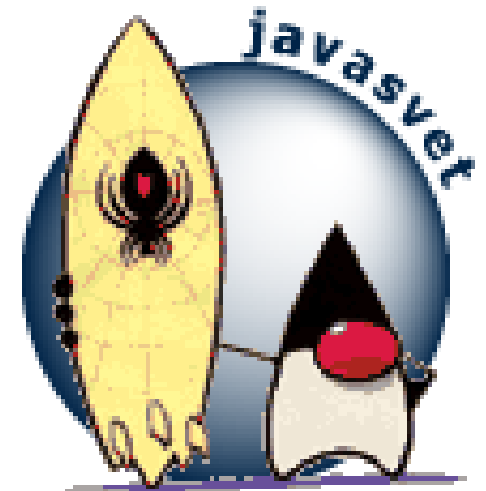




***1. Sastanak JavaSvet Zajednice***  
***14. Avgust 2004.***

# Aspektno programiranje u Javi

AOP + AspectJ



# Posledice nemodularnosti ?

- slabo praćenje toka izvršavanja
- smanjenja produktivnost
- smanjen code reuse
- smanjen krajnji kvalitet celog sistema
- teško održavanje aplikacija

# Prednosti AOP-a ?

- povećana individualnost modula
- znatno lakše održavanje sistema
- arhitekta mogu odluke kasnije da donose, a ne u početnoj fazi projektovanja sistema
- povećan code reuse
- smanjeno vreme isporuke gotovog proizvoda

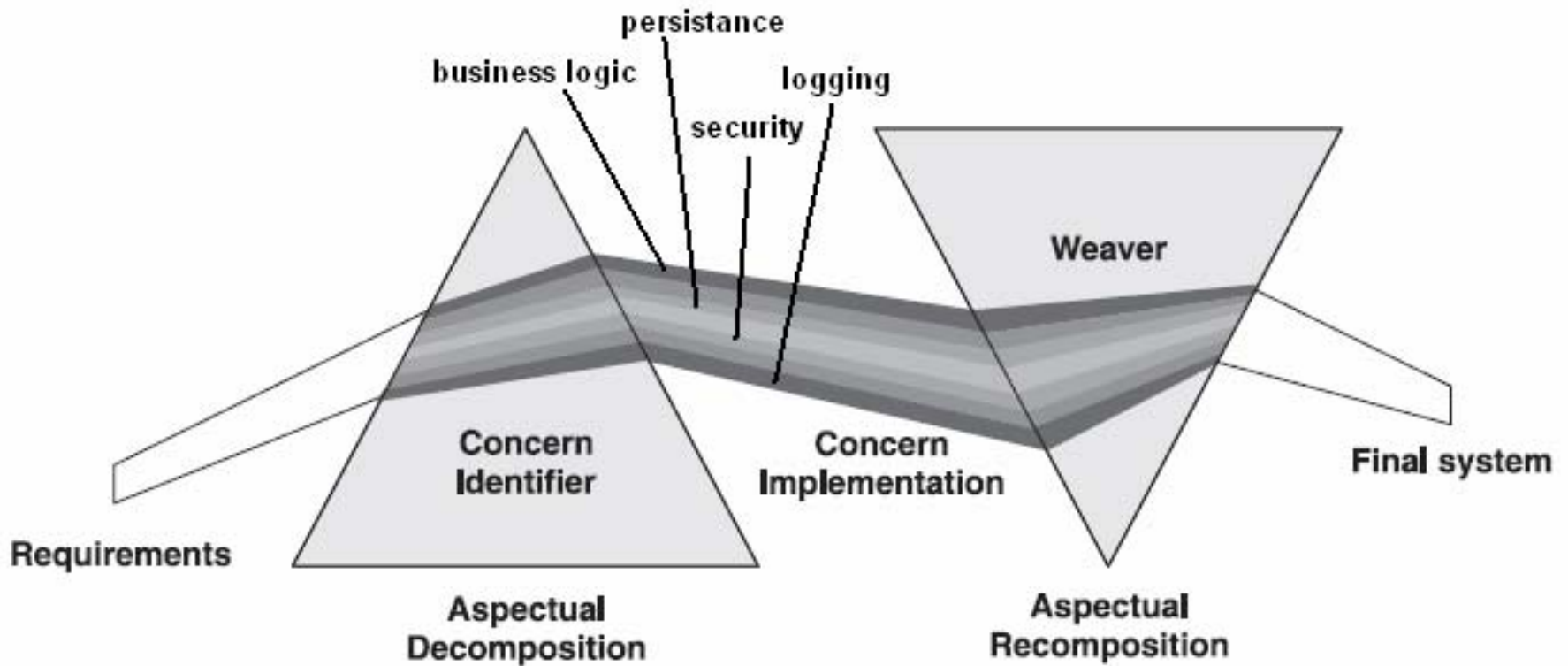
# AOP mitovi

- Da li je AOP zamena za OOP? - NE!
- Da li AOP resava neke nove probleme? – NE!
- Da li AOP moze da bude opasan? – DA!

# Rasprostranjene funkcionalnost:

- Autentikacija
- Logovanje poruka
- Resource pooling
- Keširanje
- Perzistenacija podataka
- Sigurnost
- Multithreading
- Error checking

# Analogija AOP-a sa svetlosnom prizmom



# Bez AOP-a

```
public class NekaKlasa extends NekaDrugaKlasa {
```

```
... otvaranje logger-a
```

```
public void NekaMetoda(...) {
```

```
... autorizacija
```

```
... lock-ovanje objekta da bi se obezbedio thread safety
```

```
... azuriranje cache-a
```

```
... logovanje starta operacije
```

```
... sama operacija
```

```
... logovanje zavrsetka operacije
```

```
... unlock objekta
```

```
}
```

```
}
```

Sa AOP-om klasa poprima izgled:

```
public class NekaKlasa extends NekaDrugaKlasa {  
  
    public void NekaMetoda(...) {  
        ... sama operacija  
    }  
}
```

# Dekompajlirana klasa nakon kompajliranja sa AspectJ kompajlerom:

```
public class NekaKlasa extends NekaDrugaKlasa {
```

```
    Logger _logger = ....
```

```
    public void NekaMetoda(...) {
```

```
        _logger.log (“Zapocinjemo validaciju kreditne kartice”);
```

```
        ... sama operacija
```

```
        _logger.log (“Validacija kreditne kartice je završena”);
```

```
    }
```

```
}
```

# AOP termini

- Joinpoint
- Pointcut
- Advice
- Introduction
- Compile-time declaration
- Aspect

# Hello World primer - klasa

```
public class MessageWriter {  
  
    public static void writeOut ( String message ) {  
        System.out.println ( message );  
    }  
    public static void writeOut ( String person, String message ) {  
        System.out.println ( person + " , " + message );  
    }  
    public static void main ( String args[] ) {  
        MessageWriter.writeOut( "AspectJ Hello World!" );  
        MessageWriter.writeOut( "Kosticu", "how are you?" );  
    }  
}
```

# Hello World primer – output 1

- `ajc MessageWriter.java`
- `java MessageWriter`

AspectJ Hello World!

Kosticu, how are you?

# Hello World primer – aspekt 1

```
public aspect OutputAspect {  
  
    pointcut writeOutMessage() :  
        call (* MessageWriter.writeOut(...));  
  
    before () : writeOutMessage() {  
        System.out.println("output is:");  
    }  
}
```

## Hello World primer – output 2

- `ajc MessageWriter.java OutputAspect.java`
- `java MessageWriter`

output is: AspectJ Hello World!

output is: Kosticu, how are you?

# Hello World primer – aspekt 2

```
public aspect MannersAspect {  
  
    pointcut maleTitle( String person ) :  
        call ( * MessageWriter.writeOut(String, String) )  
        && args ( person, String );  
  
    void around (String person) : maleTitle (person) {  
        proceed ( “Mr.” + person );  
    }  
  
}
```

## Hello World primer – output 3

- `ajc MessageWriter.java OutputAspect.java  
MannersAspect.java`
- `java MessageWriter`

output is: AspectJ Hello World!

output is: Mr. Kosticu, how are you?

# Primene AOP-a

- **Logging and debugging**
- **Policy enforcement**
- **Resource pooling**
- **Caching**
- **Wormhole**
- **Exception introduction**
- **Thread safety**
- **Autentikacija i autorizacija**
- **Transaction management**
- **Implementacija biznis pravila – integracija sa rule engine-ima**

# Policy enforcement

```
aspect DetectSystemOutErrorUsage {  
    declare warning : get( * System.out ) || get( * System.err ) :  
        “Consider using Logger.log() instead System.out or System.err.”  
}
```

---

```
public aspect ShoppingCartAccessAspect {  
    declare error : (call (* ShoppingCart.add*(...))  
        || call (* ShoppingCart.remove*(...)))  
        && !within(ShoppingCartOperator) :  
        “Illegal manipulation to ShoppingCart! Only ShoppingCartOperator  
        may perform such operations”;  
}
```

---

```
aspect DetectPublicAccessMembers {  
    declare warning : get(public !final * *) || set (public * *) :  
        “Consider using nonpublic access to member variables”;  
}
```

# Resource pooling aspect

```
import java.sql.*;
public aspect DBConnectionPoolingAspect {
    // Kreiranje resource pool-a
    DBConnectionPool _connPool = new SimpleDBConnectionPool();

    // Pointcut koji definise trenutak kreiranja nove konekcije
    pointcut connectionCreation(String url, String username, String password)
        : call (public static Connection DriverManager.getConnection(String, String, String))
          && args(url, username, password);

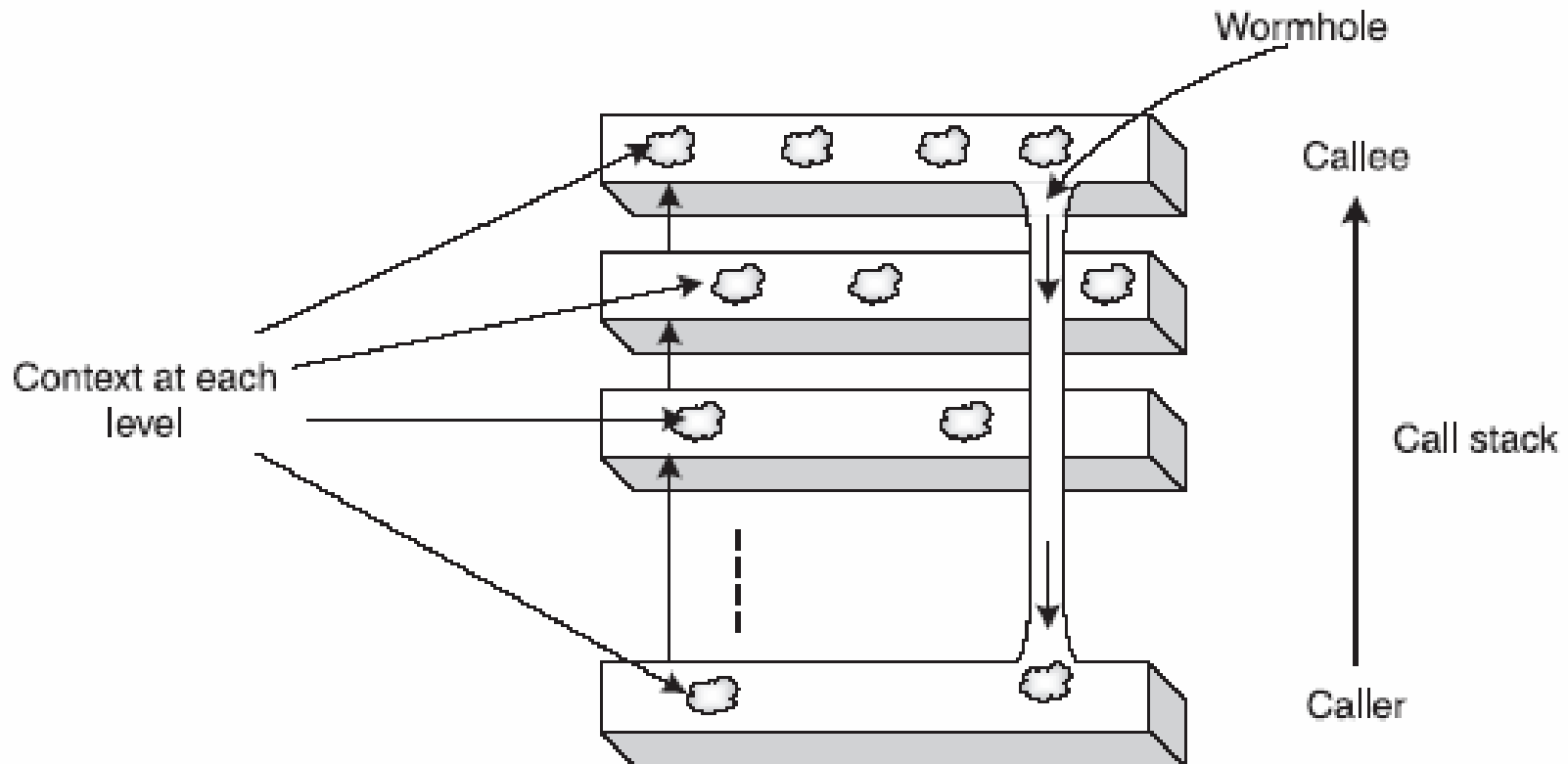
    // Pointcut koji definise trenutak oslobadjanja stare konekcije
    pointcut connectionRelease(Connection connection)
        : call (public void Connection close())
          && target(connection);

    Connection around(String url, String username, String password) throws SQLException
        : connectionCreation(url, userName, password) {

        Connection connection = connPool.getConnection(url, userName, password);
        if(connection == null){
            connection.proceed(url, userName, password);
            _connPool.registerConnection(connection, url, userName, password);
        }
    }
    return connection;
}

void around(Connection connection) : connectionRelease(connection) {
    if(!_connPool.putConnection(connection)) {
        proceed(connection);
    }
}
}
```

# Wormhole pattern



**Figure 8.1** The wormhole pattern. Each horizontal bar shows a level in the call. The wormhole makes the object in the caller plane available to the methods in the called plane without passing the object through the call stack.

# AspectJ + Java Tiger

```
.....  
@ dugotrajna_operacija  
public void NekeMetoda() {  
    .... neka duga obrada ...  
}
```

---

```
public aspect LongOperationAspect {  
    public pointcut longOperation() : call (@dugotrajna_operacija);  
    void around() : longOperation() {  
        Runnable worker = new Runnable() {  
            public void run() {  
                proceed();  
            }  
        }  
        Thread workerThread = new Thread(worker);  
        workerThread.start();  
    }  
}
```

---

```
@Upis_U_Bazu("Account", "name", "Nemanja" )
```

# Reference:

- [www.aspectprogrammer.org](http://www.aspectprogrammer.org)
  - sajt Adrian Colyera
- Knjiga “AspectJ in Action”
  - Ramnivas Laddad, Manning
- [www.aosd.net](http://www.aosd.net)
  - zvanican sajt AOP-a
- [www.eclipse.org/aspectj](http://www.eclipse.org/aspectj)
  - AspectJ
- [www.eclipse.org/ajdt](http://www.eclipse.org/ajdt)
  - AspectJ IDE plugin za Eclipse