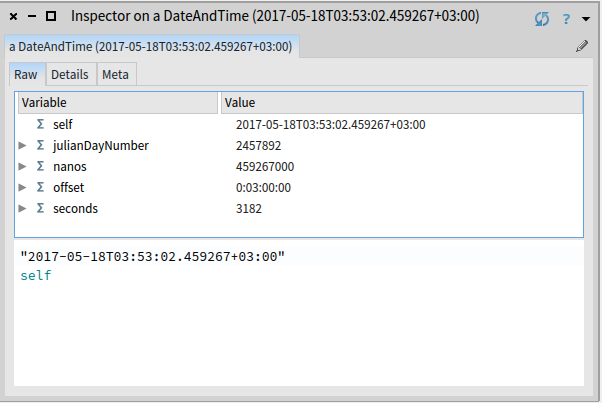
**7.3 Інспектор**

Одною з речей які принципово відрізняють Pharo від інших середовищ програмування є можливість спостерігати за об”єктами в дії, а не їх статичними знімками. Кожен з цих об”єктів може бути переглянутий програмістом і навіть модифікованим(хоча об”єкти які підтимують стан системи потрібно змінювати з обережністю). Неодмінно поекспериментуйте з цими можливостями, але зробіть резервну копію системи перед цим!

Для розкриття можливостей інспектора, введіть в пісочниці DateAndTime і виберіть з контекстного меню Inspect it (Ctrl-I) або Do in and go (Ctrl-g) (друга команда відкриває інспектор у вікні пісочниці, тоді як перша відкриває нове вікно інспектора).

Рис. 7.15, “Інспектуємо” DateAndTime now

Візьміть до уваги, що часто необхідно вибрати текст перед викликом меню. Якщо текст не вибраний, операції контекстного меню спрацюють на поточній стрічці.

Вікно схоже на показане на рис. 7.15 появиться. Це вікно інспектора, яке можна сприймати як огляд внутрішні складові певного об”єкта - в даному випадку, конкретного екземпляра типу DateAndTime, який був створений при виконанні DateAndTime now. Заголовок вікна містить репрезентацію даного об”єкта в версії для виведення.

В вкладці за замовчуванням (вкладка Raw) поля екземпляру можна переглянути вибравши їх зі списку в стовпці Variable. При виборі поля його версія для виведення появляється в стовпці Value. Також, при виборі поля, окремий інспектор для цього поля появляється в правій частині вікна.

Для змінних які є примітивними типами (boolean, integer, etc) вкладений інспектор не відрізняється від весії для виведення в стовпці Value (хоча це повнофункціональний інспектор). Але для складніших типів, вкладений інспектор містить свою вкладку Raw зі своїм списком полів екземпляра (Ці поля також можна побачити в початковому вікні інспектора, розгорнувши трикутник біля поля екземпляру).

Можна продовжити вникати в ієрархію полів екземплярів з новими вкладеними інспекторами з правої сторони батьківського вікна. Проте, щоб уникнути нагромадження вкладених вікон, панелі зміщаються вправо всередині вікна інспектора. Ви можете відслідковувати свою поточну сторінку а також переходити до попередніх використовуючи індикатори внизу вікна інспектора.

Також, існують спеціальні інспектори для об”єктів типу Dictionary, OrderedCollection, CompiledMethod і деяких інших класів. Ці інспектори,в додачу до вкладки Raw, мають інші вкладки, які полегшують огляд вмісту цих спеціальних об”єктів. Наприклад, інспектор для екземпляру класу Dictionary має додаткову вкладку Items, яка відображає ключі і значення об”єкта в інтуїтивно зрозумілій манері.

Нижня частина вікна інспектора представляє собою зменшене вікно пісочниці. Це дуже корисно, бо в цьому вікні псевдозмінна self прив”язана до об”єкту вибраного в цій частині вікна. Це означає, що Ви можете виконувати частини коду в цій частині вікна з використанням self вибраного екземпляра.

Наприклад, відкрийте інспектор на екземплярі DateAndTime now як було показано раніше в цій секції. Ви можете вибрати нижню панель з пісочницею та виконати вираз self - DateAndTime today. Результатом буде об”єкт типу Duration, який представлятиме собою інтервал від початку поточного дня до моменту створення екземпляру DateAndTime який ви оглядаєте. Також ви можете виконати фрагмент коду DateAndTime now - self; це вкаже вам на кількість часу яку ви провели на даній секції цієї книги!

Нижня панель особливо корисна у випадках, коли ви хочете змінити поля екземпляру який ви переглядаєте. За умови що у полів екземпляру є методи-аксесори, ви можете надсилати повідомлення до self та змінити його поля через ці методи.

**7.4 Відлагоджувач**

Беззаперечно, відлагоджувач є найпотужнішим засобом в збірці Pharo. Він використовується як для відлагодження, так і для написання нового коду. Щоб продемонструвати роботу відлагоджувача, давайте спочатку створимо неполадку.

Використовуючи браузер, додайте наступний метод до класу String:

suffix

"assumes that I'm a file name, and answers my suffix, the part after

the last dot"

| dot dotPosition | dot := '.'. dotPosition := (self size to: 1 by: -1) detect: [ :i | (self at: i)

= dot ]. ^ self copyFrom: dotPosition to: self size

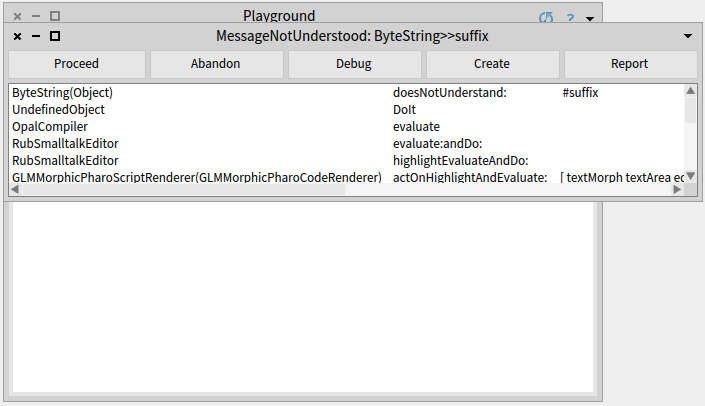
****

Figure 7.16: Вікно PreDebugWindow сповіщає про неполадку.

Звісно, ми впевнені, що настільки тривіальний метод буде працювати. Тому, замість того, щоб створити SUnit тест, ми просто напишемо ‘readme.txt’ suffix в пісочниці і виконаємо Print it (Ctrl-p). Яка несподіванка! Замість виведення очікуваного результату ‘txt’, ми отримали вікно PreDebugWindow, як показано на рис. 7.16.

Вікно PreDebugWindow має заголовок, який інформує яка помилка виникла і містить розгортку стеку повідомлень, які привели до появи цієї помилки. З низу розгортки UndefinedObject>>DoIt представляє код який був скомпільований і запущений в момент вибору тексту ‘readme.txt’ suffix в пісочниці і натискання Print It. Звысно, в цоьому коді посилається повідомлення suffix до об”єкту типу ByteString (‘readme.txt’). Це спричинило виклик методу suffix батьківського класу String. Вся ця інформація записана в наступній стрічці розгортки стеку, ByteString(String)>>suffix. Переміщаючись вверх по стеку, можна замітити, що suffix послав повідомлення detect:... і наприкінці селектор detect:ifNone послав errorNotFound:.

Щоб визначити, чому розділювач крапка не був знайдений, нам необхідний сам відлагоджувач, тому натискаємо на Debug. Також, Ви можете відкрити відлагоджувач, натиснувши на стрічки розгортки стеку. Якщо ви відкриєте відлагоджувач даним методом, він відкриється на відповідному методі.

Відлагоджувач показаний на рис. 7.17. Виглядає залякуюче на перший погляд, але його досить легко використовувати. Загловок та верхня панель аналогічні до PreDebugWindow. Але, відлагоджувач комбінує панель з розгорткою стеку з браузером методів, тож, коли ви вибираєте стрічку в розгортці, відповідний метод відображається в нижній панелі. Важливим є розуміння, що виконуваний код який викликав помилку все ще перебуває в образі Pharo, але в призупиненому стані. Кожна стрічка розгортки стеку представляє собою фрейм на стеку викликів зі всією інформацією необхідною для продовження роботи.

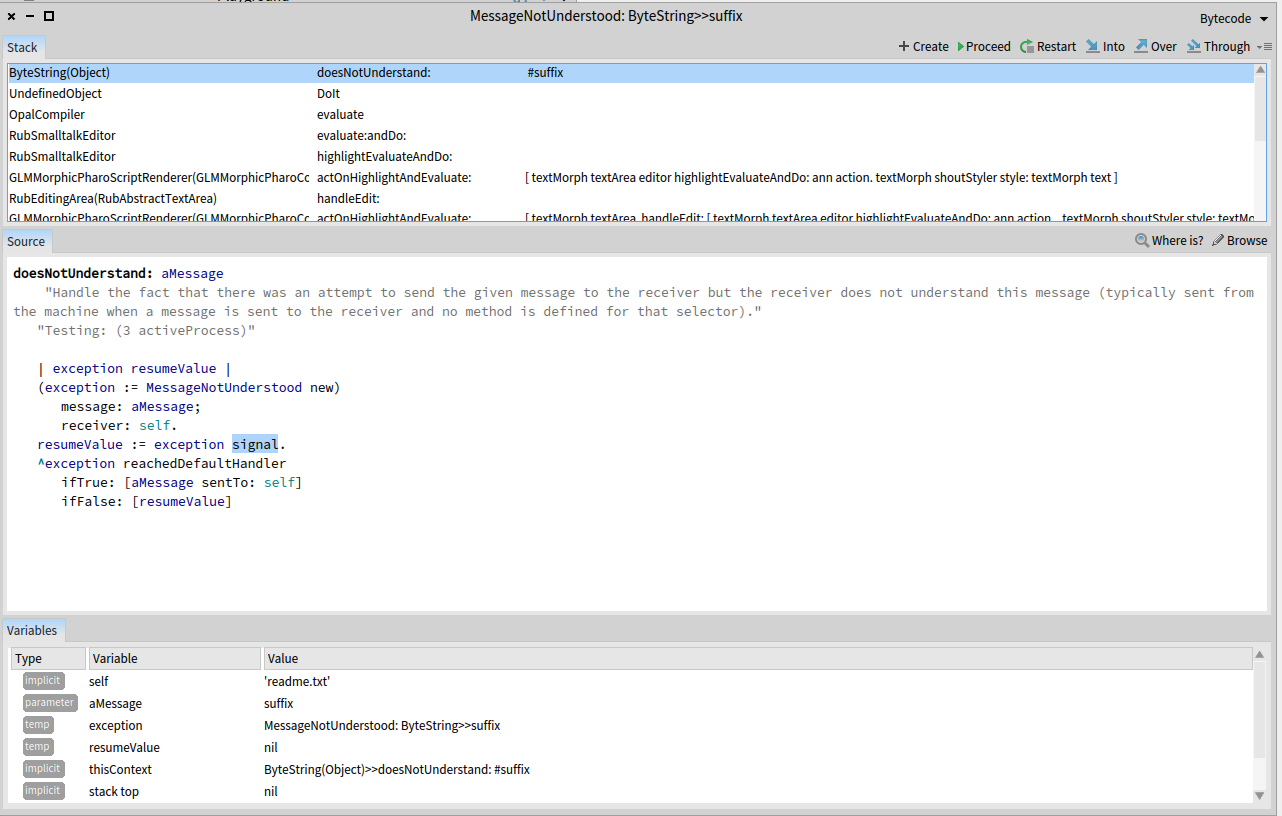


Рис. 7.17. Вікно відлагоджувача зі зображеним стеком викликів та станами різноманітних об”єктів.

Це включає всі об”єкти використані при виконанні операції з їхніми полями та локальними змінними методів.

На рис.7.17 в нас вибраний метод doesNotUnderstand: в верхній панелі. Тіло методу відображається в центральні панелі; фрагмент, виділений синім, відображає що поточний метод відіслав повідомлення та очікує на відоповідь.

Панель Variables внизу відлагоджувача є свого роду інспектором без пісочниці. Ви можете вибрати одну зі змінних та відкрити додаткову панель інспектора з уже відомими вкладками (Raw, …) і з вкладкою Evaluator, яка виконує роль пісочниці для виконання коду в контексті вибраної змінної. В стовпці Type панелі Variables можуть міститися дані типи :

• parameter - параметр переданий даному методу

• temp - локальні змінні

• attribute - поля класів доступні з контексту даного отримувача повідомлення

• implicit - псевдозмінні (self, thisContext, stackTop) використовувані в даному контексті.

При виборі різних фреймів стеку, псевдозмінна self буде змінюватись відповідно до контексту, як і список Variables. Якщо натиснути на self в панелі знизу зліва, можна замітити що self є інтервалом (10 to: 1 by -1), що підтверджує наші очікування. Ви завжди можете вибрати self і перейти на панель Evaluator щоб виконати фрагменти коду в контексті даного отримувача. Через те що всі змінні також є в області видимості панелі методу, ви можете вибирати вирази та виконувати вирази прямо з панелі методу. Ви завжди можете відмінити (l) ваші змінні використовуючи контекстне меню або Ctrl-L.

При виборі thisContext зі списку змінних відображається поточний контекст об”єкту.

Як можна замітити на розгортці стеку exceptionBlock є [self errorNotFound: ...], тож, нічого дивного в відображенні відповідного повідомлення про помилку немає.

Також, якщо ви бажаєте відкрити повноцінний інспектор для одної зі змінних відображених в міні-інспекторах просто натисніть двічі по імені змінної або виберіть змінну зі списку та натисніть Inspect (i). Це може бути корисним, якщо ви хочете прослідкувати за станом цієї змінної при подальшому виконанні коду.

Подивившись на панель методу, ми бачимо, що ми очікували знайти ‘.’ в стрічці ‘readme.txt’ і при виконанні метод не повинен був досягнути останньої стрічки коду. Pharo не надає можливості переходів назад при виконанні коду, але воно дає можливість викликати метод заново, що дуже добре працює з методами які не міняють об”єкти, а створюють нові.

Натисніть Restart і ви побачите як фокус виконання повертається до першої конструкціїї даного методу. Синє виділення покаже вам, що наступне повідомлення наділсане є do: (рис.7.18)

Кнопки Into та Over надають нам 2 різні способи проходу коду. Якщо ви натиснете Over, Pharo виконає надсилання поточного повідомлення в один крок, якщо це не викликає помилки. Тобто, Over перенесе нас на наступне надсилання повідомлення в поточному методі, яке є value, що є місцем початку і не є достатньо інформативним. Нам потрібно визначити, чому do: не знаходить шуканий символ.

Після натискання Over, натисніть Restart щоб вернутись до стану описаного на рис. 7.18.

Натисніть Into двічі; Pharo перейде в метод відповідно до виділеного надсилання повідомлення, в даному випадку - Interval>>do:.

Але, цей перехід є також не достатньо інформативним, бо ми можемо бути впевнені, що Interval>>do: працює коректно. Тому несправність скорше всього є в запиті до Pharo. Through є кнопкою яка нам потрібна в даному випадку : ми хочемо проігнорувати деталі do: і перейти до виконання блоку.

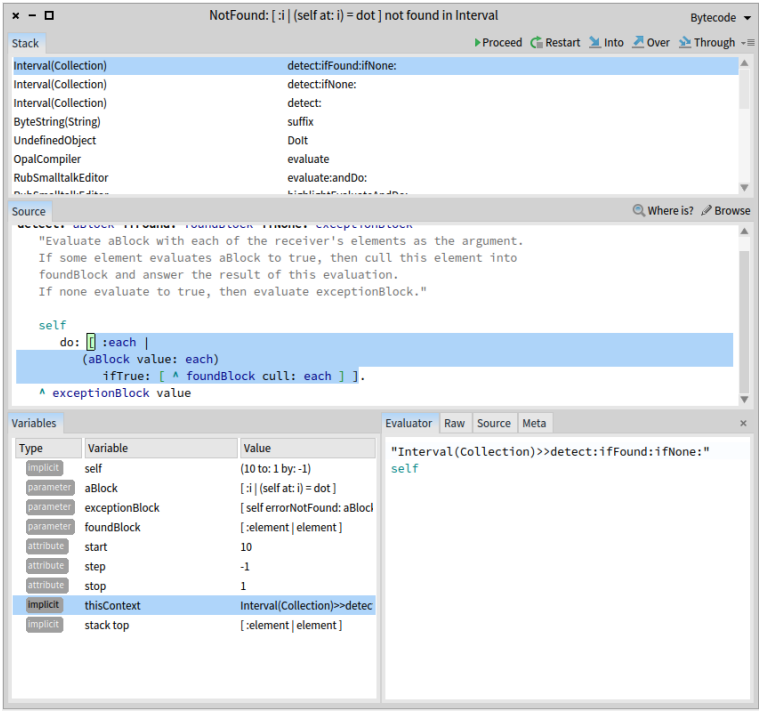


Рис7.18: Відлагоджувач після перезапуску на методі detect:ifFound:IfNone:

Виберемо метод detect:ifFound:IfNone: знову і натиснемо Restart, щоб перейти до початку виконання показаного на рис. 7.18. Тепер натиснемо на Through декілька разів. Виберіть each у вікні контексту. Спостерігайте як each зменшується починаючи від 10 поки do: виконується.

Коли each рівне 7, ми очікуємо на виконання блоку ifTrue:, але він не виконується. Щоб побачити, що відбувається не так, перейдіть в виконання value: як показано на рис.7.19.

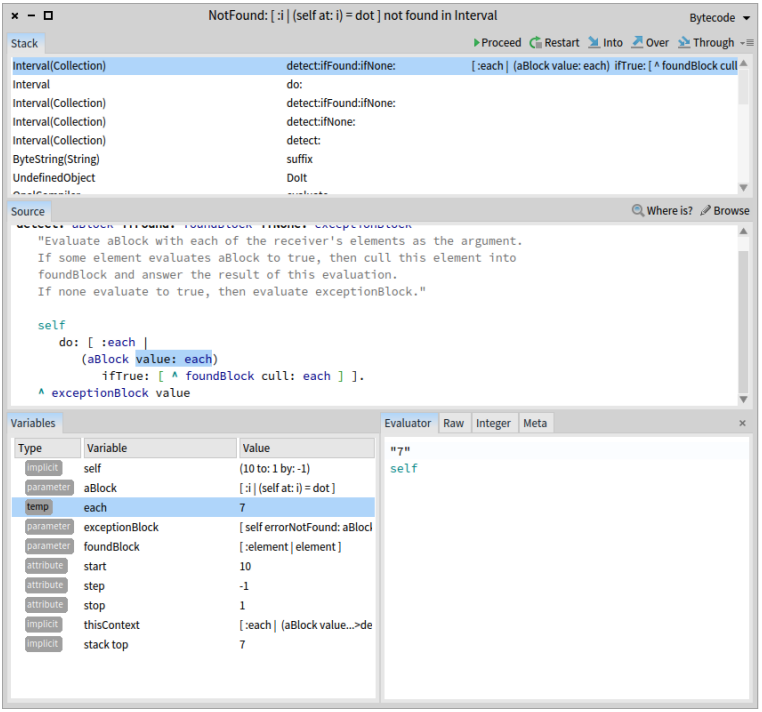


Рис. 7.19 Відлагоджувач після проходження Through через do: декілька разів

Після натискання Into, ми бачимо ситуацію описану на рис. 7.20. Ми перейшли назад в метод suffix, так як виконуваний блок був сформований в даному методі.

Якщо ви виберете крапку в інспекторі контексту, ви побачите, що значення є ‘.’. І тепер можна зрозуміти, чому вони є не рівними : сьомий символ стрічки ‘readme.txt’ є символом, тоді як крапка представлена стрічкою.

Тепер, коли неполадку локалізовано, виправлення є очевидним, необхідно конвертувати крапку до символа перед початком пошуку.

Змінимо крапку прямо у відлагоджувачі, так що присвоєння вигялає як dot := $. і приймемо зміни.

Оскільки ми виконуємо код всередині блоку, який є всередині detect:, декілька фреймів стеку повинні стати невалідними для цих змін. Pharo запитається, чи це дія на яку ми очікуєм (рис. 7.21). Натиснемо yes. Pharo збереже і скомпілює новий метод.

Виконання виразу 'readme.txt' suffix виконається успішно та виведе результат '.txt'.

Чи є цей результат коректним? На жаль, ми не можем цього визначити. Чи повинен суфікс бути .txt, чи txt? Коментарі до методу не є достатньо точними. Щоб уникнути такого роду проблем, можна написати SUnit тест, який визачає правильну відповідь.

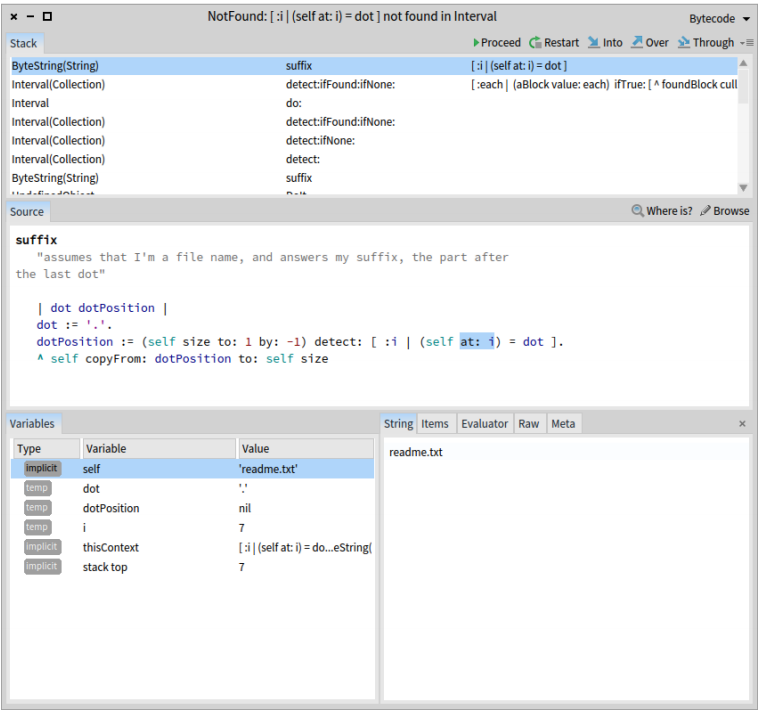


Рис 7.20: Відлагоджувач, який відображає чому 'readme.txt' at: 7 не рівне змінній dot.

testSuffixFound

self assert: 'readme.txt' suffix = 'txt'

Зусилля затрачені на написання цього фрагменту були трішки більшими, ніж запуск аналогічного тесту в пісочниці, але використання SUnit зберігає тест як виконувану документацію і полегшує його запуск для інших користувачів. Також, якщо ви додасте testSuffix до класу StringTest і запустите набір тестів ви можете швидко переміститися до відлагодження помилки. SUnit відкриває відлагоджувач на помилковому твердженні, тому вам необхідно перейти тільки на один фрейм стеку, виконати Reset і Into в метод suffix щоб виправити помилку, як зображено на рис. 7.22. Це займає миті роботи щоб натиснути на кнопку Run Failures в SUnit Test Runner щоб підтвердити правильність зміненого коду.

Наведемо покращений тест: testSuffixFound

self assert: 'readme.txt' suffix = 'txt'. self assert: 'read.me.txt' suffix = 'txt'

Ключові засоби при розробці у Pharo

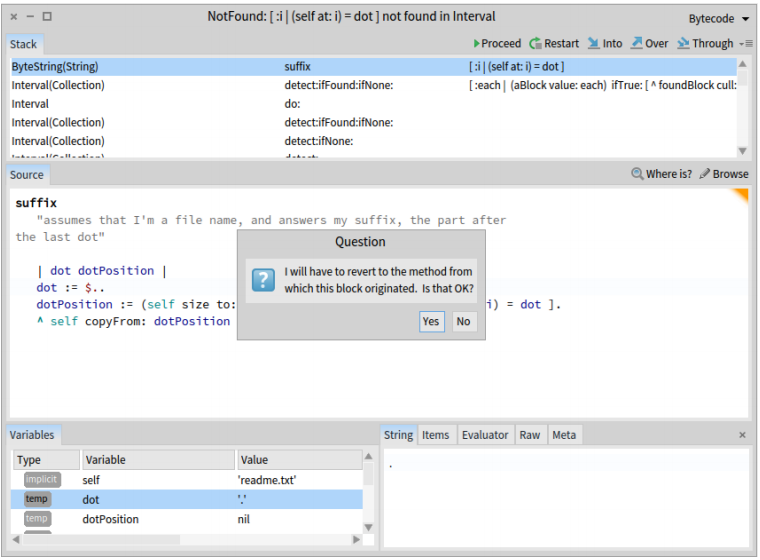


Рис 7.21: Внесення змін до методу suffix у відлагоджувачі: запит на підтверження виходу з внутрішнього блоку.

Чим даний тест кращий? Він сповіщає читачеві яким буде результат виконання при наявності більше ніж одної крапки в стрічці.

Існує декілька інших способів викликати відлагоджувач, крім отримання помилок та помилок у assert:

Якщо ви виконаєте код, який заходить в безкінечний цикл, ви можете перервати його та зайти у відлагоджувач натиснувши Ctrl-. (крапка). (також корисно знати, що завжди можна викликати резервний відлагоджувач комбінацією Ctrl-Shift-.). Ви також можете відредагувати підозрілий код і вставити Halt now.. Отож, для прикладу, ми можемо відредагувати метод suffix наступним чином : suffix

"assumes that I'm a file name, and answers my suffix, the part after

the last dot"

| dot dotPosition | dot := FileDirectory dot first. dotPosition := (self size to: 1 by: -1) detect: [ :i | (self at: i)

= dot ]. Halt now. ^ self copyFrom: dotPosition to: self size

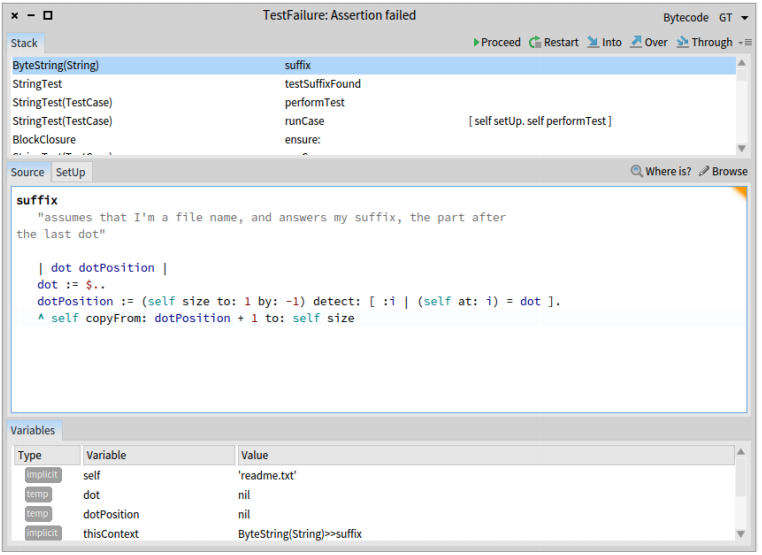


Рис7.22: Внесення змін до методу suffix через відлагоджувач: виправлення помилок одна за одною після SUnit невдач у твердженнях тесту (assert)

Коли ми запустимо цей метод, виконання Halt now викличе попередній відлагоджувач, в якому можна або продовжити виконання, або викликати власне відлаголжувач (і оглянути змінні, перейти між виразами, ітд).

Це покриває всі аспекти відлагоджувача, але не всі аспекти методу suffix. Початкова неполадка мала привести моменту, коли при відсутності крапки в стрічці виникне помилка. Дана поведінка не є очікуваною, тому додамо ще один тест, який буде демонструвати поведінку методу в цьому випадку.

testSuffixNotFound

self assert: 'readme' suffix = ''

Додайте testNoSuffix до набору тестів у класі StringTest і побачите виникнення помилки. Зайдіть у відлагоджувач вибравши помилковий тест у SUnit і змініть код щоб тести успішно виконувались.Найпростішим способом є заміна detect: на detect:ifNone:, де другим аргументом є блок, який повертає розмір переданої стрічки.

Ви дізнаєтесь більше про SUnit у розділі : SUnit.

Ключові засоби при розробці у Pharo

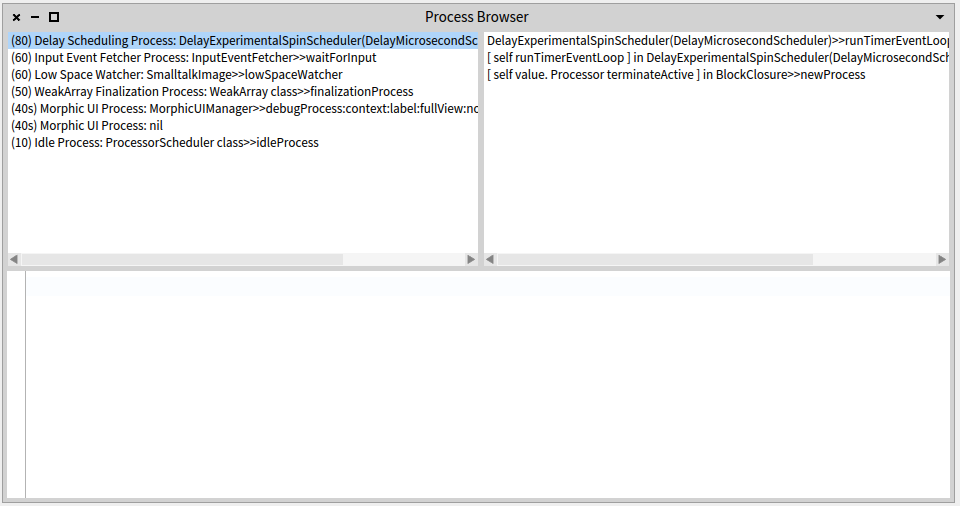


Рис 7.23: Браузер процесів, який відображає всі активні потоки за пріоритетом.

**7.5 Браузер процесів**

Pharo є багатопотоковою системою, тому в ній існує багато необтяжливих процесів (також відомих як потоки) запущених паралельно в одному образі. В майбутньому віртуальна машина Pharo буде мати можливість використовувати декілька процесорів, але на даний момент, паралелізм реалізований квантуванням часу.

Браузер процесів є родичем відлагоджувача який дозволяє вам переглядами процеси які запущені всередині Pharo. Ви можете відкрити його використовуючи World Menu, вибравши Tools > Process Browser (рис. 7.23 відображає знімок екрану). Ліва панель перераховує всі процеси у образі в порядку пріоритетів від таймеру переривань з пріоритетом 80 до незайнятого процесу з пріоритетом 10. Звісно, на одному процесорі, єдиним процесом який виконується на даний момент буде інтерфейсний процес, решта процесів будуть очікувати на певну подію.

За замовчуванням, відображення процесів є статичним; можна включити оновлення натиснувши правою клавішею миші і вибравши Turn on auto-update (Сtrl-a).

Якщо ви виберете процес у лівій верхній панелі, його розгортка стеку відобразиться у верхній правій панелі, аналогічно до відлагоджувача. Якщо ви виберете фрейм стеку, відповідний метод відобрахиться у нижній панелі. Браузер процесів не містить міні-інспекторів для self та thisContext, але натиснувши правою клавішею миші можна отримати еквівалентний функціонал.

Ключові засоби середовища Pharo

**7.6 Знаходження методів**

Finder є одним з засобів пошуку коду у Pharo, який допомагає знаходити методи за іменем (або навіть функціоналом). Ця тема була частково обговорена в розділі “A Quick Tour of Pharo.”

**7.7 Підсумок Розділу**

Для ефективної розробки в Pharo, необхідно прикласти трохи зусиль щоб ознайомитись з засобами наявними в середовищі.

• Стандартний браузер є основним інтерфейсом для перегляду існуючих пакетів, класів, протоколів методів та методів, а також для визначення нових.

• Браузер надає декілька корисних скорочень щоб переходити до відправників або реалізацій повідомлення, версій методів ітд.

• З кожного засобу ви можете виділити назву класу або методу і перейти до браузера використовучи скорочення Ctrl-b.

• Ви також можете переглянути систему програмно надсилаючи повідомлення до SystemNavigation default.

• Інспектор є засобом якйи є корисний для огляду та взаємодії робочих об”єктів образу середовища.

• Відлагоджувач є засобом який не тільки дозволяє переглянути стек програми при виникненні помилки, але також дозволяє взаємодіяти з всіма об”єктами застосунку, включно з сирцевим кодом. В багатьох випадках ви можете змінити сирцевий код з відлагоджувача та продовжити виконання. Відлагоджувач є особливо ефективним засобом при підтримці розробки тестів у тандемі з SUnit (розділ : SUnit).

• Браузер процесів дозволяє вам відслідковувати та взаємодіяти з процесами запущеними у вашому образі середовища Pharo.

• Пошуковик є засобом для пошуку методів.