INTRODUCTION À JAVA EE

LA NÉBULEUSE JAVA

- Java Micro Edition (JME)
  - développement d'applications embarquées
- Java Standard Edition (JSE)
  - développement d'applications classiques
- **Java Enterprise Edition (JEE)**
  - développement d'applications d'entreprise

APPLICATIONS D'ENTREPRISE

peu coûteuse en temps et en budget

portables

disponibles

sécurisée

maintenables

montée en charge

sûres

extensibles

intégrables

adaptables

de qualité

répondent aux besoins exprimés par les utilisateurs !

ARCHITECTURE APPLICATIVE

- application centralisée
- application clients / serveurs
- application distribuée

APPLICATION CENTRALISÉE

APPLICATION CLIENTS / SERVEURS

APPLICATION DISTRIBUÉE

LES BESOINS EXPRIMÉS

- Besoins de normalisation
  - Etablir un ensemble de règles ayant pour objet de simplifier et de rationaliser la production
- Besoins d'abstraction
  - Opération de désolidariser un objet de son contexte
- Besoins de communication
- Besoins de composants

BESOINS DE NORMALISATION

Pour que les applications soient:

- intégrables
- communicantes / distribuées
- adaptables
- maintenables
- portables

BESOINS D'ABSTRACTION

Pour que les applications soient :

- portables
- maintenables
- extensibles
- intégrables / distribuées
- adaptables

BESOINS DE COMMUNICATION

Pour que les applications soient :

- intégrables
- sécurisée
- distribuées

BESOINS DE COMPOSANTS

Pour que les applications soient :

- maintenables
- sûres
- extensibles
- adaptables
- portables
- disponibles / distribuées

et surtout...

Comment réduire les temps et les coûts de développement et d'évolutions d'une application ?

Quelques principes....

PRINCIPE D'OUVERTURE/ FERMETURE

Les composantes d'une application doivent être ouvertes à extension mais fermées à modification !

PRINCIPE DE SUBSTITUTION DE LISKOV

Si S est un sous-type de T, alors les objets de type T peuvent être remplacés avec des objets de type S

PRINCIPE DE SUBSTITUTION DE LISKOV

Programmation par contrat:

- Pré-conditions ne peuvent être plus fortes dans une sous-classe
- Post-conditions ne peuvent être plus faibles dans une sous-classe

UNE SEULE RESPONSABILITÉ

Chaque objet ne doit avoir qu'une seule responsabilité !

Comment respecter ces grands principes ?

UNE PARTIE DE LA SOLUTION...

- Des paradigmes de programmation
- Des patrons de conception
- Des frameworks
- Des composants

QUELQUES RAPPELS

- Inversion de contrôle
- Injection de dépendances
- Programmation par aspects
- Patron de conception

INVERSION DE CONTRÔLE

Le code générique/réutilisable contrôle l'exécution du code spécifique

Don't call us, we call you!

INVERSION DE CONTRÔLE

```
public abstract class WorkerTask< I, O > implements Runnable {
    protected O doPerform(I input);
    public void run() {
        I input = receive();
        O output = doPerform(input);
        send(output);
}

public class ReadWorkerTask extends WorkerTask< Input, Output > {
    @Override
    protected Output doPerform(Input in) {
        InputStream is = in.getInputStream();
        is.read(b);
        Output out = new Output(b);
        return out;
    }
}
```

INJECTION DE DÉPENDANCES

Une manière automatique et directe de fournir une dépendance externe dans un composant logiciel

INJECTION DE DÉPENDANCES

```
public class AsyncWriter {
    @Inject
    public AsyncWriter(BufferingStrategy strategy) {
        this.strategy = strategy;
    }
}

public class AsyncWriterModule extends AbstractModule {
    @Override
    protected void configure() {
        bind(BufferingStrategy.class).to(JPMBufferingStrategy.class);
    }
}
```

INJECTION DE DEPENDANCES

```
public static void main(String[] args) throws Exception {
    Injector injector = Guice.createInjector(new AsyncWriterModule());
    AsyncWriter writer = injector.getInstance(AsyncWriter.class);
}
```

PROGRAMMATION PAR ASPECTS

Augmenter la modularité en améliorant la séparation des préoccupations

PROGRAMMATION PAR ASPECTS

```
public @interface Log {
}

public class LogMethodInterceptor implements MethodInterceptor {
  public Object invoke(MethodInvocation i) throws Throwable {
        System.out.println("Start: " + i.getMethod().getName());
        Object ret = i.proceed();
        System.out.println("End: " + i.getMethod().getName());
        return ret;
    }
}
```

PROGRAMMATION PAR ASPECTS

```
public class MonModule extends AbstractModule {
    @Override
    protected void configure() {
        bindInterceptor(Matchers.inPackage(Package.getPackage(
            "com.ullink.designpattern.test")),
            Matchers.annotatedWith(Log.class),
            new LogMethodInterceptor());
    }
}
```

DESIGN PATTERN

Définition:

Une solution générale et réutilisable d'un problème courant

Nous utilisons des patrons de conception sans forcément le savoir!

DESIGN PATTERN

Pros:

Abstraction

Capitalisation de la connaissance

“Design Patterns – Elements of reusable OO Software”

Cons:

Abstraction

Diluer dans du code

Revenons à notre problématique...

LES BESOINS EXPRIMÉS

- Besoins de normalisation
  - Etablir un ensemble de règles ayant pour objet de simplifier et de rationaliser la production
- Besoins d'abstraction
  - Opération de désolidariser un objet de son contexte
- Besoins de communication
- Besoins de composants

OBJECTIF

Avoir une « plateforme » pour développer des applications d'entreprise rapidement, de qualitées, sûres, sécurisées, portables, performantes, disponibles, maintenables, extensibles et ce... à moindre coûts !

DÉFINITION

Java Enterprise Edition est une norme proposée par Sun visant à définir un standard de développement d'applications d'entreprises multi-niveaux basées sur des composants.

Principes d’architecture

ARCHITECTURE MULTI-NIVEAUX

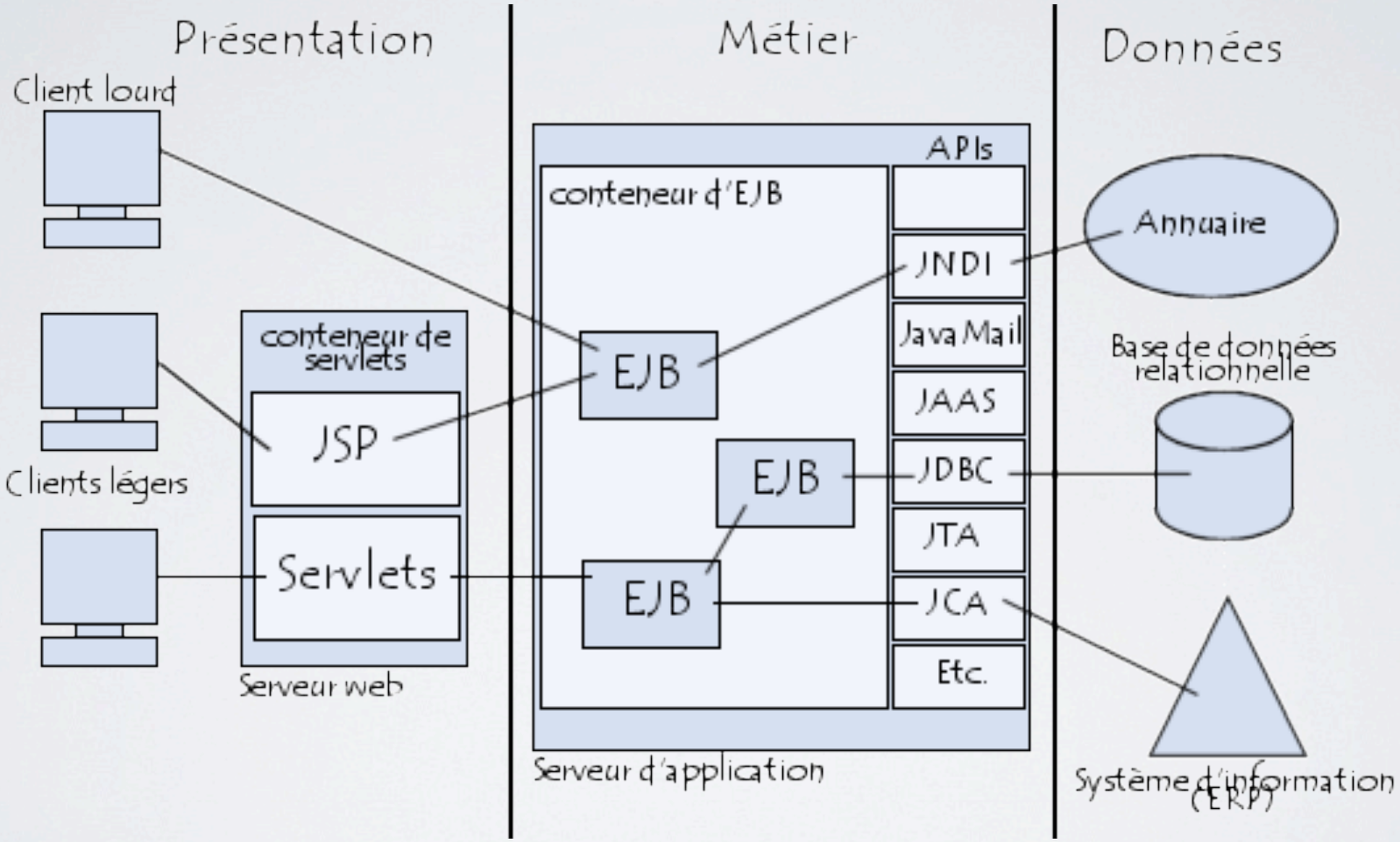
- Un niveau par besoin fonctionnel
- Augmentation de la cohésion du code
- Découplage fort entre les couches
- Code plus facilement réutilisable

ARCHITECTURE JEE

Typiquement c'est une architecture 3-tiers :

- Tiers présentation : affichage des données
- Tiers métier : gestion du métier de l'application
- Tiers donnée : persistance des données

ARCHITECTURE JEE



ARCHITECTURE JEE

Permet une séparation claire entre :

- l'interface homme-machine
- les traitements métiers
- les données

ARCHITECTURE JEE

Basée sur des composants qui sont :

- distincts
- interchangeables
- distribués

ARCHITECTURE JEE

De nouveaux patrons de conception:

- Data Access Object
- Data Transfer Object
- Session Facade
- Front controller
- Modèle Vue Controleur

N’oublions pas une problématique en entreprise...

LES CONTRAINTES DE LA PRODUCTION

- Réduction des coûts
  - Migration d'une version de serveurs d'applications
  - Maintenance de plusieurs serveurs d'applications
- Stabilité du Système d'Information

La production a tendance à freiner l'innovation !

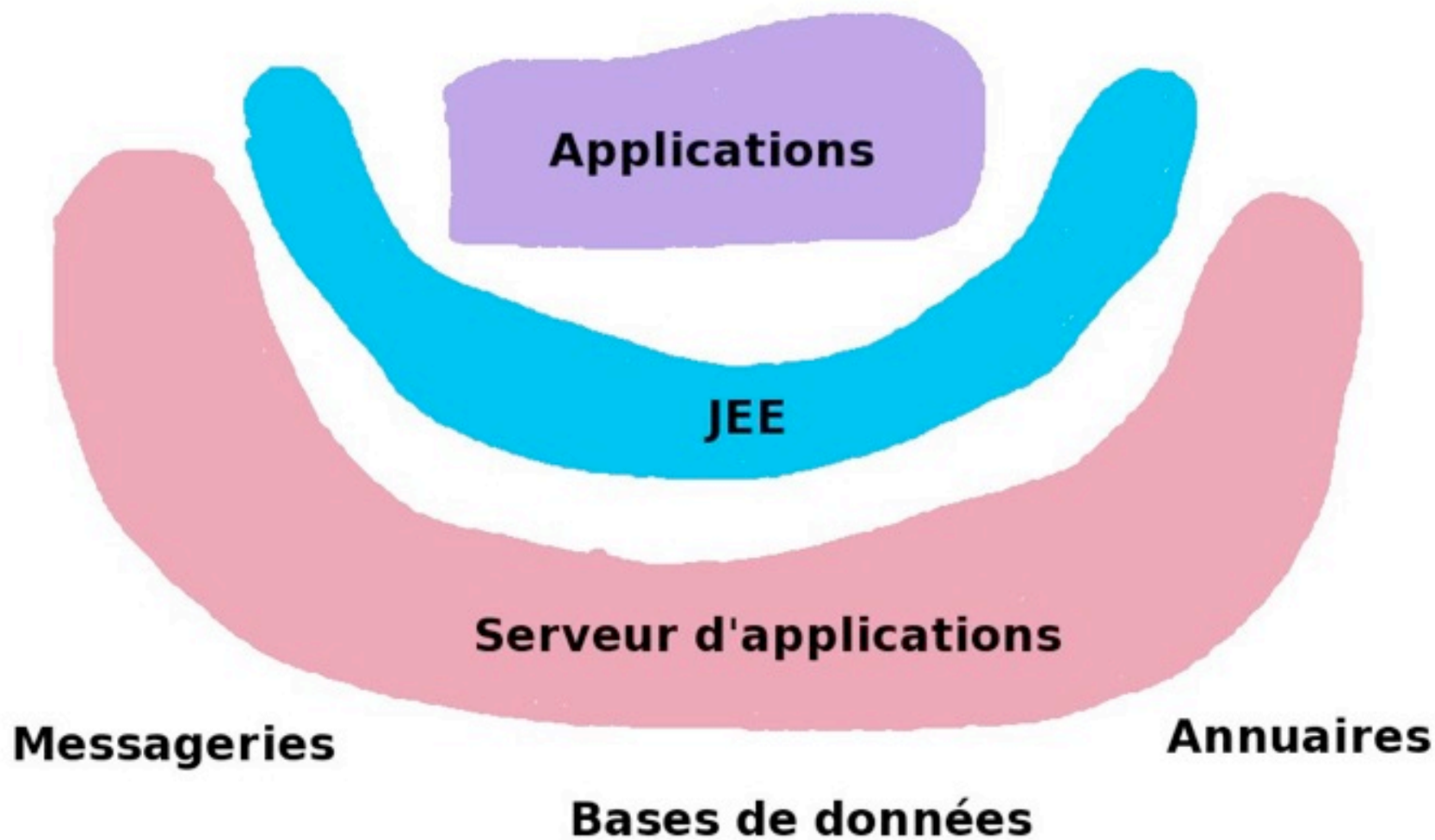
PRODUCTION VS ÉTUDE

- La production est conservatrice / les études sont des acteurs du changement

PROBLÉMATIQUES

- Respecter les contraintes de la production
- Continuer à innover

ARCHITECTURE JEE



EXEMPLE

- Vous faites du JEE depuis 2003, la production a certifié une version d'un serveur d'application compatible JEE 1.3

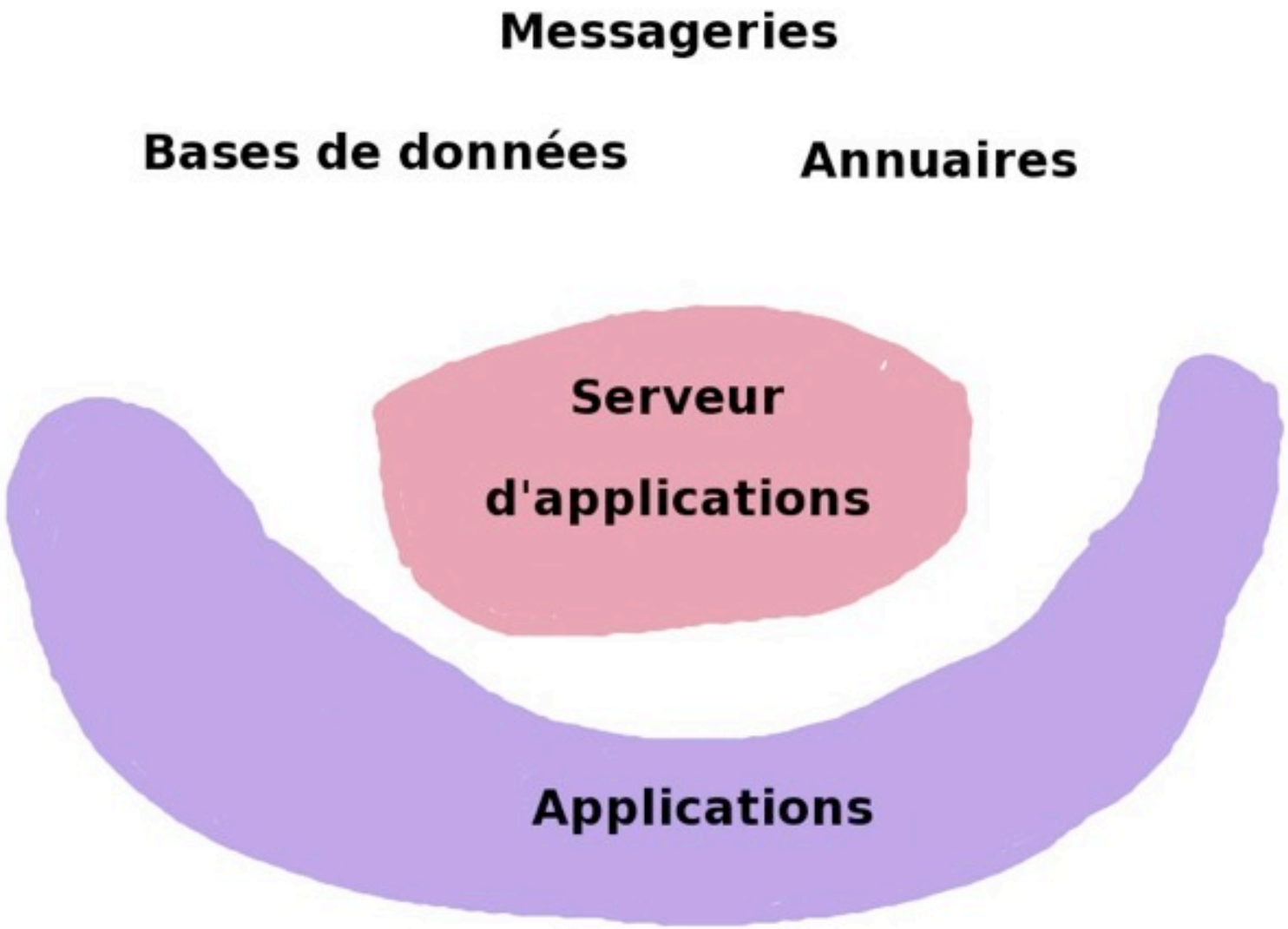
- En 2008, tant que la production n'a pas homologué et certifié JEE 5, vous ne pouvez pas l'utiliser!

Bloqué dans l'amélioration de votre productivité, dans votre innovation!

Pourquoi ne pas inverser la relation entre le container et l'application ?

Ce n'est plus le container qui contient l'application mais l'application qui contient le container !

CONTENEUR LÉGER



CONTENEUR LÉGER

- Gestion du cycle de vie des objets
- Description des relations entre les objets
- Similaire à un serveur d'application classique
- Les objets ne doivent pas implanter une interface particulière pour être pris en charge par le framework (différence avec les serveurs d'application JEE / EJB

EXEMPLE

- La production a homologué un conteneur de servlet
- Une nouvelle version de votre conteneur léger arrive

- La production n'a rien à homologuer

Utilisation immédiate sans validation par la production!

DE L'AGILITÉ...

Toute la valeur du passage à un conteneur léger est là!

QUELQUES EXEMPLES

- Spring
- Pico
- Avalon
- HiveMind

Pourquoi vous présentez SPRING ?

Spring apporte l'inversion de contrôle

Spring apporte la programmation par aspect

Spring apporte une couche d'abstraction

JEE apporte aussi ses principes !

mais...

Spring apporte l'inversion de contrôle

Spring apporte la programmation par aspect

Spring apporte une couche d'abstraction

Spring apporte aussi pleins de connecteurs !

Spring améliore la qualité et les coûts de production d'applications

SPRING

Spring apporte beaucoup de choses:

- Spring Batch
- Spring MVC
- Spring webflow
- Spring security
- AOP
- Connecteurs JDBC, Hibernate, iBatis, LDPA, ...

UN STANDARD DU MARCHÉ

SPRING

- Simple
- Standard de fait
- Tient très bien la charge
- Répond à nos problématiques!

SPRING UNE RÉPONSE AUX BESOINS EXPRIMÉS

- Besoins de normalisation
  - Etablir un ensemble de règles ayant pour objet de simplifier et de rationaliser la production
- Besoins d'abstraction
  - Opération de désolidariser un objet de son contexte
- Besoins de communication
- Besoins de composants