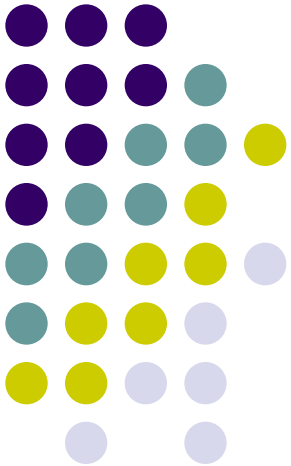
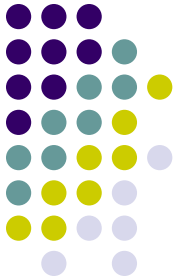
Hibernate

Spring AOP

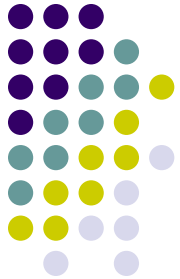


Contenidos



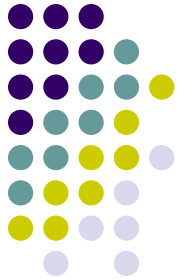
- Intro
- Spring AOP
- Ejemplos

AOP



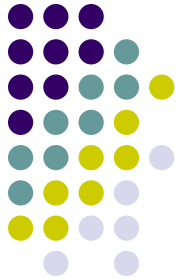
La Programación Orientada a Aspectos (POA) es un paradigma de programación cuya intención es permitir una adecuada modularización de las aplicaciones y posibilitar una mejor separación de incumbencias.

AOP



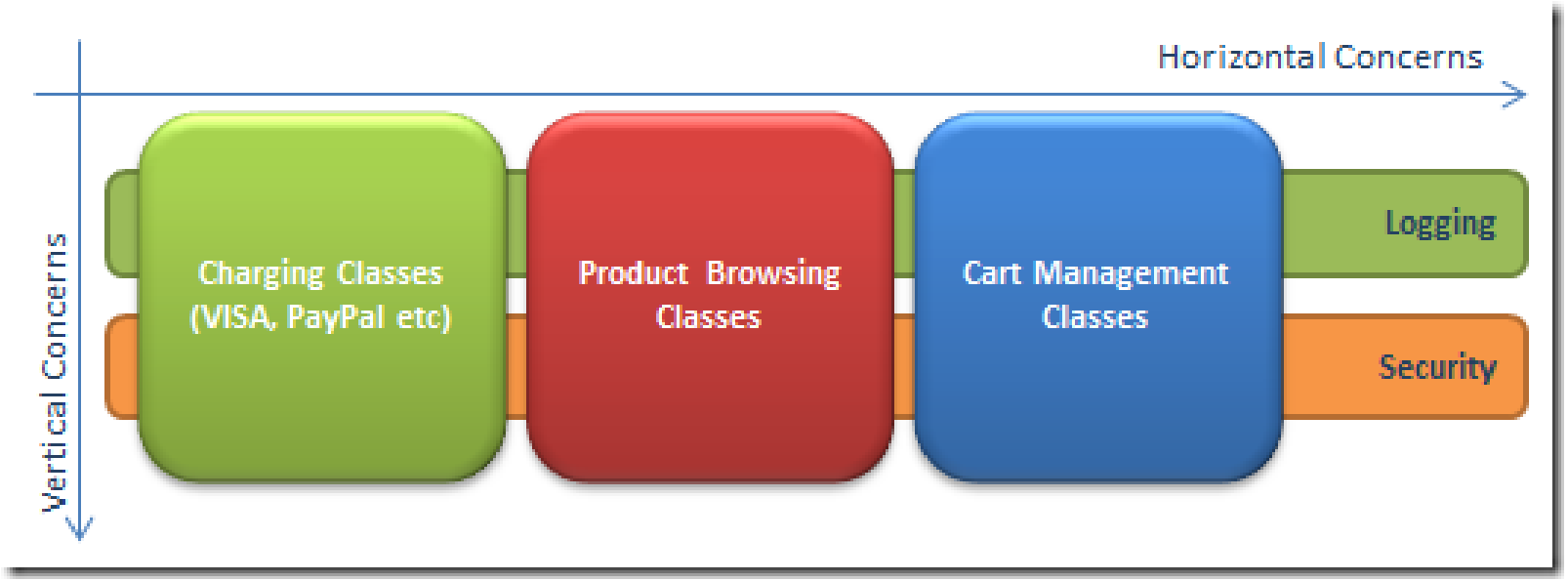
Gracias a la POA se pueden encapsular los diferentes conceptos que componen una aplicación en entidades bien definidas, eliminando las dependencias entre cada uno de los módulos.

AOP

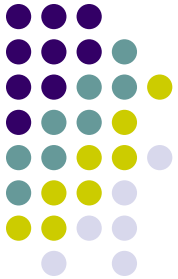


De esta forma se consigue razonar mejor sobre los conceptos, se elimina la dispersión del código y las implementaciones resultan más comprensibles, adaptables y reusables.

AOP


Horizontal Concerns
Vertical Concerns
Charging Classes (VISA, PayPal etc)
Product Browsing Classes
Cart Management Classes
Logging
Security

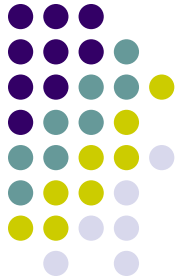
Spring + AOP



Spring utiliza AOP en un montón de módulos. EJ: Seguridad, Transacciones, Cacheo, entre otros.

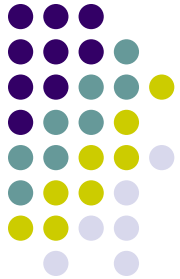
Pero al mismo tiempo nos permite utilizar las bondades de AOP directamente en nuestra aplicación programando mediante algún framework.

Conceptos Básicos



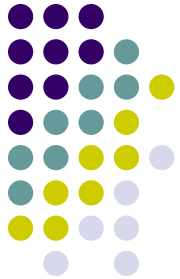
Aspect (Aspecto) es una funcionalidad transversal (cross-cutting) que se va a implementar de forma modular y separada del resto del sistema. El ejemplo más común y simple de un aspecto es el logging (registro de sucesos) dentro del sistema, ya que necesariamente afecta a todas las partes del sistema que generan un suceso.

Conceptos Básicos



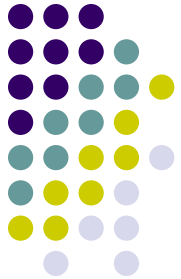
Join point (Punto de Cruce o de Unión) es un punto de ejecución dentro del sistema donde un aspecto puede ser conectado, como una llamada a un método, el lanzamiento de una excepción o la modificación de un campo. El código del aspecto será insertado en el flujo de ejecución de la aplicación para añadir su funcionalidad.

Conceptos Básicos



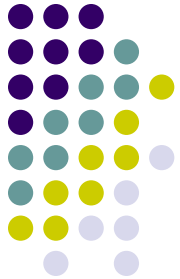
Advice (Consejo) es la implementación del aspecto, es decir, contiene el código que implementa la nueva funcionalidad. Se insertan en la aplicación en los Puntos de Cruce.

Conceptos Básicos



Pointcut (Puntos de Corte) define los Consejos que se aplicarán a cada Punto de Cruce. Se especifica mediante Expresiones Regulares o mediante patrones de nombres (de clases, métodos o campos), e incluso dinámicamente en tiempo de ejecución según el valor de ciertos parámetros.

Conceptos Básicos



execution(public * *(..))

execution(* set*(..))

execution(* get*(..))

execution(* com.xyz.service.MyService.*(..))

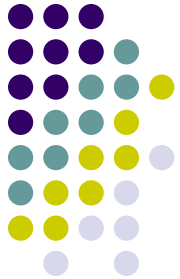
execution(* com.xyz.service.*.*(..))

execution(* com.codeproject.*.GamesImpl.save(..))

execution(* com.codeproject.*.GamesImpl.list*(..))

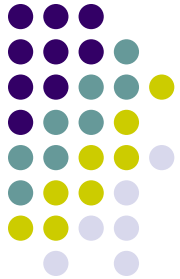
execution(* set*(java.lang.String))

Conceptos Básicos



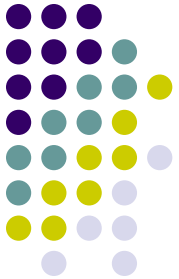
Target (Destinatario) es la clase aconsejada. Sin AOP, esta clase debería contener su lógica, además de la lógica del aspecto.

Conceptos Básicos



Proxy (Resultante) es el objeto creado después de aplicar el Consejo al Objeto Destinatario. El resto de la aplicación únicamente tendrá que soportar al Objeto Destinatario (pre-AOP) y no al Objeto Resultante (post-AOP).

Ejemplos



@Before

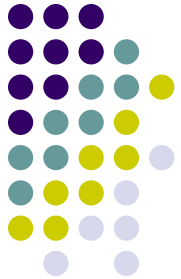
@After

@AfterReturning

@AfterThrowing

@Around

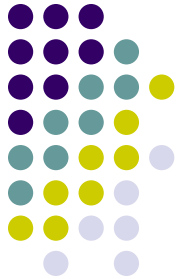
Ejemplos



```
@Aspect
public class LoggingAspect {
        @Before("execution(* com.customer.bo.CustomerBo.addCustomer(..))")
        public void logBefore(JoinPoint joinPoint) {
        .....
        }
}

<bean id="logAspect" class="com.aspect.LoggingAspect" />
 <aop:config>
   <aop:aspect id="aspectLoggging" ref="logAspect" >
     <aop:pointcut id="pointCutBefore"
        expression="execution(* com.customer.bo.CustomerBo.addCustomer(..))" />
    <aop:before method="logBefore" pointcut-ref="pointCutBefore" />
  </aop:aspect>
</aop:config>
```

Ejemplos



```
@Aspect
public class LoggingAspect {
        @After("execution(* com.customer.bo.CustomerBo.addCustomer(..))")
        public void logAfter(JoinPoint joinPoint) {
        ……..
        }
}
```

```
<bean id="logAspect" class="com.aspect.LoggingAspect" />
<aop:config>
  <aop:aspect id="aspectLoggging" ref="logAspect" >
    <aop:pointcut id="pointCutAfter"
        expression="execution(* com.customer.bo.CustomerBo.addCustomer(..))" />
    <aop:after method="logAfter" pointcut-ref="pointCutAfter" />
  </aop:aspect>
</aop:config>
```

Ejemplos

```
@Aspect
public class LoggingAspect {
  @AfterReturning(
   pointcut = "execution(* com.customer.bo.CustomerBo.addCustomerReturnValue(..))",
   returning= "result")
   public void logAfterReturning(JoinPoint joinPoint, Object result) { …   }
}

<bean id="logAspect" class="com.aspect.LoggingAspect" />
<aop:config>
  <aop:aspect id="aspectLoggging" ref="logAspect" >
    <aop:pointcut id="pointCutAfterReturning"
      expression="execution(*
com.customer.bo.CustomerBo.addCustomerReturnValue(..))" />
    <aop:after-returning method="logAfterReturning" returning="result"
      pointcut-ref="pointCutAfterReturning" />
  </aop:aspect>
</aop:config>
```

Ejemplos

```
@Aspect
public class LoggingAspect {
  @AfterThrowing(pointcut = "execution(*
com.customer.bo.CustomerBo.addCustomerThrowException(..))", throwing= "error")
  public void logAfterThrowing(JoinPoint joinPoint, Throwable error) {
                //...
  }
}

<bean id="logAspect" class="com.aspect.LoggingAspect" />
<aop:config>
  <aop:aspect id="aspectLoggging" ref="logAspect" >
    <aop:pointcut id="pointCutAfterThrowing"
      expression="execution(*
com.customer.bo.CustomerBo.addCustomerThrowException(..))" />
    <aop:after-throwing method="logAfterThrowing" throwing="error"
      pointcut-ref="pointCutAfterThrowing"  />
  </aop:aspect>
</aop:config>
```

Ejemplos

```
@Aspect
public class LoggingAspect {
        @Around("execution(*
com.customer.bo.CustomerBo.addCustomerAround(..))")
        public void logAround(ProceedingJoinPoint joinPoint) throws Throwable {
                //...
        }
}

<bean id="logAspect" class="com.aspect.LoggingAspect" />
<aop:config>
  <aop:aspect id="aspectLoggging" ref="logAspect" >
  <aop:pointcut id="pointCutAround"
    expression="execution(* com.customer.bo.CustomerBo.addCustomerAround(..))" />
  <aop:around method="logAround" pointcut-ref="pointCutAround"  />
 </aop:aspect>
</aop:config>
```

Ejercicio

Partiendo del ejercicio del banco y las cuentas utilizar AOP para:

1) Auditar todas las operaciones realizadas contra el sistema (ej: todas las llamadas a los métodos)
2) Al extraer dinero de las cuentas bancarias, aplicar un costo extra a la operación