Programowanie w Java Rozszerzenie

Łukasz Rybka · Gdańsk 2016/17

Klasy wewnętrzne

Klasy wewnętrzne

Klasa wewnętrzna to taka, która została zdefiniowana w ciele innej klasy

Klasy wewnętrzne

Klasa wewnętrzna to taka, która została zdefiniowana w ciele innej klasy

Klasy wewnętrzne posiadają "specjalny łącznik" dający im dostęp do pól klasy zewnętrznej

Klasy wewnętrzne

Klasa wewnętrzna to taka, która została zdefiniowana w ciele innej klasy

Klasy wewnętrzne posiadają "specjalny łącznik" dający im dostęp do pól klasy zewnętrznej

Odwołania do klasy zewnętrznej za pomocą .this oraz .new

Klasy wewnętrzne

Klasa wewnętrzna to taka, która została zdefiniowana w ciele innej klasy

Klasy wewnętrzne posiadają "specjalny łącznik" dający im dostęp do pól klasy zewnętrznej

Odwołania do klasy zewnętrznej za pomocą .this oraz .new

Aby utworzyć obiekt klasy wewnętrznej, konieczny jest obiekt klasy zewnętrznej!

Klasy wewnętrzne - przykład

```
package pl.org.dragonia.oopl;

public class DoThis {
    void f() {
        System.out.println("DoThis.f()");
    }

    public class Inner {(1)
        public DoThis outer() {
            return DoThis.this;(2)
        }
    }

    public Inner inner() {
        return new Inner();(3)
    }
}
```

(1) Definicja klasy wewnętrznej

(2) Zwrócenie referencji do obiektu klasy zewnętrznej z obiektu klasy wewnętrznej

(3) Stworzenie obiektu klasy wewnętrznej

Klasy wewnętrzne - przykład wykorzystania

```
package pl.org.dragonia.oopl;

public class Main {
    public static void main(String[] args) {
        DoThis dt = new DoThis();(1)
        DoThis.Inner dtinner = dt.inner();(2)
        dtinner.outer.f();(3)

        DoThis.Inner inner = dt.new Inner();(4)
    }
}
```

1. Stworzenie obiektu klasy zewnętrznej
2. Stworzenie obiektu klasy wewnętrznej
3. Wywołanie metody obiektu klasy zewnętrznej z referencji w klasie wewnętrznej
4. Stworzenie obiektu klasy wewnętrznej z użyciem operatora **.new**

Nowości dotyczące interfejsów w Java 8

Nowości dotyczące interfejsów w Java 8

Domyślne (default) implementacje metod

Nowości dotyczące interfejsów w Java 8

Domyślne (default) implementacje metod

Metody statyczne

Nowości dotyczące interfejsów w Java 8

Domyślne (default) implementacje metod

Metody statyczne

Nowe problemy…

Metody domyślne

```
public interface Interface1 {
    void method1(String str);

    default void log(String str) {
        System.out.println("I1 logging::"+str);
        print(str);
    }
}

public class Impl1 implements Interface1 {
    @Override
    void method1(String str) {
        // ...
    }

    public static void main(String[] args) {
        Impl1 impl = new Impl1();

        impl.method1();
        impl.log("Something...");
    }
}
```

Metody domyślne

```
public interface Interface2 {
    void method2();

    default void log(String str) {
        System.out.println("I2 logging::"+str);
    }

}

public class Impl1 implements Interface1, Interface2 {
    @Override void method1(String str) { /** ... */ }
    @Override void method2(String str) { /** ... */ }

    public static void main(String[] args) {
        Impl1 impl = new Impl1();
        impl.log("Something..."); // ????
    }
}
```

Metody domyślne

```
public class Impl1 implements Interface1, Interface2 {
    @Override void method1(String str) { /** ... */ }
    @Override void method2(String str) { /** ... */ }

    @Override
    public void log(String str){
        System.out.println("Impl1 logging::"+str);
        Interface1.print("abc");
    }

    public static void main(String[] args) {
        Impl1 impl = new Impl1();
        impl.log("Something...");
    }
}
```

Statyczne metody w interfejsie

```
public interface StringUtils {
    default void print(String str) {
        if (!isNull(str)) {
            System.out.println("MyData Print::" + str);
        }
    }

    static boolean isNull(String str) {
        System.out.println("Interface Null Check");

        return str == null ? true : "".equals(str) ? true : false;
    }
}
```

Statyczne metody w interfejsie - main (?!)

```
public interface StringUtils {
    default void print(String str) {
        if (!isNull(str)) {
            System.out.println("MyData Print::" + str);
        }
    }

    static boolean isNull(String str) {
        System.out.println("Interface Null Check");

        return str == null ? true : "".equals(str) ? true : false;
    }

    public static void main(String[] args) {
        /**
         * _ _"
         */
    }
}
```

Interfejs LinkedList

Interfejs LinkedList

Optymalny typ kolekcji z dostępem sekwencyjnym

Interfejs LinkedList

Optymalny typ kolekcji z dostępem sekwencyjnym

Zoptymalizowany pod kątem operacji wstawiania i usuwania elementów ze środka

Interfejs LinkedList

Optymalny typ kolekcji z dostępem sekwencyjnym

Zoptymalizowany pod kątem operacji wstawiania i usuwania elementów ze środka

Wolne operacje swobodnego dostępu (np. pobrania elementu o wskazanym indeksie)

Przykład kolekcji LinkedList

```
public class Main {
    public static void main(String[] args) {
        LinkedList<String> strings = new LinkedList<>(); // !

        strings.add("A");
        strings.add("B");
        strings.add("C");

        strings.addFirst("F");
        strings.addLast("E");

        strings.add("G");
        strings.removeFirst();
        strings.removeLst();

        System.out.println(strings);
    }
}
```

Interfejs Queue

Interfejs Queue

Kontener reprezentujący kolejkę jednokierunkową

Interfejs Queue

Kontener reprezentujący kolejkę jednokierunkową

FIFO - first-in, first-out

Interfejs Queue

Kontener reprezentujący kolejkę jednokierunkową

FIFO - first-in, first-out

Typowa implementacja Queue - LinkedList

Interfejs Queue

Kontener reprezentujący kolejkę jednokierunkową

FIFO - first-in, first-out

Typowa implementacja Queue - LinkedList

Interfejs ten zawiera dodatkowe metody dostępne za pomocą rzutowania w górę

Przykład zastosowania kolejki Queue

```
public class Main {
    public static void main(String[] args) {
        Queue<Integer> queue = new LinkedList<Integer>();
        Random rand = new Random(47);

        for (int i = 0; i < 10; i++) {
            queue.offer(rand.nextInt(i + 10));
        }

        System.out.print(queue);

        while (queue.peek() != null) {
            System.out.print(queue.remove() + " - ");
        }

        System.out.print(queue);
    }
}
```

Metody interfejsu Queue

Metody interfejsu Queue

offer() - wstawia element na koniec kolejki (jeśli to możliwe)

Metody interfejsu Queue

offer() - wstawia element na koniec kolejki (jeśli to możliwe)

peek() - zwraca element z przodu bez usuwania lub false

Metody interfejsu Queue

offer() - wstawia element na koniec kolejki (jeśli to możliwe)

peek() - zwraca element z przodu bez usuwania lub **false**

element() - zwraca element z przodu bez usuwania lub rzuca NoSuchElementException

Metody interfejsu Queue

Metody interfejsu Queue

pool() - usuwa i zwraca element z czoła kolejki lub false

Metody interfejsu Queue

pool() - usuwa i zwraca element z czoła kolejki lub false

remove() - usuwa i zwraca element z czoła kolejki lub rzuca NoSuchElementException

Porównywanie obiektów

Porównywanie obiektów

Do porównywania dwóch obiektów służą dwa interfejsy - Comparator oraz Comparable

Porównywanie obiektów

Do porównywania dwóch obiektów służą dwa interfejsy - Comparator oraz Comparable

Interfejs Comparator służy do implementacji logiki porównywania obiektów dwóch typów

Porównywanie obiektów

Do porównywania dwóch obiektów służą dwa interfejsy - Comparator oraz Comparable

Interfejs Comparator służy do implementacji logiki porównywania obiektów dwóch typów

Interfejs Comparable służy do implementacji logiki porównywania obiektów tego samego typu

Wartości zwracane przez metody compare() i compareTo()

Wartości zwracane przez metody compare() i compareTo()

0 - jeśli obiekty są sobie równe

Wartości zwracane przez metody compare() i compareTo()

0 - jeśli obiekty są sobie równe

wartość pozytywna - jeśli obiekt pierwszy jest "większy" od obiektu drugiego

Wartości zwracane przez metody compare() i compareTo()

0 - jeśli obiekty są sobie równe

wartość pozytywna - jeśli obiekt pierwszy jest "większy" od obiektu drugiego

wartość negatywna - jeśli obiekt pierwszy jest "mniejszy" od obiektu drugiego

Przykładowa klasa Student

```
public class Student {
    private String name;
    private int grade;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getGrade() {
        return grade;
    }

    public void setGrade(int grade) {
        this.grade = grade;
    }

    @Override
    public String toString() {
        return "[name=" + this.name + ", grade=" + this.grade + "]";
    }
}
```

Porównanie stopni

```
class StudentGradeComparator implements Comparator<Student> {
    @Override
    public int compare(Student s1, Student s2) {
        return s2.getGrade() - s1.getGrade();
    }
}
```

Porównanie nazw

```
class StudentNameComparator implements Comparator<Student> {
    @Override
    public int compare(Student s1, Student s2) {
        return s1.getName().compareTo(s2.getName());
    }
}
```

Przygotowanie danych

```
public class ComparatorExample {
        public static void main(String args[]) {
                Student student[] = new Student[3];

                student[0] = new Student();
                student[0].setName("Nick");
                student[0].setGrade(19);

                student[1] = new Student();
                student[1].setName("Helen");
                student[1].setGrade(12);

                student[2] = new Student();
                student[2].setName("Ross");
                student[2].setGrade(16);

                // ...
    }
}
```

Sortowanie po stopniach

```
public class ComparatorExample {
        public static void main(String args[]) {
            // ...

        Arrays.sort(student, new StudentGradeComparator());
                System.out
                        .println("Order of students after sorting by student
grade is");

                for (int i = 0; i < student.length; i++) {
                        System.out.println(student[i].getName() + "\t"
                        + student[i].getGrade());
                }
        }
}
```

Sortowanie po nazwach

```
public class ComparatorExample {
        public static void main(String args[]) {
            // ...

        Arrays.sort(student, new StudentNameComparator());
                System.out
                        .println("Order of students after sorting by student
name is");

                for (int i = 0; i < student.length; i++) {
                        System.out.println(student[i].getName() + "\t"
                        + student[i].getGrade());
                }
        }
}
```

Porównanie obiektu klasy Student z innym

```
public class Student implements Comparable<Student> {
    // ...

    @Override
    public int compareTo(Student s) {
        return ???;
    }
}
```

ANY QUESTIONS?