# Java e Bancos de Dados

Profa Andréa Schwertner Charão

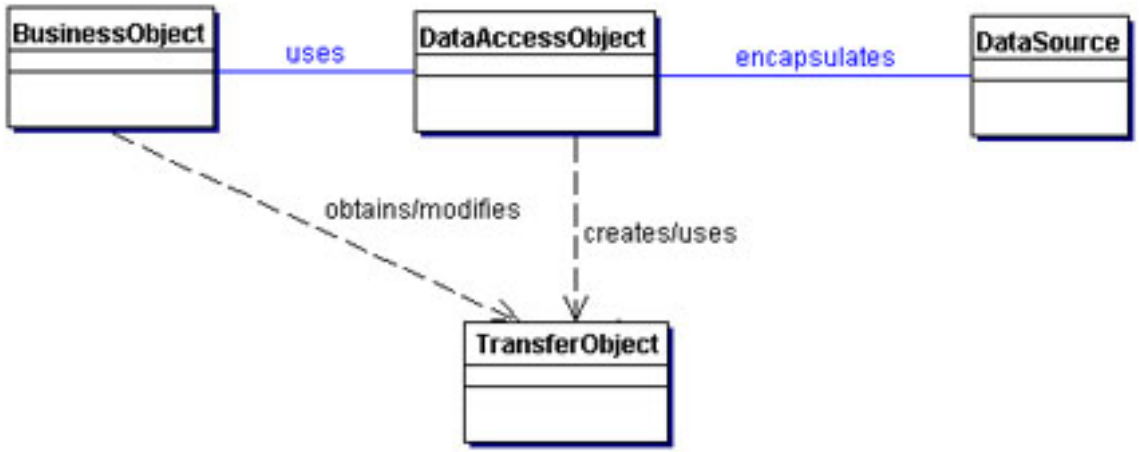
DELC/CT/UFSM

# Problemas de ExemploJDBC.java

- Execução de comandos SQL pode gerar falhas de segurança (*SQL injection*)
- Comandos SQL não são reutilizados
- Não usa orientação a objetos para manipular ACGs
- Mistura código de acesso a dados com código de interface com o usuário (percorre cada registro e mostra na saída padrão)

# Melhorando o exemplo

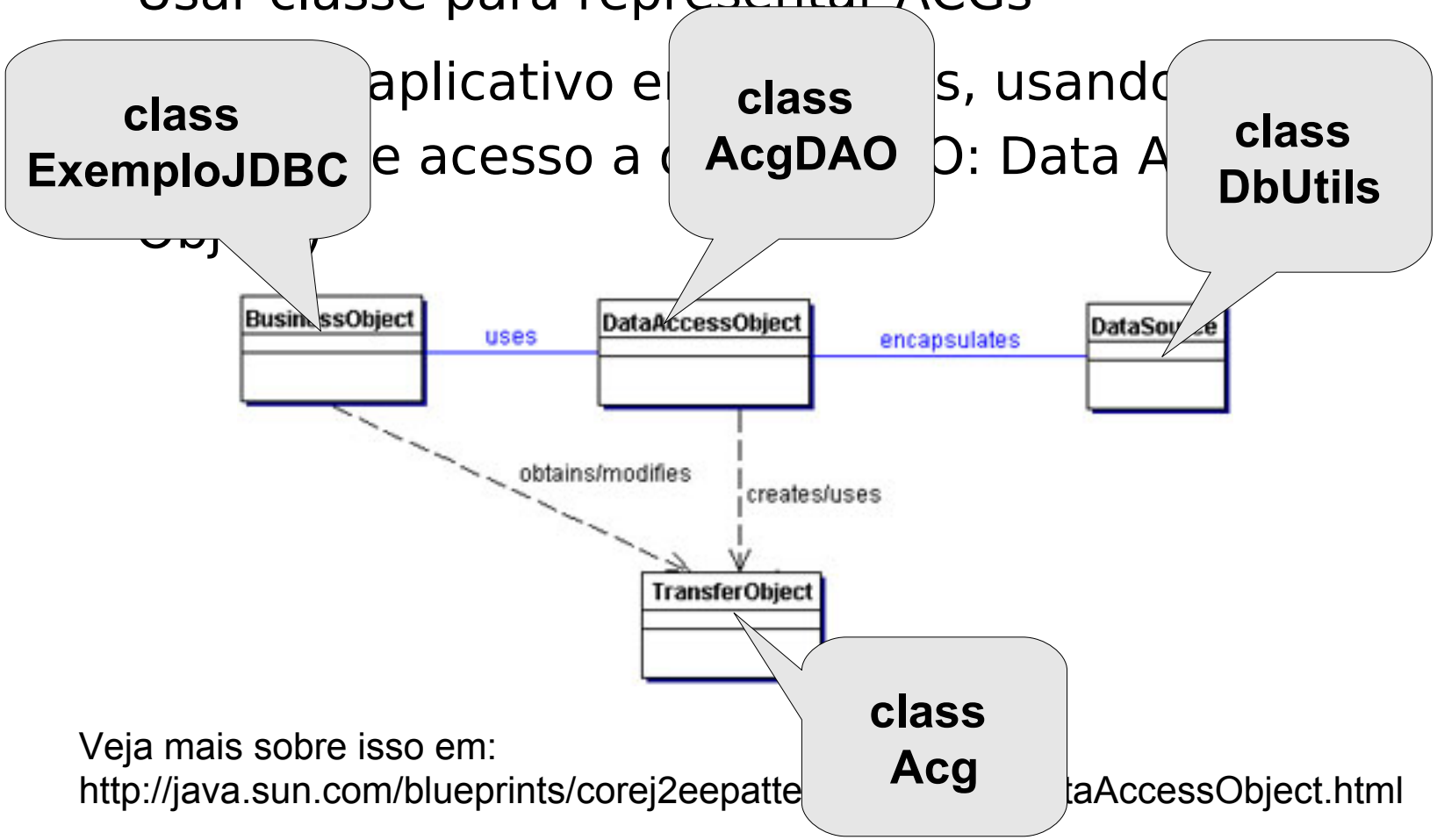
- Usar classe para representar ACGs
- Dividir o aplicativo em camadas, usando classes de acesso a dados (DAO: Data Access Object)




Veja mais sobre isso em:
http://java.sun.com/blueprints/corej2eepatterns/Patterns/DataAccessObject.html

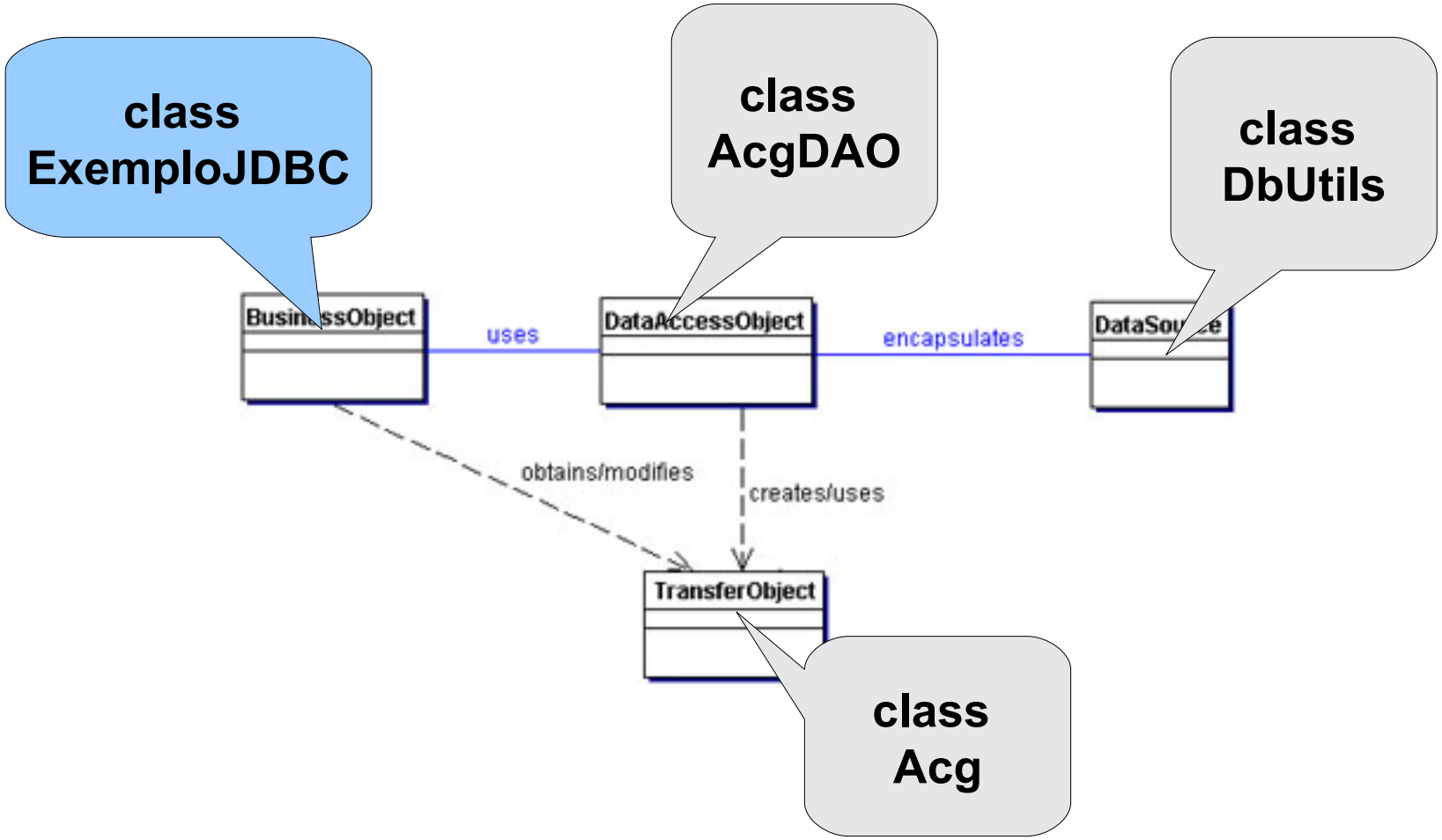
# Melhorando o exemplo

- Usar classe para representar ACGs




Veja mais sobre isso em:
http://java.sun.com/blueprints/corej2eepatte taAccessObject.html

# Reescrevendo classe ExemploJDBC



# Classe ExemploJDBC

ANTES

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;

public class ExemploJDBC {

   // Constantes para conexao com HSQLSB (modo standalone)
   static final String JDBC_DRIVER = "org.hsqldb.jdbcDriver";
   static final String DB_NAME = "acgdb";
   static final String DB_URL = "jdbc:hsqldb:file:" + DB_NAME;
   static final String DB_USER = "sa";
   static final String DB_PASSWD = "";

   public static void main(String[] args) throws ClassNotFoundException, SQLException {
      // Carrega o driver JDBC do HSQLDB.
      // O driver hsqldb.jar deve estar no classpath.
      Class.forName(JDBC_DRIVER);

      // Conecta o programa com o banco de dados.
      // Os arquivos do banco sao criados se nao existirem.
      Connection c = DriverManager.getConnection(DB_URL,
                                                 DB_USER,
                                                 DB_PASSWD);
```

# Classe ExemploJdbcDAO

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;

public class ExemploJdbcDAO {

   public static void main(String[] args) throws ClassNotFoundException, SQLException {

      Connection c = DbUtils.getConnection();
```

**DEPOIS**

Constantes e métodos para estabelecer conexão com o DB ficam encapsulados em DbUtils

# Classe ExemploJDBC: criação da tabela e inserção de dados

**ANTES**

```
// Statement recebe comandos SQL e os executa no banco
Statement s = c.createStatement();
s.executeUpdate("CREATE TABLE acg (id INTEGER, descr VARCHAR, ch INTEGER, PRIMARY KEY(id))");
s.executeUpdate("INSERT INTO acg VALUES(1, 'Participação na ERAD 2010', 30)");
s.executeUpdate("INSERT INTO acg VALUES(2, 'Pesquisa no LSC', 60)");
s.executeUpdate("INSERT INTO acg VALUES(3, 'Participação na ERRC 2009', 30)");

// ResultSet vai conter os resultados da consulta
ResultSet rs;
rs = s.executeQuery("SELECT * FROM acg");
while (rs.next())
   System.out.println(rs.getInt("id") + " " +
                      rs.getString("descr") + " " +
                      rs.getInt("ch"));
```

# Classe ExemploJdbcDAO: criação da tabela e inserção de dados

```
AcgDAO acgs = new AcgDAO(c);
acgs.dropTable();
acgs.createTable();

Acg acg1 = new Acg(1, "Participação na ERAD 2010", 30);
Acg acg2 = new Acg(2, "Pesquisa no LSC", 60);
Acg acg3 = new Acg(3, "Participação na ERRC 2009", 30);
acgs.create(acg1);
acgs.create(acg2);
acgs.create(acg3);

//System.out.println(acgs.findAll());
for (Acg a : acgs.findAll())
   System.out.println(a);
```

**DEPOIS: classes Acg e AcgDAO**

**Métodos de AcgDAO:**
dropTable
createTable
create
findAll
update
delete
etc.

# Classe ExemploJDBC: mostrar todos os registros

ANTES

```
// Statement recebe comandos SQL e os executa no banco
Statement s = c.createStatement();
s.executeUpdate("CREATE TABLE acg (id INTEGER, descr VARCHAR, ch INTEGER, PRIMARY KEY(id))");
s.executeUpdate("INSERT INTO acg VALUES(1, 'Participação na ERAD 2010', 30)");
s.executeUpdate("INSERT INTO acg VALUES(2, 'Pesquisa no LSC', 60)");
s.executeUpdate("INSERT INTO acg VALUES(3, 'Participação na ERRC 2009', 30)");

// ResultSet vai conter os resultados da consulta
ResultSet rs;
rs = s.executeQuery("SELECT * FROM acg");
while (rs.next())
   System.out.println(rs.getInt("id") + " " +
                      rs.getString("descr") + " " +
                      rs.getInt("ch"));
```

# Classe ExemploJdbcDAO: mostrar todos os registros

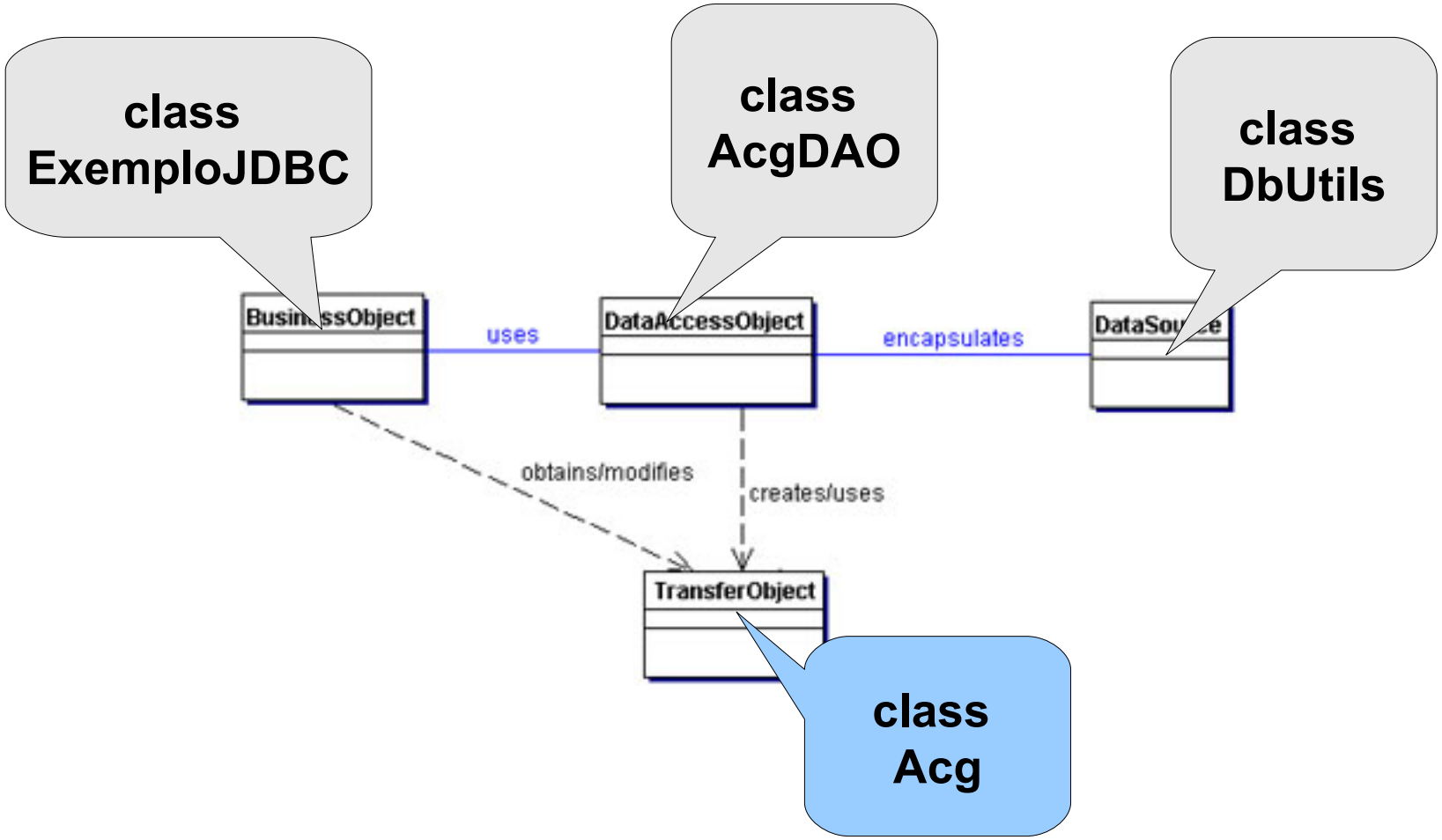
```
AcgDAO acgs = new AcgDAO(c);
acgs.dropTable();
acgs.createTable();

Acg acg1 = new Acg(1, "Participação na ERAD 2010", 30);
Acg acg2 = new Acg(2, "Pesquisa no LSC", 60);
Acg acg3 = new Acg(3, "Participação na ERRC 2009", 30);
acgs.create(acg1);
acgs.create(acg2);
acgs.create(acg3);

//System.out.println(acgs.findAll());
for (Acg a : acgs.findAll())
   System.out.println(a);
```

**DEPOIS**

**Método findAll retorna uma coleção (ArrayList) de Acg's**

**Classe Acg sobrescreve toString**

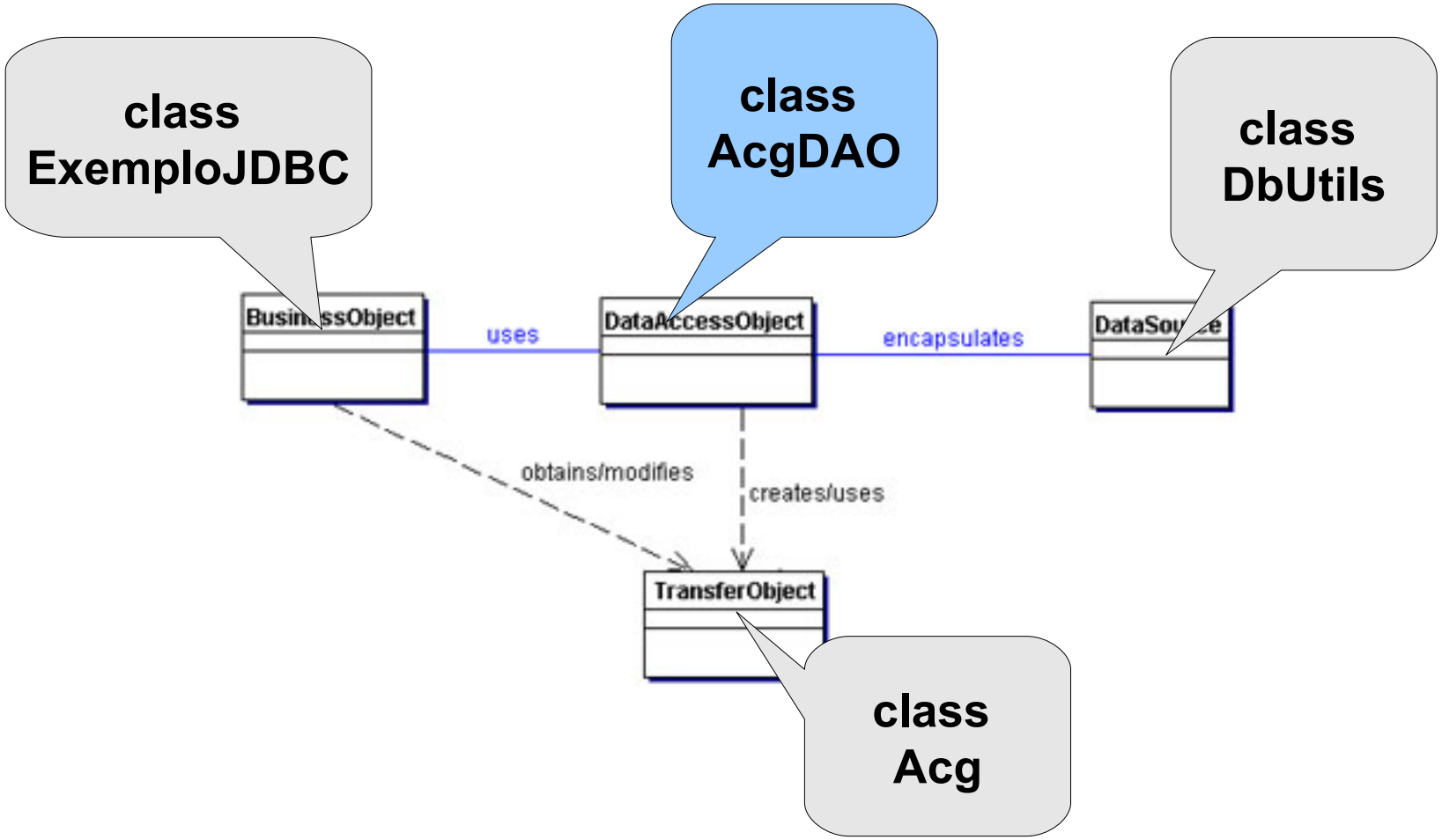
# Classe Acg


class ExemploJDBC
class AcgDAO
class DbUtils
BusinessObject
uses
DataAccessObject
encapsulates
DataSource
obtains/modifies
creates/uses
TransferObject
class Acg

# Classe Acg

```java
public class Acg {

    private Integer id;
    private String descr;
    private Integer ch;

    public Acg() {
    }
    public Acg(Integer id, String descr, Integer ch) {
        setId(id);
        setDescr(descr);
        setCh(ch);
    }
    public Integer getId() {
        return id;
    }
    public void setId(Integer id) {
        this.id = id;
    }
    public String getDescr() {
        return descr;
    }
    public void setDescr(String descr) {
        this.descr = descr;
    }
    public int getCh() {
        return ch;
    }
    public void setCh(int ch) {
        this.ch = ch;
    }
    public String toString() {
        return getId() + " " +
               getDescr() + " " +
               getCh();
    }
}
```

# Classe AcgDAO



# Interface AcgDAOInterface

```java
import java.sql.SQLException;
import java.util.List;

public interface AcgDAOInterface {

    public void createTable() throws SQLException;

    public void dropTable() throws SQLException;

    public void create(Acg a) throws SQLException;

    public Acg find(Integer id) throws SQLException;

    public List<Acg> findAll() throws SQLException;

    public void update(Acg a) throws SQLException;

    public void delete(Integer id) throws SQLException;
}
```

**C**reate
**R**ead
**U**pdate
**D**elete

# Classe AcgDAO

```
public class AcgDAO implements AcgDAOInterface {

   private static final String SQL_INSERT =
      "INSERT INTO acg (id, descr, ch) VALUES (?, ?, ?)";

   private static final String SQL_SELECT =
      "SELECT id, descr, ch FROM acg WHERE id = ?";

   private static final String SQL_UPDATE =
      "UPDATE acg SET descr = ?, ch = ? WHERE id = ?";

   private static final String SQL_DELETE =
      "DELETE FROM acg WHERE id = ?";

   private static final String SQL_SELECT_ALL =
      "SELECT * FROM acg";

   private static final String SQL_CREATE =
      "CREATE TABLE acg " +
         "(id INTEGER, descr VARCHAR(255), ch INTEGER, PRIMARY KEY(id))";

   private static final String SQL_DROP =
      "DROP TABLE acg IF EXISTS";

   private Connection conn;

   public AcgDAO(Connection conn) {
      this.conn = conn;
   }
```

# Classe AcgDAO

```
public void createTable() throws SQLException {
   Statement s = null;
   try {
      s = conn.createStatement();
      s.executeUpdate(SQL_CREATE);
   } finally {
      close(s);
   }
}
```

# Classe AcgDAO

```
public void create(Acg a) throws SQLException {
   PreparedStatement ps = null;
   try {
      ps = conn.prepareStatement(SQL_INSERT);
      ps.setInt(1, a.getId());
      ps.setString(2, a.getDescr());
      ps.setInt(3, a.getCh());
      ps.executeUpdate();
   } finally {
      close(ps);
   }
}
```

# Classe AcgDAO

```
public List<Acg> findAll() throws SQLException {
   PreparedStatement ps = null;
   ResultSet rs = null;
   try {
      ps = conn.prepareStatement(SQL_SELECT_ALL);
      rs = ps.executeQuery();
      List<Acg> results = getResults(rs);
      return results;
   } finally {
      close(rs);
      close(ps);
   }
}
```

```
private List<Acg> getResults(ResultSet rs) throws SQLException {
   List<Acg> results = new ArrayList<Acg>();
   while (rs.next()) {
      Acg a = new Acg();
      a.setId(rs.getInt("id"));
      a.setDescr(rs.getString("descr"));
      a.setCh(rs.getInt("ch"));
      results.add(a);
   }
   return results;
}
```

# Melhorando ExemploJdbcDAO

- Se tivermos várias classes (Acgs, Alunos, etc.), vai haver replicação de código para algumas operações (CRUD)
- Uma solução: criar um DAO genérico (exercício)

```
public interface GenericDAOInterface <T, PK> {
    ...
}
public abstract class GenericDAO <T, PK>
    implements GenericDAOInterface <T, PK> {
    ...
}
public class AcgDAO extends GenericDAO <Acg, Integer> {
    ...
}
```

# Tratamento de exceções

- **Exceção**: evento "excepcional" que interrompe o fluxo normal da execução de um programa
- Exemplos:
  - Uso indevido de referência null
  - Divisão por zero, raiz quadrada de número negativo, etc.
  - Uso de índice inválido em arrays e coleções
  - Problemas na manipulação de arquivos
  - Problemas no acesso a banco de dados
  - Etc.
- Exceções tratadas deixam o programa mais confiável

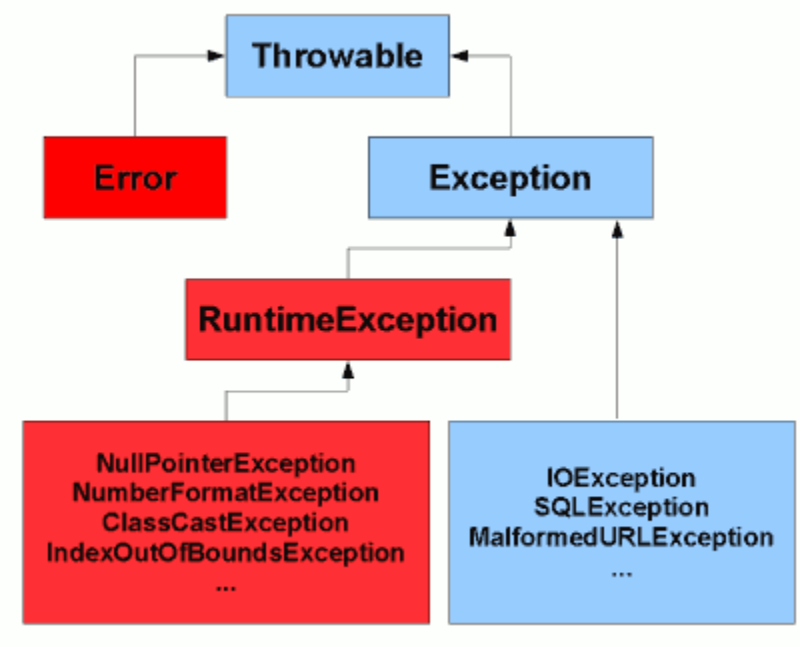
# Tratamento de exceções em C

- Em C, geralmente testamos o valor de retorno das funções para verificar se ocorreu erro/exceção
- Programador deve se lembrar de testar cada valor retornado
- Código pode ficar pouco legível

```
FILE *fpt = fopen(file, "rb");
if (!fpt) {
  // some error occurred -- find out what
} else {
  // read from file
}
```

# Exceções em Java

- Java oferece vários recursos (classes e sintaxe de linguagem) para lidar com exceções
- Classes
  - Exception: hierarquia de classes que representam exceções tratáveis
  - Error: erro grave que normalmente não pode ser tratado




Fonte:
http://www.javamex.com/tutorials/exceptions/exceptions_hierarchy.shtml

# Exceções em Java

- Recursos da linguagem
  - Bloco try/catch:
    - ✓ try: declara bloco de código que pode produzir exceções
    - ✓ catch: trata exceções que ocorrerem no bloco try
    - ✓ finally: declara bloco de código que sempre executa, com ou sem exceção ocorrida (geralmente para código de "limpeza")
  - throw: lança exceção explicitamente
  - throws: indica exceções que um método/classe podem produzir

# Exceções em Java

```
class ExemploExcecao {
   public static void main (String[] args) {
      int n = args.length + 10;
      String s = args[n];
      System.out.println("Vou terminar a execucao");
   }
}
```

```
Exception in thread "main"
java.lang.ArrayIndexOutOfBoundsException: 0
        at ExemploExcecao.main(ExemploExcecao.java:4)
```

# Exceções em Java

```
class ExemploTrataExcecao {
   public static void main (String[] args) {

      try {
         int n = args.length + 10;
         String s = args[n];
      }
      catch (ArrayIndexOutOfBoundsException e)
      {
         System.out.println("Indice fora dos limites");
      }
      System.out.println("Vou terminar a execucao");
   }
}
```

Saída:
Indice fora dos limites
Vou terminar a execucao

# Exceções em Java

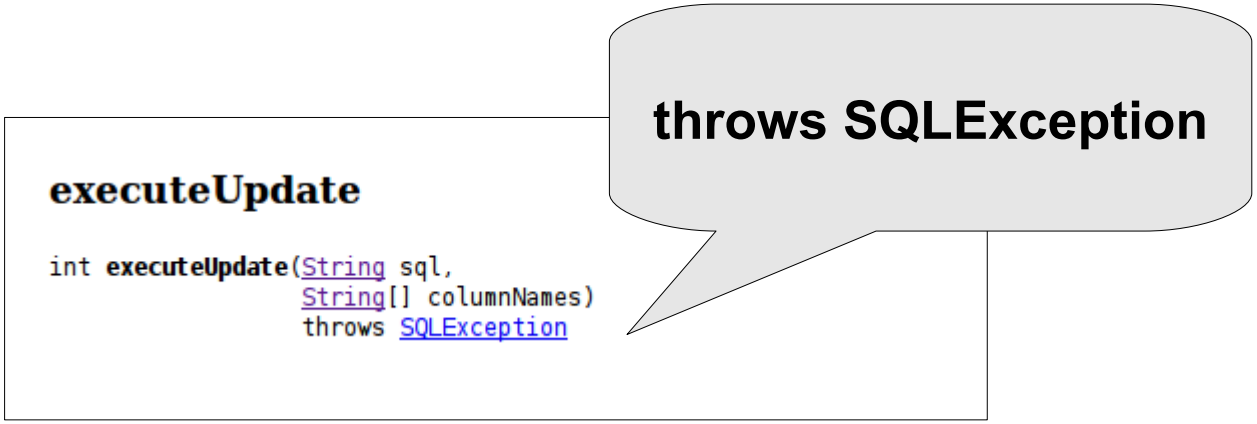
```
class ExemploTrataE
   public static vo
      try {
         int n = ar
         String s = arg
      }
      catch (ArrayIndexOutOfBoundsException e) {
         System.out.println("Indice fora dos limites");
      }
      catch (Exception e) {
         e.printStackTrace();
      }
      finally {
         System.out.println("Codigo sempre executado");
      }
      System.out.println("Vou terminar a execucao");
   }
}
```

**Ordem de teste:**
**classes específicas antes de classes gerais**

Saída:
Indice fora dos limites
Codigo sempre executado
Vou terminar a execucao

# Exceções em Java

- Uso de throws: indica exceções que um método/classe podem produzir
- Algumas exceções são verificadas em classes/métodos da plataforma Java



# Exceções em Java

- Para usar um método que verifica uma exceção:
  - Capturar a exceção em um bloco try-catch
  - Ou delegar o tratamento para quem invocou o método (usando throws)
- Se isso não for feito, compilação vai gerar erro

```
public void newCreateTable() {
    Statement s = null;
    try {
        s = conn.createStatement();
        s.executeUpdate(sqlCreate);
    }
    catch (SQLException e) {
        e.printStackTrace();
    }
    finally {
        close(s);
    }
}
```

# Exceções em Java

- Para usar um método que verifica uma exceção:
  - Capturar a exceção em um bloco try-catch
  - Ou delegar o tratamento para quem invocou o método (usando throws)
- Se isso não for feito, compilação vai gerar erro

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.ResultSetMetaData;
import java.sql.SQLException;
import java.sql.Statement;

public class ExemploJdbcDAO {

   public static void main(String[] args) throws ClassNotFoundException, SQLException {

      Connection c = DbUtils.getConnection();
```

# Exceções em Java

- Objetivos/vantagens:
  - Separar código "normal" do código que trata situações "anormais"
  - Facilitar propagação de erros na pilha de chamadas
  - Agrupar e diferenciar tipos de erros
- Mais sobre isso em:

  http://download.oracle.com/javase/tutorial/essential/exceptions/advantages.html

# Exceções:
# o caso Ariane 5

- Em 1996, foguete Ariane 5, da Agência Espacial Européia, explode durante lançamento
- Bug causado pelo reuso de software de Ariane 4
- **Exceção** ocorreu na conversão de um float (64 bits) para um inteiro (16 bits)
- Sistema primário não tratou exceção e falhou, sistema secundário assumiu, mas rodava o mesmo software e falhou também
- Mais sobre isso em: http://www.ima.umn.edu/~arnold/disasters/ariane5rep.html