Klasy zagnieżdżone

Spis treści

Zapiski prowadzącego Karola Rogowskiego i uczestnika Bootcampu Zajavka Bartek Borowczyk aka Samuraj Programowania.

Czym są klasy zagnieżdżone?

Czyli klasy definiowane w klasie. Taka matrioszka. Możemy zdefiniować klasę, w klasie, w klasie, w klasie itp.

Występuje tutaj jedno rozróżnienie:

- **Nested class** (klasa zagnieżdżona) - klasa zdefiniowana w innej klasie
- **Inner class** (klasa wewnętrzna) - nested class, która nie jest statyczna (ktoś tak wymyślił nazewnictwo, w praktyce i tak na wszystko mówi się klasa zagnieżdżona)

Kiedy się tego używa, jak nie chcesz, żeby definicja klasy była dostępna na zewnątrz, czyli chcesz zdefiniować klasę, tylko na potrzeby innej Twojej klasy. Czyli chcesz mieć klasę o scopie prywatnym.

Występują 4 typy klas zagnieżdżonych.

Member inner class

Member inner class (członkowa klasa wewnętrzna? wolę angielskie nazwy...) - która nie jest statyczna. Jest definiowana na tym samym poziomie co pola w klasie, metody oraz konstruktory. Najczęściej nazywana po prostu klasą zagnieżdżoną (*inner class*). Może być `public`, `private`, `protected` albo `default`. Może rozszerzać inne klasy i implementować interfejsy. Może być abstract i może być `final`. Nie może natomiast definiować statycznych pól ani metod. Może odnosić się do pól i metod prywatnych klasy, która ją opakowuje, czyli klasy, w której jest zdefiniowana (*outer class*).

Przykład

```
public class InnerMain {

    private final String hello = "hello";

    private int a = 10;

    public static void main(String[] args) {
        InnerMain main = new InnerMain();
        main.call();

        // error: cannot be referenced from static context
        // Nie możemy stworzyć instancji klasy zagnieżdżonej
        // bezpośrednio w metodzie statycznej,
        // później pokażę jak to obejść
        // Inner inner = new Inner();
        // inner.call();
    }

    public void call() {
        // Możemy stworzyć instancję klasy zagnieżdżonej
        Inner inner = new Inner("innerHello");
        inner.call();

        System.out.println();
        System.out.println(inner.innerHello);
    }

    class Inner {
        private String innerHello;

        private int a = 20;

        // Nie może mieć ani statycznych pól ani metod
        // public static String abc;

        // public static void staticMethod() {}

        public Inner(final String innerHello) {
            this.innerHello = innerHello;
        }

        public void call() {
            // możemy odwołać się do pola prywatnego w klasie InnerMain
            // możemy również używać słówka this w odniesieniu do klasy InnerMain
            System.out.println(InnerMain.this.hello + " " + innerHello);
        }
    }
}
```

Local inner class

Klasa zdefiniowana w metodzie. Taka konstrukcja jest możliwa, chociaż w praktyce jest stosowana bardzo rzadko. Klasa taka nie ma określonego modyfikatora dostępu. Nie może również być zadeklarowana jako statyczna i nie może jednocześnie deklarować statycznych pól ani metod. Może odnosić się do wszystkich pól i metod klasy, która jest jej właścicielem (*outer class*). Może mieć dostęp do

zmiennych metody, w której została stworzona pod warunkiem, że są one zdefiniowane jako **final**, lub są **effectively final** (mówiliśmy o tym przy lambdach).

Przykład

```
public class Main {

    private String hello = "hello";

    public static void main(String[] args) {
        Main main = new Main();
        main.call();

        // Nie możemy stworzyć instancji z metody statycznej
        // Inner inner = new Inner();
    }

    public void call() {
        final String helloFromMethod = "helloFromMethod";

        // Nie możemy też korzystać z tej klasy wyżej niż jest zdefiniowana.
        // Nie możemy też z niej korzystać poza obrębem metody call()
        // Inner inner = new Inner();
        // inner.call();

        class LocalInner {
            public void call() {
                System.out.println(Main.this.hello + helloFromMethod);
            }
        }

        LocalInner inner = new LocalInner();
        inner.call();
    }

    public void call2() {
        // ten kod się nie skompiluje,
        // bo klasa ta jest zdefiniowana w innej metodzie
        // LocalInner inner = new LocalInner();
        // inner.call();
    }
}
```

Anonymous inner class (klasa anonimowa)

Klasa, której implementację piszemy w locie, nie tworzymy nawet dla niej oddzielnego pliku, nie ma też nadanej nazwy, dlatego jest anonimowa. Jej deklaracja występuje od razu w miejscu, gdzie potrzebujemy jej użyć. Możemy ją napisać jako klasa dziedzicząca inną klasę albo implementująca jakiś interface tu i teraz. Klasy anonimowe nie mogą jednocześnie rozszerzać klasy i implementować interfejsu. Można o niej myśleć jak o nienazwanej klasie lokalnej (*local inner class*).

Przykład

```
public class AnonymousMain {

    private String generalName = "Romański";

    public static void main(String[] args) {
        AnonymousMain main = new AnonymousMain();
        main.call();
    }

    private void call() {
        /*
        wywołując metodę singASong() możemy przekazać implementację interfejsu Singable na 3 sposoby:
        - tworząc klasę w pliku, która implementuje ten interface,
            stworzyć jej instancję i przekazać jako argument tej metody
        - wykorzystać klasę anonimową co widać poniżej,
        - użyć lambdy
        */

        // Konstrukcja klasy anonimowej wygląda w ten sposób, jakbyśmy wołali jej konstruktor,
        // ale otwieramy nawias klamrowy jak w definicji klasy i normalnie piszemy definicję takiej klasy
        // tzn. musimy zaimplementować metody wymagane przez implementowany interface
        Singable singable = new Singable() {
            @Override
            public String singASong(final String songName) {
                return "I'm singing a song " + songName;
            }
        }; // tu musi być średnik
        System.out.println(singASong("Merry Christmas", singable));

        // zamiast tworzenia klasy anonimowej jak powyżej, można też użyć lambdy
        System.out.println(singASong("Merry Christmas", songName -> "I'm singing a song " + songName));
    }

    // metoda przyjmuje interface Singable jako argument
    public String singASong(String songName, Singable singable) {
        return singable.singASong(songName);
    }

    // interface tak jak klasa, też może być w klasie
    interface Singable {
        String singASong(String songName);
    }

}
```

Z tym używaniem lambdy zamiast klasy anonimowej jest związanych kilka ciekawych kwestii:

Tabela 1. Porównanie klas anonimowych i lambdy

Klasa anonimowa	Lambda
klasa anonimowa może implementować interfejsy, które mają więcej niż jedną metodę	lambda może implementować tylko interfejsy funkcyjne, czyli takie, które mają tylko jedną metodę
klasa anonimowa może rozszerzać klasy abstrakcyjne	lambda nie może rozszerzać/implementować klas abstrakcyjnych

Static nested class

Statyczna klasa zagnieżdżona, definiowana na tym samym poziomie co zmienne statyczne. Ważna kwestia tutaj jest taka, że obiekt klasy Member inner class może być utworzony tylko, jeżeli został utworzony obiekt klasy dla niej zewnętrznej (outer class). W przypadku *static nested class* możemy obiekt takiej klasy zagnieżdżonej utworzyć bez tworzenia obiektu klasy zewnętrznej. Z racji, że jest statyczna, to nie może mieć dostępu do pól instancyjnych klasy zewnętrznej. Może być `public`, `private`, `protected` lub `default`. Klasa zewnętrzna może się odnosić do jej pól lub metod.

Przykład

```
public class StaticMain {

    public static class Nested {
        private static String staticHello = "staticHello";

        private String hello = "hello";

        public static void staticMethod() {
            System.out.println("staticMethod");
        }

        public void method() {
            System.out.println("method: " + hello);
        }
    }

    public static void main(String[] args) {
        // W ten sposób możemy odwołać się do metody klasy statycznej
        // Nie musimy tworzyć obiektu klasy StaticMain żeby odwołać się do klasy Nested.
        StaticMain.Nested.staticMethod();
    }
}
```

Klasa zagnieżdżona a kompilacja

Po skompilowaniu kodu z klasami zagnieżdżonymi wejdź do katalogu `out` i poszukaj plików `.class`. Przypomnę tylko, że pliki `.class` reprezentują nasze klasy napisane w kodzie, tyle że są skompilowane. Zwróć teraz uwagę, w jaki sposób wyglądają pliki odpowiedzialne za klasy zagnieżdżone.

Zaznaczę tutaj, że trzeba to zrobić inaczej, niż wykorzystując Intellij, czyli np. przez explorator Windows. Intellij automatycznie inaczej reprezentuje pliki `.class`, więc nie zobaczymy tutaj, co chcę pokazać.

Jeżeli znajdziesz teraz takie pliki `.class` to zobaczysz, że wyglądają one np. w ten sposób:

```
AnotherClass.class
Main$Nested$NestedNested.class
Main$Nested.class
Main.class
```

Dla każdej klasy zagnieżdżonej Java na etapie kompilacji tworzy oddzielny plik, gdzie klasy zagnieżdżone w swojej nazwie będą miały dodany znaczek $.