Java

Öröklés, polimorfizmus, absztrakt osztályok

Dr. Kiss Gábor

Öröklés

Egy meglévő osztályból új osztályt készítünk.
A meglévő osztály az ősosztály (szülő osztály), az új osztály pedig a származtatott osztály (utódosztály, leszármazott osztály, gyermek osztály).

Származtatott osztályból is készíthetünk új osztályt, ebben az esetben osztályhierarchiát (öröklési fát) hozunk létre.

Az öröklés lényege, hogy olyan osztályt hozunk létre, amely az ősosztály minden tulajdonságával (mezőjével, adatával) és metódusával (műveletével) rendelkezik, ezen kívül más tulajdonságokkal, metódusokkal egészítjük/bővítjük ki.

A Java nyelv az egyszeres öröklést támogatja, azaz az utódosztályok pontosan egy ősosztályból örökölhetnek, egy ősük lehet.
Egy osztályt egy meglévő osztály kiterjesztéseként definiálunk.

A Java-ban minden osztály kiterjesztett, ugyanis, ha nem adunk meg mást (eddig így csináltuk), akkor a **java.lang.Object** osztályból származik az osztály. Az öröklési fa legfelső szintje.
Az **Object** definiálja azokat a tagokat, amelyekre minden osztályban szükség van, pl. **getClass**, **toString**, **equals**.

Object(java.lang.Object) osztály metódusai:
– protected Object clone()
– boolean equals(Object obj)
– void finalize()
– Class<?> getClass()
– int hashCode()
– void notify()
– void notifyAll()
– String toString()
– void wait()

Az örökítést az **extends** kulcsszó jelzi.

```
public class Utodosztaly extendsOsosztaly{
}
```

A leszármazott osztály rendelkezik a szülő osztály minden tagjával, tehát az utód az ős minden tulajdonságát és metódusát automatikusan megörökli, de csak azokat látja, amelyekhez a szülő hozzáférést engedélyezett.

–**public**: bármelyik osztályból elérhető
–**private**: nem elérhető, csak az osztályon belülről
–**protected**: elérhető, de csak az utódosztályban, vagy a csomagban (package)
– jelzés nélküli: csak azonos csomagban definiált osztályból érhető el

A konstruktorokat a leszármazott nem örökli.
A gyermek konstruktorának első sorában meghívhatjuk az ős bármelyik konstruktorát a **super** kulcsszóval.

Ha ezt nem jelöljük, a fordító meghívja a szülő paraméter nélküli konstruktorát.

Ez megelőzi az utódosztályban deklarált példányváltozók inicializálását és az inicializáló blokkok végrehajtását is (így „veszi” át a szülő minden tulajdonságát, metódusait).

Példányosításkor az öröklési hierarchiában lévő osztályok konstruktorai sorra meghívódnak (legfelső őstől kezdve lefelé)

A leszármazott osztályban az ős mezőinek, illetve metódusainak elérésére szintén a super kulcsszó szolgál
`super.mező` vagy `super.Metódus()`

```
public class Auto {
        String rendszam;
        String tipus;
        int ajtoszam;
        int hengerszam;
        int suly;
        int ev;
        int kilometer;
        float fogyasztas;
        final int BENZINAR = 417;

        // konstruktorok
        public Auto(String rendszam, String tipus, int ajtoszam, int hengerszam,
                                int ev, int suly, int kilometer, float fogyasztas) {
                this.rendszam = rendszam;
                this.tipus = tipus;
                this.ajtoszam = ajtoszam;
                this.hengerszam = hengerszam;
                this.ev = ev;
                this.suly = suly;
                this.kilometer = kilometer;
                this.fogyasztas = fogyasztas;
        }
```

```
	public Auto() {
		this("", "", 0, 0, 0, 0, 0, 0.0f);
	}

	public Auto(String rendszam, String tipus) {
		this(rendszam, tipus, 0, 0, 0, 0, 0, 0.0f);
	}

	// metódusok
	private int Benzinkoltseg(int km) {
		System.out.println("Benzinkoltseg");
		return ((int) (km * this.fogyasztas / 100 * this.BENZINAR));
	}

	protected int Egyebkoltseg(int km) {
		System.out.println("Egyebkoltseg");
		return (0);
	}

	public int Utikoltseg(int km) {
		System.out.println("Utikoltseg");
		return (this.Benzinkoltseg(km) + this.Egyebkoltseg(km));
	}
}
```

```
public class AutoTest {
	public static void main(String[] args) {
		// Eltero konstruktorokat hivunk meg
		Auto kocsi = new Auto(„ABC-123", "BMW", 5, 4, 2023, 2000, 56000, 8.7f);
		System.out.println("kocsi.rendszam: " + kocsi.rendszam);
		System.out.println("kocsi.tipus: " + kocsi.tipus);
		System.out.println("kocsi.ajtoszam: " + kocsi.ajtoszam);
		System.out.println("kocsi.hengerszam: " + kocsi.hengerszam);
		System.out.println("kocsi.ev: " + kocsi.ev);
		System.out.println("kocsi.suly: " + kocsi.suly);
		System.out.println("kocsi.kilometer: " + kocsi.kilometer);
		System.out.println("kocsi.fogyasztas: " + kocsi.fogyasztas);

		Auto kocsi2 = new Auto(„DEF-456", „Skoda Fabia");
		System.out.println("kocsi.rendszam: " + kocsi2.rendszam);
		System.out.println("kocsi.tipus: " + kocsi2.tipus);
		System.out.println("kocsi.ajtoszam: " + kocsi2.ajtoszam);
		System.out.println("kocsi.hengerszam: " + kocsi2.hengerszam);
		System.out.println("kocsi.ev: " + kocsi2.ev);
		System.out.println("kocsi.suly: " + kocsi2.suly);
		System.out.println("kocsi.kilometer: " + kocsi2.kilometer);
		System.out.println("kocsi.fogyasztas: " + kocsi2.fogyasztas);
	}
}
```

Polimorfizmus (többalakúság)

Egy gyermek rendelkezik őseinek minden tulajdonságával, ezért minden környezetben használható, ahol az ős használható.

Az automatikus típuskonverzió révén egy Auto típusú változónak értékül adható egy Taxi típusú (lásd példa).

A gyermek osztályban a a szülőosztálytól örökölt metódusok felüldefiniálhatók (módosíthatók).

Ezt a két esetet nevezzük polimorfizmusnak.

```
public class Taxi extends Auto {
        String ceg;
        int kmdij;
        public Taxi(String rendszam, String tipus, int ajtoszam, int hengerszam,
                        int ev, int suly, int kilometer, float fogyasztas, String ceg,
                        int kmdij) {
                super(rendszam, tipus, ajtoszam, hengerszam, ev, suly, kilometer,
                                fogyasztas);
                this.ceg = ceg;
                this.kmdij = kmdij;
        }
        public Taxi() {
                this("", "", 0, 0, 0, 0, 0, 0.0f, "", 0);
        }
        public Taxi(String ceg, int kmdij) {
                this("", "", 0, 0, 0, 0, 0, 0.0f, ceg, kmdij);
        }
        @Override
        protected int Egyebkoltseg(int km) {
                System.out.println("Egyebkoltseg-Taxi");
                return (km * this.kmdij);
        }
}
```

```
public class Taxitest {

	public static void main(String[] args) {
		Taxi egyTaxi = new Taxi("IZV-186", "Picasso", 5, 4, 2013, 1300, 20000,
					5.5f, "Fotaxi", 10);

		int dij, egyebdij;

		dij = egyTaxi.Utikoltseg(200);
		egyebdij = egyTaxi.Egyebkoltseg(200);

		System.out.println("rendszam: " + egyTaxi.rendszam);
		System.out.println("tipus: " + egyTaxi.tipus);
		System.out.println("ajtoszam: " + egyTaxi.ajtoszam);
		System.out.println("hengerszam: " + egyTaxi.hengerszam);
		System.out.println("ev: " + egyTaxi.ev);
		System.out.println("suly: " + egyTaxi.suly);
		System.out.println("kilometer: " + egyTaxi.kilometer);
		System.out.println("fogyasztas: " + egyTaxi.fogyasztas);
		System.out.println("Utikoltseg: " + dij + " Ft");
		System.out.println("Egyebkoltseg: " + egyebdij + " Ft");

	}
}
```

```
public class Taxitest {

        public static void main(String[] args) {
                Taxi egyTaxi = new Taxi("IZV-186", "Picasso", 5, 4, 2013, 1300, 20000,
                                        5.5f, "Fotaxi", 10);
                Auto egyAuto = new Auto("IZV-187", "Picasso", 5, 4, 2015, 1300, 20000,
                                        5.5f);
                Auto kocsi;
                // kocsi = egyAuto; // vagy ez
                kocsi = egyTaxi; // vagy ez
                int dij, egyebdij;

                dij = kocsi.Utikoltseg(200);
                egyebdij = kocsi.Egyebkoltseg(200);
                System.out.println("Utikoltseg: " + dij + " Ft");
                System.out.println("Egyebkoltseg: " + egyebdij + " Ft");
        }
}
```

Absztrakt osztályok és metódusok

- Egy „ősosztály” definiálásakor kialakítunk egy „közös nevezőt”, definiálunk egy olyan alap metóduskészletet, amelyet a leszármazottak mindegyike tartalmazni fog, így egységes módon kezelhetők lesznek.
- Gyakori megoldás az, hogy a közös ős csak ezt a közös felületet határozza meg, tényleges implementációt nem tartalmaz.
- Ilyenkor csak a metódus fejét definiáljuk, a törzsét nem.
- Az ilyen törzs nélküli metódust nevezzük absztrakt metódusnak, az **abstract** kulcsszóval jelöljük.
- Azt az osztályt, amiben van legalább egy absztrakt metódus, absztrakt osztálynak nevezzük és szintén az **abstract** kulcsszóval jelöljük.
- Az absztrakt osztályok nem példányosíthatók, azonban ilyen típusú változót lehet deklarálni.

- Az absztrakt osztályokat ki kell terjeszteni, hogy az absztrakt metódusok implementációt kapjanak.

- Ilyenkor a kiterjesztő osztály felüldefiniálja az absztrakt metódus(oka)t. Ezt úgy mondjuk, hogy az osztály *megvalósítja*, vagy *implementálja* az absztrakt metódust.

- Nem kötelező, hogy a kiterjesztő osztály minden absztrakt metódust megvalósítson, de ellenkező esetben ő maga is absztrakt marad.

- Absztrakt metódus nem lehet: **private**, **final**, **static**, és **native**.

```
public abstract class Sikidom {
        private boolean keruletKiszamitva = false;
        private boolean teruletKiszamitva = false;
        private double kerulet;
        private double terulet;

        public double kerulet() {
                if (!this.keruletKiszamitva) {
                        this.kerulet = this.keruletSzamit();
                        this.keruletKiszamitva = true;
                }
                return this.kerulet;
        }
        public double terulet() {
                if (!this.teruletKiszamitva) {
                        this.terulet = this.teruletSzamit();
                        this.teruletKiszamitva = true;
                }
                return this.terulet;
        }

        protected abstract double keruletSzamit();
        protected abstract double teruletSzamit();
}
```

```
public class Kor extends Sikidom {
	private final double r;

	public Kor(double r) {
		this.r = r;
	}

	public double r() {
		return this.r;
	}

	@Override
	protected double keruletSzamit() {
		return 2 * Math.PI * this.r;
	}

	@Override
	protected double teruletSzamit() {
		return this.r * this.r * Math.PI;
	}
}
```

```
public class Teglalap extends Sikidom {
        private final double a, b;

        public Teglalap(double a, double b) {
                this.a = a;
                this.b = b;
        }

        public double a() {
                return this.a;
        }

        public double b() {
                return this.b;
        }

        protected double keruletSzamit() {
                return 2 * (this.a + this.b);
        }

        protected double teruletSzamit() {
                return this.a * this.b;
        }
}
```

```
public class Foprogram {
	public static void main(String[] args) {
		Kor kor = new Kor(3);
		Teglalap teglalap = new Teglalap(2, 4);
		System.out.println("A kör kerülete: " + kor.keruletSzamit());
		System.out.println("A kör területe: " + kor.teruletSzamit());
		System.out.println("A téglalap kerülete: " + teglalap.keruletSzamit());
		System.out.println("A téglalap területe: " + teglalap.teruletSzamit());
	}

}
```