МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Озерский технологический институт — филиал НИЯУ МИФИ

Кафедра прикладной математики

Вл. Пономарев

ПРАКТИКУМ

по объектно-ориентированному программированию

Учебно-методическое пособие

Часть 4. Объектно-ориентированный анализ и проектирование

2019 г.

УДК 681.3.06
П 56

Вл. Пономарев. Практикум по ООП. Учебно-методическое пособие по объектно-ориентированному программированию. Часть 4. Объектно-ориентированный анализ и проектирование. Озерск: ОТИ НИЯУ МИФИ, 2019. — 15 с.

В пособии излагается, как выполнять практические работы по дисциплине «Объектно-ориентированное программирование».

В качестве основного материала при выполнении практических работ пособие предназначено для студентов, обучающихся по направлению подготовки 09.03.01 «Информатика и вычислительная техника» и специальности 09.05.01 «Применение и эксплуатация автоматизированных систем специального назначения».

Рецензенты:
1.
2.

УТВЕРЖДЕНО
Редакционно-издательским
Советом ОТИ НИЯУ МИФИ

Содержание

Общие цели занятий

В ходе практических работ изучаются основы использования классов в рамках методологии объектно-ориентированного программирования.

В этой части работ рассматриваются следующие темы:

1) изучение среды моделирования StarUML 2.0;

2) создание диаграмм классов;

3) создание отношений;

3) создание диаграмм прецедентов;

4) создание диаграмм последовательности;

5) создание диаграмм состояний.

К практическим работам приписаны контрольные вопросы и упражнения. Контрольные вопросы могут быть заданы преподавателем в ходе защиты работы, однако преподаватель может задавать и другие вопросы, не указанные в списке.

Для защиты знаний и навыков, полученных в ходе выполнения практических работ студент должен иметь тетрадь (12-18 листов). Для каждой выполненной работы в тетрадь записывается отчет.

Отчет по работе начинается с заголовка:

1. Фамилия Имя Отчество
2. Группа
3. Дата начала выполнения работы
4. Код работы
5. Название работы
6. Цели работы
7. Задачи работы

При необходимости при выполнении работы в отчет записываются контрольные значения.

Во время защиты тетрадь используется для вопросов преподавателя и ответов студента.

1. Работа ООП-401 (2 часа). Введение в моделирование на UML

Цели:
- Изучение среды моделирования StarUML 2.0.
Задачи:
- изучение интерфейса;
- создание диаграммы классов;
- создание отношений между классами;
- создание диаграммы прецедентов;
- создание диаграммы последовательности.

1.1. Элементы интерфейса среды моделирования

При необходимости нужно установить среду на компьютер.

После открытия среды, а также по ходу выполнения работы может появляться диалог предложения купить продукт, в котором нажимайте кнопку «Evaluate» (рисунок 1).

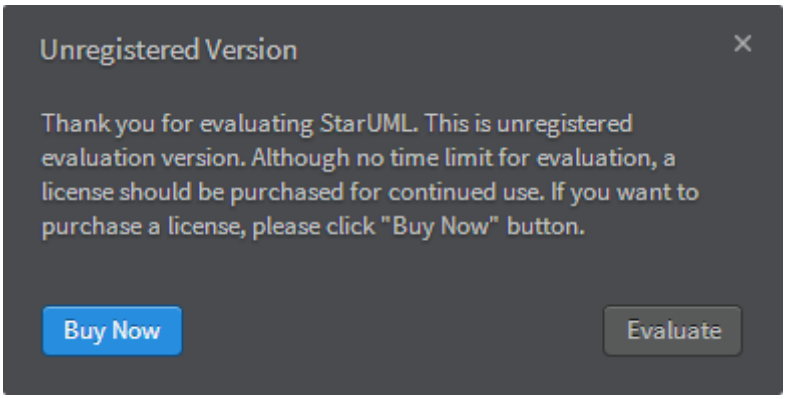

Рисунок 1 — Ненавязчивый диалог

Открываем среду моделирования.

Изучаем основные элементы среды.

Вверху находится меню приложения, а в левой части окна вверху находится список рабочих диаграмм (рисунок 2).

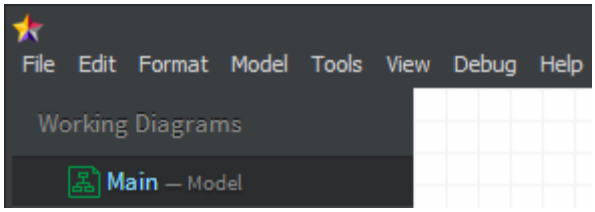

Рисунок 2 — Меню

После первого открытия в списке диаграмм одна диаграмма Main.

В левой части окна внизу находятся панель Toolbox. Здесь выбираются элементы, которые размещаются на диаграммах (рисунок 3).

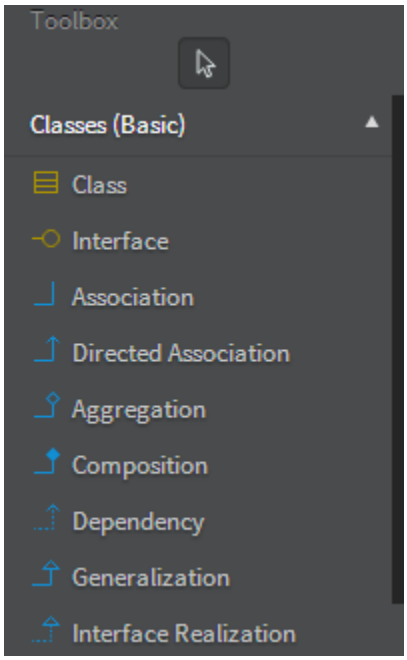

Рисунок 3 — Инструменты

В правой части окна вверху находится Model Explorer, показывающий структуру проекта, который пока не имеет названия (рисунок 4).

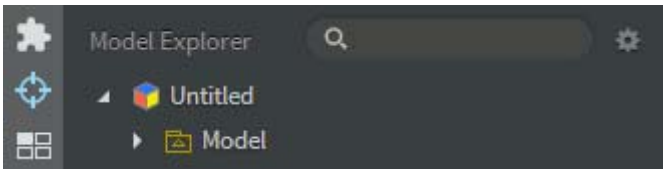

Рисунок 4 — Обозреватель модели

В правой части окна внизу расположен блок управления стилями, форматированием и выравниванием элементов (рисунок 5).

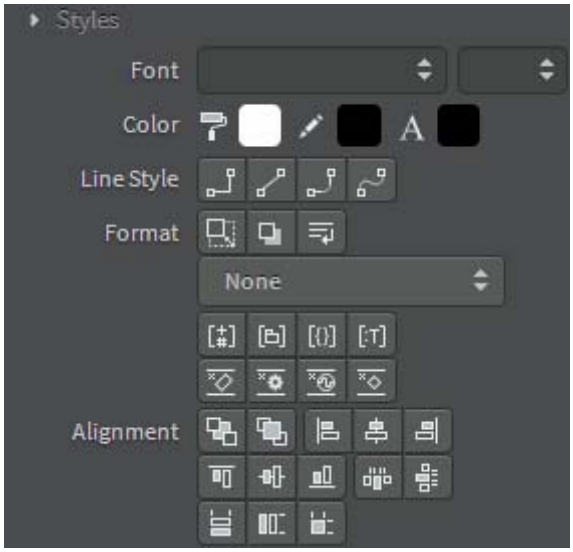

Рисунок 5 — Панель стилей

В правой части внизу также располагается панель свойств выбранного элемента или диаграммы. Панель появляется при выборе элемента, например, в структуре проекта.

Выберем в Model Explorer элемент Untitled (рисунок 6).

Рисунок 6 — Свойства модели

Вместо названия модели Untitled введем PaintApplication и нажмем Enter.
Выберем в меню приложения Save или введем сочетание Ctrl+S.
Название файла модели — тоже PaintApplication.
Каталог для размещения модели: C:\OOP401.

1.2. Диаграмма классов

Создадим диаграмму классов.
Выбираем в меню приложения Model — Add Diagram — Class Diagram.
Диаграмма появляется в структуре проекта (рисунок 7).

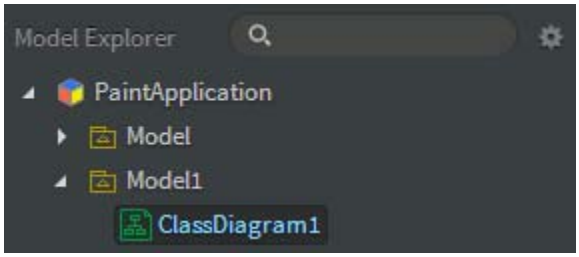

Рисунок 7 — Диаграмма в обозревателе модели

Диаграмма классов выделена (если диаграмма не выделена, то сейчас следует выделить ее), поэтому можно дать ей подходящее название, например, MainClasses. (рисунок 8).

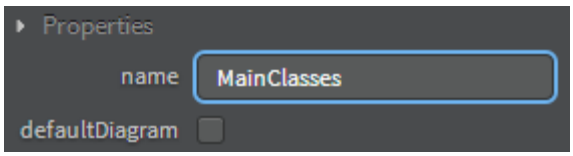

Рисунок 8 — Свойства диаграммы

Добавляем на диаграмму класс фигуры.

Выбираем в инструментах Class и щелкаем левой кнопкой мыши в области диаграммы (область диаграммы похожа на тетрадный лист в клеточку) где-нибудь вверху слева (рисунок 9).

Рисунок 9 — Кнопки быстрого редактирования

Сразу нужно изменить название класса на figure.
Как видим, возле пиктограммы класса есть много кнопочек.
Это действия, связанные с классом, которые можно выполнить.
Если эти кнопочки исчезли, нужно дважды щелкнуть на элемент.
Нажмем кнопку «Add Sub-Class».
Появится подкласс, который переименуем в `square`.
Дважды щелкнем на класс figure и добавим еще один подкласс circle.
В результате получим иерархию классов (рисунок 10).

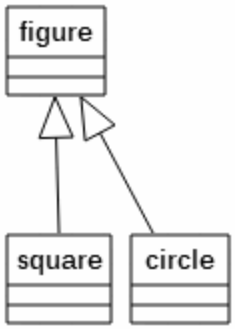

Рисунок 10 — Иерархия классов фигур

Выберем класс figure.
В окне свойств установим флажок Abstract.
Название абстрактного класса в пиктограмме отображается курсивом.
Для добавления атрибута щелкнем правой кнопкой на класс figure и выберем в контекстном меню «Add — Attribute». Сразу после появления атрибута в панели свойств меняем название на x, вписываем тип int, видимость protected, и начальное значение, равное нулю.
Аналогичным образом добавляем второй атрибут y.
Добавим метод в класс figure. Для этого выбираем в том же контекстном меню «Add — Operation». Сразу после появления метода изменяем его название на draw, и помечаем метод как абстрактный.
Аналогично добавляем неабстрактный метод draw в подклассы.

В результате получим следующую диаграмму классов (рисунок 11).

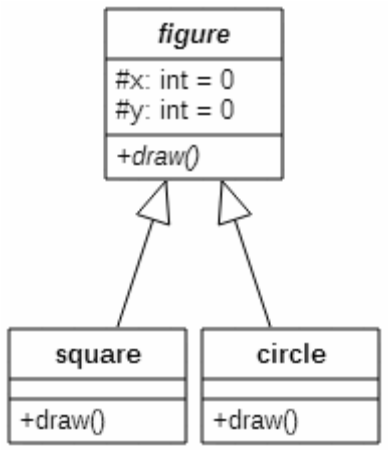

Рисунок 11 — Законченная иерархия классов фигур

Далее нам нужен класс приложения PaintApp.
Размещаем его на диаграмме и моделируем.
Для хранения фигур добавляем атрибут figures.
Видимость: private.
Тип: figure*.
Кратность (*multiplicity*): 0..*.
Флажок IsOrdered включим.
Добавляем операции:
show(): void;
add(): figure*;
item(): void.
Возвращаемые значения операций приписываем вручную.
Чтобы добавить параметр метода item, щелкаем правой на метод и выбираем в контекстном меню «Add parameter». Затем в свойствах параметра выбираем его тип. Аналогичным образом формируем параметр метода add.
Результат формирования класса показан на рисунке 12.

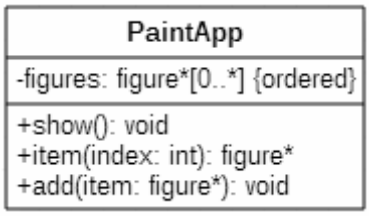

Рисунок 12 — Класс приложения

Между классами приложения и фигуры существует связь, которая должна быть отображена на диаграмме классов.
Связь отображается в виде отношения ассоциации, которое может быть просто ассоциацией, агрегацией или композицией.

Отношение добавляется следующим образом.

Выбираем инструмент, например, ассоциацию.

Затем на диаграмме нажимаем левую кнопку мыши на целевом классе, и, не отпуская кнопку, тянем мышь до исходного класса, после чего кнопку отпускаем. Если просто щелкнуть на класс, появится рефлексивная (замкнутая на класс) связь.

В принципе неважно, какое отношение будет добавлено, потому что и ассоциация, и агрегация, и композиция суть одно и то же, различие только в свойствах. Свойства отдельно описывают первый и второй концы. Если, например, нужно, чтобы появился ромбик у какого-то конца, нужно для этого конца выбрать свойство aggregation. Значение shared указывает на агрегацию, значение composition указывает на композицию, значение none указывает на отсутствие ромбика. Чтобы сразу протянуть связь типа «агрегация», тянуть нужно от части к целому.

Так же задается навигация. Значение свойства navigable указывает на наличие возможности навигации в сторону данного конца.

Результат добавления отношения показан на рисунке 13.

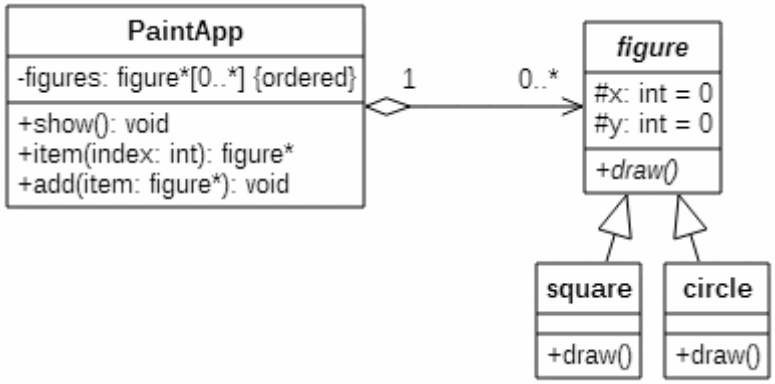

Рисунок 13 — Отношение агрегации

1.2.1. Генерирование кода

Чтобы сгенерировать код построенной диаграммы классов, в среде моделирования нужно установить расширение.

Если расширение не установлено, то в меню «Tools» не видно названия никакого языка программирования. На рисунке 14 показан вариант, когда расширение для генерации кода на C++ установлено.

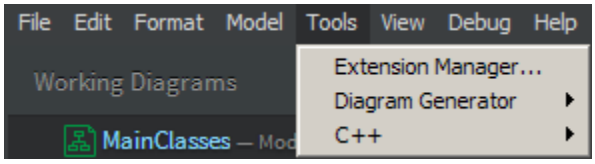

Рисунок 14 — Установлено расширение для генерации кода на C++

Чтобы установить расширение, нужно выбрать в меню «Tools» пункт «Extension Manager…». Появится диалог менеджера расширений, в котором нужно найти расширение «C++ code generation…» и щелкнуть кнопку «Install».

Чтобы сгенерировать код при установленном расширении, выбираем в меню «Tools» язык программирования и далее «Generate Code…».

Появится диалог для выбора области, которая генерируется.

Выбираем «Model1» и нажимаем кнопку «OK».

Далее при помощи стандартного каталога для выбора папки ищем место для размещения кода, при необходимости создаем папку при помощи этого же диалога. Рекомендуется для работы с проектом создать папку в корневом каталоге, в которой создать папку Code для размещения кода.

По завершении генерации, которое никак не обозначается, можно изучить полученные файлы кода. Меня результат не впечатлил, генератор кода слабенький, увы. Для чистой виртуальной функции он сгенерировал тело, непонятно зачем. На установленные отношения между классами он никак не реагирует. Зато подробно комментирует параметры. Одно только и радует, что код в основном соответствует ТПМ ☺.

Заметим, что при установленной кратности атрибута figures генерируется элемент данных типа Vector (не vector), при этом никаких включений модулей не наблюдается. Наверное, разработчик предусматривал возможность генерирования массива, если в настройках генератора кода выбрать соответствующую опцию, однако у меня этот фокус не удался. Настройки генератора кода находятся в меню «Tools — C++ — Configure…». Я понял так, что Vector — это в принципе единственно возможный вариант, независимо от указаний модели.

Заметим, что установленное расширение должно выполнять обратное преобразование кода на C++ в диаграмму классов, однако ничего внятного не происходит.

1.2.2. Конструкторы и деструкторы

Самостоятельно добавляем операции конструкторов всем классам.

Для класса figure конструктор имеет два параметра a и b со значениями по умолчанию, равными нулю. Конструкторы других классов параметров не имеют.

Пробуем сгенерировать код, убеждаемся, что операции генерируются, однако они имеют вид методов, а не конструкторов. Видимо, нужно использовать либо специальное название, либо какой-то стереотип операции.

1.2.3. Сокрытие разделов класса

Напоследок для классов square и circle можно скрыть области атрибутов, выбирая в контекстном меню «Format — Supress Attributes».

Таким же образом скрывается область операций.

Если скрыть обе области, получится просто прямоугольник.

1.2.4. Удаление элементов

При удалении каких-либо элементов, в зависимости от порядка действий, они могут либо оставаться в модели, либо нет. Иначе говоря, при удалении чего-либо нужно удалять из модели, а не из диаграммы, выбирая соответствующий пункт контекстного меню.

1.3. Диаграмма прецедентов

Добавим диаграмму прецедентов.

Выбираем в меню «Model — Add Diagram — Use Case Diagram».

Появится новый чистый лист, в обозревателе моделей новая модель с названием «Model2», в которой расположена диаграмма «UseCaseDiagram1».

Диаграмму переименуем в «MainCases».

Далее добавляем на диаграмму актера «User» и при помощи кнопок быстрого редактирования добавляем вариант использования «add».

Затем снова дважды щелкаем на актера и добавляем еще один вариант использования с названием «show».

Результат этих действий показан на рисунке 15.

Рисунок 15 — Действия пользователя

Теперь добавим еще один прецедент с названием «item», как часть прецедента «show». Для этого нужно выбрать кнопку быстрого редактирования «Add Included Use Case» прецедента «Shows figures».

Поскольку метод «item» может использоваться сам по себе, а не только как часть прецедента «show», между пользователем и прецедентом «item» нужно установить отношение ассоциации (рисунок 16).

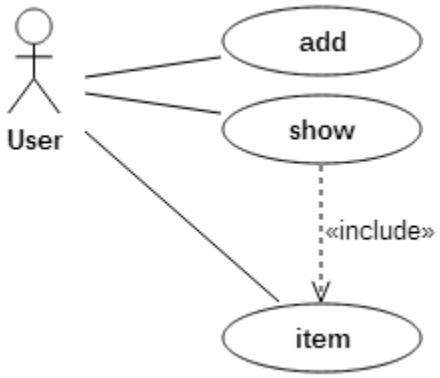

Рисунок 16 — Включаемый прецедент

Поскольку данные прецеденты просты, точек расширения у них нет, хотя можно попробовать вместо зависимости прецедента «show» от прецедента «item» ввести точку расширения первого прецедента во второй прецедент.

Для этого нужно сначала удалить из модели зависимость прецедентов при помощи контекстного меню, а затем при помощи кнопки быстрого редактирования «Add Extension Point» прецедента «show» добавить точку расширения с именем «item». Надо сказать, что точки расширения среда моделирования отображает не в соответствии с UML (рисунок 17).

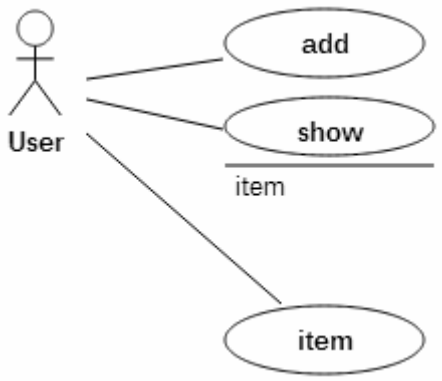

Рисунок 17 — Точка расширения прецедента

В соответствии с UML точка расширения находится в овале прецедента под горизонтальной чертой, разделяющей овал надвое.

Кроме того, непонятно, где описывается спецификация прецедента.

Спецификация состоит из названия, краткого описания, списка главных и второстепенных актеров, предусловий, описания основного потока (сценария), постусловий и альтернативных потоков.

Видимо, для этой цели предусмотрено поле под названием Documentation на панели свойств, но мне кажется, оно недостаточно большое для этого, кроме того, как формировать разделы спецификации, неясно.

1.4. Диаграмма последовательности

Особых последовательностей взаимодействия объектов у нас нет, поэтому эта диаграмма очень проста. Приложение может вызвать метод `draw()` какого-либо объекта.

Добавляем диаграмму последовательности (*sequence diagram*), выбирая ее в контекстном меню модели 2 на дереве обозревателя.

На диаграмме должно быть как минимум две линии жизни (LifeLine, прямоугольник объекта с пунктирной линией вниз). Добавить линию жизни можно двумя способами:

- используя инструмент LifeLine;
- перетягиванием класса или интерфейса или другого объекта с дерева в обозревателе на диаграмму последовательности.

Добавим линии жизни, перетаскивая из дерева объектов обозревателя классы PaintApp и figure. Для знакомства с возможностями в контекстном меню нарисованных элементов выберем «Format — Autosize» (автоматический размер), затем «Format — Show Shadow» (скрыть тень). После этого выключите флажок «Format — Autosize».

Заменим названия линий жизни на «A:PaintApp» и «figures:figure».

Кстати, проверить правильность можно при помощи маленькой зеленой кнопочки с чеком внизу справа с подсказкой Validation Results.

Теперь выбираем инструмент «Messaage» и протягиваем сообщение от левой линии жизни до правой. Сразу после этого нажимаем кнопку быстрого редактирования «Select Operation» и выбираем метод draw.

Примерный результат показан на рисунке 18.

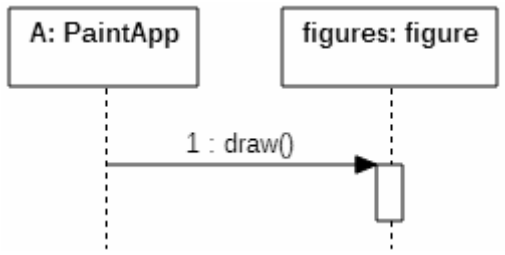

Рисунок 18 — Последовательность сообщений 1

Все это чрезвычайно глупо, потому что сообщения должны посылаться от одних объектов к другим, а у нас практически нет взаимодействующих объектов. Тем не менее, мы в общих чертах научились строить диаграмму последовательности.

Добавим еще одну диаграмму последовательности от модуля 2.

Первая линия жизни, — это User из диаграммы прецедентов, вторая линия жизни, — это PaintApp из диаграммы классов. Обозначим первый объект как «U:User», второй как «A:PaintApp». Протянем три сообщения: два раза add, один раз show (рисунок 19).

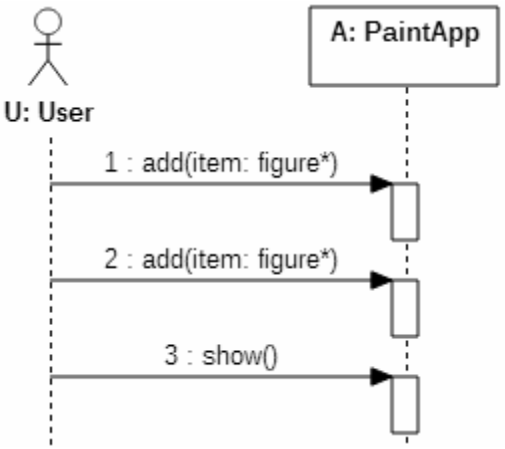

Рисунок 19 — Последовательность сообщений 2

В этой диаграмме хоть какой-то смысл есть. Чтобы было совсем интересно, нужно добавить еще одну линию жизни «figures:figure», и протянуть сообщение draw от метода show на линии жизни «PaintApp» до линии жизни «figures:figure».

К сообщению нужно приписать стереотип «call» (рисунок 20).

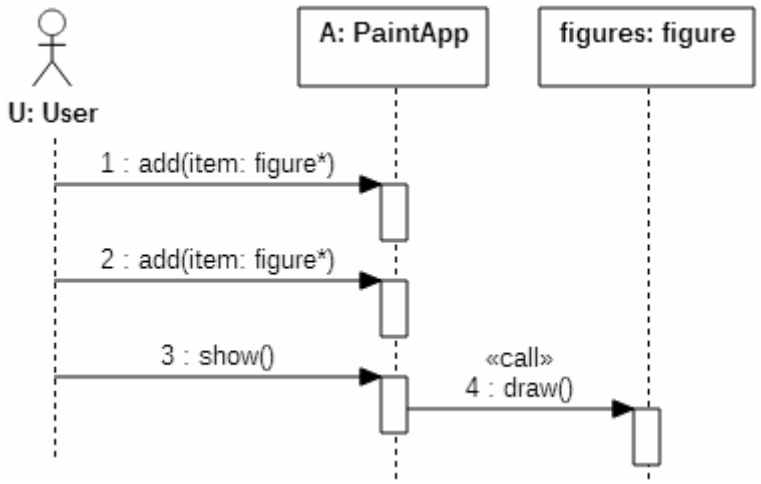

Рисунок 20 — Последовательность сообщений 3

К сожалению, вызов прикрепляется не к методу show, а к линии жизни, что в общем-то неверно, потому что именно show вызывает draw.

Никакого кода модель 2 не генерирует, хотя должна бы.

1.5. Вопросы и упражнения

1. Зачем нужны прецеденты?
2. Что содержит спецификация прецедента?
3. Перечислите синтаксические элементы ассоциации.
4. Что показывает кратность ассоциации?
5. Что показывает возможность навигации?
6. Объясните семантику агрегации.
7. Объясните семантику композиции.
8. Что такое стереотип?