Министерство образования и науки России

Рыбинский государственный авиационный технический университет

имени П.А. Соловьева

КАФЕДРА ВЫЧИСЛИТЕЛЬЫХ СИСТЕМ

КОНТРОЛЬНАЯ РАБОТА

По дисциплине «Объектно-ориентированное программирование»

На тему «Динамическое связывание»

Выполнил Ухов Д.Д.

(фамилия, имя, отчество)

студент гр. ЗИС-23

Преподаватель Пруктишина В.А.

(фамилия, имя, отчество)

Оценка \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Подпись преподавателя \_\_\_\_\_\_\_\_\_\_\_\_

Дата \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

г. Рыбинск

2024 г.

**Содержание**

[Введение 3](#_Toc187689532)

[1. Основные концепции ООП 4](#_Toc187689533)

[2. Динамическое связывание: концепция и особенности 4](#_Toc187689534)

[Заключение 8](#_Toc187689535)

[Список использованных источников и литературы 9](#_Toc187689536)

### ****Введение****

Объектно-ориентированное программирование (ООП) представляет собой парадигму, основанную на концепции объектов, которые инкапсулируют данные и методы для их обработки. Важной характеристикой ООП является способность программ взаимодействовать с объектами через динамическое связывание. Это позволяет в процессе выполнения программы изменять поведение объектов и функций, обеспечивая гибкость и адаптивность к изменениям. В Python, как и в других языках ООП, динамическое связывание играет важную роль в обеспечении полиморфизма и работы с методами объектов. В данной работе рассматривается механизм динамического связывания в ООП, а также его использование на примерах с Python.

Для достижения цели поставлены следующие задачи:

1. Изучить основные принципы объектно-ориентированного программирования.
2. Рассмотреть определения динамического связывания.
3. Проанализировать примеры использования динамического связывания на языке Python.

Данная работа, основывается на теоретическом материале и примерах, и поможет лучше понять, как использовать динамического связывания для создания приложений.

### ****Основные концепции ООП****

Перед тем как подробно рассматривать динамическое связывание, важно напомнить основные принципы объектно-ориентированного программирования:

* **Инкапсуляция** — объединение данных и методов, работающих с этими данными, в единый объект.
* **Наследование** — возможность создания новых классов на основе существующих.
* **Полиморфизм** — способность объектов разных типов реагировать на одинаковые сообщения (методы) по-разному.
* **Абстракция** — выделение важных характеристик объекта и скрытие несущественных деталей.

Одним из ключевых аспектов полиморфизма является **динамическое связывание** — механизм, который позволяет вызывать методы объектов в зависимости от их типа, определяя поведение программы во время ее выполнения.

### Динамическое связывание: концепция и особенности

Динамическое связывание (или позднее связывание) — это процесс, при котором решение о том, какой метод нужно вызвать, принимается не на этапе компиляции (как в статическом связывании), а в момент выполнения программы. Это означает, что программа может вызвать метод объекта, даже если конкретный тип этого объекта неизвестен на этапе написания кода.

Динамическое связывание активно используется для реализации полиморфизма. Когда объект вызывает метод, система динамически выбирает нужную реализацию этого метода в зависимости от фактического типа объекта.

Python — это язык с динамической типизацией, что означает, что типы переменных определяются во время выполнения программы. В связи с этим, динамическое связывание является важной частью работы с объектами и методами в Python.

Пример 1: Полиморфизм и динамическое связывание.

|  |
| --- |
| class Animal:  def make\_sound(self):  print("Some generic animal sound")  class Dog(Animal):  def make\_sound(self):  print("Bark")  class Cat(Animal):  def make\_sound(self):  print("Meow")  def animal\_sound(animal):  animal.make\_sound()  # Создание объектов  dog = Dog()  cat = Cat()  # Динамическое связывание: метод вызывается в зависимости от типа объекта  animal\_sound(dog) # Output: Bark  animal\_sound(cat) # Output: Meow |

В данном примере классы Dog и Cat наследуют метод make\_sound от класса Animal, но переопределяют его. Когда вызывается функция animal\_sound, она вызывает метод make\_sound, но фактическое поведение зависит от типа объекта, который был передан в функцию. Это и есть пример динамического связывания: Python определяет, какой метод нужно вызвать, в зависимости от типа объекта, который передан в функцию.

В Python все методы являются виртуальными по умолчанию. Это означает, что методы можно переопределять в подклассах, и правильный метод будет вызван в момент выполнения программы, в зависимости от типа объекта. Это отличается от статического связывания, когда компилятор заранее определяет, какой метод будет вызван.

Пример 2: Виртуальные методы в Python.

|  |
| --- |
| class Vehicle:  def start\_engine(self):  print("Starting engine of vehicle")  class Car(Vehicle):  def start\_engine(self):  print("Starting engine of car")  class Boat(Vehicle):  def start\_engine(self):  print("Starting engine of boat")  # Создание объектов  car = Car()  boat = Boat()  # Вызов методов с динамическим связыванием  car.start\_engine() # Output: Starting engine of car  boat.start\_engine() # Output: Starting engine of boat |

Здесь классы Car и Boat переопределяют метод start\_engine класса Vehicle. Когда метод вызывается для конкретного объекта, Python динамически определяет, какой именно метод следует вызвать, основываясь на типе объекта. Это позволяет создавать гибкие и расширяемые системы, где поведение объектов может быть изменено в зависимости от их типа без необходимости изменения кода.

**Помимо полиморфизма, динамическое связывание в Python также используется в контексте динамической загрузки модулей и привязки методов. В Python можно загружать модули и вызывать методы, не зная заранее их точных имен. Это возможно благодаря встроенным функциям, таким как getattr(), которые позволяют динамически обращаться к атрибутам и методам объектов.**

**Пример 3: Использование getattr() для динамического вызова метода.**

|  |
| --- |
| class Greeting:  def say\_hello(self):  print("Hello!")  def say\_goodbye(self):  print("Goodbye!")  # Создание объекта  greet = Greeting()  # Динамический вызов метода с использованием getattr  method\_name = "say\_hello"  getattr(greet, method\_name)() # Output: Hello!  method\_name = "say\_goodbye"  getattr(greet, method\_name)() # Output: Goodbye! |

В этом примере мы используем функцию getattr(), чтобы динамически получить и вызвать метод объекта Greeting на основе строкового имени метода. Это позволяет работать с объектами и их методами без явного указания всех деталей на этапе написания программы.

### Заключение

Динамическое связывание в объектно-ориентированном программировании является мощным механизмом, который позволяет создавать гибкие, масштабируемые и адаптивные системы. В Python этот механизм реализуется через полиморфизм, виртуальные методы и возможности динамической загрузки, что позволяет программам работать с объектами, не зная их точных типов заранее. Использование динамического связывания дает возможность создавать более эффективные и менее зависимые от жесткой структуры кода программы, улучшая таким образом их расширяемость и поддержку в будущем.

Таким образом, динамическое связывание играет ключевую роль в обеспечении гибкости и функциональной богатстве объектно-ориентированных программ. В Python эта концепция реализована легко и эффективно, что способствует созданию удобных и мощных решений для различных задач.

### Список использованных источников и литературы

1. Статья «Python - Динамическое привязывание», URL: https://ru.hexlet.io/courses/python-classes/lessons/dynamic-dispatch/theory\_unit
2. Книга «Python Подробный справочник», автор: Дэвид Бизли, 2010 г.