

# Software product lines

Paulo Borba

Informatics Center

Federal University of Pernambuco



# Variability implementation mechanisms, part II

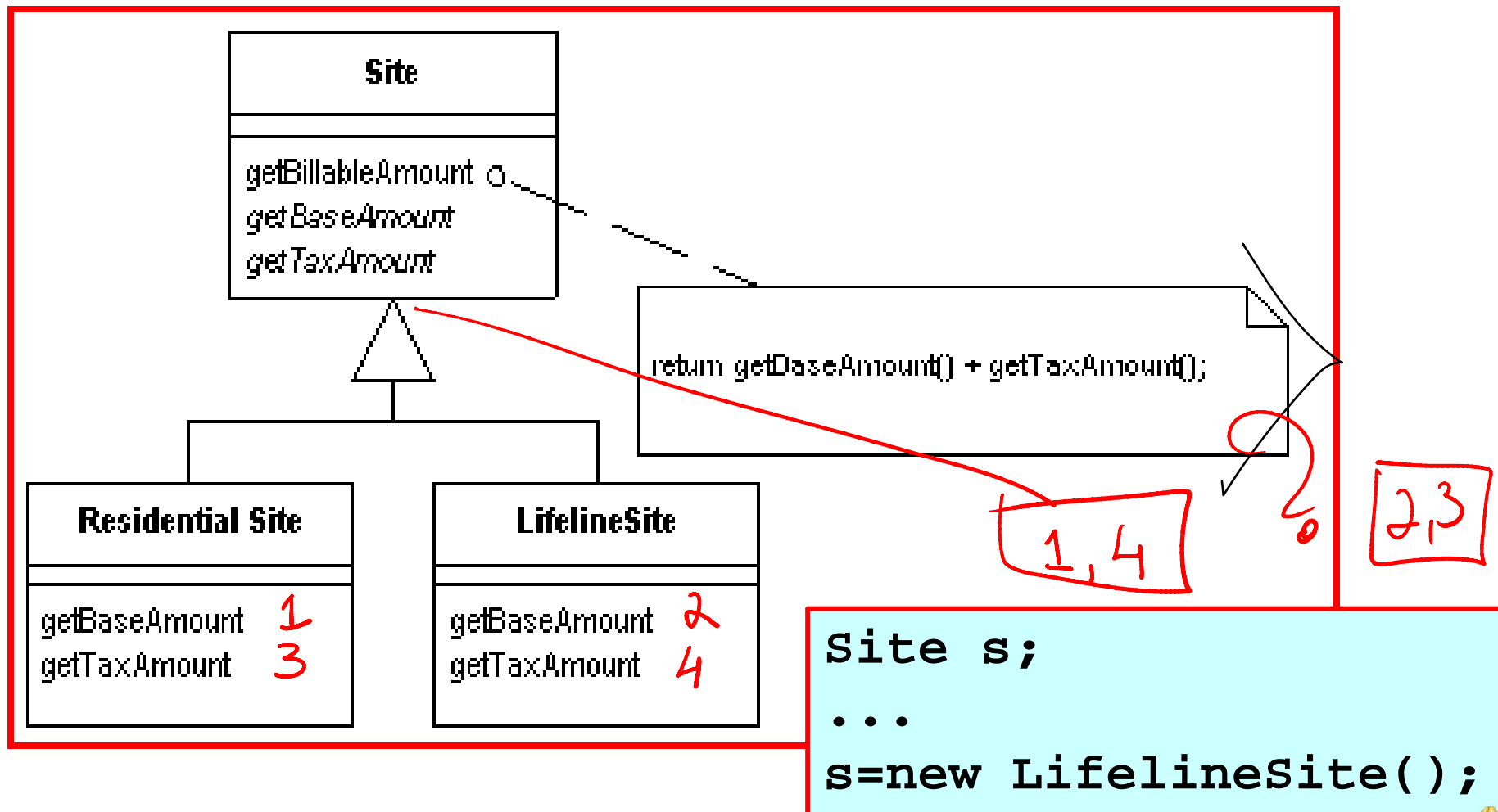
Paulo Borba

Informatics Center

Federal University of Pernambuco



# Subtype polymorphism...



# and OO patterns

```
abstract class TestCase {...  
    void buildTest() {...  
        buildPreconditions();...  
        buildProcedures();...  
    }  
}
```

```
class TC5 extends TestCase {...  
    void buildProcedures() {...  
        launchApp(Phone.SOUNDS);  
        createMelody(...);...  
    }  
}
```

TCs



# Template method variation

```
abstract class TestCase { ...  
    void buildTest() { ...  
        buildPreconditions(); ...  
        buildProcedures(); ...  
    }  
}
```

```
class TC5 extends TestCase { ...  
    void buildProcedures() { ...  
        launchApp(Phone.SOUNDS);  
        createMelody(...); ...  
    }  
}
```

```
class TC5Bluetooth extends TC5 {  
    void buildProcedures() {  
        super.buildProcedures();  
        setBluetooth(ON); ...  
    }  
}
```

```
class TC5Transflash extends TC5 {  
    void buildProcedures() {  
        super.buildProcedures();  
        storeMultimedia(...); ...  
    }  
}
```



# SP core assets

Java files,  
abstract

```
abstract class TestCase { ...  
    void buildTest() { ...  
        buildPreconditions(); ...  
        buildProcedures(); ...  
    }  
}  
  
class TC5 extends TestCase { ...  
    void buildProcedures() { ...  
        launchApp(Phone.SOUNDS);  
        createMelody(...); ...  
    }  
}
```

Java files,  
concrete

```
class TC5Bluetooth extends TC5 {  
    void buildProcedures() {  
        super.buildProcedures();  
        setBluetooth(ON); ...  
    }  
}  
  
class TC5Transflash extends TC5 {  
    void buildProcedures() {  
        super.buildProcedures();  
        storeMultimedia(...); ...  
    }  
}
```

Java files,  
decision

...  
new TC5Bluetooth

...  
new TC5Transflash

Makefiles



# PDC, bridge variation...

```
class CadastroContas {
    private RepositorioContas contas;...

    public void cadastrar(Conta conta) {
        if (conta != null) {
            String numero = conta.getNumero();
            if (!contas.existe(numero)) {
                contas.inserir(conta);
            }
        }
    }
}
```



# Variation is whole data structure

```
interface RepositorioContas {  
    void inserir(Conta conta);  
    Conta procurar(String numero);  
    boolean existe(String numero);  
}
```

```
class ArrayContas implements  
    RepositorioContas {...}
```

```
class VectorContas implements  
    RepositorioContas {...}
```





# Analyzing subtype polymorphism

- Variation point
  - supertypes (methods) in the program text
- Variation
  - subtypes (including methods and fields)
- Binding time
  - compilation time, based on makefile choice
- Mechanism
  - subtype definition, variation modularization



# Decision mechanisms for subtype polymorphism

- Makefiles (different mains or factories)
- Makefiles + property\_files + conditionals
- Makefiles + property\_files + reflection
- Input + conditionals
- Input + reflection

D  
e  
v  
e  
l  
o  
p  
e  
r

U  
s  
e  
r



# Environment decision, with dependency injection...

```
class CadastroContas {  
    private RepositorioContas contas;  
    void setRepositorio(RepositorioContas r) {  
        contas = r;  
    }  
    ...  
}
```

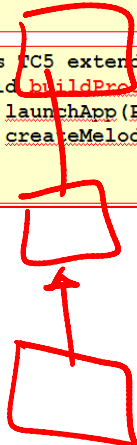
Not a developer decision,  
nor a user decision!



# Mixins

```
abstract class TestCase { ...  
    void buildTest() { ...  
        buildPreconditions(); ...  
        buildProcedures(); ...  
    }  
}
```

```
class TC5 extends TestCase { ...  
    void buildProcedures() { ...  
        launchApp(Phone.SOUNDS);  
        createMelody(...); ...  
    }  
}
```



```
cclass TC5Bluetooth extends TC5 {  
    void buildProcedures() {  
        super.buildProcedures();  
        setBluetooth(ON); ...  
    }  
}
```

```
cclass TC5Transflash extends T5 {  
    void buildProcedures() {  
        super.buildProcedures();  
        storeMultimedia(...); ...  
    }  
}
```

```
cclass TC5TransBlue extends  
    TC5Transflash & TC5Bluetooth {}
```



# Components

- Improvement over subtype polymorphism
  - not by defining subtypes of abstract classes, but of interfaces
  - clientship + subtype polymorphism
  - deployment independence
  - service location and management
  - class visibility and versions



# OSGi startup and shutdown notification

```
package com.javaworld.sample.helloworld;
import org.osgi.framework.BundleActivator;
import org.osgi.framework.BundleContext;
public class Activator implements BundleActivator {
    public void start(BundleContext context) throws Exception {
        System.out.println("Hello world");
    }
    public void stop(BundleContext context) throws Exception {
        System.out.println("Goodbye World");
    }
}
```



# OSGi Metadata

```
Manifest-Version: 1.0
Bundle-ManifestVersion: 2
Bundle-Name: HelloWorld Plug-in
Bundle-SymbolicName: com.javaworld.sample.HelloWorld
Bundle-Version: 1.0.0
Bundle-Activator: com.javaworld.sample.helloworld.Activator
Bundle-Vendor: JAVAWORLD
Bundle-Localization: plugin
Import-Package: org.osgi.framework;version="1.3.0"
```



# Code generation and transformation

Meta-variable  
for class names

```
class #class [extends #superClass]
{
    public #type #field;
    #fields;
    #methods;
}
```

Optional  
matching

Meta-variable for  
field declarations





# Semantics-aware matching

```
class #class [extends #superClass] {  
    public #type #field;  
    #fields;  
    #methods;  
}
```



```
class Account {  
    private String number;  
    public double balance;  
    ...  
}
```



# Transformations

```
class #class [extends #superClass] {  
    public #type #field;  
    #fields; #methods;  
}
```

LHS

```
class #class [extends #superClass] {  
    private #type #field;  
    #fields; #methods;  
}
```

RHS

Pre-conditions...



# Iterative expressions

```
class #class [extends #superClass] {  
  #fields; #methods;  
  for all #f in #fields {  
    let #n = <#f.getName()>;  
    #t = <#f.getType()>;  
    in  
    #t <#n.addPrefix("get")>() {  
      return this.#n;  
    }  
  }  
}
```



# Transformations

#classes

```
public class Person implements java.io.Serializable {  
    private String nome; ...  
    public Person(String nome, ...) {  
        this.nome = nome; ...  
    }  
    public String getNome() {  
        return nome;  
    } ...  
}
```

```
public class Person implements java.io.Serializable {  
    private String nome; ...  
    public Person(String nome, ...) {  
        this.nome = nome; ...  
    }  
    public String getNome() {  
        return nome;  
    } ...  
}
```



```
public interface IFacade extends java.rmi.Remote {  
    public void updateComplaint complaint)  
        throws TransactionException, RepositoryException,  
        ObjectNotFoundException, ObjectNotValidException,  
        RemoteException;  
    ...  
}
```

```
public class Person implements java.io.Serializable {  
    private String nome; ...  
    public Person(String nome, ...) {  
        this.nome = nome; ...  
    }  
    public String getNome() {  
        return nome;  
    } ...  
}
```

```
public class Person implements java.io.Serializable {  
    private String nome; ...  
    public Person(String nome, ...) {  
        this.nome = nome; ...  
        private X x = new X();  
    }  
}
```

```
Class X {  
    public String getNome() {  
        return nome;  
    } ...  
}
```

#classes

Pre-conditions...



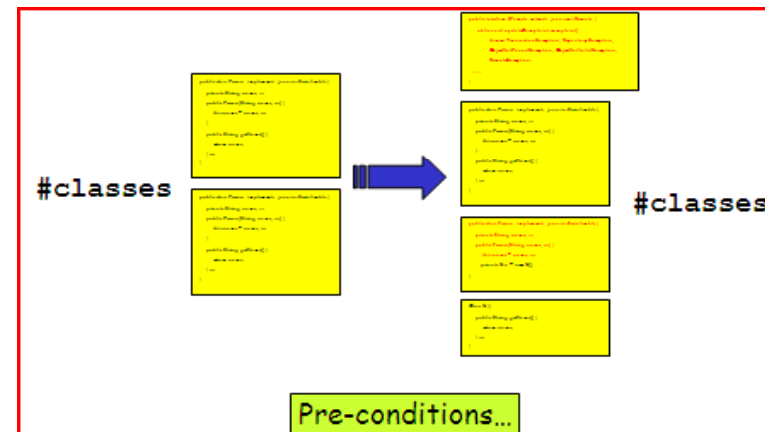
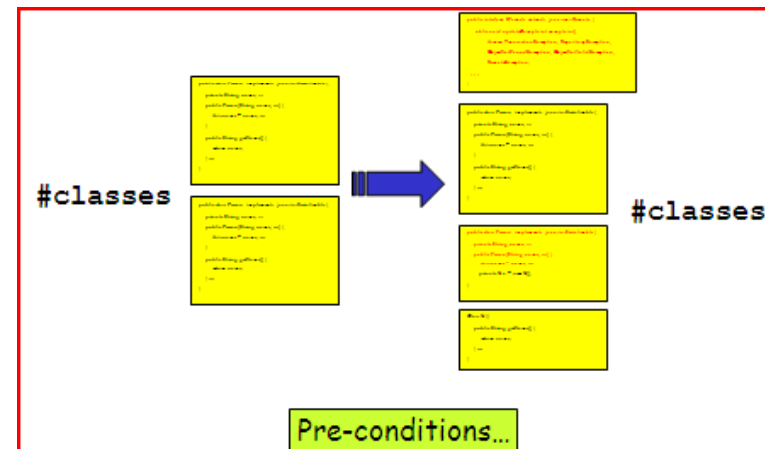
# CGT core assets

Java files  
(common)


```
class Account {  
    private String number;  
    public double balance;  
    ...  
}
```

Makefiles

Transformation files



# Analyzing CGT

- Variation point
  - any element (not point) in the program text
- Variation
  - elements (including methods and fields)
- Binding time
  - precompilation time, based on makefile choice
- Mechanism
  - instantiation, transformation, generation 

# Aspects core assets

## Java files

```
class Account {  
    private String number;  
    public double balance;  
    ...  
}
```

## Makefiles

## Aspects

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

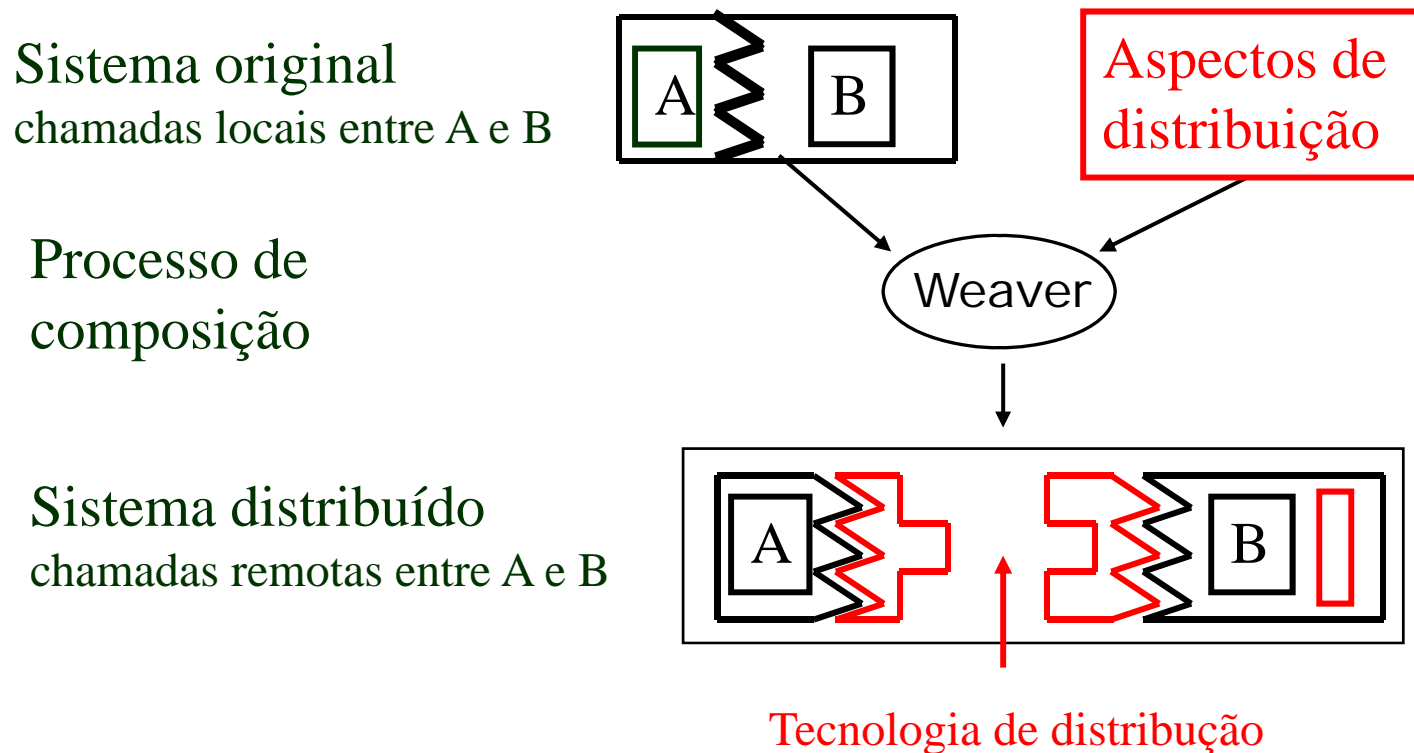
```
aspect HelloAOPWorld {  
    pointcut printCall():  
        call(public void *.print(String));  
    after():  
        printCall() && within(HelloWorld) {  
            System.out.println(" AOP World!");  
        }  
}
```

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



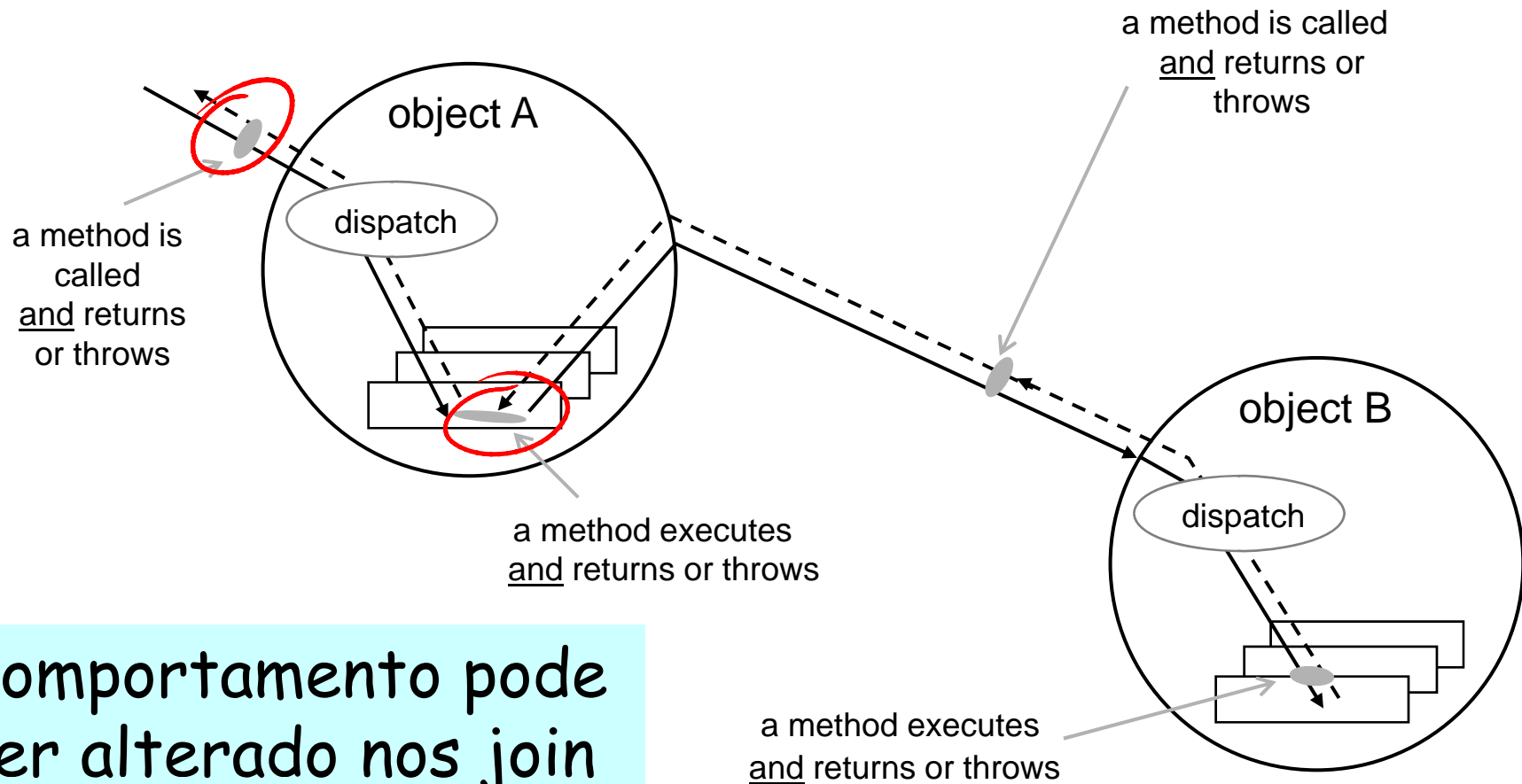
# Weaving é usado para...

- Compor a base do sistema com os aspectos





# 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
  - `&&`, `||` e `!`



# 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 join points

- 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



# Changing class hierarchy

```
declare parents: HealthWatcherFacade  
implements IFacade;
```

```
declare parents: Complaint || Person  
implements java.io.Serializable;
```

```
public static void  
HealthWatcherFacade.main(String[] args){  
try {...  
    java.rmi.Naming.rebind("/HW");  
} catch ...  
}
```

Alterando a hierarquia de tipos

Adicionando o método main na classe fachada



# Aspectos agrupan pointcuts, advices, 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





# Aspects for variations

```
abstract class TestCase { ...
    void buildTest() { ...
        buildPreconditions(); ...
        buildProcedures(); ...
    }
}

class TC5 extends TestCase { ...
    void buildProcedures() { ...
        launchApp(Phone.SOUNDS);
        createMelody(...); ...
    }
}
```

```
privileged aspect Bluetooth {
    pointcut bluetooth(BaseTestCase test):
        execution(void TC5.buildProcedures(..))
        && this(test);
    after(BaseTestCase test):bluetooth(test){
        setBluetooth(ON); ...
    }
}
```



# Variations interaction, and precedence

```
privileged aspect Transflash {  
    pointcut transflash(BaseTestCase test):  
        execution(void TC5.buildProcedures(..))  
        && this(test);  
    after(BaseTestCase test):transflash(test){  
        storeMultimedia(...);...  
    }  
}
```

```
public aspect TransflashBluetooth {  
    declare precedence: Transflash,Bluetooth;  
}
```



# Analyzing aspects

- Variation point
  - joinpoints
- Variation
  - behavior, methods and fields, hierarchy
- Mechanism
  - variation separation, behavior modification, structure transformation



# Aspects and binding time

- Compilation time, based on makefile choice
  - but can be used to implement variation 1, 2, and dynamic decision about 1 and 2
- Execution time
  - with dynamic aspect loading
  - based on user, developer, or environment choice, depending on decision mechanism

conditionals +



# Variability implementation mechanisms, part II

Paulo Borba

Informatics Center

Federal University of Pernambuco