Розділ 17
Класи і метакласи

У Pharo все є об'єктами, а кожний об'єкт – екземпляр класу. Самі класи теж не становлять виняток – усі класи є об'єктами, причому об'єкти-класи є екземплярами інших класів. Така об'єктна модель компактна, проста, елегантна та однорідна. Вона повністю охоплює суть об'єктно-орієнтованого програмування. Проте новачка наслідки такої побудови можуть збити з пантелику.

Зауважимо, що для того, щоб вільно програмувати на Pharo, не обов'язково повністю розуміти всі тонкощі побудови об'єктної моделі Pharo і наслідки її однорідності, та все ж. Мета цього розділу двояка: (1) заглибитись, на скільки вдасться, у суть питання і (2) з'ясувати, що тут немає ніякої магії чи особливих складнощів: лише прості правила, однаково застосовані у всіх випадках. Послуговуючись цими правилами, можна завжди розуміти, чому ситуація саме така, яка вона є.

17.1. Правила для класів

Об'єктна модель Pharo ґрунтується на обмеженій кількості концепцій, які застосовують завжди однаково. Пригадаємо правила об'єктної моделі, розглянуті в розділі 10 «Об'єктна модель Pharo».

Правило 1. Кожна сутність є об'єктом.

Правило 2. Кожен об'єкт є екземпляром якогось класу.

Правило 3. У кожного класу є надклас.

Правило 4. Усе відбувається через надсилання повідомлень.

Правило 5. Алгоритм відшукання методу перебирає ланцюжок наслідування.

Правило 6. Класи також є об'єктами і діють за тими самими правилами.

З Правила 1 випливає, що *класи також є об'єктами*, а з Правила 2 – що класи теж мусять бути екземплярами класів. Клас класу називають *метакласом*.

17.2. Метакласи

Метаклас створюється автоматично кожного разу, коли створюється клас. У переважній більшості випадків не потрібно турбуватися чи навіть думати про метакласи. Проте кожного разу, коли ви використовуєте Системний оглядач для пошуку інформації про клас на *стороні класу*, не зайвим буде пригадати, що насправді ви переглядаєте інший клас. Клас і метаклас є окремими класами, попри те, що перший є екземпляром другого. Справді, окрема точка відрізняється від класу *Point*, і так само клас *Point* відрізняється від свого метакласу.

Щоб належно пояснити класи і метакласи, доповнимо правила з розділу 10 «Об'єктна модель Pharo» додатковими.

Правило 7. Кожен клас є екземпляром свого метакласу.

Правило 8. Ієрархія метакласів паралельна ієрархії класів.

Правило 9. Кожен клас наслідує від класів *Class* і *Behavior*.

Правило 10. Кожен метаклас є екземпляром класу *Metaclass*.

Правило 11. Метаклас класу *Metaclass* є екземпляром класу *Metaclass*.

Разом ці одинадцять простих правил становлять об'єктну модель Pharo.

Спершу на невеликому прикладі коротко повторимо п'ять перших правил. Після цього детальніше розглянемо нові правила, використовуючи той самий приклад.

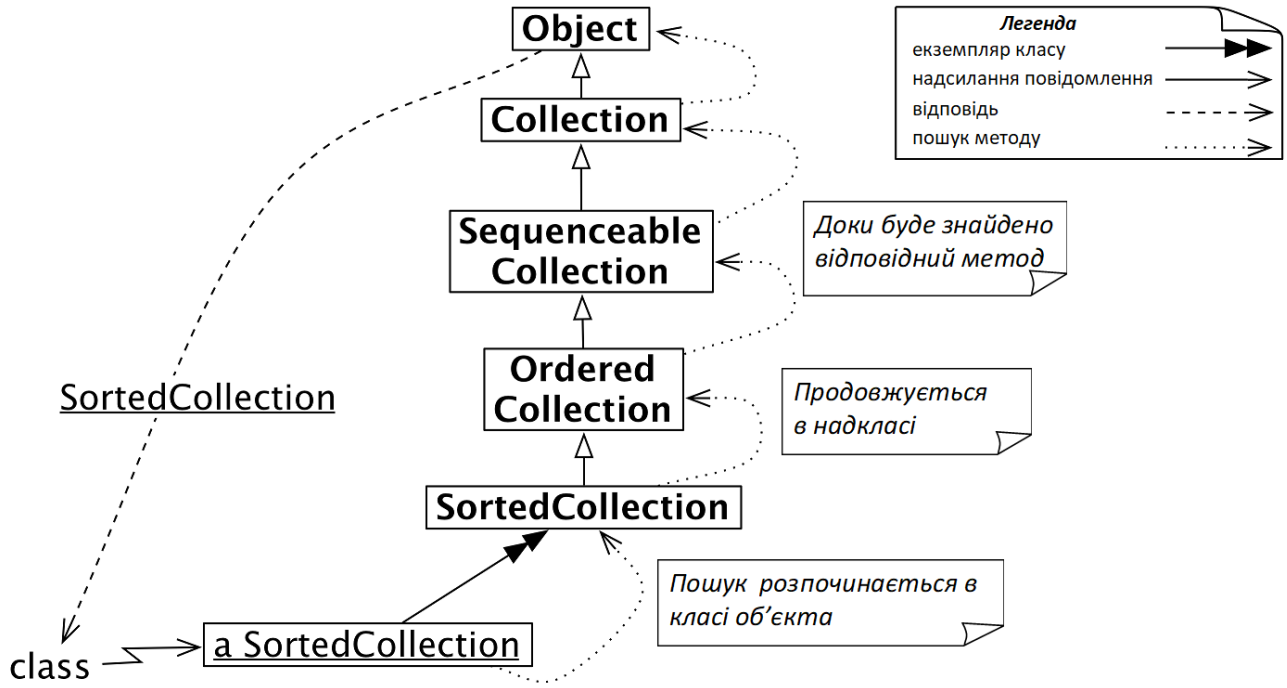

Рис. 17.1. Надсилання повідомлення *class* до впорядкованої колекції

17.3. Ще раз про об'єктну модель Pharo

Правило 1. Оскільки будь-що є об'єктом, то послідовна колекція у Pharo також є об'єктом.

```
OrderedCollection withAll: #(4 5 6 1 2 3)
>>> an OrderedCollection(4 5 6 1 2 3)
```

Правило 2. Кожен об'єкт є екземпляром класу. Послідовна колекція є екземпляром класу *OrderedCollection*.

```
(OrderedCollection withAll: #(4 5 6 1 2 3)) class
>>> OrderedCollection
```

Правило 3. У кожного класу є надклас. Надкласом класу *OrderedCollection* є *SequenceableCollection*, а надкласом *SequenceableCollection* – *Collection*.

```
OrderedCollection superclass
>>> SequenceableCollection
```

```
SequenceableCollection superclass
>>> Collection

Collection superclass
>>> Object
```

Правило 4. Усе відбувається через надсилання повідомлень, звідки випливає, що *withAll:* – повідомлення, надіслане до *OrderedCollection*, *class* – повідомлення екземпляра послідовної колекції, а *superclass* – повідомлення до класів *OrderedCollection*, *SequenceableCollection* і *Collection*. У кожному випадку отримувачем є об'єкт, бо будь-що є об'єктом, але деякі з них є також класами.

Правило 5. Пошук методу відбувається за ланцюжком наслідування, тому коли надсилають повідомлення *class* результатові обчислення виразу «*(OrderedCollection withAll: #(4 5 6 1 2 3)) asSortedCollection*», воно буде опрацьовано, коли відповідний метод буде знайдено в класі *Object*, як показано на рис.17.1.

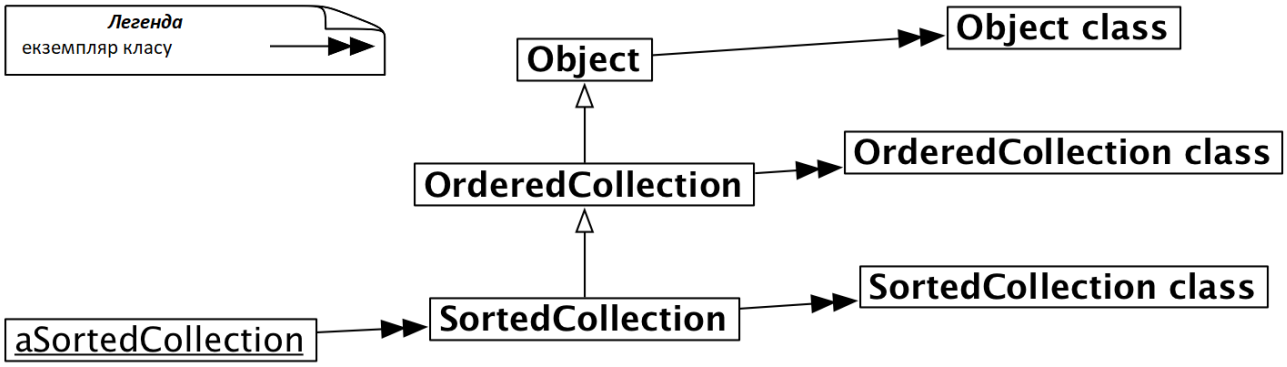

Рис. 17.2. Метакласи та надкласи (вибрані) *SortedCollection*

17.4. Кожен клас є екземпляром свого метакласу

Як вже згадували у параграфі 17.2, класи, чиї екземпляри самі є класами, називають *метакласами*. Інший термін використовують, щоб бути певним у точному розумінні, про що мовиться: про клас *Point*, чи про клас класу *Point*.

Метакласи є неявними

Метакласи створюються автоматично під час визначення класу. Кажуть, що вони *неявні*, оскільки програміст ніколи не повинен хвилюватися про них. Неявний метаклас створюється для *кожного* створеного класу, тому кожен метаклас має єдиний екземпляр.

На відміну від звичайних класів, кожного з яких потрібно назвати, метакласи залишаються анонімними. Але на анонімний метаклас завжди можна послатись через клас, що є його екземпляром. Наприклад, класом класу *SortedCollection* є *SortedCollection class*, а класом класу *Object* є *Object class*.

```
SortedCollection class
>>> SortedCollection class

Object class
```

```
>>> Object class
```

Насправді метакласи не повністю анонімні, їхнє ім'я випливає з їхнього єдиного екземпляра.

```
SortedCollection class name
>>> 'SortedCollection class'
```

```
Object class name
>>> 'Object class'
```

З рис. 17.2 видно, що кожен клас є екземпляром свого метакласу. Зазначимо, що на рисунку пропущено класи *SequenceableCollection* і *Collection* тільки для економії місця. Те, що їх не зображено, загалом не змінює суті.

17.5. Запити до метакласів

Той факт, що класи також є об'єктами, дає змогу легко звертатися до них за допомогою надсилання повідомлень.

```
OrderedCollection subclasses
>>> {ObjectFinalizerCollection. SortedCollection. WeakOrderedCollection.
     OCLiteralList. GLMMultiValue. RSGroup}
```

```
SortedCollection subclasses
>>> #()
```

```
SortedCollection allSuperclasses
>>> an OrderedCollection(OrderedCollection SequenceableCollection
     Collection Object ProtoObject)
```

```
SortedCollection instVarNames
>>> #(#sortBlock)
```

```
SortedCollection allInstVarNames
>>> #(#array #firstIndex #lastIndex #sortBlock)
```

```
SortedCollection selectors asSortedCollection
>>> a SortedCollection(#, #= #add: #addAll: #addFirst: #at:put: #collect:
     #copyEmpty #defaultSort:to: #flatCollect: #fromSton: #groupedBy:
     #indexForInserting: #insert:before: #intersection: #join: #median
     #reSort #sort: #sort:to: #sortBlock #sortBlock: #stonOn:)
```

17.6. Ієрархія метакласів паралельна ієрархії класів

Правило 8 стверджує, що надкласом метакласу не може бути звичайний клас: він змушений бути метакласом надкласу єдиного екземпляра цього метакласу. Метаклас класу *SortedCollection* наслідує метаклас *OrderedCollection* (надкласу *SortedCollection*).

```
SortedCollection class superclass
>>> OrderedCollection class
SortedCollection superclass class
>>> OrderedCollection class
```

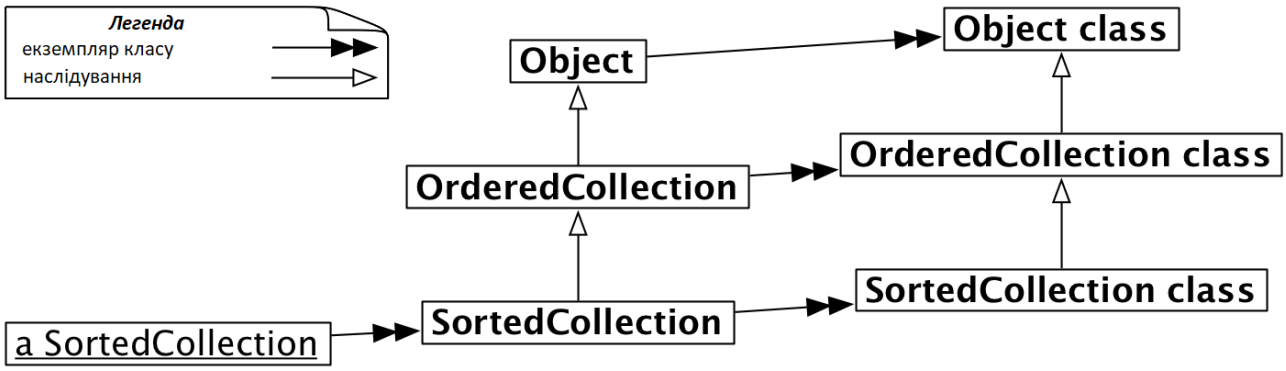

Рис. 17.3. Ієрархія метакласів (вибраних)паралельна ієрархії класів

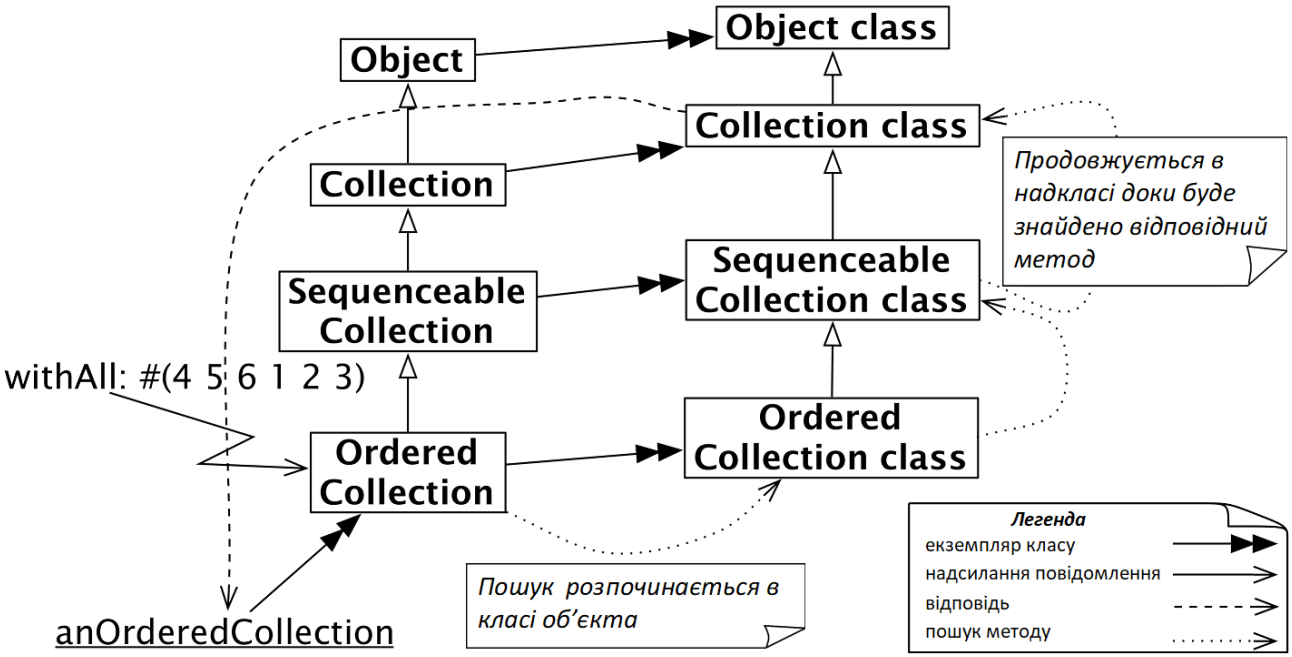

Рис. 17.4. Алгоритм пошуку методів класів такий самий, як методів звичайних об'єктів

Це те, що означає твердження «ієрархія метакласів паралельна ієрархії класів». На рис. 17.3 продемонстровано це на прикладі ієрархії *SortedCollection*.

```
SortedCollection class
>>> SortedCollection class
```

```
SortedCollection class superclass
>>> OrderedCollection class
```

```
SortedCollection class superclass superclass
>>> SequenceableCollection class
```

```
SortedCollection class superclass superclass superclass superclass
>>> Object class
```

17.7. Однорідність класів і об’єктів

Цікаво зупинитися на мить, щоб оглянути загальну картину, й усвідомити, що немає різниці між надсиланням повідомлення об’єкту та класу. В обох випадках пошук відповідного методу *розпочинається в класі отримувача* і *продовжується вгору по ланцюжку наслідування*.

Тому повідомлення, які надсилають класам, прямують ланцюжком наслідування метакласів. Розглянемо, наприклад, метод *withAll:*, реалізований на стороні класу *Collection*. Якщо надіслати повідомлення *withAll:* класові *OrderedCollection*, то пошук методу відбудеться так само, як і для будь-якого іншого повідомлення. Він розпочнеться з класу *OrderedCollection class* (пошук починається з класу отримувача, а отримувачем є *OrderedCollection*) і рухатиметься вгору по ієрархії метакласів, доки метод не буде знайдено у класі *Collection class* (див. рис. 17.4). Метод поверне новий екземпляр класу *OrderedCollection*.

```
OrderedCollection withAll: #(4 5 6 1 2 3)
>>> an OrderedCollection (4 5 6 1 2 3)
```

Єдиний спосіб пошуку методів

У Pharo існує єдиний спосіб пошуку методів. Класи є лише об’єктами і поводяться як інші об’єкти. Класи можуть створювати нові екземпляри лише завдяки тому, що вміють відповідати на повідомлення *new*, а метод опрацювання *new* вміє створювати нові екземпляри.

Зазвичай об’єкти «не класи» не розуміють цього повідомлення, однак ніщо не заважає додати метод *new* у «не метаклас», якщо на це є вагома причина.

17.8. Інспектування об’єктів і класів

Оскільки класи є об’єктами, то їх також можна інспектувати.

Проінспектуйте два об’єкти: *OrderedCollection withAll: #(4 5 6 1 2 3)* і *OrderedCollection*.

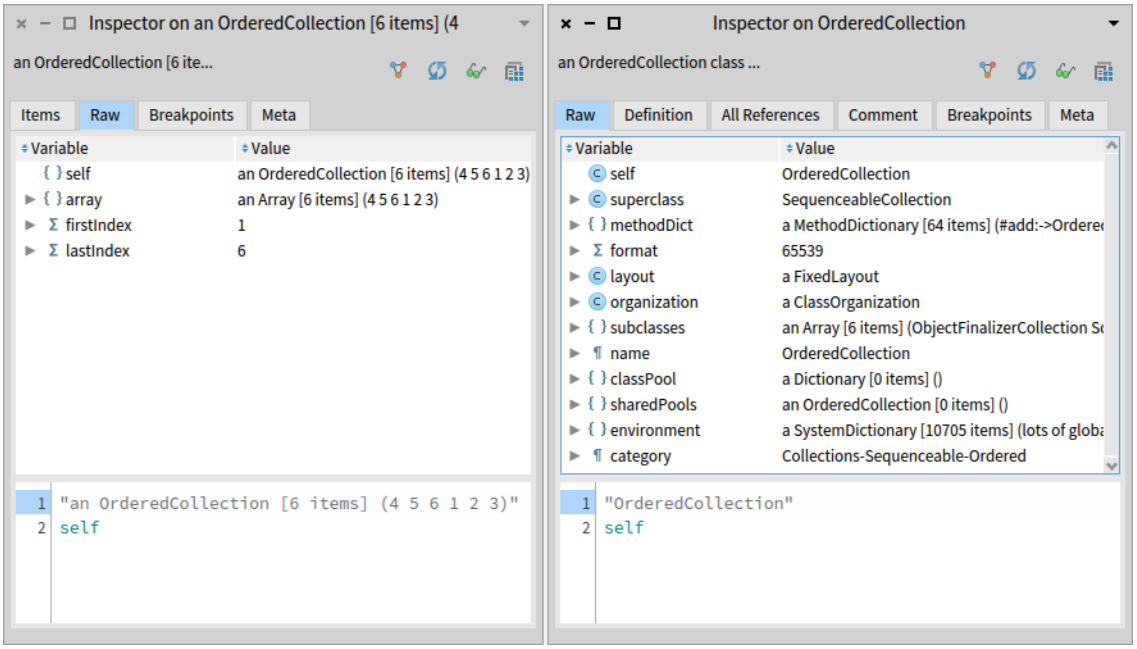

Рис. 17.5. Класи також є об’єктами

Зауважимо, що в першому випадку маємо справу з *екземпляром* класу *OrderedCollection*, а в другому – з самим класом *OrderedCollection*. Різницю можна не помітити відразу, бо заголовок інспектора містить ім’я *класу* інспектованого об’єкта, але у випадку екземпляра наявний також артикль «*an*» або «*a*».

Інспектор з класом *OrderedCollection* дає змогу побачити надклас, змінні екземпляра, словник методів тощо метакласу *OrderedCollection*, як показано на рис. 17.5.

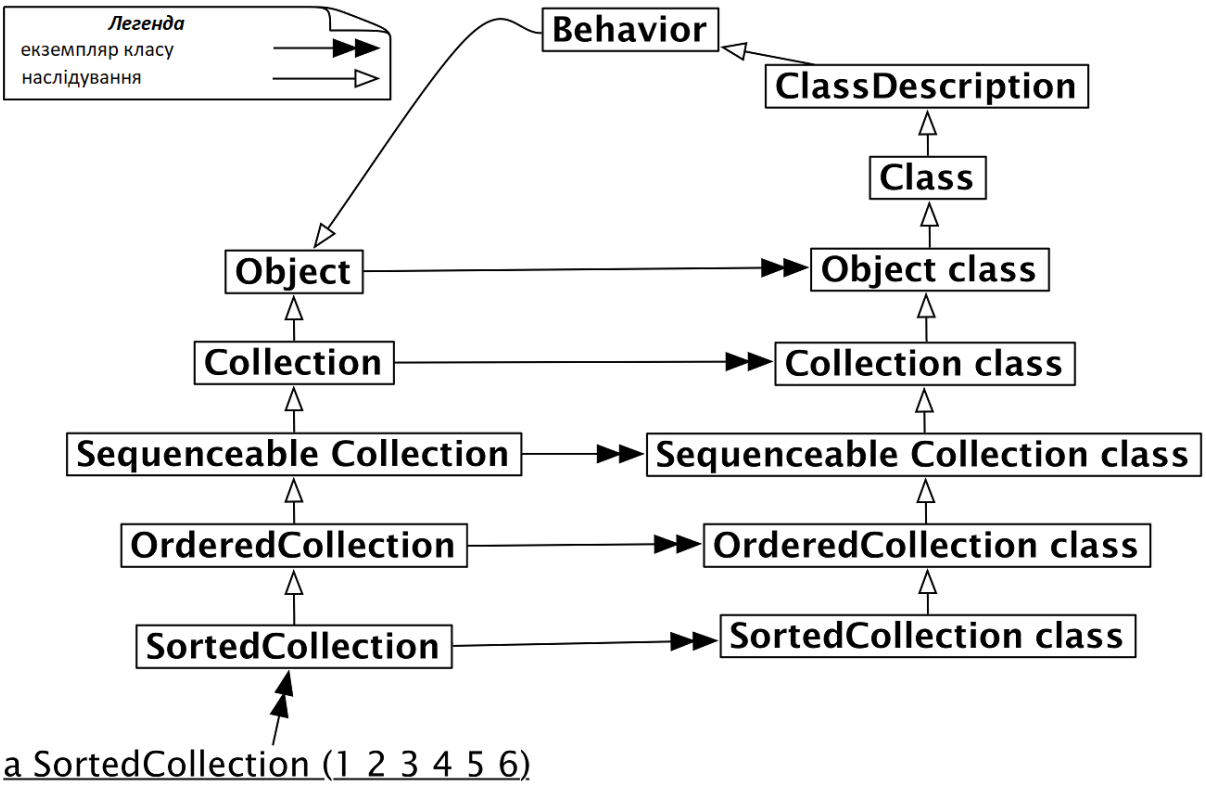

Рис. 17.6. Метакласи наслідують *Class* і *Behavior*

17.9. Кожен метаклас наслідує класи *Class* і *Behavior*

Кожен метаклас є різновидом класу (таким, що має єдиний екземпляр), тому наслідує клас *Class*. Так само *Class* наслідує свої надкласи *ClassDescription* і *Behavior*. Оскільки будь-що у Pharo є об'єктом, то всі ці класи у підсумку наслідують клас Object. Повну картину можна бачити на рис. 17.6.

Де визначено *new*?

Щоб зрозуміти важливість того факту, що метакласи наслідують *Class* і *Behavior*, достатньо подумати, де визначено *new*, і як його знаходить алгоритм пошуку методів.

Коли класові надсилають повідомлення *new*, відповідний метод шукається у ланцюжку метакласів і, зрештою, у їхніх надкласах *Class*, *ClassDescription* і *Behavior*, як зображено на рис. 17.7.

Коли повідомлення *new* надсилають класові *SortedCollection*, пошук розпочинається в метакласі *SortedCollection class* і просувається догори ланцюжком наслідування. Нагадуємо, що алгоритм пошуку такий самий, як для будь-якого об'єкта.

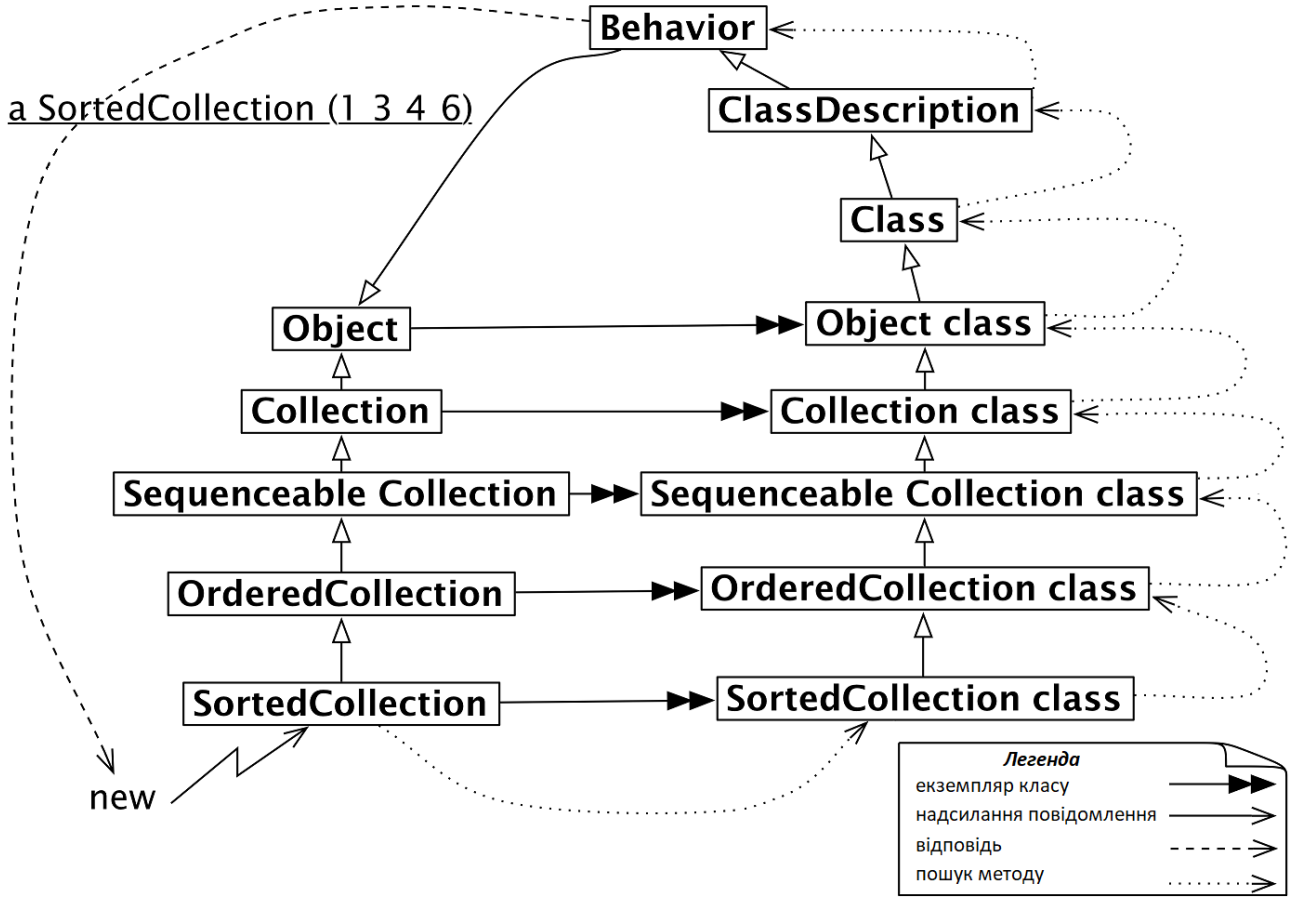

Рис. 17.7. *new* – звичайне повідомлення, відповідний метод шукається в ланцюжку метакласів

Питання *«Де визначено new?»* має вирішальне значення. Вперше *new* визначено в класі *Behavior* і може бути перевизначено у підкласах, включно з довільними метакласами визначених класів, якщо є така потреба.

Тоді, коли повідомлення *new* надіслано класові, пошук стартує, як годиться, з метакласу цього класу і продовжується по ланцюжку надкласів до класу *Behavior*, якщо метод не було перевизначено десь на цьому шляху.

Зазначимо, що результат надсилання *SortedCollection new* є екземпляром *SortedCollection*, а *не Behavior*, хоча метод і міститься в класі *Behavior*!

Метод *new* завжди повертає екземпляр *self*, тобто класу, який отримав повідомлення, навіть якщо воно реалізоване в іншому класі.

```
SortedCollection new class
>>> SortedCollection "не Behavior!"
```

Поширена помилка

Початківці часто допускають таку помилку: шукають *new* у надкласі отримувача. Те саме стосується і *new:*, стандартного повідомлення для створення об'єкта заданого розміру. Наприклад, *Array new: 4* створює масив з 4 елементів. Ви не знайдете цього методу ні в класі *Array*, ні в жодному з його надкласів. Замість цього вам потрібно шукати його в класі *Array class* і його надкласах, оскільки пошук розпочинається саме з нього (див. рис. 17.7).

Методи *new* і *new:* визначені в метакласах, бо їх виконують у відповідь на повідомлення, надіслане класові.

Додамо також, що клас є об'єктом, тому може бути отримувачем повідомлень, методи для яких визначені в класі *Object*. Коли класові *Point* надсилають повідомлення *class* або *error:*, алгоритм пошуку приведе ланцюжком наслідування метакласів (через *Point class*, *Object class*, …) аж до *Object*.

17.10. Призначення класів *Behavior*, *ClassDescription* і *Class*

Behavior

Behavior підтримує мінімальний набір змінних, необхідний для об'єктів, які мають екземпляри; сюди входять: посилання на надклас, словник методів і формат класу. Формат класу – це ціле число, що позначає відмінності у способі виділення пам'яті для екземплярів: вказівник чи не вказівник, компактне розміщення чи розгорнуте, а також базовий розмір екземплярів. Клас *Behavior* наслідує *Object*, тому і він, і всі його підкласи можуть поводитись як об'єкти.

Behavior також є базовим інтерфейсом для компілятора. Він містить методи, потрібні для створення словника методів, компілювання методів, створення екземплярів (наприклад, *new*, *basicNew*, *new*:, *basicNew*:), управління ієрархією класів (наприклад, *superclass:*, *addSubclass:*), доступу до методів (наприклад, *selectors*, *allSelectors*, *compiledMethodAt:*), доступу до екземплярів і змінних (наприклад, *allInstances*, *instVarNames*, …), доступу до ієрархії класів (наприклад, *superclass*, *subclasses*) і запитів (наприклад, *hasMethods*, *includesSelector:*, *canUnderstand*:, *inheritsFrom*:, *isVariable*).

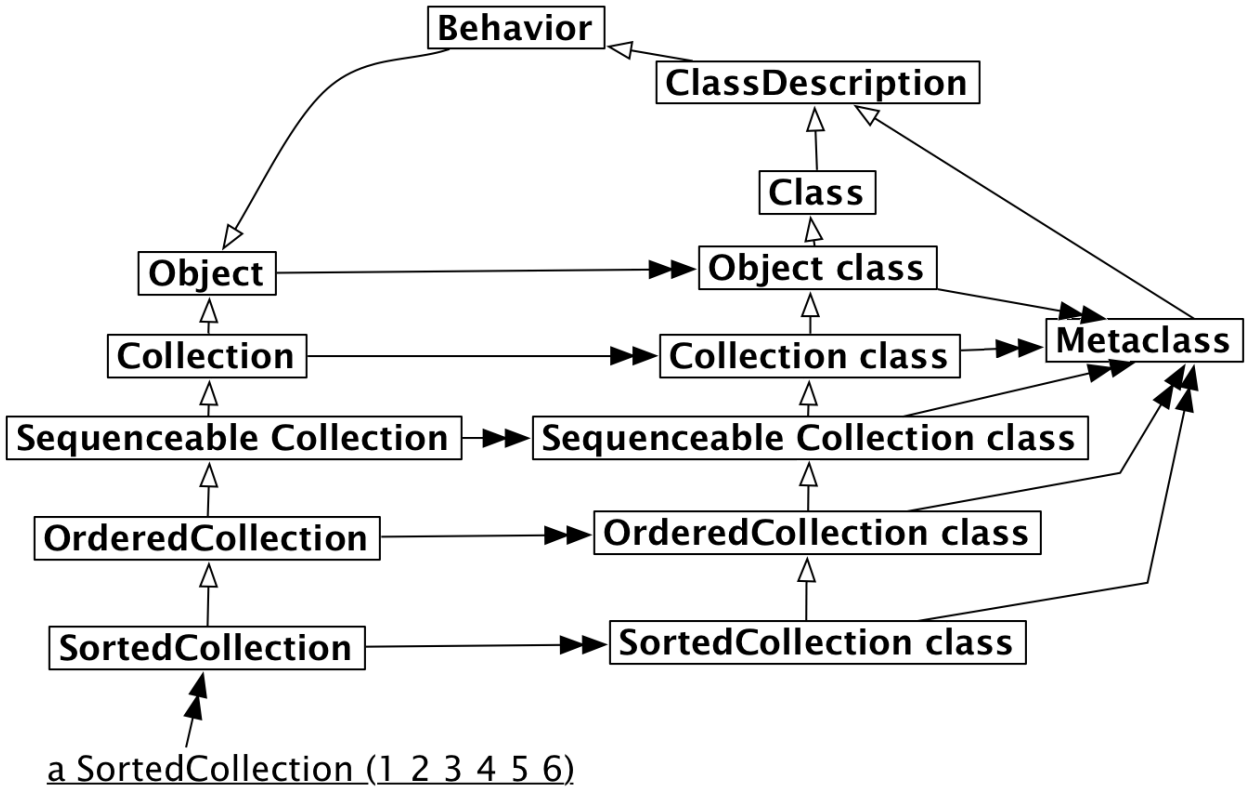

Рис. 17.8. Кожен метаклас – екземпляр класу *Metaclass*

ClassDescription

ClassDescription – абстрактний клас, що надає засоби, потрібні двом його безпосереднім підкласам, *Class* і *Metaclass*. *ClassDescription* розширює базову функціональність *Behavior*: іменовані змінні екземпляра, категоризація методів у протоколи, підтримка наборів змін та їхнє протоколювання, більшість механізмів, потрібних для внесення змін.

Class

Class представляє загальну поведінку всіх класів. Він підтримує ім'я класу, методи компіляції, сховище методів та змінні екземпляра. Він надає конкретне представлення іменам змінних класів і змінних спільного пулу (*addClassVarName*:, *addSharedPool*:, *initialize*). Оскільки метаклас є класом для свого єдиного екземпляра (не метакласу), то всі метакласи, зрештою, наслідують від класу *Class* (див. рис. 17.6–17.9).

17.11. Кожен метаклас є екземпляром класу *Metaclass*

Наступне питання стосується метакласів. Оскільки вони теж об'єкти, то мусять бути екземплярами якогось класу, але якого? Метакласи – також об'єкти, екземпляри класу *Metaclass*, як зображено на рис. 17.8. Екземпляри класу *Metaclass* – анонімні класи, кожен з яких має точно один екземпляр, який є класом.

Metaclass визначає загальну поведінку метакласів. Він містить методи для створення екземплярів (*subclassOf:*), створення ініціалізованого екземпляра – єдиного екземпляра метакласу, ініціалізації змінних класу, екземплярів метакласу, компіляції методів та отримання інформації про клас (зв'язки наслідування, змінні екземплярів тощо).

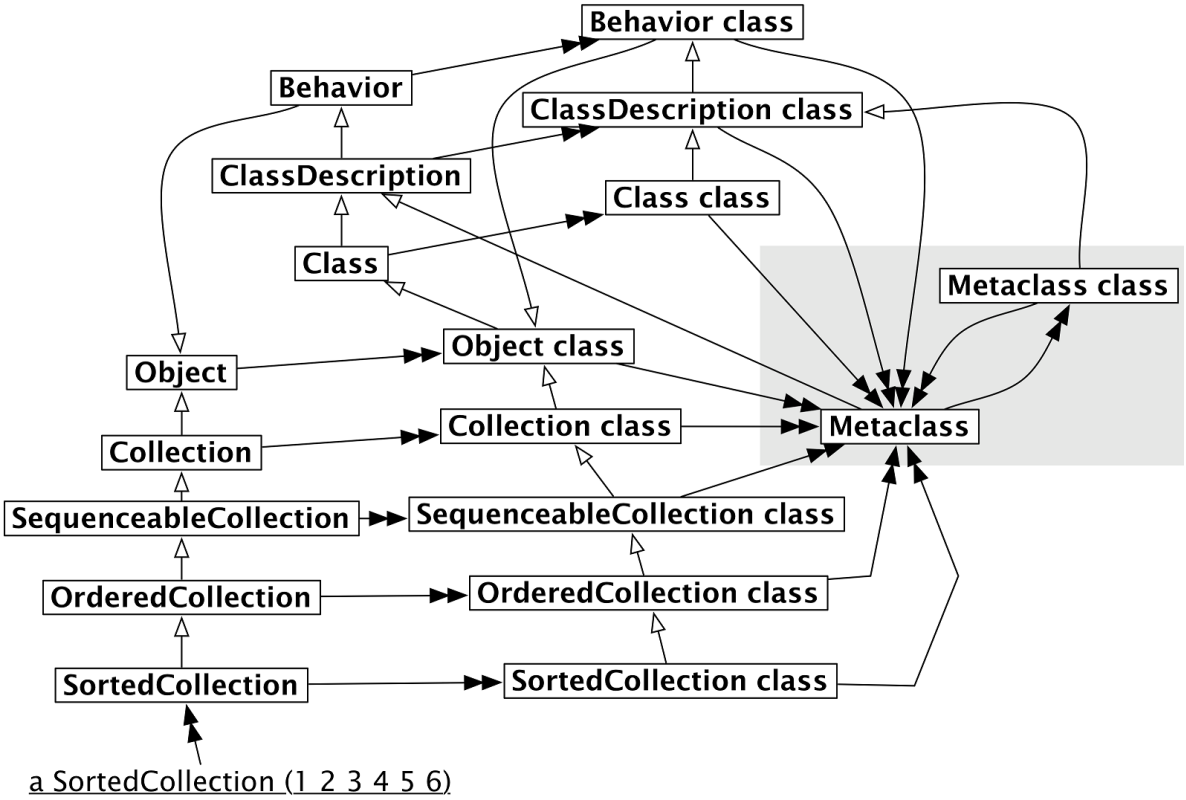

Рис. 17.9. Усі метакласи є екземплярами класу *Metaclass*, навіть метаклас класу *Metaclass*

17.12. Метаклас класу *Metaclass* є екземпляром класу *Metaclass*

Останнє питання, на яке потрібно дати відповідь: а що є класом класу *Metaclass* class? Відповідь проста – це метаклас, тому він має бути екземпляром класу *Metaclass*, як і всі інші метакласи в системі (див. рис. 17.9).

З рис. 17.9 видно, що всі метакласи є екземплярами класу *Metaclass* включно з метакласом самого класу *Metaclass*. Порівнявши рисунки 17.8 і 17.9, можна побачити, що ієрархія метакласів ідеально віддзеркалює ієрархію класів на всьому шляху аж до *Object class*.

Наведений нижче приклад ілюструє, як можна дослідити ієрархію класів, щоб продемонструвати правильність рис. 17.9. (Правду кажучи, ми дещо прибрехали – *Object class superclass >>> ProtoObject class*, не *Class*. У Pharo потрібно піднятись на один рівень вище, щоб досягти класу *Class*).

```
Collection superclass
>>> Object
```

```
Collection class superclass
>>> Object class
```

```
Object class superclass superclass
>>> Class
```

```
Class superclass
>>> ClassDescription
```

```
ClassDescription superclass
>>> Behavior


Behavior superclass
>>> Object


"Класом кожного метакласу є Metaclass"
Collection class class
>>> Metaclass

Object class class
>>> Metaclass

Behavior class class
>>> Metaclass

Metaclass class class
>>> Metaclass


"Metaclass є особливим різновидом класу"
Metaclass superclass
>>> ClassDescription
```

17.13. Підсумки розділу

Цей розділ дав змогу заглянути в глибину однорідної об'єктної моделі, і ретельніше пояснив, як організовані класи. Якщо ви розгубилися чи заплуталися, то пригадайте, що ключем є надсилання повідомлень: метод потрібно шукати у класі отримувача. Це працює для *будь-якого* отримувача. Якщо метод не знайдено в класі отримувача, то пошук продовжується у його надкласах.

- Кожен клас є екземпляром свого метакласу. Метакласи неявні. Метаклас створюється автоматично кожного разу, коли створюється клас, який є його єдиним екземпляром. Простіше кажучи, метаклас – це клас, єдиним екземпляром якого є клас.
- Ієрархія метакласів паралельна ієрархії класів. Пошук методу класу такий самий, як пошук методу звичайного об'єкта, і прямує ланцюжком наслідування метакласів.
- Кожен метаклас наслідує *Class* і *Behavior*. Метакласи – також класи, тому вони наслідують *Class*. *Behavior* реалізує поведінку, спільну для всіх сутностей, що мають екземпляри.
- Кожен метаклас є екземпляром класу *Metaclass*. *ClassDescription* підтримує все, що є спільного в класах *Class* і *Metaclass*.
- Метаклас класу *Metaclass* є екземпляром класу *Metaclass*. Відношення «*є екземпляром*» утворює замкнуте коло, тому клас класу *Metaclass class* – це *Metaclass*.