Klasy abstrakcyjne

Klasy abstrakcyjne w odróżnieniu od zwykłych klas, nie mogą mieć swoich obiektów, tj. nie możemy stworzyć obiektu takiej klasy za pomocą konstruktora. Klasy abstrakcyjne służą raczej jako baza dla innych klas. Nie można utworzyć obiektu klasy abstrakcyjnej, ale można dziedziczyć po niej i rozszerzać ją w innych klasach.

Klasy abstrakcyjne są bardzo przydatne, gdy mamy do czynienia z grupą obiektów, które mają wiele wspólnych cech. Abstrakcyjna klasa bazowa ma jedynie umożliwiać zdefiniowanie zachowań i cech wspólnych. Na przykład, mamy klasę abstrakcyjną User, która reprezentuje ogólnego użytkownika systemu do logowania na uczelni. Klasy Student i Professor dziedziczą po User i rozszerzają go o dodatkowe funkcjonalności, odpowiednie dla studenta i profesora. Dzięki temu możemy wyodrębnić wspólne cechy użytkowników systemu i jednocześnie zapewnić unikalne funkcjonalności dla studentów i profesorów. User w takim przypadku jest tylko bazową klasą pomocniczą i nigdy nie będziemy chcieli tworzyć obiektów tej klasy bezpośrednio. Dlatego możemy z niej zrobić klasę abstrakcyjną i tym samym “zablokować” możliwość tworzenia obiektów tej klasy.

Aby utworzyć klasę abstrakcyjną w Javie, należy użyć słowa kluczowego abstract, np. abstract class User. W klasie abstrakcyjnej możemy zdefiniować zmienne i metody, tak jak w zwykłej klasie. Jednak w przeciwieństwie do zwykłej klasy, w klasie abstrakcyjnej możemy również zdefiniować metody abstrakcyjne.
Metody abstrakcyjne są to metody, które nie posiadają implementacji w klasie abstrakcyjnej, ale muszą być zaimplementowane w klasach potomnych. Oznaczamy je za pomocą słowa kluczowego abstract, np. abstract void login(). Klasy potomne, które dziedziczą po abstrakcyjnej klasie, muszą zaimplementować wszystkie abstrakcyjne metody swojej klasy nadrzędnej. W ten sposób możemy uniknąć sytuacji, w której klasa dziedzicząca nie posiada wszystkich cech swojej klasy nadrzędnej.

Interfejsy

Interfejsy są podobne do klas, ale nie mogą zawierać implementacji metod, a jedynie ich sygnaturę (wyjątkiem są metody ze specyfikatoremdefault). Interfejsy służą jako wzorzec do implementacji, ale nie posiadają własnej implementacji.
Interfejsy są używane w celu zapewnienia, że klasy, które implementują dany interfejs, posiadają wymagane metody i pola. Dzięki temu możemy być pewni, że klasy te będą działać w określony sposób i będą posiadać wymagane funkcjonalności. Aby stworzyć interfejs w Javie, należy użyć słowa kluczowego interface, na przykład public interface Employee.

Interfejsy mogą zawierać zarówno metody abstrakcyjne, jak i metody, które mają implementację. Metody abstrakcyjne nie posiadają implementacji (jedynie sygnaturę) i muszą być zaimplementowane przez klasy, które implementują dany interfejs. W klasach abstrakcyjnych takie metody musiały być oznaczone słówkiem abstract, w interfejsach nie muszą.
Metody z implementacją to na przykład metody domyślne oznaczone słówkiem default. Inne metody, które mogą mieć implementację w interfejsie to metody statyczne oraz niestatyczne metody prywatne. Interfejsy nie mogą zawierać konstruktorów, niepublicznych metod abstrakcyjnych oraz pól innych niż public static final (chociaż wcale nie musimy pól oznaczać tymi słówkami, po prostu będą takie domyślnie).
Aby klasa mogła implementować dany interfejs, należy użyć słowa kluczowego implements i określić nazwę interfejsu, który ma być implementowany, na przykład public class OfficeWorker implements Employee. Każda klasa może implementować dowolną ilość interfejsów. Interfesy mogą z kolei rozszerzać dowolną ilość interfejsów. Do rozszerzania interfejsów przez inne interfejsy używamy słówka extends, np. public interface Employee extends User.

Klasy abstrakcyjne i interfejsy: różnice

- Klasa abstrakcyjna może mieć abstrakcyjne i nieabstrakcyjne metody instancyjne, podczas gdy interfejs może mieć abstrakcyjne albo domyślne metody instancyjne.
- Klasa abstrakcyjna może rozszerzać inną klasę abstrakcyjną lub nie abstrakcyjną, podczas gdy interfejs może rozszerzać tylko interfejsy.
- Interfejs pozwala na wielokrotne dziedziczenie.
- Klasa abstrakcyjna może mieć zmienne statyczne, niestatyczne, final, czy nie, a interfejs może mieć tylko public final static.
- Klasa abstrakcyjna może implementować interfejs, ale interfejs nie może implementować klasy abstrakcyjnej.
- Klasa abstrakcyjna może mieć konstruktor, interfejs nie.

- W klasie abstrakcyjnej słówko abstract jest obowiązkowe jeżeli chcemy zadeklarować metodę abstrakcyjną, w interfejsie jest to opcjonalne.

Klasa abstrakcyjna	Interfejs
Dziedziczenie	
Klasa abstrakcyjna może dziedziczyć tylko jedną klasę, ale może za to dziedziczyć dowolną liczbę interfejsów.	Interfejs nie może dziedziczyć klas, może za to dziedziczyć dowolną liczbę interfejsów.
Metody abstrakcyjne	
Klasa abstrakcyjna może zawierać metody abstrakcyjne. Ale może także nie zawierać żadnych.	Wszystkie niestatyczne i niedomyślne metody interfejsu są abstrakcyjne, tj. nie mają żadnych konkretnych implementacji. Interfejs może nie posiadać żadnych metod.
Metody z implementacją	
Klasa abstrakcyjna może zawierać metody z implementacją.	Interfejs może zawierać metody domyślne.
Dane	
Bez ograniczeń.	Interfejs zawiera tylko publiczne, finalne dane statyczne.
Tworzenie obiektu	
Nie możesz tworzyć instancji klasy abstrakcyjnej.	Nie możesz tworzyć instancji interfejsu.

```java
public interface Vehicle {

    String getBrand();

    String speedUp();

    String slowDown();

    default String turnAlarmOn() {
        return "Turning the vehicle alarm on.";
    }

    default String turnAlarmOff() {
        return "Turning the vehicle alarm off.";
    }
}
```

```java
public class Car implements Vehicle {

    private String brand;

    // constructors/getters

    @Override
    public String getBrand() {
        return brand;
    }

    @Override
    public String speedUp() {
        return "The car is speeding up.";
    }

    @Override
    public String slowDown() {
        return "The car is slowing down.";
    }
}
```

```java
public static void main(String[] args) {
    Vehicle car = new Car("BMW");
    System.out.println(car.getBrand());
    System.out.println(car.speedUp());
    System.out.println(car.slowDown());
    System.out.println(car.turnAlarmOn());
    System.out.println(car.turnAlarmOff());
}
```