

Формы наследования

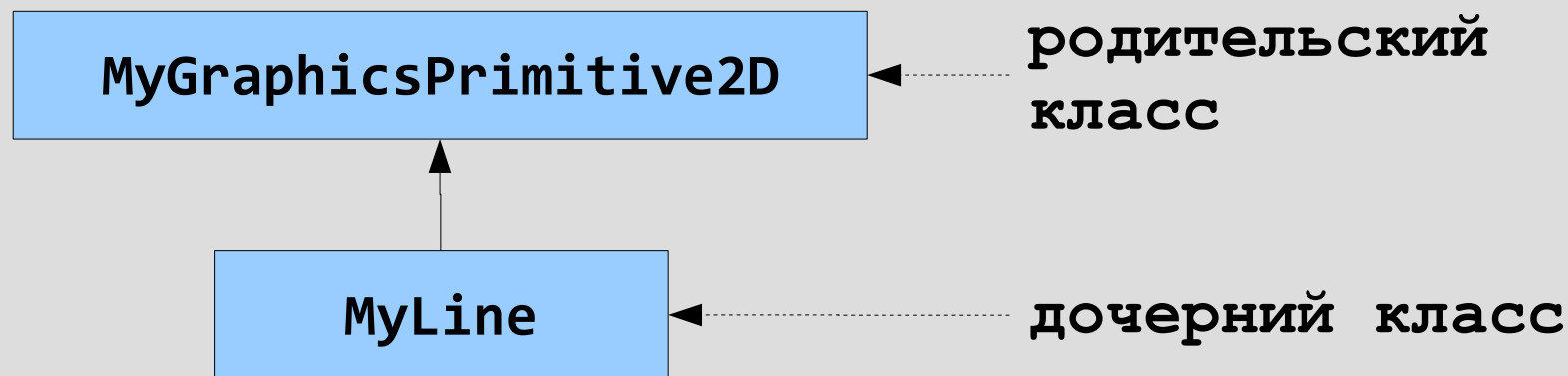
- Понятие наследования
- Формы наследования
- Правильные формы наследования

Понятие наследования

- Наследование – это такое **отношение** между классами, когда один класс **повторяет** структуру и поведение другого класса (одиночное наследование) или других (множественное наследование) классов
- Наличие механизма наследования отличает **объектно-ориентированные** языки от объектных

Понятие родительского и дочернего классов

- Класс, структура и поведение которого наследуется, называется **суперклассом**, **надклассом**, **базовым** или **родительским** классом
- Класс, производный от суперкласса, называется **подклассом**, **производным** или **дочерним** классом



Формы наследования

- В подклассе структура и поведение исходного суперкласса могут дополняться, переопределяться или ограничиваться
- Исходя из этого, можно выделить различные формы наследования

Формы наследования

- (1) Специализация
- (2) Спецификация
- (3) Обобщение
- (4) Расширение
- (5) Ограничение
- (6) Конструирование
- (7) Варьирование
- (8) Комбинирование

Специализация

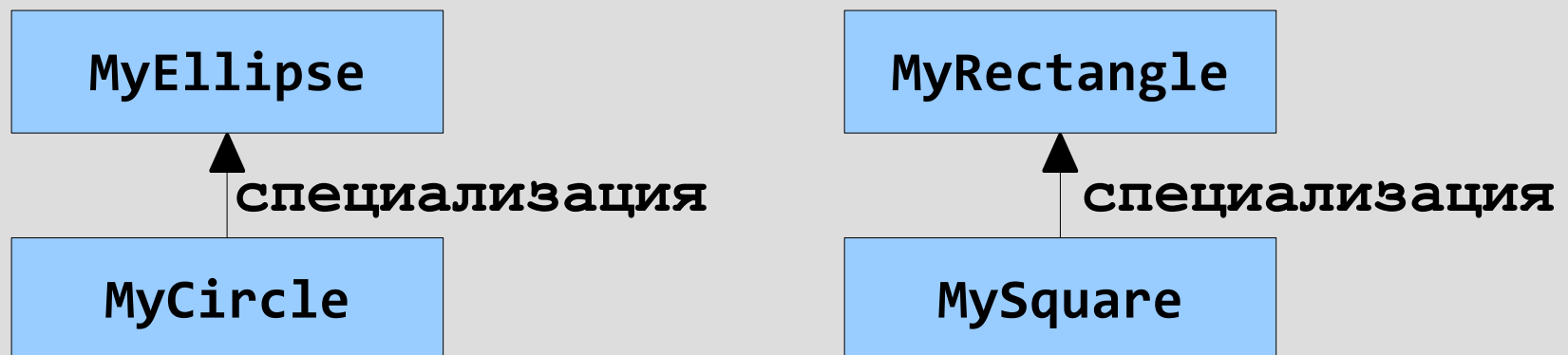
- Дочерний класс является более конкретным, частным или специализированным случаем родительского класса
- Дочерний класс удовлетворяет спецификациям родителя во всех существенных моментах, т.е. его можно использовать вместо родительского класса
- Поведение базового класса, в основном, переопределяется

Пример специализации

- Имеется класс `MyEllipse`, который представляет собой эллипс и поддерживает следующие операции:
 - `MyEllipse(MyPoint, int, int)` - инициализация при создании (конструктор)
 - `setPos(MyPoint)` - задание позиции на плоскости
 - `move(int, int)` - перемещение на заданное смещение по горизонтали и вертикали
 - `draw()` - отрисовка самого себя

Пример специализации

- Класс **MyCircle**, представляющий собой окружность, может быть порожден как специализация базового класса **MyEllipse**, т.к. поддерживает те же операции, но переопределяет их реализацию
- Класс **MySquare** (квадрат) является специализацией класса **MyRectangle** (прямоугольник)



Спецификация

- Родительский класс **описывает** поведение, которое реализуется в дочернем классе, но оставлено **нереализованным** в родительском
- В таких случаях родительский класс называют **абстрактно-специфицированным** классом

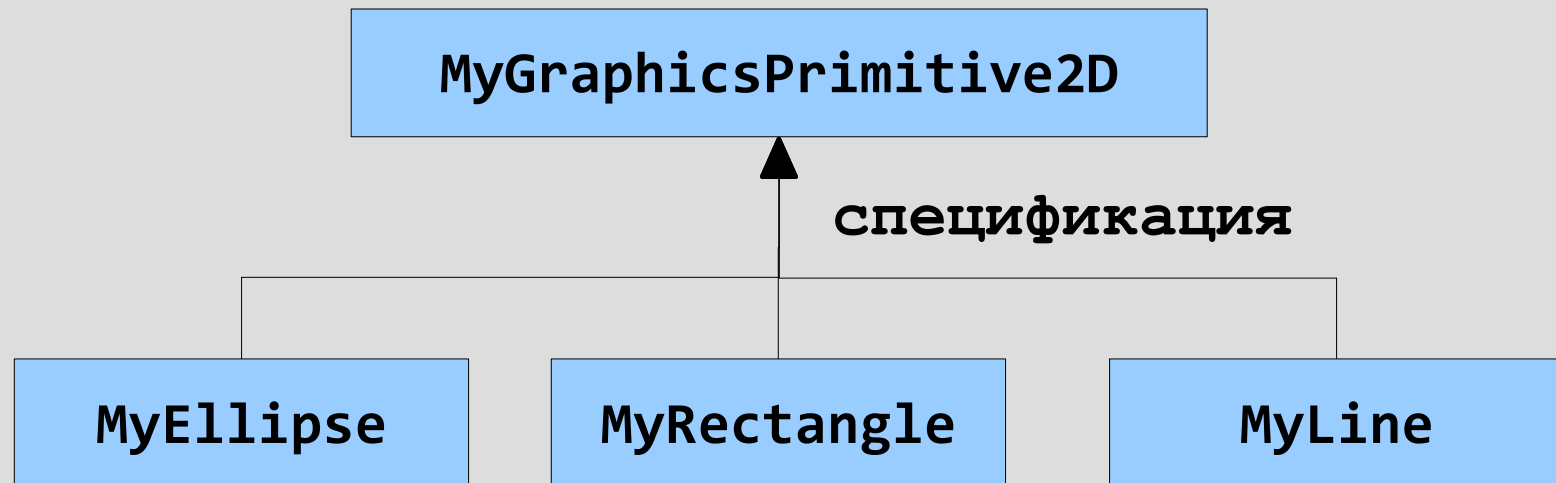
Пример спецификации

- Имеются классы плоскостных графических примитивов – `MyLine`, `MyRectangle`, `MyTriangle` и `MyEllipse`
- Классы должны поддерживать **единый набор** операций:
 - `setPos(int, int)` – задание позиции на плоскости
 - `move(int, int)` – перемещение на заданное смещение по горизонтали и вертикали
 - `draw()` – отрисовка самого себя

Пример спецификации

- Чтобы **гарантировать** наличие указанных операций, нужно определить обобщенный графический примитив **MyGraphicsPrimitive2D**, который будет содержать в себе эти операции
- Ни одна из указанных операций **не будет реализована** в этом классе, т.к. их реализация зависит от типа примитива
- Класс **MyGraphicsPrimitive2D** является абстрактным и от него **нельзя породить** экземпляры

Пример спецификации

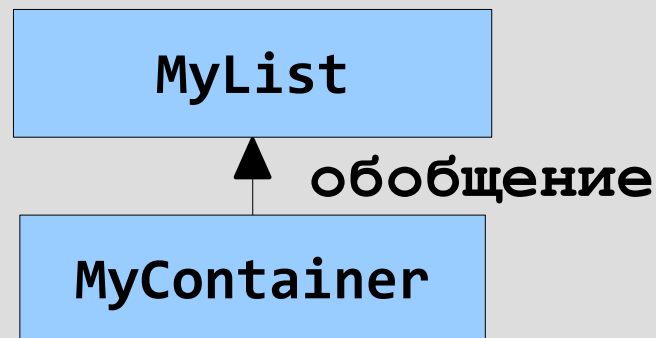


Обобщение

- Дочерний класс **модифицирует** и **переопределяет** некоторые методы родительского класса с целью получения объекта **более общей категории**
- Данная форма наследования противоположна специализации и обычно применяется, когда построение происходит на основе **существующих** классов, которые мы не хотим или не можем изменять

Пример обобщения

- От класса «список» **MyList**, который реализует список с возможностью доступа к его голове и хвосту, наследуется класс «контейнер» **MyContainer**, который реализует произвольный доступ к элементам
- В дочернем классе по сравнению с родительским имеются методы добавления элементов в начало, конец и произвольную позицию контейнера и т.д.

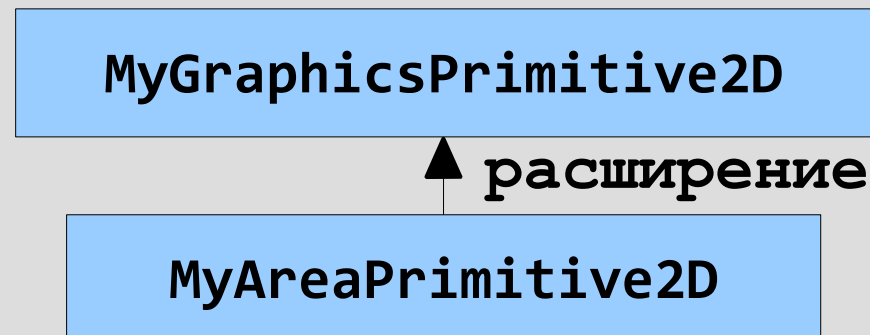


Расширение

- Дочерний класс **добавляет** новые функциональные возможности к родительскому классу, но **не меняет** наследуемое поведение
- В отличие от обобщения или специализации при расширении дочерний класс **не переопределяет** ни одного метода базового класса, а добавленные методы слабо связаны с существующими методами родителя

Пример расширения

- На основе обобщенного графического примитива `MyGraphicsPrimitive2D` в двумерной плоскости можно определить производный класс `MyAreaPrimitive2D`, который определяет все примитивы, имеющие площадь
- По сравнению с базовым классом в класс `MyAreaPrimitive2D` добавлен метод `area()`, который возвращает площадь плоскостной фигуры



Ограничение

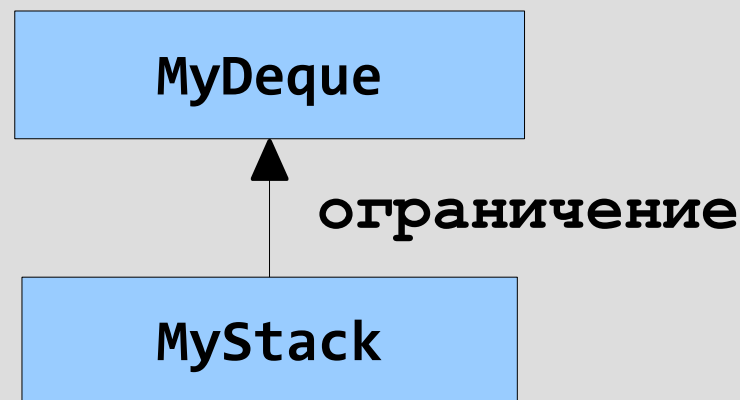
- Дочерний класс **ограничивает** использование некоторых методов родительского класса
- Так же, как и при обобщении, порождение для ограничения чаще всего возникает, когда, программист строит класс на основе **существующей** иерархии классов, которая не должна или не может быть изменена

Пример ограничения

- Имеется класс **MyDeque**, представляющий собой двустороннюю очередь (элементы добавляются и извлекаются с любого конца)
- На основе класса **MyDeque** порождается класс **MyStack**, представляющий собой стек, в котором добавление и извлечение элементов осуществляется только из одного конца

Пример ограничения

- Класс **MyStack** должен скрыть методы добавления и извлечения элементов из другого конца очереди

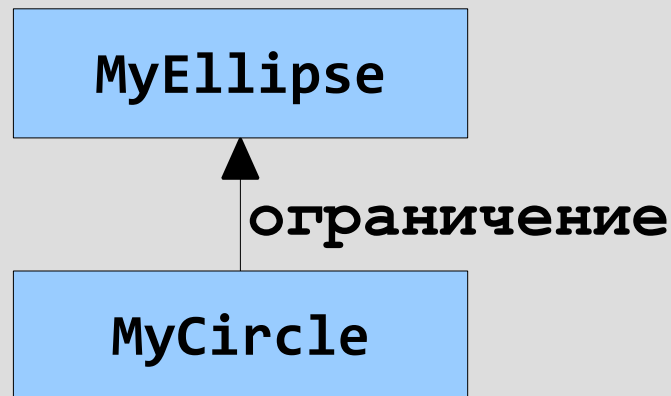


Пример ограничения

- Имеется класс `MyEllipse`, который представляет собой эллипс и поддерживает следующие операции:
 - `MyEllipse(MyPoint, int, int)` - инициализация при создании (конструктор)
 - `setPos(MyPoint)` - задание позиции на плоскости
 - `setRadius(int, int)` - задание радиусов
 - `draw()` - отрисовка самого себя

Пример ограничения

- Класс **MyCircle**, представляющий собой окружность, должен ограничить базовый класс **MyEllipse**, т.к. для него не применим метод **setRadius(int, int)**



Конструирование

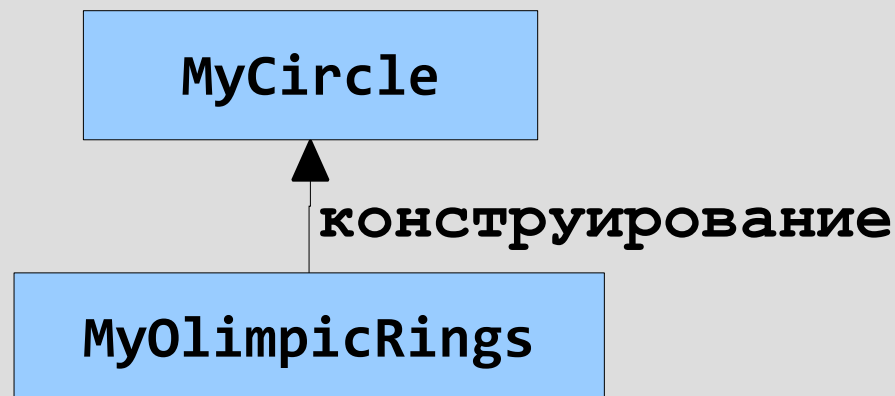
- Дочерний класс **использует** методы, предопределяемые родительским классом
- Между дочерним и родительским классами **отсутствует** отношение «is-a» или «быть экземпляром», т.е. дочерний класс не является более специализированной формой родительского класса

Конструирование

- Обычно для реализации такой формы наследования используется механизм **закрытого наследования**
- Дочерний класс часто **изменяет** не только имена методов базового класса, но и аргументы

Пример конструирования

- Класс олимпийских колец `MyOlympicRings` наследуется от класса `MyCircle`
- Класс `MyOlympicRings` для собственной отрисовки использует операции базового класса `move(int, int)` и `draw()`



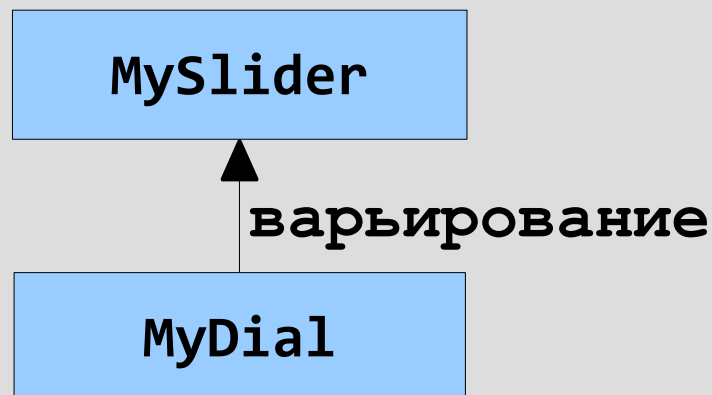
Варьирование

- Варьирование наблюдается, когда два класса имеют **сходную реализацию** и находятся примерно **на одном уровне иерархии**, т.е. являются частными случаями более общего понятия

Пример варьирования

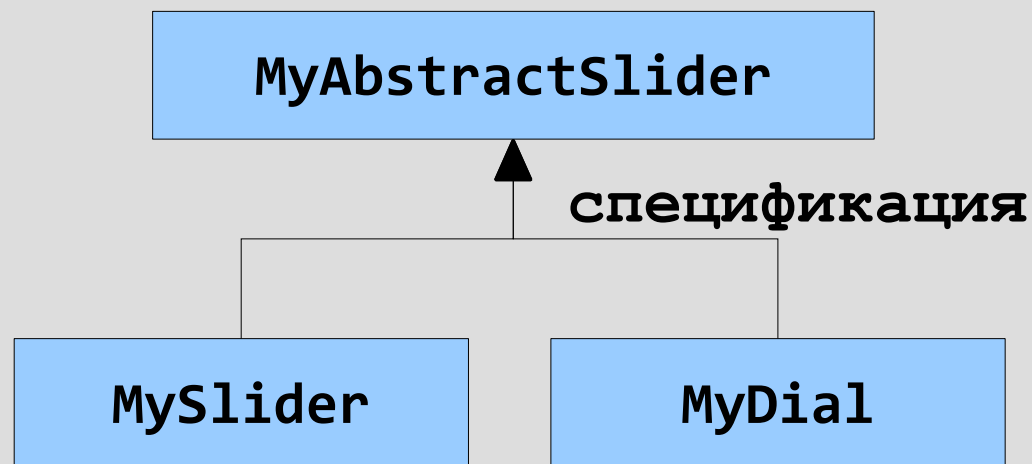
Имеется класс **MySlider**, который представляет собой элемент управления «бегунок»

- На его основе можно породить класс **MyDial**, представляющий собой «циферблат». Этот класс во многом **схож** с **MySlider**, т.к. по своей сути является «круговым бегунком»



Пример избавления от варьирования

- Однако лучшей альтернативой является выделение общего кода в **абстрактный** класс, например, **MyAbstractSlider**, и порождение обоих классов от этого общего предка
- Такой путь может быть недоступен при доработке уже **существующего** класса



Комбинирование

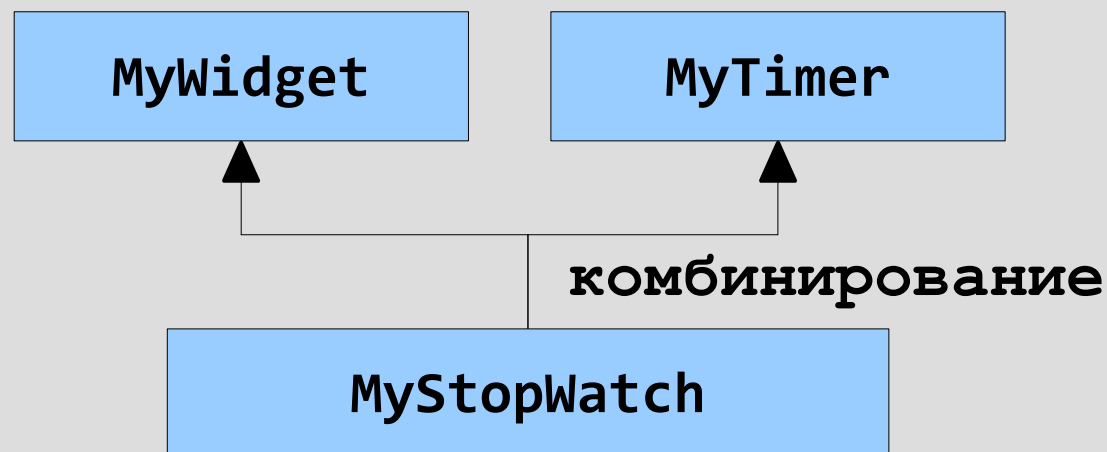
- Дочерний класс наследует черты более чем одного родительского класса
- Для комбинирования классов используется механизм множественного наследования

Пример комбинирования

- Имеется класс `MyWidget`, который обеспечивает базовое поведение всех элементов управления
- Имеется класс `MyTimer`, представляющий собой таймер

Пример комбинирования

- Для реализации виджета «секундомер» создается класс **MyStopWatch**, который наследует поведение сразу двух базовых классов
- При таком подходе класс **MyStopWatch** можно использовать и как элемент управления, и как таймер ("играет" сразу **две роли**)



Правильные формы наследования (подход "is-a")

- К правильным формам относятся только те, в которых между дочерним и родительским классом возникает **отношение "is-a"** или "быть экземпляром" или "обобщение-специализация", т.е. дочерний класс является **частным случаем** своего предка
- "Лакмусовая бумажка" наследования – это **обратная проверка**: если **B** не есть **A**, то **B** не стоит производить от **A**

Примеры правильных форм наследования (подход "is-a")

- Утверждение «прямоугольник не есть частный случай квадрата» **верно**, поэтому класс **MyRectangle** не должен наследоваться от **MySquare**
- Утверждение «окружность не есть частный случай эллипса» **неверно**, поэтому класс **MyCircle** должен наследоваться от **MyEllipse**
- Утверждение «олимпийские кольца не есть частный случай окружности» **верно**, поэтому класс **MyOlympicRings** не должен наследоваться от **MyCircle**

Правильные формы наследования (подход "is-a")

- (1) Специализация
- (2) Спецификация
- (3) Расширение (в большинстве случаев)
- (4) Ограничение (иногда)
- (5) Комбинирование (иногда)

Правильные формы наследования (подход "тип-подтип")

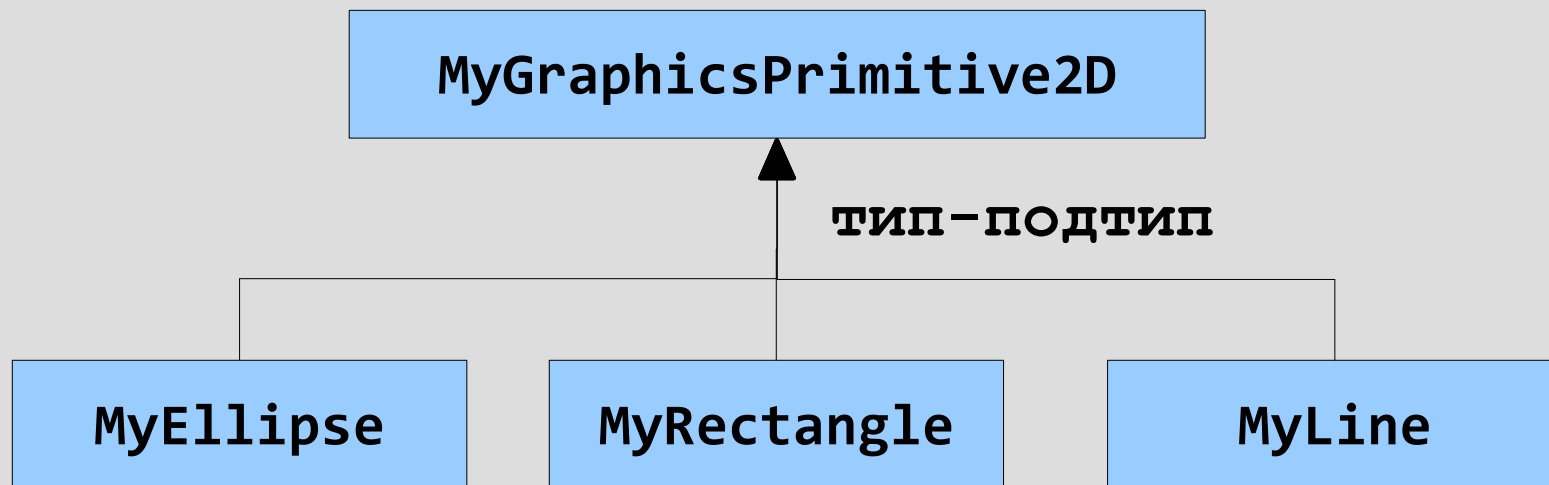
- Форму наследования можно считать правильной, если между суперклассом и подклассом имеется **отношение "тип-подтип"**, т.е. выполняется принцип подстановки
- **Принцип подстановки**: говорят, что тип **B** является подтипом **A**, если мы можем в любой ситуации **подставить** экземпляр класса **B** вместо экземпляра класса **A** без каких-либо видимых изменений в поведении

Правильные формы наследования (подход "тип-подтип")

- Для того чтобы класс **В** был **подтипом** класса **А**, он должен иметь **схожее** поведение с классом **А** во всех существенных моментах
- Схожее поведение означает, что класс **В** должен поддерживать **тот же набор методов**, что и класс **А**, при этом **семантика** реализуемых методов должна быть **сопоставима**

Примеры правильных форм наследования (подход "тип-подтип")

- В любом месте программы вместо класса `MyGraphicsPrimitive2D`, представляющего собой обобщенный графический примитив на плоскости, можно подставить классы `MyLine` (линия), `MyRectangle` (прямоугольник) и `MyEllipse` (эллипс)



Правильные формы наследования (подход "тип-подтип")

- (1) Специализация
- (2) Спецификация
- (3) Обобщение
- (4) Расширение
- (5) Конструирование (очень редко)
- (6) Варьирование (иногда)
- (7) Комбинирование

"Абсолютно" правильные формы наследования

- (1) Специализация
- (2) Спецификация
- (3) Расширение - если производный класс есть частный случай базового
- (4) Комбинирование - если производный класс есть частный случай базовых и является подтипом базовых типов

"Допустимые" формы наследования

Если используется готовая библиотека классов, то допустимы следующие формы наследования:

- (1) Обобщение
- (2) Ограничение
- (3) Варьирование
- (4) Комбинирование - если производный класс не является частным случаем базовых

"Недопустимые" формы наследования

- **Конструирование** всегда является недопустимой формой наследования и всегда может быть **заменено** на отношение агрегации или использования между классами

Пример различных форм наследования

