Génie Logiciel
Validation par le test

Lydie du Bousquet
UJF – Grenoble I

Objectifs du cours d'aujourd'hui

- Donner des réponses aux questions suivantes :
  - Qu'est-ce que « tester un programme » ?
  - Pourquoi tester ?
  - Quand tester ?
  - Comment tester ?
  - Quand s'arrêter de tester ?

Qu'est-ce que tester un programme ?

Exercice 1 :

Inscrivez sur une feuille ce que « tester un programme » signifie pour vous.

Qu'est-ce que tester un programme ?

- Tester un programme, c'est :
  - l'exécuter
  - en maîtrisant les données en entrée
  - en s'assurant que le comportement est celui attendu
- Exécution du programme sous test => exécutable
- Maîtrise des entrées
  - Sélectionner = *choisir*
  - assez de données, mais pas trop
- Vérification du comportement
  - pouvoir observer le comportement
  - adopter une vision critique

Pourquoi tester ?

- C'est une question clé
- Pourquoi est-ce que vous testez ?

Exercice 2 :

Inscrivez sur une feuille « pourquoi est-ce que vous testez (pourquoi faire) ?».

Pourquoi tester ?

- Parce que programmer est une activité où l'on commet des erreurs
- Parce que les autres techniques de validation (relecture de code, preuves formelles ou autres)
  - Pas toujours possibles
  - Pas toujours complètement convaincantes
  - Est-ce que vous accepteriez de monter dans un avion qui n'a jamais été testé en vol mais dont les fabricants vous garantissent qu'ils ont «!prouvé!» que tout allait bien se passer ?
- Réponse de Myers 1979

« Tester pour rechercher des erreurs »

Pourquoi tester ?

- On teste pour rechercher des erreurs
- Quels types d'erreur ?

Exercice 3 :

Inscrivez sur une feuille « quel type d'erreur peut-on chercher ?».

Quel type d'erreur recherche-t-on ?

- On attend du logiciel certaines propriétés
  - absence d'erreur (correction fonctionnelle),
  - robustesse,
  - performances,
  - utilisabilité, ...
- Caractérisation des tests selon l'objectif
  - Test de conformité
  - Test de robustesse
  - Test de performances, de montée en charge

Quel type d'erreur recherche-t-on ?

- A votre niveau, objectif principal :
  - correction fonctionnelle

S'assurer que le programme satisfait
ce que l'on attend de lui
= ses spécifications

Objectifs du cours d'aujourd'hui :

- donner des réponses aux questions suivantes :
  - Qu'est-ce que « tester un programme » ?
  - Pourquoi tester ?
  - Quand tester ?
  - Comment tester ?
  - Quand s'arrêter de tester ?

Quand tester un programme ?

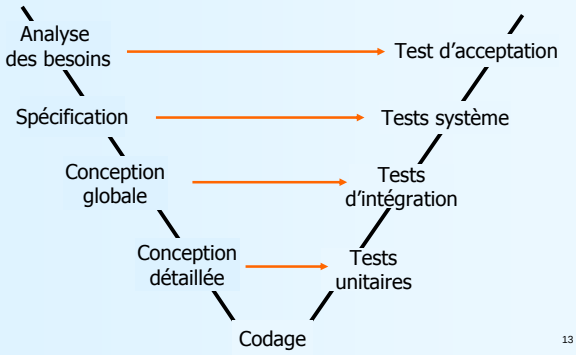
Exercice 4 :

Inscrivez sur une feuille votre réponse à la question « quand teste-t-on un programme ? »

Quand tester ?

- On peut tester à différents moments
  - Au cours du développement de chaque fonction
  - Quand on intègre les modules entre eux
  - Avant et pendant la livraison
  - Après les modifications (maintenance, évolution…)
- Caractérisation des tests par rapport à la « phase » de développement
  - Test unitaire
  - Test d'intégration
  - Test système
  - Test d'acceptation (ou de recette)
  - Test de non-régression

Quand tester ?



Quand tester ?

- A votre niveau, objectif principal :
  - Test unitaire
  - (Test d'intégration)
  - Test système

Objectifs du cours d'aujourd'hui :

- Donner des réponses aux questions suivantes :
  - Qu'est-ce que « tester un programme » ?
  - Pourquoi tester ?
  - Quand tester ?
  - Comment tester ?
    - Comment choisir les données ?
    - Comment exécuter les tests ?
    - Comment décider de la correction ?
  - Quand s'arrêter de tester ?

Comment choisir des données ?

Exercice 5 :

Inscrivez sur une feuille votre réponse à la question « comment choisir des données de test ? »

Comment choisir les données ?

- Tester = rechercher des erreurs
- Ici, on s'intéresse à la correction fonctionnelle

- Il existe deux grandes stratégies de test
  - En utilisant la spécification du programme
  - En utilisant le code

Comment choisir les données ?
Exemple : longueur d'un mot est-elle paire ?

On cherche à tester un programme dont on attend le comportement suivant (spécification informelle)

Le programme lit un mot à partir du clavier.
Si le mot a un nombre pair de lettres, le programme écrit
"L contient un nombre pair de lettres"
Sinon le programme écrit
"L contient un nombre impair de lettres"

Quels tests pour ce programme ?

Exercice 6 :

Proposez des tests pour cet exemple.

Comment choisir les données ?
Exemple : longueur d'un mot est-elle paire ?

- Produire les tests à partir de la spécification
  - « Couverture des fonctionnalités »
  - « Couverture des cas d'utilisation »
- Ici, la spécification décrit explicitement 2 cas généraux
  - Mot avec nombre de lettres pair
  - Mot avec nombre de lettres impair
- On peut imaginer des cas limites :
  - Mot vide
  - Mot très long
  - Mot avec un ou deux caractères

Comment choisir les données ?
Exemple : longueur d'un mot est-elle paire ?

- Mot avec nombre de lettres pair « impair » (p)
- Mot avec nombre de lettres impair « mot » (i)
- Mot vide « » (p)
- Mot comprenant un caractère « h » (i)
- Mot à deux caractères « kw » (p)
- Mot très long « anticonstitutionnellement » (i)

- Qu'est-ce qu'un mot ?
  - Majuscules ? Espaces ? Caractères spéciaux ? Chiffres ?

Comment choisir les données ?
Exemple : longueur d'un mot est-elle paire ?

- Mot avec des majuscules
  « Essai » « essAI » « ESSAI » (i)
  « AmsTramGramm » (p)
- Mot avec des espaces « test unitaire » (i)
- Mot avec des caractères spéciaux
  « test_d'intégration » (p)
  « β-tests » (i)
- Mot avec des chiffres
  « test44 » (p)

Tests de robustesse ?

Comment choisir les données ?
Exemple : longueur d'un mot est-elle paire ?

Mot : séquence de caractères + marque de fin
rangé dans un tableau de [0..N]

```
1.  M <- LireMot
2.  L = 0
3.  Tant que M[L] != MarqueDeFin faire
4.      L = L + 1
5.  Fin

6.  Si (L mod 2) = 0
7.  Alors Ecrire("L contient un nombre pair")
8.  Sinon Ecrire("L contient un nombre impair")
9.  Ecrire("de lettres")

10. Si (L==23)
11. Alors Ecrire(".")
12. Sinon Ecrire( "…")
13. Fin
```

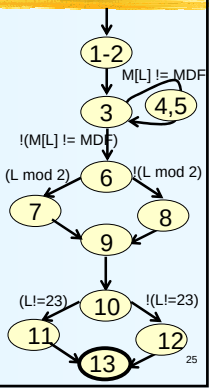

Comment choisir les données ?
Exemple : longueur d'un mot est-elle paire ?

- Produire les tests à partir du code
- Graphe de flot de contrôle
  - Représentation intermédiaire du code
  - Nœuds et transitions représentent les exécutions possibles
- On s'assure que l'on est passé « partout » selon certains critères
  - Toutes les instructions exécutées
  - Toutes les branches
  - Toutes les conditions
  - Tous les chemins

Comment choisir les données ?
Exemple : longueur d'un mot est-elle paire ?

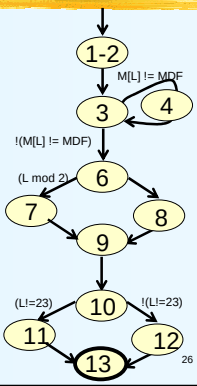
```
1.  M <- LireMot
2.  L = 0
3.  Tant que M[L] != MarqueDeFin faire
4.      L = L + 1
5.  Fin

6.  Si (L mod 2) = 0
7.  Alors Ecrire("L contient un nombre pair")
8.  Sinon Ecrire("L contient un nombre impair")
9.  Ecrire("de lettres")

10. Si (L!=23)
11. Alors Ecrire(".")
12. Sinon Ecrire( "…")
13. Fin
```

=> Graphe de flot de contrôle



Comment choisir les données ?
Exemple : longueur d'un mot est-elle paire ?

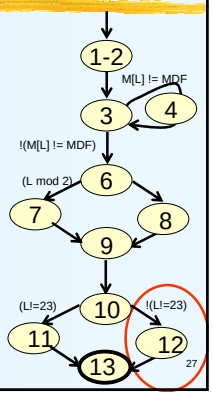
- Couvrir les instructions
  - Couvrir les états
- Couvrir les branches
  - Couvrir les transitions
- Couvrir les chemins
  - Certains non exécutables



Comment choisir les données ?
Couverture des instructions

- « impair » (p)
- « mot » (i)
- « » (p)
- « h » (i)
- « kw » (p)
- « anticonstitutionnellement » (i)

- Il faut un test avec un mot de 23 lettres



A propos des objectifs de couverture

- Couverture de tous les chemins exécutables est souvent infaisable
- Couverture des instructions est différente de la couverture des branches

Exercice 8 :

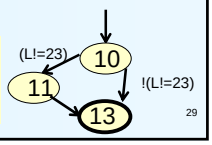
Proposez des exemples de programme simple pour illustrer ces 2 affirmations.

A propos des objectifs de couverture

- Couverture des chemins exécutables est souvent infaisable
  - Reprenons l'exemple précédent
  - La couverture des chemins exécutables = tester un mot au moins de chaque longueur entre 0 et l'infini
- Couverture des instructions est différente de la couverture des branches

```
10 Si (L!=23)
11 Alors Ecrire(".")
12 Sinon Ecrire( "…")
```

```
10 Si (L!=23)
11 Alors Ecrire(".")
```



A propos des objectifs de couverture

- Rappel : on teste pour découvrir des erreurs
- Tous les moyens sont bons pour générer les données
  - Spécification
  - Code
- Aucune méthode ne peut garantir l'absence d'erreur
- Stratégies complémentaires
- A propos de la couverture
  - Indicateur : **le test n'est pas fini** !
  - Ne garantit pas l'absence d'erreur

Objectifs du cours d'aujourd'hui :

- Donner des réponses aux questions suivantes :
  - Qu'est-ce que « tester un programme » ?
  - Pourquoi tester ?
  - Quand tester ?
  - Comment tester ?
    - Comment choisir les données
    - Comment exécuter les tests
    - Comment décider de la correction
  - Quand s'arrêter de tester ?

Exécuter des tests

Exercice 9 :

Comment faites-vous ou feriez-vous pour exécuter des tests ?

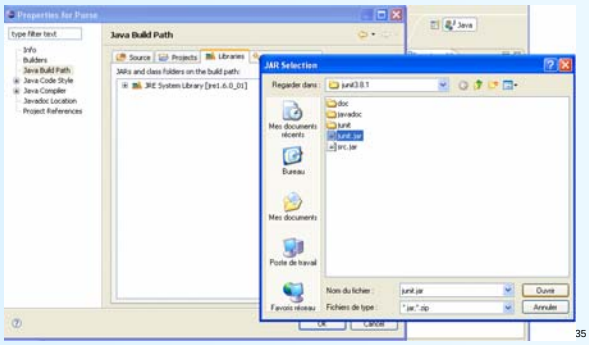
Comment exécuter les tests ?

- Exécuter le système sous test en
  - fournissant les données de tests « à la main »
  - observant les sorties
- Programmes de test / batch
- Utilisation d'outils spécialisés
  - J-unit, Cpp-unit
  - Watij (pour le test de site web)

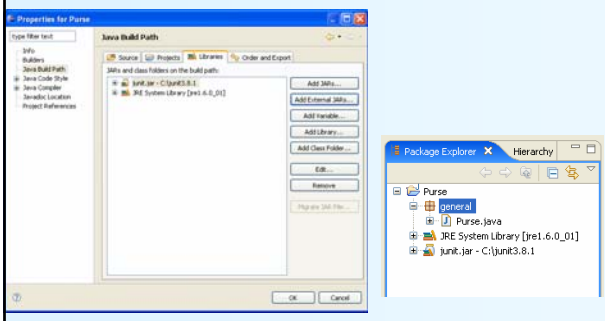
JUnit

- Framework open source pour :
  - les tests unitaires
  - des classes Java
  - Automatisation de l'exécution
- Existe comme un plug-in Eclipse
- Exemple : porte-monnaie
  - 3 opérations : *getBalance(), credit(), débit()*
  - Maximum Balance : 100 euros
  - Projet Java développé sous Eclipse, *Purse.java*

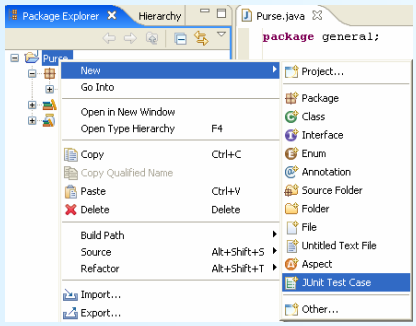
Utiliser JUnit
1. ajouter le JUnit.jar dans le classpath



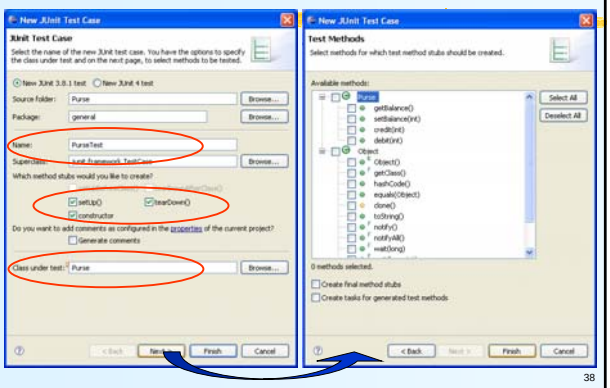
Utiliser JUnit
1. ajouter le JUnit.jar dans le classpath



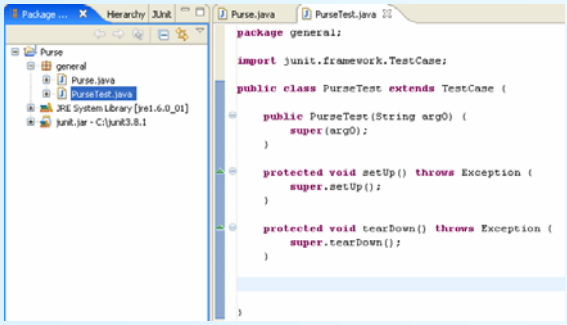
Utiliser JUnit
2. Créer un fichier de test JUnit



2. Créer un fichier de test JUnit



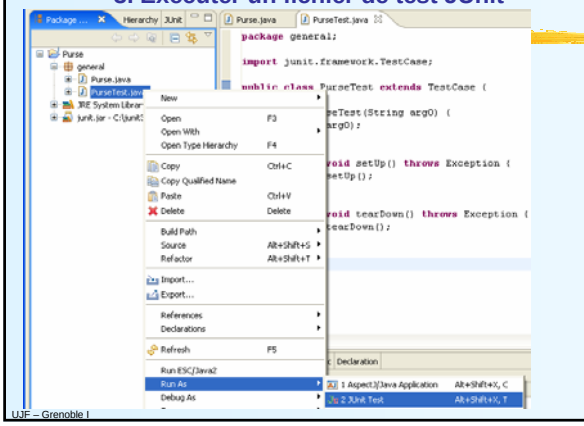
2. Créer un fichier de test JUnit



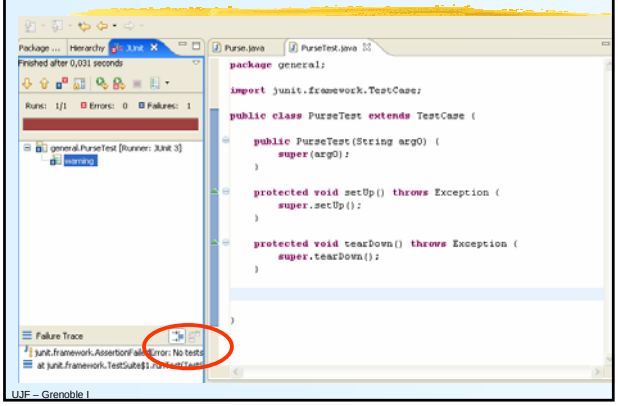
2. Créer un fichier de test JUnit

- Des méthodes de la classe de test
  - setUp()
    - Pour initialiser les éléments utiles pour tous les tests
    - Exemple : initialiser une instance de la classe sous test
  - tearDown()
    - Pour effectuer des traitements à la fin de tous les tests
    - Exemple : libérer l'instance de la classe sous test
- Il faut ajouter des méthodes = cas de tests
  - Nom des méthodes doit commencer par *test*

3. Exécuter un fichier de test JUnit

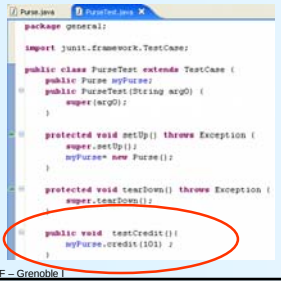


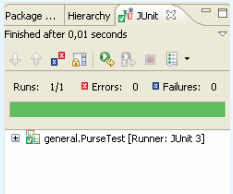
3. Exécuter un fichier de test JUnit



4. Compléter un fichier de test JUnit

- Pour le porte-monnaie, on a
  - Maximum balance = 100 euros
  - On essaie le cas où l'on crédite 101 euros





Pourquoi ce test passe-t-il ?

Objectifs du cours d'aujourd'hui :

- Donner des réponses aux questions suivantes :
  - Qu'est-ce que « tester un programme » ?
  - Pourquoi tester ?
  - Quand tester ?
  - Comment tester ?
    - Comment choisir les données ?
    - Comment exécuter les tests ?
    - Comment décider de la correction ?
  - Quand s'arrêter de tester ?

Comment décider de la correction ?

- Une exécution d'un programme est un test si on juge le résultat
- « problème de l'oracle »
- Pour testCredit(),
  - on exécute la procédure crédit
  - on ne juge pas le résultat
- Pour tout test automatisé,
  - Il faut automatiser le jugement de la correction

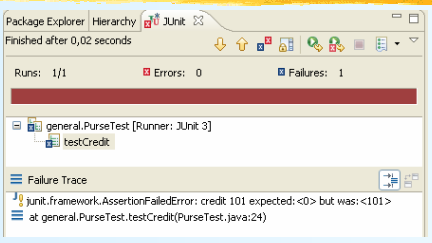
```
public void  testCredit(){
    myPurse.credit(101) ;
}
```


Comment décider de la correction ?
5. Oracle du test dans JUnit

- Différent type d'assertions :
  - assertTrue("message",condition) / (condition)
  - assertEquals("message", arg1,arg2) / (arg1,arg2)
  - assertNull, assertNotNull, assertSame, …
- Pour notre exemple
  - On s'attend à ce que credit(101) ne modifie pas la balance.

```
public void  testCredit(){
    int oldBalance;
    oldBalance=myPurse.getBalance();
    myPurse.credit(101) ;
    assertEquals("credit 101", oldBalance, myPurse.getBalance());
}
```


Comment décider de la correction ?
5. Oracle du test dans JUnit



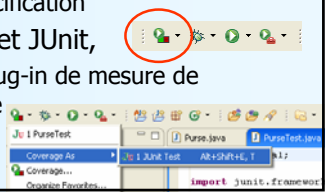

```
public int credit (int a_credit){
    Balance = Balance + a_credit;
    return (Balance);
}
```


Objectifs du cours d'aujourd'hui :

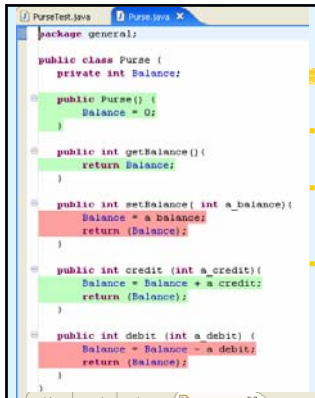
- Donner des réponses aux questions suivantes :
  - Qu'est-ce que « tester un programme » ?
  - Pourquoi tester ?
  - Quand tester ?
  - Comment tester ?
    - Comment choisir les données ?
    - Comment exécuter les tests ?
    - Comment décider de la correction ?
  - Quand s'arrêter de tester ?

Quand s'arrêter ?

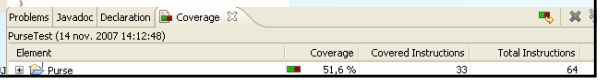
- On n'a jamais fini de tester
- On peut juste savoir quand on n'a pas assez testé
  - Couverture de code
  - Couverture de spécification
- Pour Eclipse, Java et JUnit,
  - EclEmma est un plug-in de mesure de couverture de code



Quand s'arrêter ?



- Ici : critère =
  couverture des instructions
- Un critère d'arrêt signifie
  - si le critère n'est pas couvert, on n'a **pas fini** de tester
- Un critère d'arrêt NE signifie PAS
  - Quand on a atteint le critère on a fini de tester
  - Quand on a atteint le critère on a trouvé toutes les erreurs



Résumé / Conclusions

En résumé

- Tester : rechercher des erreurs
- Difficultés et grandes stratégies
  - Choix des données
  - Arrêt du test
  - Problème de l'oracle

Tester, c'est difficile (1)

- Difficultés théoriques
  - On ne peut pas tout tester (test exhaustif)
    - Infinité de données et/ou pas le temps/argent
  - Quelles données choisir ?
  - Comment décider que le résultat est correct ?
  - Quand peut-on / doit-on s'arrêter ?
- Difficulté psychologique
  - Programmation : activité constructive
  - Test : activité « destructive »
  - Il est difficile de détruire ce que l'on a construit

Tester, c'est difficile : (2)
Quelles données choisir ?

- Test exhaustif impossible
- Choisir des données pour découvrir des erreurs
  - Quelles données pour trouver *des* erreurs ?
  - Quelles données pour trouver *toutes* les erreurs ?
  - Chaque programme est particulier = aucune technique générale
  - Démarches classiques, basées sur l'expérience
    - Étude de la spécification
    - Etude du code
    - Couvrir l'ensemble des cas (spécification, parties de code)
    - Tests aux limites (domaine des entrées, cas particuliers)

Tester, c'est difficile : (3)
Le résultat est-il correct ?

- Décider si le comportement du programme est correct
- Problème de l'oracle
- Pouvoir observer
  - quelles sont les sorties du logiciel
- Savoir décider
  - Comment décide-t-on que le comportement est OK ?
  - Exple : Soit un programme de tri alphabétique

Lagaf G. La Poste	La Poste Lagaf G.

Tester, c'est difficile : (4)
Quand s'arrêter ?

- Le test n'a pas révélé d'anomalies
  - Absence d'erreur ?
  - Mauvais choix de données ?
- Le test a révélé des anomalies
  - Existe-t-il d'autres erreurs ?

Références

- Plug-in
  - http://www.junit.org/
  - http://www.eclemma.org/
- Utiliser JUnit
  - http://junit.sourceforge.net/
  - http://jmdoudoux.developpez.com/java/eclipse/?page=Chap_010
  - http://open.ncsu.edu/se/tutorials/junit/
  - http://www.cs.umanitoba.ca/~eclipse/10-JUnit.pdf