// **MARK: Method dispatch in Swift**

\*Когда выполняемая программа сталкивается с вызовом метода, она должна направляться по адресу реализации этого метода, которая определяется во время выполнения.

// **MARK: Определение**

Диспетчеризация (method dispatch) - процесс между вызовом метода и его выполнением, в ходе которого система определяет имплементацию метода (вариант реализации метода). Если, к примеру, есть разные переопределения метода в дочерних классах или используются протоколы с их расширениями.

Схема: Вызов метода -> диспетчеризация -> выполнение метода

// **MARK: Подходы к вызову методов объектов.**

1. Статический подход.

/// При нем метод диспетчеризации уже выбран на этапе компиляции, что повышает скорость выполнения программы. Реализацию методов диспетчеризации, использующей статистический подход, называют "прямой диспетчеризацией".

/// Используется в методах value типов (струтур), final классов, extension классов и протоколов, т.е. там, где метод не может быть переопределен.

1. Динамический подход.

При нем выбор метода диспетчеризации определяется во время выполнения программы (runtime).

// **MARK: Методы диспетчеризации**

1. Direct Dispatch / Прямая диспетчеризация (статическая)

Примеры:



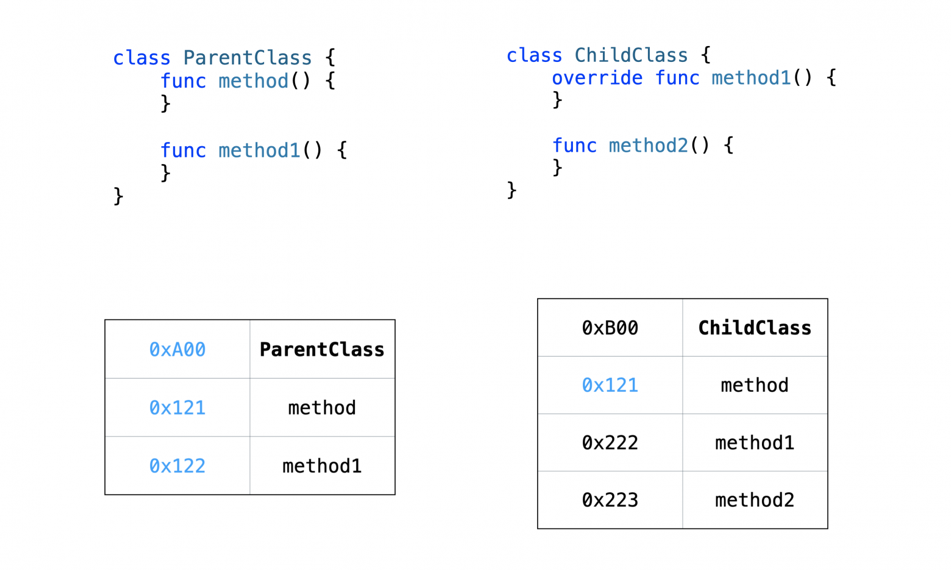
1. Table Dispatch/ Табличная диспетчеризация (динамическая). Включает в себя использование таблиц для определения вызываемого метода.

На этапе компиляции формируется таблица с адресами методов сущности. Адрес метода ищется по таблице во время **runtime.**

Делится на:

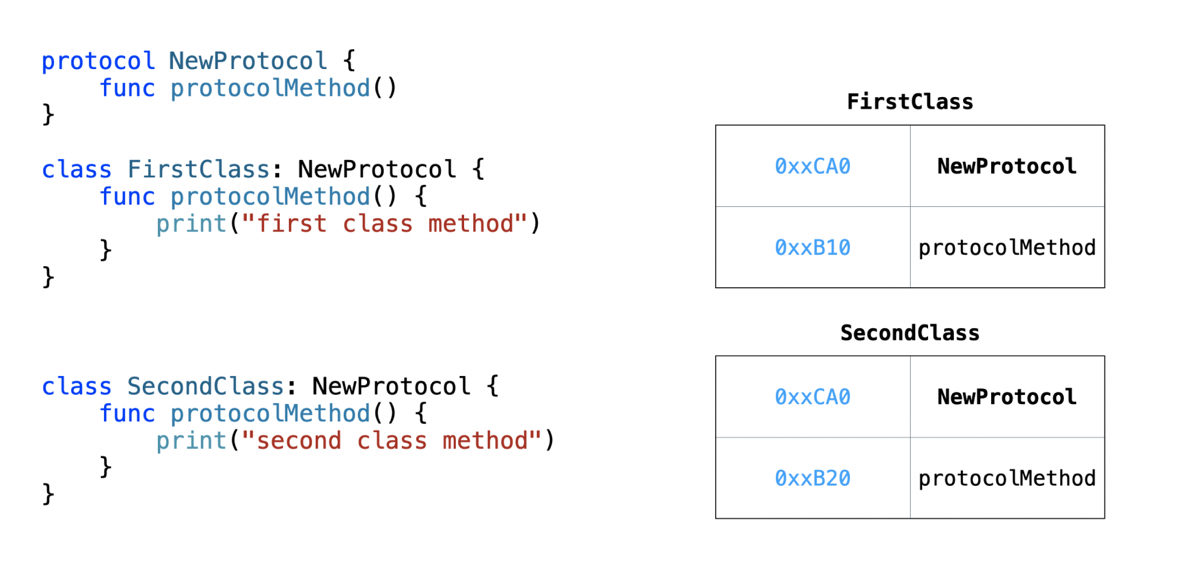
* 1. Virtual Table/Виртуальные таблицы

Данный тип диспетчеризации в основном используется для случаев наследования, то есть для классов. Для каждого класса и для каждого его сабкласса создается виртуальная таблица с адресами методов, при этом для дочернего класса эта таблица копируется вместе с адресами. В случае, когда реализация метода полностью переходит из родительского класса, адрес в таблице наследника такой же (в примере ниже это метод method). Если какой-то метод переопределен через override (метод method1), то адрес меняется на другую реализацию.



* 1. Witness Table

Выделен в Swift для работы с протоколами. В примере ниже для класса, реализующего протокол NewProtocol создается отдельная таблица с адресами методов протокола, и при вызове метода у объекта с типом протокола NewProtocol используется Witness Table диспетчеризация. В случае реализации классом нескольких протоколов, для каждого протокола будет создаваться отдельная таблица. Отличие от Virtual Table заключается в отсутствии наследования.

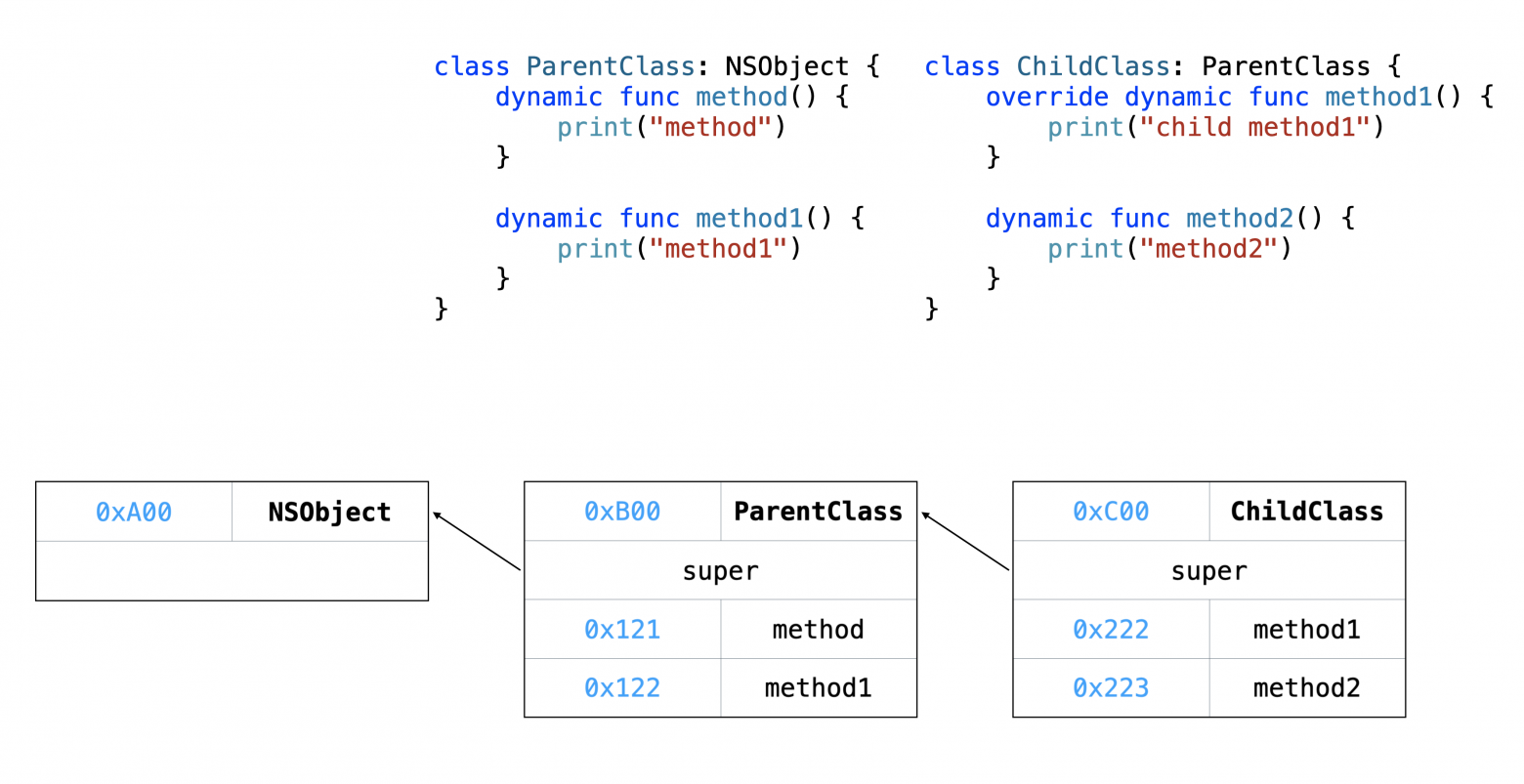


1. Message Dispatch / Диспетчеризация на сообщениях

Обычно используется в связке с Objective-C

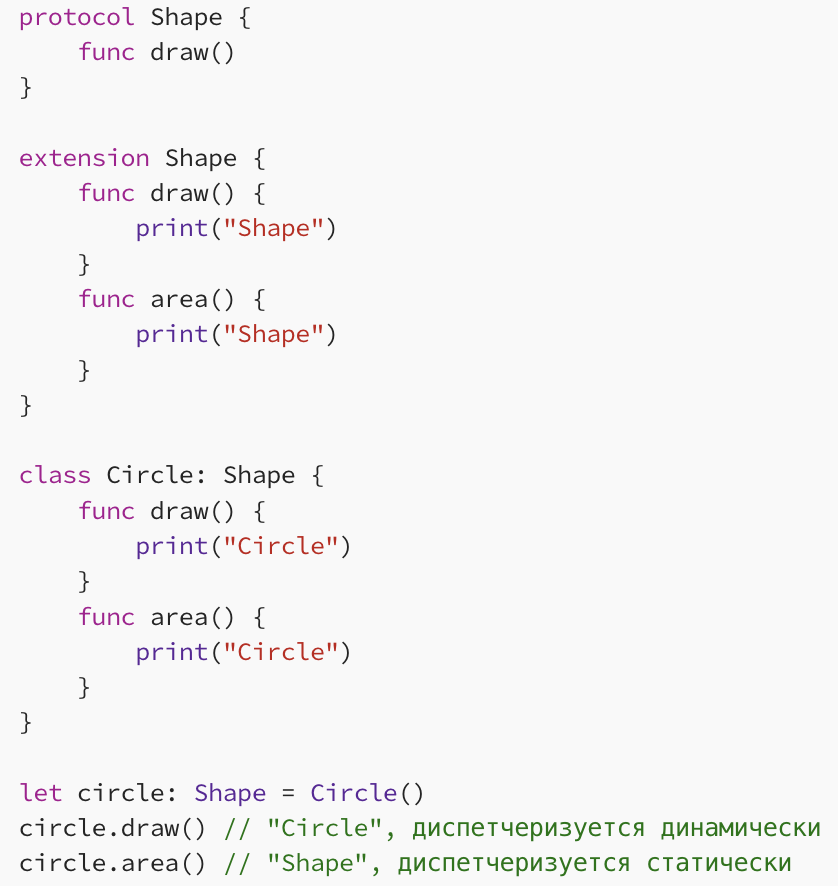
Работает полностью в runtime.

На примере ниже видим, что тоже присутствуют родительский и дочерний классы, но родительский класс при этом наследуется от NSObject. Для работы Message Dispatch, методу добавляется модификатор @objc dynamic. Для каждого класса тоже есть таблицы, но они, в отличии от Table Dispatch, не копируются при наследовании, в таблице дочернего класса есть ссылка super на родительский класс. При вызове метода у объекта дочернего класса, сначала идет поиск адреса у самого класса. Если не адрес не найден, то поиск продолжается у родительского класса, и так далее до NSObject.



// **MARK: Подводные камни**

**Протокол**

****

Здесь метод area диспетчеризуется статически, а еще реализован в классе Circle, но в компиляторе все равно для выполнения выбирается реализация по умолчанию Shape.