

# Pharo -- L'IDE

- SystemBrowser (Édition de code)
- Iceberg (gestion de version GIT)
- TestRunner (Tests unitaires)
- Debugger

# Mise en application

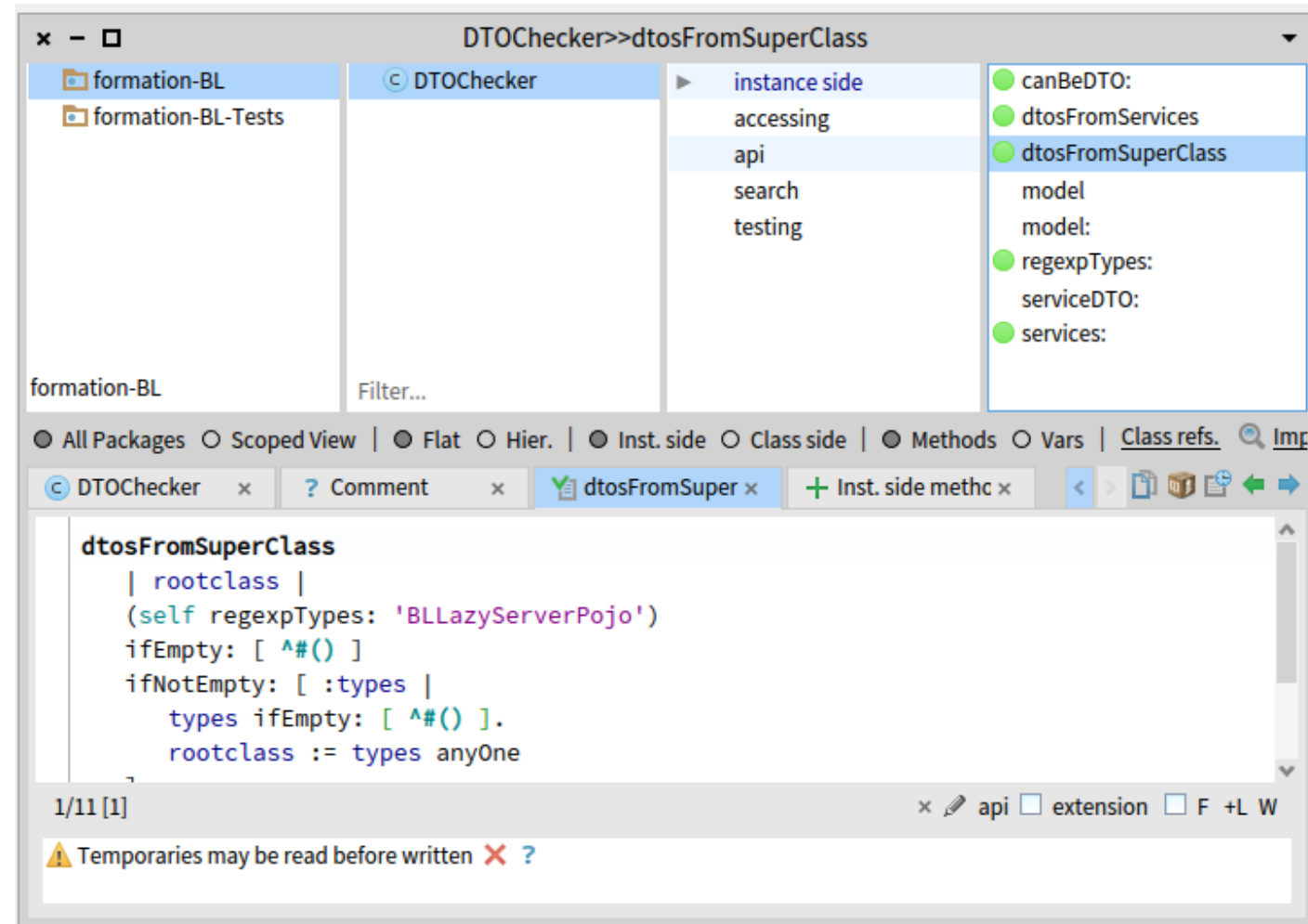
Exercice pratique mêlant les outils (édition de code, gestion de version, tests unitaires, *debugger*(?) ).

Créer une classe pour automatiser la recherche de DTOs :

- à partir de leur super-classe ( `BLLazyServerPojo` )  
v. exercices à la fin de [Pharo -- Environnement d'exécution](#)
- a partir des types des paramètres des services

# SystemBrowser

- Édition du code
- menu **Browse**



# Commandes de base

- `ctrl-m` : *implementors*, toutes les implémentations d'une méthode
- `ctrl-n` : *senders*, toutes les utilisations d'une méthode (ou une classe)
- `ctrl-b` : *SystemBrowser* sur une classe

## Exercice simple

Écrire la méthode *summiel* pour les entiers ( `Integer` ) qui calcul un "factoriel" basé sur l'addition au lieu de la multiplication

Note: Pas de triche, ne pas regarder la méthode `factorial`

# Iceberg

- Interface utilisateur pour GIT (GitHub et GitLab)
- Interface aux actions courantes (*push, pull, fetch, commit*)
- Gestion des branches (création, *checkout, merge*)
- Gestion des *remotes*
- etc.

# Iceberg

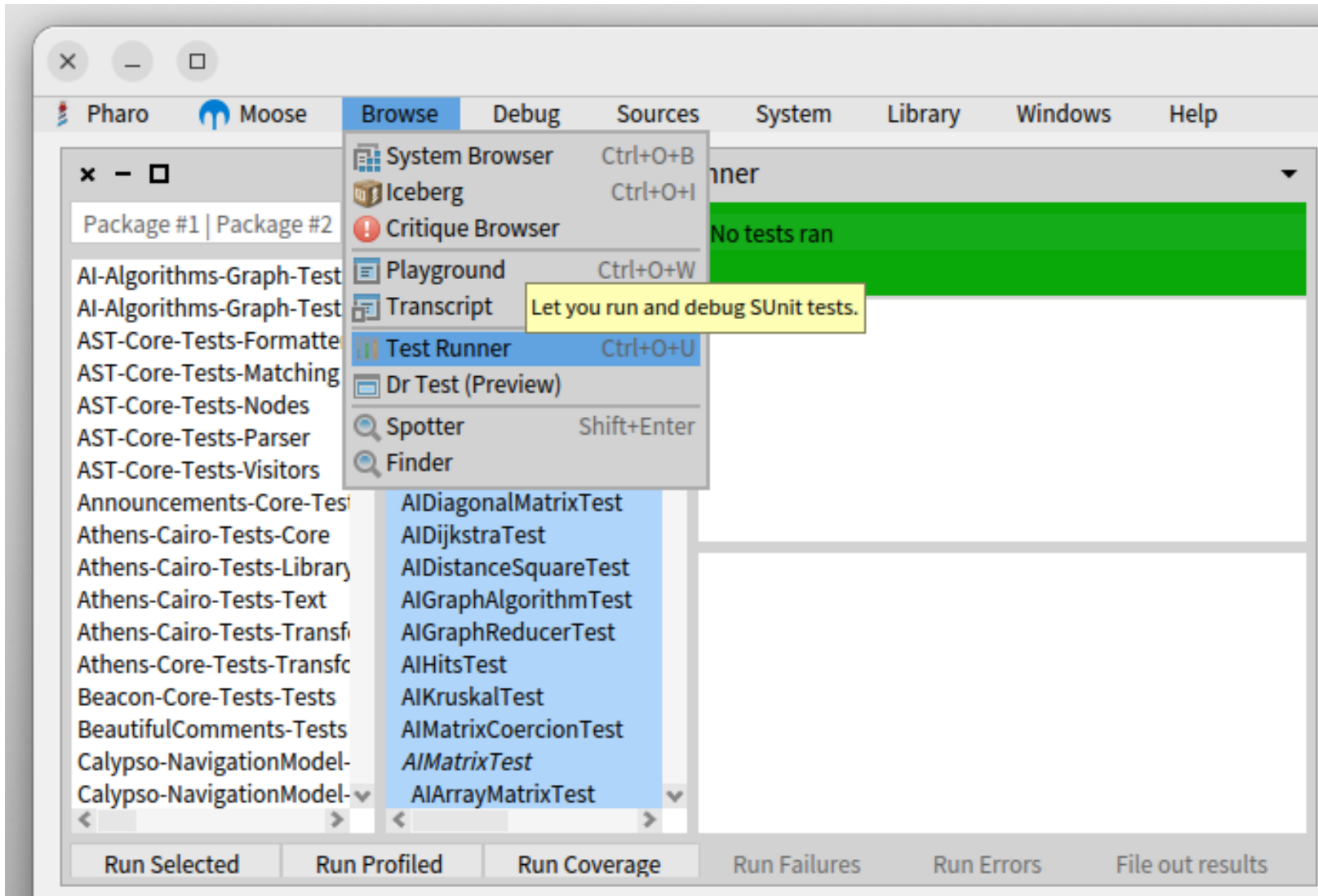
- Documentation dans le MOOC Pharo (<https://mooc.pharo.org/>)
- MOOC, semaine 1, Redo 4  
<http://rmod-pharo-mooc.lille.inria.fr/MOOC/PharoMOOC-Videos/FR/Week1/W1-Redo4-FR-final.mp4>
- MOOC, semaine 4, Live A  
<http://rmod-pharo-mooc.lille.inria.fr/MOOC/PharoMOOC-Videos/FR/Week4/W4-LiveA-FR-final.mp4>
- MOOC, semaine 4, Live B  
<http://rmod-pharo-mooc.lille.inria.fr/MOOC/PharoMOOC-Videos/FR/Week4/W4-LiveB-FR-final.mp4>

# ***Test Runner***

- 3 outils
  - TestRunner (outil historique)
  - Dr Test (outil plus récent)
  - Dans le *SystemBrowser* (convention de nomage)



# Test Runner

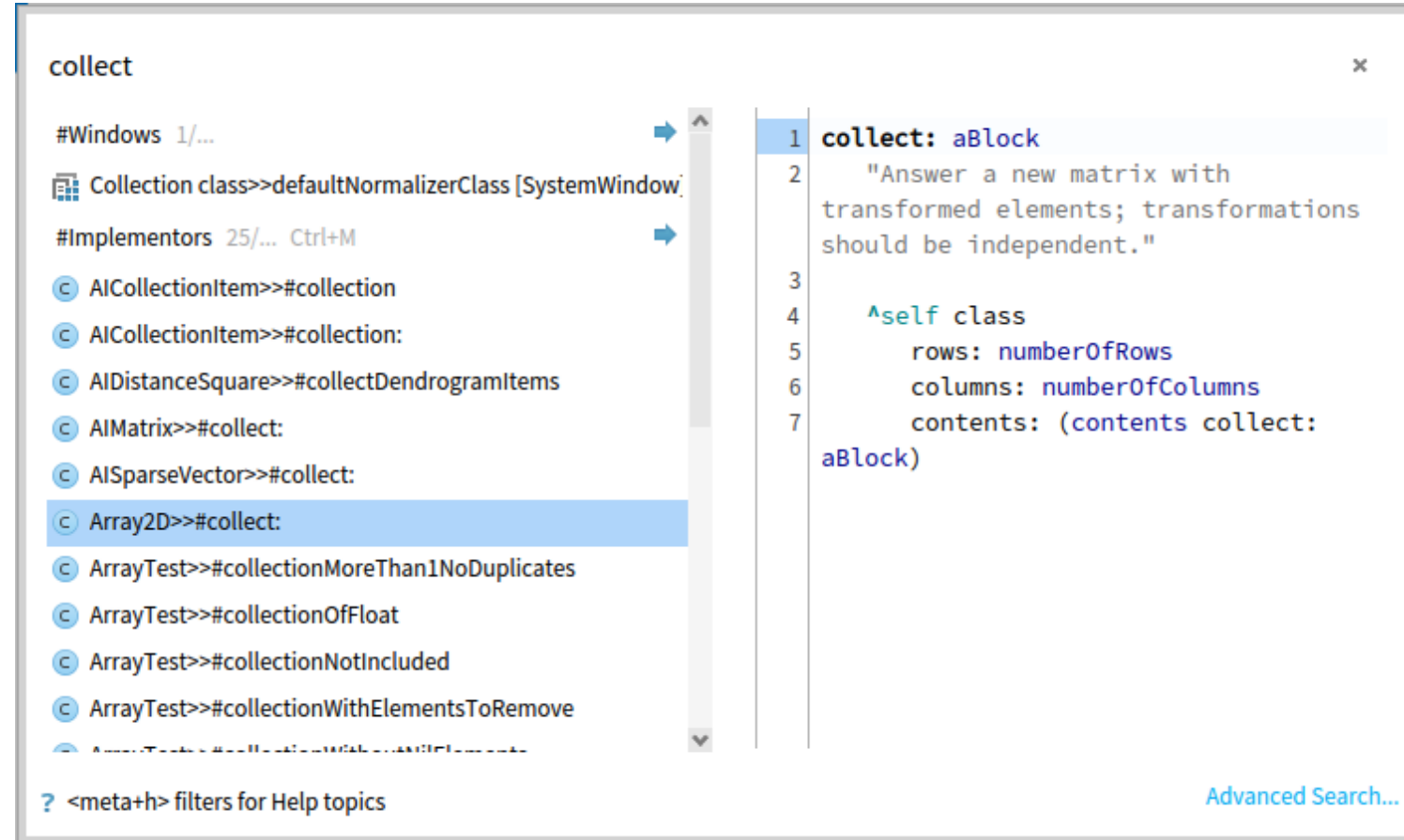


# Autres outils

- Spotter
- Finder
- Epicea (*Code changes*)
- Profiler

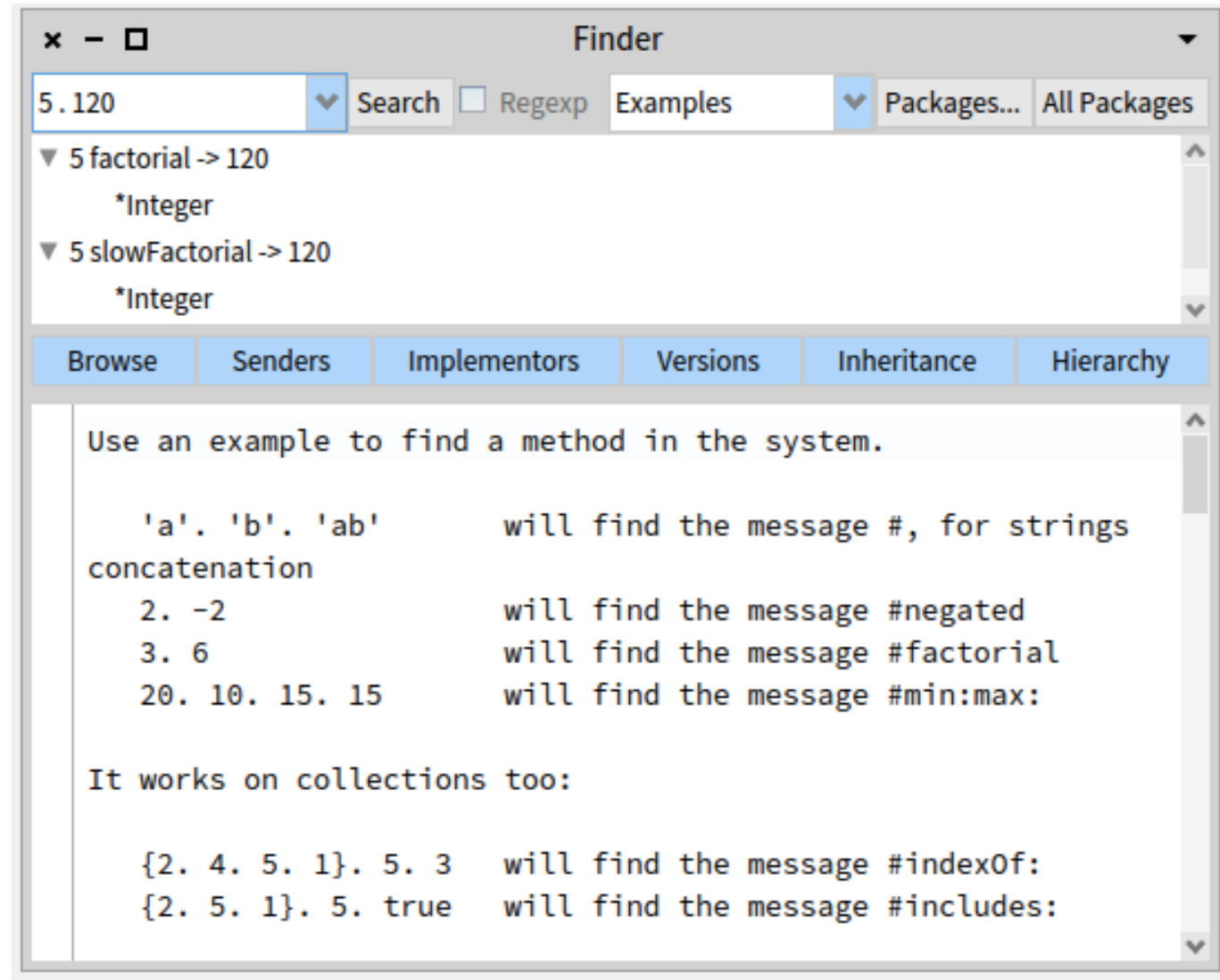
# Spotter

- Recherche textuelle
  - classes
  - méthodes  
(*implementors*)
  - *help*
  - Fenêtres
- `<shift-enter>`



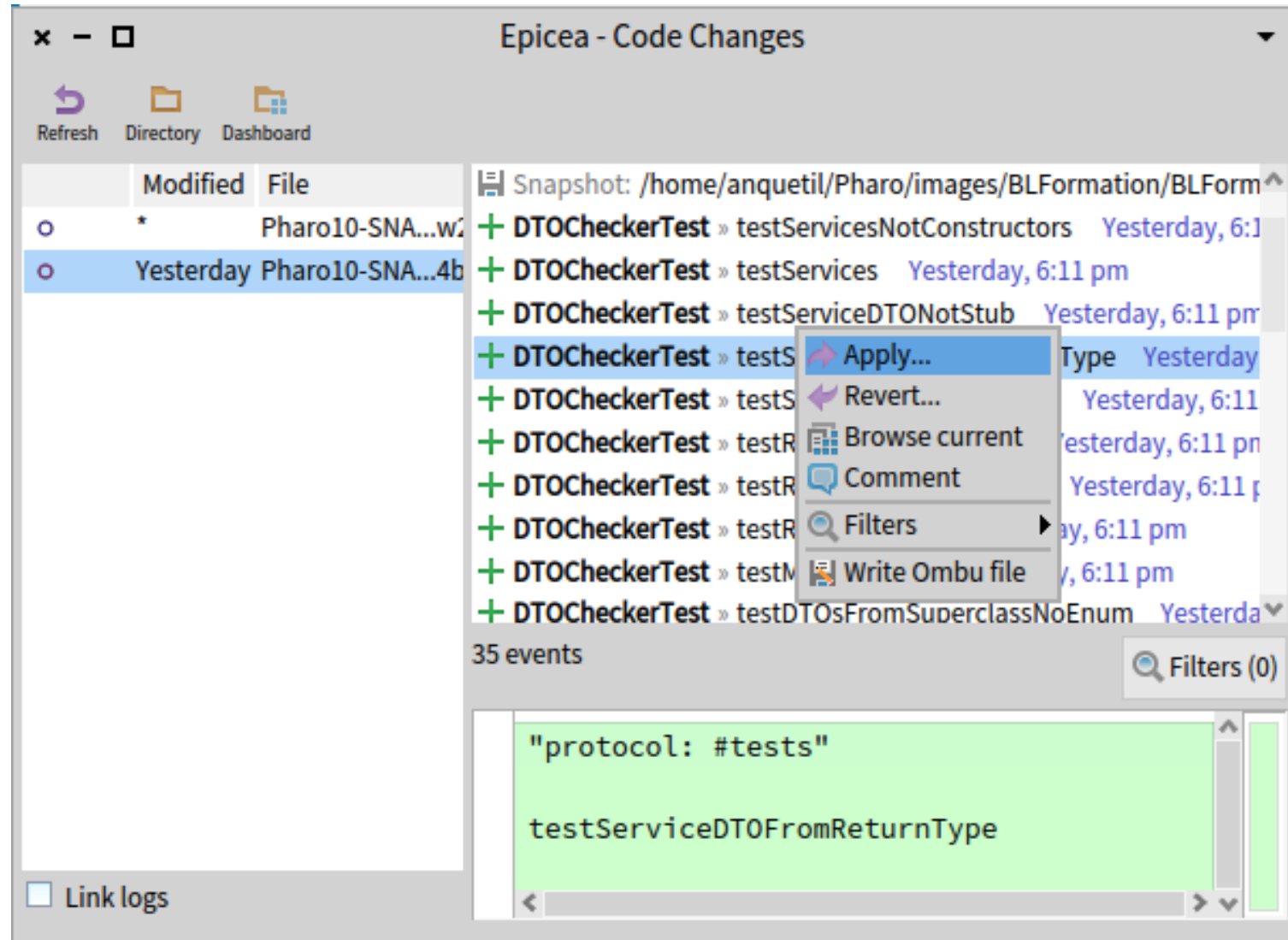
# Finder

- Recherches
  - à partir d'exemples
  - contenu de méthodes
  - *implementors*
- menu **Browse**



# Epicea

- Local code history
- menu Sources > Code Changes



# Profiler

- menu `Debug`
- `[ ... ] timeProfile`

The screenshot shows the 'Time profiler' window in the Pharo IDE. At the top, there is a 'Profile it' button, a 'Min time %' field set to 3.0, a 'Threshold' field set to 0, a 'Full tree' dropdown menu, and a checked 'All processes' checkbox. Below this, a code snippet is displayed: 

```
[  
    (MooseModel root first entityNamed:  
        'fr::sedit::omaje::talk::model::blmarket') queryAllIncoming  
    opposites atScope: FamixJavaPackage ]
```

. The main area is divided into two tabs: 'Tallies Tree' (selected) and 'Full report'. The 'Tallies Tree' tab shows a hierarchical list of operations with their respective percentages and durations. The top entry is '50.0% {2ms} FullBlockClosure(BlockClosure)>>newProcess'. Below it is '50.0% {2ms} MooseIncomingQueryResult>>opposites', which is expanded to show '50.0% {2ms} OrderedCollection(Collection)>>flatCollectAsSet:', which is further expanded to show '50.0% {2ms} OrderedCollection(Collection)>>flatCollect:as:', which is expanded to show '50.0% {2ms} OrderedCollection>>do:', which is expanded to show '50.0% {2ms} OrderedCollection(Collection)>>flatCollect:as:', which is expanded to show '50.0% {2ms} MooseIncomingQueryResult(TDependencyQueryResult)>>opposite', which is expanded to show '50.0% {2ms} MooseIncomingQueryResult>>opposite:'. The 'Full report' tab is also visible. At the bottom, there are three tabs: 'Code' (selected), 'Statistics', and 'Result'. The 'Code' tab shows the code snippet: 

```
opposites  
    ^ self newObjectResultWith: (self storage flatCollectAsSet: [ :d | (s
```

.

# Exercice -- Commencer en Pharo

Créer une classe pour automatiser la recherche de DTOs

- Dans Pharo (SystemBrowser), créer un package `formation-BL`
- créer une classe `DTOChecker`
  - lui donner une variable d'instance `model`
  - créer les accesseurs

# Exercice -- Commencer en GIT

- créer un projet dans Iceberg
  - ajouter le package `formation-BL` dans ce projet
  - faire un commit
- créer un projet dans GitHub (ou GitLab)
- ajouter l'URL au projet Iceberg
- faire un push
  - attention au nom de la branche par défaut ( `master` vs. `main` )



# Exercices -- Commencer le TDD

- créer la classe de test `DTOCheckerTest`
  - Cliquez droit sur `DTOChecker` puis *"Jump to test class"*
  - Cliquez droit sur `DTOChecker` puis *"New test class"*
  - Dans le SystemBrowser comme une classe normale
- Suggestion :

```
setUp
  super setUp.

model := FamixJavaModel new.
checker := DTOChecker new model: model ; yourself
```

# Exercice

## Exercices à faire en TDD

- créer une méthode `#regexTypes: <regexp>` pour rechercher les types dont le nom matche la regexp
  - faire un commit
- créer une méthode `#dtosFromSuperclass` pour trouver les classes qui héritent de `BLLazyServerPojo` et dont le nom ne contient pas 'Enum'
  - faire un commit

# Exercice

- créer une méthode `#serviceDTO:` qui reçoit une `FamixJavaMethod` et retourne les types de ses paramètres et son type de retour qui ne sont pas des types Java
  - Note: la création d'une méthode utilitaire `#canBeDTOS:` peut être une bonne idée
  - faire un commit
- créer une méthode `#services:` qui recherche les méthodes d'une classe pouvant être des services (publique, pas constructeur, pas `<Initializer>`)
  - faire un commit

# Exercice

- créer une méthode `#dtosFromServices` pour trouver les DTOs en paramètre des services des classes dont le nom commence par 'Uc'
  - faire un commit

Étape supplémentaire comparer les résultats de `#dtosFromSuperclass` et `#dtosFromServices` sur le modèle *Omaje*

# Cheat Sheet -- Pharo

- Regex matching: `<string> asRegex matches: <string>`
- Les classes de test héritent de `TestCase`
- Les méthodes de test ont un nom: `test...`
- Les méthodes de setup s'appellent: `setUp`
- Principales assertions : `#assert:equals:` , `#assert:` , `#deny:`
- Méthodes sur les collections : `#flatCollect:` , `#flatCollectAsSet:`

# Cheat Sheet -- Moose/Famix

- Création d'un modèle Java : `FamixJavaModel new`
- Création d'une entité dans un modèle : `FamixJavaClass new mooseModel: <model>`
- Classe d'une méthode Java : `#parentType` , `#parentType:` ,
- Paramètres d'une méthode: `#parameters` (opposé: `#parentBehaviouralEntity` )
- Type déclaré d'une méthode/d'un paramètre: `#declaredType`
- Les entités qui sont utilisées mais pas définies dans un projet (ex. `java.lang.System` ) ont une propriété `#isStub` vraie