

# Objektovo orientované programovanie

Učiteľ:  
**Ing. Jozef Wagner PhD.**

Učebnica:  
<https://oop.wagjo.com/>

# OPG

## Teória 11

1. Zapuzdrenie
2. Prekrytie metód
3. Zabránenie dedenia
4. IS-A versus HAS-A

```
class Osoba {  
    String meno;
```

```
    Osoba(String meno) {  
        this.meno = meno;  
    }
```

```
    String getMeno() {  
        return meno;  
    }
```

```
}
```

# Dedičnosť

Podradená trieda, **potomok**, prevezme vlastnosti a správanie nadradenej triedy - **rodiča**

*Anglicky – subclass a superclass*

Pri dedení používame kľúčové slovo **extends**

Trieda môže pomocou dedenia prevziať atribúty (premenné) a metódy z inej triedy

K týmto atribútom a metódam potom môže podradená trieda voľne pristupovať

# Dedičnosť

```
class Student extends Osoba {  
    String skola;
```

```
    Student(String meno, String skola) {  
        super(meno);  
        this.skola = skola;  
    }
```

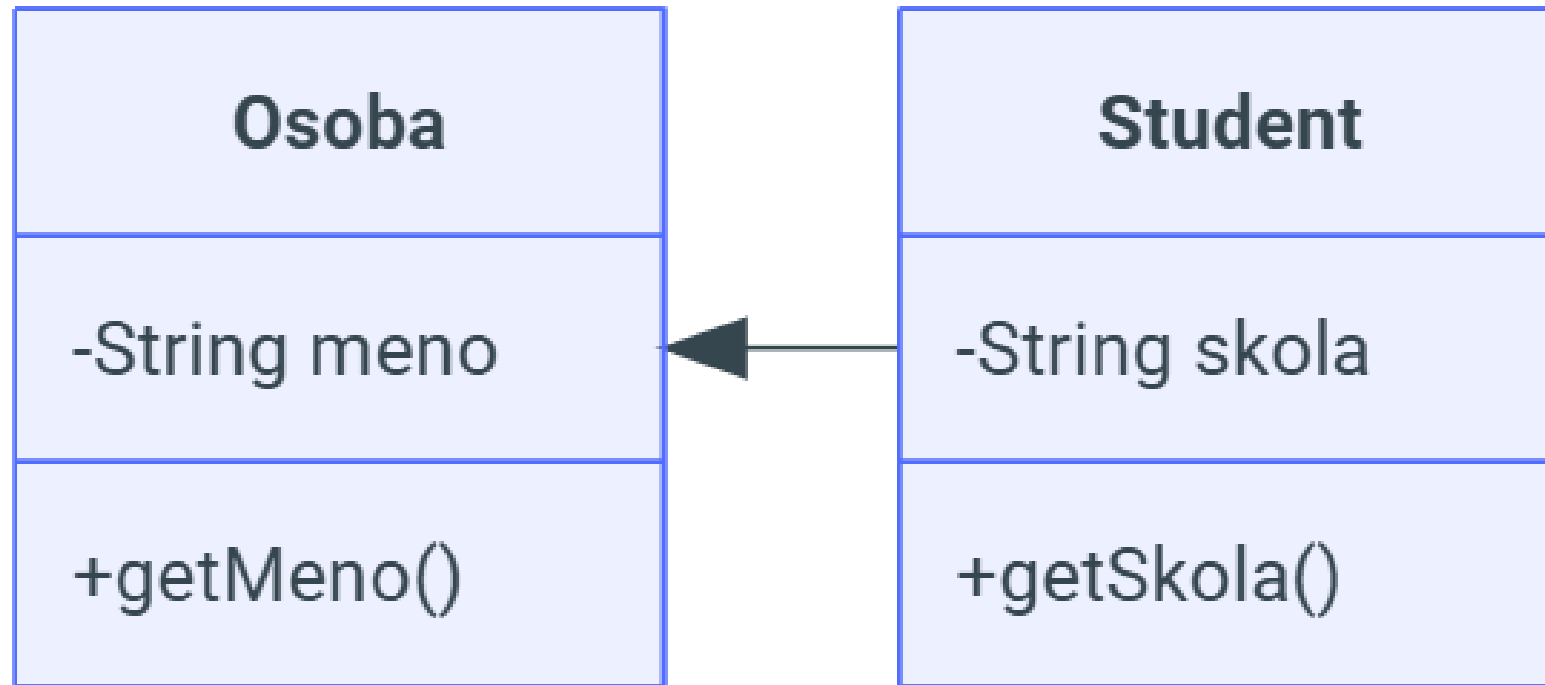
```
    String getSkola() {  
        return skola;  
    }  
}
```

# Dedičnosť

```
public class Main {  
    public static void main(String[] args) {  
        Student s = new Student("Fero", "SPŠE Prešov");  
        System.out.println(s.getMeno());  
        System.out.println(s.getSkola());  
    }  
}
```

# UML Class diagram

Dedičnosť sa znázorňuje uzatvorenou šípkou smerujúcou k rodičovi



# Dedičnosť

## Vlastnosti

- Vieme dediť iba z jednej triedy
- Objekt podtriedy môžeme použiť všade tam, kde sa očakáva objekt nadradenej triedy
- Atribúty a metódy s modifikátorom prístupu `private` sa nededia



# Dedičnosť

Použitie podtriedy tam, kde sa očakáva nadradená trieda

```
void vypisOsobu(Osoba osoba) {  
    System.out.println(osoba.getMeno());  
}
```

```
Osoba o = new Osoba("Alica");  
Student s = new Student("Bob", "SPŠE Prešov");
```

```
vypisOsobu(o);  
vypisOsobu(s);
```

# Dedičnosť

Konštruktory pri dedení

- Konštruktory sa nededia
- Rodičovský konštruktor voláme pomocou `super()`
- Ak nezavoláme nadradený konštruktor, musíme inicializovať zdedené atribúty sami

# Dedičnosť

Rodičovská trieda `Object`

- `java.lang.Object` je vrcholom hierarchie tried
- definuje metódy `toString` a `equals`
- Triedu `Object` môžeme použiť všade tam, kde akceptujeme objekt akejkoľvek triedy.

```
static void analizuj(Object o) {  
    System.out.println(o.getClass().toString());  
    System.out.println(o.toString());  
    if(o instanceof Osoba) {  
        System.out.println(((Osoba)o).getMeno());  
    }  
    if(o instanceof Student) {  
        System.out.println(((Student)o).getSkola());  
    }  
}
```

```
analizuj(osoba);  
analizuj(student);  
analizuj(67);  
analizuj("Foo");
```

Nový modifikátor prístupu – **protected**

viditeľnosť z rovnakého balíka a aj z podtried

| Modifier  | Class | Package | SubClass | Global |
|-----------|-------|---------|----------|--------|
| Public    | ✓     | ✓       | ✓        | ✓      |
| Protected | ✓     | ✓       | ✓        | ✗      |
| Default   | ✓     | ✓       | ✗        | ✗      |
| Private   | ✓     | ✗       | ✗        | ✗      |

# Prekrytie metód - overriding

Potomok môže prepísať (*anglicky override*) metódu rodiča

Metóda musí mať rovnaký názov, parametre a návratový typ

Odporúča sa použiť anotáciu `@Override`

Zavolanie pôvodnej prekrytú metódy pomocou `super.metoda()`

Statické metódy nie je možné prekryť

# Prekrytie metód - overriding

```
class Osoba {  
    String meno;
```

```
    Osoba(String meno) {  
        this.meno = meno;  
    }
```

```
    void predstavSa() {  
        System.out.println("Som " + meno);  
    }  
}
```

# Prekrytie metód - overriding

```
class Student extends Osoba {  
    String skola;
```

```
    Student(String meno, String skola) {  
        super(meno);  
        this.skola = skola;  
    }
```

```
    @Override  
    void predstavSa() {  
        System.out.println("Študujem na " + skola);  
        super.predstavSa();  
    }  
}
```



# Zabránenie dedenia

Pomocou kľúčového slova **final** môžeme zablokovat možnosť dediť triedu alebo prekryť metódu

```
final class Auto {  
    String model;  
    String vyrobca;  
}
```

```
class Osoba {  
    String meno;  
  
    final String getMeno() {  
        return meno;  
    }  
}
```

# IS-A verzus HAS-A

**IS-A (je)** znamená, že trieda je špeciálnym prípadom inej triedy.  
Vtedy použijeme dedičnosť

**HAS-A (má)** znamená, že trieda obsahuje inú triedu ako atribút.  
Vzniká kompozíciou alebo agregáciou

- Kompozícia používa atribúty
- Auto **je** Vozidlo, ale Auto **má** Motor
- Radšej zvoliť kompozíciu a rozhrania ako dedičnosť

# IS-A versus HAS-A

```
class Adresa {  
    String mesto;  
}
```

```
class Student extends Osoba {    // Student IS-A Osoba  
    Adresa adres;                // Student HAS-A Adresa
```

```
    Student(String meno, String mesto) {  
        this.meno = meno;  
        this.adres = new Adresa();  
        this.adres.mesto = mesto;  
    }  
}
```

# UML Class diagram

Kompozícia sa znázorňuje plným diamantom

