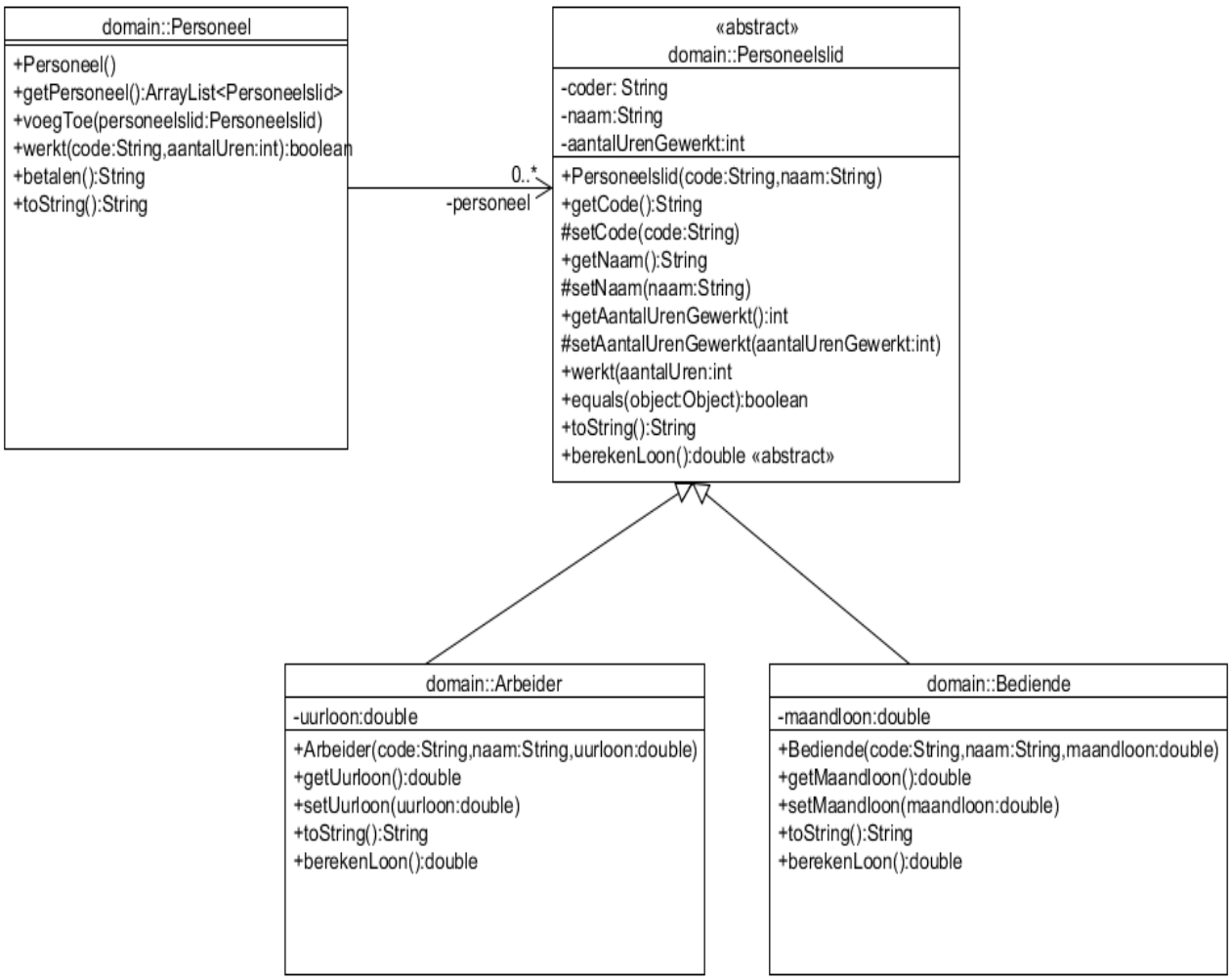
Labo Abstract klassen - Polymorfie

In dit labo leer je abstacte klassen maken en polymorfe oplossingen bedenken

We wensen een app te ontwikkelen voor het berekenen van de lonen voor al de personeelsleden van een bedrijf . De loonberekening voor arbeiders en bedienden is verschillend. Bedienden krijgen een vast maandloon ongeacht het aantal gewerkte uren. Arbeiders hebben een uurloon en worden betaald voor het aantal uren dat ze gewerkt hebben. Indien een arbeider meer dan 38 werkt wordt hij voor de gewerkte uren boven 38 tien procent extra betaald. De overuren van een bediende worden niet betaald.
We moeten kunnen meegeven hoeveel uren ieder personeelslid werkt. Bij betaling van het loon worden de gewerkte uren terug op nul gezet.
Personeelsleden hebben een unieke code. Een personeelslid mag geen tweemaal worden toegevoegd aan het personeel.

Dit resulteert in volgend klassendiagram:



Voorbeeld abstract superklasse:

```
package domain;

public abstract class Personeelslid {
	private String naam;
	private String code;
	private int aantalUrenGewerkt;

	public Personeelslid(String code, String naam)
			throws IllegalArgumentException{
		setCode(code);
		setNaam(naam);
	}

	public String getCode() {
		return code;
	}

	protected void setCode(String code)
			throws IllegalArgumentException{
		if (code == null){
			throw new IllegalArgumentException("Geen code");
		}
		this.code = code;
	}

	public String getNaam() {
		return naam;
	}

	public int getAantalUrenGewerkt(){
		return aantalUrenGewerkt;
	}

	protected void setAantalUrenGewerkt(int aantalUrenGewerkt)
			throws IllegalArgumentException{
		if (aantalUrenGewerkt < 0){
			throw new IllegalArgumentException(
				"Aantal gewerkte uren moet positief zijn");
		}
		this.aantalUrenGewerkt = aantalUrenGewerkt;
	}

	protected void setNaam(String naam)
			throws IllegalArgumentException{
		if(naam == null) throw new IllegalArgumentException(
				"Geen naam");
			this.naam = naam;
	}

	public void werkt(int aantalUren){
		if (aantalUren <= 0){
			throw new IllegalArgumentException(
```

```
                "Aantal gewerkte uren moet strikt positief zijn");
        }
        setAantalUrenGewerkt(getAantalUrenGewerkt() +
                                                aantalUren);
    }

    @Override
    public boolean equals(Object o){
        boolean result = false;
        if(o instanceof Personeelslid){
            Personeelslid p= (Personeelslid)o;
            if (p.getCode().equals(this.getCode())){
                result = true;
            }
        }
        return result;
    }

    @Override
    public String toString(){
        return "Code : " + this.getCode() + " Naam : " +
                                            this.getNaam();
    }

    public abstract double berekenLoon();
}
```

Voorbeeld subklasse:

```
package domain;

public class Arbeider extends Personeelslid {
    private double uurloon;

    public Arbeider(String code, String naam, double uurloon)
                            throws IllegalArgumentException{
        super(code,naam);
        this.setUurloon(uurloon);
    }

    public double getUurloon() {
        return uurloon;
    }

    public void setUurloon(double uurloon)
                        throws IllegalArgumentException{
        if (uurloon <= 0) throw new IllegalArgumentException(
                    "uurloon moet strikt positief zijn");
        this.uurloon = uurloon;
    }
```

```
    @Override
    public double berekenLoon(){
        double loon = getAantalUrenGewerkt() <= 38?
           getAantalUrenGewerkt() * getUurloon():
           38 * getUurloon() +
           (getAantalUrenGewerkt() - 38) * getUurloon() * 1.1;
        return loon;
    }

    @Override
    public String toString() {
        return "Arbeider "+ super.toString() + "
                uurloon:"+getUurloon();
    }
}
```

Voorbeeld polymorfie:

```
package domain;
import java.util.ArrayList;

public class Personeel {
    private ArrayList<Personeelslid> personeel;

    public Personeel(){
        this.personeel= new ArrayList<Personeelslid>();
    }

    public ArrayList<Personeelslid> getPersoneel() {
        return personeel;
    }

    public void voegToe(Personeelslid personeelslid)
                        throws IllegalArgumentException{
        if (personeelslid == null){
            throw new IllegalArgumentException(
                                    "geen personeelslid");
        }
        if (this.personeel.contains(personeelslid)){
            throw new IllegalArgumentException(
             "Personeelslid met zelfde code komt reeds voor");
        }
        this.personeel.add(personeelslid);
    }

    public boolean werkt(String code, int aantalUren)
                          throws IllegalArgumentException {
        if (code == null){
            throw new IllegalArgumentException(
                                     "Code mag niet null zijn");
        }
        Personeelslid personeelslid = zoekPersoneelslid(code);
        if (personeelslid == null){
            throw new IllegalArgumentException(
                                        "code bestaat niet");
```

```
            }
            if (aantalUren <= 0){
                throw new IllegalArgumentException(
                 "aantal gewerkte uren moet strikt positief zijn");
            }
            personeelslid.werkt(aantalUren);
            return true;
        }


        public String betalen() {
            String output = "";
            for (Personeelslid personeelslid : personeel) {
                output += personeelslid.toString() +
                          "\ngewerkte uren: " +
                          personeelslid.getAantalUrenGewerkt() +
                          " loon: "+ personeelslid.berekenLoon() +
                          "\n\n";
                personeelslid.setAantalUrenGewerkt(0);
            }
            return output;
        }

        private Personeelslid zoekPersoneelslid(String code) {
            Personeelslid personeelslidGezocht = null;
            for (Personeelslid personeelslid : personeel) {
                if (code.equals(personeelslid.getCode())){
                    personeelslidGezocht = personeelslid;
                }
            }
            return personeelslidGezocht;
        }

        public String toString(){
            String resultaat = "";
            for (Personeelslid personeelslid: this.personeel){
                resultaat = personeelslid + "\n";
            }
            return resultaat;
        }
}
```

- De klasse Personeelslid is een abstracte klasse. We kunnen van deze klasse geen objecten aanmaken.

- De klasse Personeel bevat een ArrayList van het type Personeelslid waar we objecten kunnen aan toevoegen van alle klassen die overerven van Personeelslid. Wanneer we de klasse Personeel ontwerpen hoeven we niet te weten welke soorten personeelsleden er zijn. In de methodes van de klasse personeel werken we alleen met Personeelslid variabelen. Als we later aan ons model andere soorten personeelsleden toevoegen (nieuwe klassen die overerven van Personeelslid) hoeft de code in de klasse Personeel niet gewijzigd te worden. Dit noemen we ook wel het Open Closed principe.

- Ondanks dat we van de klasse Personeel geen objecten kunnen aanmaken plaatsen we toch een constructor in deze klasse. Deze constructor wordt aangeroepen in de eerste lijn code van de constructoren van de overervende klassen (met super(…)).

- Ondanks dat we voor de berekenLoon methode in de klasse Personeelslid geen zinnige code kunnen schrijven nemen we deze methode toch op in deze klasse en we maken deze methode abstract. Moesten we deze methode niet opnemen in de klasse Personeelslid dan zou de methode betalen() in de Personeel klasse niet werken. Deze methode zal immers voor elk Personeelslid object uit de arraylist de berekenLoon methode aanroepen en verwacht dus dat deze methode gedefinieerd is in de klasse Personeelslid.

- Bij het doorlopen van de arraylist zal automatisch de juiste implementatie van de berekenLoon methode uitgevoerd worden.

- Wanneer een klasse overerft van een abstract klasse moet deze klasse alle abstracte methodes van deze klasse implementeren, zoniet moet de overervende klasse ook als abstract worden gedefinieerd.

- Een klasse die overerft van een abstracte klasse mag natuurlijk ook niet abstracte methodes overriden als die publiek of protected zijn (voorbeeld toString)

PersoneelUI

In deze klasse maken voegen we een aantal arbeiders en bedienden toe aan het personeel, registreren hun gewerkte uren en betalen het personeel.
Vervolgens verhogen we het uurloon van alle arbeiders met 5 %

```
package ui;
import domain.Personeel;
import domain.Arbeider;
import domain.Bediende;
import domain.Personeelslid;
import java.util.ArrayList;

public class PersoneelUI {

	public static void main(String[] args) {
		Personeel personeel = new Personeel();
		try{
			personeel.voegToe(new Bediende("102","Jan", 2000));
			personeel.werkt("102", 30);
		}
		catch (IllegalArgumentException e){
			System.out.println(e.getMessage());
		}
		try{
			personeel.voegToe(new Arbeider("222","An", 40));
			personeel.werkt("222", 30);
		}
		catch (IllegalArgumentException e){
			System.out.println(e.getMessage());
		}
		try{
			personeel.voegToe(new Bediende("104","Piet", 2200));
			personeel.werkt("104", 36);
		}
		catch (IllegalArgumentException e){
			System.out.println(e.getMessage());
		}
		try{
			personeel.voegToe(new Arbeider("226","Bert", 44));
			personeel.werkt("226", 40);
		}
		catch (IllegalArgumentException e){
			System.out.println(e.getMessage());
		}

		System.out.println(personeel.betalen());

		String output = "";
		ArrayList <Personeelslid> personeelLijst =
				personeel.getPersoneel();
```

```
            for (Personeelslid personeelslid:personeelLijst){
                 if (personeelslid instanceof Arbeider){
                      Arbeider arbeider = (Arbeider) personeelslid;
                      output += arbeider.getNaam()+ " Oude uurloon:"
                                          +arbeider.getUurloon();
                      arbeider.setUurloon(
                                        arbeider.getUurloon() * 1.05);
                      output += "  Nieuwe uurloon:
                                           "+arbeider.getUurloon()+"\n";
                 }
            }
            System.out.println(output);
      }
}
```

- Wij bepalen "at runtime" welke soorten personeelsleden wij willen toevoegen aan personeel.
- Bij het verhogen van het uurloon van de arbeiders moeten we bij het doorlopen van de arraylist eerst met instanceof gaan kijken of het een arbeider betreft.
- Om het uurloon van de arbeider te verhogen gebruiken we de getUurloon en setUurloon methodes uit de klasse Arbeider. Dit zijn specifieke methodes voor de klasse arbeider die niet voorkomen in de superklasse Personeelslid. Om deze methodes te kunnen toepassen moet we dus eerst het opgehaalde Personeelslid object uit de arraylist gaan casten.

Juist of fout (indien fout hoe verbeteren?):

```
PersoneelsLid personeelsLid = new Bediende ("p1", "Rik", 3500);
personeelsLid.setMaandloon(3505);
```

```
Bediende personeelsLid = new Bediende("p2", "Tom", 3400);
personeelsLid.setMaandLoon(3405);
```

```
Object personeelsLid = new Bediende("p2", "Tom", 3400);
personeelsLid.setMaandLoon(3405);
```

```
PersoneelsLid personeelsLid = new BediendeTijdelijk("p1", "Rik",
3500, 12);
```
(BediendeTijdelijk is een subklasse van Bediende met als 3de parameter de duurtijd van contract in maanden. De salarisberekening is dezelfde als voor een gewone bediende, de klasse heeft dus geen berekenLoon methode)
```
double salaris = personeelsLid.berekenLoon();
```

```
double totSalaris = 0.0;
for (PersoneelsLid personeelsLid : personeel)
     totSalaris+= personeelsLid.berekenLoon();
```

```
Arbeider personeelsLid = new PersoneelsLid("Frans");
System.out.println(personeelsLid);
```