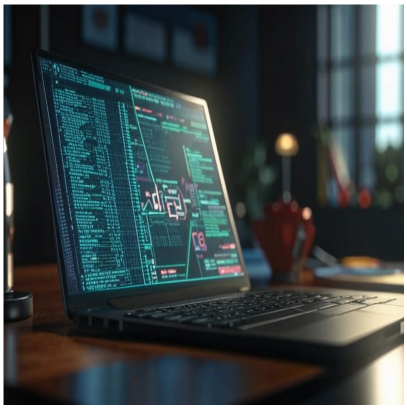


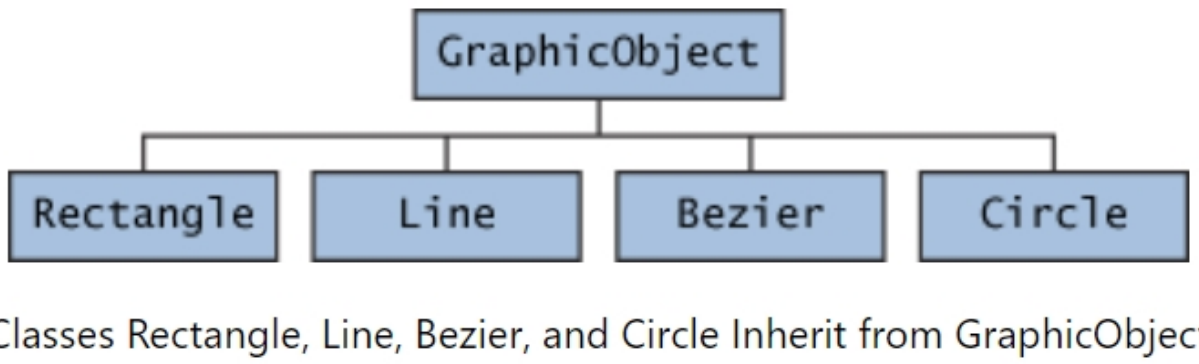
Абстрактный класс и интерфейс в Java с примерами

Описание

Абстрактные классы и интерфейсы необходимы для достижения абстракции в Java. Абстракция в объектно-ориентированном программировании означает сосредоточение на основных характеристиках объекта при стратегическом сокрытии базовых деталей реализации. Это позволяет вам определить, что *делает* объект, не раскрывая специфики того *, как* он это делает. Давайте рассмотрим каждый из них и поймем, почему они используются.

Что такое абстрактный класс

Класс, который не может быть инстанцирован как объект и может содержать или не содержать абстрактные методы, в Java называется абстрактным классом. Абстрактный метод – это метод, который при объявлении не имеет тела реализации.



Classes Rectangle, Line, Bezier, and Circle Inherit from GraphicObject

Пример абстрактного класса GraphicObject – Oracle

Вы можете создать абстрактный класс, указав ключевое слово `abstract` перед ключевым словом `class`.

```
abstract class abstractClass {
    void run() {
        System.out.println("ran");
    }
}
```

Абстрактный класс может быть расширен другими классами. Другими словами, он может быть подклассом.

```
abstract class AbstractClass {
    void run() {
        System.out.println("ran");
    }
}

class ExtendingAbstractClass extends AbstractClass {
    void newMethod() {
        System.out.println("new");
    }

    @Override
    void run() {
        System.out.println("override");
    }
}
```

Абстрактные классы используются для реализации общих методов для нескольких классов, которые расширяют данный абстрактный класс. Кроме того, возможность определять абстрактные методы внутри абстрактных классов делает их невероятно полезными для классов, у которых есть похожие методы, но с разными реализациями. Рассмотрим пример. Рассмотрим автомобиль, который обладает некоторыми функциональными возможностями, такими как запуск, остановка, задний ход и т. д. Эти функции являются общими для всех типов автомобилей. Но как быть с функциями автоматизации, такими как самостоятельное вождение? Реализация этих функций может отличаться для разных типов автомобилей. Давайте посмотрим, как можно создать объектно-ориентированную программу, связанную с этим. Прежде всего, создайте класс `Car`, который будет расширяться несколькими классами различных типов автомобилей.

```
abstract class Car {
    void start() {
        // implementation
        System.out.println("runs car");
    }
```

```
    void stop() {
        // implementation
        System.out.println("engine stops");
    }

    void reverse() {
        // implementation
        System.out.println("reverse mode enabled");
    }

    abstract void selfDrive();
}
```

Методы `start()`, `stop()` и `reverse()` – это методы, которые являются общими для всех автомобилей. Поэтому их реализация уже определена внутри самого класса Car. Однако у определенного типа автомобиля могут быть разные реализации режима самодвижения. Таким образом, вы можете определить `selfDrive()` как абстрактный метод и реализовать его по-разному в разных классах разных типов автомобилей.

```
class CarTypeA extends Car {
    @Override
    void start() {
        super.start();
    }

    @Override
    void stop() {
        super.stop();
    }

    @Override
    void reverse() {
        super.reverse();
    }

    void selfDrive() {
        // custom implementation
        System.out.println("Type A self driving mode enabled");
    }
}
```

```
class CarTypeB extends Car {
    // ...all similar methods

    void selfDrive() {
        // custom implementation
        // different implementation than CarTypeB
        System.out.println("Type B self driving mode enabled");
    }
}
```

Важно отметить, что если подкласс не реализует все абстрактные методы, определенные в абстрактном классе, то он сам должен быть объявлен как абстрактный класс.

Что такое интерфейс

Интерфейс – это способ указать классу, какие методы должны быть реализованы в нем. Например, если рассмотреть пример с автомобилем, то у него есть несколько основных функций. Он может заводиться, двигаться и останавливаться. Эти функции присущи всем автомобилям. Так, если вы реализуете интерфейс автомобиля в классе, вы должны реализовать все его методы, чтобы автомобиль функционировал правильно и безопасно. Как и в случае с абстрактными классами, мы не можем инстанцировать или создавать объекты интерфейса. Его можно считать полностью абстрактным классом, поскольку он содержит только абстрактные методы, то есть методы без тела реализации. Вы можете создать интерфейс с помощью ключевого слова `interface`.

```
interface CAR {
    void start();
    void stop();
    void move();
}
```

Реализуйте интерфейс, используя ключевое слово `implements` при определении класса.

```
class CarTypeB implements CAR {
    public void start() {
        System.out.println("started");
    }

    public void stop() {
        System.out.println("stopped");
    }

    public void move() {
```

```
        System.out.println("running");
    }
}
```

Сходство абстрактных классов и интерфейсов

Отсутствие инстанцирования в качестве объекта – это единственное, что объединяет абстрактные классы и интерфейсы.

Различия между абстрактными классами и интерфейсами

Приведенная ниже сводная таблица даст вам краткое представление о том, чем отличаются абстрактный класс и интерфейс.

	Абстрактный класс	**Интерфейс**
Наследование и реализация	Только один абстрактный класс может быть унаследован классом.	В одном классе может быть реализовано несколько интерфейсов.
Типы переменных	В нем могут быть конечные, не конечные, статические и нестатические переменные.	В нем могут быть только статические и финальные переменные.
Типы методов	Он может содержать как абстрактные, так и неабстрактные методы.	Он может содержать только абстрактные методы, но статические методы являются исключением.
Модификаторы доступа	Абстрактный класс может иметь модификатор доступа.	Сигнатуры методов, определенных в интерфейсе, по умолчанию являются общедоступными. Интерфейс не имеет модификатора доступа.

Конструкторы и деструкторы	В нем можнообъявлять конструкторы и деструкторы.	Он не может объявлять конструкторы или деструкторы.
Скорость	Быстрый	Медленный

Различия между абстрактным классом и интерфейсом

Когда использовать абстрактный класс и интерфейс?

Используйте абстрактные классы, когда:

- Вы хотите разделить некоторые общие методы и поля между несколькими классами.
- Объявление ????????????? и ??????????? полей для изменения состояния объекта, к которому они привязаны.

Вы можете использовать интерфейсы, когда:

- Вы хотите определить поведение класса, реализующего интерфейс, но вас не волнует, каким образом он будет реализован.
- Вы хотите убедиться, что класс реализует все методы для правильного функционирования.

Заключение

Интерфейсы в основном используются для создания API, потому что они могут обеспечить структуру для реализации функциональности, не беспокоясь о фактической реализации. Абстрактные классы обычно используются для совместного использования общих абстрактных и неабстрактных методов несколькими классами, которые расширяют абстрактный класс, чтобы сделать код более пригодным для повторного использования.

Дата Создания
19.05.2024