

פיתוח מערכות תוכנה בשפת Java

חתכי רוחב בתוכנה

Crosscutting Concerns

אוהד ברזילי

בית הספר למדעי המחשב
אוניברסיטת תל אביב

מודולריות בתכנות מונחה עצמים

■ לתכנות מונחה עצמים יש החסרונות שלו וצריך להיות ערים להם

■ חסרון בולט קשור לניהול של חתכי רוחב (crosscutting concerns) במערכת תוכנה – הפוגע במודולריות של המערכת

■ ההבנה כי למודולריות נכונה של תוכנה יש השפעה על סיבוכיות התוכנה והבנתה מתוארת ב- **On the criteria to be used in decomposing systems into modules** מאת David Parnas משנת 1972

Separation of Concerns

■ המונח **Separation of Concerns** נתבע ע"י Edsger W. Dijkstra בשנת 1974 במאמר On the role of scientific thought

it from that viewpoint only; we also know that it should be efficient and we can study its efficiency on another day, so to speak. In another mood we may ask ourselves whether, and if so: why, the program is desirable. But nothing is gained --on the contrary!-- by tackling these various aspects simultaneously. It is what I sometimes have called "the separation of concerns", which, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts, that I know of. This is what I mean by "focussing one's attention upon some aspect": it does not mean ignoring the other aspects, it is just doing justice to the fact that from this aspect's point of view, the other is irrelevant. It is being one- and multiple-track minded simultaneously.

דוגמא

- כחלק מהפתולוגיה של מערכת תוכנה מוכוונת עצמים אנו מגדירים מחלקות לייצוג Core Concerns (או Business Logic) אולם אנו מזניחים עניינים (הבטים) אחרים
- נניח שכתבנו תוכנה שעושה משהו
- במערכת התוכנה נמצא את המחלקה `SomeBusinessClass` עם השרות `someOperation`
- למשל המחלקה `BankAccount` עם השרות `withdraw` (רק לצורך הדוגמא – הדבר תקף כמעט לכל תוכנה אמיתית)

The wrong way

```
public class SomeBusinessClass extends OtherBusinessClass {  
    // Core data members  
    // Override methods in the base class  
    public void someOperation(OperationInformation info) {  
        // ==== Perform the core operation ====  
    }  
    ...  
}
```

The wrong way(2)

- But what about logging capabilities ?

```
public class SomeBusinessClass extends OtherBusinessClass {  
  
    // Core data members  
    ...Log stream ;  
  
    // Override methods in the base class  
  
    public void someOperation(OperationInformation info){  
        ...log the start of operation  
        // ==== Perform the core operation ====  
        ...log the completion of operation  
    }  
}
```

The wrong way(3)

- **Actually, we want it multithreaded...**

```
public class SomeBusinessClass extends OtherBusinessClass {  
  
    // Core data members  
    ...Log stream ;  
    // Override methods in the base class  
  
    public void someOperation(OperationInformation info) {  
        ...lock the object - thread safety  
        ...log the start of operation  
        // ==== Perform the core operation ====  
        ...log the completion of operation  
        ...unlock the object  
    }  
}
```

The wrong way(4)

- Who enforces your contract ?

```
public class SomeBusinessClass extends OtherBusinessClass {  
  
    // Core data members  
    ...Log stream ;  
    // Override methods in the base class  
  
    public void someOperation(OperationInformation info) {  
        ...ensure info satisfies contract  
        ...lock the object - thread safety  
        ...log the start of operation  
        // ==== Perform the core operation ====  
        ...log the completion of operation  
        ...unlock the object  
    }  
}
```


The wrong way(5)

■ Authorization ? Authentication ?

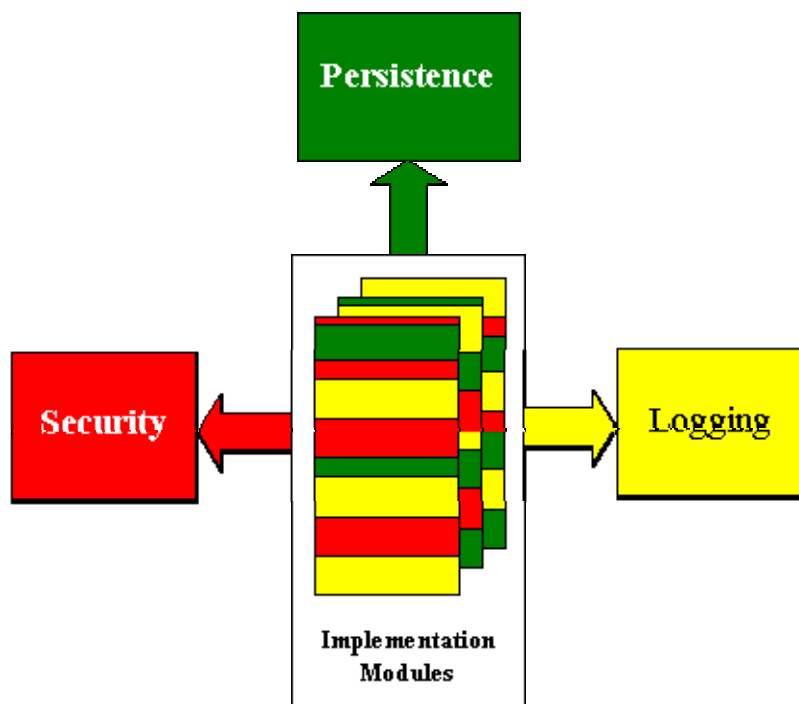
```
public class SomeBusinessClass extends OtherBusinessClass {  
  
    // Core data members  
    ...Log stream ;  
    // Override methods in the base class  
  
    public void someOperation(OperationInformation info) {  
        ...ensure authorization  
        ...ensure info satisfies contract  
        ...lock the object - thread safety  
        ...log the start of operation  
        // ==== Perform the core operation ====  
        ...log the completion of operation  
        ...unlock the object  
    }  
}
```

The wrong way(6)

■ Persistence ? Cache consistency ?

```
public class SomeBusinessClass extends OtherBusinessClass {  
  
    // Core data members  
    ...Log stream ;  
    ...cache_update_status ;  
    // Override methods in the base class  
  
    public void someOperation(OperationInformation info) {  
        ...ensure authorization  
        ...ensure info satisfies contract  
        ...lock the object - thread safety  
        ...ensure cache is up to date  
        ...log the start of operation  
        // ==== Perform the core operation ====  
        ...log the completion of operation  
        ...unlock the object  
    }  
    public void save(PersitanceStorage ps) {...}  
    public void load(PersitanceStorage ps) {...}  
}
```

מה קיבלנו?



בלאגן בשתי רמות:

■ ברמת המיקרו (השרות הבודד):

- Code Tangling
- הוא כבר לא עושה "רק משהו אחד" - לא מודולרי
- ראו תרשים <=

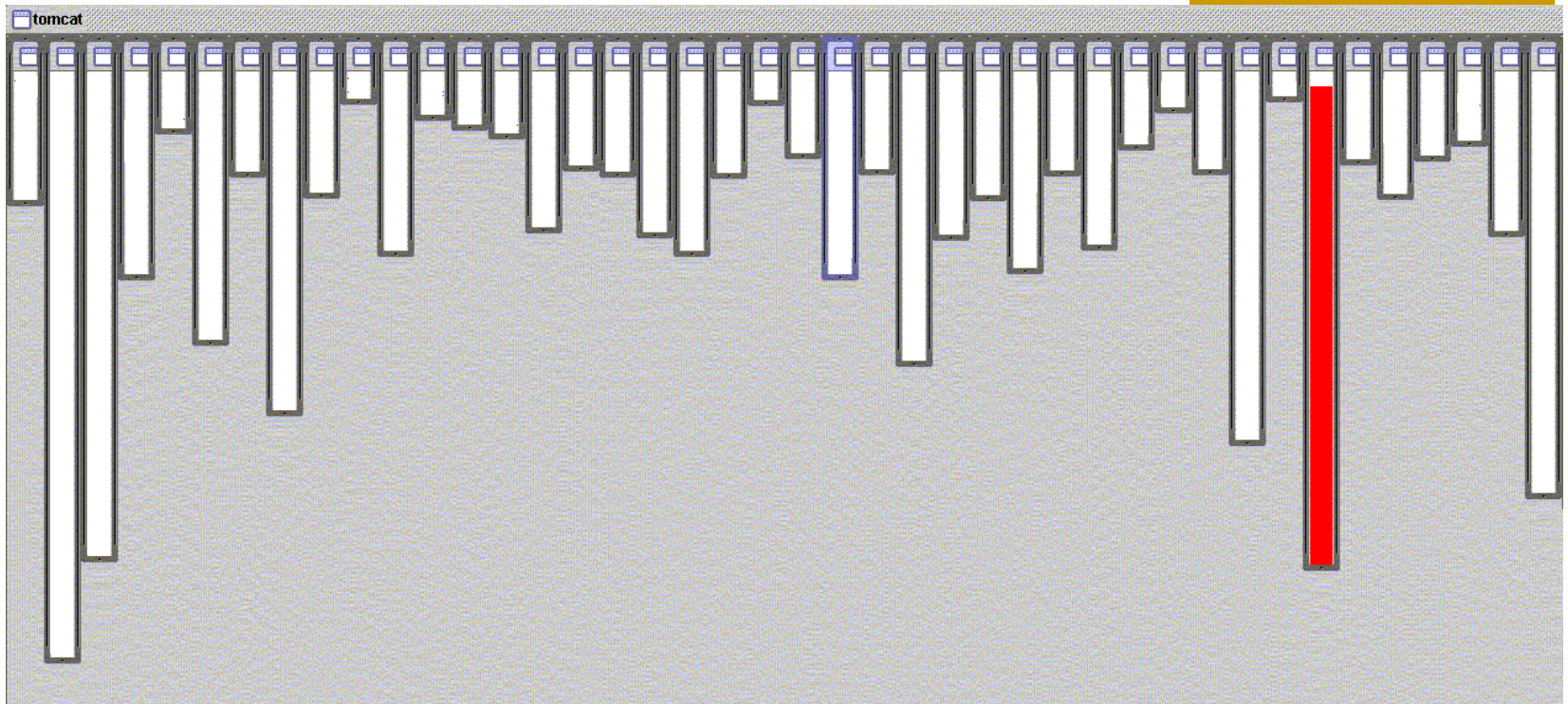
■ ברמת המאקרו (מערכת התוכנה):

- Code Scattering
- שכפול קוד, קטעי קוד קשורים אינם מופיעים יחד
- ראו תרשימים גם בשקפים הבאים

■ שבירת המודולריות נוצרת בגלל אופי הספק-לקוח של תכנות מונחה עצמים

good modularity

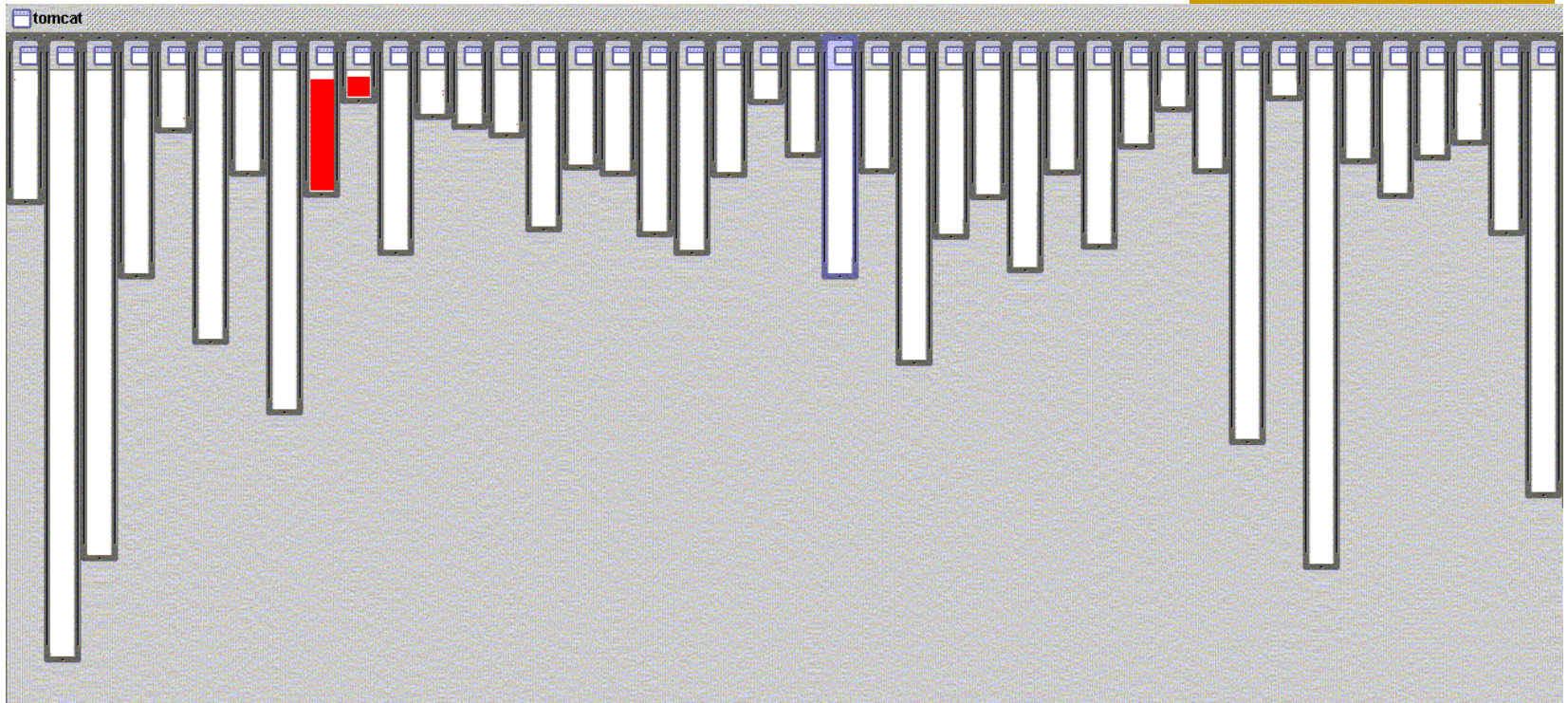
XML parsing



- XML parsing in org.apache.tomcat
 - red shows relevant lines of code
 - nicely fits in one box

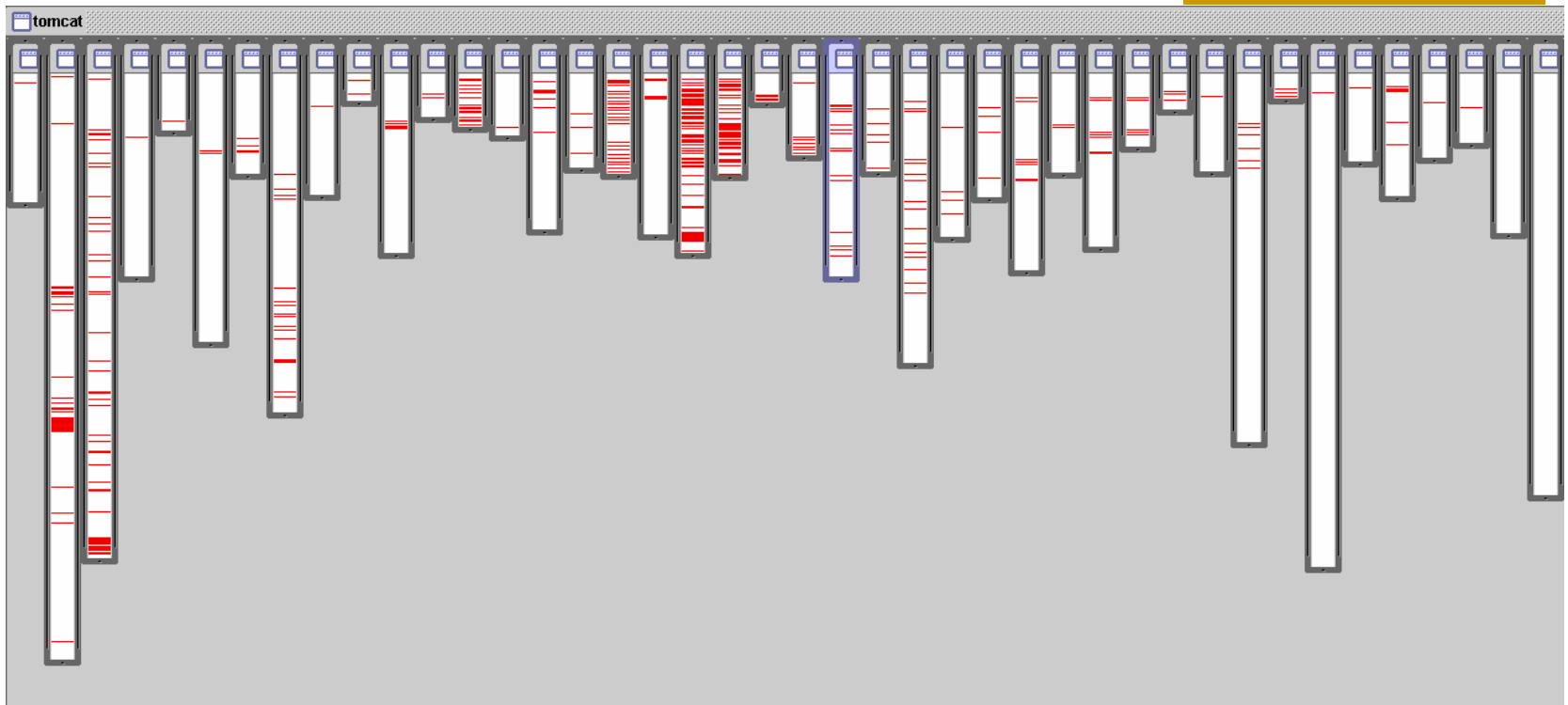
good modularity

URL pattern matching



- URL pattern matching in org.apache.tomcat
 - red shows relevant lines of code
 - nicely fits in two boxes (using inheritance)

logging is not modularized...



- where is logging in org.apache.tomcat
 - red shows lines of code that handle logging
 - not in just one place
 - not even in a small number of places

אילו רק יכולנו...

ApplicationSession

```
ApplicationSession.java
...
public ApplicationSession() {
    // ...
}
...
public void setAttribute(String name, Object value) {
    // ...
}
...
public Object getAttribute(String name) {
    // ...
}
...
}
```

StandardSession

```
StandardSession.java
...
public StandardSession() {
    // ...
}
...
public void setAttribute(String name, Object value) {
    // ...
}
...
public Object getAttribute(String name) {
    // ...
}
...
}
```



ServerSession

```
ServerSession.java
...
public ServerSession() {
    // ...
}
...
public void setAttribute(String name, Object value) {
    // ...
}
...
public Object getAttribute(String name) {
    // ...
}
...
}
```

SessionInterceptor

```
SessionInterceptor.java
...
public SessionInterceptor() {
    // ...
}
...
public void intercept(...) {
    // ...
}
...
}
```

StandardManager

```
StandardManager.java
...
public StandardManager() {
    // ...
}
...
public void createSession(...) {
    // ...
}
...
}
```

StandardSessionManager

```
StandardSessionManager.java
...
public StandardSessionManager() {
    // ...
}
...
public void createSession(...) {
    // ...
}
...
}
```

ServerSessionManager

```
ServerSessionManager.java
...
public ServerSessionManager() {
    // ...
}
...
public void createSession(...) {
    // ...
}
...
}
```

שבירת המודולריות

■ נזכיר 3 גישות לפתרון הבעיה:

■ מעבר לשימוש ברכיבים (components) במקום עצמים

■ כגון: Servlets או EJB's – נקדיש להם שיעור בהמשך הקורס

■ חרון: Domain Specific Framework

■ פתרונות ברמת שפת התכנות ותבניות העיצוב:

■ כגון: Mixin או Dynamic Proxy – דוגמא בהמשך

■ חרון: דורש "תחזוקה ידנית" של העיצוב

■ מעבר לשפת תכנות בפרדיגמה התומכת ביחסים נוספים בין מחלקות

■ כגון: AspectJ או שפת E – דוגמא בהמשך

■ חרון: לימוד שפה חדשה

Proxy Design Pattern

Proxy – יצירת פונדקאי או שומר מקום לעצם כדי לבצע הפשטה על הגישה אליו ■

■ לצורכי יעילות, פיקוח, מודולריות ועוד...

■ לדוגמא:

■ תמונות "כבדות" במסמך, מצביעים חכמים

■ Access Proxy

■ Firewall Proxy

■ Virtual Proxy (Lazy Proxy)

■ Remote Proxy

■ Synchronization Proxy

■ הרעיון ממומש במערכות תוכנה ובספריות רבות

■ Java מספקת את המחלקה `InvocationHandler` המאפשרת לנו להגדיר Proxy משלנו

```
/** A Proxy that intercepts String arguments & converts them to  
uppercase Then, as usual, it will forward method calls to the  
enclosed object */
```

```
import java.util.*;
```

```
import java.lang.reflect.*;
```

```
class UppercaseProxy implements InvocationHandler {  
    private Object obj;
```

```
    public UppercaseProxy(Object obj) {  
        this.obj=obj;  
    }  
}
```

```
    public Object invoke(Object proxy, Method m, Object[] args)  
        throws Throwable {  
        if (args!=null){  
            for (int i = 0; i < args.length; i++) {  
                if ( args[i] instanceof String) {  
                    String s = (String)args[i];  
                    args[i] = s.toUpperCase();  
                }  
            }  
        }  
        return m.invoke(obj, args);  
    }  
}
```

```

/** You can now wrap this proxy around any object (e.g: List),
    provided you only work through interfaces */

public class ProxyTest {
    public static void main(String[] args) throws Exception {

        ArrayList myList=new ArrayList();

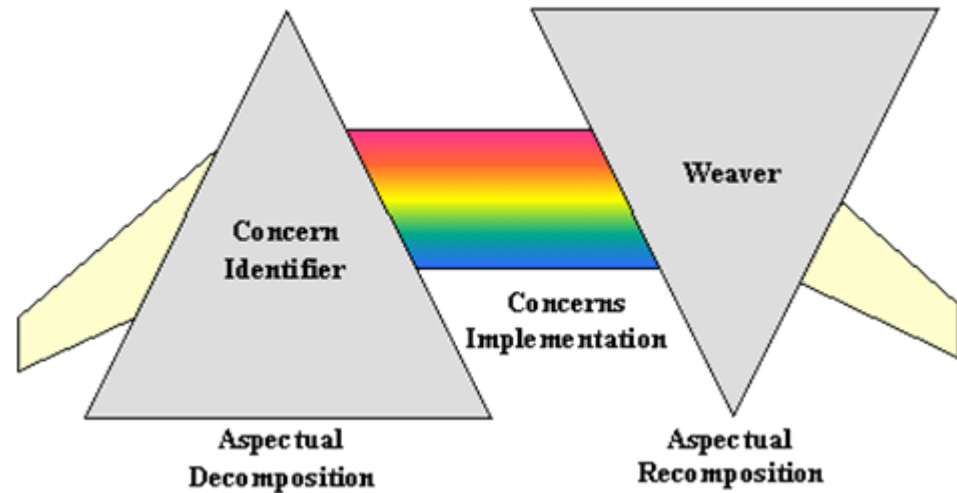
        // Create a proxy that wraps myList and implements
        // interface List:
        Object proxy = Proxy.newProxyInstance(
            java.util.List.class.getClassLoader(),
            new Class[] {java.util.List.class}, // interfaces
            new UppercaseProxy(myList)); // wrapped obj

        // Add items to list, through the proxy:
        List pList= (List) proxy;
        pList.add("Aa");
        pList.add("bbb");
        System.out.println(pList);
    }
}

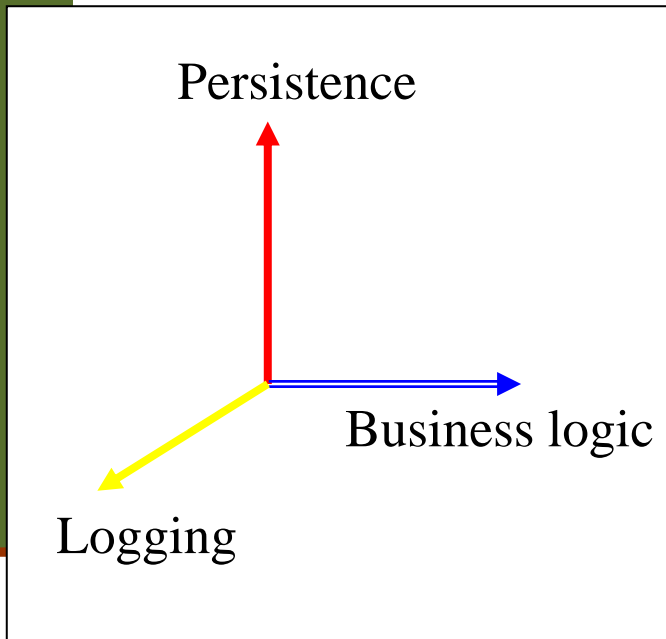
```

AspectJ Terminology

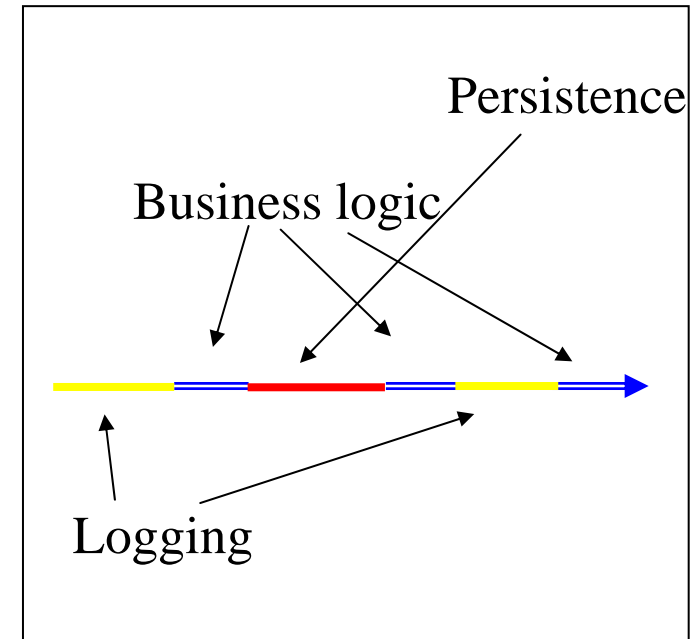
- A Weaver
- Join point
- Pointcut
- Advice
- Aspect



Weaving orthogonal concerns



Implementation mapping



■ HelloWorld.java

```
public class HelloWorld {  
  
    public static void say(String message) {  
        System.out.println(message);  
    }  
}
```

■ Test.java

```
public class Test {  
  
    public static void main(String[] args){  
        HelloWorld.say("Hello World");  
    }  
}
```

➤ `ajc HelloWorld.java Test.java`

➤ `java Test`

Hello World

■ MannersAspect.java

```
public aspect MannersAspect {

    pointcut saying() :
        call(public static void HelloWorld.say*(..));

    before() : saying() {
        System.out.print("Good day! ");
    }

    after() : saying() {
        System.out.println("Thank you!");
    }
}
```

➤ `ajc HelloWorld.java MannersAspect.java Test.java`

➤ `java Test`

Good day! Hello World

Thank you!

Framework Support

■ בשיעורים הבאים נציג את J2EE Framework של Sun אשר מתמודדת עם crosscutting concerns בפיתוח Web Applications