Spring

**...a che ci serve? Come lo uso? Dove lo metto?
Cosa mi risolve? Quali patterns implementa?**

JavaDay

Sabato 2
Dicembre

Università degli studi Roma Tre

Giorgio Vinci
Senior Consultant
g.vinci@k-tech.it

Simone Federici
Java Architect
s.federici@k-tech.it

AGENDA

- Inversion of Control e Dependency Injection
  Introduzione e Definizioni
- Dependency Injection con Spring
  Funzionamento del Container
- Aspect Oriented Programming
  Transazioni Dichiarative e Custom Aspect
- Considerazioni

Simone Federici & Giorgio Vinci
s.federici(at)gmail(dot)com & giorgiovinci(at)gmail(dot)com

Inversion of Control

Nel 2004 ***Martin Fowler***

Inversion of Control Containers and the Dependency Injection pattern

http://www.martinfowler.com/articles/injection.html

The question is: **what aspect of control are they inverting?**

“When I first ran into inversion of control, it was in the main control of a user interface. Early user interfaces were controlled by the application program. You would have a sequence of commands like "Enter name", "enter address"; your program would drive the prompts and pick up a response to each one. With graphical (or even screen based) UIs the UI framework would contain this main loop and your program instead provided event handlers for the various fields on the screen. The main control of the program was inverted, moved away from you to the framework”.

Inversion of Control (IoC)

Anche detto Hollywood principle:

“*don't call us, we'll call you*”

“*Non ci chiamate, vi chiamiamo noi*”

Nel 1997 gia si parlava di IoC riferendosi alla gestione degli eventi sulle interfacce utente.
Il passaggio del mouse veniva gestito dalla finestra e poi passato a tutti i suoi figli (che ignoravano il messaggio se non diretto a loro). Mentre all'inverso il mouse poteva essere gestito dall'oggetto grafico che propagava il messaggio al suo contenitore.

Definizione di IoC

L'inversion of control è la parte chiave che differenzia un framework da una libreria, ogni chiamata ad una libreria effettua un compito e ritorna il controllo al chiamante.

Il framework è più complesso. Il nostro codice si integra con il framework al punto che questo stesso chiama il nostro codice (da cui l'inversione).

“ [...] dire che questi container sono speciali perché usano l'Inversion of Control, è come dire che la mia macchina è speciale perché ha le ruote.”

Martin Fowler (2004)

Il segreto del suo successo

Se tutti i Framework applicano IoC cos'ha di speciale Spring?!

Il successo di Spring è la semplicità con cui aiuta lo sviluppo di applicazioni basate sulla Dependency Injection.

Semplice, ma geniale!
(o geniale perchè Semplice?!)

Dependency Injection (DI)

C'è molta confusione nel comprendere le differenze tra IoC e Dependency Injection (DI). Molti la reputano la stessa cosa.

In verità la DI risale solo al 2002, quando Martin Fowler coniò questo nuovo termine per isolare una specifica tecnica di IoC.

La Dependency Injection è la tecnica grazie alla quale un container “inietta” un oggetto (setta un attributo) in un altro senza che questi si conoscano se non attraverso un'interfaccia.

Tipologie di Dependency Injection

- by Constructor
- by Setter Methods
- by Interface (avalon)
- by Reflection

DI: Constructor Injection

```
public class TalkC {

    private IViewer slidesViewer;

    public TalkC(IViewer slidesViewer) {
        this.slidesViewer = slidesViewer;
    }


    public void start(String slide) {
       return slidesViewer.show(slide);
    }

}
```

DI: Setter Method Injection

```
public class TalkS {

    private IViewer slidesViewer;

    public TalkS () { }

    public void setViewer(IViewer slidesViewer) {
        this.slidesViewer = slidesViewer;
    }

    public void start(String slide) {
       return slidesViewer.show(slide);
    }

}
```

Isolation of Concerns

Le responsabilità di creazione e di associazione tra gli oggetti vengono rimosse dagli stessi. Le troviamo invece in un Container (qualcosa che "sovrasta" il nostro programma).
Il vantaggio è che gli oggetti si concentrano solo sulla logica di business.

Dimensione Orizzontale e Verticale

Immaginiamo di disegnare il nostro progetto orizzontalmente per ogni oggetto disaccoppiato e disegnarlo in verticale per ogni associazione.

In medio stat virtus.

Un piccolo grande trucco

La difficoltà nell'applicare la Dependency Injection è l'individuazione della parte ***variante*** e di quella ***invariante*** del nostro progetto.

Ci aiuta se usato nella parte variante, ma ha l'effetto contrario se usato nelle parti invarianti.

Cosa ci risolve DI?

Riduce l'accoppiamento!
GRASP Low Coupling

Stabilisce un livello di astrazione attraverso l'uso delle interfacce eliminando quindi le dipendenze tra i componenti.
Grazie a questo approccio emerge una architettura a “plug-in”

Semplifica la divisione di un progetto in team

Facilita i test
Easy Test Driven Developing with Mock

Implementazioni di IoC/DI

NanoContainer
http://nanocontainer.org/

PicoContainer
http://www.picocontainer.org

Apache Excalibur
http://excalibur.apache.org/

Seasar
http://www.seasar.org/

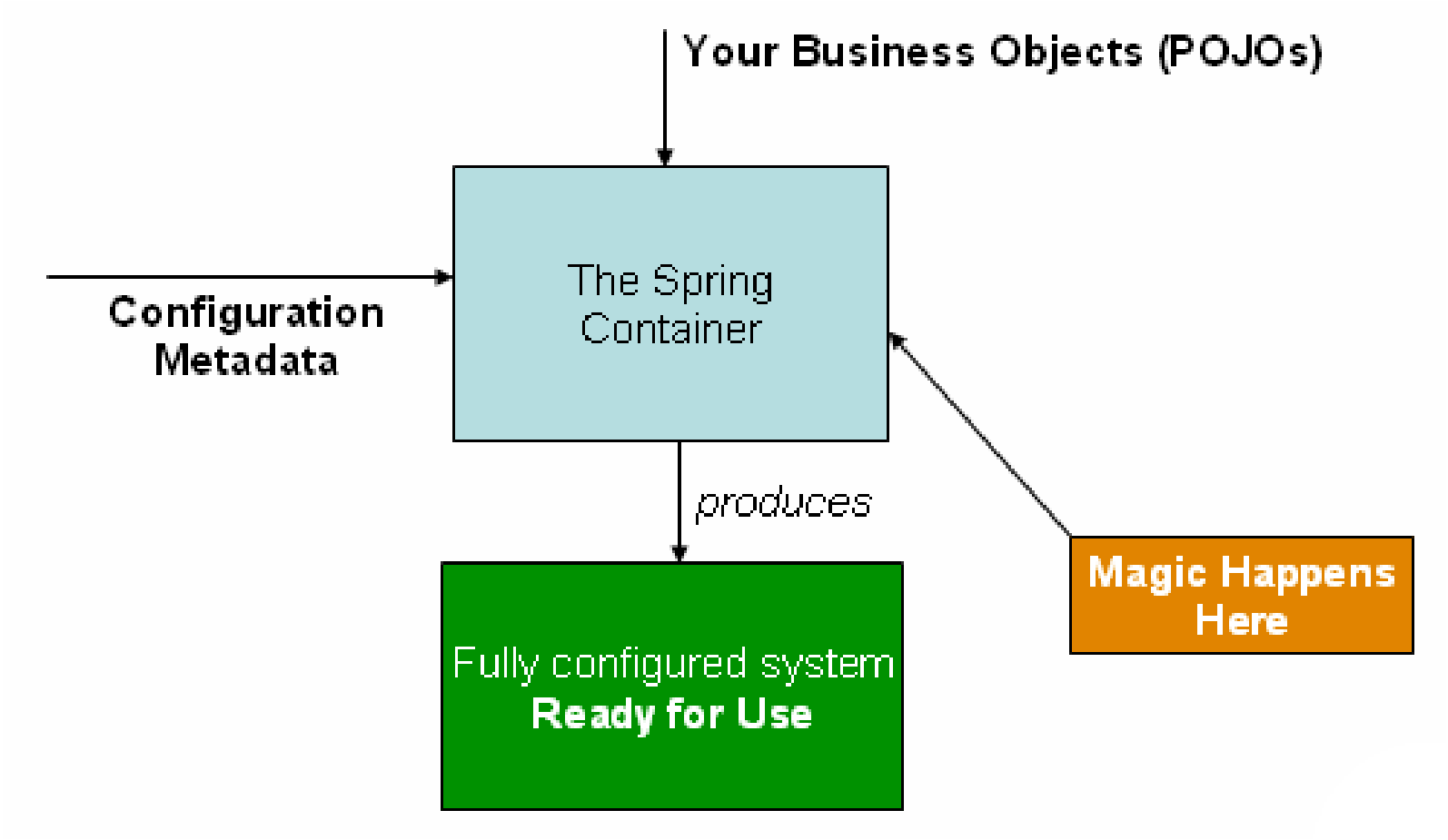
HiveMind
http://hivemind.apache.org/

DPML
http://www.dpml.net/

AGENDA

- Inversion of Control e Dependency Injection
  Introduzione e Definizioni
- Dependency Injection con Spring
  Funzionamento del Container
- Aspect Oriented Programming
  Transazioni Dichiarative e Custom Aspect
- Considerazioni

Spring container



Simone Federici & Giorgio Vinci
s.federici(at)gmail(dot)com & giorgiovinci(at)gmail(dot)com

XML Bean Factory

```
<beans xmlns="http://www.springframework.org/schema/beans">
  <bean id="oOViewer" class="it.roma.jug.OpenOfficeSlides" />
  <bean id="pdfViewer" class="it.roma.jug.PDFSlides"/>

  <bean id="talkConstructor" class="it.roma.jug.TalkC">
      <constructor-arg>
          <bean ref=”oOViewer” />
       </constructor-arg>
  </bean>

  <bean id="talkSetter" class="it.roma.jug.TalkS">
      <property name="viewer">
          <bean ref=”pdfViewer” />
      </property>
  </bean>

  <!-- more bean definitions go here... -->
</beans>
```

Instantiating a container

```
Resource resource = new FileSystemResource("beans.xml");
BeanFactory factory = new XmlBeanFactory(resource);
ITalk  talk = factory.getBean(“talkSetter”);
```

... or...

```
ClassPathResource resource = new ClassPathResource("beans.xml");
BeanFactory factory = new XmlBeanFactory(resource);
```

... or...

```
ApplicationContext context = new ClassPathXmlApplicationContext(
    new String[] {"applicationContext.xml", "applicationContext-part2.xml"});
```

Bean scopes

Singleton	un bean unico.
Prototype	ogni volta un bean identico
Request	un bean in request
Session	un bean in session

```
<aop:scoped-proxy/>
```

Ciclo di vita dei beans

E' possibile gestire il ciclo di vita di un oggetto tramite un file XML

..oppure..

implementando le “Lifecycle Interfaces” di Spring. In quest'ultimo caso però, sto legando la mia implementazione al framework.

Code Example Lifecycle

```
<bean id="talkConstructor"
      class="it.roma.jug.TalkC"
      init-method=”init”>
</bean>

..or...

public class TalkS  implements InitializingBean{

      public void afterPropertiesSet(){
            // do some init work
      }
   [...]
}
```

Spring! Non solo DI

Middle Tier Data Access

Aspect Oriented Programming

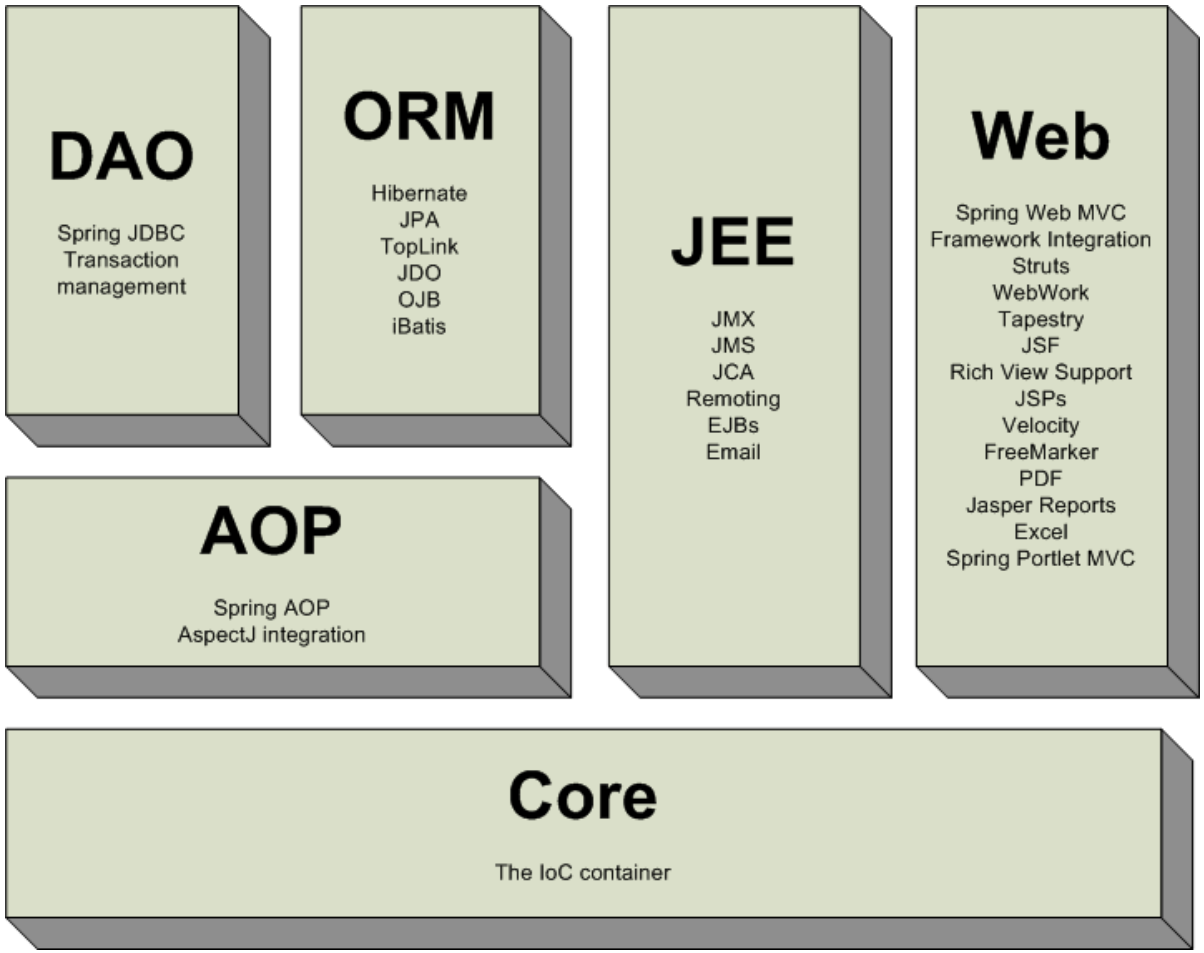
Unit testing and Integration testing

Validation, Data-binding, the BeanWrapper, and PropertyEditors

Web MVC framework

Simone Federici & Giorgio Vinci
s.federici(at)gmail(dot)com & giorgiovinci(at)gmail(dot)com

L' Architettura



Simone Federici & Giorgio Vinci
s.federici(at)gmail(dot)com & giorgiovinci(at)gmail(dot)com

AGENDA

- Inversion of Control e Dependency Injection
  Introduzione e Definizioni
- Dependency Injection con Spring
  Funzionamento del Container
- Aspect Oriented Programming
  Transazioni Dichiarative e Custom Aspect
- Considerazionii

Introduzione AOP

Che vuol dire programmare ad aspetti?
Aggiungere, separatamente, dei comportamenti (aspetti) a delle mie funzionalità (classi) nell'ottica di mantenere divisi compiti diversi (Separate of Concerns).

Spring AOP

Transazioni JDBC/JTA (local/global)

- Dichiarative (XML)
- @Transactional Annotation

Sistemi AOP per Logging o Caching

Transazioni AOP

Configuriamo un servizio ad es. una transazione.

```
<bean id="dataSource" class="org.apache.commons.dbcp.BasicDataSource"
      destroy-method="close">
            <property name="driverClassName"
                  value="oracle.jdbc.driver.OracleDriver"/>
            <property name="url" value="jdbc:oracle:thin:@rj-t42:1521:elvis"/>
</bean>

<bean id="txManager"
class="org.springframework.jdbc.datasource.DataSourceTransactionManager">
            <property name="dataSource" ref="dataSource"/>
</bean>
```

Transazioni AOP

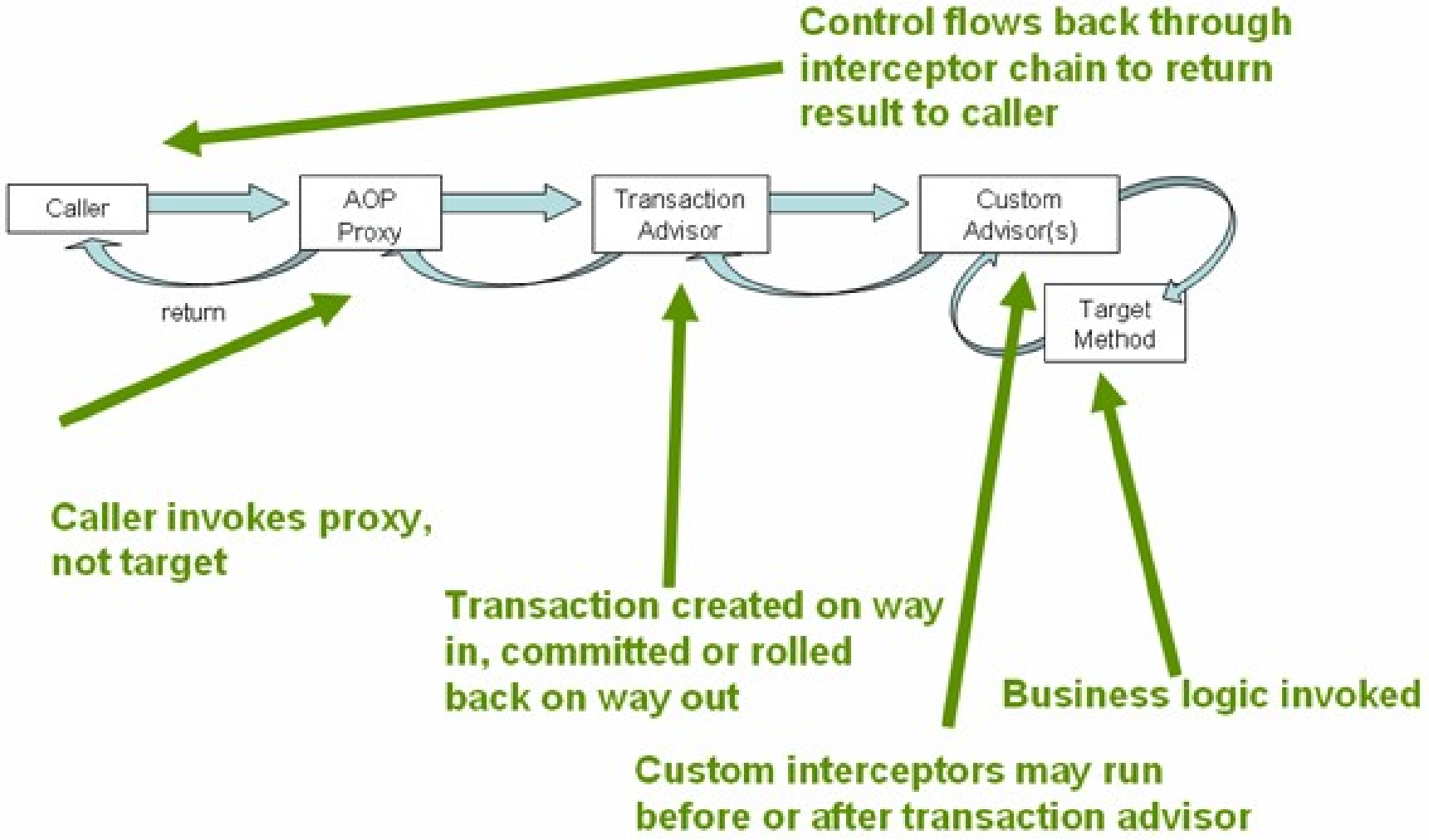
Associamo in modo totalmente dichiarativo il servizio al nostro Oggetto!

```
<bean id="myBusinessObject" class="it.jug.roma.MyBusinessObject"/>

<aop:config>
    <aop:pointcut id="myPointcut"
        expression="execution(* it.jug.roma.MyBusinessObject.*(..))"/>
    <aop:advisor advice-ref="txAdvice" pointcut-ref="myPointcut"/>
</aop:config>

<tx:advice id="txAdvice" transaction-manager="txManager">
    <tx:attributes>
        <tx:method name="load*" read-only="true"/>
        <tx:method name="save*" rollback-for=”it.jug.roma.MyBOException” />
        <tx:method name="*"/>
    </tx:attributes>
</tx:advice>
```

AOP e Transaction Advisor



Simone Federici & Giorgio Vinci
s.federici(at)gmail(dot)com & giorgiovinci(at)gmail(dot)com

AOP per Caching

```
<bean id="myCacheManager"  class="it.jug.roma.CacheManager">
      <property name="expireTime" value="60000" />
</bean>

<aop:config>
      <aop:pointcut id="getService"
            expression="execution(* it.jug.roma.MyService.getNews())"/>

      <aop:aspect id="myAspect" ref="myCacheManager">
            <aop:around pointcut-ref="getService" method="getCachedResults"/>
      </aop:aspect>
</aop:config>
```

CacheManager

```
public class CacheManager{

	private long expireTime = 10000;
	Object retVal = null;
	long lastTime= 0L;

	public Object getCachedResults(ProceedingJoinPoint pjp) throws Throwable {
		long now = new Date().getTime();
		if (now > lastTime + expireTime) {
			lastTime = now;
			retVal = pjp.proceed();
		}
		return retVal;
	}

	public void setExpireTime(long et){ expireTime = et; }
}
```

Ricordate!

Tutto quello che ora si può fare con AOP si è sempre fatto (e si continua a fare) con OOP

L'AOP ci consente di progettare in modo diverso, separando i compiti e disaccoppiando le classi del nostro modello. Aggiungendo possibilità di riutilizzo (e configurazione) degli aspetti comuni.

AGENDA

- Inversion of Control e Dependency Injection
  Introduzione e Definizioni
- Dependency Injection con Spring
  Funzionamento del Container
- Aspect Oriented Programming
  Transazioni Dichiarative e Custom Aspect
- Considerazioni

Una buona progettazione è meglio di un ottimo tool (framework)

Ed è molto facile che tutto questo, se non compreso bene, si trasformi in uno Codice a forma di Spaghetti

Simone Federici & Giorgio Vinci
s.federici(at)gmail(dot)com & giorgiovinci(at)gmail(dot)com

Prima di scegliere un framework, chiediamoci cosa dobbiamo realizzare.
Una volta che conosciamo il nostro obbiettivo saremo in grado di scegliere il giusto framework.

A volte usare il framework “giusto” nel progetto “sbagliato” può indurre in una gas factory

Non tutto ciò che è “fico” è la soluzione giusta, bisogna tener conto del tempo necessario ad introdurre un framework all'interno di un progetto.
Trasferire know-how richiede tempo!

A volte la voglia di essere sempre all'avanguardia ci porta a usare un bazooka per sparare ad una mosca.