

Klasių hierarchija

Mindaugas Karpinskas
2018



Klasių hierarchija

- Visos klasės paveldi klasės **java.lang.Object**
- Jei nėra nurodyta tėvinė klasė, tai pagal nutylėjimą paveldės Object savybes
- Jei kokia nors tėvinė klasė nurodyta, tada paveldės Object savybes per tėvinę klasę

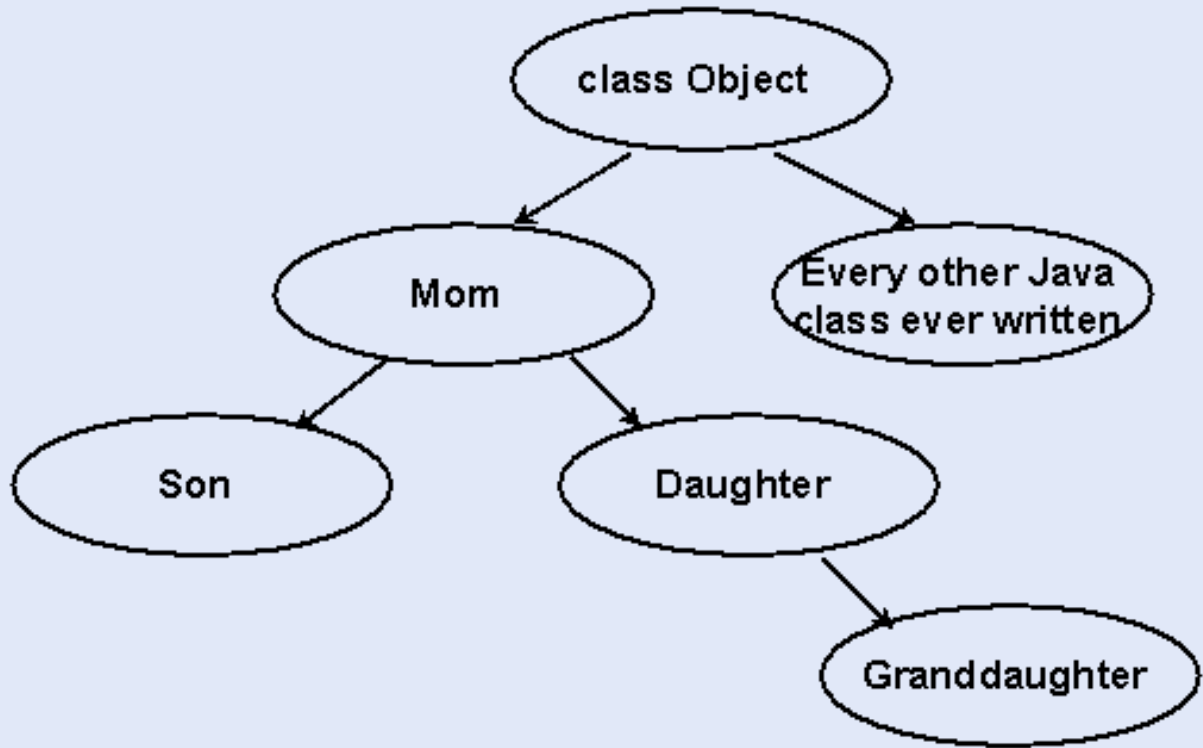
Pvz.

```
class Mom {  
    // declarations, definitions  
}  
class Son extends Mom {  
    // declarations, definitions  
}  
class Daughter extends Mom {  
    // declarations, definitions  
}
```

- Son ir Daughter klasės paveldės Mom klasės kintamuosius ir metodus
- Mom yra pagrindinė klasė, į kurią remiasi kitos dvi klasės.
- Son ir Daughter yra Mom klasės poklasės (subclasses) ir Mom yra Son ir Daughter. superklasė (superclass)
- Java kalboje kiekviena klasė turi tik **vieną** tiesioginę superklasę

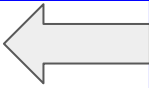
Mom klasės superklasė?

Paveikslėlis iliustruoja šio klausimo atsakymą. Trumpai tariant visų klasių hierarchijos viršūnėje yra Object klasė ir, jei nenurodytas paveldimumas, tai nutylimasis paveldimumas yra **extends** Object



Taigi mūsų hierarchijos pavyzdys ekvivalentus tokiam kodui:

```
class Mom extends Object {  
    // declarations, definitions  
}  
class Son extends Mom {  
    // declarations, definitions  
}  
class Daughter extends Mom {  
    // declarations, definitions  
}
```



- Kodėl tokia globali hierarchija naudinga?
 - kadangi žinome, kad visos klasės turi bendrą superklasę Object, jos metodus ir kintamuosius gali naudoti bet kuri kita klasė.
- Object klasė apibrėžia **equality** metodą skirtą patikrinti, ar dviejų klasių turinys vienodas.
- Taip pat Object klasėje yra realizuotas daugiagijškumo (multithreading) savybės.
- Taip pat atpuola daugelio hierarchijų tarpusavio sąsajų problema, nes visos jos yra vienos globalios hierarchijos dalys.
- Naudinga ir tai, kad esame garantuoti, kad kiekviena klasė turi savo superklasę.

Keletas klasės **Object** metodų

- equals – lygina objektų turinį
- clone – sukuria naują objektą - to paties objekto kopiją
- toString – gražina tekstinę objekto informaciją

Metodo equals pavyzdys

```
class A {  
}
```

instanceof - patikrina kokios klasės objektas

B b = (B) obj;

(B) - **cast**, traktuojama, kad objektas gali būti priskirtas kitos klasės kintamajam

```
class B {  
    int i;  
    A a = new A();  
    @Override  
    public boolean equals(Object obj) {  
        if (obj instanceof B) {  
            B b = (B) obj;  
            if ((i == b.i) &&  
                a.equals(b.a))  
                return true;  
            else  
                return false;  
        }  
        return false;  
    }  
}
```

Užduotis 1

Sukurti klasę Asmuo, su dviem laukais: vardas, pavardė. Klasėje perrašyti equals metodą.

Main metode sukurti keletą egzempliorių / objektų ir palyginti juos.

Metodo toString pavyzdys

```
class A {  
    // toString....  
}
```

```
class B {  
    int i;  
    A a = new A();  
  
    public String toString() {  
        return "i=" + i + " a=" + a.toString();  
    }  
}
```

Užduotis2

Sukurti klasę `Automobilis`, su dviem laukais: `valstNr`, `marke`. Klasėje perrašyti `toString` metodą.

Sukurti klasę `AutoParkas` kuri turi deklaruotą kintamąjį/masyvą `Automobilis[] autos`; `AutoParkas` taip pat perrašo metoda `toString` ir gražina visą info apie automobilius, kiek automobilių, koks pirmas, koks paskutinis, visų automobilių info...

Main metode sukurti keletą egzempliorių / objektų ir išspausdinti `toString` rezultatą.

```
public class Auto {  
    private final String valNr;  
    private final String marke;  
    public Auto(String valNr, String marke) {  
        this.valNr = valNr;  
        this.marke = marke;  
    }  
    @Override  
    public String toString() {  
        return "Valstybinis numeris: " + this.valNr  
            + ", marke: " + this.marke + ";"  
    }  
}
```

```

public class AutoParks {
    private Auto[] autos = new Auto[5];
    public void setAutos(Auto[] autos) {
        this.autos = autos;
    }
    public void prideti(String valNr, String mar) {
        for (int i = 0; i < autos.length; i++) {
            if (autos[i] == null) {
                autos[i] = new Auto(valNr, mar);
                return;
            }
        }
    }
    @Override
    public String toString() {
        String s = "";
        s += "Mindaugo automobilių parkas\n";
        s += "Iš viso automobilių parke: " + this.autos.length + "\n";
        s += "Pirmas automobilis: " + this.autos[0] + "\n";
        s += "Paskutinis automobilis: " + this.autos[this.autos.length - 1] + "\n";
        s += "Visi automobiliai: " + Arrays.toString(this.autos);
        return s;
    }
}

```

```
public static void main1(String[] args) {  
    AutoParks autoParkas = new AutoParks();  
    autoParkas.prideti("AAA123", "Audi");  
    autoParkas.prideti("BBB123", "Audi");  
    autoParkas.prideti("CCC123", "Audi");  
    autoParkas.prideti("DDD123", "Audi");  
    autoParkas.prideti("EEE123", "Audi");  
    System.out.println(autoParkas.toString());  
  
}
```

```
public static void main(String[] args) {  
    AutoParks autoParkas = new AutoParks();  
    Auto[] autos = new Auto[3];  
    autos[0] = new Auto("AAA001", "Auduke");  
    autos[1] = new Auto("AAA002", "Prius");  
    autos[2] = new Auto("AAA003", "Mokka");  
    autoParkas.setAutos(autos);  
    System.out.println(autoParkas.toString());  
  
}
```

clone*

Metodo clone 1 pavyzdys (*tik pavyzdys!)

```
class A {  
    protected A clone() {  
        return new A();  
    }  
}
```

```
class B {  
    int i;  
    A a = new A();  
  
    protected B clone() {  
        B b = new B();  
        b.i = i;  
        b.a = a.clone();  
        return b;  
    }  
}
```

Metodo clone 2 pavyzdys (*tik pavyzdys!)

```
class A {  
    protected A clone() {  
        return new A();  
    }  
}
```

```
class B {  
    int i;  
    A a = new A();  
  
    B(int i, A a) {  
        this.i = i;  
        this.a = a.clone();  
    }  
  
    protected B clone() {  
        return new B(i, a);  
    }  
}
```


Užduotis3*

Sukurti klasę Adresas, su dviem laukais: gatvė, numeris. Sukurti klasę Studentas, su dviem laukais: vardas, pavardė, adresas. Klasėje perrašyti clone metodą.

Main metode sukurti keletą egzempliorių / objektų ir sukurti jų klonus / kopijas. Patikrinti ar objektai sutampa, ar objektų laukų reikšmės sutampa...

Objekt metodai

Ką išmokome:

Equals

toString

Clone*

Paveldėjimo išnaudojimas

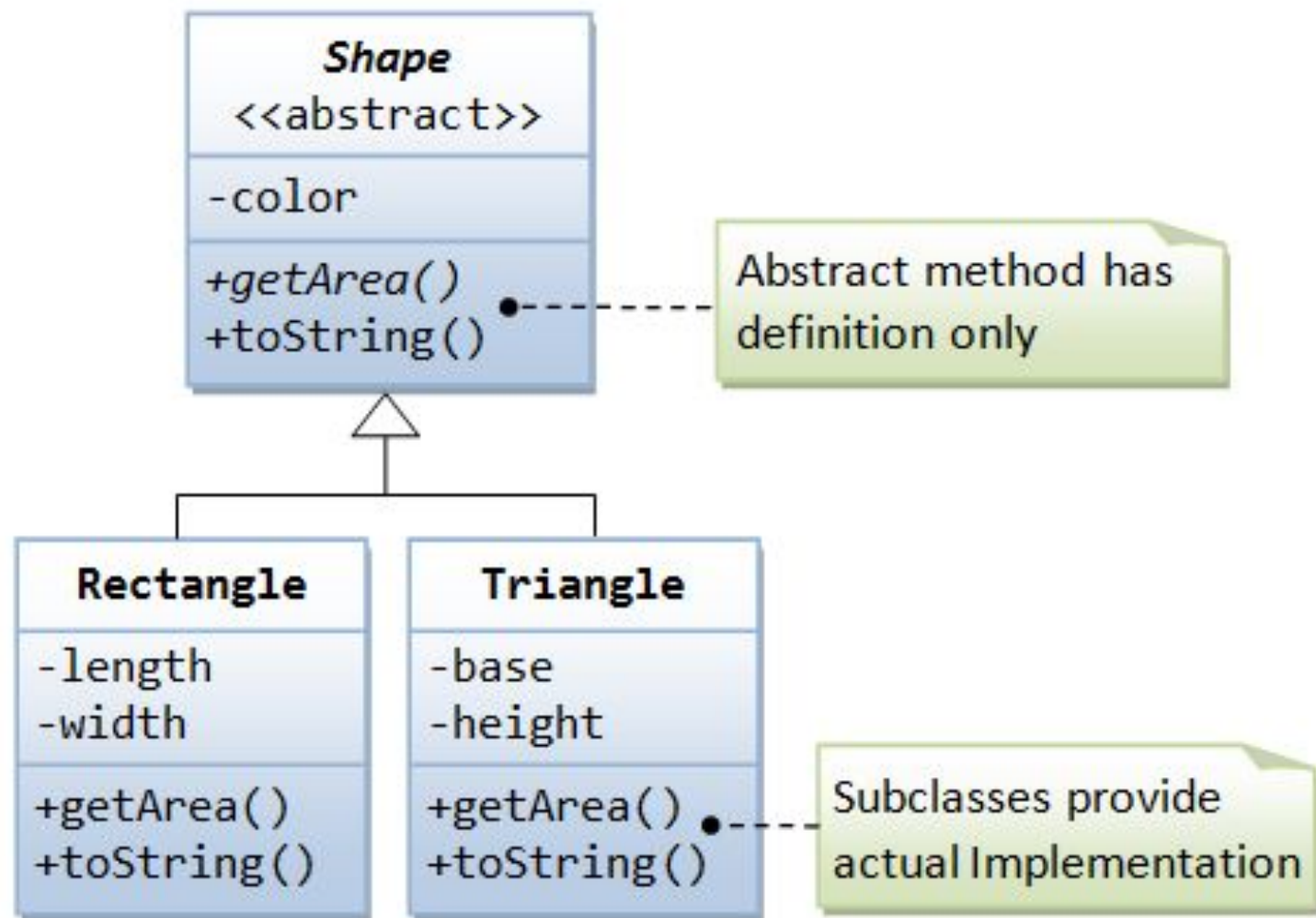
Susipažinome su dviem Java objektinio programavimo sąvokomis: duomenų apsauga ir apgauba.

Dabar pakalbėsime apie paveldėjimą. Paveldėjimas Javoje gali būti reguliuojamas naudojant **abstrakčias** klases ir metodus. Šie Java elementai naudingi kuriant Java klasių hierarchiją.

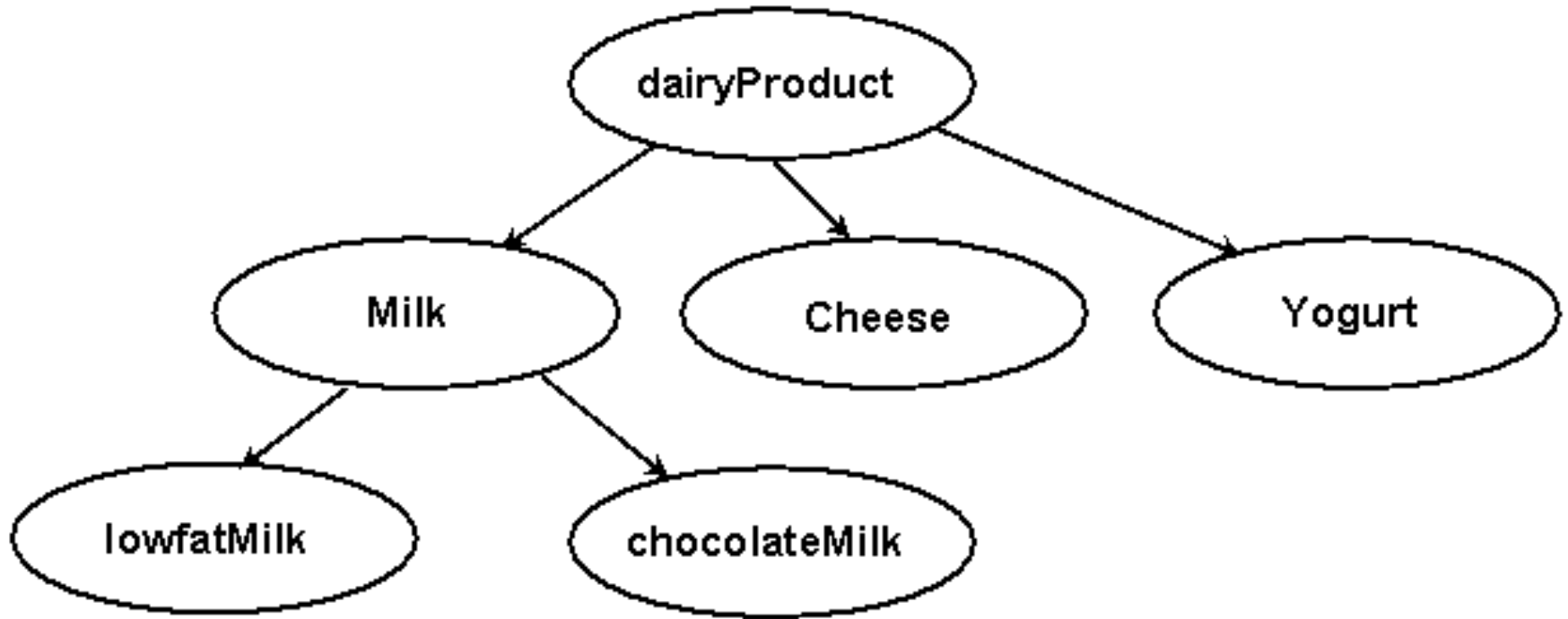
Klasių hierarchijos struktūrizavimas

Paveldėjimo privalumą mes grindėme tuo, kad jis leidžia neperrašinėti iš naujo klasės kodo ir visą jį paveldėti poklasėms. Tačiau tai tik viena naudinga paveldimumo savybė.

Kitas paveldimumo panaudojimas yra susietas su klasių hierarchijos konstravimu.



Pieno produktų hierarchijos pavyzdys



Pieno produktų atveju bus būtina pasirūpinti metodu, kuris praneš ar produktas tinkamas naudoti.

Iliustracijai parašykime **abstraktų** kodą:

```
public class DairyProduct {
```

```
//Kintamieji, konstruktoriai
```

```
public boolean sourThisWeek() {  
    //atitinkamas kodas  
}
```

```
public void putOnSale() {  
    //kodas pateikiantis  
    produktą vartojimui  
}
```

```
}
```

cast

Duomenų tipo pakeitimas (casting). Tarkime turėjome lowfatMilkType kintamąjį ir jį norime pakeisti dairyProduct tipu. Tai galima padaryti taip:

```
LowfatMilk m=new LowfatMilk();  
DairyProduct d=m;  
if (d.sourThisWeek()) (  
    System.out.println("Nepirk");}
```


Hierarchyjos nauda

Tarkime mūsų parduotuvės valdytojas nori patikrinti, kokie pieno pakeliai suges šią savaitę. Tokius produktus reikėtų skubiai pateikti prekybai. Analogiškus veiksmus galėsite atlikti su LowfatMilk, Milk, Cheese ir Yogurt objektais panaudodami vieną ir tą patį metodą:

```
public void dumpSourGoods(DairyGood d) {  
    if (d.sourThisWeek()) {  
        d.putOnSale();  
    }  
}
```

Jeigu nebūtume pirma sukūrę struktūrizuotos klasės, mums būtų tekę kiekvieno tipo produktui parašyti atskirą metodą.

Abstrakčios klasės ir metodai

Aprašytame pavyzdyje mes sukūrėme klasių hierarchiją, kuri buvo naudinga pritaikant tą patį metodą įvairioms produktų rūšims. Bet mūsų DairyProduct klasė turi metodus, kurių turinys neaprašytas. Kai mes juos rašėme turėjome omenyje, kad jie bus perkrauti poklasėse. Tačiau kitas programuotojas gali nesuprasti jūsų intencijos ir nerealizuoti reikiamų metodų. Kad išvengti tokių programavimo klaidų, Java programuotojams siūlo specialų **abstract** modifikatorių.

Kai panaudojate abstract modifikatorių, visos poklasės yra priverčiamos realizuoti abstrakčius metodus.

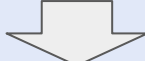
DairyProduct su **abstract** metodais

```
public class DairyProduct {  
  
    //kintamieji, konstruktoriai  
  
    public abstract boolean sourThisWeek();  
  
    //kiti metodai  
    public abstract void putOnSale();  
  
}
```


DairyProduct su **abstract** klasė

DairyProduct klasė vis dar gali būti inicijuojama kitose klasėse.

Tačiau mes galime paskelbti ir ją abstrakčia, kad uždrausti jos tiesioginį inicijavimą:



```
public abstract class DairyProduct {  
  
    //kintamieji, konstruktoriai  
  
    public abstract boolean sourThisWeek();  
  
    //kiti metodai  
    public abstract void putOnSale();  
  
}
```



Be body

Abstract sitanksè

Isiminti

```
abstract class ClassName {  
  
    public void printA(){  
        System.out.println("A");  
    }  
  
    public abstract void printB();  
  
}
```

Uzduotis4

Sukurti abstrakčia klasę Asmuo, su dviem laukais: vardas pavarde. Klasėje perrašyti toString metodą.

Sukurti abstraktų metodą: spausdinkInformacija();

Sukurti klases Studentas, Destytojas, kurios paveldi Asmuo klasę.

Main metode sukurti keleta egzempliorių/objektų ir iškviesti spausdinkInformacija() metodą.

Uzduotis5

- Trys klasės:
 - Asmuo ->
 - Studentas
 - Destytojas
- Sukurti metodą, kuris turi parametą masyvą: Asmuo[] asmenys;
 - Metodas turi patikrinti ar perduotame masyve visos reikšmės skirtingos. Grąžinti **true** reikšmę tik tuo atveju jei visos reikšmės masyve yra unikalios (skirtingos);
- Main metode išbandyti metodą, masyvą užpildyti Studentas, Destytojas egzemplioriais.
- pvz
- for ...
 - for ...

```
public abstract class Asmuo {  
    private final String vardas;  
    private final String pavarde;  
    public Asmuo(String vardas, String pavarde) {  
        this.vardas = vardas;  
        this.pavarde = pavarde;  
    }  
    @Override  
    public boolean equals(Object obj) {  
        if (obj instanceof Asmuo) {  
            Asmuo asmuo = (Asmuo) obj;  
            boolean ar = asmuo.pavarde.equals(this.pavarde)  
                && asmuo.vardas.equals(this.vardas);  
            return ar;  
        }  
        return false;  
    }  
    @Override  
    public String toString() {  
        return "Vardas: " + this.vardas  
            + " pavarde: " + this.pavarde;  
    }  
}
```



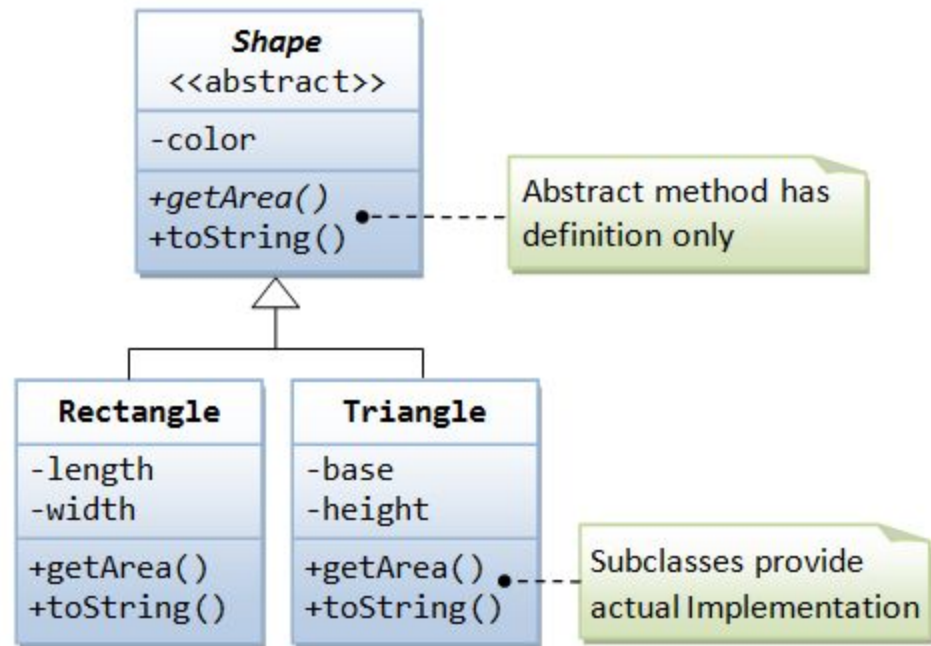
```
public class Destytojas extends Asmuo {  
  
    public Destytojas(String vardas, String pavarde) {  
        super(vardas, pavarde);  
    }  
  
    @Override  
    public boolean equals(Object obj) {  
        return obj instanceof Destytojas && super.equals(obj);  
    }  
  
}
```

```
public class Studentas extends Asmuo {  
  
    public Studentas(String vardas, String pavarde) {  
        super(vardas, pavarde);  
    }  
  
    @Override  
    public boolean equals(Object obj) {  
        return obj instanceof Studentas && super.equals(obj);  
    }  
}
```

```
public class MainTest {  
    public static void main(String[] args) {  
        Asmuo[] asmenys = { new Destytojas("Jonas", "Jonaitis")  
            , new Studentas("Jonas", "Jonaitis") };  
        boolean ar = patikrinkArUnikalu(asmenys);  
        System.out.println(ar);  
    }  
    private static boolean patikrinkArUnikalu(Asmuo[] asmenys) {  
        for (int i = 0; i < asmenys.length; i++) {  
            for (int j = i + 1; j < asmenys.length; j++) {  
                if (asmenys[i].equals(asmenys[j])) {  
                    return false;  
                }  
            }  
        }  
        return true;  
    }  
}
```

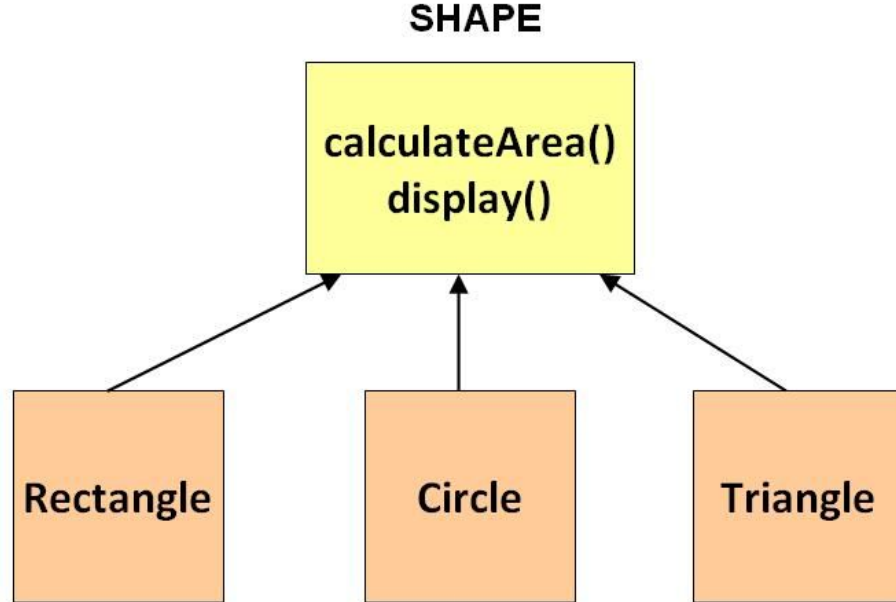
U6 Užd. geo. figūros

- Trys klasės: Figura, Keturkampis, Trikampis
- Figūra turi
 - abstraktų metodą plotas(); kur kitos vaikinės klasės turi apskaičiuoti ir grąžinti figūros plotą
 - lauką spalva
 - toString() metode reikia atspausdinti figūros spalvą ir plotą
- Testavimo klasėje
 - reikalingas metodas kuris turėtų parametą Figūra, ir mokėtų patikrinti ar perduotos figūros plotas nėra didesnis nei 50. Grąžina tru/false.
 - Main metode patestuoti



U 7Užd. geo. figūros

Atsiranda nauja figūrą Apskiritimas.



- Testavimo klasėje

- reikalingas metodas kuris turėtų parametą Figūra, ir mokėtų patikrinti ar perduotos figūros plotas nėra didesnis nei 50. Grąžina tru/false.
- Main metode patestuoti

```
public class TestavimoKlase {  
    public static void main(String[] args) {  
        Figura ket = new Keturkampis("raudona", rnd(), rnd());  
        System.out.println(ket);  
        System.out.println("Ar keturkampio plotas nėra didesnis už 50 kv.m.? "  
                               + plotoTikrinimas(ket));  
  
        Figura trikampis = new Trikampis("Žalia", rnd(), rnd());  
        System.out.println(trikampis);  
        System.out.println("Ar trikampio plotas nėra didesnis už 50 kv.m.? "  
                               + plotoTikrinimas(trikampis));  
  
        Figura apskritimas = new Apskritimas("geltona", rnd());  
        System.out.println(apskritimas);  
        System.out.println("Ar apskritimo plotas nėra didesnis už 50 kv.m.? "  
                               + plotoTikrinimas(apskritimas));  
    }  
    public static double rnd() {  
        Random random = new Random();  
        return random.nextInt(14) + 1;  
    }  
    public static boolean plotoTikrinimas(Figura figura) {  
        if (figura.plotas() < 50) {  
            return true;  
        }  
        return false;  
    }  
}
```

Linkai

<http://docs.oracle.com/javase/tutorial/java/landl/abstract.html>

kiti

<http://www.java-made-easy.com/polymorphism-in-java.html>

<https://www.youtube.com/watch?v=0xw06loTm1k>

<https://www.youtube.com/watch?v=d4Q5GrevAtw>

<https://www.youtube.com/watch?v=-rfGmXbKDgw>

https://www.youtube.com/watch?v=A8Dlmt_qlzk

?

Toliau polimorfizmas