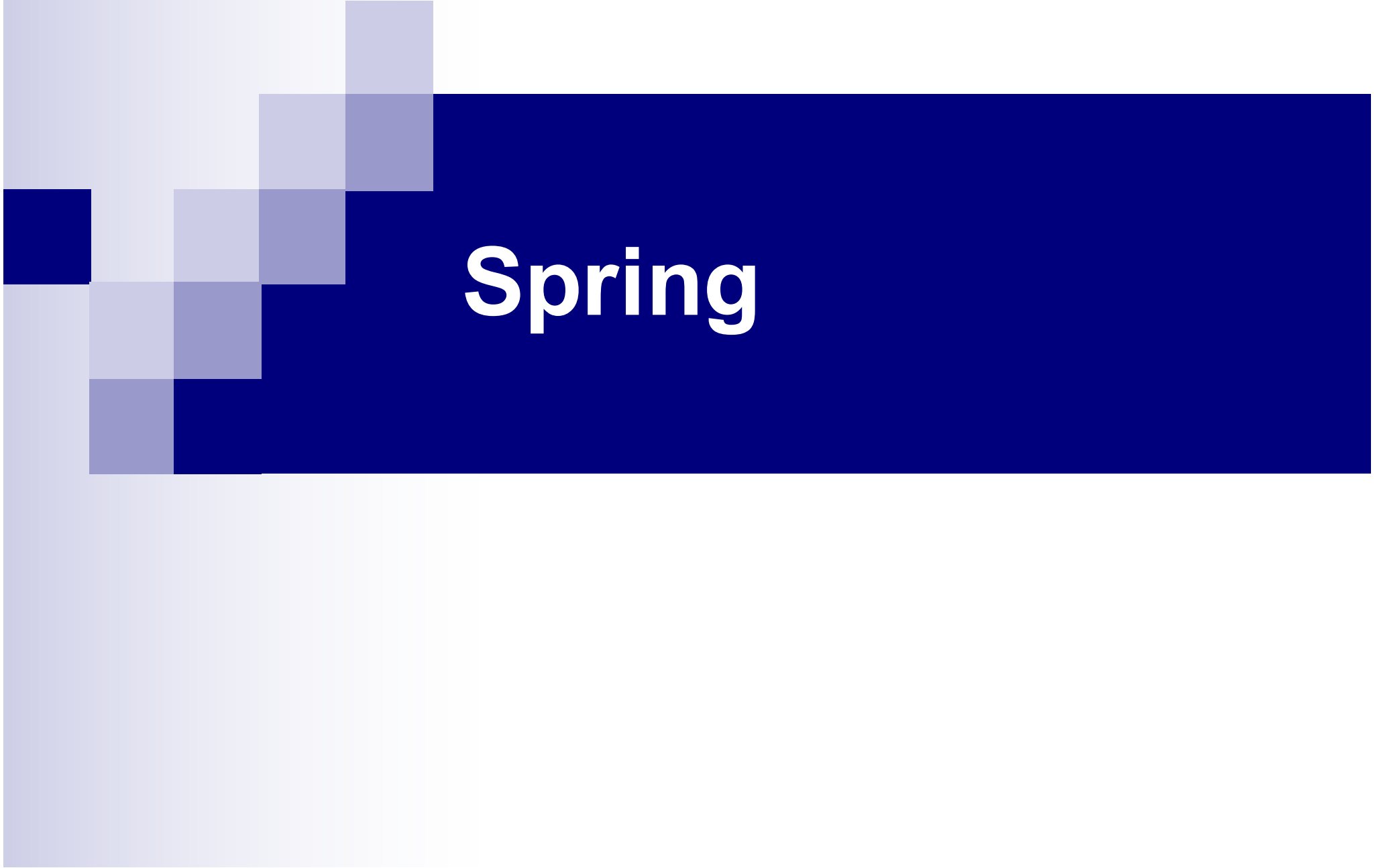

Spring

- SPRING est effectivement un conteneur dit «léger», c à d une infrastructure similaire à un serveur d'application J2EE.
- Il prend en charge la création d'objets et la mise en relation d'objets par l'intermediaire d'un fichier de configuration qui décrit les objets à fabriquer et les relations de dépendances entre ces objets.

- Le gros avantage par rapport aux serveurs d'application est qu'avec SPRING, vos classes n'ont pas besoin d'implementer une quelconque interface pour être prises en charge par le Framework (au contraire des serveurs d'applications J2EE et des EJBs).
- C'est en ce sens que SPRING est qualifié de conteneur «léger».

- Pour assurer une meilleure évolutivité des Applications et aussi améliorer grandement leur testabilité

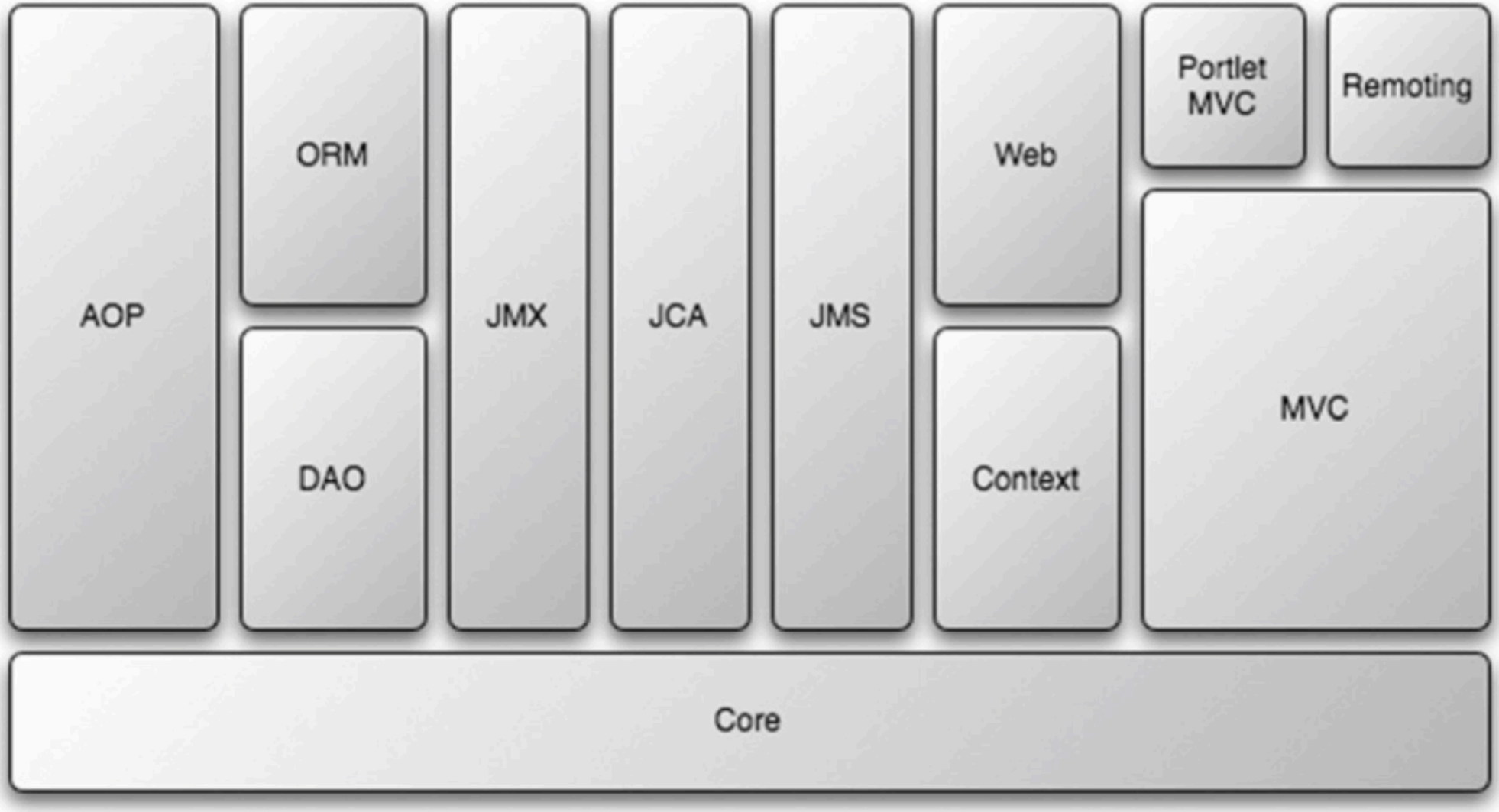
- le conteneur léger encourage les bonnes pratiques de programmation : par interface, et à faible couplage

Les design Pattern

- Spring repose sur des concepts éprouvés, patrons de conception et paradigmes

1. IoC (Inversion of Control),
2. le singleton,
3. la programmation orientée Aspect,
4. Le modèle de programmation dit "par Template".
5.

L’écosystème Spring



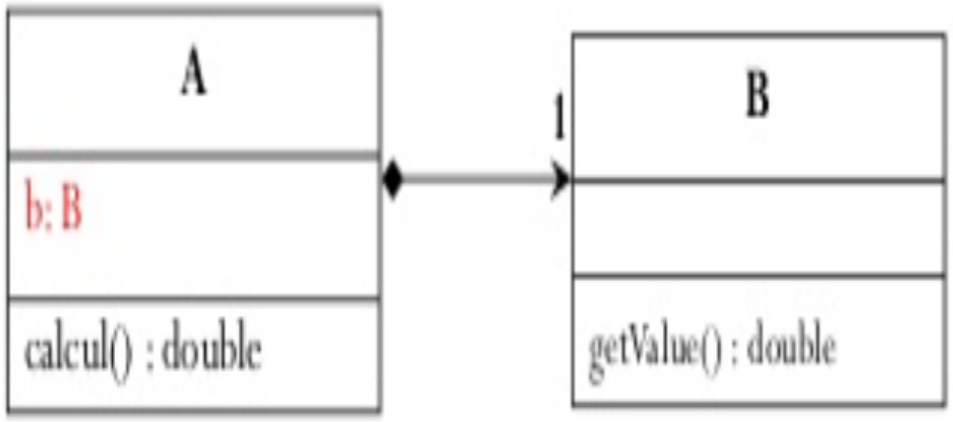
- Core, le noyau, qui contient à la fois un ensemble de classes utilisées par toutes les briques du framework et le conteneur léger.
- AOP, le module de programmation orientée aspect, qui s'intègre fortement avec AspectJ, un framework de POA à part entière.
- DAO, qui constitue le socle de l'accès aux dépôts de données, avec notamment une implémentation pour JDBC

- ORM, qui propose une intégration avec des outils populaires de mapping objet-relationnel, tels que Hibernate, JPA, EclipseLink ou iBatis. Chaque outil peut bénéficier de la gestion des transactions fournie par le module DAO.
- Web, le module comprenant le support de Spring pour les applications Web. Il contient notamment Spring Web MVC, la solution de Spring pour les applications Web, et propose une intégration avec de nombreux frameworks Web et des technologies de vue

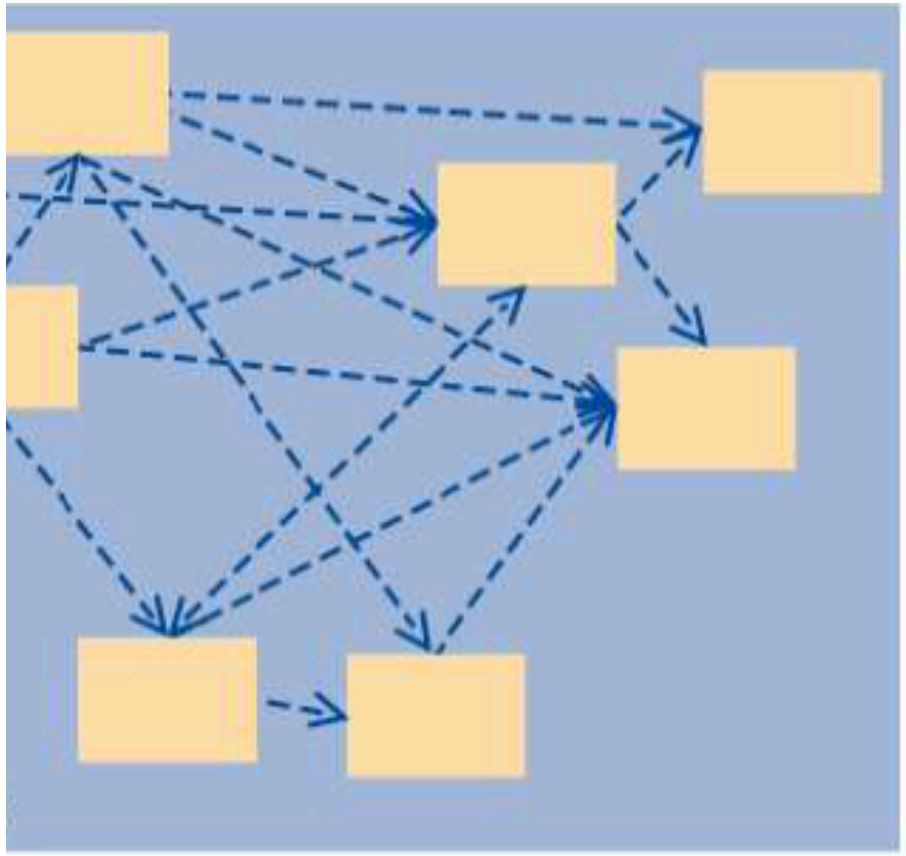
Inverse of Control
IoC

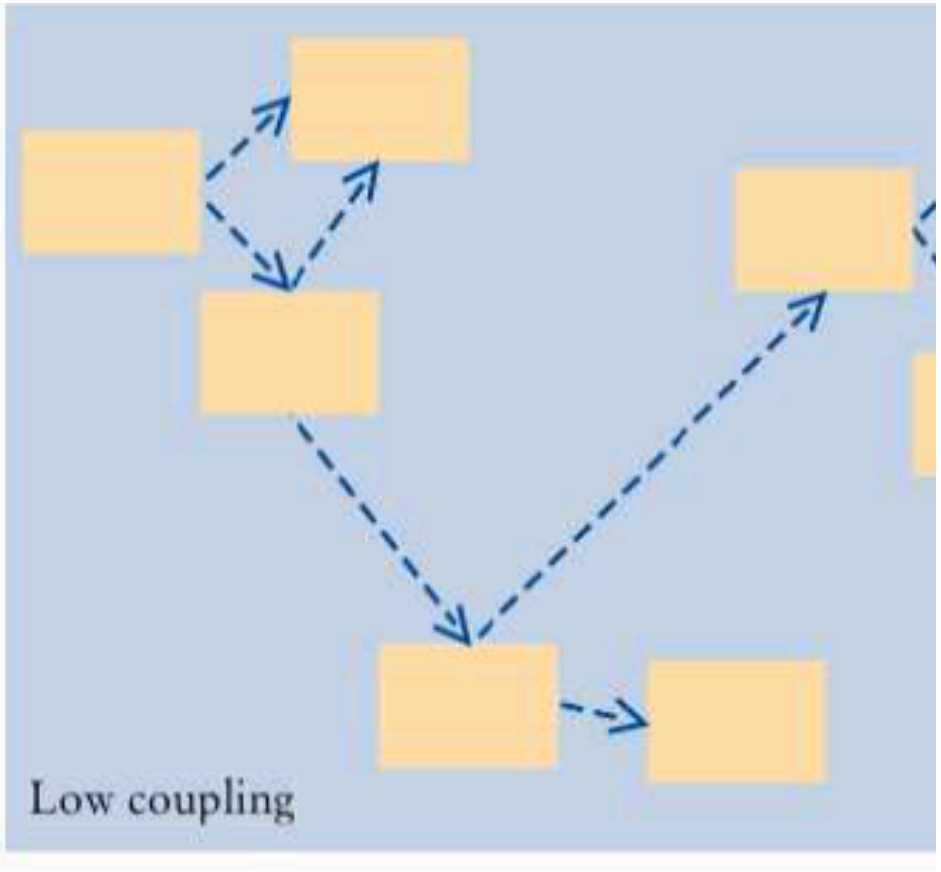
Couplage fort

- Quand une classe A est liée à B, on dit que A est fortement couplée à B.
- A ne peut fonctionner qu'en présence de B.
- Si on modifie B, on est obligé de modifier A.
- Ce qui rend la maintenance de l'application « Mission Impossible »



Couplage fort





```java
public interface IOutputGenerator
{
    public void generateOutput();
}
```

```java
public class CsvOutputGenerator implements IOutputGenerator
{
    public void generateOutput(){
        System.out.println("Csv Output Generator");
    }
}
```

```java
public class JsonOutputGenerator implements IOutputGenerator
{
    public void generateOutput(){
        System.out.println("Json Output Generator");
    }
}
```

Première méthode

```
public class App
{
    public static void main( String[] args )
    {
        IOutputGenerator output = new CsvOutputGenerator();
        output.generateOutput();
    }
}
```

Deuxième méthode : Helper class

```
public class OutputHelper
{
    IOutputGenerator outputGenerator;

    public OutputHelper(){
        outputGenerator = new CsvOutputGenerator();
    }

    public void generateOutput(){
        outputGenerator.generateOutput();
    }

}
```

```
public class App
{
    public static void main( String[] args )
    {
        OutputHelper output = new OutputHelper();
        output.generateOutput();
    }
}
```

Troisième méthode : Réflexion

```
public class OutputHelper
{
    IOutputGenerator outputGenerator;

    public void generateOutput(){
        outputGenerator.generateOutput();
    }

    public void setOutputGenerator(IOutputGenerator outputGenerator){
        this.outputGenerator = outputGenerator;
    }
}
```

```
class App
{
    public static void main( String[] args )
    {
        Scanner scanner = new Scanner(new File("config.txt"));
        String outPutClassName= scanner.next();
        Class outputGenerator = Class.forName(outPutClassName);
        IOutPutGenerator output= (IOutPutGenerator) outputGenerator.newInstance();
        ..........
```

Quatrième méthode :Spring

```xml
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="outputHelper" class="OutputHelper">
        <constructor-arg>
            <bean class="CsvOutputGenerator" />
        </constructor-arg>
    </bean>

<bean id="CsvOutputGenerator" class="CsvOutputGenerator" />
<bean id="JsonOutputGenerator" class="JsonOutputGenerator" />

</beans>
```

```java
public class App
{
    public static void main( String[] args )
    {
        ApplicationContext context =
           new ClassPathXmlApplicationContext(new String[] {"Spring-Common.xml"});

        OutputHelper output = (OutputHelper)context.getBean("OutputHelper");
        output.generateOutput();

    }
}
```

Injection de dépendance

- L'injection des dependances ou IoC est un concept qui intervient generalement au debut de l'execution de l'application.
- Spring commence par lire un fichier XML qui déclare quelles sont les classes a instancier et assurer les dépendances entres les instances.
- Qd on a besoin d'integrer une nouvelle implementation à une App, il suffirait de la declerer dans le fichier XML « bean spring ».

Exercice 1 : Hello world

Initialisation et destruction :

```
<bean id="exampleBean"
        class="examples.ExampleBean" init-method="init"/>
```

```
public class ExampleBean {
   public void init() {
      // do some initialization work
   }
}
```

```
<bean id="exampleBean"
        class="examples.ExampleBean" destroy-method="destroy"/>
```

```
public class ExampleBean {
   public void destroy() {
      // do some destruction work
   }
}
```

Spring DI via setter method

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="outputHelper" class="OutputHelper">
        <property name="outputGenerator"  ref="csvOutputGenerator" />
    </bean>

    <bean id="csvOutputGenerator" class="CsvOutputGenerator" />
</beans>
```

Spring DI via Constructor

```
<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
    http://www.springframework.org/schema/beans/spring-beans-2.5.xsd">

    <bean id="outputHelper" class="OutputHelper">
        <property name="outputGenerator"  ref="csvOutputGenerator" />
    </bean>

    <bean id="csvOutputGenerator" class="CsvOutputGenerator" />
</beans>
```

Exercice 2 : calcul de la superficie d’une forme

Spring Collections

- List – <list/>
- Set – <set/>
- Map – <map/>
- Properties – <props/>

```
public class Customer
{
    private List<Object> lists;
    private Set<Object> sets;
    private Map<Object, Object> maps;
    private Properties pros;

    //...
}
```

```
<property name="lists">
    <list>
        <value>1</value>
        <ref bean="personneBean" />
        <bean class="Personne">
            <property name="name" value="maList" />
            <property name="address" value="address" />
            <property name="age" value="28" />
        </bean>
    </list>
</property>
```

```
</property>
<property name="sets">
    <set>
        <value>1</value>
        <ref bean="personneBean" />
        <bean class="Personne">
            <property name="name" value="mySet" />
            <property name="address" value="address" />
            <property name="age" value="28" />
        </bean>
    </set>
</property>
```

```
</property>
    <property name="maps" >
    <map>
        <entry key="Key 1" value="1" />
        <entry key="Key 2" value-ref="personneBean" />
        <entry key="Key 3">
            <bean class="Personne">
                <property name="name" value="myMap" />
                <property name="address" value="address" />
                <property name="age" value="28" />
            </bean>
        </entry>
    </map>
</property>
```

```
<property name="pros">
        <props>
            <prop key="admin">admin@nospam.com</prop>
            <prop key="support">support@nospam.com</prop>
        </props>
    </property>
```

Scoop

- Singleton
- Prototype
- Request
- Session
- GlobalSession

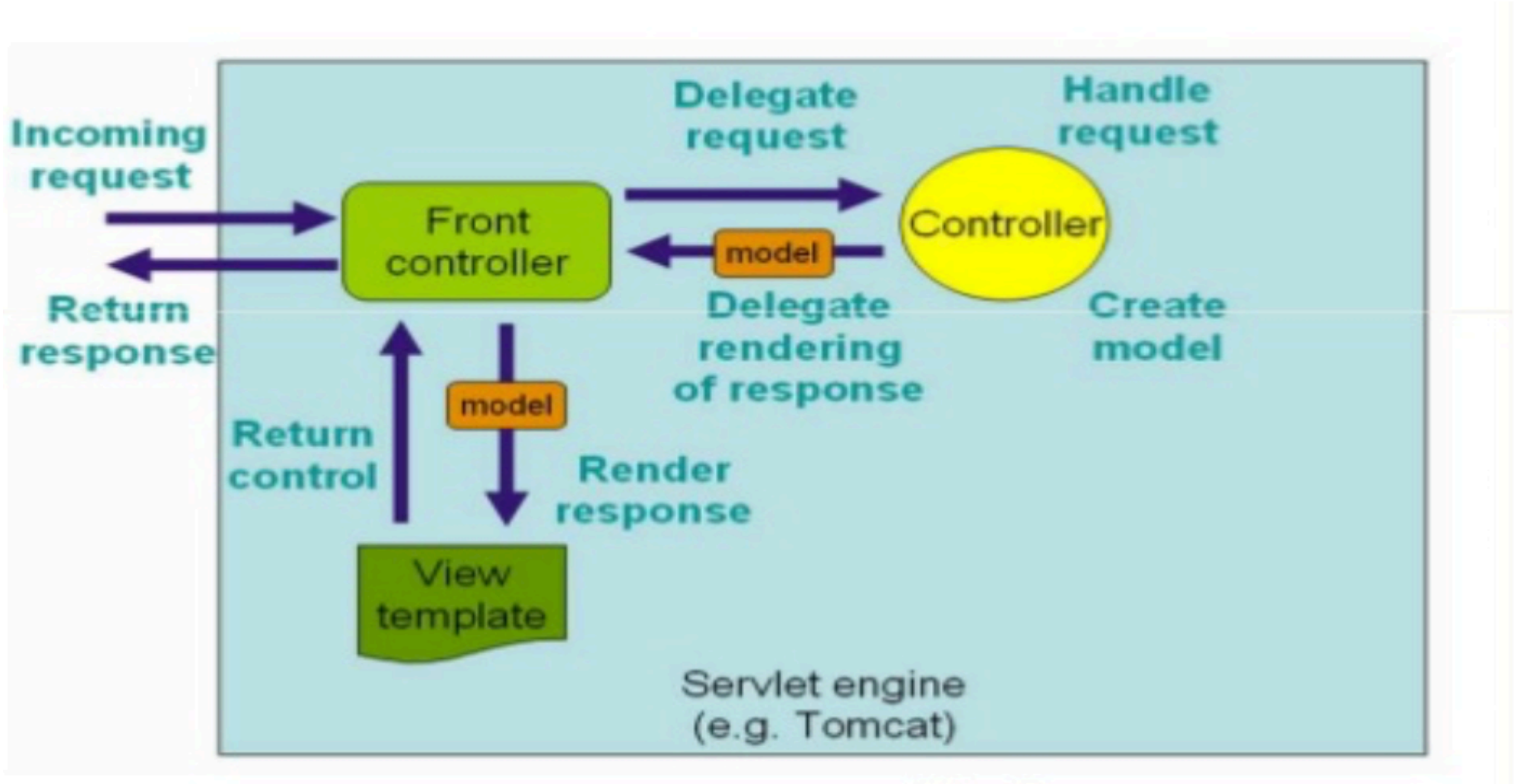
```
public class CustomerService
{
    String message;

    public String getMessage() {
        return message;
    }

    public void setMessage(String message) {
        this.message = message;
    }
}
```

Prototype vs Singleton

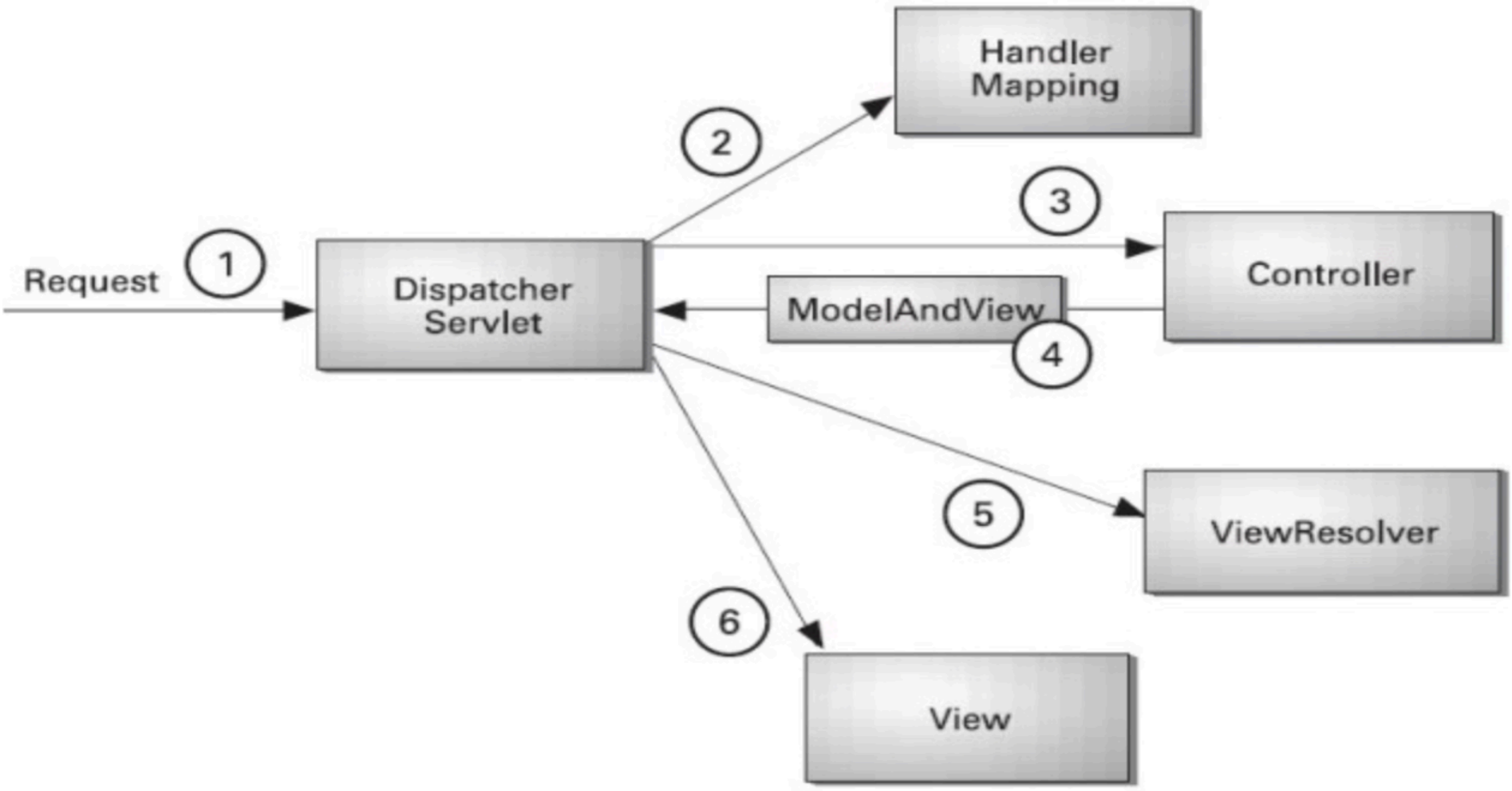
Autowired

Spring MVC

Process de Spring



Process de Spring



Spring MVC

- Après avoir reçu une requête HTTP, DispatcherServlet consulte le HandlerMapping et va appeler le contrôleur approprié.
- Le contrôleur prend la requête et appelle les méthodes de service approprié. La méthode de service fixera les données du modèle basé sur la logique métier défini et retourne le nom de la vue au DispatcherServlet.
- Le DispatcherServlet prendre de l'aide de ViewResolver pour chercher la vue demandée

Spring MVC

- Une fois la vue est finalisée, Le DispatcherServlet transmet les données du modèle à la vue qui est finalement rendue sur le navigateur.

Hello World MVC

```
<web-app id="WebApp_ID" version="2.4"
    xmlns="http://java.sun.com/xml/ns/j2ee"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/j2ee
    http://java.sun.com/xml/ns/j2ee/web-app_2_4.xsd">

    <display-name>Spring MVC Application</display-name>

   <servlet>
      <servlet-name>HelloWeb</servlet-name>
      <servlet-class>
         org.springframework.web.servlet.DispatcherServlet
      </servlet-class>
      <load-on-startup>1</load-on-startup>
   </servlet>

   <servlet-mapping>
      <servlet-name>HelloWeb</servlet-name>
      <url-pattern>*.jsp</url-pattern>
   </servlet-mapping>

</web-app>
```

Hello World MVC

```
<beans xmlns="http://www.springframework.org/schema/beans"
   xmlns:context="http://www.springframework.org/schema/context"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="
   http://www.springframework.org/schema/beans
   http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
   http://www.springframework.org/schema/context
   http://www.springframework.org/schema/context/spring-context-3.0.xsd">

   <context:component-scan base-package="com.tutorialspoint" />

   <bean
class="org.springframework.web.servlet.view.InternalResourceViewResolver">
      <property name="prefix" value="/WEB-INF/jsp/" />
      <property name="suffix" value=".jsp" />
   </bean>

</beans>
```

Hello World MVC

```java
@Controller
@RequestMapping("/hello")
public class HelloController{

   @RequestMapping(method = RequestMethod.GET)
   public String printHello(ModelMap model) {
      model.addAttribute("message", "Hello Spring MVC Framework!");
      return "hello";
   }

}
```

```java
@Controller

public class HelloWorldController {
 @RequestMapping("/hello")
    public ModelAndView helloWorld() {
        String message = "HELLO  MVC FRamework";
        return new ModelAndView("hellopage", "message", message);
    }
```