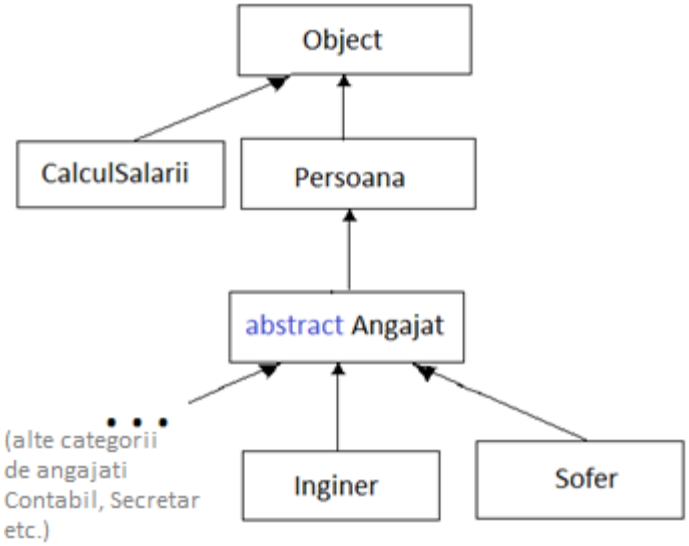
Utilizarea unei clase abstracte Angajat

Vom rezolva o parte din Laboratorul 2 in care angajatii au si un ID. Ierarhia de clase este urmatoarea



Clasa abstractă este o clasă care conține cel putin o metodă abstractă (nedefinită încă). O astfel de clasă are un caracter de generalitate pentru clasele care o vor extinde deoarece in ea se găsesc:

- metodele comune tuturor angajatilor (setNrOreLucrate(), salariu() – componenta comună, cea care calculează salariul după nr. de ore lucrate (toti angajații sunt platiti in primul rand dupa nr. de ore lucrate, apoi urmeaza bonusurile)
- metodele abstracte, care trebuie implementate obligatoriu în clasele derivate (Inginer, Sofer, …) deoarece acestea *depend de specificul* categoriei de angajati (aici doar getNrMaxLucrate())

Clasa abstractă fiind un supertip pentru clasele derivate este utilizata in construirea unor tablouri polimorfe. Exemplu:

```
Angajat[] salariati = new Angajat[]{ing1,ing2,sofr};
```

Toate elementele dintr-un tablou polimorf sunt tratate unitar pe partea comuna a acestora

```
for(int i=0; i<salariati.length; i++)
    salariati[i].setNrOreLucrate( ore[i]);
//...

double salariuTotal=0;
for(Angajat p: salariati)
    salariuTotal += p.salariu();
```

Exercitiu. Comparati clasa abstracta cu interfata IAngajat si cele 2 implementari cerute de lucrare.

Clasa CalculSalarii

```
package ro.usv;

public class CalculSalarii {
    public static void main(String arg[]){
        Inginer ing1=new Inginer("Barbulescu", "Barbu",101);
        Inginer ing2=new Inginer("Trestie", "Tudor", 23);
        Sofer   sofr= new Sofer("Repede","Raul", 302);

        Angajat[] salariati = new Angajat[]{ing1,ing2,sofr};
        int[] ore = new int[]{100,200,250};

        for(int i=0; i<salariati.length; i++)
            salariati[i].setNrOreLucrate(  ore[i] );

        sofr.setKm_parcursi(1000);  // demo ca in salariati[2] este ref.
                                    // la acest obiect si nu un obiect
        double salariuTotal=0, sal;
        for(Angajat p: salariati){
            salariuTotal += (sal = p.salariu());
            System.out.println(p  + " - salariu=" + sal);
        }
        System.out.println("Total salarii=" + salariuTotal);
    }
}
```

Clasa Persoana

```
package ro.usv;

public class Persoana {
    private String nume;
    private String prenume;

    public Persoana(String nume, String prenume) {
        this.nume = nume;
        this.prenume = prenume;
    }
    @Override
    public String toString() {
        return nume + " " + prenume;
    }

}
```

Clasa Angajat

```
package ro.usv;

public abstract class Angajat extends Persoana{
    private static double salariu_orar_minim=15;
    private int nrOreLucrate;
    private int id;

    public abstract int getNrMaxOreLucrate();

    public Angajat(String nume, String prenume, int id) {
        super(nume, prenume);
        this.id = id;
    }
```

```
    public void setNrOreLucrate(int nrOreLucrate){  //interesant
        this.nrOreLucrate = Math.max(nrOreLucrate, 0);
        this.nrOreLucrate = Math.min(this.nrOreLucrate, getNrMaxOreLucrate());
    }

    public double salariu() {
        return salariu_orar_minim * nrOreLucrate * getCoeficientSlarial();
    }

    public double getCoeficientSlarial(){
        return 1;
    }

    @Override
    public String toString() {
        return getClass().getSimpleName() + " " + super.toString() +
                " (ID "+ id+") "+ " a lucrat " + nrOreLucrate+ " ore";
    }
}
```

Clasa Inginer

```
package ro.usv;

public class Inginer extends Angajat {
    private static final int nrMaxOreLucrate = 250;
    private double coeficientSlarial = 1.5;

    public Inginer(String nume, String prenume, int id) {
        super(nume, prenume, id);
    }

    @Override
    public int getNrMaxOreLucrate() {
        return nrMaxOreLucrate;
    }

    public double getCoeficientSlarial() {
        return coeficientSlarial;
    }

}
```

Observati ca aceasta clasa nu are toString(), dar clasa Sofer are.

Clasa Sofer

```
package ro.usv;

public class Sofer extends Angajat{
        private static final int nrMaxOreLucrate = 300;
        private static final int kmParcursiMax = 5000;
        private int kmParcursi=0;

        public Sofer(String nume, String prenume, int id) {
            super(nume, prenume, id);
        }

        @Override
        public int getNrMaxOreLucrate() {
            return nrMaxOreLucrate;
        }

        @Override
        public double salariu() {
            return super.salariu() + kmParcursi * 0.1;
        }
```

```
        public void setKm_parcursi(int kmParcursi) {
            this.kmParcursi = Math.max(kmParcursi, 0);
            this.kmParcursi = Math.min(this.kmParcursi, kmParcursiMax);
        }

        @Override
        public String toString() {
            return super.toString() +", a parcurs "+kmParcursi+ " km";
        }
}
```

Observati ca in clasele derivate s-a adus tot ce era specific (campuri si metode) categoriilor de salariati pe care le modeleaza.

Iesirea produsa este cea ceruta de lucrare la care s-a adaugat ID pentru a face legatura cu partea a treia in care se citeste fisierul pontaj.txt

```
Inginer Barbulescu Barbu (ID 101)  a lucrat 100 ore - salariu=2250.0
Inginer Trestie Tudor (ID 23)  a lucrat 200 ore - salariu=4500.0
Sofer Repede Raul (ID 302)  a lucrat 250 ore, a parcurs 1000 km -
salariu=3850.0
Total salarii=10600.0
```

Pentru rezolvarea partii cu citire pontaj din fisier se va avea in vedere ca in situatii reale crearea descrierii angajatilor – aici crearea obiectelor – se face **o singura data**, iar calculul salariilor – pe baza unui pontaj – **se face periodic** (lunar).

Observatie. Bineînțeles că polimorfismul tabloului *salariati[]* putea fi realizat și printr-o clasă **concretă,** fără metode abstracte.

Astfel clasa abstractă **Angajat** ar putea fi transformată într-una concretă definind metoda `getNrMaxOreLucrate()` ca una concretă astfel:

```
public int getNrMaxOreLucrate(){
    if (this instanceof Inginer)
        return Inginer.nrMaxOreLucrate;
    else if (this instanceof Sofer)
        return Sofer.nrMaxOreLucrate;
    else
        return defaultValue;
}
```

În acest caz variabila statică *nrMaxOreLucrate* trebuie declarată public în clasele **Inginer** și **Sofer**, iar aceste clase nu ar mai conține metoda `getNrMaxOreLucrate()`.

Această abordare nu este recomandabilă deoarece conduce la un cod stufos și scade din generalitatea abordării și specificul claselor derivate afectând dezvoltările ulterioare ale aplicației.

Avantajul utilizării unei **clase abstracte Angajat** se vede atunci când ar trebui introdusă o nouă clasă, de exemplu Economist. În acest caz vom scrie doar noua clasă, **fără a fi nevoiți să operăm modificări în restul codului**.

Dacă **Angajat** ar fi o **clasă concretă** cu o metodă `getNrMaxOreLucrate()` ca cea de mai sus este clar că la apariția clasei Economist trebuie să intervenim în această metodă adăugând un nou

```
else if (this instanceof Economist)…
```

Și acesta este doar un exemplu simplu.