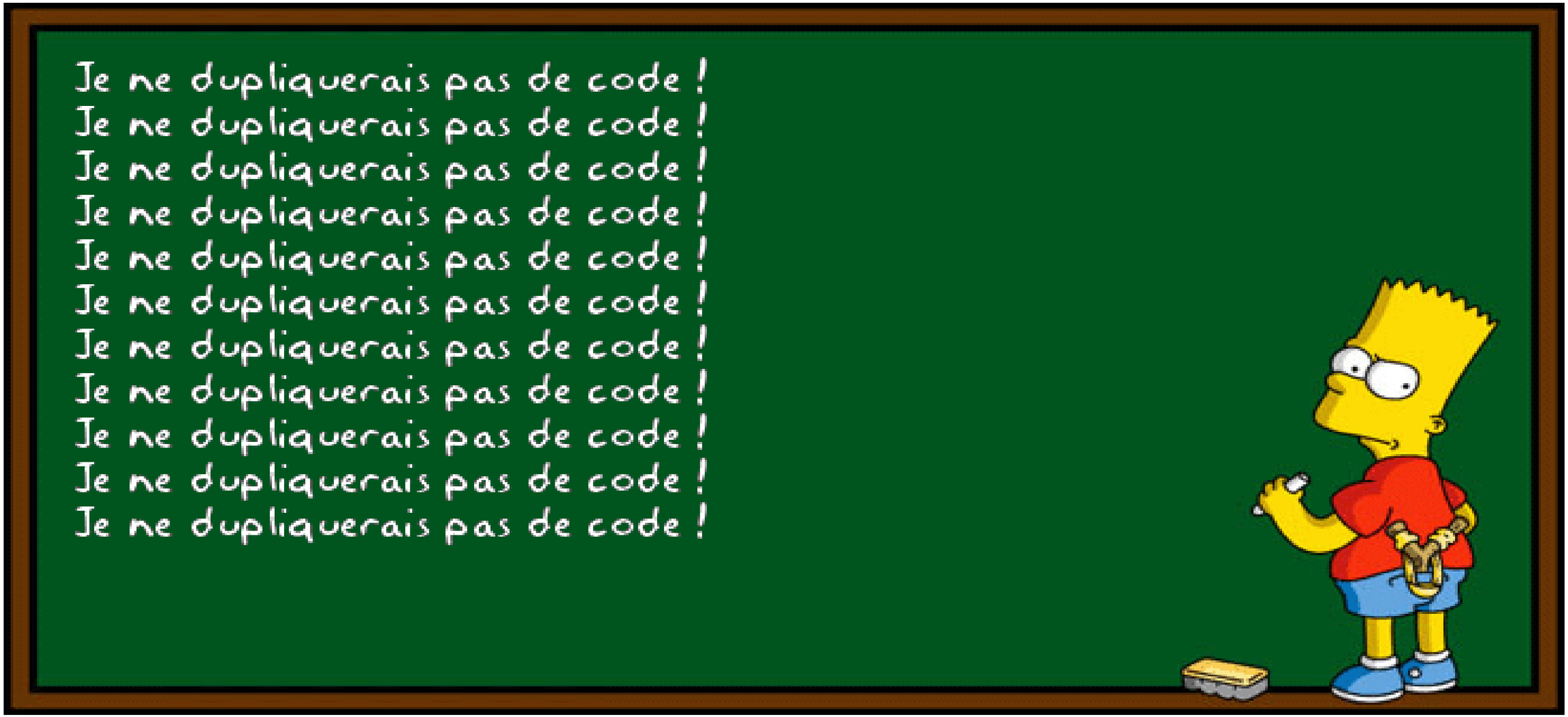
Plan de la présentation

- Bon principes de codage / conception
- Tests unitaires

Bonnes règles de codage 1/3

- DRY : Don't Repeat Yourself



Bonnes règles de codage 2/3

- KISS : Keep It Simple Stupid !
- Un programme simple est :
  - plus facile à comprendre / maintenir / faire évoluer
  - Un composant doit faire une seule chose
  - Bon exercice : lire un programme qu'on a écrit il y a 1 an
- YAGNI : You Ain't Gonna Need It
  - Ne développer que ce qui est strictement nécessaire
  - Éviter l'over-engineering

Bonnes règles de codage 3/3

- Bien nommer ses classes, méthodes, variables ... :
  - Code plus facile à comprendre/ maintenir / faire évoluer
  - « ce qui se conçoit bien s'énonce clairement »
  - Un problème de nommage vient souvent d'un problème de conception, d'un concept qui n'est pas clair
  - C'est un exercice difficile !

Tests unitaires

- Pourquoi faire des tests ?
- Tests unitaires avec JUnit
- Utiliser les assertions riches d'AssertJ
- Mock et stub avec Mockito

Pourquoi faire des tests ?

1. Prouver que son programme marche

2. Vérifier la non régression de son programme suite à des évolutions

3. Montrer comment utiliser le composant testé (doc)

Caractéristiques d'un bon test

- Auto vérifiant
- Facile à lancer
- Rapide pour avoir un feedback ... rapide !
- Lisible (penser au pattern : given / when / then)
- Indépendant des autres tests

Démo JUnit

Présentation d'AssertJ

- AssertJ offre des assertions pour les tests :
  - Riches, facile à utiliser (IDE friendly)
  - Proche du langage naturel
  - Documentées (javadoc et doc github)

- AssertJ modules :
  - assertj-core : assertions pour les types du JDK
  - assertj-guava : assertions pour Guava
  - assertj-joda-time : assertions pour Joda Time

Exemple

```
// assertions simple
assertThat(gandalf.getName()).isEqualTo("Gandalf");

// il y a 3 anneaux elfiques dans le Seigneur des Anneaux
List<Ring> elvesRings = newArrayList(vilya, nenya, narya);

// assertions sur les collections
assertThat(elvesRings).isNotEmpty()
                      .hasSize(3)
                      .contains(nenya, vilya, narya)
                      .doesNotContain(oneRing);
```

AssertJ : Démo

Mock et stub

- Il est difficile de tester un composant qui dépend d'autres composants (collaborateurs)
- Pour cela, il faut spécifier le comportement des composants collaborateurs.
- Stub : spécifier le comportement d'un composant collaborateur
- Mock : vérifier que le composant collaborateur a bien été sollicité comme prévu.

comment utiliser Mockito ?

Conclusions

- FAITES DES TESTS
- JUnit + Mockito + AssertJ = couvre une grande partie des outils pour des tests efficaces (en java)
- Librairies de tests complémentaires :
  - Tests R : RUnit (même les statisticiens peuvent tester!)
  - Tests Base de données : DBUnit / NoSqlUnit
  - Tests IHM : pas facile mais réalisables

Ressources

- JUNIT : https://github.com/junit-team/junit/wiki
- AssertJ : https://github.com/joel-costigliola/assertj-core#readme
- Mockito : http://code.google.com/p/mockito/