Polimorfizmas

Mindaugas Karpinskas
2017

Trumai

- Paveldėjimas
- Plečiamumo problemos
- Polimorfizmas
- abstract
- interface

Dar apie paveldėjimą

- Java kalboje kiekvienas paveldėtos klasės objektas gali būti naudojamas ten, kur reikalingas tėvinės klasės objektas!

Car bus train

```
abstract class Car {
    abstract void go();
}
class Bus extends Car {
    @Override
    void go() {
        System.out.println("Bus goes...");
    }
}
class Train extends Car {
    @Override
    void go() {
        System.out.println("Train goes...");
    }
}
```

Car bus train

```
public class RunCars {

    public static void main(String[] args) {
        Train a = new Train();
        Bus b = new Bus();
        Car c = new Bus();

        makeMove(a);
        makeMove(b);
        makeMove(c);
    }

    static void makeMove(Car c) {
        c.go();
    }

}
```

????
?
?

Polimorfizmas: Užduotis 1 (Produktai)

- NENAUDUOJAME paveldėjomo!
- Klasės: Pienas, Duona, Sviestas, Suris, Jogurtas;
- Visos klasės turi po metodą arGalimaVartoti();, kuris grąžina true reikšme tik tuo atveju jei atsitiktinis skaičiuos, sugeneruotas objekto kūrimo metu nuo 1-5 yra lygus:
  - Pienas, Jogurtas atveju 1 arba 2;
  - Kitais klasių atvejais 1, 2, 3;
- Klasė ProduktuTikrintuojas - turi turėti galimybę (metodus patikrink(....* produktas)) patikrinti ar galime vartoti produktą. Jo pagalba išvesime į konsolę informaciją ar galima vartoti produktą, - ar negalima

Polimorfizmas: Užduotis 1 (Produktai)

- NENAUDUOJAME paveldėjomo!
- Klasės: Pienas, Duona, Sviestas, Suris, Jogurtas;
- Visos klasės turi po metodą arGalimaVartoti();, kuris grąžina true reikšme tik tuo atveju jei atsitiktinis skaičiuos, sugeneruotas objekto kūrimo metu nuo 1-10 yra lygus:
  - Pienas, Jogurtas atveju 1 arba 2 arba 7;
  - Kitais klasių atvejais 1, 2, 3, 6;
- Klasė ProduktuTikrintuojas - turi turėti galimybę (metodus patikrink(....* produktas)) patikrinti ar galime vartoti produktą. Jo pagalba išvesime į konsolę informaciją ar galima vartoti produktą, - ar negalima

```
class Pienas {
        private final int sk;
        public Pienas(int sk) {
                this.sk = sk;
        }
        public boolean arGalimaVartoti() {
                return sk > 0 && sk < 3;
        }
}
class Duona {
        private final int sk;
        public Duona(int sk) {
                this.sk = sk;
        }
        public boolean arGalimaVartoti() {
                return sk > 0 && sk < 4;
        }
}
```

ProduktuTikrintuojas

```
class ProduktuTikrintuojas {
	public void patikrint(Pienas pienas) {
		if (pienas.arGalimaVartoti()) {
			System.out.println("Produktas tinkamas vartojimui.");
		} else {
			System.out.println("Produkto vartojomo terminas pasibaigęs!!");
		}
	}
	public void patikrint(Duona duona) {
		if (duona.arGalimaVartoti()) {
			System.out.println("Produktas tinkamas vartojimui.");
		} else {
			System.out.println("Produkto vartojomo terminas pasibaigęs!!");
		}
	}
...
}
```

```
public static void main(String[] args) {
        Pienas p1 = new Pienas(1 + random.nextInt(5));
        Pienas p2 = new Pienas(1 + random.nextInt(5));
        Pienas p3 = new Pienas(1 + random.nextInt(5));
        Duona d1 = new Duona(1 + random.nextInt(5));
        Duona d2 = new Duona(1 + random.nextInt(5));
        Duona d3 = new Duona(1 + random.nextInt(5));

        ProduktuTikrintuojas tikrintojas = new ProduktuTikrintuojas();
        tikrintojas.patikrint(p1);
        tikrintojas.patikrint(p2);
        tikrintojas.patikrint(p3);

        tikrintojas.patikrint(d1);
        tikrintojas.patikrint(d2);
        tikrintojas.patikrint(d3);

}
```

Paveldėjimo pavyzdys

//k=f; ?

```
public class Kvadratas extends Figura {
    public static void main(String[] args) {
        Kvadratas k = new Kvadratas();
        Figura f = new Figura();
        f.metodas(k); // Kvadratas vietoje Figura
        f = k; // k=f;
    }
}

class Figura {
    void metodas(Figura F) {
        System.out.println("F");
    }
}
```

Klasė Gyvunas

```
abstract class Gyvunas {
	abstract void kalba() ;
}
class Kate extends Gyvunas {
	void kalba() {
		System.out.println("Kate miaukia");
	}
}
class Suo extends Gyvunas {
	void kalba() {
		System.out.println("Suo loja");
	}
}
class Lusis extends Kate {
	void kalba() {
		System.out.println("Lusis urzgia");
	}
}
```

Pavyzdys be paveldėjimo

Panaudojimo pavyzdys, be polimorfizmo

```
class GyvunaiMoka {
    void kalbeti(Kate k) {
        k.kalba();
    }
    void kalbeti(Suo s) {
        s.kalba();
    }
    void kalbeti(Lusis l) {
        l.kalba();
    }
    public void prakalbink() {
        Kate k = new Kate();
        kalbeti(k);
        Suo s = new Suo();
        kalbeti(s);
        Lusis l = new Lusis();
        kalbeti(l);
    }
}
```

Plečiamumo problemos

- Jei atsiranda naujas gyvūnas, reikia perrašyti metodą kalbėti
- Jei atsiranda naujas metodas, jį reikia parašyti visiems gyvūnams

Pavyzdys su paveldėjimu

```
class GyvunaiMoka {
    void kalbeti(Gyvunas k) {
        k.kalba();
    }

    public void prakalbink() {
        Kate k = new Kate();
        kalbeti(k);
        Suo s = new Suo();
        kalbeti(s);
        Lusis l = new Lusis();
        kalbeti(l);
    }
}
```

Rezultatas

Programa atspausdins:

>Kate miaukia

>Suo loja

>Lusis urzgia

Iš kur programa žino, kada kurios klasės metodą kviesti?

```
    public static void main(String[] args) {
        GyvunaiMoka gyvunai = new GyvunaiMoka();
        gyvunai.prakalbink();
    }
Arba
    public static void main(String[] args) {
        GyvunaiMoka gyvunai = new GyvunaiMoka();
        Kate k = new Kate();
        gyvunai.kalbeti(k);
        Suo s = new Suo();
        gyvunai.kalbeti(s);
        Lusis l = new Lusis();
        gyvunai.kalbeti(l);
    }
class GyvunaiMoka {
    void kalbeti(Gyvunas k) {
        k.kalba();
    }
}
```

Polimorfizmas: Užduotis 2 (Produktai)

- SU paveldėjimu!
- Klasės: Produktas (turi abstraktų metodą arGalimaVartoti() ir int protected kintamąjį)->
  - Reikalingos kitos vaikinės klasės, kurios paveldi Produktas klasę: Pienas, Duona, Sveistas, Suris, Jogurtas;
- Visos klasės turi po metodą arGalimaVartoti();, kuris grąžina true reikšme tik tuo atveju jei atsitiktinis skaičiuos, sugeneruotas objekto kūrimo metu nuo 1-5 yra lygus:
  - Pienas, Jogurtas atveju 1 arba 2;
  - Kitais klasių atvejais 1, 2, 3;
- Klasė ProduktuTikrintuojas - turi turėti metodus patikrink(Produktas produktas), kurių pagalba išvesime į konsolę informacija ar galima vartuoti produktą, - ar negalima

Polimorfizmas: Užduotis 3 (Skaičiai)

- Abstrakti klasė Skaicius turi turėti
  - protected int tipo masyvą - masyvo dydis (atsitiktinis skaičius, 20 - 200 elemntų)
  - public abstraktų metodą generuok()
  - public metodą suma() - grąžina masyvo elementų sumą

+toString perašomas
Arrays.toString(masyvas);

- Sukuri kitas klases, kurios paveldi Skaicius klasę
  - TeigimasSkaicius - implementuoja generuok ir užpildo masyvą teigiamais atsitiktiniais (max random 1000) skaičiais
  - LyginisSkaicius - generuok metode užpildo elementus lyginiais skaičiais
  - NelyginisSkaicius - generuok - užpildo nelyginiais skaiciais
  - NeigiamiSkaiciai - generuok - neigiamais skaičiais
- Sukurti statinį metodą, kuris turi Skaicius tipo parametrą sk ir:
  - Parametro kintamajam kviečia metodą generuok()l
  - Metodo suma() rezultatą išveda į konsolę
- Main metode sukurti įvarių skaičių tipų egzempliorių ir su jais iškviesti anksčiau apsirašytą statinį metodą

Skaicius

```
abstract class Skaicius {
        protected int[] masyvas;
        public abstract void generuok();
        public int suma() {
                int s = 0;
                for (int i = 0; i < masyvas.length; i++) {
                        s += masyvas[i];
                }
                return s;
        }
}
        .....

        static void run(Skaicius sk) {
                sk.generuok();
                System.out.println("Suma: " + sk.suma());
        }
```

Polimorfizmas

- **Polimorfizmas** - galimybė turėti skirtingas elgsenas (skirtingas metodų realizacijas) skirtingose situacijose naudojant tipus
  - Polimorfizmu vadiname Java savybę, kuri suteikia galimybę grupei metodų naudoti tą patį metodą, tačiau kiekvienas metodo panaudojimas gali duoti skirtingus rezultatus.
- Jei imame nuorodą į objektą ir ją naudojame kaip nuorodą į bazinę klasę, toks veiksmas vadinamas **nukirtimu** iš viršaus (angl. upcasting)
  - Pvz., Kvadratas a = new Kvadratas();
  - Figura b = a;
- Nukirtimas iš apačios (angl. downcasting) yra nuorodos į bazinę klasę naudojimas vietoje nuorodos į paveldėtą klasę
  - Pvz., Figura a = new Kvadratas();
  - Kvadratas b = (Kvadratas)a;

Polimorfizmo naudojimo privalumai

- Programinis kodas lengvai rašomas ir skaitomas
- Klasės sąsaja vienoda, tipų specifika svarbi realizacijoje. Ne įtakojamos klasės - naudotojos

Polimorfizmo naudojimo trūkumai

- Jei reikia vaikinės klasės savybių, klasėse-naudotojose reikia būtinai panaudoti downcast veiksmą
  - Pvz., Figura a = new Kvadratas();
  - Kvadratas b = (Kvadratas)a;

Abstrakti klasė

- Abstrakti klasė (angl. abstract class) parūpina bendrą sąsają klasėms, kurios paveldi abstrakčios klasės savybes
- Abstraktus metodas bei klasė žymimi raktiniu žodžiu abstract
- Abstraktus metodas turi deklaraciją, bet neturi realizacijos

Abstrakti klasė

- Abstrakčios klasės objektai nekuriami - kompiliatorius neleis šito daryti
  - abstract klasės objektą (egzempliorių) draudžiama kurti, tačiau apibrėžti šio tipo kintamąjį yra leistina.
- Abstraktūs metodai realizuojami vaikinėse klasėse
- Abstrakti klasė taip pat gali būti vaikine abstrakčios klasės klase.
- Abstrakti klasė gali turėti konstruktorius
- Abstrakčioji klasė - ne visiškai realizuota klasė, turinti deklaruotų bet nerealizuotų (abstract) metodų.
  - Įgalina abstraktųjį programavimą, kai ne visos klasės veikimo detalės yra žinomos kodo rašymo metu, t.y., paliekamos realizuoti išvestinėse klasėse.
- Jeigu klasė būdama abstrakčiosios klasės palikuoniu nerealizuoja visų bazinių klasių abstrakčiųjų metodų, ji taip pat turi būti deklaruojama kaip abstract.

Interfeisai

- Interfeisus (interface) reikėtų suprasti kaip visiškai abstrakčias klases. Įgalina modeliuoti klasių daugialypį paveldėjimą, kuris Javoje draudžiamas.
- Kiekviena klasė gali turėti (extend) tik vieną bazinę klasę, tačiau įgyvendinti/realizuoti (implement) keletą interfeisų.
    - Savo ruožtu, interfeisas gali išplėsti (extend) vieną ar daugiau interfeisų, ir tai suprasime kaip baziniuose interfeisuose deklaruotų metodų aibių apjungimą.
- Kiekvienas interfeiso metodas traktuojamas kaip "public abstract", o laukai - "public static final" - konstantos.
- Leidžiama į interfeisą įtraukti "default" metodų realizacijas. Nuo java 8-sios versijos.

Interface klasė

- **Interface** klasė deklaruojama žodžiu **interface**, nerašant žodžio class(!)
- Vaikinė **interface** klasės klasė turi naudoti raktinį žodį **implements** ir realizuoja atitinkamus metodus
- Pagal nutylėjimą interface klasės metodai yra **public**, tod ėl j ų realizacijos irgi public
- **Interface** klasė gali būti ir interface klasės vaikinė klasė I
- **interface** neturi konstruktorių

Interfeisų panaudojimas konstantų rinkiniams sudaryti

Reikšmės pagal nutylėjimą yra **public**, **static** ir **final**.

Jos naudojamos taip: **Months.JANUARY**

```
public interface Months {
    int JANUARY = 1,
    FEBRUARY = 2,
    MARCH = 3,
    APRIL = 4,
    MAY = 5,
     JUNE = 6,
    JULY = 7,
    AUGUST = 8,
    SEPTEMBER = 9,
    OCTOBER = 10,
    NOVEMBER = 11,
    DECEMBER = 12;
}
```

Nukirtimas (upcast) iki kelių klasių

```
interface A {
	void af();
}
interface B {
	void bf();
}
class C implements A, B {
	public void af() {
		System.out.println("af");
	}
	public void bf() {
		System.out.println("bf");
	}
}
```

```
	public static void main(String[] args) {
		C cc = new C();
		A aa = cc;
		aa.af();
		B bb = cc;
		bb.bf();
	}
```

Vardų kolizijos jungiant interfeisus

```
interface I1 {
	void f();
}
interface I2 {
	int f(int i);
}
interface I3 {
	int f();
}
class C {
	public int f() {
		return 1;
	}
}
```

```
class C2 implements I1, I2 {
	public void f() {
	}
	public int f(int i) {
		return 1;
	} // perkrovimas
}
class C3 extends C implements I2 {
	public int f(int i) {
		return 1;
	} // perkrovimas
}
class C4 extends C implements I3 {
	public int f() {
		return 1;
	} // sutampa, todėl gerai
}
```

Vardų kolizijos jungiant interfeisus

```
interface I1 {
	void f();
}
interface I2 {
	int f(int i);
}
interface I3 {
	int f();
}
class C {
	public int f() {
		return 1;
	}
}
```

```
class C5 extends C implements I1 {
} // blogai, nes skiriasi tik
// The return types are incompatible for the
inherited methods I1.f(), C.f()

interface I4 extends I1, I3 {
} // grąžinamas tipas
// The return types are incompatible for the
inherited methods I1.f(), I3.f()
```

Interfeisas vs. abstraktiklasė

- Tiek interfeisas, tiek abstrakti klasė pateikia programuotojui sąsają, o realizacija nebūtina
- Be to, interfeisas leidžia nukirsti (upcast) iki daugiau nei vienos klasės
- Interfeisas turi visus abstrakčios klasės privalumus

abstract VS interface

```
abstract class Figura {
      abstract void metodas();
}
```

```
interface Figura {
      void metodas();
}
```

Klausimai