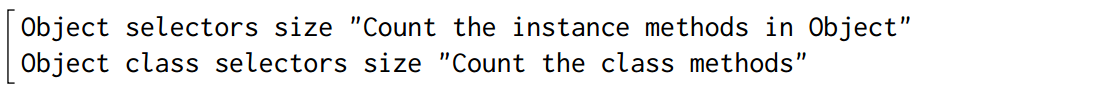
**Глава 10. Базові класи**

Pharo - це проста, але дуже потужна мова програмування. Частина її потужності полягає не в самій мові, а в її бібліотеках класів. Щоб ефективно програмувати на ній, вам потрібно буде дізнатися, як бібліотеки класів підтримують мову і середовище. Бібліотеки класів повністю написані в Pharo і можуть легко бути продовжені. (Нагадаємо, що пакет може додати нові функції до класу навіть якщо він не визначає цей клас). Наша мета тут не представити нудні деталі всіх бібліотек класів Pharo, а вказати на основні класи та методи, які вам потрібно буде використовувати щоб ефективно програмувати. У цьому розділі ми будемо охоплювати основні класи, які будуть необхідні для майже будь-якого застосування: Object, Number і його підкласи, Character, String, Symbol, and Boolean.

**10.1 Object**

Для всіх намірів і цілей, об'єкт є коренем ієрархії успадкування. Насправді, в Pharo істинний корінь ієрархії це ProtoObject, який використовується для визначення мінімальних сутностей, які замаковані як об'єкти, але ми можемо це ігнорувати на деякий час. Об'єкт визначає майже 400 методів (іншими словами, кожен клас, який ви визначили, автоматично забезпечить всі ці методи). Примітка: Ви можете порахувати число методів в класі так:



Клас Object забезпечує поведінку за замовчуванням, загальну для всіх нормальних об'єктів, таких, як доступ, копіювання, порівняння, обробка помилок, відправки повідомлення і відображення. Тут також визначені допоміжні повідомлення, на які повинні реагувати всі об'єкти. Об'єкт не має змінних екземпляра, і не повинно нічого бути додано. Це пов'язано з тим, що кілька класів об'єктів, які успадковують від Object, які мають спеціальні реалізації (SmallInteger і UndefinedObject, наприклад), віртуальна машина знає про них та вони залежать від структури і розташування деяких стандартних класів.

Якщо ми починаємо переглядати протоколи методу на стороні Object ми бачимо ключову поведінку, яку він надає.

**Друк**

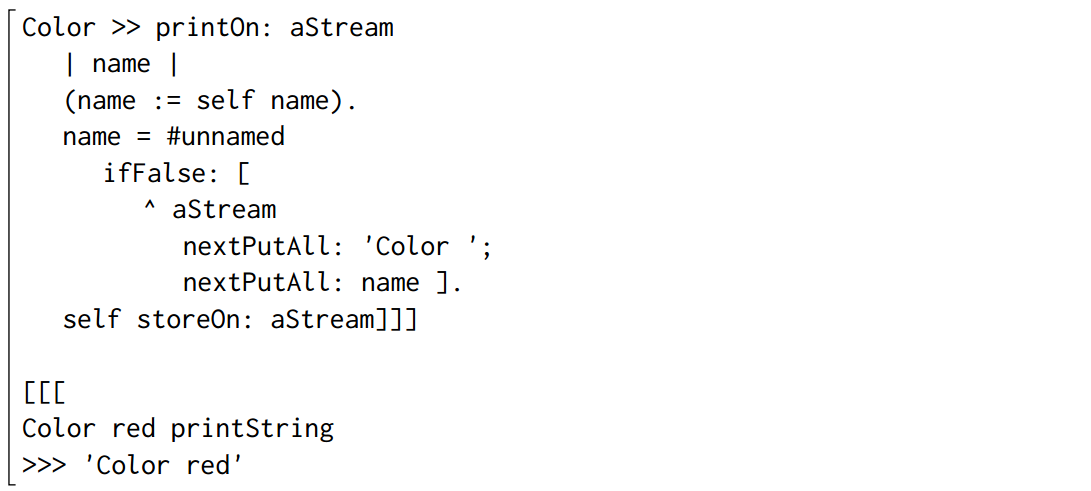
Кожен об'єкт може повернути себе у друкованій формі. Ви можете вибрати будь-який вираз в textpane,а потім вибрати пункт меню Print it: це виконує вираз і просить повернутий об'єкт видрукувати себе. Насправді це надсилається повідомлення printString для об'єкта, що повертається. Метод printString, який є шаблоном методу, посилає в його ядрі повідомлення printOn: до його отримувача. Повідомлення printOn: це гачок, який може бути спеціалізований.

Метод об'єкта >> printOn: - дуже ймовірно, що це один з методів, які ви будете найчастіше перевантажувати. Цей метод приймає в якості аргументу Stream, який буде представлено як String. Реалізація за замовчуванням просто записує ім'я класу з ‘a’ або ‘an’. Object >> printString повертає рядок, який записується.

Наприклад, в класі OpalCompiler не перевантажують метод printOn: і відправка повідомлення printString до примірника виконує методи визначені у Object.



Наприклад для класу Color printOn: спеціалізація. Він видрукує Ім'я класу за яким слідує ім'я методу класу , що використовувався для генерації кольору.

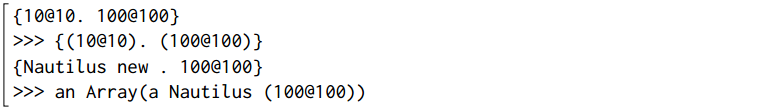


Зверніть увагу, що повідомлення printOn: це не те ж саме, що і storeOn:. Повідомлення storeOn: пише в свій потік аргументів вираз, який може бути використаний для оновлення приймача. Цей вираз виконується, коли потік прочитаний використовуючи повідомлення readFrom:. З іншого боку, повідомлення printOn: просто повертає текстову версію приймача. Звичайно, може статися так, що це текстове представлення може являти собою приймач в якості самооцінки виразу.

**Слово про представлення і самооцінки представлення.** У функціональному програмування, вирази повертають значення при виконанні. У Pharo, повідомлення (вирази) повертають об'єкти (значення). Деякі об'єкти мають вдастивість nice, їх значення дорівнюють собі. Наприклад, значення об'єкта true є об'єкт true. Ми називаємо такі об'єкти самооцінювальні об'єкти. Ви можете побачити друковану версію значення об'єкта при друку об'єкта. Ось деякі приклади таких самооцінювальних виразів.

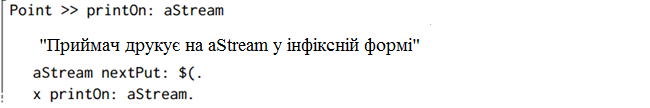
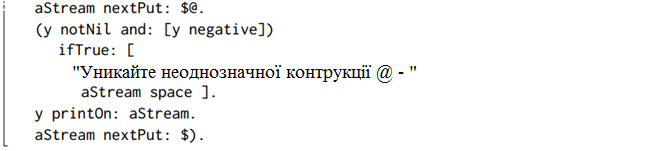


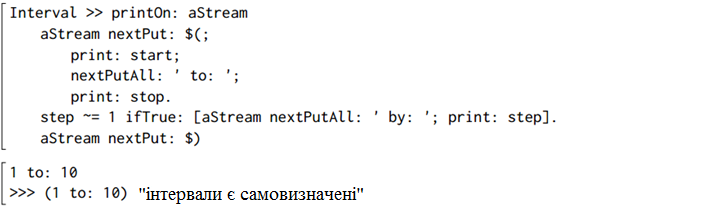
Зверніть увагу, що деякі об'єкти, такі як масиви є самооцінювальні або незалежними від об'єктів, які вони містять. Наприклад, масив логічних значень є самооцінювальним, в той час як масив осіб не є. Наступний приклад показує, що динамічний масив є самооцінювальний, тільки якщо її елементи:



Пам'ятайте, що літеральні масиви можуть містити тільки літерали. Звідси випливає, що масив не містить двох точок, а, швидше за шість літералів.

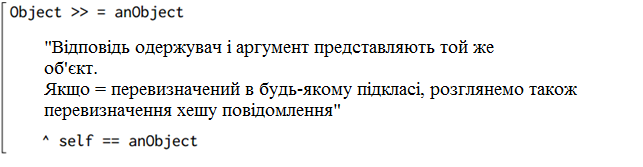


Більшість спеціалізацій методу printOn: реалізовують самооцінювальну поведінку. Реалізації Point >> printOn: і Interval >> printOn: є самооцінювальні. 

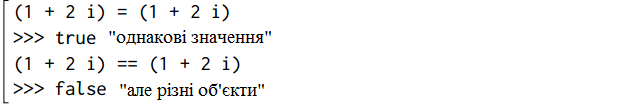


**Ідентичність і рівність**

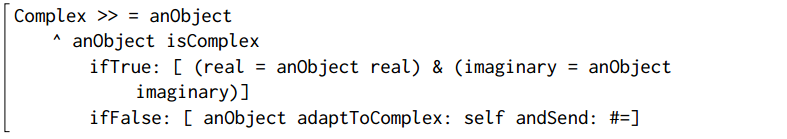
У Pharo, повідомлення = випробування рівності об'єкта (тобто, незалежно від того, чи представляють два об'єкти те ж саме значення), тоді як == випробувань ідентичності об'єкта (чи об’єкти з двох виразів являють собою один і той же об'єкт). Реалізація за замовчуванням рівності об'єкта є перевіркою ідентичності об'єкта:



Це метод, який ви часто хочете перевизначити. Розглянемо випадок комплексних чисел, як це визначено в SciSmalltalk/PolyMath (PolyMath являє собою набір пакетів, які пропонують підтримку для чисельних методів):



Це працює тому, що Complex перевантажує = наступним чином:

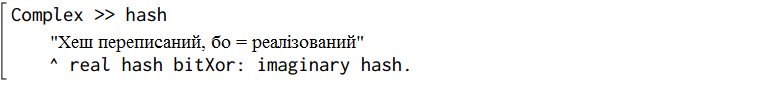


Реалізація за замовчуванням Object >> ~ = (тест на нерівність) просто

нівелює Object >> =, і не повинен, як правило бути змінений.



Якщо перевизначити =, то ви повинні розглянути перевантажений хеш. Якщо екземпляри вашого клас коли-небудь використовувати в якості ключів в словнику, то ви повинні переконатися, що випадки, які вважаються рівними мають однакове значення хеш-функції:



Хоча ви повинні перевизначити = і хеш разом, ви ніколи не повинні перевизначати ==. Семантика ідентичності об'єкта однакова для всіх класів. Повідомлення == є примітивним методом ProtoObject.

Зверніть увагу, що Pharo має деяку дивну поведінку рівності в порівнянні з іншими середовищами Smalltalks. Наприклад, символ і рядок можуть бути рівні. (Ми вважаємо, що це помилка, а не вдастивість.)



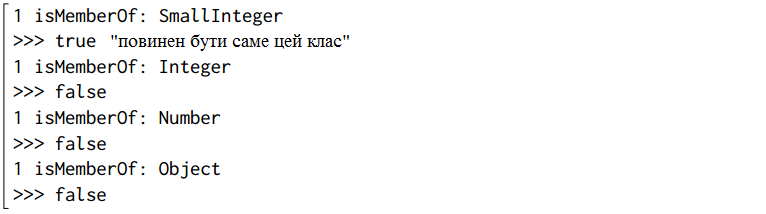
**Члени класу**

Деякі методи дозволяють запросити клас об'єкта.

**class.** Ви можете запитати будь-який об'єкт про свій клас, використовуючи повідомлення class.

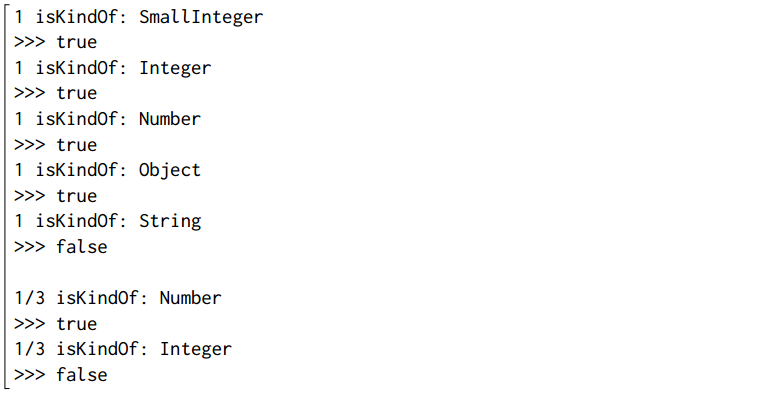


**isMemberOf:.** З іншого боку, ви можете запитати чи об'єкт є екземпляром конкретного класу:



Так як Pharo написано в собі, ви дійсно можете переміщатися по своїй структурі використовуючи правильну комбінацію суперкласу і класу повідомлень (див главу: Класи і Метакласи).

**isKindOf:.** Object>> isKindOf: відповіді чи клас одержувача такий самий, або є підкласом класу аргументу.



1/3, який є класом Fraction і свого роду класом Number, так як клас Number є суперкласом класу Fraction, а 1/3 не є класом Integer.

**respondsTo:.** Object >> respondsTo: відповіді чи одержувач розуміє

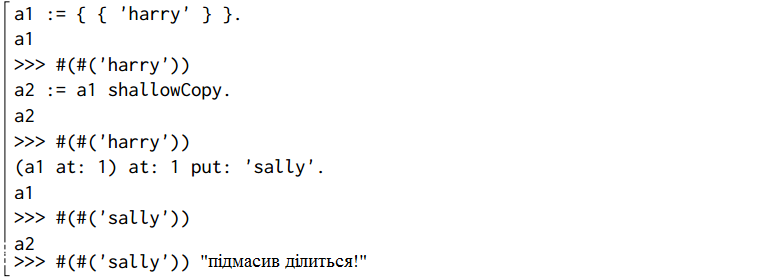
селектор повідомлень, що заданий як аргумент.



Зауваження про використання respondsTo:. Зазвичай це погана ідея запрошувати об'єкт для свого класу, або запитувати його, які повідомлення він розуміє. Замість того, щоб приймати рішення, базовані на класі об'єкта, потрібно просто відправити повідомлення на об'єкт і дати це вирішити йому(на основі свого класу), як він повинен себе вести. (Ця концепція іноді згадується як duck typing).

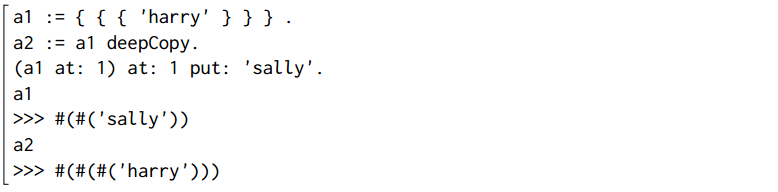
**Копіювання**

Для копіювання об'єктів вводять деякі гострі питання. Так як змінні екземпляра доступні за посиланням, неповна копія об'єкта розділить свої посилання на змінні екземпляра з вихідним об'єктом:



Object >> shallowCopy примітивний метод, який створює неповну копію об'єкта. Так як a2 тільки неповна копія a1, два масиви використовують посилання на вкладений масив, який вони містять.

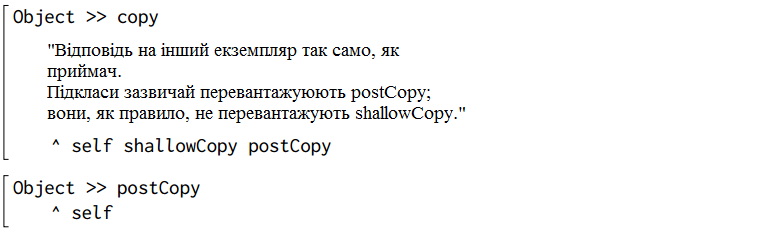
Object >> DeepCopy робить довільно глибоку копію об'єкта.



Проблема з DeepCopy в тому, що він не завершиться, коли застосовується взаємно рекурсивна структура:



Альтернативним рішенням є використання копії повідомлення. Він реалізований у Object в такий спосіб:



**Налагодження**

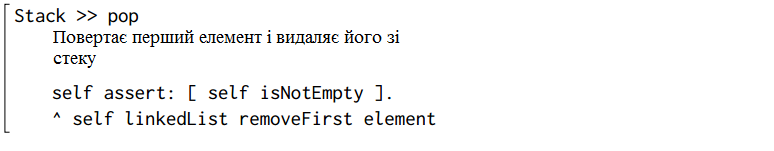
**halt.** Найбільш важливий метод тут є halt. Для того, щоб встановити точку зупинки в методі, потрібно просто вставити вираз send self half в певний момент в тіло методу. Коли це повідомлення відправлено, виконання буде перервано, і відладчик відкриє даний момент у вашій програмі (дивіться главу :. Середовище Pharo для отримання більш докладної інформації про відладчик).

Ви можете також використовувати Halt один раз або Halt if: aCondition. Подивіться на клас Halt, який є спеціальним винятком.

**assert:.** Наступне найбільш важливе повідомлення це assert:, який очікує блок в якості аргументу. Якщо блок істинний, виконання триває. В іншому випадку виключення AssertionFailure буде викинуто. Якщо цей виняток ще не спійманий, відладчик буде відкритий в цей момент у виконанні.

assert: особливо корисний для підтримки проектування за контрактом. Найтиповішим використанням є перевірка нетривіальних попередніх умов для публічних методів об'єктів.

Stack>>pop можна було легко реалізованати наступним чином:



Не плутайте Object >> Assert: з TestCase >> assert:, яке відбувається в рамкf[ тестування SUnit (див главу: SUnit). Поки перший очікує блок - як аргумент (насправді, він буде приймати будь-якої аргумент, який розуміє значення, в тому числі Boolean), останній розраховує логічне значення. Хоча обидва корисні для налагодження, кожен з них служить зовсім іншою метою.

**Обробка помилок**

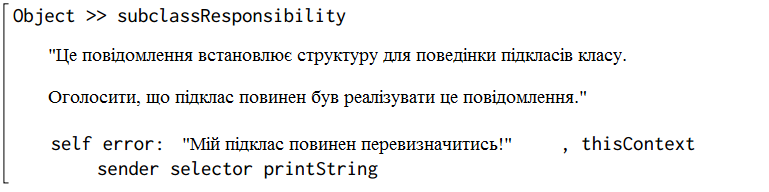
Цей протокол містить кілька способів, які корисні для виявлення помилок під час виконання.

**deprecated:.** Надсилаючи self deprecated: сигналізує, що даний метод більше не повинен використовуватися, якщо deprecated: включено. Ви можете включити / вимкнути його в розділі “Налагодження” за допомогою налаштувань браузера. Аргумент повинен описати альтернативу. Подивіться на відправник повідомлень deprecated: щоб отримати уявлення.

**doesNotUnderstand**:. doesNotUnderstand: (зазвичай скорочено в обговоренні як DNU або MNU) відправляється щоразу, коли пошук повідомлення невдалий. Реалізація за замовчуванням, тобто Object >> doesNotUnderstand: запустить відладчик в поточний момент. Це може бути корисно для перевизначення doesNotUnderstand: щоб забезпечити деяку іншу поведінку.

**error**. Object >> error і Object >> error: це загальні методи, які можуть бути використані для генерації виключень. (Як правило, краще створювати свої власні виключення, щоб можна було відрізняти помилки, що виникають з вашого коду, від помилок, що виникають з класів ядра.)

**subclassResponsibility**. Абстрактні методи реалізуються за угодою з тілом self subclassResponsibility. Якщо абстрактний клас створюється випадково, то виклики абстрактних методів приведуть до виконання Object >> subclassResponsibility.



Magnitude, Number і Boolean є класичними прикладами абстрактних класів, які ми незабаром побачимо в цьому розділі.



**shouldNotImplement**. Self shouldNotImplement відправляється за угодою, щоб сигналізувати, що успадкований метод не підходить для цього підкласу. Це, як правило, ознака того, що щось не зовсім правильно з дизайном ієрархії класів. Через обмеження одиночного наслідування, однак, іноді дуже важко уникнути таких обхідних рішень.

Типовим прикладом є Collection >> remove: який успадковується Dictionary, але позначений як нереалізований. (Словник забезпечує removeKey: замість цього).

**Тестування**

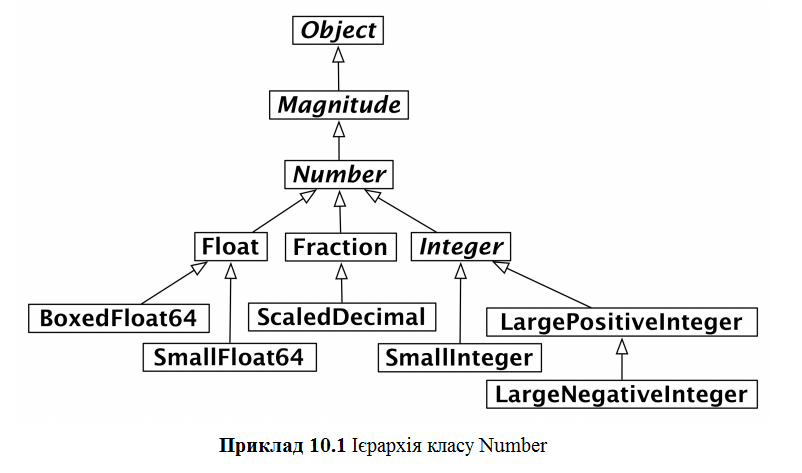
Методи тестування не мають нічого спільного з тестуванням SUnit! Метод тестування - це метод, який дозволяє задати питання про стан одержувача і повертає логічне значення.

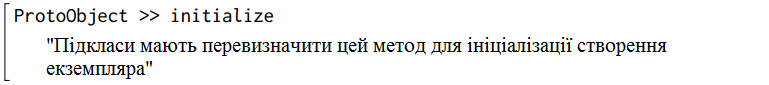
Численні методи тестування надаються Object. Є isArray, isBoolean, isBlock, isCollection і так далі. Як правило, такі методи слід уникати, оскільки запит об'єкта на його клас є формою порушення інкапсуляції. Замість того, щоб тестувати об'єкт для свого класу, потрібно просто відправити запит і дозволити об'єкту вирішити, як з ним поводитися.

Проте деякі з цих методів тестування, безсумнівно, корисні. Найбільш корисними, ймовірно, є ProtoObject >> isNil і Object >> notNil (хоча шаблон проектування Null Object може позбавити від необхідності використання навіть цих методів).

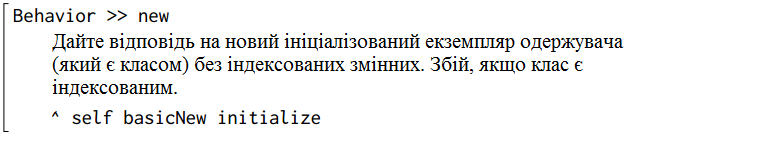
**Ініціалізація**

Ініціалізує остаточний ключовий метод, який відбувається не в Object, а в ProtoObject.





Важлива причина цього в тому, що в Pharo новий метод за замовчуванням, визначений для кожного класу в системі, буде відправляти ініціалізацію в новостворені екземпляри.



Це означає, що просто перевизначаючи метод hook ініціалізації, нові екземпляри вашого класу будуть автоматично ініціалізовані. Метод initialize повинен зазвичай виконувати супер ініціалізацію, щоб встановити інваріант класу для будь-яких успадкованих змінних екземпляра.

**10.2 Numbers**

Числа в Pharo не є примітивними значеннями даних, а є істинними об'єктами. Звичайно, числа ефективно реалізовані на віртуальній машині, але ієрархія Number так само доступна і розширювана, як і будь-яка інша частина ієрархії класів.

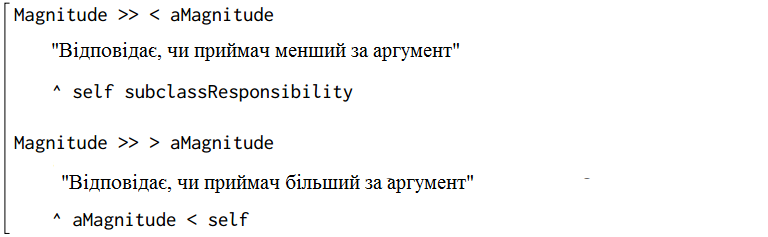
Абстрактним коренем цієї ієрархії є Magnitude, яка представляє всі види класів, що підтримують оператори порівняння. Number додає різні арифметичні і інші оператори в основному абстрактні методи. Float і Fraction представляють, відповідно, числа з плаваючою комою і дробові значення. Підкласи Float (BoxedFloat64 і SmallFloat64) представляють Float на певних архітектурах.

Наприклад, BoxedFloat64 доступний тільки для 64-бітових систем. Ціле число також є абстрактним, таким чином, розрізняючи підкласи SmallInteger, LargePositiveInteger і LargeNegativeInteger. Здебільшого користувачам не потрібно знати різницю між трьома класами Integer, оскільки значення автоматично перетворюються по міру необхідності.

**Magnitude**

Масштабування є батьком не тільки класів Number, але також інших класів, що підтримують операції порівняння, таких як Character, Duration і Timespan.

Методи < і = є абстрактними. Решта операторів визначені в загальному випадку. Наприклад:



**Number**

Аналогічно, Number визначає +, -, \* і / як абстрактні, але всі інші арифметичні оператори визначені в загальному.

Всі об'єкти Number підтримують різні оператори перетворення, такі як asFloat і asInteger. Існують також численні методи конструктора ярликів, які генерують тривалості, такі як годину, день і тиждень.

Числа безпосередньо підтримують загальні математичні функції, такі як sin, log, raiseTo:, squared, sqrt і так далі.

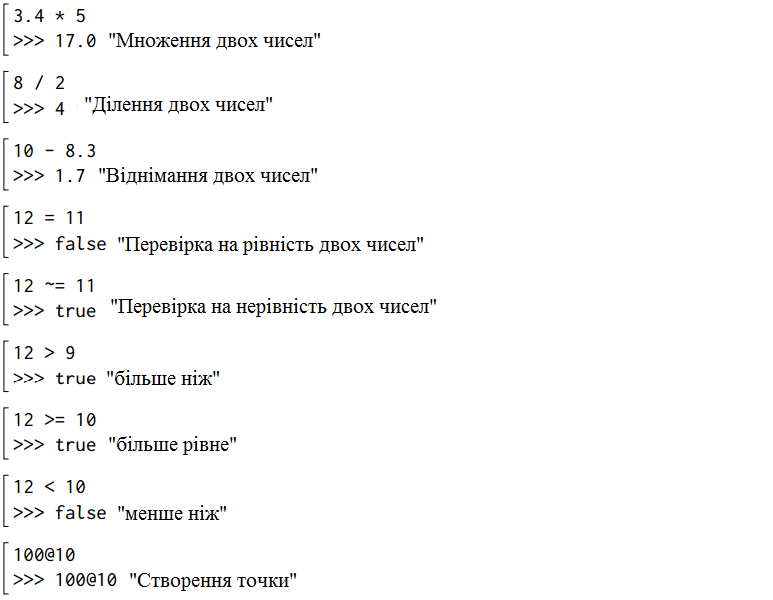
Метод Number >> printOn: реалізований в термінах абстрактного методу Number >> printOn: base:. (За замовчуванням використовується значення 10.)

Методи тестування включають в себе парні, непарні, позитивні і негативні.

Number перевизнача isNumber. Більш цікаво, isInfinite визначено для повернення false.

Методи усічення включають підлогу, стелю, integerPart, fractionPart і т. д.





Наступний приклад працює на диво добре в Pharo:



Зверніть увагу, щоб факторіал 1000 дійсно обчислюється, що в багатьох інших мовах може бути досить складно обчислити. Це прекрасний приклад автоматичного примусу і точна обробка числа.

**Спробуйте зробити.** Показати результат 1000 факторіала. Потрібно більше часу для відображення, ніж для його обчислення!

**Float**

Float реалізує абстрактні методи Number для чисел з плаваючою комою.

Цікавіше те, що клас Float (тобто сторона класу Float) надає методи для повернення наступних констант: е, infinity, nan та pi.



**Fraction**

Фракції представлені змінними екземпляру для чисельника і знаменника, які повинні бути цілими числами. Фракції зазвичай створюються за допомогою цілочисельного ділення (замість використання методу конструктора Fraction >> чисельник: знаменник:):



Множення фракції на ціле число або інший дріб може дати ціле число:

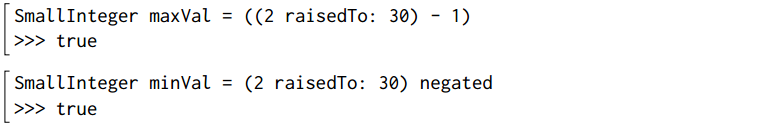


**Integer**

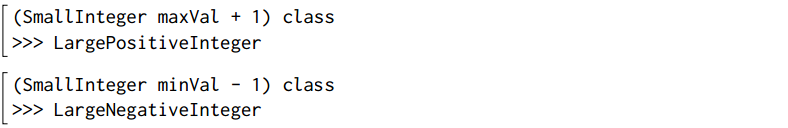
Integer є абстрактним батьком трьох конкретних цілочисельних реалізацій. Крім надання конкретних реалізацій багатьох абстрактних методів класу Number, він також додає кілька методів, специфічних для цілих чисел, таких як factorial, atRandom, IsPrime, НСД: і багато інших.

SmallInteger є спеціальним в тому, що його екземпляри представлені компактно - замість того, щоб зберігатися як посилання, SmallInteger представляється безпосередньо, використовуючи біти, які в іншому випадку використовувалися б для зберігання посилання. Перший біт посилання на об'єкт вказує, чи є об'єкт SmallInteger чи ні. Тепер віртуальна машина абстрагує це від вас, тому ви не можете бачити це безпосередньо під час перевірки об'єкта.

Методи класу minVal і maxVal говорять нам про діапазон SmallInteger:

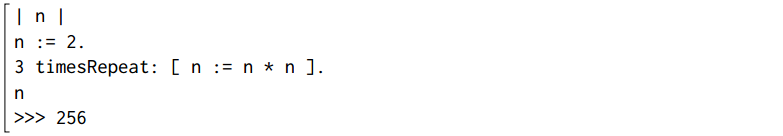


Коли SmallInteger виходить з цього діапазону, він автоматично перетворюється в LargePositiveInteger або LargeNegativeInteger, якщо необхідно:



Великі цілі числа при необхідності перетворюються назад в маленькі цілі числах.

Як і в більшості мовах програмування, цілі числа можуть бути корисні для вказівки ітеративної поведінки. Існує спеціальний метод timesRepeat: для повторної оцінки блоку. Ми вже бачили аналогічний приклад в розділі: Синтаксис в двох словах.



**Characters**

Character визначається підкласом Magnitude. Друковані символи представлені в Pharo як $ <символ>. наприклад:



Недруковані символи можуть бути згенеровані різними методами класу. Character class>>value: приймає цілочисельне значення Unicode (або ASCII) в якості аргументу і повертає відповідний символ. Протокол, який одержує доступ до недрукованих символів, містить кілька методів конструктора зручності, як backspace, cr, escape, euro, space, tab і т. д.



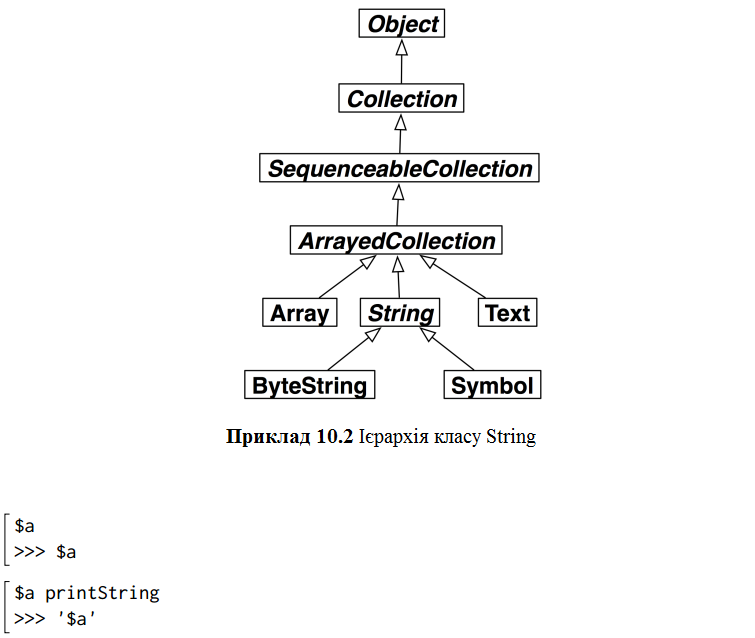
Метод printOn: досить розумний, щоб знати, який з трьох способів генерації символів пропонує найбільш підходяще представлення:



Вбудовуються різні зручні методи тестування: isAlphaNumeric, isCharacter, isDigit, isLowercase, isVowel і т. д.

Щоб перетворити символ в рядок, що містить тільки цей символ, відправте AsString. В цьому випадку AsString і printString дають різні результати:





Як і SmallInteger, символ є безпосереднім значенням, а не посиланням на об'єкт.

У більшості випадків ви не побачите ніякої різниці і можете використовувати об'єкти класу символів, як і будь-які інші. Але це означає, що символи з рівним значенням завжди однакові:



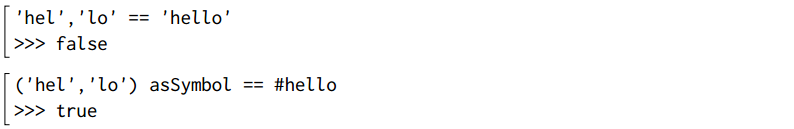
**Strings**

String - це індексована колекція, яка містить тільки символи.

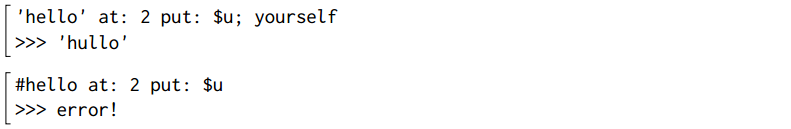
Фактично, String є абстрактним, а рядки Pharo є екземплярами конкретного класу ByteString.



Іншим важливим підкласом String є Symbol. Головна відмінність в тому, що існує тільки один екземпляр Symbol з заданим значенням. (Це іноді називають унікальну властивість екземпляра). Навпаки, два окремо побудовані рядки, які містять одну і ту ж послідовність символів, часто будуть різними об'єктами.



Інша важлива відмінність полягає в тому, що String є змінним, а Symbol - незмінним.



Легко забути, що, оскільки рядки є колекціями, вони розуміють ті ж повідомлення, що і інші колекції:



Хоча String не успадковується від Magnitude, він підтримує звичайні методи порівняння <, = і так далі. Крім того, рядок >> match: корисний для деяких базових шаблонів зіставлення шаблонів в стилі glob:



Регулярні вирази будуть обговорюватися більш докладно в розділі: Регулярні вирази в Pharo.

String підтримує досить велику кількість методів перетворення. Багато з них є методами конструктора скороченого виклику для інших класів, такі як asDate, asInteger і т. д. Є також ряд корисних методів для перетворення рядка в інший рядок, наприклад, з великої літери і translateToLowercase.

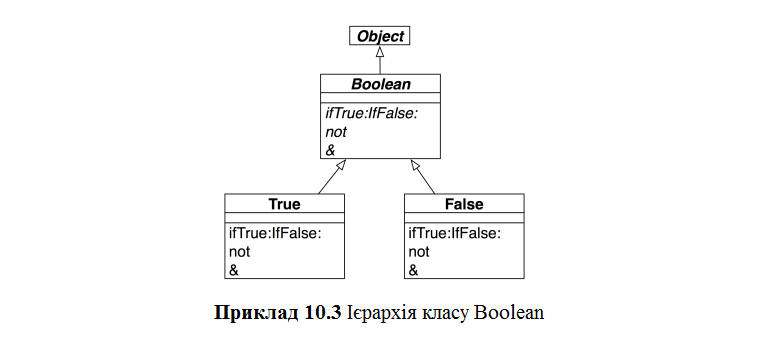
Додаткові відомості про рядках і колекціях см. У розділі: Колекції

**Boleans**

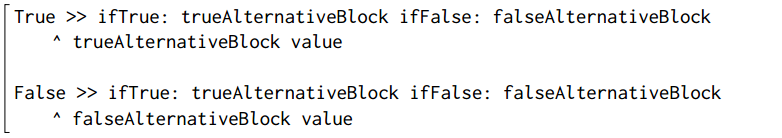
Клас Boolean пропонує захоплююче уявлення про те, скільки мовою Pharo було вміщено в бібліотеці класів. Boolean - це абстрактний суперклас одноелементних класів True і False.

Велика частина поведінки булевих змінних можна зрозуміти розглянувши метод ifTrue: ifFalse :, який приймає два блоки в якості аргументів.



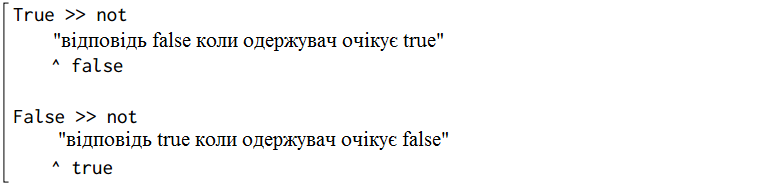


Метод ifTrue: ifFalse: є абстрактним в класі Boolean. Реалізації в його конкретних підкласах тривіальні:

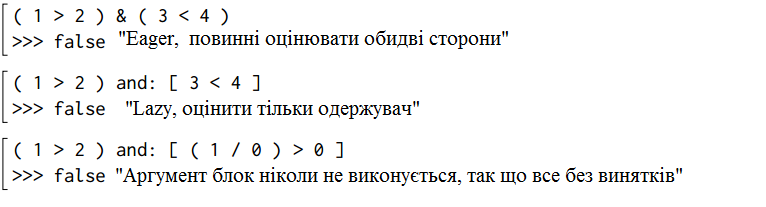


Кожен з них виконує правильний блок в залежності від одержувача повідомлення. Фактично, це сутність ООП: коли повідомлення відправляється об'єкту, сам об'єкт визначає, який метод буде використовуватися для відповіді.

В цьому випадку екземпляр True просто виконує справжню альтернативу, в той час як екземпляр False виконує помилкову альтернативу. Всі абстрактні логічні методи реалізовані таким чином для True і False. Наприклад, реалізація заперечення (повідомлення not) визначається таким же чином:



Boolean пропонує кілька корисних методів зручності, такий як ifTrue :, ifFalse :, і ifFalse: ifTrue. У вас також є вибір між eager і lazy кон'юнкції і диз'юнкції.



У першому прикладі виконуються обидва булевих подвирази, оскільки & приймає логічний аргумент. У другому і третьому прикладах виконується тільки перший, так як і очікується блок в якості аргументу. Блок виконується, тільки якщо перший аргумент має значення true.

**Спробуйте зробити**. Cпробуйте уявити, як and: і or: реалізовані. Перевірте реалізації в Boolean, True і False.

**Висновки глави**

• Якщо ви перевантажили =, тоді ви також повинні перевантажили хеш.

• Перевизначайте postCopy для правильної реалізації копіювання для ваших об'єктів.

• Використовуйте self halt. Для установки точки зупинки.

• Повертайте self subclassResponsibility для створення абстрактного методу.

• Щоб дати об'єкту представлення String, ви повинні перевизначити printOn:.

• Перевизначайте метод hook методу для правильної ініціалізації екземплярів.

• Number методи автоматично конвертуються між Float, Fractions і Integer.

• Фракції дійсно представляють раціональні числа, а не float.

• Всі символи подібні унікальним екземплярам.

• Strings змінювані; Symbols - ні. Однак, не мутуйте рядкові літерали!

• Symbols унікальні; Strings - ні.

• Рядки і символи є колекціями і тому підтримують звичайні методи Collection.