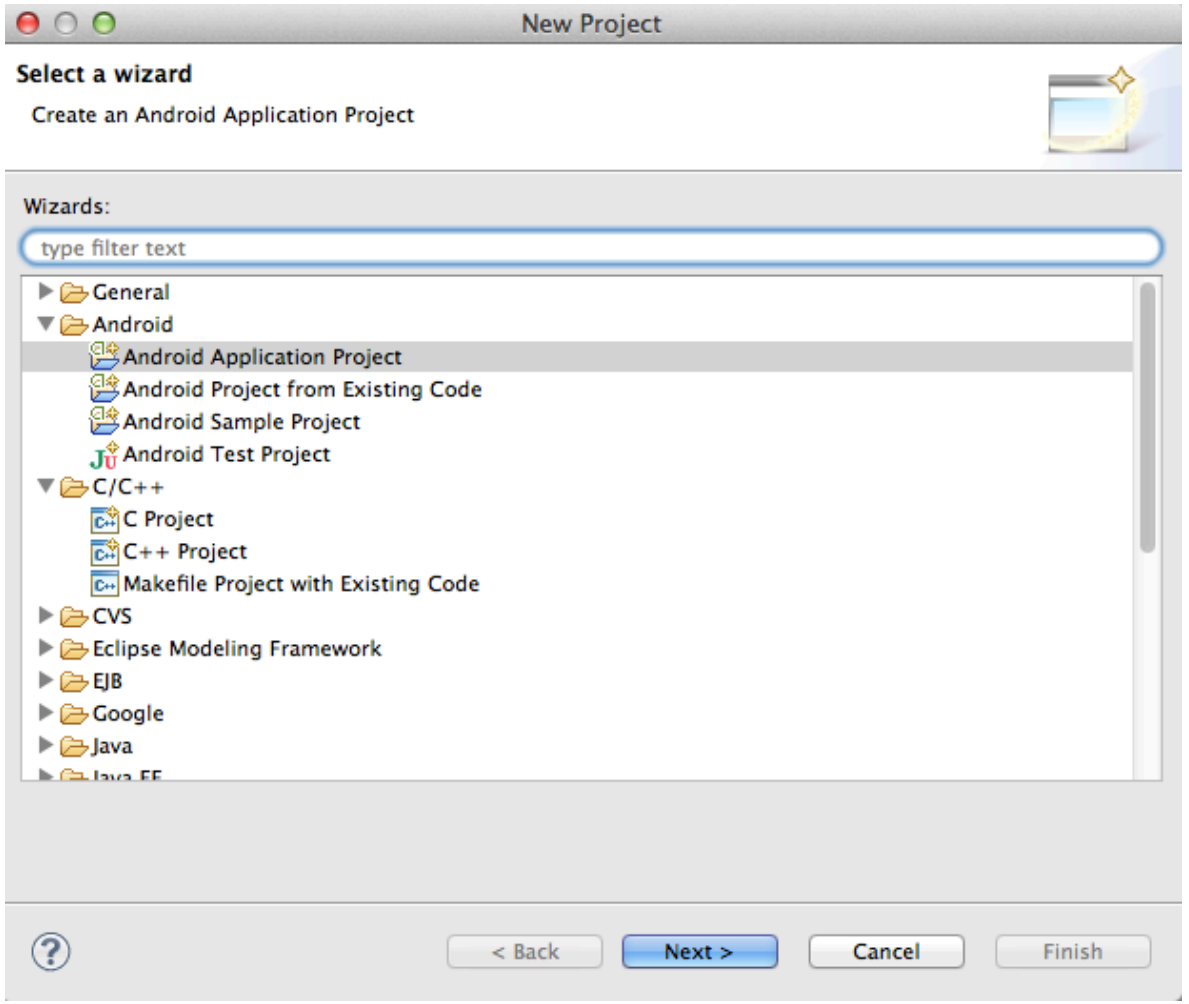


Exemplo

- Log de Eventos
  - Construir um banco de dados que armazene "ações" (eventos)
  - Uma evento possui um título e o tempo em que aconteceu
  - Abstração:
    - Classe principal (inicializa a chamada ao banco e aos eventos)
    - Eventos são salvos através do método de uma classe EventData e apresentados através de métodos dessa mesma classe.

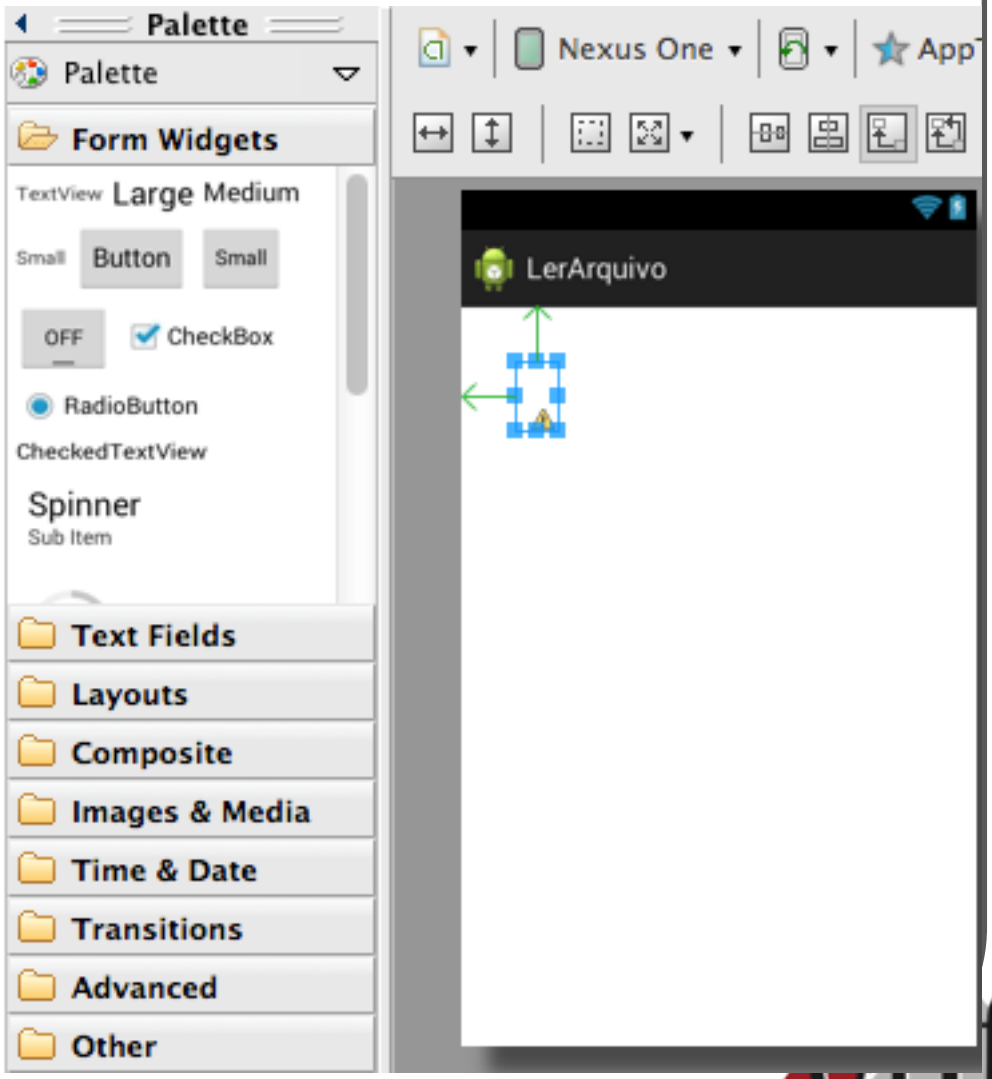


Acessando uma base de dados



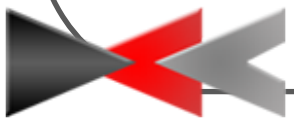
Continuando ...

- Activity_main.xml
- Escolha um novo TextView
- Remova a propriedade text “deixe vazia”



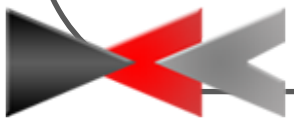
Activity

```
private EventsData events;

public static  String TABLE_NAME = "events";
public static  String TIME = "time";
public static  String TITLE = "title";
private static String[] FROM = { _ID, TIME, TITLE, };
private static String ORDER_BY = TIME + " DESC";

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    events = new EventsData(this);
    try {
      addEvent("Hello, Android!");
      Cursor cursor = getEvents();
      showEvents(cursor);
    } finally {
      events.close();
    }
}
```

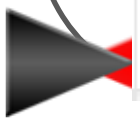


SQLite

- Uma das maneiras de acessar o banco de dados consiste em implementar a class SQLiteOpenHelper
  - Oferece os métodos básicos para gerenciar a base
  - Construtor: informar o nome da base de dados
  - OnCreate: criar as tabelas necessárias
  - onUpgrade: gerenciar as atualizações realizadas (se necessário)

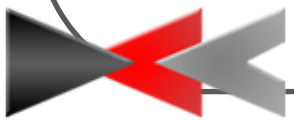


```
import static android.provider.BaseColumns._ID;
import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;

public class EventsData extends SQLiteOpenHelper {
      private static final String DATABASE_NAME = "events.db";
      private static final int DATABASE_VERSION = 1;
      private static final String TABLE_NAME = "events";
      private static final String TIME = "time";
      private static final String TITLE = "title";

      public EventsData(Context ctx) {
        super(ctx, DATABASE_NAME, null, DATABASE_VERSION);
      }
      @Override
      public void onCreate(SQLiteDatabase db) {
        db.execSQL("CREATE TABLE " + TABLE_NAME + " (" + _ID
        + " INTEGER PRIMARY KEY AUTOINCREMENT, " + TIME
        + " INTEGER," + TITLE + " TEXT NOT NULL);");
      }
      @Override
      public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
        db.execSQL("DROP TABLE IF EXISTS " + TABLE_NAME);
        onCreate(db);
      }
    }
```



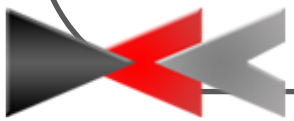
Activity (addEvent)

```
private void addEvent(String string) {
        SQLiteDatabase db = events.getWritableDatabase();
        ContentValues values = new ContentValues();
        values.put(TIME, System.currentTimeMillis());
        values.put(TITLE, string);
        db.insertOrThrow(TABLE_NAME, null, values);
}
```



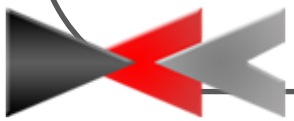
Activity (getEvents)

```
@SuppressWarnings("deprecation")
private Cursor getEvents() {
        SQLiteDatabase db = events.getReadableDatabase();
        Cursor cursor = db
            .query(TABLE_NAME, FROM, null, null, null, null, ORDER_BY);
        startManagingCursor(cursor);
        return cursor;
  }
```

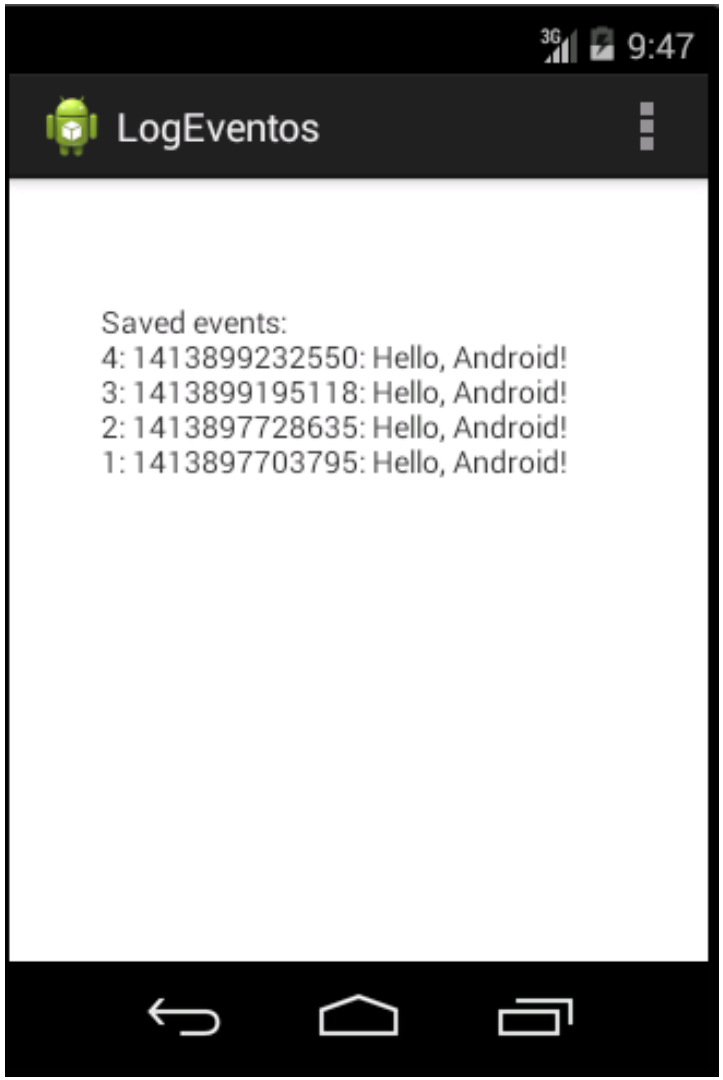


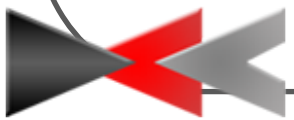
Activity (showEvents)

```
private void showEvents(Cursor cursor) {
        StringBuilder builder = new StringBuilder("Saved events:\n");
        while (cursor.moveToNext()) {
          long id = cursor.getLong(0);
          long time = cursor.getLong(1);
          String title = cursor.getString(2);
          builder.append(id).append(": ");
          builder.append(time).append(": ");
          builder.append(title).append("\n");
        }
        TextView text = (TextView) findViewById(R.id.textView1);
        text.setText(builder);
 }
```



LogEventos

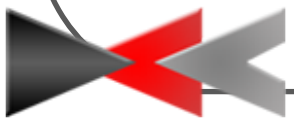

9:47
LogEventos
Saved events:
4: 1413899232550: Hello, Android!
3: 1413899195118: Hello, Android!
2: 1413897728635: Hello, Android!
1: 1413897703795: Hello, Android!



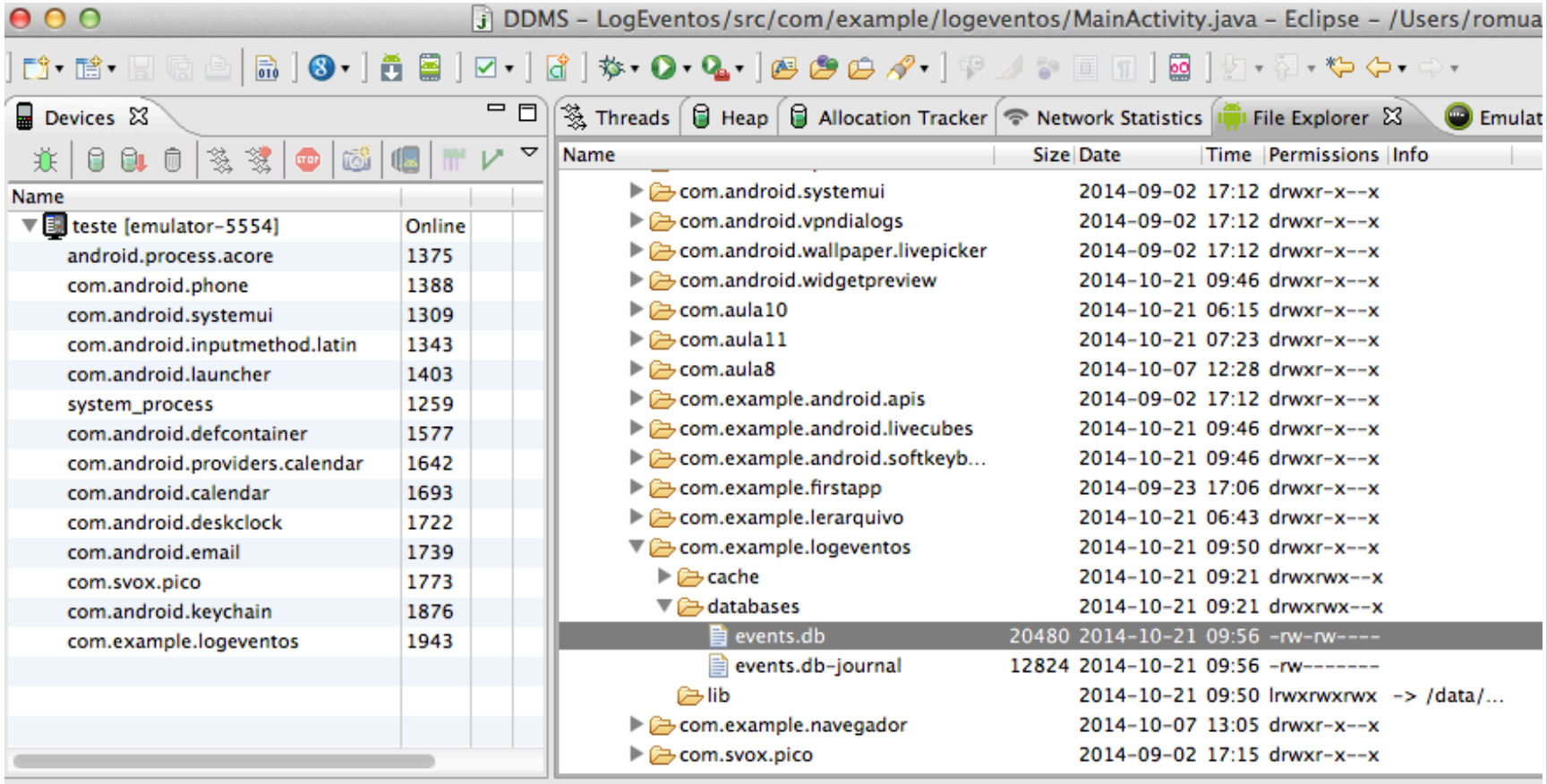

ufjf

Acesso a Base de Dados

- Abra a janela Window > Open Perspective > Other > DDMS
  - Aba File Explorer
  - Pasta data/data/nome-do-projeto(com.example.logeventos)/databases
  - Arquivo .db

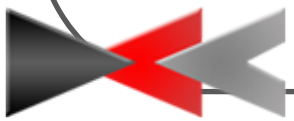


File Explorer



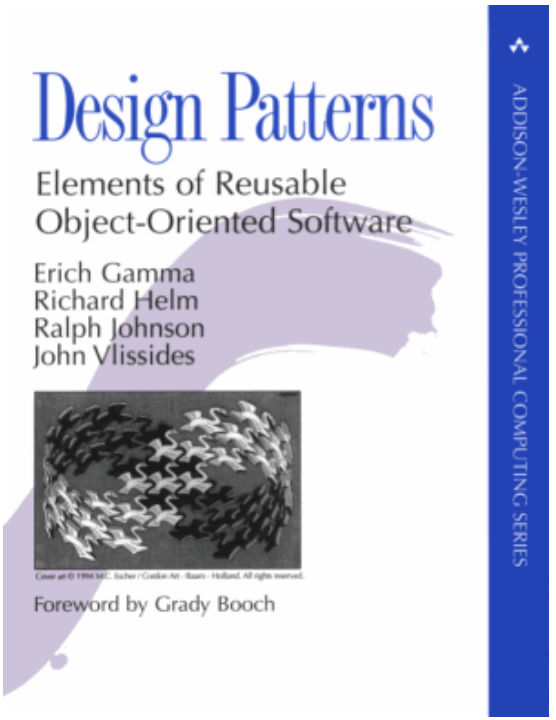
Problema

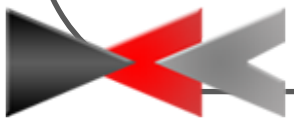
- Colocar o código SQL dentro das classes muitas vezes não reflete a modelagem do sistema:
  - Perda da representatividade
  - Dificuldade de manutenção
  - Entendimento da modelagem de dados
- Idéia: remover o código de acesso ao banco das classes do sistema e colocá-lo em uma classe responsável pelo acesso aos dados



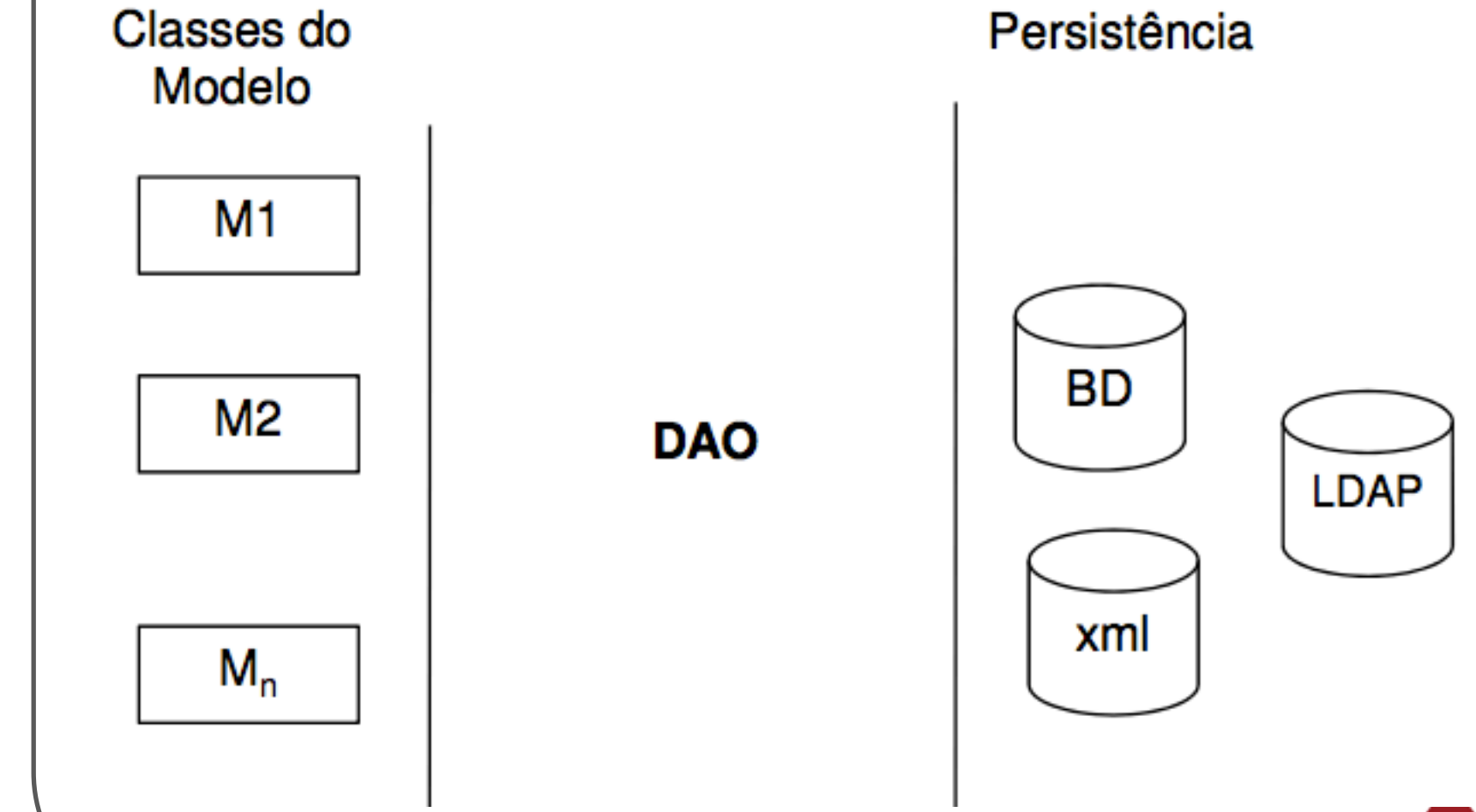
Design Pattern

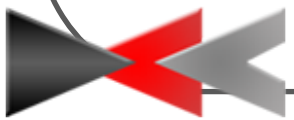
- Padrões de Projeto:
  - Factory
  - Data Access Object: DAO
    Isola a camada do modelo (sistema) da camada de persistência





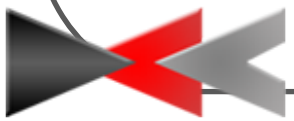
DAO





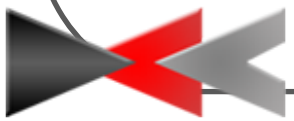
Vantagens

- Independência entre o modelo de persistência e o modelo do sistema:
  - Pode-se mudar a persistência sem modificar o sistema
  - Clientes diferentes podem utilizar SGBDs diferentes
  - Mecanismos de persistência distintos podem ser utilizados (Relacional, BigData, Arquivos etc.)
  - Encapsulamento das conexões quando necessárias



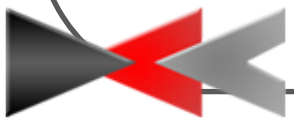
DAO com SQLite

- SQLiteOpenHelper
  - Construtor: informar o nome da base de dados
  - OnCreate: criar as tabelas necessárias
  - onUpgrade: gerenciar as atualizações realizadas na estrutura
- Manipulação dos dados
  - Objeto da classe SQLiteDatabase
  - Mantido dentro da classe que estende SQLiteOpenHelper



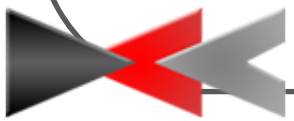
Exemplo

- Mover o acesso aos dados para a classe EventsData
  - Colocar um listview na Activity que recebe uma lista dos eventos armazenados

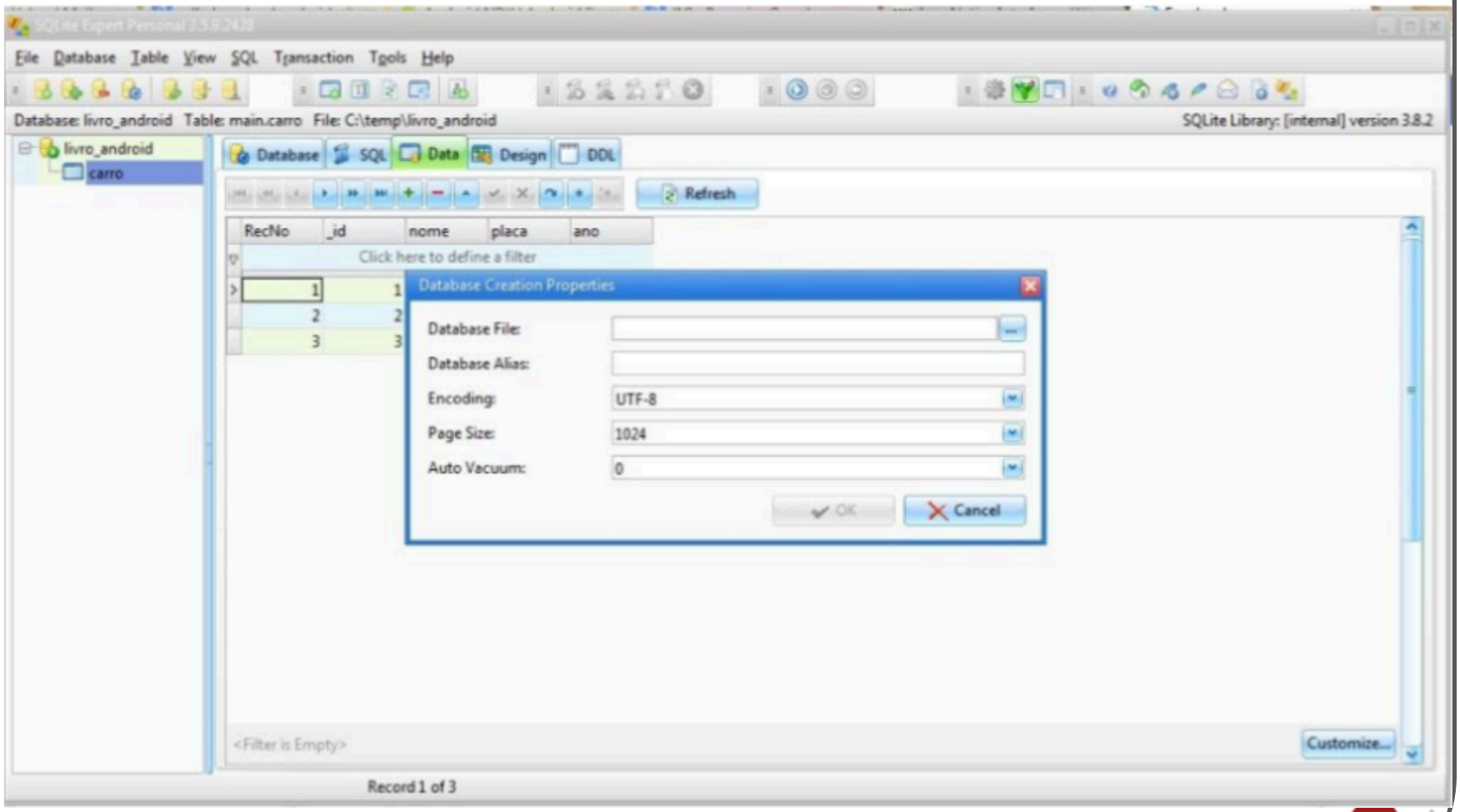


Ferramentas

- SQLite Expert Personal
  - Versão Grátis
  - Criação e manipulação de bases de dados .db
  - Windows
- SQLite Browser
  - Totalmente gratuito
  - Projeto livre para a criação e manipulação de .db
  - Diferentes sistemas operacionais (interface qt)



SQLite Expert Personal



SQLite Database Browser