Conception objet en Java avec BlueJ
une approche interactive

6. Des objets bien conçus

David J. Barnes, Michael Kölling
version française: Patrice Moreaux

Principaux concepts étudiés

- Test
- Mise au point
- Automatisation des tests
- Écrire pour faciliter la maintenance

Nous devons gérer les erreurs (1)

- Les premières erreurs sont en général des *erreurs de syntaxe*
  - le compilateur les détecte.
- Les erreurs plus tardives sont généralement des *erreurs logiques*
  - le compilateur ne peut pas nous aider.
  - appelées "bogues" ("bugs").

Nous devons gérer les erreurs (2)

- Certaines erreurs logiques ne sont pas immédiatement détectables.
  - Les logiciels commerciaux sont rarement sans erreur.

Prévention versus détection (Développeur versus mainteneur)

- Nous pouvons diminuer le risque d'erreurs
  - utiliser des techniques de génie logiciel ("software engineering"), comme l'encapsulation.
- Nous pouvons accroître les chances de détection des erreurs
  - utiliser les pratiques de génie logiciel, comme la modularisation et la documentation.
- Nous pouvons développer nos capacités de détection.

Test et mise au point

- Ce sont des compétences cruciales.
- Les tests recherchent les erreurs.
- La mise au point recherche les causes des erreurs.
  - une erreur peut tout à fait survenir “loin" de sa source.

Techniques de test et de mise au point

- Test unitaire (sous BlueJ)
- Automatisation des tests
- Parcours (exécution) à la main
- Instructions d'affichage (`print`)
- Metteurs au point (débogueurs)

Test unitaire

- Chaque unité d'une application peut être testée
  - méthode, classe, module (paquetage en Java).
- Peut (devrait) être fait pendant le développement
  - trouver et corriger tôt réduit les coûts de développement (ex.: temps du programmeur)
  - constituer ainsi une suite de tests

Test - principes

- Comprendre ce que doit faire l'unité – son *contrat*
  - chercher les cas de violation.
  - utiliser des tests positifs et des tests négatifs
- Tester aux *limites (bornes)*
  - zéro, un, tous
    - rechercher dans une collection vide
    - ajoutée à une collection pleine

Test unitaire sous BlueJ

- Création d'objets de chaque classe
- Appel de chaque méthode possible
- Les inspecteurs fournissent une vue toujours actualisée de l'état d'un objet
- Entraînez-vous avec le projet *diary-prototype*

Automatisation des tests (1)

- De bons tests demandent de l'imagination mais …
- ... les tests approfondis prennent du temps et sont répétitifs
- *Les tests de non régression* impliquent la ré-exécution de tests
- L'utilisation d'un cadre de tests peut alléger ce travail
  - classes spécifiques pour les tests
  - l'imagination se concentre sur la création de ces classes

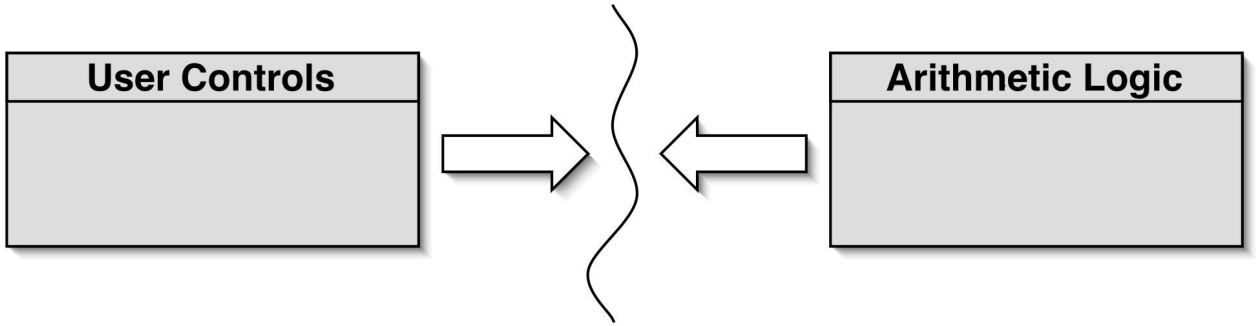
Automatisation des tests (2)

- Travailler avec le projet *diary-testing*
  - une analyse manuelle des résultats est nécessaire
- Examiner l'automatisation plus poussée du projet *diary-test-automation*
  - une intervention est nécessaire seulement si un erreur est signalée

Modularisation et interfaces

- Les applications sont souvent constituées de différents modules.
  - par exemple pour que plusieurs équipes travaillent en même temps
- L'*interface* entre modules doit être clairement spécifiée
  - permet les développements indépendants en parallèle
  - augmente les chances d'une intégration réussie

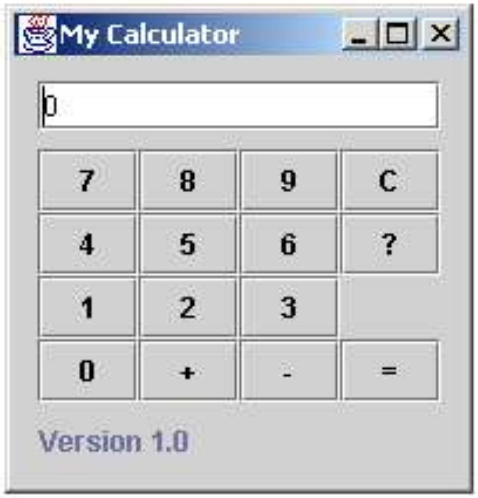
Modularisation dans un calculateur



- Un module n'a pas besoin de connaître les détails d'implantation d'un autre
  - les contrôles utilisateur peuvent être une IUG ou un dispositif matériel
  - La logique peut être logicielle ou matérielle

Signatures de méthode comme interface

```
// Renvoie la valeur à afficher.
public int getDisplayValue();

// A appeler si un chiffre est pressé.
public void numberPressed(int number);

// Si un opérateur est pressé.
public void plus();

// Si le moins est pressé.
public void minus();

// Pour terminer un calcul.
public void equals();

// Pour réinitialiser le calculateur.
public void clear();
```



Mise au point

- Il est important de développer des compétences en lecture de code
  - la mise au point porte souvent sur du code “externe”
- Techniques et outils existent pour aider au processus de mise au point
- Expérimentez avec le projet *calculator-engine*

Exécutions manuelles

- Relativement peu utilisées.
  - une approche de faible technicité
  - plus puissante qu'appréciée
- Laissez l'ordinateur de côté !
- Exécutez le programme à la main
- Vues haut niveau (pas logique, "step") ou bas niveau (pas instruction, "step into")

Notez les états d'un objet

- Le comportement d'un objet est généralement déterminé par son état
- Un comportement incorrect est souvent dû à un état incorrect
- Enregistrez les valeurs de tous les champs dans un tableau
- Documentez les changements d'état après chaque appel de méthode

Exécutions avec explicitations orales

- Expliquez à quelqu'un ce que fait le code
  - il peut détecter l'erreur
  - le fait d'expliquer peut vous aider à la trouver vous-même
- Il existe des techniques de groupe pour réaliser des exécutions formalisées (inspections)

Instructions d'affichage

- La méthode la plus utilisée
- Ne nécessite aucun outil spécial
- Disponible dans tous les langages de programmation
- Utile seulement si les méthodes pertinentes sont documentées
- La quantité de résultats peut être très grande!
- Activer/désactiver à bon escient demande de la perspicacité

Débogueur

- Les débogueurs sont spécifiques d'un langage et d'un environnement
  - BlueJ comporte un débogueur intégré
- Points d'arrêt
- Exécution pas à pas et par appel de méthode
- Séquence d'appels (pile)
- État des objets

Résumé

- Les programmes comportent des erreurs
- De bonnes techniques de génie logiciel peuvent en réduire le nombre
- Les compétences en test et mise au point sont essentielles
- Faites du test une habitude
- Automatisez les tests si possible
- Acquérez un ensemble de compétences en mise au point

Sommaire général