Aspect-oriented programming

Paulo Borba

Informatics Center

Federal University of Pernambuco

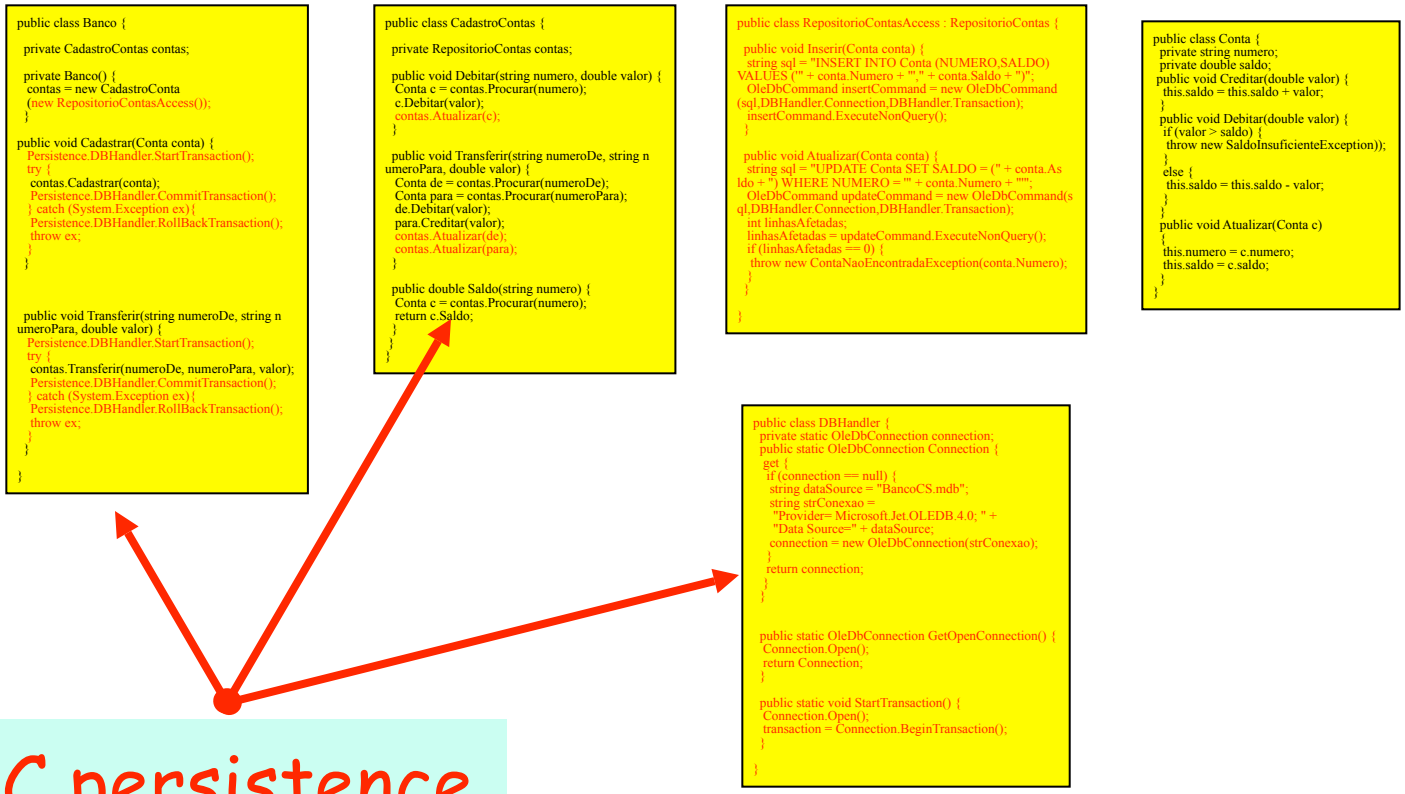
Basic AspectJ concepts

Paulo Borba

Informatics Center

Federal University of Pernambuco

OO problems with crosscutting concerns



```
public class Banco {

  private CadastroContas contas;

  private Banco() {
   contas = new CadastroConta
   (new RepositorioContasAccess());
  }

public void Cadastrar(Conta conta) {
    Persistence.DBHandler.StartTransaction();
    try {
     contas.Cadastrar(conta);
     Persistence.DBHandler.CommitTransaction();
    } catch (System.Exception ex){
     Persistence.DBHandler.RollBackTransaction();
     throw ex;
    }
  }


   public void Transferir(string numeroDe, string n
umeroPara, double valor) {
    Persistence.DBHandler.StartTransaction();
    try {
     contas.Transferir(numeroDe, numeroPara, valor);
     Persistence.DBHandler.CommitTransaction();
    } catch (System.Exception ex){
     Persistence.DBHandler.RollBackTransaction();
     throw ex;
    }
   }

}
```

```
public class CadastroContas {

  private RepositorioContas contas;

  public void Debitar(string numero, double valor) {
   Conta c = contas.Procurar(numero);
   c.Debitar(valor);
   contas.Atualizar(c);
  }

  public void Transferir(string numeroDe, string n
umeroPara, double valor) {
    Conta de = contas.Procurar(numeroDe);
    Conta para = contas.Procurar(numeroPara);
    de.Debitar(valor);
    para.Creditar(valor);
    contas.Atualizar(de);
    contas.Atualizar(para);
  }

  public double Saldo(string numero) {
   Conta c = contas.Procurar(numero);
   return c.Saldo;
  }
 }
}
```

```
public class RepositorioContasAccess : RepositorioContas {

  public void Inserir(Conta conta) {
   string sql = "INSERT INTO Conta (NUMERO,SALDO)
VALUES ('" + conta.Numero + "'," + conta.Saldo + ")";
   OleDbCommand insertCommand = new OleDbCommand
(sql,DBHandler.Connection,DBHandler.Transaction);
   insertCommand.ExecuteNonQuery();
  }

  public void Atualizar(Conta conta) {
   string sql = "UPDATE Conta SET SALDO = (" + conta.As
ldo + ") WHERE NUMERO = '" + conta.Numero + "'";
   OleDbCommand updateCommand = new OleDbCommand(s
ql,DBHandler.Connection,DBHandler.Transaction);
   int linhasAfetadas;
   linhasAfetadas = updateCommand.ExecuteNonQuery();
   if (linhasAfetadas == 0) {
    throw new ContaNaoEncontradaException(conta.Numero);
   }
  }

 }
```

```
public class Conta {
  private string numero;
  private double saldo;
  public void Creditar(double valor) {
   this.saldo = this.saldo + valor;
  }
  public void Debitar(double valor) {
   if (valor > saldo) {
    throw new SaldoInsuficienteException));
   }
   else {
    this.saldo = this.saldo - valor;
   }
  }
  public void Atualizar(Conta c)
  {
   this.numero = c.numero;
   this.saldo = c.saldo;
  }
}
```

```
public class DBHandler {
  private static OleDbConnection connection;
  public static OleDbConnection Connection {
   get {
    if (connection == null) {
     string dataSource = "BancoCS.mdb";
     string strConexao =
      "Provider= Microsoft.Jet.OLEDB.4.0; " +
      "Data Source=" + dataSource;
     connection = new OleDbConnection(strConexao);
    }
    return connection;
   }
  }


  public static OleDbConnection GetOpenConnection() {
   Connection.Open();
   return Connection;
  }

  public static void StartTransaction() {
   Connection.Open();
   transaction = Connection.BeginTransaction();
  }

}
```

JDBC persistence code in red…

Persistência na fachada

```
public class Banco {
  private CadastroContas contas;...
  public void cadastrar(Conta conta) {
    try {
      getPM().startTransaction();
      contas.cadastrar(conta); ...
      getPM().commitTransaction();
    } catch (Exception ex){
      getPM().rollBackTransaction();...
    }
  }...
```

Código de negócio misturado com transações (não pode ficar na coleção de dados)

Same problem with Hibernate!

```
public Blog createBlog(String name) ... {
  Blog blog = new Blog().setName(name); ...
  Session session = _sessions.openSession();
  Transaction tx = null;
  try {
    tx = session.beginTransaction();
    session.persist(blog);
    tx.commit();
  } catch (HibernateException he) {
    if (tx!=null) tx.rollback();
    throw he;
  } finally { session.close(); } ...
}
```

From http://www.hibernate.org/hib_docs/reference/en/html/example-weblog.html

Persistência na coleção de negócio

```
public class CadastroContas {
  private RepositorioContas contas;...
  public void creditar(String numero,
                       double valor) {
    Conta c = contas.procurar(numero);
    c.creditar(valor);
    contas.atualizar(c);
  }...
```

Sincronização de estados entre objetos e tabelas (registros)

Let's use aspects to...

modularize

this crosscutting concern

Sem aspectos

```
public class Banco {

  private CadastroContas contas;

  private Banco() {
   contas = new CadastroConta
   (new RepositorioContasAccess());
  }

 public void Cadastrar(Conta conta) {
   Persistence.DBHandler.StartTransaction();
   try {
    contas.Cadastrar(conta);
    Persistence.DBHandler.CommitTransaction();
   } catch (System.Exception ex){
    Persistence.DBHandler.RollBackTransaction();
    throw ex;
   }
  }

  public void Transferir(string numeroDe, string n
umeroPara, double valor) {
   Persistence.DBHandler.StartTransaction();
   try {
    contas.Transferir(numeroDe, numeroPara, valor);
    Persistence.DBHandler.CommitTransaction();
   } catch (System.Exception ex){
    Persistence.DBHandler.RollBackTransaction();
    throw ex;
   }
  }

}
```

```
public class CadastroContas {

  private RepositorioContas contas;

  public void Debitar(string numero, double valor) {
   Conta c = contas.Procurar(numero);
   c.Debitar(valor);
   contas.Atualizar(c);
  }

  public void Transferir(string numeroDe, string n
umeroPara, double valor) {
   Conta de = contas.Procurar(numeroDe);
   Conta para = contas.Procurar(numeroPara);
   de.Debitar(valor);
   para.Creditar(valor);
   contas.Atualizar(de);
   contas.Atualizar(para);
  }

  public double Saldo(string numero) {
   Conta c = contas.Procurar(numero);
   return c.Saldo;
  }
 }
}
```

```
public class RepositorioContasAccess : RepositorioContas {

  public void Inserir(Conta conta) {
   string sql = "INSERT INTO Conta (NUMERO,SALDO)
VALUES ('" + conta.Numero + "'," + conta.Saldo + ")";
   OleDbCommand insertCommand = new OleDbCommand
(sql,DBHandler.Connection,DBHandler.Transaction);
   insertCommand.ExecuteNonQuery();
  }

  public void Atualizar(Conta conta) {
   string sql = "UPDATE Conta SET SALDO = (" + conta.As
ldo + ") WHERE NUMERO = '" + conta.Numero + "'";
   OleDbCommand updateCommand = new OleDbCommand(s
ql,DBHandler.Connection,DBHandler.Transaction);
   int linhasAfetadas;
   linhasAfetadas = updateCommand.ExecuteNonQuery();
   if (linhasAfetadas == 0) {
    throw new ContaNaoEncontradaExeception(conta.Numero);
   }
  }

}
```

```
public class Conta {
  private string numero;
  private double saldo;
  public void Creditar(double valor) {
   this.saldo = this.saldo + valor;
  }
  public void Debitar(double valor) {
   if (valor > saldo) {
    throw new SaldoInsuficienteException));
   }
   else {
    this.saldo = this.saldo - valor;
   }
  }
  public void Atualizar(Conta c)
  {
   this.numero = c.numero;
   this.saldo = c.saldo;
  }
 }
```

```
public class DBHandler {
  private static OleDbConnection connection;
  public static OleDbConnection Connection {
   get {
    if (connection == null) {
     string dataSource = "BancoCS.mdb";
     string strConexao =
      "Provider= Microsoft.Jet.OLEDB.4.0; " +
      "Data Source=" + dataSource;
     connection = new OleDbConnection(strConexao);
    }
    return connection;
   }
  }
  public static OleDbConnection GetOpenConnection() {
   Connection.Open();
   return Connection;
  }

  public static void StartTransaction() {
   Connection.Open();
   transaction = Connection.BeginTransaction();
  }

 }
```

Com aspectos

```
public class Banco {

  private CadastroContas contas;

  private Banco() {
   contas = new CadastroConta
}

public void Cadastrar(Conta conta) {
contas.Cadastrar(conta);
}

  public void Transferir(string numeroDe, string n
umeroPara, double valor) {
contas.Transferir(numeroDe, numeroPara, valor);
}

}
```

```
public class CadastroContas {

  private RepositorioContas contas;

  public void Debitar(string numero, double valor) {
   Conta c = contas.Procurar(numero);
   c.Debitar(valor);
}

  public void Transferir(string numeroDe, string n
umeroPara, double valor) {
   Conta de = contas.Procurar(numeroDe);
   Conta para = contas.Procurar(numeroPara);
   de.Debitar(valor);
   para.Creditar(valor);
}

  public double Saldo(string numero) {
   Conta c = contas.Procurar(numero);
   return c.Saldo;
  }
 }
}
```

```
public class Conta {
  private string numero;
  private double saldo;
 public void Creditar(double valor) {
   this.saldo = this.saldo + valor;
  }
 public void Debitar(double valor) {
   if (valor > saldo) {
    throw new SaldoInsuficienteException));
   }
   else {
    this.saldo = this.saldo - valor;
   }
  }
 public void Atualizar(Conta c)
  {
   this.numero = c.numero;
   this.saldo = c.saldo;
  }
}
```

Código base do sistema

```
public class RepositorioContasAccess : RepositorioContas {

   public void Inserir(Conta conta) {
    string sql = "INSERT INTO Conta (NUMERO,SALDO)
VALUES ('" + conta.Numero + "'," + conta.Saldo + ")";
    OleDbCommand insertCommand = new OleDbCommand
(sql,DBHandler.Connection,DBHandler.Transaction);
    insertCommand.ExecuteNonQuery();
   }

   public void Atualizar(Conta conta) {
    string sql = "UPDATE Conta SET SALDO = (" + conta.As
ldo + ") WHERE NUMERO = '" + conta.Numero + "'";
    OleDbCommand updateCommand = new OleDbCommand(s
ql,DBHandler.Connection,DBHandler.Transaction);
    int linhasAfetadas;
    linhasAfetadas = updateCommand.ExecuteNonQuery();
    if (linhasAfetadas == 0) {
     throw new ContaNaoEncontradaException(conta.Numero);
    }
   }

}
```

```
public class DBHandler {
  private static OleDbConnection connection;
  public static OleDbConnection Connection {
   get {
    if (connection == null) {
     string dataSource = "BancoCS.mdb";
     string strConexao =
      "Provider= Microsoft.Jet.OLEDB.4.0; " +
      "Data Source=" + dataSource;
     connection = new OleDbConnection(strConexao);
    }
    return connection;
   }
  }
  public static OleDbConnection GetOpenConnection() {
   Connection.Open();
   return Connection;
  }

  public static void StartTransaction() {
   Connection.Open();
   transaction = Connection.BeginTransaction();
  }

}
```

```
public class Conta {
  private string numero;
  private double saldo;
 public void Creditar(double valor) {
   this.saldo = this.saldo + valor;
  }
  public void Debitar(double valor) {
```

```
public class Conta {
  private string numero;
;
  }
  public void Debitar(double valor) {
```

Código do concern de persistência

Pointcuts especificam join points

- Identificam joint points de um sistema
  - chamadas e execuções de métodos (e construtores)
  - acessos a atributos
  - tratamento de exceções
  - inicialização estática e dinâmica
- Composição de joint points
  - `&&`, `||` e `!`

Advice especifica comportamento extra nos join points

- Define código adicional que deve ser executado…
  - `before`
  - `after`
    - `after returning`
    - `after throwing`
  - `ou around`

  join points

Aspecto de persistência, advices para sucesso

```
public aspect PersistenceAspect {
  before(): transMethods() {
    getPM().startTransaction();
  }

  after() returning(): transMethods() {
    getPM().commitTransaction();
  }
```

Aspecto de persistência, advice para falha

Exception is not captured

```
after() throwing(): transMethods() {
    getPM().rollBackTransaction();
}
...
```

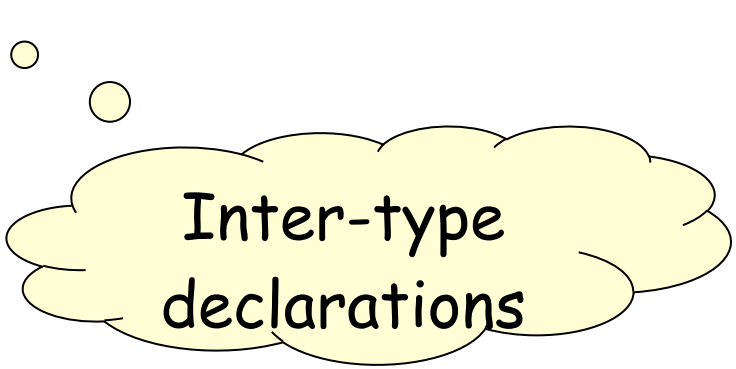
Aspecto de persistência, pointcut (alternativa simples)

```
pointcut TransMethods():
  execution(public * Banco.cadastrar(..)) ||
  execution(public * Banco.creditar(..)) ||
  execution(public * Banco.debitar(..)) ||
  ...
```

```
pointcut TransMethods():
  execution(public * Banco.*(..));
```

Além de dynamic crosscutting com advice...

- Temos também static crosscutting
  - alterar relação de subtipo
  - adicionar membros a classes



Aspecto de persistência, pointcut

interface local ao aspecto

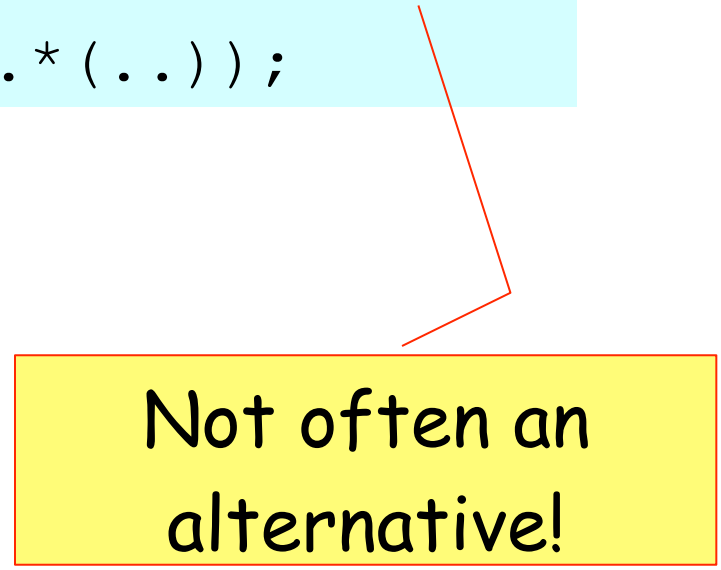
```
pointcut transMethods():
  execution(public * Trans.*(..));
private interface Trans {
  public void cadastrar(Conta conta);...
}
declare parents: Banco implements Trans;
```

altera a hierarquia de tipos

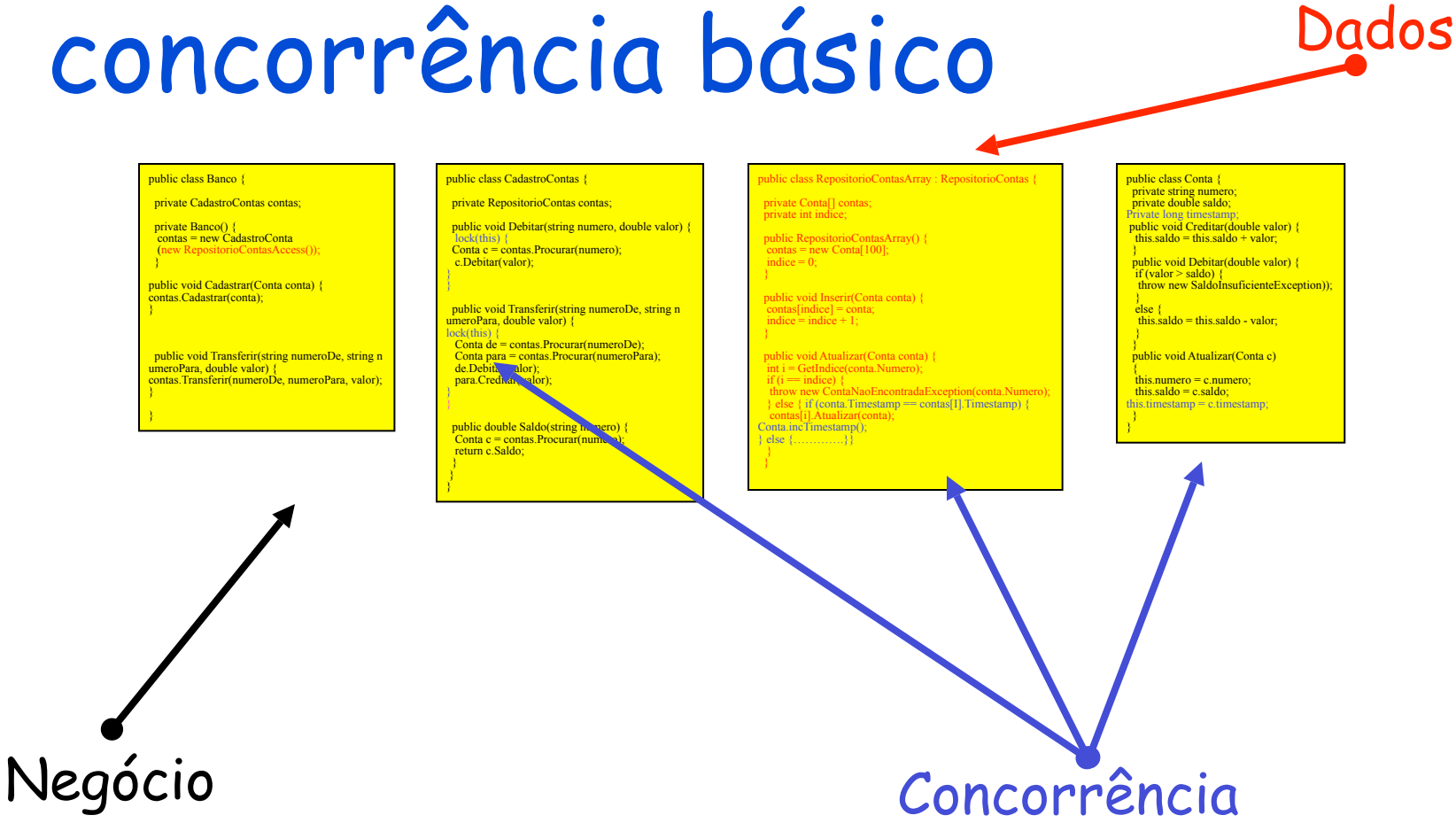
call versus execution

```
pointcut transMethods():
  execution(public * Trans.*(..));

pointcut transMethodsAlternative():
  call(public * Trans.*(..));
```



Banco com controle de concorrência básico



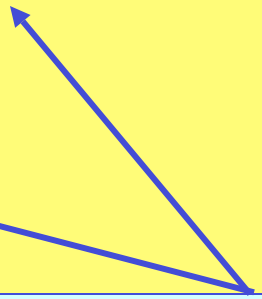
Controle de concorrência na coleção de negócio

```
public void debitar(String n, double v) {
   synchronized(this) {...
     Conta c = contas.procurar(n); ...
     c.debitar(v);...
   }
}...
```

Controle de concorrência para evitar interferências indesejadas entre os métodos da fachada

Controle de concorrência na coleção de dados

```
public void atualizar(Conta conta) {
    ...
    c = contas[i];
    if (conta.getTimestamp() ==
        c.getTimestamp()) {
      c.atualizar(conta);
      c.updateTimestamp();
    } else {...}
}...
```

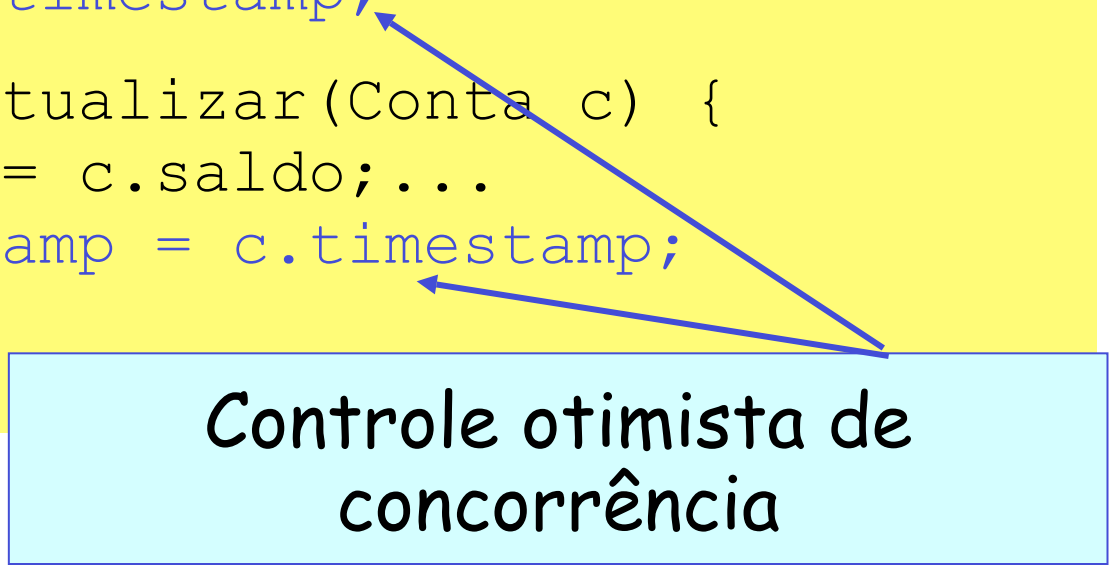


Controle de concorrência para evitar manipulação de versões desatualizadas de objetos

Controle de concorrência na classe básica

```
public class Conta {
   private double saldo;...
   private long timestamp;

   public void atualizar(Conta c) {
      this.saldo = c.saldo;...
      this.timestamp = c.timestamp;
   }...
}
```



Let's use aspects to...

modularize

another crosscutting concern

Os pointcuts podem expor o contexto dos join points

- Informações disponíveis nos join points
  - argumentos de métodos
  - objetos responsáveis pela execução
  - objetos alvo
  - variáveis de instância

Aspecto de concorrência, pointcut

Informação exposta

```
public aspect ConcurrencyAspect {
  pointcut concurrentMethods(Object t):
    ((call(void CadContas.cadastrar(Conta))
      && target(t)) || ...
    );
```

Associação de `t` com o alvo da chamada de método

Aspecto de concorrência, advice

```
void around(Object t):
    concurrentMethods(t) {
    synchronized (t) {
        proceed(t);
    }
}
```

O corpo do `around` poderá usar a informação exposta

A execução do join point interceptado deve continuar

call versus execution

```
public aspect ConcurrencyAspect {
  pointcut concurrentMethods(Object t):
    ((execution(void CadContas.cadastrar(Conta))
      && this(t)) || ...
    );
```

Aspecto de persistência, outro advice

```
void around(Banco b):
   set(CadastroContas Banco.contas) &&
   withincode(private Banco.new()) && this(b) {
      b.contas = new CadastroContas(
            new RepositorioContasAccess());
   }
```

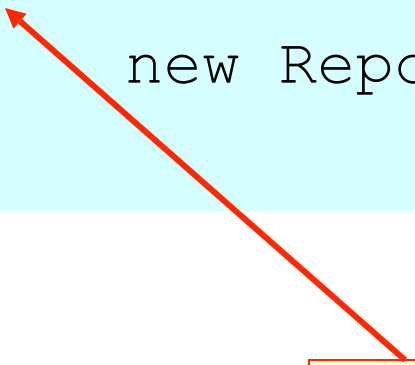
atribuições à variável de instância ou estática

escopo sintático

Associação de `b` com o objeto sendo inicializado

Quebra de encapsulamento

```
void around(Banco b):
   set(CadastroContas Banco.contas) &&
   withincode(private Banco.new()) && this(b) {
     b.contas = new CadastroContas(
           new RepositorioContasAccess());
   }
```



Variável de instância privada!
Aspecto deve ser definido como `privileged`

execution versus withinwith

```
void around(Banco b):
   set(CadastroContas Banco.contas) &&
   execution(private Banco.new()) && this(b) {
     b.contas = new CadastroContas(
           new RepositorioContasAccess());
   }
```

Mesmo call e target

não daria certo!

Basic AspectJ concepts

Paulo Borba

Informatics Center

Federal University of Pernambuco