Nuxeo@Eclipse Loops

Objectif

- Installer le plugin Nuxeo IDE
- Configurer Nuxeo IDE
- Créer un service qui stocke les résultats d'une classe
- Accepte différents formats (XML, CSV, etc.)
- Exposer en REST

Installer Nuxeo IDE : les sources

- Télécharger les sources de Nuxeo
  - Github => long
  - SDK
    - Avec une distribution Nuxeo
    - Rapide

http://community.nuxeo.com/static/releases/nuxeo-5.6/nuxeo-cap-5.6-tomcat-sdk.zip

- Dézipper

Nuxeo IDE : Objectif

- Simplifier l'écriture des codes d'infrastructure ennuyeux
- Focaliser sur votre code
- Chargement à chaud de votre code sur le serveur
- Intégration à Nuxeo Studio

Installer Nuxeo IDE : plugin

- Aller dans marketplace.eclipse.org
- Rechercher Nuxeo
- Glisser-Déposer le bouton install dans Eclipse

Installer Nuxeo IDE : Lier

- Dans Préférences d'Eclipse > Nuxeo
- Aller dans Nuxeo SDK
- Donner le chemin de la racine du SDK
- Cocher
- Ok

Vous avez tout l'environnement de développement Nuxeo

Créer un bundle Nuxeo

- Rappel un bundle est jar pouvant contenir
  - un ou + services
  - un ou + points d'extension
  - des resources

Créer un bundle Nuxeo

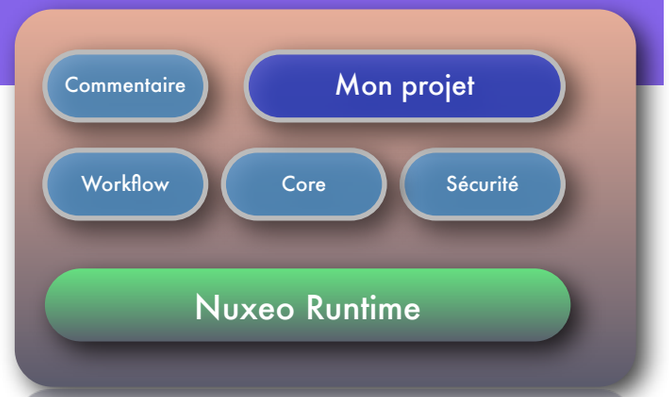
- Dans Nuxeo IDE
  - Cliquer sur le bouton Nuxeo 

  - Choisir le template *Nuxeo Plugin Project*
    - Project Id sera le nom du Bundle (**Run**)
    - Maven Settings - Identifier le projet dans Maven (**Build**)
      - Maven est un outil de build Java (comme ant)
      - Chaque projet maven est identifié de manière unique dans le cadre du **build**

Créer un bundle Nuxeo

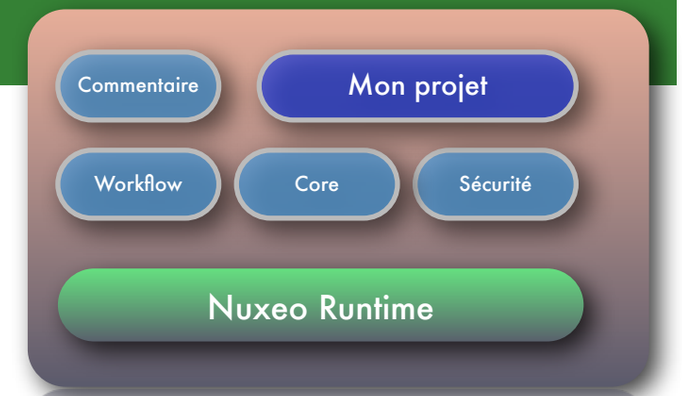
- 2 Fichiers intéressants
  - pom.xml : carte d'identité maven => build
  - MANIFEST.MF : carte d'identité pour la JVM/OSGi => run

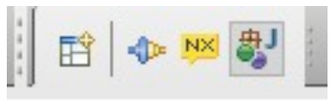
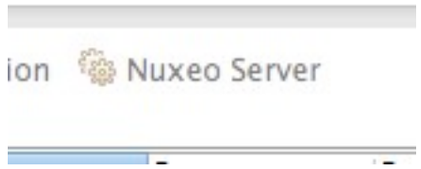
Déployer un bundle

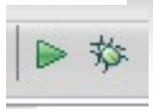
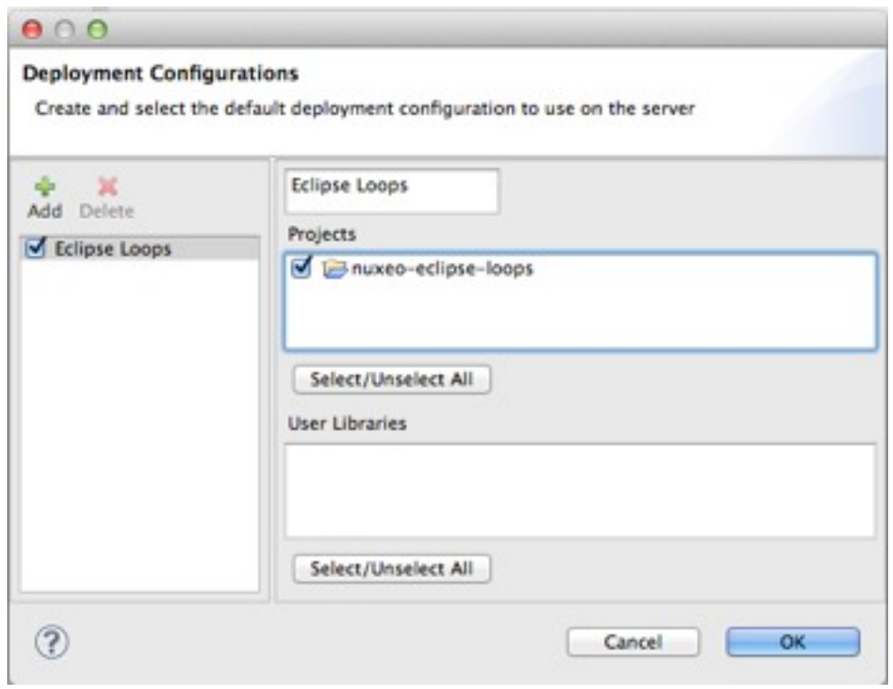


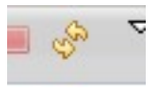
- Active le code dans le serveur d'application
- Active les points d'extension
- Active les contributions aux points d'extension
- Deploie les ressources web

Déployer un bundle



- Aller dans la perspective Nuxeo 
- Cliquer sur la view *Nuxeo Server* 

- Démarrer le serveur en mode debug ou non 
- Créer une configuration de déploiement 

  - Ajouter une nouvelle configuration
  - Nommer
  - Sélectionner le projet
  - Valider
- Déployer à chaud avec le bouton *refresh* 

Déployer un bundle

- Le bundle est déployé mais sans aucune fonctionnalité

Ajouter des ressources web

- *src/main/resources* est copié à la racine du jar par Maven
- *web/nuxeo.war* est déployé dans les resources web par Nuxeo Runtime

- Créer une page *hello_world.html*
- Redéployer votre bundle
- Aller au *localhost:8080/nuxeo/hello_worl.html*

Créer un service

- Les classes Java doivent être créées dans *src/main/java*
- Les tests
  - dans *src/test/java* pour les tests unitaires
  - dans *src/test/resources* pour les resources utiles pour les tests

Créer un service

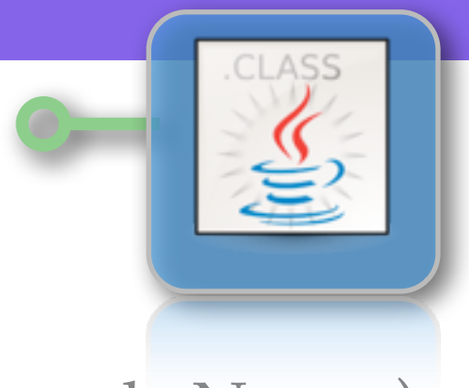
- Cliquer sur le bouton *Nuxeo*
- Choisir le template *Nuxeo Component*
- Renseigner la classe *ResultParserServiceImpl* implémentant le service
- Cocher la case *Expose this component as a service*

Créer un service

- Eléments intéressants
  - *OSGI-INF/extensions/.....xml* composant décrivant le service
  - *MANIFEST.MF* description du composant pour le runtime

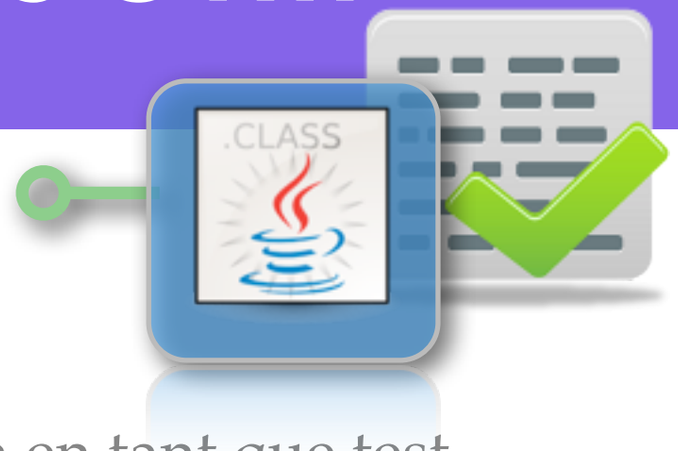
- Il est possible (et conseillé) de créer une interface *MonService*
- Penser à remplacer le nom de l'interface dans la balise service dans le xml

Créer un service



- Récupération de votre service en JAVA (ou ceux de Nuxeo)
  - *Framework.getLocalService(MonService.class)*
- Les services fournis par défaut par Nuxeo sont visible dans
  - explorer.nuxeo.org > 5.7-xxxx > Explore > Services
  - Exemple le ConversionService qui implémente toutes les logiques de conversions de Nuxeo

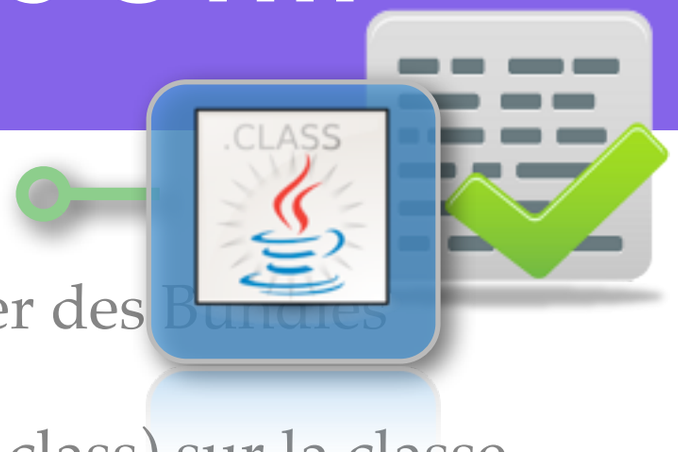
Tester le service : JUnit



- Créer une classe standard dans *src/test/java*
- *@Test* décrit une méthode que JUnit exécutera en tant que test
- *@Before* et *@After* sont exécuté avant et après chaque test
- les méthodes d'assertions sont disponibles dans *org.junit.Assert*

- Executer un test : Click Droit sur la méthode > *Run As* > *JUnit Test*
- Executer tous les tests d'une classe : idem mais click droit sur la classe
- Executer tous les tests d'un projet : idem mais sur le répertoire *src/test*

Tester le service : JUnit



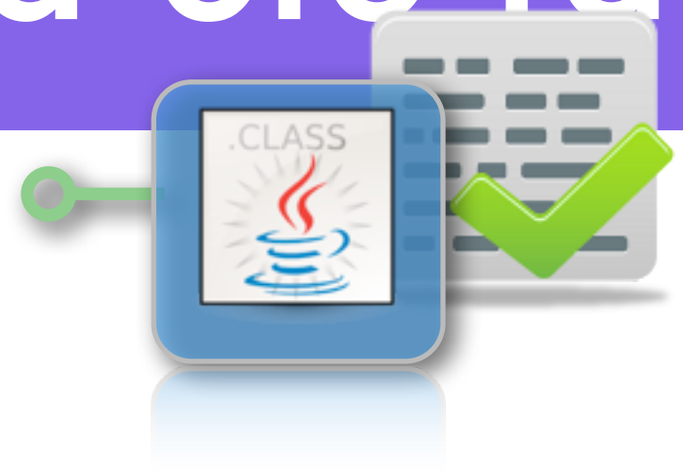
- Nuxeo fournit un Runtime dans JUnit pour déployer des Bundles
  - Ajouter l'annotation @RunWith(FeaturesRunner.class) sur la classe
  - Ajouter @Features(RuntimeFeature.class) pour activer le runtime
    - Ajouter @Deploy("leNomDuBundle") déploie le bundle nommé
      - le nom du bundle est dans le Manifest => bundle-symbolic-name

- La récupération d'un service (pas uniquement dans JUnit)
  - Framework.getLocalService(NomDuService.class)

Tester le service : JUnit

- Objectif du service
  - Gestion des résultats des expériences
- Ajouter et couvrir d'un test une méthode
  - Result addResult(File csvResult)

Résumé de ce qui a été fait

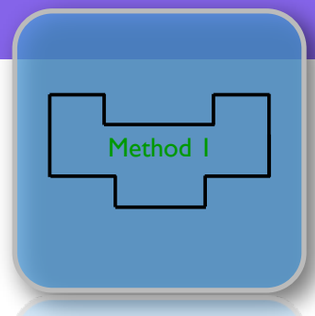


- Création d'un projet Nuxeo qui produit un bundle
- Ajout d'un service
- Création d'un test et de l'implémentation du service

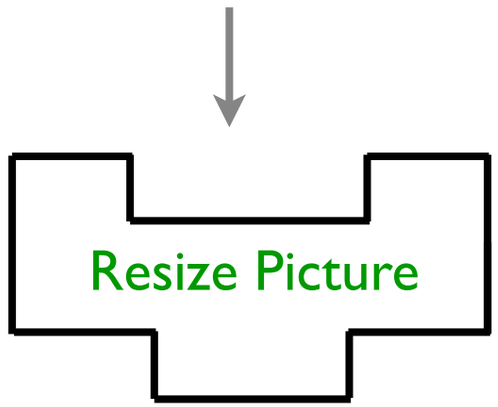
Création d'un 2ème bundle

- Créer un deuxième projet Nuxeo avec un service qui persiste les objets résultat
  - Result storeResult(Result result)
  - List<Result> getResults()
  - persistence en mémoire dans une map
- Modifier le premier service qui utilise le 2ème
- Tester l'intégration

Operation rappel



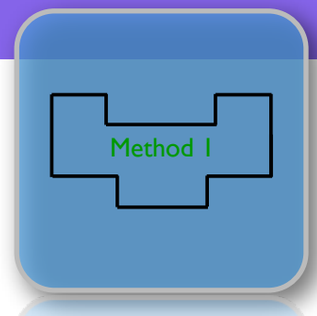
input => Fichier



height = 10px
width = 100px

output => Fichier

Operation



- En Nuxeo 5.6 uniquement Blob / Document
  - en cause la librairie cliente
- En Nuxeo 5.7 choix libre

Core Repository Nuxeo

- La Base documentaire est le coeur du système
- CoreSession guichet unique d'accès
- DocumentModel représentation d'un document

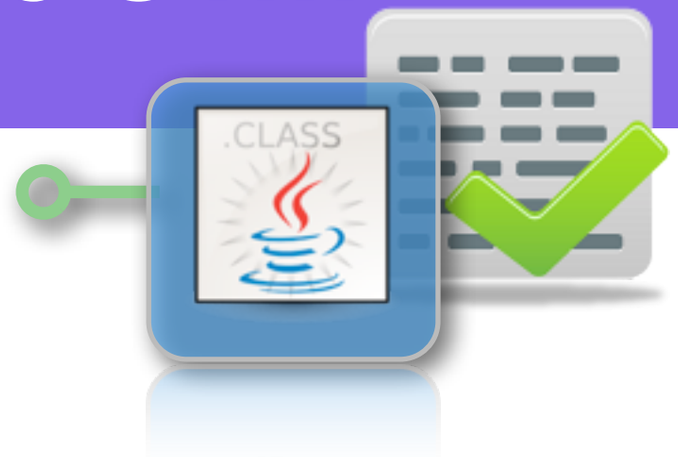
Exemples de code

```
// CREATION
DocumentModel mydoc = coreSession.createDocumentModel("File");
// DocumentModel mydoc = coreSession.createDocumentModel("/", "toto", "File");
mydoc.setPathInfo("/", "toto");
mydoc.setPropertyValue("dc:title", "Toto");
mydoc = coreSession.createDocument(mydoc);
coreSession.save();

// MODIFICATION
mydoc.setPropertyValue("dc:description", "My Description");
mydoc = coreSession.saveDocument(mydoc);
coreSession.save();

// RECUPERATION
DocumentModelList docs = coreSession.query("SELECT * FROM Document WHERE dc:title = 'Toto'");
assertEquals(1, docs.size());
mydoc = docs.get(0);
assertEquals("toto", mydoc.getName());
assertEquals("Toto", mydoc.getPropertyValue(“dc:title”));
assertEquals("My Description", mydoc.getPropertyValue(“dc:description”));
```

Tester le service : JUnit



- Nuxeo fournit un FeatureRunner dans JUnit pour activer un Repository Documentaire
  - Remplacer @Features(RuntimeFeature.class)
  - Par @Features(CoreFeature.class)
- Récupération de la session
  - @Inject public CoreSession session;

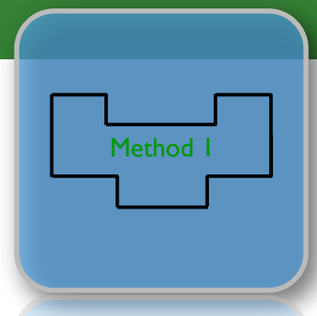
Test unitaire

- Créer un Test unitaire qui crée un document de type “Note”
- Le récupérer par une requête sur le titre
- Vérifier qu’il est bien de type “Note”

DocumentAdapter

- Pattern de Factorisation de la transformation entre 2 Objets
  - DocumentModel : Objet générique persisté dans Nuxeo
  - Result : BusinessModel utilisé par notre service

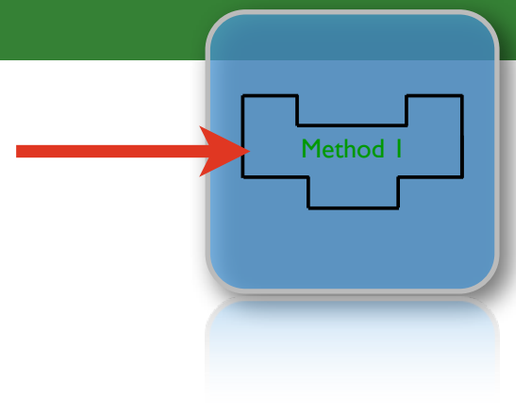
Exposer le service en tant qu'opération



- Cliquer sur le bouton *Nuxeo* 

- Choisir le template *Operation*
- Saisir un nom et un Id pour votre opération : *MonOperation*
- La catégorie est utilisée dans Nuxeo Studio
- Input et Output décrivent l'entrée/sortie de l'opération
- Iterable operation si le traitement peut être itéré

Appel REST d'une opération : Java

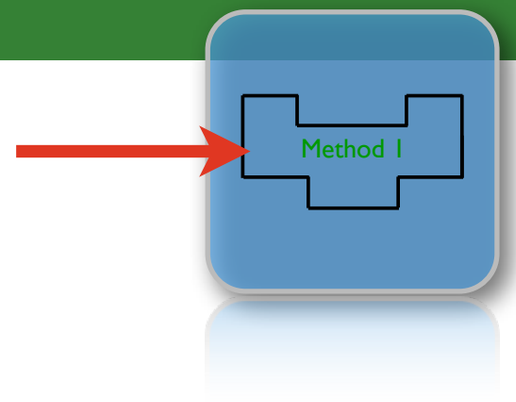



```
HttpAutomationClient client = new HttpAutomationClient(
        "http://localhost:8080/nuxeo/site/automation");

Session session = client.getSession("Administrator", "Administrator");
session.newRequest("MonOperation").execute();

client.shutdown();
```

Appel REST d'une opération : PHP




```
$client = new PhpAutomationClient('http://localhost:8080/nuxeo/site/automation');

$session = $client->getSession('Administrator','Administrator');

$answer = $session->newRequest("MonOperation");
```