



C3P :

Présentation week 5

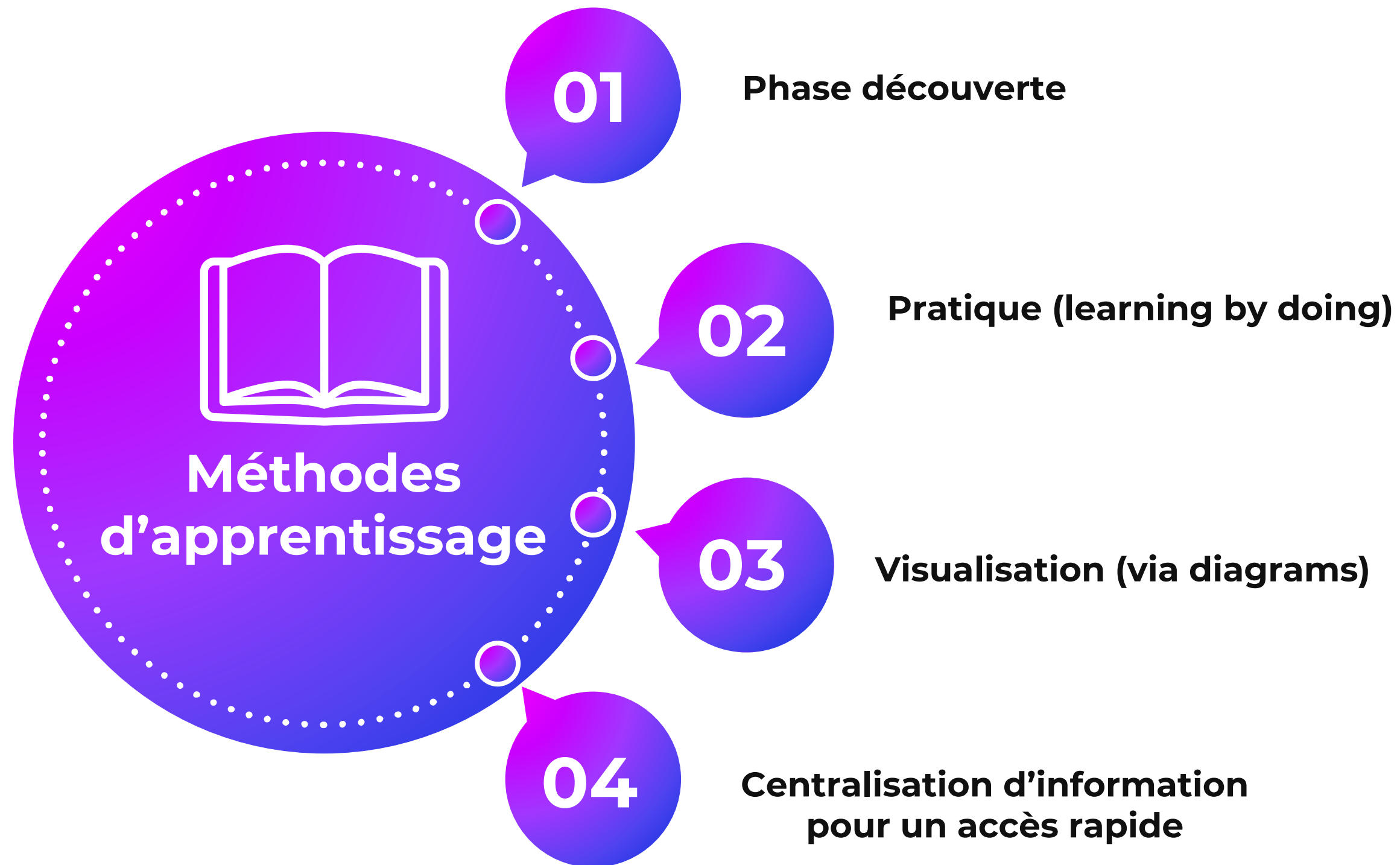
Harrar M'hamed



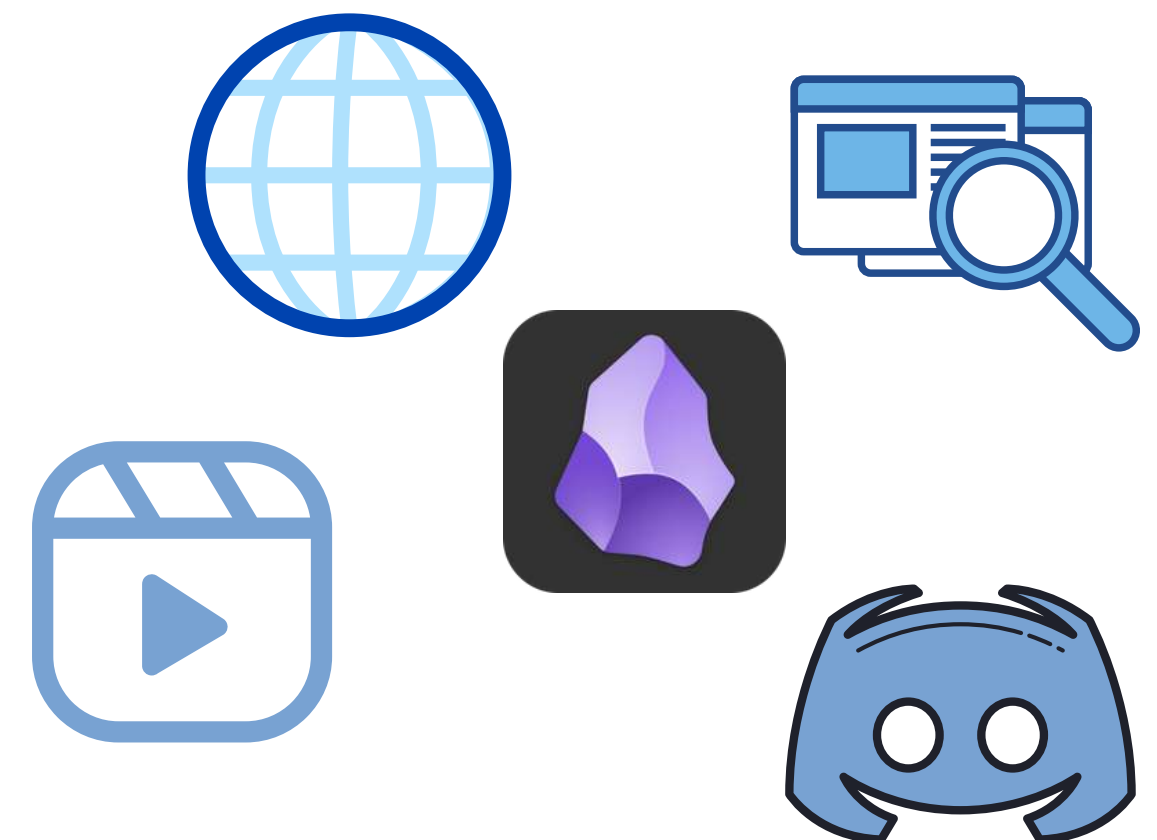
Plan

- **MÉTHODES D'APPRENTISSAGE ET RESSOURCES**
- **ANALYSE AVL**
- **ANALYSE ARTEFACT**
- **CONCLUSION**





Ressources



Projet AVL



Overview

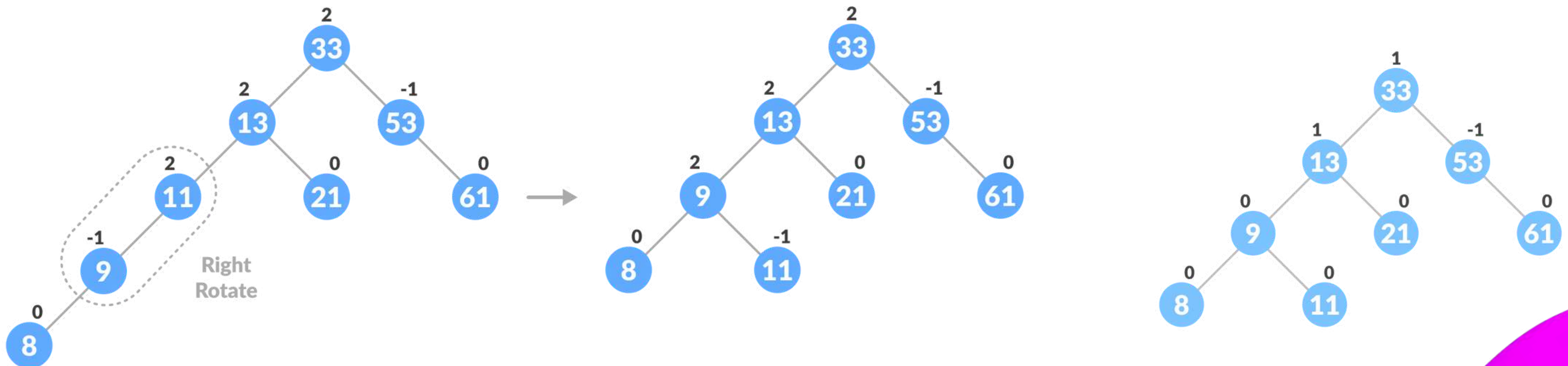


- Installation:
 - Instructions fournies.
- Guides d'Utilisation:
 - Absents.
- Documentation non explicite
- Objectif et Justification du Projet:
 - Absence d'une proposition de valeur claire.

Arbre binaire AVL

- Arbre binaire de recherche équilibré.
- Ajustements après des insertions et des suppressions.
 - Maintient d'une hauteur équilibrée.
- Garantit des complexités de temps spécifiques pour les opérations
 - **$O(\log n)$** pour la recherche, l'insertion et la suppression

Facteur d'équilibre dans $S = \{-1, 0, 1\}$



Dans Pharo



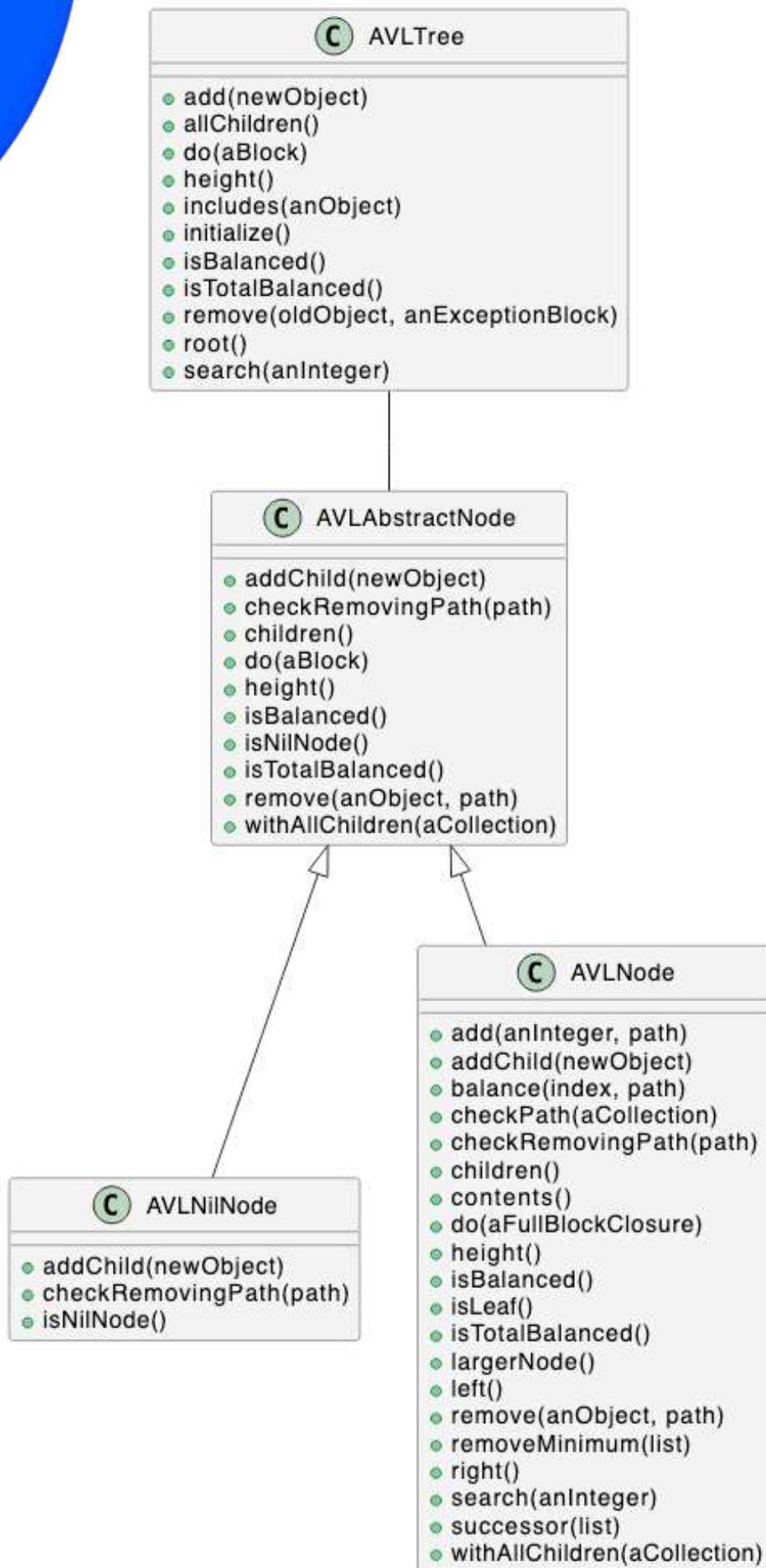
Do it Publish Bindings Versions Pages

```
1 | avlTree visualizer canvas numbers |
2 avlTree := AVLTree new.
3 numbers := (1 to: 22) asArray shuffled.
4 numbers do: [:each | avlTree add: each].
5
6 visualizer := AVLTreeVisualizer new.
7 visualizer tree: avlTree.
8
9 canvas := RSCanvas new.
10 visualizer renderIn: canvas.
11 canvas open.
```

Morph Layout Raw Breakpoints Meta

Variable	Value
self	a SystemWindow(932166656) named: Roassal
bounds	(20.0@38.0) corner: (520.0@538.0)
owner	a WorldMorph(687437312) [world]
submorphs	an Array [10 items] (a WindowEdgeGripMorph(1022502400) a WindowEdgeGripMorph(1034223104) a Windo[...])
fullBounds	(20@38) corner: (520@538)
color	(Color r: 0.823069403714565 g: 0.823069403714565 b: 0.823069403714565 alpha: 1.0)
extension	a MorphExtension (458826752) [other: (kmDispatcher -> a KMDispatcher) (announcer -> an Announce[...])]
borderWidth	1
borderColor	Color lightGray
model	nil
labelString	Roassal
stripes	an Array [2 items] (a Morph(601506304) a Morph(237715968))
label	a LabelMorph(976453632)'Roassal'
closeBox	a MultistateButtonMorph(1055655936)
collapseBox	a MultistateButtonMorph(186692608)
paneMorphs	an Array [1 item] (a RSAThensMorph(272817664))
collapsedFrame	nil
fullFrame	(20.0@38.0) corner: (520.0@538.0)
isCollapsed	false
menuBox	a MultistateButtonMorph(369427968)
mustNotClose	false
labelWidgetAllowance	133
updatablePanels	an Array [0 items] ()
labelArea	an AlignmentMorph(973696000)
expandBox	a MultistateButtonMorph(604396544)
embeddable	nil
isResizable	true
taskbarTask	a TaskbarTask

Structure



1. AVLAbstractNode

- Définit les opérations communes
- Exemples : addChild, checkRemovingPath, children, etc.

2. AVLNilNode

- Représente les nœuds feuilles
- Surcharge des méthodes : addChild, isNilNode, etc.

3. AVLNode

- Représente les nœuds composites
- Intègre des méthodes pour gérer les enfants et équilibrer l'arbre

4. AVLTree

- Classe cliente
- Gère la racine de l'arbre
- Fournit des méthodes pour ajouter, retirer et rechercher des nœuds

Utilisation du DP Composite :

- Traitement des nœuds de manière polymorphique.
- Élimination de la nécessité de vérifier le type de nœud lors de la traversée de l'arbre.

Organisation du code

- Le code suit le principe de responsabilité unique, chaque classe a un objectif spécifique.

Invariants

- Les classes AVLNode et AVLTree sont des points d'attention pour comprendre comment ces invariants sont gérés :
 - **Invariant Principal:**
 - Équilibre de l'arbre maintenu à chaque opération.
 - via son facteur d'équilibre

■ Vérification de l'Équilibre:

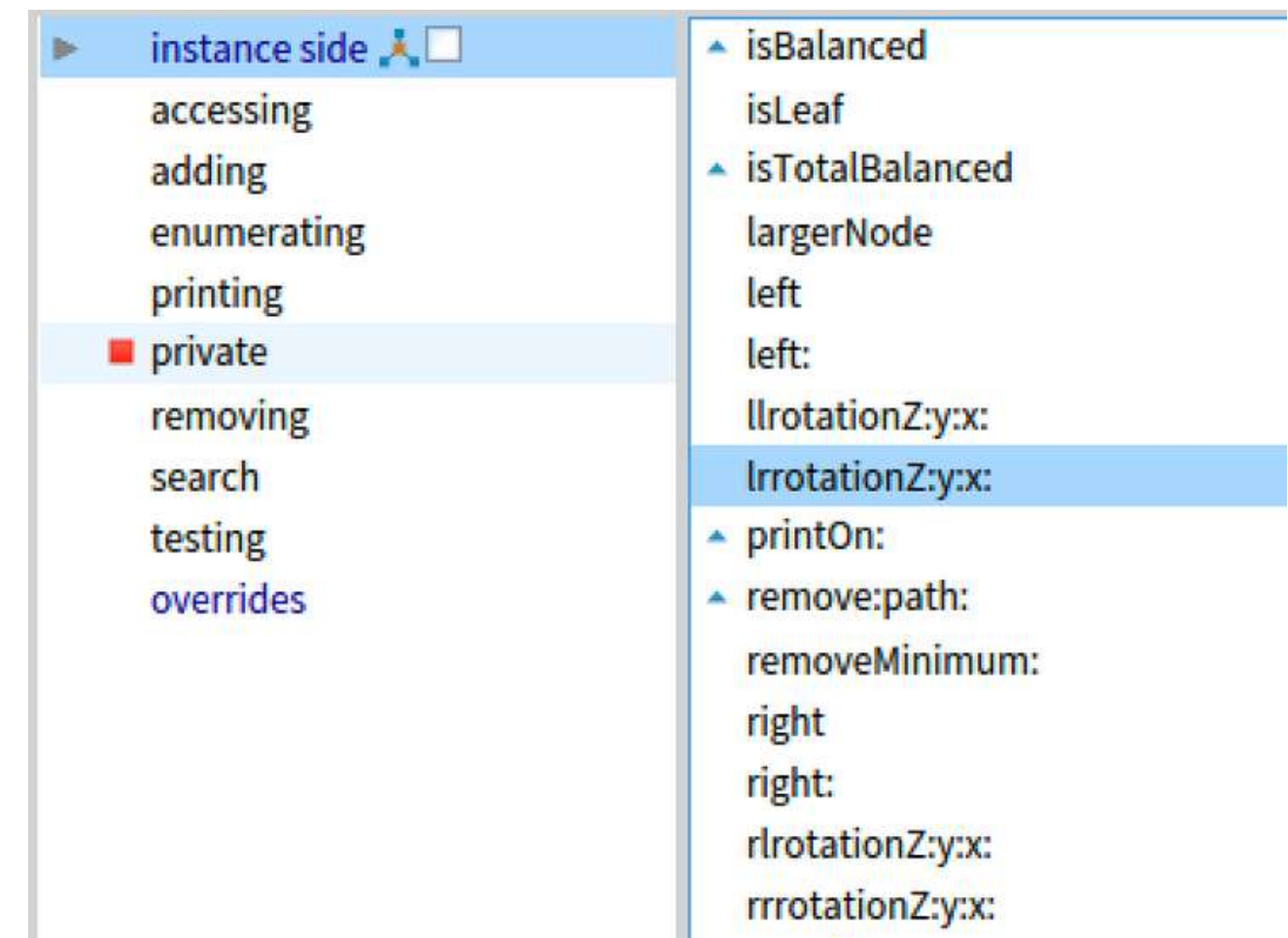
- Méthode **isBalanced** calcule la hauteur des sous-arbres gauche et droit, vérifie que leur différence absolue est ≤ 1 .

■ Maintien de l'Équilibre:

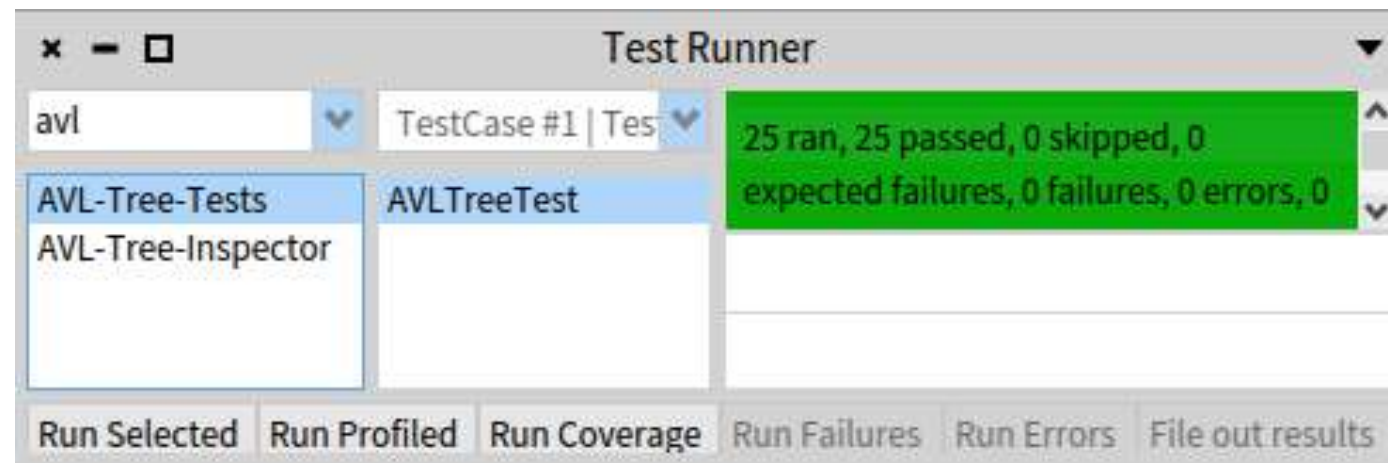
- Utilisation de quatre types de rotations pour rééquilibrer les nœuds déséquilibrés.

■ Post-Opérations:

- checkPath: Parcours du chemin d'ajout pour rééquilibrer si nécessaire.
- checkRemovingPath: Vérifie l'équilibre après une suppression et rééquilibre si nécessaire.



Tests



Tests Positifs

- Vérifie le bon fonctionnement avec des entrées valides (*testAddForLLrotation*, *testAddOneElement*).

Tests Négatifs

- Examine le comportement avec des entrées inattendues (*testRemoveZero*).

Tests de Limite

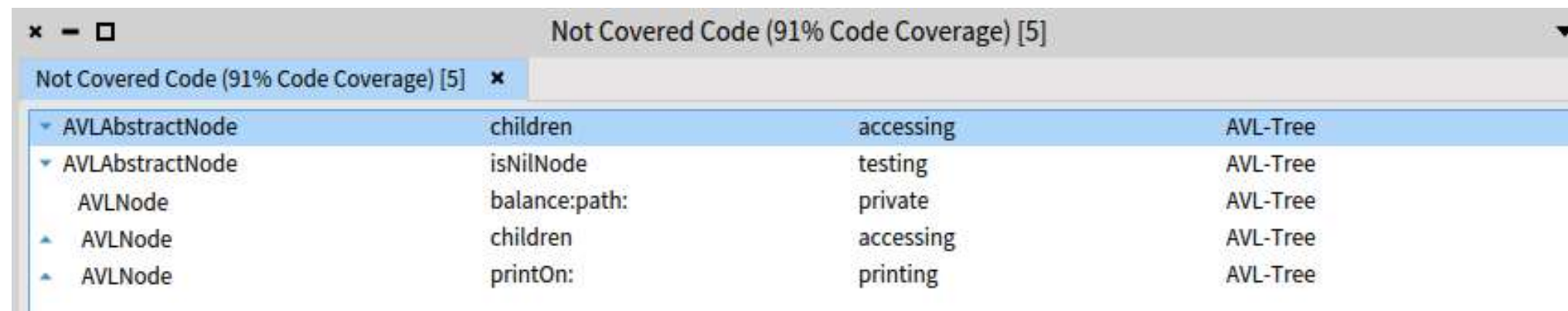
- Évalue le comportement aux limites (*testEmpty*, *testIsLeaf*).

Tests d'Inspecteur

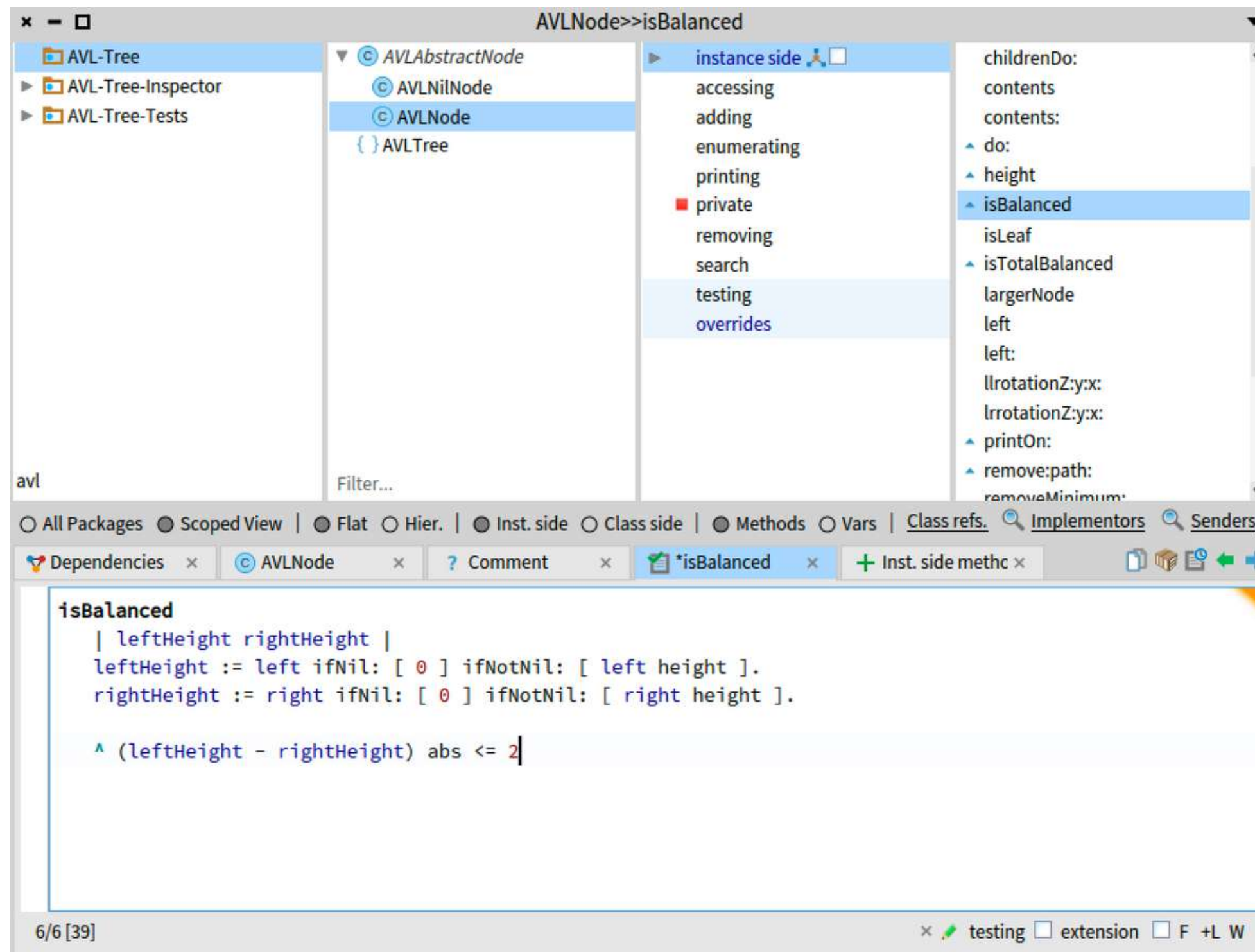
- Tests pour la représentation visuelle (*testCreateCanvas*).

Autres Types de Tests

- Tests de stress et d'intégration (*testSeriousAdd*).



Mutation



The screenshot shows the AVLNode class in a code editor. The left sidebar displays the project structure with folders AVL-Tree, AVL-Tree-Inspector, and AVL-Tree-Tests. The main editor shows the AVLNode class with the isBalanced method highlighted. The right sidebar shows the instance side of the class, listing methods like accessing, adding, enumerating, printing, private, removing, search, testing, and overrides. The bottom pane shows the implementation of the isBalanced method:

```
isBalanced
| leftHeight rightHeight |
leftHeight := left ifNil: [ 0 ] ifNotNil: [ left height ].
rightHeight := right ifNil: [ 0 ] ifNotNil: [ right height ].

^ (leftHeight - rightHeight) abs <= 2
```



The screenshot shows the Test Runner window. The left sidebar lists the test cases: AVL-Tree-Tests and AVL-Tree-Inspector. The right pane shows the test results for AVL-Tree-Tests, indicating that 25 tests ran, 25 passed, 0 were skipped, 0 were expected failures, 0 were failures, 0 were errors, and 0 were warnings.

Test Case	Test Results
AVL-Tree-Tests	25 ran, 25 passed, 0 skipped, 0 expected failures, 0 failures, 0 errors, 0

Point important du maintien de
l'invariant non respecté

Tests ajoutés

Vérifiez si l'arbre revient à un état équilibré après une série de suppressions et d'ajouts déséquilibrants.

```
testConvergence [  
    "The tree should converge to a balanced state after unbalancing operations"  
    tree addAll: { 1. 2. 3. 4. 5 }.  
    tree remove: 2.  
    tree remove: 3.  
    tree add: 6.  
    self assert: tree isBalanced.  
]
```

```
26 ran, 26 passed, 0 skipped, 0 expected failures, 0 failures, 0 errors, 0 passed  
unexpected
```

Tests ajoutés



```
testStressTestingWithRandomOperations [  
  "A stress test that performs a series of random add and remove operations  
  while checking that the AVL property holds."  
  
  | r size numbers ops |  
  r := Random new.  
  r seed: 12345. "Using a seed for reproducibility"  
  size := 1000.  
  numbers := Set new: size.  
  
  "Generate a series of random operations (either add or remove)"  
  ops := (1 to: size) collect: [ :i |  
    (r nextInteger: 100) < 50 ifTrue: [ #add ] ifFalse: [ #remove ]  
  ].  
  
  ops do: [ :op |  
    | num |  
  
    "Generate a random number for this operation"  
    num := r nextInteger: size.  
  
    "Perform the operation and check AVL property"  
    op = #add ifTrue: [  
      tree add: num.  
      numbers add: num.  
    ] ifFalse: [  
      tree remove: num ifAbsent: [ "do nothing" ].  
      numbers remove: num ifAbsent: [ "do nothing" ].  
    ].  
  
    "Check that the tree is balanced and contains the correct elements"  
    self assert: tree isBalanced.  
    self assert: (tree asSortedCollection = numbers asSortedCollection).  
  ].  
]
```

Test de l'Arbre AVL avec Opérations Aléatoires

1. Objectif

- Simuler un usage réel avec des opérations imprévisibles sur l'arbre AVL.

2. Méthodologie

- Utilise un générateur de nombres aléatoires pour les opérations add et remove.

3. Vérifications

- a. L'arbre reste équilibré (Invariant AVL).
- b. Les opérations add et remove sont correctes.
{nombres dans l'arbre} = {nombres qui ont été ajoutés}

27 ran, 27 passed, 0 skipped, 0 expected failures, 0 failures, 0 errors, 0 passed unexpected

Projet Artefact



Overview

- **Installation:**
 - Installation simple avec Metacello en utilisant GitHub.
- **Guides d'Utilisation:**
 - Exemples et démos dans le package "Artefact-Examples".
- **Objectif et Justification du Projet:**
 - Génération efficace de documents PDF en Smalltalk.
 - Aucune dépendance native requise.
 - Styles réutilisables et éléments prédéfinis (datagrids)

Core components



The screenshot displays the Artefact IDE interface. The top-left pane shows a project tree with 'Artefact-Core' selected. The top-right pane shows the 'PDFPage' class structure, including instance side methods like 'accessing', 'action', 'initialization', 'overridden', and 'overrides'. The bottom pane shows the 'Class: PDFPage' documentation, which includes a description: 'A PDFPage is a page of a PDFDocument. A page is composed by a collection of PDFElements.' and a list of instance variables: 'document: <Object>', 'elements: <Object>', 'format: <Object>', and 'margins: <Object>'.

- **ManifestArtefactCore** : core functionalities
- **PDFDocument** : Gestion et création de documents PDF.
- **PDFGenerator** : Responsable de la création de PDFs.
- **PDFPage** : Manipulation des pages PDF.

Oragnisation du projet



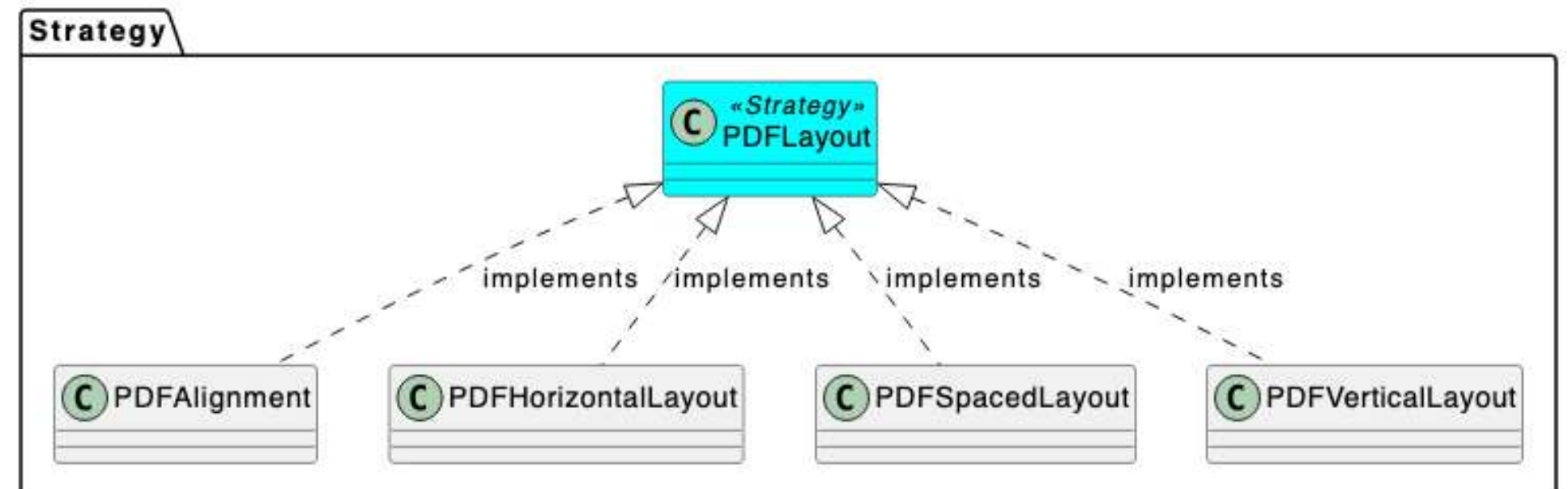
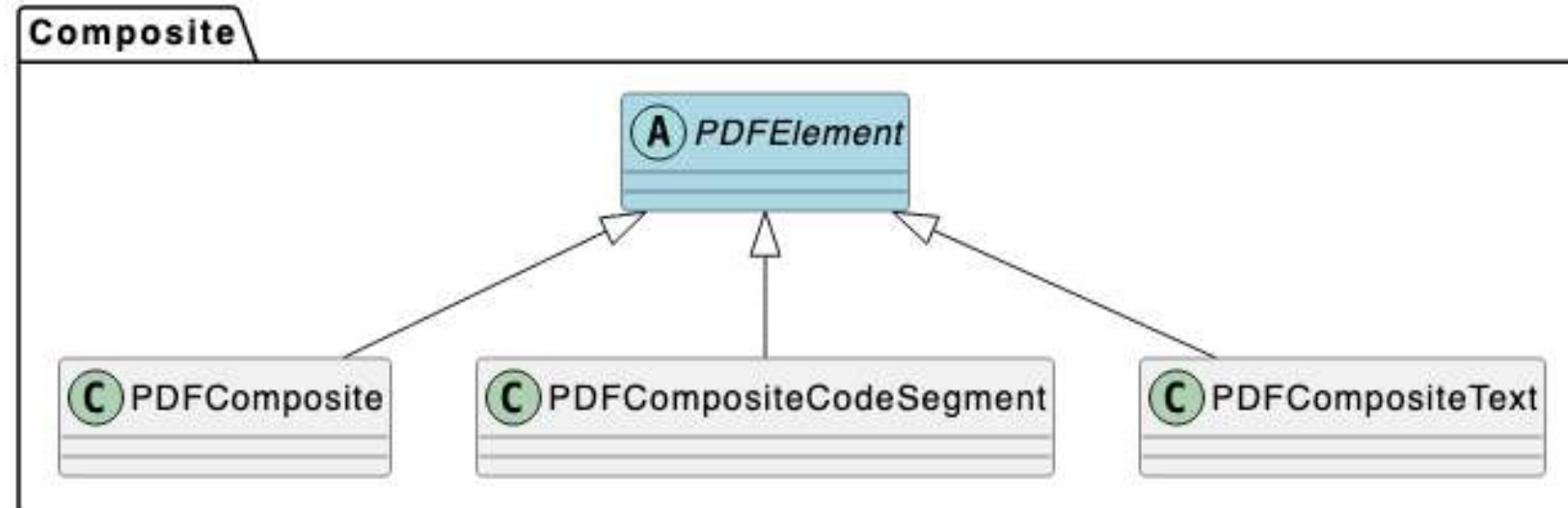
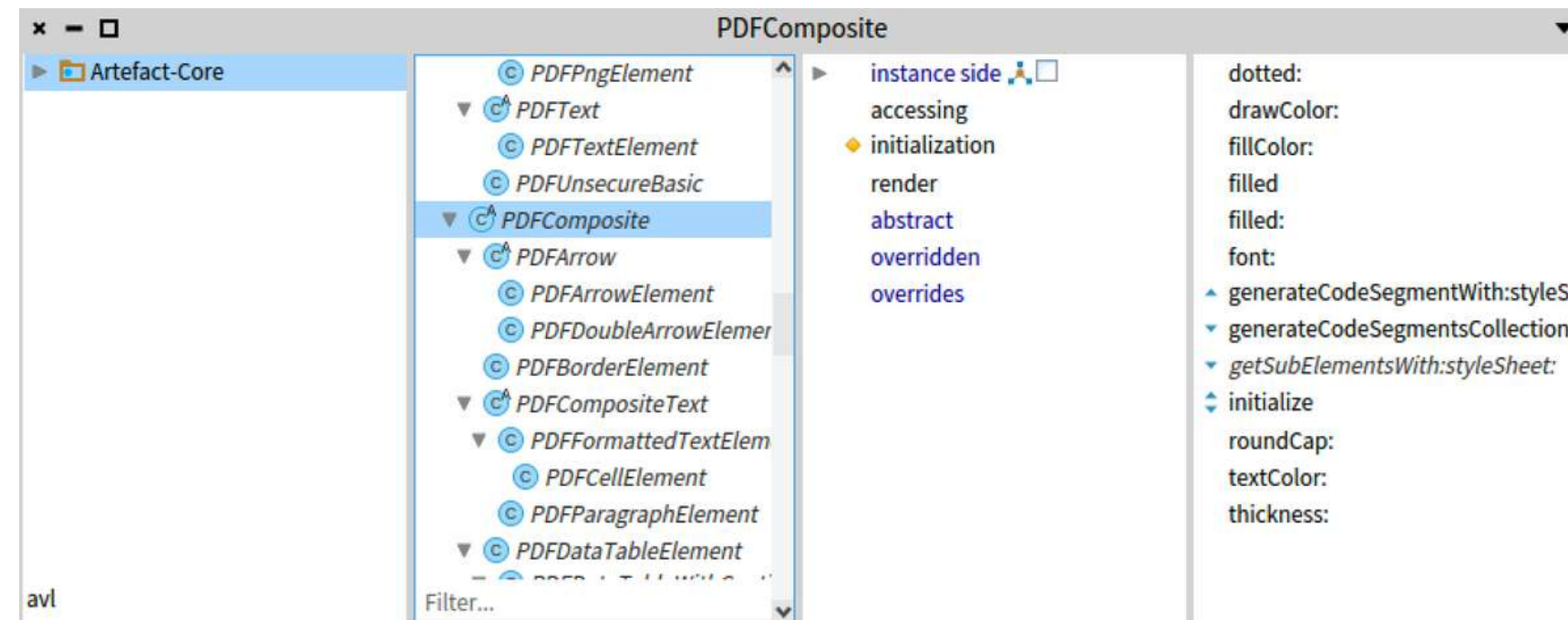
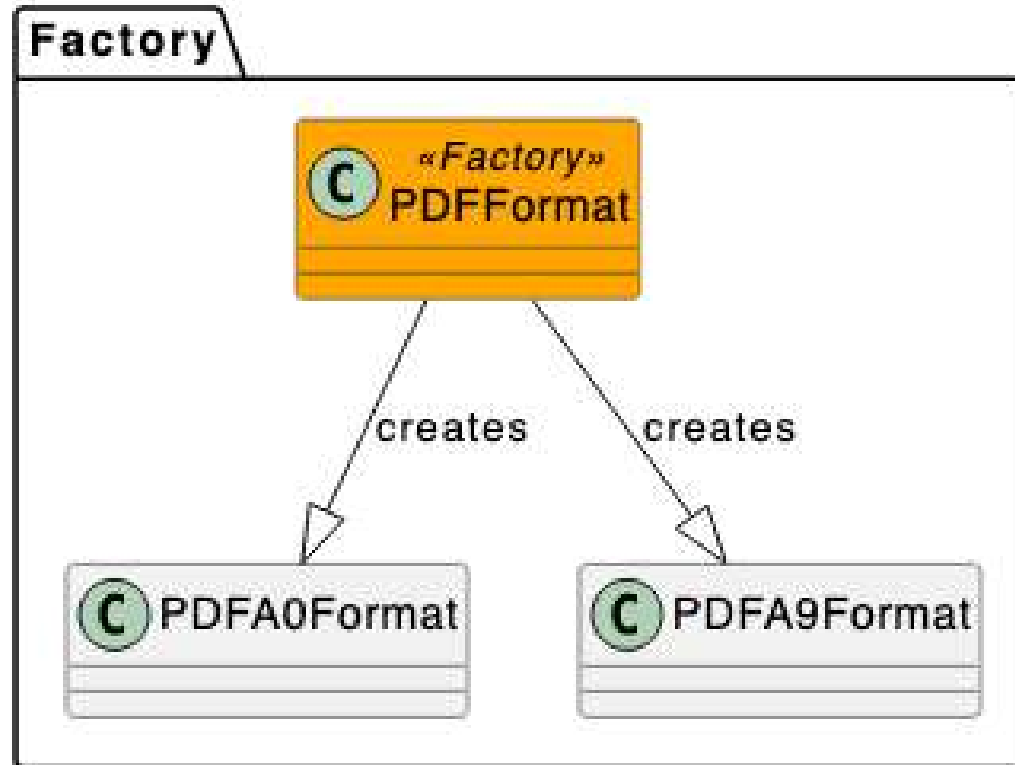
Organisation du Projet

- *Structure claire*: Le projet est bien organisé avec une séparation claire des préoccupations.
- *Classes spécifiques*: Plusieurs classes sont dédiées à des aspects spécifiques. Avec des nomenclature descriptive

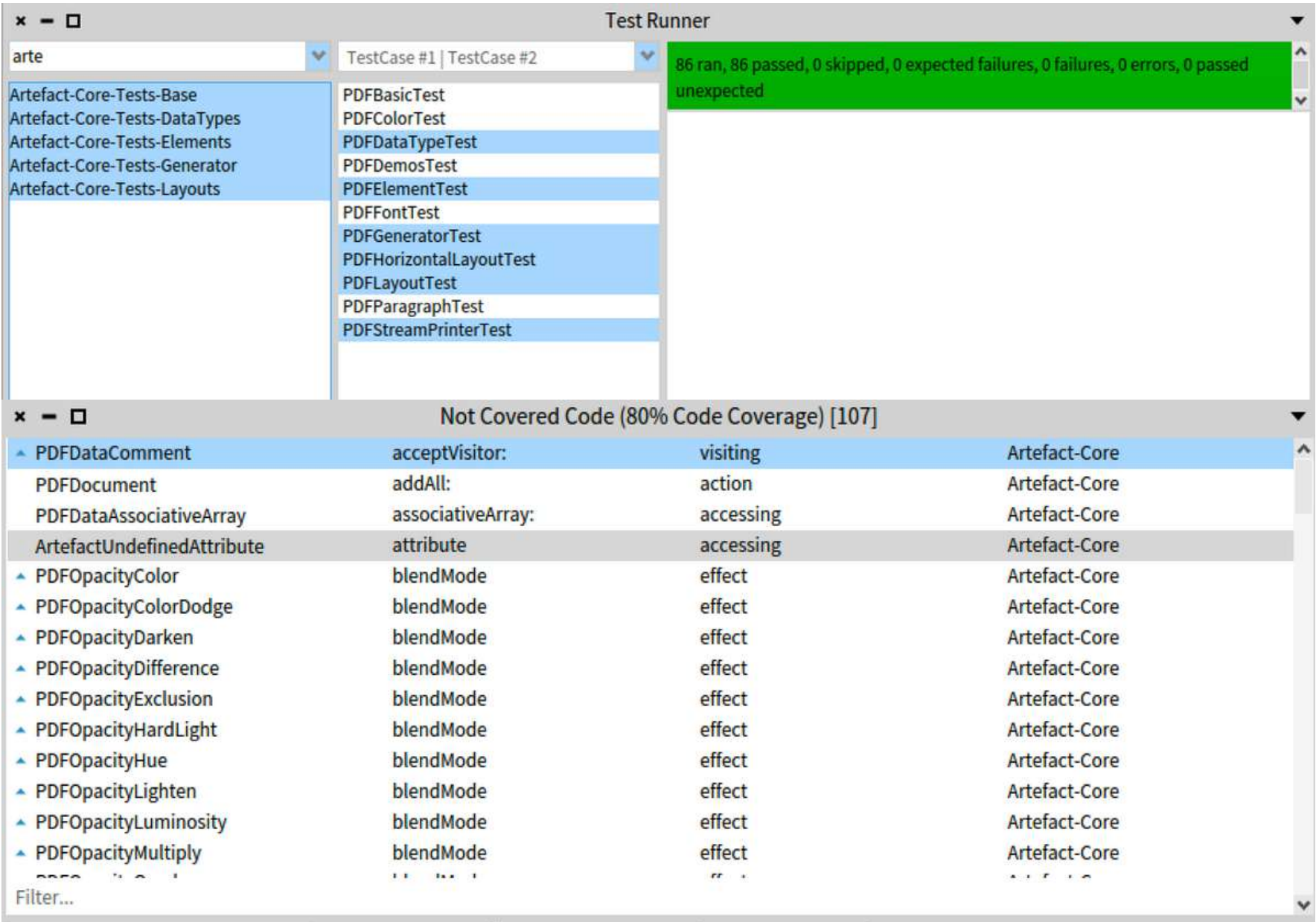
Manipulation des Données

- *Encapsulation des données*: Des classes telles que PDFDataArray, PDFDataComment, etc., montrent l'effort pour encapsuler divers types de données PDF.

Structure et Design Patterns



Tests



Tests Positifs

- Vérifie le bon fonctionnement avec des entrées valides (PDFBasicTest.class.st, PDFColorTest.class.st, PDFDataTypeTest.class.st, PDFElementTest.class.st, PDFFontTest.class.st).

Tests Négatifs

- Examine le comportement avec des entrées inattendues (PDFDemosTest.class.st, PDFStreamPrinterTest.class.st).

Tests de Limite

- Évalue le comportement aux limites (PDFGeneratorTest.class.st, PDFHorizontalLayoutTest.class.st).

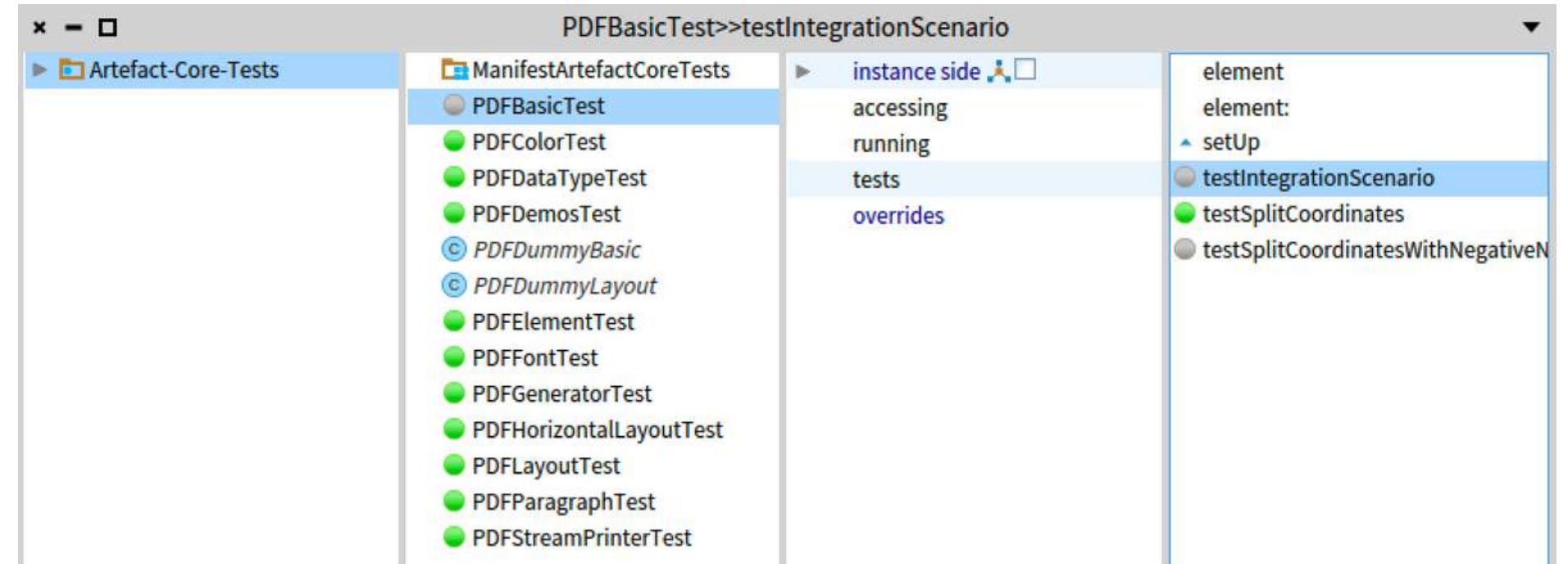
Autres Types de Tests

- Tests de performance, des tests d'intégration, etc.
- (PDFDummyBasic.class.st, ManifestArtefactCoreTests.class.st)

Test ajoutés

Test d'intégration

```
testIntegrationScenario [  
  "Etape 1: Configuration initiale"  
  self setUp.  
  self assert: element notNil.  
  self assert: element class = PDFDummyBasic.  
  
  "Etape 2: Vérification des coordonnées"  
  self assert: (self element splitCoordinates: 20@30) equals: '20 30'.  
  
  "Etape 3: Modification et re-vérification"  
  self element: PDFDummyBasic new.  
  self assert: (self element splitCoordinates: 40@50) equals: '40 50'.  
  
  "Etape 4: Test de coordonnées négatives"  
  self assert: (self element splitCoordinates: -20@30) equals: '-20 -30'.  
  
  "Etape 5: Test avec coordonnées zéro"  
  self assert: (self element splitCoordinates: 0@0) equals: '0 0'.  
  
  "Etape 6: Nettoyage et vérification finale"  
  element := nil.  
  self assert: element isNil.  
]
```





Merci pour votre attention

Question ?

