Injeção de Dependências e Spring

Daniel Cukier

Prof. Fabio Kon

IME-USP

Conteúdo

- Exemplo – Melhor maneira de aprender
- Injeção de Dependência (DI)
- Spring
- Service Locator

Exemplo

```
class PhoneLister
   public Phone[] phonesOwnedByCompany(String arg) {
       List allPhones = finder.findAll();
       for (Iterator it = allPhones.iterator(); it.hasNext();) {
           Phone phone = (Phone) it.next();
           if (!Phone.getCompany().equals(arg)) it.remove();
       }
       return (Phone[]) allPhones.toArray(...);
   }
```

Exemplo

- Criando interface

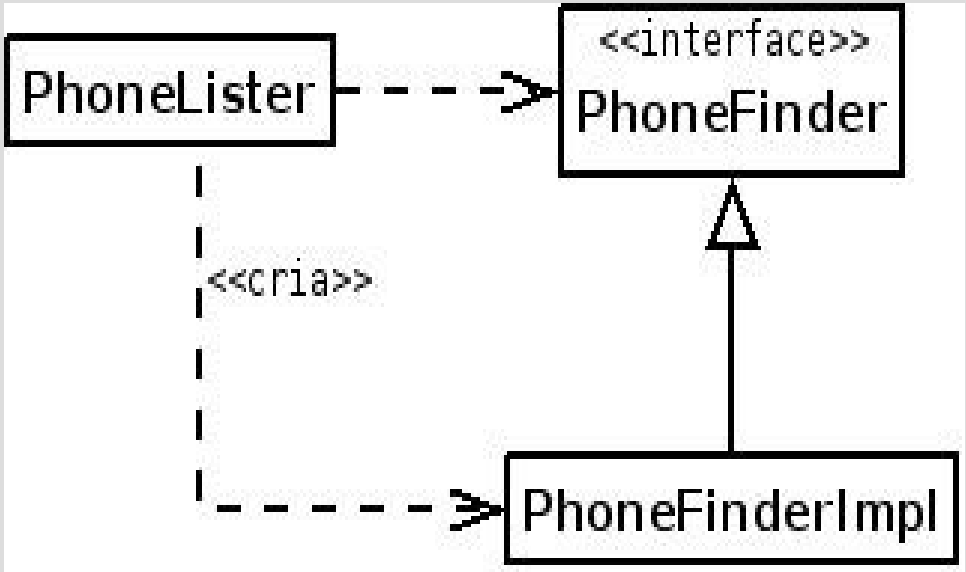
```
public interface PhoneFinder {
    List findAll();
}
```

- Criação de instância continua obrigatória

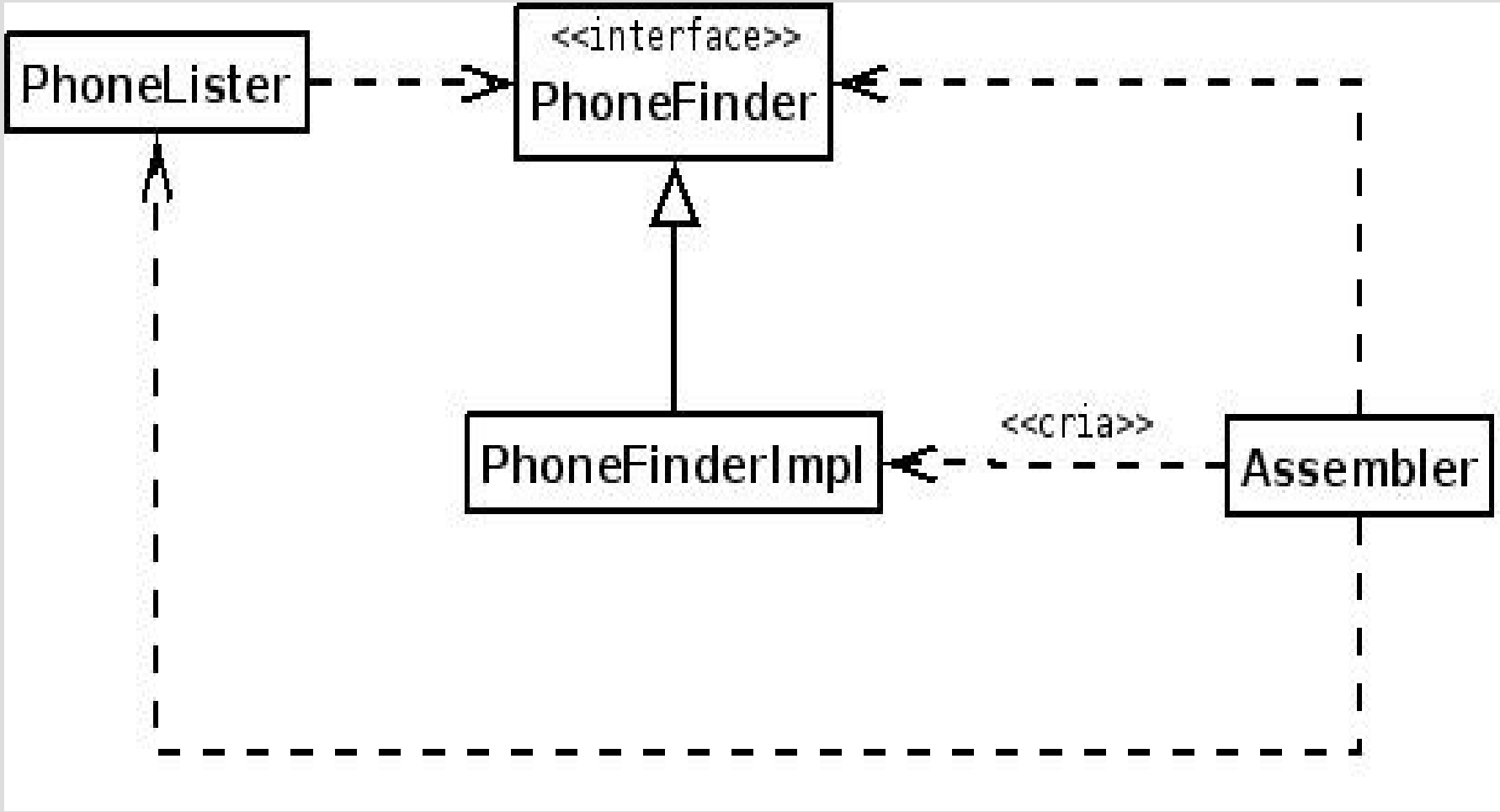
```
class PhoneLister...
  private PhoneFinder finder;
  public PhoneLister() {
    finder = new MySqlPhoneFinder("myDbName");
  }
```

Exemplo - Dependência

- PhoneLister depende da interface e da implementação
- Como depender somente da interface?



Idéia Básica - Assembler



Tipo de Injeção de Dependência

- Injeção por construtor (IoC tipo 3)
- Injeção por *setter* (IoC tipo 2)
- Injeção por interface (IoC tipo 1 – menos usada)

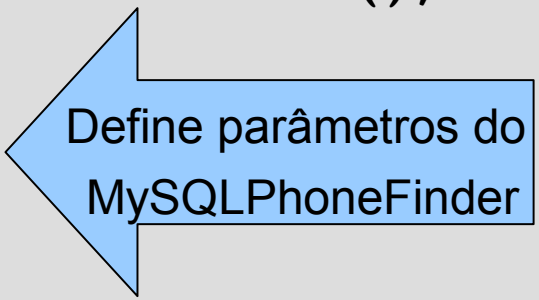
*IoC – Inversion of Control

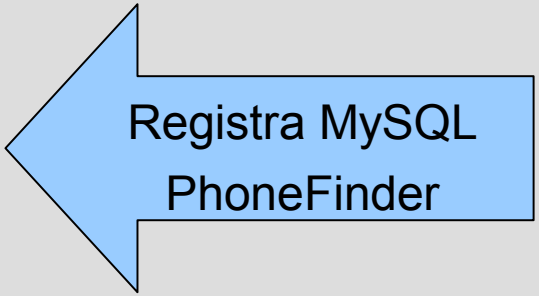
Por construtor ex: PicoContainer

```
class PhoneLister...
    public PhoneLister(PhoneFinder finder) {
        this.finder = finder;
    }

class MySqlPhoneFinder implements PhoneFinder
    public MySqlPhoneFinder(String dbName) {
        this.dbName = dbName;
    }
```

Por construtor ex: PicoContainer

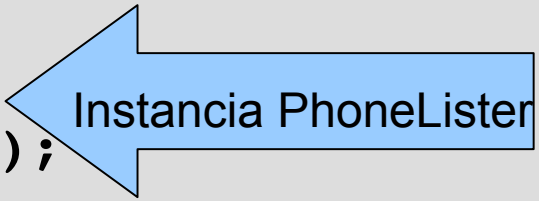
```
private MutablePicoContainer configureContainer() {
    MutablePicoContainer pico = new DefaultPicoContainer();

    Parameter[] finderParams =  {new
 ConstantParameter("myDBName")};

    pico.registerComponentImplementation
 (PhoneFinder.class, MySqlPhoneFinder.class,
 finderParams);

    pico.registerComponentImplementation(PhoneLister.class);
    return pico;
}
```





Por construtor ex: PicoContainer

```
public void testWithPico() {

  MutablePicoContainer pico = configureContainer();


  PhoneLister lister = (PhoneLister)
 pico.getComponentInstance(PhoneLister.class);


  Phone[] phones = lister.phonesOwnedByCompany("LW
 Telecom");

  assertEquals("551121613500", phones[0].getNumber());

}
```



Spring

- Focado em prover uma forma de gerenciar seus objetos de negócio
- Arquiteturado em camadas, modular. Podem ser usadas apenas partes
- Ajuda a escrever código que é fácil de testar

Spring - Arquitetura

- Ajuda a eliminar proliferação de singletons
- Menos arquivos de configurações em vários formatos diferentes
- Facilita boas práticas de programação: uso de interfaces no lugar de classes
- Não envasivo: o código do sistema depende o mínimo possível da API do Spring
- Fácil de escrever testes de unidade e também de integração (usando `TestContext Framework`)

Spring - Arquitetura

- Torna o uso de EJB uma escolha de implementação e não uma decisão de arquitetura
- Uma alternativa ao EJB que pode ser adequada a muitas aplicações
- Arcabouço consistente para acesso a dados, usando JDBC ou mapeamento O/R (hibernate, jdo, etc)
- Não reinventa a roda, apenas torna tecnologias disponíveis mais fáceis de usar
- Portável entre servidores de aplicação
- **Solução mais simples para seus problemas**

Spring – Inversão de Controle

- Fábrica de *beans*
  - Singleton
    - Instância única compartilhada do objeto
    - Uso padrão, mais comum
    - Objetos de serviço sem estado (*stateless*)
  - Protótipos
    - Cada chamada cria um novo objeto
  - Escopos de Objetos Customizados
    - Objetos armazenados fora do controle do container (ex: request, session em uma aplicação Web)

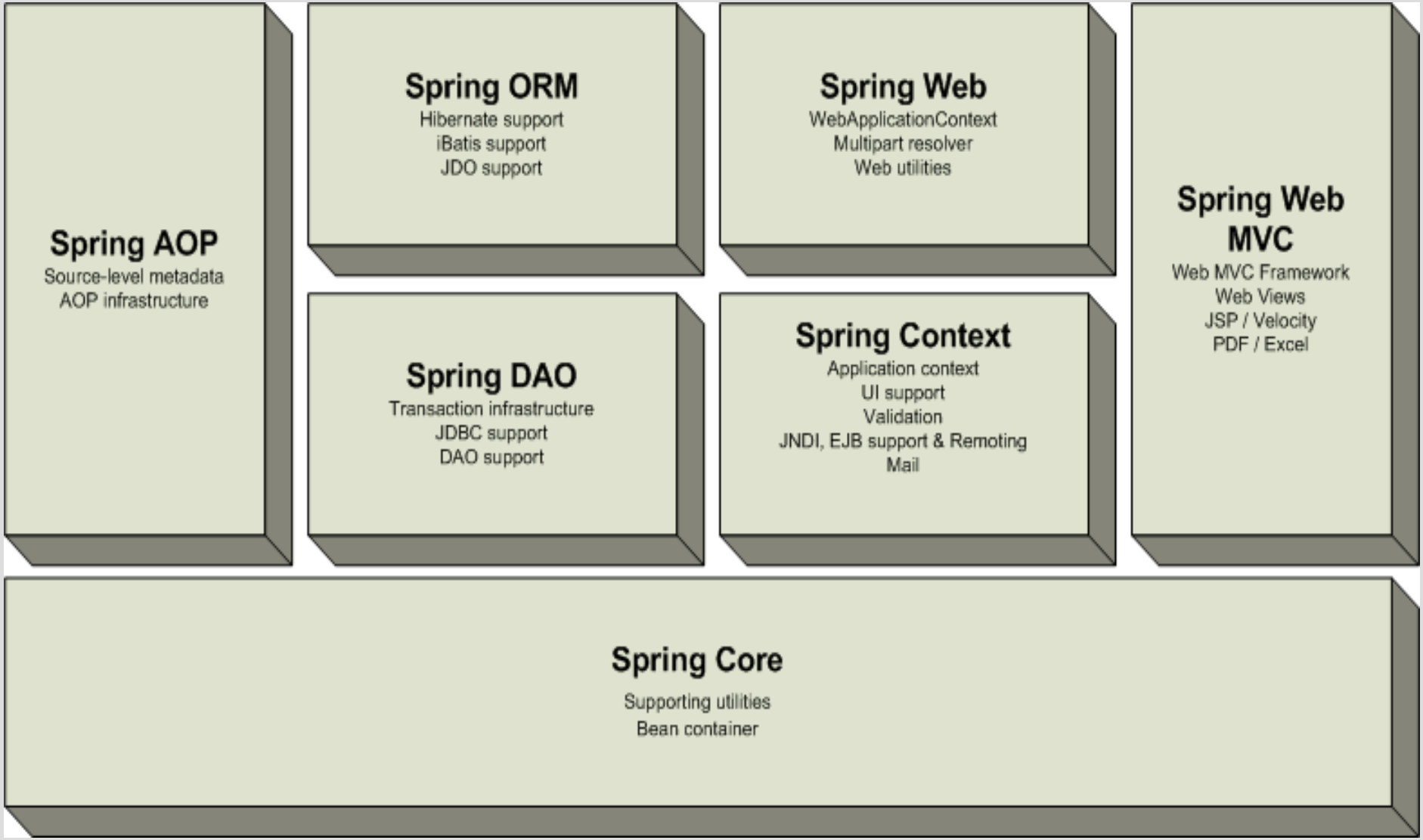
Spring - Resumo

"É uma tenologia dedicada a habilitar você a construir aplicações usando POJOs (*Plain Old Java Objects*). Permite que você crie componentes como POJOs, contendo somente lógica de negócio, enquando o Spring toma conta de outras coisas importantes que você precisa numa aplicação corporativa, mesmo em áreas que você nem havia considerado antes de contruir sua aplicação"

Spring - Injeção de Dependência

- Como os componentes não precisam procurar colaboradores em tempo de execução, o código fica mais fácil de escrever e manter
- Testes mais fáceis: é só criar objetos e atribuir as propriedades desejadas usando os *setters*
- Dependências são explícitas e evidentes
- Objetos de negócio não dependem da API do Spring

Módulos Spring



Por setter: Spring

```
class PhoneLister...
    private PhoneFinder finder;
    public setFinder(PhoneFinder finder) {
        this.finder = finder;
    }

class MySqlPhoneFinder implements PhoneFinder
    public setDbName(String dbName) {
        this.dbName = dbName;
    }
```

Por setter: Spring – XML

```
<beans>
    <bean id="PhoneLister"
       class="spring.PhoneLister">
        <property name="finder">
            <ref local="PhoneFinder"/>
        </property>
    </bean>
    <bean id="PhoneFinder"
       class="spring.MySqlPhoneFinder">
        <property name="dbName">
            <value>myDBName</value>
        </property>
     </bean>
</beans>
```

Por setter: Spring – XML

- Usa-se **`ref`** para referenciar outro bean
- Pode-se usar qualquer tipo básico Java, o Spring trata de converter tipos
- Possui suporte a listas, mapas, e outros tipos de coleções
- *Autowiring* – descobre dependência

Por setter: Spring – teste

```
public void testWithSpring() throws Exception {

   ApplicationContext ctx = new
   FileSystemXmlApplicationContext("spring.xml");

   PhoneLister lister =
     (PhoneLister) ctx.getBean("PhoneLister");

   Phone[] phones = lister.phonesOwnedByCompany("LW Telecom");
   assertEquals("551121613500", phones[0].getNumber());
}
```

Cria contexto Spring baseado no arquivo XML de configuração

Instancia PhoneLister

Aplicação Spring em 3 passos

- mvn archetype:create
  -DgroupId=seuGrupo -DartifactId=seuProjeto
  -DarchetypeArtifactId=appfuse-basic-spring
  -DarchetypeGroupId=org.appfuse.archetypes
- MySQL (Ubuntu): apt-get install mysql-server
- mvn jetty:run-war

Outros Arcabouços

- **PicoContainer**

  http://www.picocontainer.org/

  uso de anotações no lugar de XML
- **Google Guice**

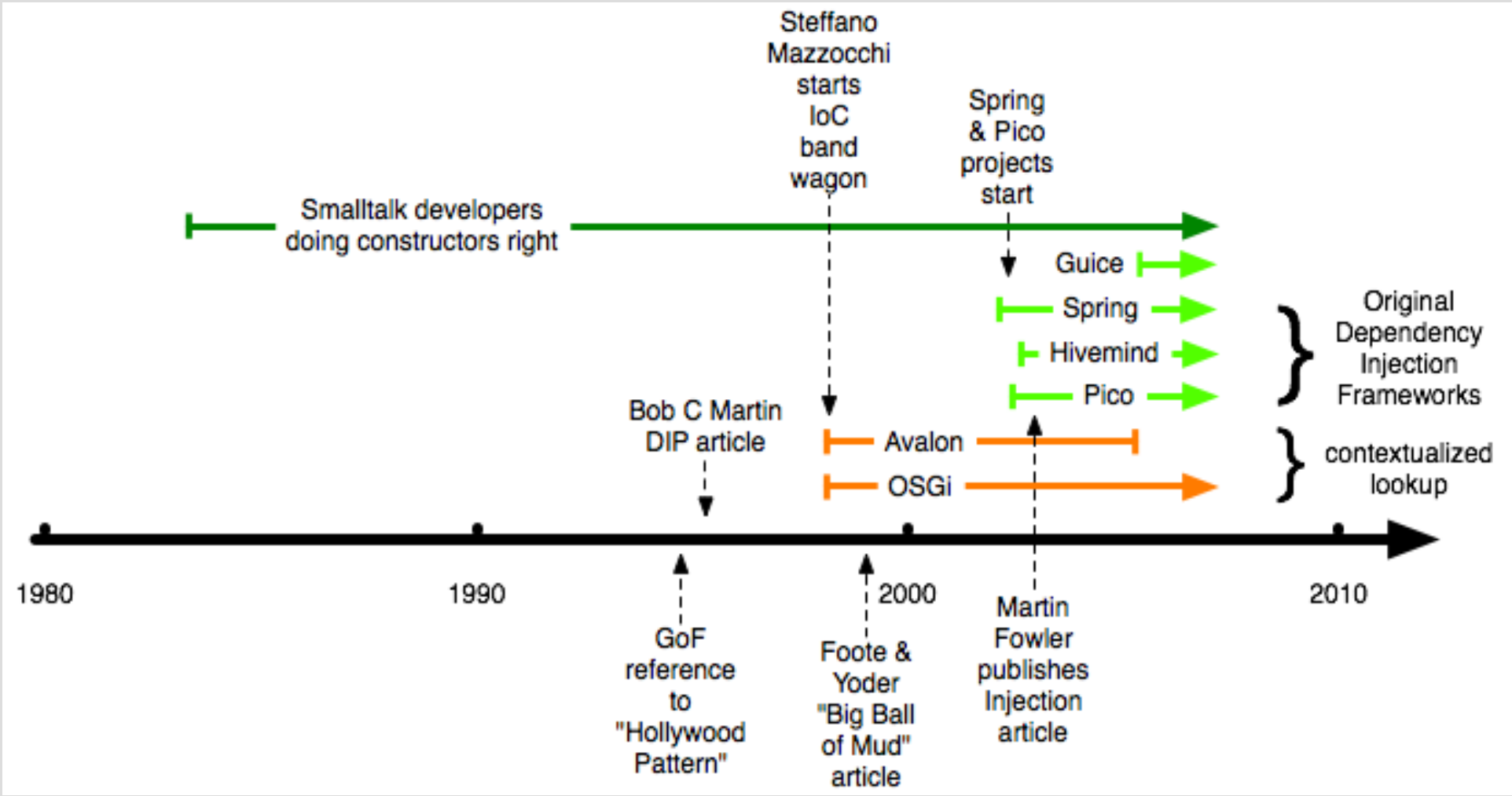
  http://code.google.com/p/google-guice/

  também usa anotações
- **Azuki**

  http://www.azuki-framework.org/
- **HiveMind**

  http://hivemind.apache.org/

DI – Linha do Tempo



Construtor ou Setter?

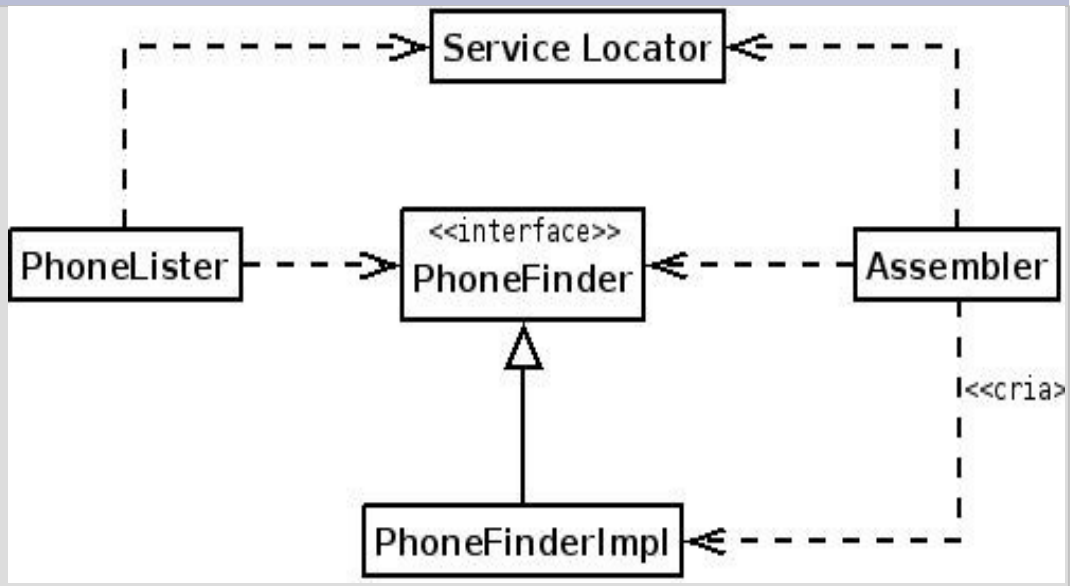
- Construtor com parâmetros deixa claro o que é preciso para criar o objeto
- Usando construtor evita campos imutáveis de serem alterados
- Cuidado: contrutores com muitos parâmetros pode ser um indicativo de objeto com responsabilidades demais
- Construtor é ruim se tiver parâmetros simples como Strings: com setter você cria um método que identifica o que a string significa
- Receita geral: comece com construtor e mude para setter se a coisa ficar complicada demais

Spring - Mercado

- Serviços: segurança, gerenciador de sistemas, workflow, persistência
- Melhorias: vários releases por ano
- Suporte: muitos arcabouços consagrados e produtos do mercado usam Spring
- Vasta literatura e documentação.

Service Locator

- Singleton que provê serviços




```
class PhoneLister...
  PhoneFinder finder =
  ServiceLocator.phoneFinder();

class ServiceLocator...
   public static PhoneFinder phoneFinder() {
       return soleInstance.phoneFinder;
   }
   private static ServiceLocator soleInstance =
      new ServiceLocator(new MySqlPhoneFinder("myDBName"));
   private PhoneFinder phoneFinder;
   ...
```

DI ou SL

- O que é melhor: Injeção de Dependência ou Service Locator?

	DI	SL
Desacoplamento	total	apenas da implementação, ainda depende de conhecer o Locator
Clareza de Código	fácil de identificar dependências	precisa olhar o código fonte do SL
Melhor Usar Quando	Muitas classes que usam serviços	Poucas classes dependem do SL
Testabilidade	Criando mocks dos serviços	Criando uma implementação do SL que retorne mocks
Implementação	automática	explícita

Referências

- http://www.springframework.org/documentation
- http://martinfowler.com/articles/injection.html
- http://appfuse.org/
- http://www.devx.com/Java/Article/21665/0/page/1
- http://www.onjava.com/pub/a/onjava/2005/05/11/spring.html
- http://www.theserverside.com/tt/articles/article.tss?l=IntrotoSpring25
- Anil Hemrajani, *Agile Java Development with Spring, Hibernate and Eclipse*