# Projet Informatique

2019

Steve Hostettler

Software Modeling and Verification Group

University of Geneva

# Aspect Oriented Programming

# Description

- Séparation des préoccupations (concerns)
- Gestion séparée des préoccupations transverses (cross-cutting concerns)
- paradigme non lié à un langage de programmation en particulier

# Exemple de préoccupations transverses

- Journalisation
- Gestion des transactions
- Sécurité
- Injection de dépendances

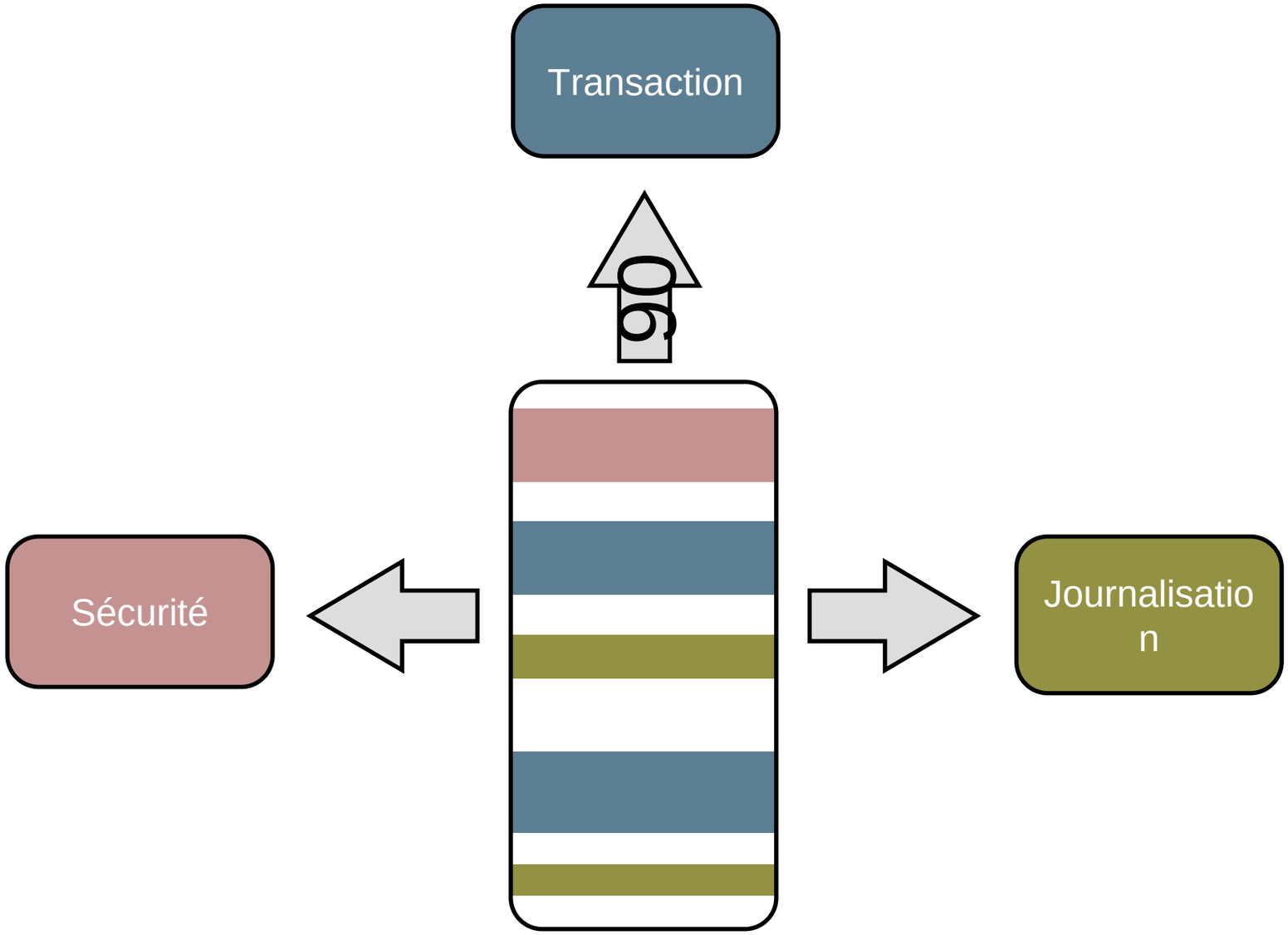
# Préoccupations transverses

- a.k.a Cross-Cutting concerns
- a.k.a Aspects
- doit être adressé dans plus d’un module

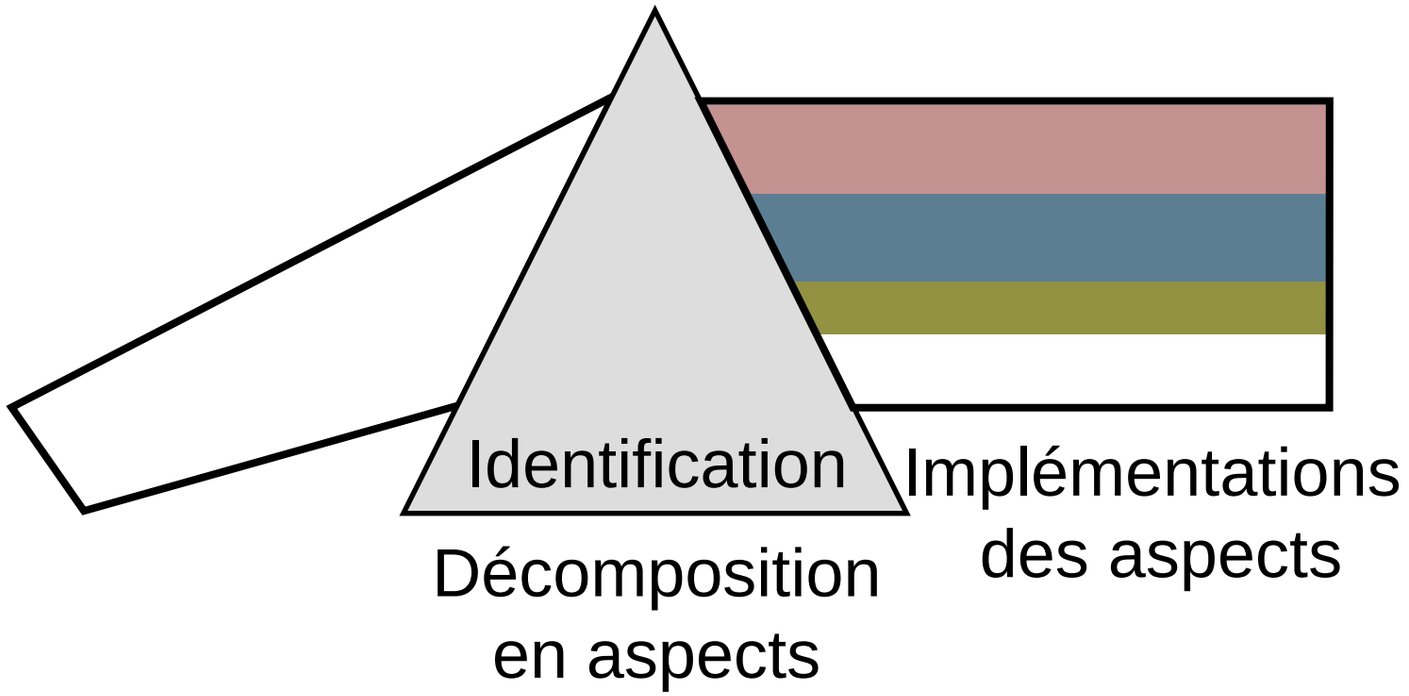
# Préoccupations transverses

- Orthogonalité par rapport à la fonctionnalité

# Préoccupations transverses



# Préoccupations transverses



# Préoccupations transverses

```
public void doSomethingUseful4TheBusiness() {
    if (!Security.getCurrentUser().getRole.equals(Security.ADMIN)) {
        Logger.getLogger().severe("User "
                    + Security.getCurrentUser()
                    + "not allowed to execute doSomethingUseful4TheBusiness");
        throw new NotAllowedException();
    }
    TransactionManager.start();
    try {

        doSomeThingVerySpecial();

        TransactionManager.commitTransaction();
        Logger.getLogger().info("Done");
    } catch() {
        TransactionManager.rollback();
    }

}
```

# Préoccupations transverses

```
public void doSomethingUseful4TheBusiness() {
    if (!Security.getCurrentUser().getRole.equals(Security.ADMIN)) {
        Logger.getLogger().severe("User "
                    + Security.getCurrentUser()
                    + "not allowed to execute doSomethingUseful4TheBusiness");
        throw new NotAllowedException();
    }
    TransactionManager.start();
    try {

        doSomeThingVerySpecial();

        TransactionManager.commitTransaction();
        Logger.getLogger().info("Done");
    } catch() {
        TransactionManager.rollback();
    }

}
```

# Préoccupations transverses

- Mélanger les préoccupations (code tangling)
  - Mauvaise traçabilité
  - Baisse de la productivité
  - Moins de réutilisations
  - Qualité du code plus basse
  - Evolution plus difficile

# Visitor et Template Method

# Template method

```
abstract class AbstractService {

    void executeService() {
        checkSecurity();

        logBeginOfTransaction();
        beginTransaction();

        executeBusiness();

        endTransaction();
        logEndOfTransaction();
    }

    private void logBeginOfTransaction() {}
    private void logEndOfTransaction() {}
    private void beginTransaction() {}
    private void endTransaction() {}
    private void checkSecurity() {}

    abstract void executeBusiness();

}
```

```
class MyService extends AbstractService {
    void executeBusiness() {
    }
}
```

# Template + Command pattern

```
interface Command { void execute();}

abstract class AroundInvoker implements Command {
   Command wrappedCommand;

   public AroundInvoker(Command c) {
      this.wrappedCommand = c;
   }

   public void execute() {
      before();
      wrappedCommand.execute();
      after();
    }

   protected abstract void after() { }
   protected abstract void before() { }
}

class SecurityCommand extends AroundInvoker {
     public SecurityCommand(Command c) {
           super(c);
        }
     protected void before() { ... }
}

class LogCommand extends AroundInvoker {
    public LogCommand(Command c) {
       super(c);
    }

    protected void before() { ...}

    protected void after() { ...}
}
```

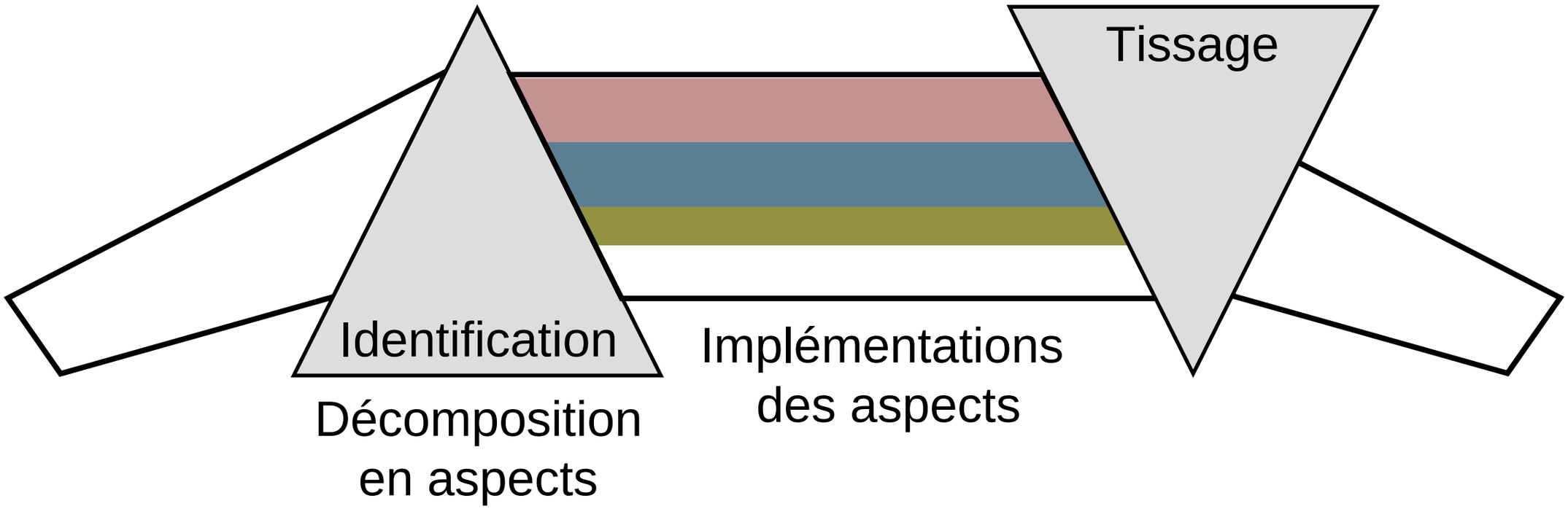
```
class TransactionCommand extends AroundInvoker {
   public TransactionCommand(Command c) {
      super(c);
   }

   protected void before() {...}

   protected void after() {...}
}


abstract class AbstractSecuredTrxLogService {
   void executeService(Command command) {
      new SecurityCommand(
        new LogCommand(
          new TransactionCommand(command)))
                .execute();
      }
}

class MyBusinessOp implements Command {
      public void execute() {
        /* Do some business stuff. */
      }
}

class MyService extends AbstractSecuredTrxLogService {
      void executeBusiness() {
         executeService(new MyBusinessOp());
      }
}
```

# Template + Command pattern

- Code toujours long à écrire
- Le contrôle du flux des aspects est toujours dans le code client.

# Aspect Oriented Programming

# Préoccupations transverses



# AOP: vocabulaire

- Greffon (advice), code exécuté à certain point de l’exécution appelé point de greffe
  - Around, Before, After
- Tissage (weaving), insertion des appels au greffons au niveau des points de greffon

# AOP: vocabulaire

- Point de greffe (pointcut), description des endroits du code ou sont implantés les greffon.
- Aspect: greffons et leurs points de greffes
- Point de jonction (joint point), endroit du code qui peut faire partie d’un point de greffon. Un ensemble de point de jonction est décrit par un point de greffon.

# AOP: stratégie

- Statique : par instrumentation du code par un post-processor
- Dynamique: instrumentation au moment du chargement de la classe (voir JVMTI).
  - Coût @Runtime, jusqu’à 10% mais la plupart du temps négligeable.

# AOP: Les +

- Découplage du métier et du code technique
- Réutilisabilité du code technique
- gain de productivité
- Amélioration de la lisibilité
- Mise à jour dynamique

# AOP: Les -

- Magie noire
- Problème dans certain cas avec le débugger
- With great power comes great responsibility....

# CDI Interceptors

# Lié un intercepteur (interception)

Cette annotation définie une interception nommée Benchmarkable

```
@InterceptorBinding
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.METHOD, ElementType.TYPE})
public @interface Benchmarkable {}
```

On peut “benchmarker” des méthodes et des types

Appliquer l’intercepteur(s) sur la méthode add

```
/** {@inheritDoc} */
@Override
@Benchmarkable
public void add(final Student student) {
    this.mStudentList.add(student);
}
```

# Intercepteur

C'est un intercepteur qui met en oeuvre "benchmarkable"

```
@Benchmarkable @Interceptor
public class PerformanceInterceptor {

    /** The default logger for the class. */
    private static final Logger LOGGER = Logger.getAnonymousLogger();

    @AroundInvoke
    public Object logPerformar                                    s Exception {
        long start = System.cu
```

Appliquer autour de la méthode annotée

```
        Object value = context.proceed();

        StringBuilder str = new StringBuilder("******  ");
        str.append(context.getMethod().getName());
        str.append(":");
        str.append(System.currentTimeMillis() - start);
        str.append(" ms  ******");
        LOGGER.info(str.toString());

        return value;
    }
}
```

Execute la méthode annotée

retourne la valeur de la méthode

# Intercepteur

```
@Benchmarkable @Interceptor
public class PerformanceInterceptor {

    /** The default logger for the class. */
    private static final Logger LOGGER = Logger.getAnonymousLogger();

    @AroundInvoke
    public Object logPerformance(InvocationContext context) throws Exception {
        long start = System.currentTimeMillis();

            this.mStudentList.add(student);

        StringBuilder str = new StringBuilder("******  ");
        str.append(context.getMethod().getName());
        str.append(":");
        str.append(System.currentTimeMillis() - start);
        str.append(" ms  ******");
        LOGGER.info(str.toString());

        return value;
    }
}
```

# Intercepteur

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:weld="http://jboss.org/schema/weld/beans"
      xsi:schemaLocation="
         http://java.sun.com/xml/ns/javaee http://jboss.org/schema/cdi/beans_1_0.xsd
         http://jboss.org/schema/weld/beans http://jboss.org/schema/weld/beans_1_1.xsd">
   <interceptors>
         <class>ch.demo.helpers.jpa.TransactionInterceptor</class>
         <class>ch.demo.business.interceptors.PerformanceInterceptor</class>
   </interceptors>
</beans>
```

L’intercepteur doit n’est pas activé par défaut

# Exercices

- Changer l'intercepteur de performances pour lui faire afficher le nom de la method invoquée
- Créez un intercepteur qui affiche tout les requête à la méthode getPieModel()