Введение в ООП. Часть 2

Методы экземпляров

Методы экземпляра являются ключевым компонентом объектно-ориентированного программирования, предоставляя объектам определенное **поведение** в дополнение к их состоянию. В отличие от статических методов, которые используют модификатор **static,** методы экземпляра вызываются на уровне конкретного объекта и могут взаимодействовать с его полями и другими методами. Таким образом, методы экземпляра могут использовать и модифицировать данные, хранящиеся в объекте, обеспечивая тем самым динамическое поведение объектов в зависимости от их текущего состояния.

```
class Car {

    double fuelVolume = 0; // объем топлива в литрах
    double maxFuelVolume; // объем бака
    double mileage = 0; // пробег в километрах
    double fuelConsumption; // расход топлива на 100 км

    Car(double maxFuelVolume, double fuelConsumption) {
        this.maxFuelVolume = maxFuelVolume;
        this.fuelConsumption = fuelConsumption;
    }

    // метод для заправки автомобиля
    void refuel(double liters) {
        if (fuelVolume + liters > maxFuelVolume) {
            fuelVolume = maxFuelVolume;
        } else {
            fuelVolume += liters;
        }
    }
}
```

В примере с классом Car, представлены несколько полей, описывающих состояние автомобиля: `fuelVolume` (объем топлива), `maxFuelVolume` (максимальный объем топливного бака), `mileage` (пробег) и `fuelConsumption` (расход топлива на 100 км). Конструктор класса инициализирует некоторые из этих полей, предоставляя основу для дальнейшей работы с объектами `Car`.

Метод экземпляра `refuel`, определенный в классе `Car`, демонстрирует типичное поведение, которое может быть реализовано с использованием методов экземпляра. Этот

метод позволяет заправлять автомобиль определенным количеством литров топлива. Внутри метода происходит проверка, чтобы убедиться, что после заправки объем топлива не превышает максимально возможный для топливного бака. Если добавляемое количество топлива приведет к переполнению, объем топлива устанавливается на максимально возможный уровень. В противном случае к текущему объему топлива добавляется указанное количество литров.

```
Car polo = new Car(45, 6);
polo.refuel(10); // вызов метода объекта
```

Вызов метода `refuel` для объекта `polo` иллюстрирует, как методы экземпляра используются для взаимодействия с конкретными объектами. Каждый объект класса `Car` может иметь различные значения своих полей, и методы экземпляра позволяют управлять этими значениями индивидуально для каждого объекта, изменяя его состояние в соответствии с выполненными действиями. Такой подход обеспечивает гибкость и мощь объектно-ориентированного программирования, позволяя создавать сложные системы с динамически взаимодействующими компонентами.

Методы с возвратом значения

Методы не только могут выполнять определенные действия с объектами, но и **возвращать значения**, предоставляя тем самым возможность для взаимодействия с **вызывающим** кодом. Эта возможность делает методы гибкими инструментами для обработки данных, выполнения вычислений и принятия решений в зависимости от текущего состояния объекта.

```
class Car {
   // …
   boolean go(double kilometers) {
   // Рассчитываем, сколько топлива потребуется
      double fuelNeeded =
         (kilometers * fuelConsumption) / 100;

   // Проверяем, хватит ли топлива для поездки
      if (fuelVolume >= fuelNeeded) {
         fuelVolume -= fuelNeeded; // Уменьшаем объем
         mileage += kilometers; // Увеличиваем пробег
         return true; // Поездка завершилась успешно
      } else {
         return false; // Не хватило топлива для поездки
      }
   }
}
```

```
Car polo = new Car(45, 6);
polo.refuel(10);

if (polo.go(5)) { // вызов метода объекта
    System.out.println("Успешная поездка");
} else {
    System.out.println("Что-то пошло не так…");
}
```

В примере с классом **`Car`**, представлен метод **`go`**, который демонстрирует использование возвращаемых значений. Метод **`go`** принимает в качестве параметра количество километров, которое автомобиль должен проехать, и возвращает логическое значение (**`boolean`**), указывающее на успех или неудачу попытки совершить поездку.
Внутри метода сначала рассчитывается, сколько топлива потребуется для поездки на указанное расстояние. Затем производится проверка, хватит ли в баке топлива для этой поездки. Если топлива достаточно, метод уменьшает объем топлива в баке на расчетное количество, увеличивает пробег автомобиля и возвращает значение **`true`**, сигнализируя об успешном завершении поездки. В противном случае, если топлива не хватает, метод возвращает **`false`**, указывая на то, что поездка не может быть совершена.

Вызов метода **`go`** для объекта **`polo`** и последующая проверка возвращаемого значения позволяют вызывающему коду адекватно реагировать на результаты выполнения метода. В данном случае, если метод возвращает **`true`**, на консоль выводится сообщение об успешной поездке, в противном случае сообщается о неудаче.

Таким образом, использование возвращаемых значений методами экземпляра класса позволяет не только изменять состояние объекта, но и предоставлять информацию о результате выполненных операций, что делает взаимодействие с объектами более контролируемым и предсказуемым. Это значительно расширяет возможности программирования, позволяя разработчикам создавать сложные логические структуры и алгоритмы обработки данных.

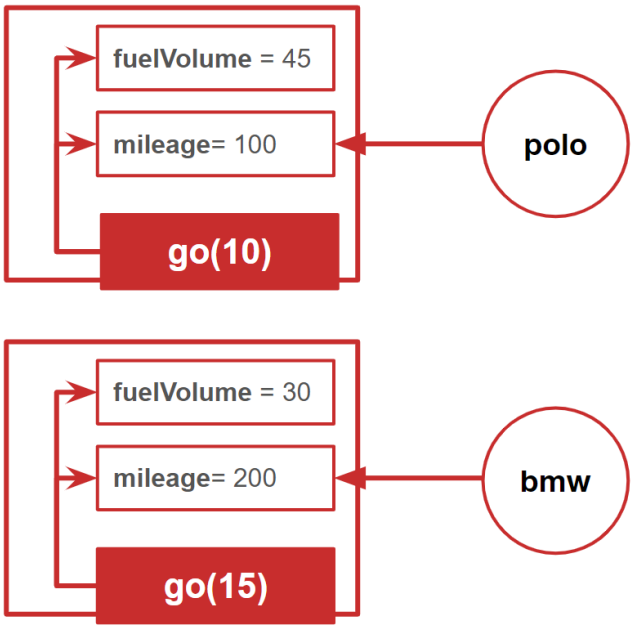
Методы и состояние объектов

Методы, применяемые к объектам, играют центральную роль в управлении их состоянием. Эти методы позволяют не только выполнять операции, связанные с данными объекта, но и изменять его внутренние поля, тем самым воздействуя на состояние объекта. Особенностью таких методов является то, что они взаимодействуют исключительно с состоянием объекта, на котором были вызваны, не затрагивая состояние других объектов того же класса.

```
Car polo = new Car(45, 6);
Car bmw = new Car(68, 6.5);

polo.go(10);
bmw.go(15);
```

Пример с автомобилями `polo` и `bmw`, созданными из класса `Car`, наглядно демонстрирует эту концепцию. Каждый объект имеет свои индивидуальные характеристики, такие как объем топливного бака и расход топлива, заданные при их создании. Когда метод `go` вызывается для каждого из этих автомобилей с различными параметрами расстояния, влияние этого метода ограничивается только тем объектом, у которого он был вызван. Так, выполнение метода go для объекта polo воздействует на его собственные поля `fuelVolume` и `mileage`, изменяя их в соответствии с переданным расстоянием, аналогично и для объекта `bmw` — изменения касаются только его состояния.



Перегруженные методы

Перегрузка методов предоставляет разработчикам гибкий инструмент для создания нескольких версий метода с одинаковым именем, но различающихся набором параметров. Это позволяет методам выполнять схожие или связанные действия, адаптируясь к разным условиям вызова, что делает код более читабельным и удобным для использования.

```
class Car {
    // перегруженные методы для изменения
    // показаний пробега
   boolean adjustOdometer(int km) {
       if (mileage >= km) {
           mileage -= km;
           return true;
       } else {
           return false;
       }
    }

    boolean adjustOdometer() {
       mileage = 0;
       return true;
    }
}

Car polo = new Car(45, 6);
polo.refuel(30);
polo.go(100);

// уменьшить пробег на 50 км
polo.adjustOdometer(50);

// обнулить пробег
polo.adjustOdometer();
```

В примере с классом `Car`, демонстрируется использование перегруженных методов `adjustOdometer` для изменения показаний пробега автомобиля. Первая версия метода принимает целочисленный параметр `km`, который указывает, на сколько километров нужно уменьшить пробег. Этот метод проверяет, достаточно ли пробега для уменьшения, и если да, то уменьшает показания пробега на указанное количество километров, возвращая при этом `true` как знак успешной операции. Если же пробег меньше указанного для уменьшения значения, метод возвращает `false`, указывая на невозможность выполнения операции.

Вторая версия метода `adjustOdometer` не принимает никаких параметров и служит для обнуления показаний пробега. Эта версия всегда возвращает `true`, поскольку операция обнуления не зависит от текущего состояния пробега и всегда может быть выполнена.

Пример использования этих методов с объектом `polo` показывает, как перегруженные методы могут быть вызваны для выполнения различных действий в зависимости от требований конкретной ситуации.

Такой подход к перегрузке методов позволяет создавать универсальные и гибкие интерфейсы в классах, упрощая их использование и повышая эффективность разработки программ.