Paveldėjimas

Pakartotinio kodo naudojimo būdai

Kodo kopijavimas ir taisymas

Kompozicija

Paveldėjimas

Kompozicija

Esamų klasių objektų kūrimas naujoje klasėje pvz.,

```
class A {
      int i;
}

class B {
      int j;
      A a;
}
```

Klasėje B kintamasis a inicializuojamas reikšme null, todėl reikia sukurti klasės A objektą

Objektinių kintamųjų inicializavimas

Deklaravimo metu

```
class B {
	int j;
	A a = new A();
}
```

Konstruktoriuje

```
class B {
	int j;
	A a;

	B() {
		a = new A();
	}
}
```

Prieš panaudojimą

```
class B {
	int j;
	A a;

	int didinti() {
		a = new A();
		return a.i++;
	}
}
```

Paveldėjimas

Siekiant išvengti kodo kartojimo, objektinėje paradigmoje įprasta naudoti paveldėjimą (angl. inheritance)

Paveldėjimas leidžia sukurti naują klasę panaudojant ankstesnės klasės savybes

Tėvinės klasės savybės būdingos ir vaikinei klasei

Galima pridėti naujų savybių

Paveldėjimui nusakyti naudojamas žodis extends

Paveldėjimo pavyzdys

```
class A {
	int i;
}

class B extends A {
	int j;

	int didinti() {
		return i++;
	}
}
```

Konstruktoriai ir paveldimumas

Konstruktoriai nėra paveldimi

Tėvinės klasės kintamieji turi būti inicijuoti vaikinėje klasėje

Galima kviesti tėvinės klasės konstruktorių iš vaikinės klasės konstruktoriaus

Kuriant vaikinių klasių objektus, yra tokia veiksmų seka:

- Išskiriama vieta
- Statiniai kintamieji inicializuojami pradžioje tėvinėse klasėse, paskui vaikinėse
- Konstruktorius kviečiamas pradžioje tėvinėms klasėms, paskui vaikinėms

Konstruktoriai

Tėvinės klasės konstruktorius kviečiamas su super() nurodant parametrus

Kuriant objektus, pradžioje kviečiamas tėvinės klasės konstruktorius, o tik tada vaikinės klasės konstruktorius. Taip būna net tada, jei programoje neparašyta, kad reikia kviesti tėvinės klasės konstruktorių

Kviečiant tėvinės klasės konstruktorių, negali būti jokio kito žingsnio vaikinės klasės konstruktoriuje prieš šį žingsnį

Tos pačios klasės konstruktorius kviečiamas panaudojant this

Konstruktoriai. 1 pavyzdys

```
class A {
	A() {
		System.out.print("Kuriam A");}
}
class B extends A {
	B() {
		System.out.print(" Kuriam B");}
}
class C extends B {
	C() {
		System.out.print(" Kuriam C");}
	public static void main(String[] args) {
		C c = new C();
	}
} // atspausdins: Kuriam A Kuriam B Kuriam C
```

Konstruktoriai. 2 pavyzdys

```
class A {
	int i;
	A() {
		i = 5;
	}
}

class B extends A {
	int j;

	B() {
		// šioje vietoje kviečiamas klasės A konstruktorius
		j = i + 1;
	}
}
```

Konstruktoriai. 3 pavyzdys

```
class A {
        int i;

        A(int k) {
                i = k;
        }
}

class B extends A {
        int j;

        B(int m) {              // čia nieko įterpti negalima
                super(m);       // būtinas
                j = i + 1;
        }
}
```

Perrašomi metodai ir kintamieji

Metodo perrašymas (angl. overriding) - tai metodo kūno (angl. body) perrašymas vaikinėje klasėje

Perrašyti metodai ir kintamieji turi tokį pat aprašymą kaip tėvinėje klasėje

Vaikinėje klasėje abi versijos yra pasiekiamos

Prie tėvinės klasės kintamojo arba metodo galima prieiti panaudojant super

Perrašant metodus, jų priėjimo teisių negalima sumažinti

Metodo perrašymo pavyzdys

```
class A {
	int i;
	int didinti() {
		return i + 1;
	}
}

class B extends A {
	int didinti() {
		return i + 2;
	}
	void spausdinkAbu() {
		System.out.println(didinti());
		System.out.println(super.didinti());
	}
}
```

Perrašymas vs. perkrovimas

```
class A {
	int metodas() {
		return 5;
	}

	// perkrovimas (angl. overloading)
	int metodas(int k) {
		return k + 1;
	}
}

class B extends A {
	// perrašymas (angl. overriding)
	int metodas() {
		return 6;
	}
}
```

final kintamieji

Raktas final prie kintamojo reiškia, kad kintamojo reikšmė nekis – galutinė

Kintamajam reikšmę galima priskirti jį deklaruojant

Jei reikšmę priskiria kompiliatorius, kintamojo vardas rašomas didžiosiomis raidėmis, žodžiai skiriami pabraukimais

Pvz., final float MANO_PI=3.14;

Tuščias final kintamasis

Taip pat galima deklaruoti tuščią final kintamąjį, o jam reikšmę priskirti kiekviename konstruktoriuje

```
class B {
	final int j;

	B() {
		j = 1;
	}
}
```

Objektinio tipo final kintamasis

Jei kintamasis ne primityviojo, o objektinio tipo, negalima jam priskirti kitą objektą, tačiau patį objektą modifikuoti galima

```
class P {
}

class D {
	final P p1;
	final P p2 = new P();
	D() {
		p1 = new P();
	}
	public static void main(String[] args) {
		D d = new D();
		// negalima: d.p1=new P(); d.p2=d.p1;
	}
}
```

staic final kintamasis

Raktų pora static final rodo, kad tai konstanta visiems klasės objektams

```
static final float MANO_PI=3.14;
```

final metodas

Jei metodas yra final, jo negalima perrašyti paveldėtose klasėse

Privatūs metodai ir taip yra final net nerašant raktinio žodžio

Jei final parašysime prie klasės, tai iš jos negalima paveldėti

Klasių hierarchija

Visos klasės paveldi klasės java.lang.Object savybes

Jei nėra nurodyta tėvinė klasė, tai pagal nutylėjimą paveldės Object savybes

Jei kokia nors tėvinė klasė nurodyta, tada paveldės Object savybes per tėvinę klasę

Klasės Object metodai

equals – lygina objektų turinį

clone – sukuria naują objektą – to paties objekto kopiją

toString – grąžina tekstinę objekto informaciją