ircam Centre Pompidou

UPMC SORBONNE UNIVERSITÉS

JAVA
- Gestion d'un projet -

Ninon Devis: ninon.devis@ircam.fr

Philippe Esling: esling@ircam.fr

License 3 Professionnelle - Multimédia

Plan du cours

I. IDE

II. JUnit

III. Gestionnaire de version

IV. Documentation Java

V. Projet

IDE *Integrated Development Environment*

Logiciel regroupant plusieurs fonctionnalités afin de développer un logiciel:

- Éditeur de texte: raccourcis clavier, auto-complétion, navigation dans le code source…
- Automation de la compilation: gestion des dépendances, mode de test/debug/production.
- Debugger: exécution du code pas à pas pour comprendre un comportement ou un bug.

➔ Il faut le configurer et l'apprendre afin qu'il soit utile

➔ Maîtriser les raccourcis claviers afin de gagner en rapidité et ne pas rompre son workflow

➔ 3 IDE principaux pour Java: **Eclipse**, **NetBeans** et **IntelliJ**

JUnit

Framework de test pour Java

- Principe du test unitaire: méthode permettant de vérifier le bon fonctionnement d'une portion de programme appelée *unité*.
  - Exécuter indépendamment le bout de programme en créant un environnement d'exécution spécifique à la portion de code.
  - Permet d'éviter de devoir recompiler tout un logiciel.
- Pourquoi tester son projet?

➔ Vérifier qu'une modification de code n'entraîne pas de bugs.

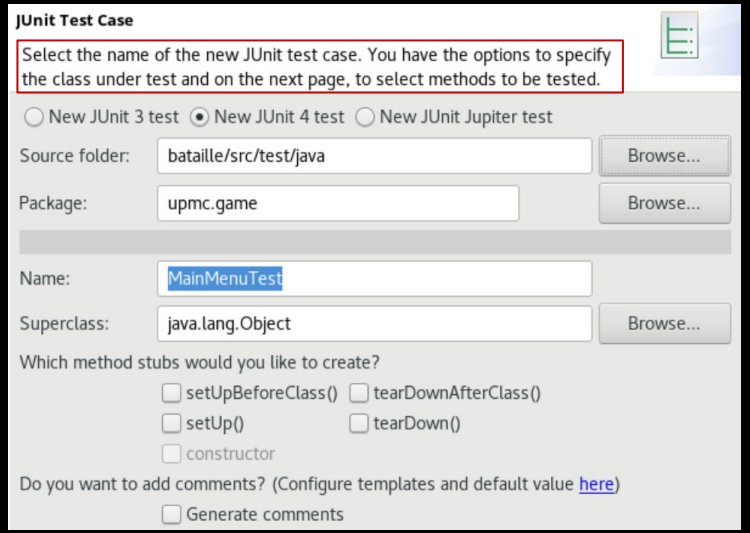
➔ Gagner du temps lors du debug.

➔ Parfois requis en entreprise.

JUnit

Framework de test pour Java

- Comment créer un test JUnit ?
  - `Eclipse`: clic droit sur le fichier à tester, puis `New > JUnit Test Case`.
  - `NetBeans`: clic droit puis Tools > Create Test



- Création d'une nouvelle classe pour chaque classe testée.
- Autant de méthodes que de tests indépendants.
  - Pas de limite dans le nombre de tests qu'on peut écrire.
  - Généralement au moins un test par méthode de la classe testée.

JUnit *Ecrire un test*

Exemple:

```
1  import static org.junit.Assert.*;
2  import org.junit.Test;
3
4  public class StringTest {
5
6      @Test
7      public void testConcatenation() {
8          String foo = "abc";
9          String bar = "def";
10         assertEquals("abcdef", foo + bar);
11     }
12
13     @Test
14     public void testStartsWith() {
15         String foo = "abc";
16         assertTrue(foo.startsWith("ab"));
17     }
18
19 }
```

➔ Permet d'accéder aux méthodes statiques `assertTrue, assertFalse, assertEquals, assertNull.`

➔ Test indépendant de la méthode `testConcatenation()`

➔ Seules les méthodes annotées avec @Test sont testées

➔ assert(Expr) vérifie l'expression donnée

JUnit

Factoriser les éléments communs entre tests

```java
import static org.junit.Assert.*;
import org.junit.Test;
import org.junit.*;

public class StringTest {

    private String foo;
    private String bar;

    @Before
    public void setup() {
        foo = "abc";
        bar = "def";
    }

    @After
    public void tearDown() {
        // on peut avoir besoin de fermer une connexion
        // à une base de données ou de fermer des fichiers
    }

    @Test
    public void testConcatenation() {
        assertEquals("abcdef", foo + bar);
    }

    @Test
    public void testStartsWith() {
        assertTrue(foo.startsWith("ab"));
    }

}
```

➔ Permet d'accéder à @Before et @After

➔ Sera appelé avant chaque test

➔ Sera appelé après chaque test

- Permet de factoriser les éléments communs à tous les tests d'une seule classe.
- Le test commence toujours par l'initialisation de quelques instances de formes différentes pour pouvoir tester les différents cas.
- On peut factoriser le code d'initialisation commun aux tests dans une méthode qui sera appelée avant chaque test.

JUnit

Un bon test : la méthode FIRST

- Fast : Il faut qu'il soit assez rapide pour être lancé fréquemment.
- Isolated : Pas d'interaction avec base de données, réseau, . . .
- Repeatable : Les tests sont déterministes, pas aléatoires, si le test échoue il doit échouer tout le temps et vice-versa si il réussit.
- Self-validating : Vérifier si un test est correct ou non ne doit pas nécessiter notre intervention (afficher sur la console et vérifier à l'oeil nu n'est pas un test)
- Timely : N'écrivez plus de Java sans le tester, dès maintenant, une méthode ≡ un test.

Gestionnaire de version

Permet de...

- Travailler facilement à plusieurs via plusieurs machines sur un projet.
- Garder une trace de toutes les modifications.
- Revenir en arrière sur une partie ou tout le projet.

Deux grandes familles:

- Gestionnaire de version centralisé (Subversion (svn), CVS…)
- Gestionnaire de version décentralisé (Git, Mercurial…)

➔ Tous les développeurs possèdent une copie locale. Un serveur conserve les anciennes versions des fichiers, les utilisateurs s'y connectent pour récupérer les nouvelles modifications ou y envoyer les leur.

➔ Plus largement utilisé car plus rapide, plus sûr, possibilité de travailler sans être connecté au serveur.

Gestionnaire de version

Introduction à Git

Avantages de Git:

- est très rapide.
- sait travailler par “branches” (versions parallèles d’un même projet) de façon flexible.
- Existence de sites web collaboratifs basés sur Git comme GitHub.

Excellent tutoriel pour vous lancer (ou vous familiariser) dans Git et GitHub:

https://openclassrooms.com/fr/courses/1233741-gerez-vos-codes-source-avec-git

Toutes les commandes git sont à taper dans la console.

Gestionnaire de version

Principes de Git

- Il faut d'abord créer un nouveau répertoire à la racine du projet avec `git init.`
- Lorsque vous ajoutez un fichier avec `git add`, il faut valider cet ajout avec `git commit`.
- Les changements sur les fichiers ajoutés aux projets sont validés en local, le commit n'envoie pas les changements sur le serveur distant.
- Pour envoyer vos `commits` (modifications) sur le serveur distant et les voir apparaître sur GitHub faire un `git push`.
- Lorsqu'une personne a push des changements que l'on veut récupérer faire un `pull` pour recevoir les changements du serveur.
- Pour récupérer en local le dossier d'un projet git, utiliser `git clone`.
- Il existe des plug-ins pour git qui sont intégrables à l'IDE, pour ajouter un nouveau fichier: clique-droit > git > ajouter.

Gestionnaire de version

Bonnes habitudes

- Lorsque vous travaillez sur un projet, seul.e ou à plusieurs, faites des commits plusieurs fois par jour.

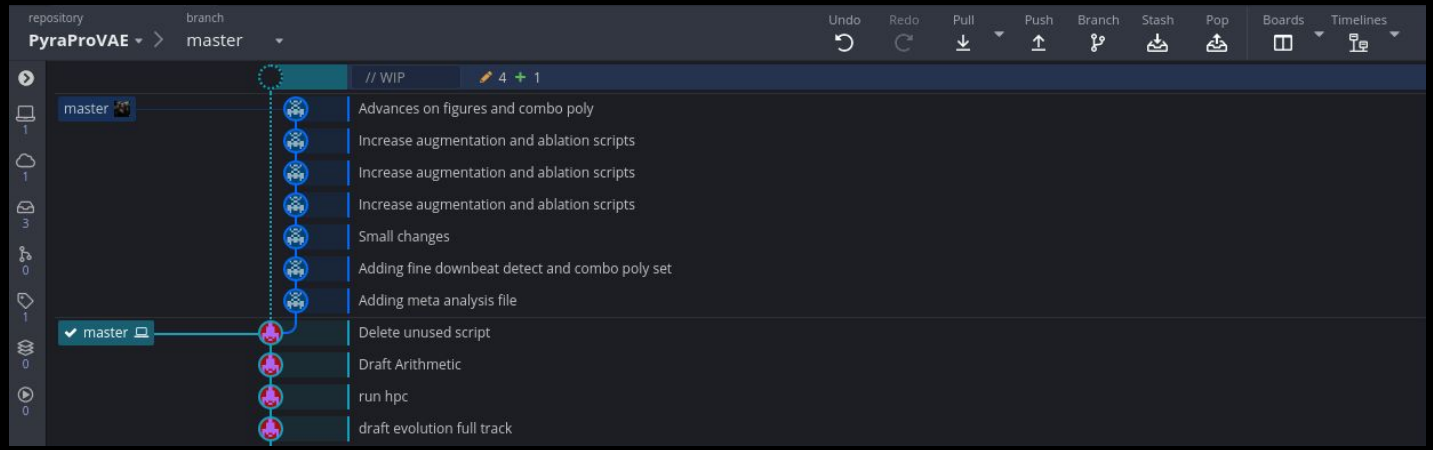
➔ Permet d'enregistrer toutes les étapes de votre travail et revenir en arrière si besoin.

- Faites environ un push par jour pour valider votre travail de la journée et permettre aux autres développeurs de récupérer vos modifications le lendemain matin.
- N'oubliez pas de pull tous les jours afin de récupérer les modifications des autres et travailler sur la version la plus récente du projet.

Gestionnaire de version

Bonnes habitudes

- Si l'interface console ne vous plait pas, GitKraken est un logiciel spécialement conçus pour Git en entreprise.



- Github Student Developer Pack ici => https://education.github.com/pack

  Version pro de très nombreux logiciels !

Documentation Java

Lien pour la documentation de Java 8: https://docs.oracle.com/javase/8/docs/api/index.html

Il est facilement possible de faire une documentation comme celle de Java en documentant votre code avec des balises spéciales.
Pour rappel, il existe trois types de commentaires:

```java
/**
 * Ceci est un commentaire de documentation Java.
 * Il commence par un slash suivis de deux étoiles.
 * Chaque ligne doit ensuite commencer par une étoile.
 * Enfin, il fini par une étoile suivie d'un slash.
 */
protected Vector<Zero> getVectorAmis(){
        // Ceci est un commentaire sur une ligne
        Vector<Zero> vector = new Vector<Zero>();
        /* Ceci est un commentaire sur
        plusieurs lignes */
        for (Zero z : listeAmis){
                vector.add(z);
        }
        return vector;
}
```

➔ Celui là vous permet de réaliser une Javadoc
Doit se trouver sur la ligne immédiatement avant le nom de la classe, de la méthode ou de la variable.

Documentation Java

Les tags JavaDoc permettent de détailler les informations sur chaque élément:

- @author: auteur
- @param: renseigne les paramètres de la méthode, derrière le tag écrire le nom du paramètre, son type sera automatiquement inclus.
- @return: l'objet retourné par la méthode
- @throws: présence d'une exception, indiquer le type et la raison de l'exception
- @see: faire référence à une autre méthode/classe

➔ La première phrase de documentation doit être une courte description.
➔ Il est possible d'utiliser des balises HTML, restez simples: <b>, <i>, <ul> et <p>

Documentation Java

Exemple de documentation de méthode:

```java
/**
 * Valide un mouvement de jeu d'Echecs.
 * @param leDepuisFile   File de la pièce à déplacer
 * @param leDepuisRangée Rangée de la pièce à déplacer
 * @param leVersFile     File de la case de destination
 * @param leVersRangée   Rangée de la case de destination
 * @return vrai (true) si le mouvement d'échec est valide ou faux (false) sinon
 */
boolean estUnDeplacementValide(int leDepuisFile, int leDepuisRangée, int leVersFile, int leVersRangée)
{
    ...
}
```

Bonnes pratiques:

- Privilégier la documentation de plus haut niveau avant de documenter les spécifiques.
- Commencer par la documentation en tête de classe puis se demander: “Qu’est ce que quelqu’un qui arrive sur ce code a besoin de savoir ?”