

Aspect-oriented programming

Paulo Borba

Informatics Center

Federal University of Pernambuco

OO modularity problems and basic AO concepts

Paulo Borba

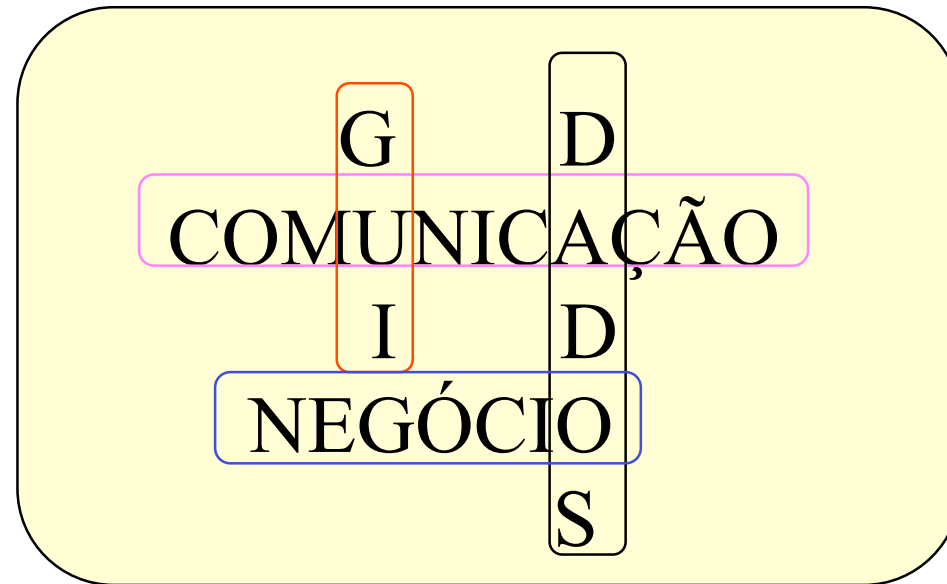
Informatics Center

Federal University of Pernambuco

Before understanding what is
aspect-oriented programming...

we will look at the
problem
that motivated its development

Projeto OO ruim

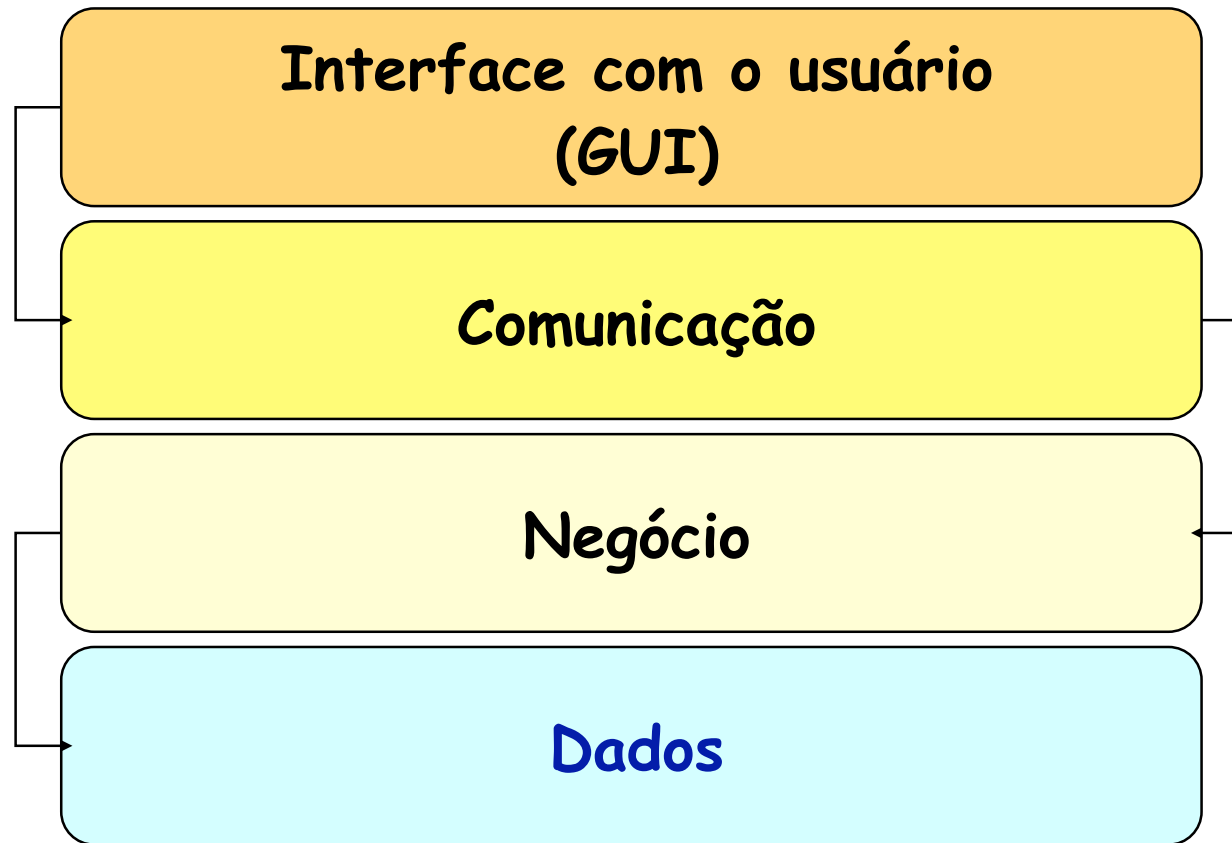


Problema: entrelaçamento de código com diferentes propósitos

Entrelaçamento

```
void cadastrar(Conta conta) {
    if (conta != null &&
        conta.valida() &&
        !this.existe(conta.getNumero())) {
        if (proxima <= maximo-1) {
            contas[proxima] = conta;
            proxima = proxima + 1;
        } else {...Espaço insuficiente...}
    } else {...Conta inválida...}
}
```

Projeto OO bom, em camadas



Arquitetura em camadas

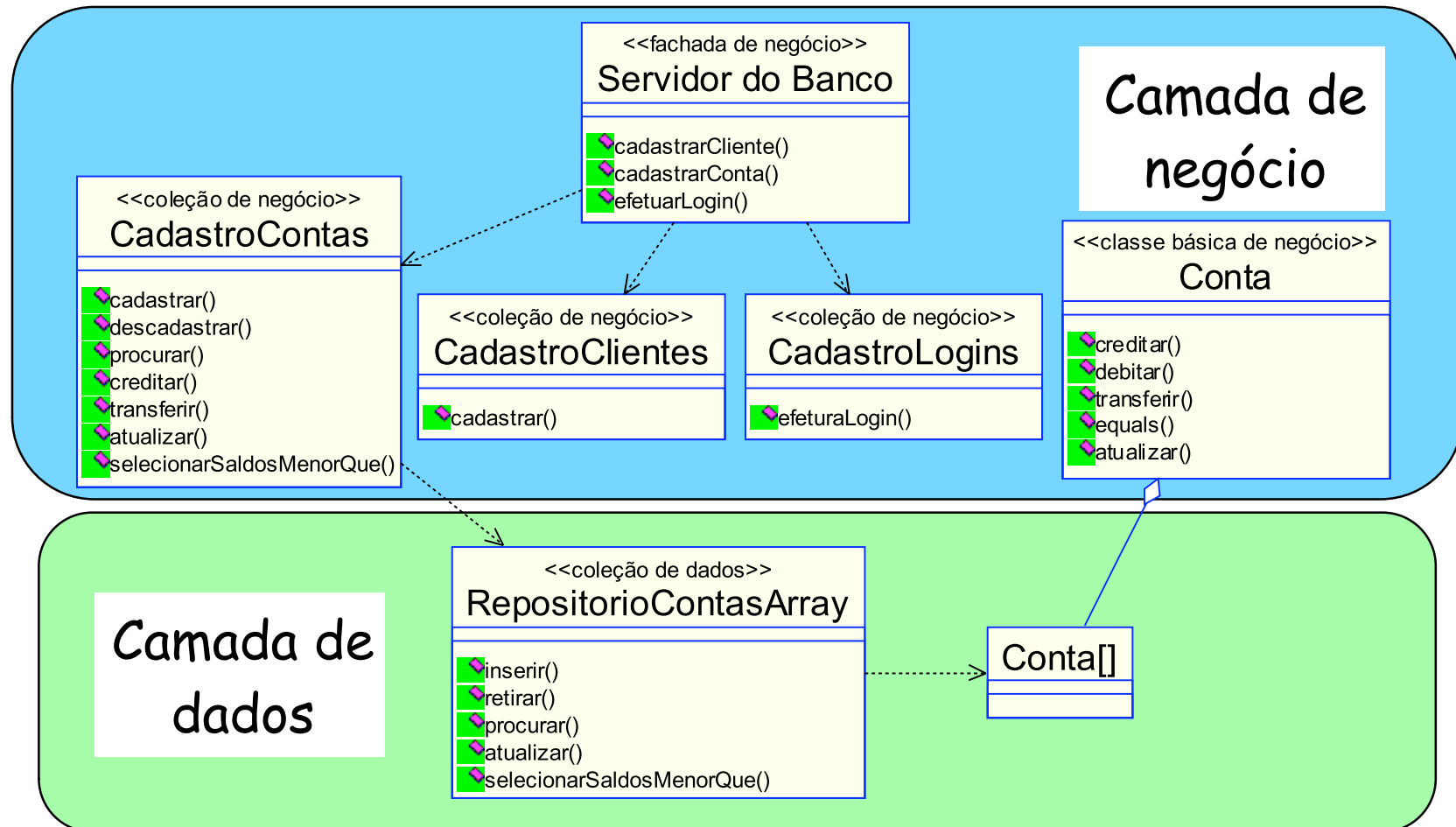
- Os vários **tipos de código** devem ser escritos separadamente
- Separação de
 - conceitos
 - preocupações
 - tipos de código

separation of concerns

Separation of concerns e modularidade

- Desenvolvimento, entendimento, e manutenção independentes
- Reusabilidade
- Mudanças em uma camada não afetam as outras, desde que as interfaces sejam preservadas

Melhor, em camadas com PDC (faltando interfaces na figura)



Boa modularização sem persistência, distribuição, ...

```
public class Programa {  
    [STAThread]  
    public static void Main(string[] args) {  
        Banco fachada = Banco.GetInstance();  
        Programa.menu(fachada);  
    }  
  
    public static void menu(Banco fachada) {  
        string numero = null;  
        double valor = 0.0;  
        Conta conta = null;  
        int opcao = 1;  
        while (opcao != 0) {  
            try {  
                System.Console.Out.WriteLine("Aperte <Enter> para continuar");  
                Util.Util.waitEnter();  
                System.Console.Out.WriteLine("\n\n\n\n\n\n\n\n");  
                System.Console.Out.WriteLine("Escolha uma das alternativas abaixo:");  
                System.Console.Out.WriteLine("1 - Cadastrar Conta");  
                System.Console.Out.WriteLine("2 - Creditar");  
                System.Console.Out.WriteLine("3 - Debitar");  
                System.Console.Out.WriteLine("4 - Transferir");  
                System.Console.Out.WriteLine("5 - Ver Saldo");  
                System.Console.Out.WriteLine("0 - Sair");  
                opcao = Util.Util.readInt();  
                switch (opcao) {
```

Interface

Negócio

Dados

if else

```
public class Banco {  
    private CadastroContas contas;  
  
    private Banco() {  
        contas = new CadastroConta  
            (new RepositorioContasAccess());  
    }  
  
    public void Cadastrar(Conta conta) {  
        contas.Cadastrar(conta);  
    }  
  
    public void Transferir(string numeroDe, string numeroPara, 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 numeroPara, 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 RepositorioContasArray : RepositorioContas {  
    private Conta[] contas;  
    private int indice;  
  
    public RepositorioContasArray() {  
        contas = new Conta[100];  
        indice = 0;  
    }  
  
    public void Inserir(Conta conta) {  
        contas[indice] = conta;  
        indice = indice + 1;  
    }  
  
    public void Atualizar(Conta conta) {  
        int i = GetIndice(conta.Numero);  
        if (i == indice) {  
            throw new ContaNaoEncontradaException(conta.Numero);  
        } else {  
            contas[i].Atualizar(conta);  
        }  
    }  
}
```

```
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;  
    }  
}
```

Mas temos problemas com persistência via JDBC

```
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 numeroPara, 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 numeroPara, 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.Saldo + "') WHERE NUMERO = '" + conta.Numero + "'";
        OleDbCommand updateCommand = new OleDbCommand(sql,DBHandler.Connection,DBHandler.Transaction);
        int 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();
    }
}
```

Código JDBC em vermelho...

Problemas com implementação OO

```
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);
        } catch (System.Exception ex) {
            Persistence.DBHandler.RollbackTransaction();
            throw ex;
        }
    }

    public void Transferir(string numeroDe, string numeroPara, double valor) {
        Persistence.DBHandler.StartTransaction();
        try {
            contas.Transferir(numeroDe, numeroPara, valor);
        } 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 numeroPara, 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 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();
    }
}
```

Entrelaçamento de código (tangling)

Espalhamento de código (scattering)

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)

Mesmo problema com Hibernate também!

```
public Blog createBlog(String name)
    throws HibernateException {...
    try {
        tx = session.beginTransaction();
        session.persist(blog);
        tx.commit();
    }
    catch (HibernateException he) {
        if (tx!=null) tx.rollback();
        throw he;
    }
    ...
}
```

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)

Tangling and scattering of...

**Concerns, elements
from a system
specification**

Tangling and scattering of...

Concerns

- Persistência
- Controle de concorrência
- Logging
- Segurança
- Eventos
- Conceitos da aplicação (Conta)
- Casos de uso
- Regras de negócio
- Feature

Relative notions

Scattering

- One concern involved
- None if unity is a complete system
- Huge if unity is command

Tangling

- Two or more concerns involved
- None if concern is Java code
- Huge if concern is assignment to variables

Crosscutting concerns...

- Afetam várias partes da aplicação
- Neste caso o termo é relativo à decomposição dominante
 - via classes no caso de OO
 - via funções no caso das linguagens funcionais

The failure of OOP

- Part of the persistence code is tangled (to the business concern) and scattered, cannot be **reused**
- Business code cannot be **reused** to work with other persistence APIs
- Business code is invaded by **changes** to persistence APIs
- Persistence policy cannot be **understood** in isolation

Persistência é um crosscutting concern

(em relação a negócio, em decomposição OO)

```
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 numeroPara, 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 numeroPara, 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.Saldo + "') WHERE NUMERO = '" + conta.Numero + "'";
        OleDbCommand updateCommand = new OleDbCommand(sql,DBHandler.Connection,DBHandler.Transaction);
        int 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();
    }
}
```

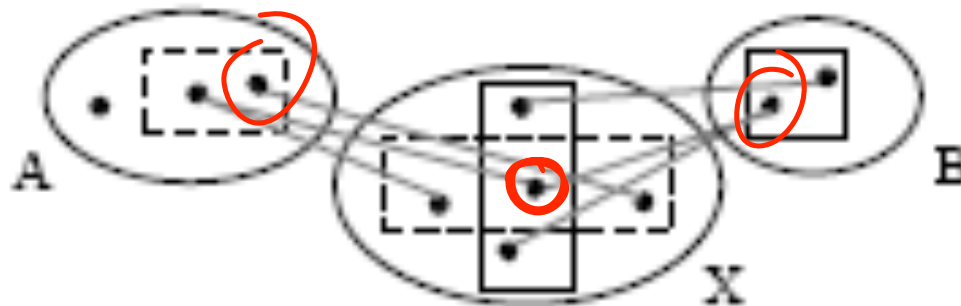
Código de persistência em vermelho... crosscuts business

Crosscutting concerns...

cannot be properly modularized by
OO constructs because their
realization affects the behavior of
several methods, possibly in several
classes, **cutting across** class
structures

A crosscuts B with respect to X...

if their projections onto X intersect, and neither of the projections is a subset of the other



From Modeling Crosscutting in Aspect-Oriented Mechanisms, Hidehiko Masuhara and Gregor Kiczales, ECOOP 2003

Improper definitions...

- a crosscutting concern is a concern that is scattered
(Filman, Elrad, Clarke, and Aksit 2005)
- ...

Aspect-oriented
programming...

is the
solution
to the illustrated problem

Aspect-oriented languages
are quite popular...

due to the promise of
modularizing
crosscutting concerns

Há várias técnicas para modularização

- Procedimentos
- Classes, herança e subtipos
- Padrões (arquitetura em camadas)
- Aspectos

Foco em modularizar
crosscutting concerns

As técnicas de modularização...

São complementares e ajudam a...

- Separar preocupações (separation of concerns)
- Aumentar modularidade (inclusive extensibilidade)
- Facilitar reuso

Tudo isso vale para aspectos (crosscutting concerns)

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);
        } 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();
    }
}
```

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 numeroPara, 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 numeroPara, 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.Saldo + "') WHERE NUMERO = '" + conta.Numero + "'";
        OleDbCommand updateCommand = new OleDbCommand(sql,DBHandler.Connection,DBHandler.Transaction);
        int 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 aspecto
de persistência

The success of AOP

- Persistence code is **localized**, can be **understood** in isolation, part of it can be **reused**
- Business code can be **reused** to work with other persistence APIs
- Business code is **not** invaded by **changes** to persistence APIs
- Less code, more code units

Weaving é usado para...

- Compor a base do sistema com os aspectos

Sistema original
chamadas locais entre A e B

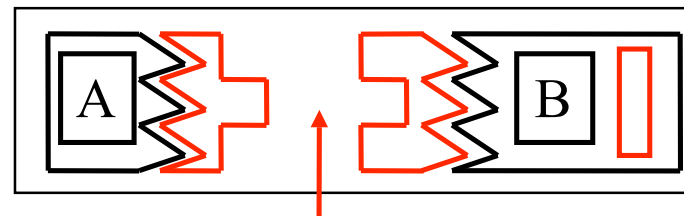


Aspectos de
distribuição

Processo de
composição

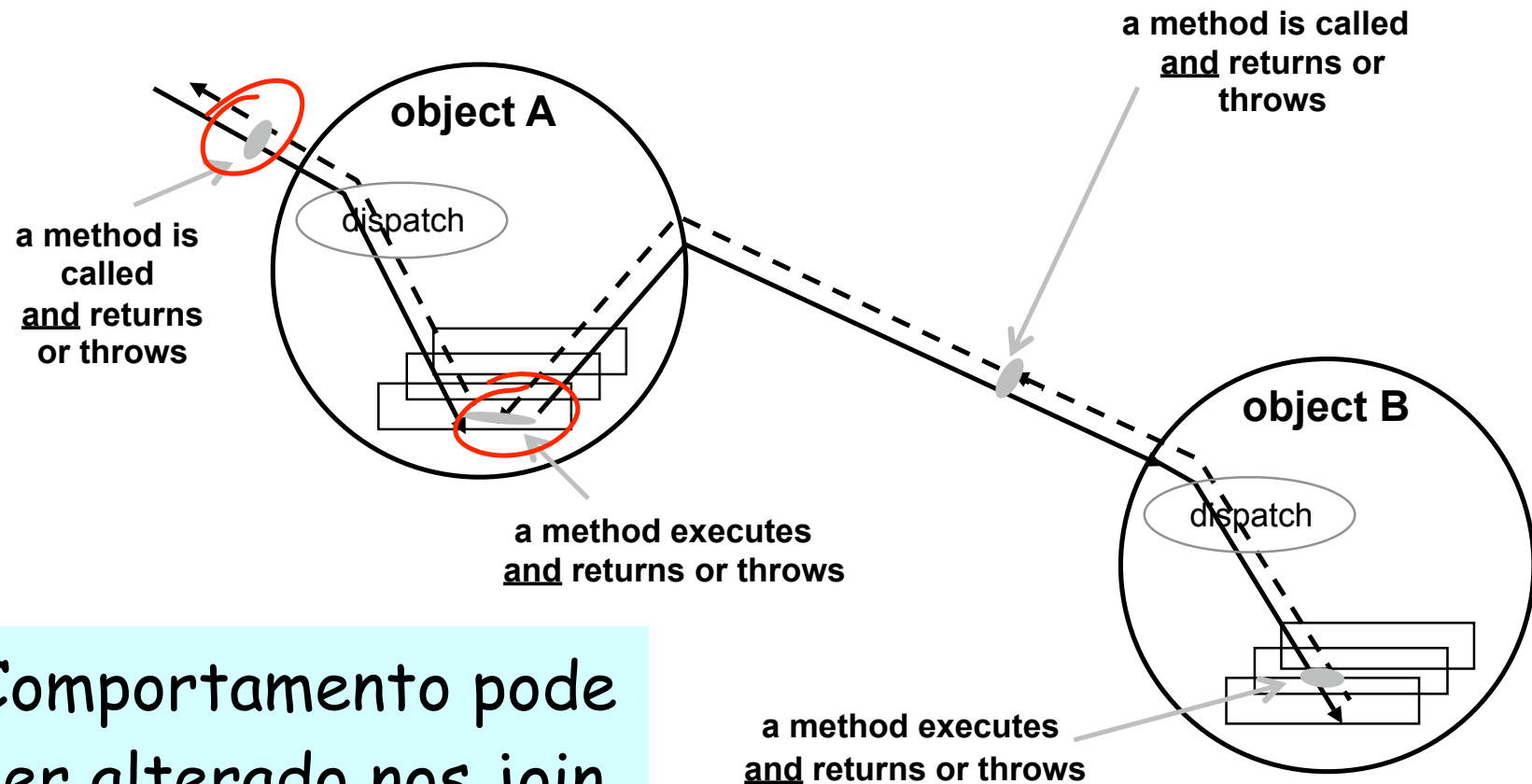


Sistema distribuído
chamadas remotas entre A e B



Tecnologia de distribuição

Composição nos join points



Comportamento pode ser alterado nos join points...

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

Identificando chamadas de métodos

nome do
pointcut

```
pointcut printCall() :  
    call(public void *.println(String)) ;
```

identifica
chamadas de

...

método
println de
qualquer classe

com
argumento
string

Advice específica comportamento extra nos

- Define código adicional que deve ser executado...
 - **before**
 - **after**
 - **after returning**
 - **after throwing**
 - **ou around**

join points

Alterando o comportamento de chamadas de métodos

após...

qualquer chamada a
Write dentro de
HelloWorld

```
after(): printCall() && within(HelloWorld)
{
    System.out.println(" AOP World");
}
```


a ação especificada
será executada

Aspectos agrupam pointcuts, advices, propriedades, etc.

```
aspect HelloAOPWorld {  
    pointcut printCall():  
        call(public void *.print(String));  
    after():  
        printCall() && within(HelloWorld) {  
        System.out.println(" AOP World!");  
    }  
}
```

Hello AOP World!

```
public class HelloWorld {  
    public static void Main(string[] args) {  
        System.out.println("Hello");  
    }  
}
```



Chamada afetada pelo advice, caso
HelloAOPWorld tenha sido composto
com HelloWorld

OO modularity problems and basic AO concepts

Paulo Borba

Informatics Center

Federal University of Pernambuco