Розділ 9

Розуміння синтаксису повідомлень

Синтаксис повідомлень Pharo надзвичайно простий, проте він нетрадиційний і може бути потрібно трохи часу, щоб звикнути до нього. У цьому розділі наведено деякі рекомендації, які допоможуть вам освоїтися з синтаксисом надсилання повідомлень. Якщо ви вже не відчуваєте труднощів з синтаксисом, то можете пропустити цей розділ, або повернутися до нього пізніше. Синтаксис Pharo близький до синтаксису Smalltalk, тому програмісти Smalltalk можуть знати синтаксис Pharo.

9.1. Розпізнавання повідомлень

У Pharo все є надсиланням повідомлень за винятком окремих синтаксичних елементів, описаних у попередньому розділі (*:= ^ . ; #() {} [:|]*). Ви можете визначити оператори для власних класів, наприклад +, але всі оператори, наявні та визначені вами, мають однаковий пріоритет. Насправді в Pharo немає операторів! Є просто повідомлення певного виду: *унарні*, *бінарні* або *ключові*. Крім того, ви не можете змінити арність селектора повідомлення. Селектор «–» завжди є селектором бінарного повідомлення, неможливо оголосити унарний мінус як унарне повідомлення з селектором «–».

Порядок надсилання повідомлень у Pharo визначено видом повідомлення. Є лише три види повідомлень: *унарні*, *бінарні* та *ключові*. Спочатку завжди надсилаються унарні повідомлення, потім – бінарні і, нарешті, ключові. Як і в більшості мов програмування, круглі дужки використовують для зміни порядку виконання. Ці правила роблять код Pharo легким для читання, і зазвичай вам не потрібно думати про правила.

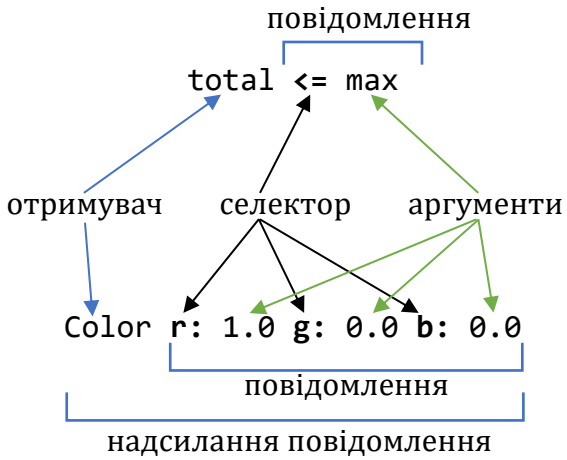

Рис. 9.1. Два приклади надсилання повідомлення, що складаються з отримувача, селектора методу та набору аргументів

Оскільки більшість обчислень у Pharo виконується через надсилання повідомлень, то правильне розпізнавання повідомлень має вирішальне значення. Нам допоможе така термінологія.

- Повідомлення складається з *селектора* та, можливо, аргументів.
- Повідомлення надсилають *отримувачу*.
- Отримувач і повідомлення до нього разом називають *надсилання повідомлення*, як показано на рис. 9.1.

Повідомлення завжди надсилається отримувачу, який може бути окремим літералом, блоком, змінною або результатом виконання іншого повідомлення. На схематичному зображенні підкреслимо отримувача повідомлення, щоб допомогти ідентифікувати його. А також обведемо кожне надсилання повідомлення еліпсом і пронумеруємо їх, починаючи від одиниці, щоб продемонструвати послідовність, у якій надсилаються повідомлення.

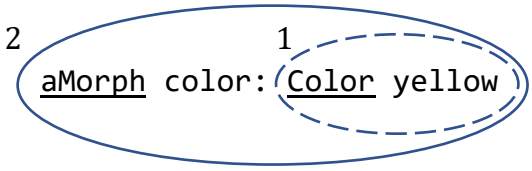

Рис. 9.2. Два надсилання повідомлень: «*Color yellow*» і «*aMorph color: Color yellow*»

На рис. 9.2 зображено два надсилання повідомлень: «*Color yellow*» і «*aMorph color: Color yellow*», тому є два еліпси. Спочатку надсилається повідомлення «*Color yellow*», тому його еліпс має номер 1. Є два отримувачі: *aMorph* отримує повідомлення *color:...*, і *Color* отримує *yellow*. Обидва отримувачі підкреслено.

Отримувачем може бути перший елемент виразу, як *100* у надсиланні повідомлення «*100 + 200*» або *Color* у «*Color yellow*». Проте отримувач може бути також результатом виконання іншого повідомлення. Наприклад, у виразі «*Pen new go: 100*» отримувач повідомлення «*go: 100*» – об'єкт, отриманий у результаті надсилання повідомлення «*Pen new*». Повідомлення завжди надсилається об'єкту, що називається отримувачем, який може бути результатом надсилання іншого повідомлення.

Надсилання повідомлення	**Вид повідомлення**	**Результат**
`Color yellow`	Унарне	Створює жовтий колір
`aPen go: 100`	Ключове	Переміщає ручку на 100 пікселів уперед
`100 + 20`	Бінарне	Додає 20 до 100
`Browser open`	Унарне	Відкриває нового оглядача
`Pen new go: 100`	Унарне та ключове	Створює нову ручку і переміщає її на 100 пікселів уперед
`aPen go: 100 + 20`	Ключове та бінарне	Обчислює 120 і переміщає ручку на 120 пікселів уперед

У таблиці наведено кілька прикладів надсилання повідомлень. Ви мали б помітити, що:

- не у всіх повідомлень є аргументи. Унарні повідомлення, як *open*, їх не мають;
- бінарні повідомлення (*+ 20*) та ключові з одним ключем (*go: 100*) мають по одному аргументу;

- є прості повідомлення та складені. Повідомлення «*Color yellow*» і «*100 + 20*» – прості: одне повідомлення надсилають одному об'єктові. Вираз «*aPen go: 100 + 20*» складається з двох повідомлень: повідомлення *go:* надсилають об'єктові *aPen* з аргументом, який є результатом виконання повідомлення *+ 20*, надісланого об'єктові *100*;
- отримувач може бути виразом (присвоєнням, надсиланням повідомлення або літералом), що повертає об'єкт. У виразі «*Pen new go: 100*» повідомлення *go: 100* надсилають об'єктові, отриманому внаслідок надсилання повідомлення *Pen new*.

9.2. Три види повідомлень

У Pharo визначено кілька простих правил, які задають порядок надсилання повідомлень. Ці правила опираються на відмінності трьох різних видів повідомлень.

- *Унарні повідомлення* – це такі повідомлення, які надсилають до об'єкта без ніякої додаткової інформації. Наприклад, у виразі *15 factorial* повідомлення *factorial* – унарне. Надсилання унарного повідомлення може виконати і базову унарну операцію, і довільний функціонал. Як би там не було, його завжди надсилають без аргументів.
- *Бінарні повідомлення* – це повідомлення, що складаються з операторів (часто арифметичних) і виконують базові бінарні операції. Їх називають бінарними, оскільки вони завжди залучають два об'єкти: отримувача та єдиний аргумент. Наприклад, у виразі «*100 + 20*» *+* бінарне повідомлення, надіслане з аргументом *20* отримувачу *100*.
- *Ключові повідомлення* – це повідомлення, які складаються з одного або кількох ключових слів, кожне з яких закінчується двокрапкою (:) і приймає один аргумент. Наприклад, у виразі «*anArray at: 1 put: 10*» селектор повідомлення – *at:put:*. Ключове слово *at:* приймає аргумент *1*, а *put:* – аргумент *10*.

Важливо зазначити, що:

- не буває надсилання ключових повідомлень без аргументів. Усі повідомлення без аргументів – унарні;
- ключові повідомлення, що мають тільки один аргумент, відрізняються від бінарних повідомлень двокрапкою, яку використовують для вказання кожного аргументу ключового повідомлення.

Унарні повідомлення

Унарні повідомлення – це повідомлення, які не потребують жодних аргументів. Вони відповідають синтаксичному шаблону: *отримувач ім'я_повідомлення*. Селектор складається просто з послідовності літер без двокрапки (:), наприклад, *factorial*, *open*, *class*.

```
89 sin
>>> 0.860069405812453
```

```
3 sqrt
>>> 1.732050807568877
```

```
Float pi
>>> 3.141592653589793
```

```
345 negated
>>> -345
```

```
'blop' size
>>> 4
```

```
true not
>>> false
```

```
Object class
>>> Object class "The class of Object is Object class (BANG)"
```

Важливо Унарні повідомлення відповідають синтаксичному шаблону *отримувач* ***селектор***.

Бінарні повідомлення

Бінарні повідомлення – це повідомлення, які потребують точно один аргумент і чий селектор складається з послідовності одної або кількох літер з набору +, –, *, /, &, =, >, |, <, ~ й @. Запам'ятайте, що селектор –– заборонено через особливості розпізнавання.

```
100@100 >>> 100@100
"creates a Point object"
```

```
3 + 4
>>> 7
```

```
10 – 1
>>> 9
```

```
4 <= 3
>>> false
```

```
(4/3) * 3 == 4
>>> true "equality is just a binary message, and Fractions are exact"
```

```
(3/4) == (3/4)
>>> false "two equal Fractions are not the same object"
```

Важливо Бінарні повідомлення відповідають синтаксичному шаблону *отримувач* ***селектор*** *аргумент*.

Ключові повідомлення

Ключові повідомлення – це повідомлення, що потребують одного чи більше аргументів, і чий селектор складається з одного або кількох ключових слів, кожне з яких закінчується двокрапкою (:).

У прикладі нижче повідомлення складається з двох ключових слів: *between:* і *and:*. Повний селектор повідомлення – *between:and:*, повідомлення надсилають числу *2*.

```
2 between: 0 and: 10
>>> true
```

Кожне ключове слово приймає аргумент. Так «*r:g:b:*» – повідомлення з трьома аргументами, «*at:put:*» – повідомлення з двома аргументами, а «*playFileNamed:*» і «*at:*» – повідомлення з одним аргументом кожне. Щоб створити екземпляр класу *Color*, можна використати повідомлення «*r:g:b:*», як у прикладі нижче. Не забувайте, що двокрапки є частиною селектора.

```
Color r: 1 g: 0 b: 0
>>> Color red "creates a new color"
```

У Java-подібному синтаксисі надсиланню повідомлення «*Color r: 1 g: 0 b: 0*» відповідатиме виклик методу «*Color.rgb(1, 0, 0)*».

```
1 to: 10
>>> (1 to: 10) "creates an interval"
```

```
| nums |
nums := Array newFrom: (1 to: 5).
nums at: 1 put: 6.
nums
>>> #(6 2 3 4 5)
```

Важливо Ключові повідомлення відповідають синтаксичному шаблону *отримувач* ***селекторСловоОдин:*** *аргумент1* ***словоДва:*** *аргумент2 …* ***словоN:*** *аргументN*.

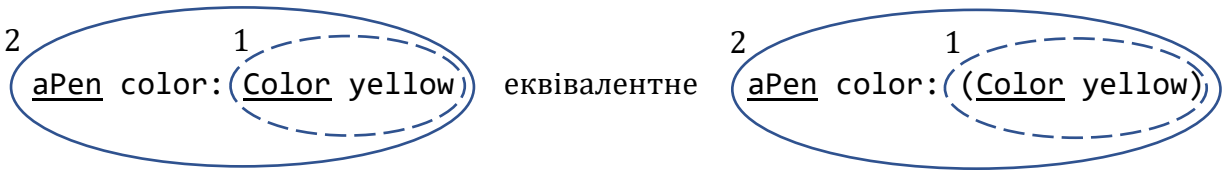

Рис. 9.3. Спочатку надсилаються унарні повідомлення, тому «*Color yellow*» буде першим. Воно поверне об’єкт, який стане аргументом для «*aPen color: Color yellow*»

9.3. Композиція повідомлень

Повідомлення кожного з трьох видів мають інший пріоритет, що дає змогу елегантно комбінувати їх.

- Унарні повідомлення завжди надсилаються першими, тоді – бінарні, і насамкінець – ключові.
- Повідомлення в круглих дужках мають найвищий пріоритет.
- Повідомлення однакового пріоритету опрацьовуються зліва направо.

Ці правила формують дуже природний порядок прочитання. Але, якщо ви хочете бути впевненими, що ваші повідомлення надсилаються в потрібному порядку, то завжди

можете поставити більше дужок, як показано на рис. 9.3. На цій схемі повідомлення *yellow* унарне, а *color:* ключове, тому першим надсилається *Color yellow*. Оскільки повідомлення, надіслані в дужках, надсилаються першими, то взяття *Color yellow* у (непотрібні) дужки лише підкреслює, що воно буде надіслано першим. Далі в розділі проілюстровано кожен із цих моментів.

Унарні > Бінарні > Ключові

Унарні повідомлення надсилаються першими, тоді – бінарні, і насамкінець – ключові. Ми також можемо сказати, що унарні повідомлення мають вищий пріоритет ніж інші.

Важливо *Унарні* > *Бінарні* > *Ключові*

Як видно з прикладів, правила синтаксису Pharo загалом гарантують, що вирази програми можна читати природно, як звичайний текст.

```
1000 factorial / 999 factorial
>>> 1000
```

```
2 raisedTo: 1 + 3 factorial
>>> 128
```

На жаль, правила занадто спрощені для надсилання арифметичних повідомлень, тому доводиться записувати дужки, щоб задати черговість виконання бінарних операторів.

```
1 + 2 * 3
>>> 9
```

```
1 + (2 * 3)
>>> 7
```

Арифметичним невідповідностям присвячено окремий параграф.

Наступний приклад трохи складніший (!), проте він добре ілюструє те, що навіть такі вирази можна читати природно.

```
[:aClass | aClass methodDict keys select: [:aMethod |
    (aClass >> aMethod) isAbstract ]] value: Boolean
>>> #(#ifTrue: #| #xor: #asBit #ifFalse:ifTrue: #ifFalse:
      #ifTrue:ifFalse: #or: #& #and: #not)
```

Тут ми хочемо довідатися, які методи класу *Boolean* абстрактні. Запитуємо якийсь клас, аргумент *aClass* блока, про ключі його словника методів і вибираємо ті методи, які є абстрактними. Потім ми прив'язуємо аргумент *aClass* до конкретного значення *Boolean*. Круглі дужки потрібні лише для того, щоб надіслати бінарне повідомлення >>, яке вибирає метод із класу, перед тим, як надіслати цьому методу унарне повідомлення *isAbstract*. З результату зрозуміло, які методи потрібно реалізувати в конкретних підкласах *True* та *False* класу *Boolean*.

Насправді, ми могли б написати простіший еквівалентний вираз «*Boolean methodDict select: [:each | each isAbstract] thenCollect: [:each | each selector].*»

Приклад. У виразі *aPen color: Color yellow* одне унарне повідомлення *yellow* до класу *Color* і одне ключове *color:* до екземпляра *aPen*. Спочатку надсилають унарні повідом-

лення, тому *Color yellow* надсилається першим. Унаслідок його виконання повернеться екземпляр, назвемо його *aColor*, який стане аргументом повідомлення *aPen color: aColor*. На рис. 9.3 графічно зображено почерговість надсилання повідомлень.

```
"Декомпозиція виконання виразу aPen color: Color yellow"
   aPen color: Color yellow
(1)  Color yellow       "спочатку надсилаються унарні повідомлення"
>>> aColor
(2)  aPen color: aColor "потім - ключові"
```

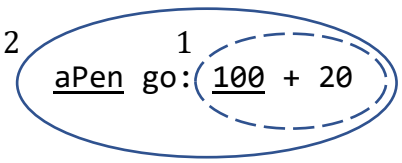

Рис. 9.4. Бінарні повідомлення надсилаються перед ключовими

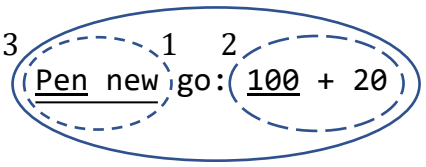

Рис. 9.5. Декомпозиція виразу *Pen new go: 100 + 20*

Приклад. У виразі *aPen go: 100 + 20* два повідомлення: *бінарне* «*+ 20*» і *ключове* «*go:…*». Бінарні повідомлення мають вищий пріоритет ніж ключові, тому спочатку виконається надсилання *100 + 20*: об'єкт *100* отримає повідомлення *+ 20*, повернеться результат – об'єкт *120*. Потім об'єктові *aPen* буде надіслано повідомлення *go: 120*. Послідовність обчислення виразу графічно зображена на рис. 9.4 і текстом програми у коді нижче.

```
"Декомпозиція обчислення виразу aPen go: 100 + 20"
   aPen go: 100 + 20
(1)  100 + 20     "спочатку надсилають бінарні повідомлення"
>>> 120
(2)  aPen go: 120 "потім - ключові"
```

Приклад. Виконайте як вправу декомпозицію виконання виразу *Pen new go: 100 + 2*, складеного з унарного, ключового та бінарного повідомлень (див. рис. 9.5).

Спочатку дужки

Повідомлення в дужках мають вищий пріоритет ніж усі інші.

Важливо (*Вираз*) > *Унарні* > *Бінарні* > *Ключові*

Наведемо кілька прикладів.

З першого прикладу бачимо, що можна обійтися без дужок, якщо порядок обчислення вже такий, як нам треба, тобто, результат обчислення виразу з дужками буде такий самий, як без дужок. Далі обчислюємо тангенс числа *1.5*, заокруглюємо його та перетворюємо на рядок.

```
1.5 tan rounded asString = (((1.5 tan) rounded) asString)
>>> true
```

Другий приклад демонструє, що обчислення факторіала має вищий пріоритет ніж додавання. Якщо потрібно спочатку додати *3* та *4*, то бінарне повідомлення *+* треба взяти в дужки.

```
3 + 4 factorial
>>> 27 "(не 5040)"
```

```
(3 + 4) factorial
>>> 5040
```

Подібно в наступному прикладі дужки потрібні, щоб надсилання повідомлення *lowMajorScaleOn:* відбулося перед *play*.

```
(FMSound lowMajorScaleOn: FMSound clarinet) play
"(1)  надсилання clarinet класові FMSound, щоб створити звук кларнета
 (2)  використання цього звуку в ключовому повідомленні lowMajorScaleOn:
       класові FMSound
 (3)  відтворення отриманого звуку повідомленням play"
```

Приклад. Результат обчислення виразу *(65@325 extent: 134@100) center* – центр прямокутника з лівою верхньою вершиною в точці (65; 325) і розмірами 134×100. Нижче описано декомпозицію виразу, що демонструє порядок надсилання повідомлень у ньому. Спочатку надсилаються повідомлення в дужках: два бінарні повідомлення *@325* і *@100* надсилаються числам *65* і *134*, відповідно, та повертають точки. Перша з них отримує ключове повідомлення *extent:* з аргументом другою точкою, що повертає прямокутник. Насамкінець унарне повідомлення *center* надсилається створеному прямокутнику і повертає точку – його центр. Обчислення виразу без дужок спровокувало б помилку, бо найвищий пріоритет мало б унарне повідомлення *center*, а об'єкт *100* його не розуміє.

```
"Приклад використання дужок"
   (65@325 extent: 134@100) center
(1)  65@325 "бінарне"
>>> aPoint
(2)  134@100 "бінарне"
>>> anotherPoint
(3)  aPoint extent: anotherPoint "ключове"
>>> aRectangle
(4)  aRectangle center "унарне"
>>> 132@375
```

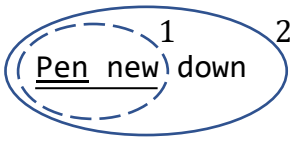

Рис. 9.6. Декомпозиція виразу *Pen new down*

Зліва направо

Тепер ми знаємо, як опрацьовуються повідомлення різних видів або пріоритетів. Залишилося з'ясувати, як надсилаються повідомлення з однаковим пріоритетом. Їх відправляють зліва направо. Зауважте, що ви вже бачили таку поведінку у прикладі *1.5 tan rounded asString*, де всі унарні повідомлення надсилаються зліва направо, що еквівалентно *(((1.5 tan) rounded) asString)*.

Важливо Порядок надсилання повідомлень однакового виду – зліва направо.

Приклад. У виразі *Pen new down* усі повідомлення унарні, тому першим буде надіслане перше зліва: спочатку виконається *Pen new*. Воно поверне нову ручку, яка отримає повідомлення *down* (див. рис. 9.6).

Арифметичні невідповідності

Правила композиції повідомлень прості. Немає поняття математичного пріоритету, тому що арифметичні повідомлення – просто бінарні повідомлення, як і будь-які інші. Тож результат їхнього виконання може видатися неправильним. Далі опишемо типові ситуації, коли потрібні додаткові дужки.

```
3 + 4 * 5
>>> 35 "(не 23) Бінарні повідомлення надсилаються зліва направо"
```

```
3 + (4 * 5)
>>> 23
```

```
1 + 1/3
>>> (2/3) "а не 4/3"
```

```
1 + (1/3)
>>> (4/3)
```

```
1/3 + 2/3
>>> (7/9) "а не 1"
```

```
(1/3) + (2/3)
>>> 1
```

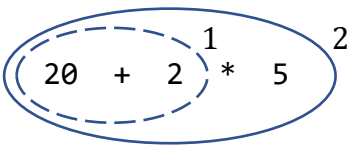

Рис. 9.7. Звичайний порядок виконання

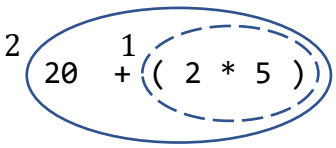

Рис. 9.8. Зміна порядку виконання за допомогою дужок

Приклад. У виразі *20 + 2 * 5* тільки бінарні повідомлення *+* і ***. А у Pharo немає окремих пріоритетів для операторів *+* і ***. Це просто бінарні повідомлення, тому *** не має вищого пріоритету ніж *+*. Першим буде надіслане перше зліва повідомлення *+ 2* об'єктові *20*, потім результат додавання отримає повідомлення ** 5*. Порядок виконання виразу показано на рис. 9.7 і в коді нижче.

```
"Усі бінарні повідомлення мають однаковий пріоритет, тому перше зліва
 повідомлення + 2 буде виконано першим, незважаючи на правила арифметики,
 згідно з якими першим мало б бути * 5."

     20 + 2 * 5
(1)  20 + 2 >>> 22
(2)      22 * 5 >>> 110
```

Як видно з попереднього прикладу, результатом обчислення виразу є 110, а не 30. Це виглядає, мабуть, несподівано, але випливає з правил надсилання повідомлень. Це ціна, яку доводиться платити за простоту моделі. Щоб отримати правильний результат, потрібно використати дужки. Повідомлення в дужках надсилається першим, тому вираз *20 + (2 * 5)* поверне правильний результат, як зображено в коді нижче та на рис. 9.8.

```
"Повідомлення в дужках надсилаються першими, тому * 5 буде надіслано
 перед + (), що дасть правильний результат."
     20 + (2 * 5)
(1)        2 * 5 >>> 10
(2)  20 + 10 >>> 30
```

Важливо Арифметичні оператори у Pharo мають однаковий пріоритет. *+* і *** просто бінарні повідомлення, тому *** не має більшого пріоритету ніж *+*. Використовуйте дужки, щоб отримати очікуваний результат.

Неявний пріоритет (за правилами)	Пріоритет явно заданий дужками
`aPen color: Color yellow`	`aPen color: (Color yellow)`
`aPen go: 100 + 20`	`aPen go: (100 + 20)`
`aPen penSize: aPen penSize + 2`	`aPen penSize: ((aPen penSize) + 2)`
`20 factorial + 4`	`(20 factorial) + 4`

Правила пріоритетності унарних, бінарних і ключових повідомлень у багатьох випадках дають змогу не використовувати дужки. У таблиці в лівому стовпці показані складені повідомлення, записані з урахуванням правил пріоритетності, а в правому – відповідні їм повідомлення, якби таких правил не було. Обидва варіанти надсилання повідомлень діють однаково, або повертають однаковий результат.

9.4. Як розпізнати ключове повідомлення

У початківців часто виникають труднощі з розумінням того, коли потрібно використовувати дужки. Давайте розглянемо, як компілятор розпізнає ключові повідомлення.

Ставити дужки чи ні?

Літери [,], (і) обмежують різні ділянки. У межах такої ділянки ключове повідомлення є найдовшою послідовністю слів, що закінчуються двокрапкою, не перерваною літерами крапка, кома, або крапка з комою.

У наступному прикладі два різних ключових повідомлення: *rotatedBy:magnify:smoothing:* і *at:put:*.

```
aDict
   at: (rotatingForm rotateBy: angle magnify: 2 smoothing: 1)
   put: 3
```

Підказка. Якщо ви відчуваєте труднощі з застосуванням правил пріоритетності, то можете ставити дужки щоразу, коли хочете відокремити два повідомлення з однаковим пріоритетом.

У фраґменті коду нижче дужки не потрібні, бо унарне повідомлення *isNil* має вищий пріоритет ніж ключове повідомлення *ifTrue: [ ]*.

```
(x isNil)
   ifTrue: [ ... ]
```

У наступному фраґменті дужки потрібні обов'язково, бо обидва повідомлення: і *includes:*, і *ifTrue:* – ключові.

```
ord := OrderedCollection new.
(ord includes: $a)
   ifTrue: [ ... ]
```

Якби тут не було дужок, колекція *ord* отримала б невідоме повідомлення *includes:ifTrue:*.

Коли використовувати [ ], а коли ( ) ?

Також можуть виникати труднощі з розумінням того, коли використовувати квадратні дужки, а коли круглі. Головним критерієм є те, скільки разів ви збираєтеся обчислювати вираз. Квадратні дужки перетворять вираз на блокове замикання, яке можна виконати довільну кількість разів, або не виконати жодного, залежно від контексту. Тому квадратні дужки застосовують тоді, коли наперед *невідомо*, скільки разів буде обчислено вираз. Нагадаємо, що вираз може бути надсиланням повідомлення, змінною, літералом, присвоєнням або блоком.

Невідомо, який з аргументів повідомлення *ifTrue:ifFalse:* буде виконано, бо це залежить від результату обчислення умови. Тому його аргументами є блоки. З таких самих міркувань отримувач і аргумент повідомлення *whileTrue:* потребують квадратних дужок. Адже не відомо, скільки разів буде обчислено чи отримувач, чи аргумент.

На противагу цьому, круглі дужки впливають тільки на порядок надсилання повідомлень. Тому кожного разу під час обчислення виразу *(expression)* обчислення *expression* відбудеться точно *один* раз.

```
"І отримувач, і аргумент мусять бути блоками"
[ x isReady ] whileTrue: [ y doSomething ]

"Аргумент буде обчислено кілька разів, тому він мусить бути блоком"
4 timesRepeat: [ Beeper beep ]

"Отримувача обчислюють один раз, тому він не блок.
 Аргумент може не бути обчислений ні разу, тому він блок"
(x isReady) ifTrue: [ y doSomething ]
```

9.5. Послідовність повідомлень

Вирази (наприклад, надсилання повідомлень, присвоєння тощо), відокремлені крапками, виконуються послідовно. Зверніть увагу, між оголошенням локальних змінних і наступним виразом крапка не потрібна. Значенням послідовності виразів є значення, отримане внаслідок обчислення її останнього виразу. Результати обчислення всіх інших виразів послідовності ігноруються. Зазначимо, що крапка є розділювачем, а не термінальним символом, тому після останнього виразу послідовності її можна не ставити.

```
| box |
box := 20@30 corner: 60@90.
box containsPoint: 40@50
>>> true
```

9.6. Каскад повідомлень

Pharo пропонує спосіб надсилання кількох повідомлень тому самому отримувачу без потреби зазначати його щоразу: достатньо відокремити повідомлення крапкою з комою (;). На жаргоні Pharo це називається каскадом.

Схематично синтаксис каскаду можна зобразити так.

```
aReceiverExpression msg1; msg2; msg3
```

Приклади. У Pharo можна програмувати без каскадів. Тоді доведеться зазначати отримувача кожного повідомлення. Наступні два фрагменти коду діють однаково.

```
Transcript show: 'Pharo is '.
Transcript show: 'fun'.
Transcript cr.
```

```
Transcript
   show: 'Pharo is ';
   show: 'fun';
   cr
```

Насправді одержувачем усіх повідомлень каскаду є одержувач першого повідомлення, залученого в каскад. Зверніть увагу, що об'єкт, який отримує каскадні повідомлення, сам може бути результатом надсилання повідомлення. У наступному прикладі *setX:setY:* перше повідомлення каскаду, бо за ним стоїть крапка з комою. Одержувач каскадного повідомлення – це щойно створена точка в результаті виконання *Point new*, а не *Point*. Наступне повідомлення *isZero* надсилається тому самому отримувачу – результатові виконання *Point new*.

```
Point new setX: 25 setY: 35; isZero
>>> false
```

9.7. Підсумки розділу

- Повідомлення завжди надсилають об'єктові. Його називають отримувачем. Він може бути результатом надсилання іншого повідомлення.
- Унарні повідомлення – це повідомлення без аргументів. Вони мають вигляд *receiver **selector***.
- Бінарні повідомлення залучають два об'єкти: отримувача й аргумент. Їхній селектор складається з однієї або більше літери з-поміж +, –, *, /, |, &, =, >, <, ~, @. Вони мають вигляд *receiver **selector** argument*.
- Ключові повідомлення залучають більше ніж один об'єкт і містять у селекторі хоча б одну двокрапку. Вони мають вигляд *receiver **selectorKeywordOne**: argumentOne **keywordTwo**: argumentTwo ... **keywordN**: argumentN*.

- **Правило Один.** Спочатку надсилаються унарні повідомлення, потім – бінарні, а насамкінець – ключові.
- **Правило Два.** Повідомлення в круглих дужках надсилаються перед усіма іншими.
- **Правило Три.** Послідовність надсилання повідомлень одного виду – зліва направо.
- Звичайні арифметичні оператори, як + і *, у Pharo мають однаковий пріоритет. І +, і * є бінарними повідомленнями, тому * не має вищого пріоритету ніж +. Щоб отримати правильний результат у арифметичних обчисленнях, потрібно використовувати круглі дужки.
- Комбінацію літер «––» заборонено використовувати як селектор бінарного повідомлення.