Programowanie w Java Interfejsy

Łukasz Rybka · Gdańsk 2016/17

Interfejsy

> *Interfejsy i klasy abstrakcyjne to strukturyzowane środki oddzielenia interfejsu od implementacji.*
>
> — Bruce Eckel

Interfejsy

> *Słowo kluczowe **interface** generuje (…) abstrakcyjną klasę bazową, która jest całkowicie pozbawiona (…) implementacji.*
>
> — Bruce Eckel

Przykład prostego interfejsu

```
public interface Drawable {
    public void draw();

    public void erase();
}

public class Circle implements Drawable {
    @Override
    public void draw() {
        // ...
    }

    @Override
    public void erase() {
        // ...
    }

    public static void main(String[] args) {
        Drawable drawable = new Circle();
    }
}
```

Przykład prostego interfejsu

```
public class Painting implements Drawable {
    @Override
    public void draw() {
        // ...
    }

    @Override
    public void erase() {
        // ...
    }

    public static void main(String[] args) {
        Drawable circle = new Circle();
        circle.draw();
        circle.erase();

        Drawable paiting = new Painting();
        paiting.draw();
        paiting.erase();
    }
}
```

Zasady tworzenia interfejsów

Zasady tworzenia interfejsów

Interfejsy nie posiadają implementacji (*)

Zasady tworzenia interfejsów

Interfejsy nie posiadają implementacji (*)

Mogą podlegać dziedziczeniu (!)

Zasady tworzenia interfejsów

Interfejsy nie posiadają implementacji (*)

Mogą podlegać dziedziczeniu (!)

Wszystkie pola interfejsu są domyślnie (niejawnie) statyczne i finalne

Implementacja wielu interfejsow

Implementacja wielu interfejsow

Pozwala na osiągnięcie efektu niedostępnego dla dziedziczenia - klasa może być wielu niezależnych typów

Implementacja wielu interfejsow

Pozwala na osiągnięcie efektu niedostępnego dla dziedziczenia - klasa może być wielu niezależnych typów

Lepsza ekapsulacja - ukrywanie części implementacji niezwiązanej z danym typem

Implementacja wielu interfejsow

Pozwala na osiągnięcie efektu niedostępnego dla dziedziczenia - klasa może być wielu niezależnych typów

Lepsza ekapsulacja - ukrywanie części implementacji niezwiązanej z danym typem

Nie jest idealna…

Implementacja wielu interfejsów

```
public interface Drawable {
    public void draw();

    public void erase();
}

public interface Printable {
    public void print();
}
```

Implementacja wielu interfejsów

```
public class Paiting implements Drawable, Printable {
    @Override
    public void draw() {
        // ...
    }

    @Override
    public void erase() {
        // ...
    }

    @Override
    public void print() {
        // ...
    }
}
```

Implementacja wielu interfejsów

```
public class Printer {
    public static void print(Printable printable) {
        printable.print();
    }
}

public class Drawer {
    public static void draw(Drawable drawable) {
        drawable.draw();
    }
    public static void erase(Drawable drawable) {
        drawable.erase();
    }
}

public class Main {
    public static void main(String[] args) {
        Painting painting = new Painting();
        Drawer.draw(painting);
        Printer.print(painting);
        Drawer.erase(painting);
    }
}
```

Problemy implementacji wielu interfejsów

```
public interface Interface1 {
    void func();
}

public interface Interface2 {
    int func(int i);
}

public interface Interface3 {
    int func();
}
```

Problemy implementacji wielu interfejsów

```
public interface Interface1 { void func(); }
public interface Interface2 { int func(int i); }
public interface Interface3 { int func(); }

public class Impl1 implements Interface1 {
    @Override
    void func() {
        // ...
    }
}

public class Impl2 implements Interface2 {
    @Override
    int func(int i) {
        // ...
    }
}

public class Impl3 implements Interface3 {
    @Override
    int func() {
        // ...
    }
}
```

Problemy implementacji wielu interfejsów

```
public interface Interface1 { void func(); }
public interface Interface2 { int func(int i); }
public interface Interface3 { int func(); }

public class Impl1And2 implements Interface1, Interface2 {
    @Override
    void func() {
        // ...
    }

    @Override
    int func(int i) {
        // ...
    }
}
```

Problemy implementacji wielu interfejsów

```
public interface Interface1 { void func(); }
public interface Interface2 { int func(int i); }
public interface Interface3 { int func(); }

public class FullImplementation implements Interface1, Interface2, Interface3
{
    // ...
}
```

Problemy implementacji wielu interfejsów

```
public interface Interface1 { void func(); }
public interface Interface2 { int func(int i); }
public interface Interface3 { int func(); }

public class FullImplementation implements Interface1, Interface2, Interface3
{
    @Override
    void func() {
        // ...
    }

    @Override
    int func(int i) {
        // ...
    }

    @Override
    int func() {
        // ...
    }
}
```

Dziedziczenie interfejsów

```
public interface Printable {
    public void print();
}

public interface MobilePrintable extends Printable {
    public void mobilePrint();
}

public class Paiting implements MobilePrintable {
    @Override
    public void print() {
        // ...
    }

    @Override
    public void mobilePrint() {
        // ...
    }
}
```

Polimorfizm raz jeszcze…

> *(…) przypisania, w których typ źródła jest inny niż typ celu są nazywane* ***przypisaniami polimorficznymi*****.**
>
> — Bertrand Meyer

ANY QUESTIONS?