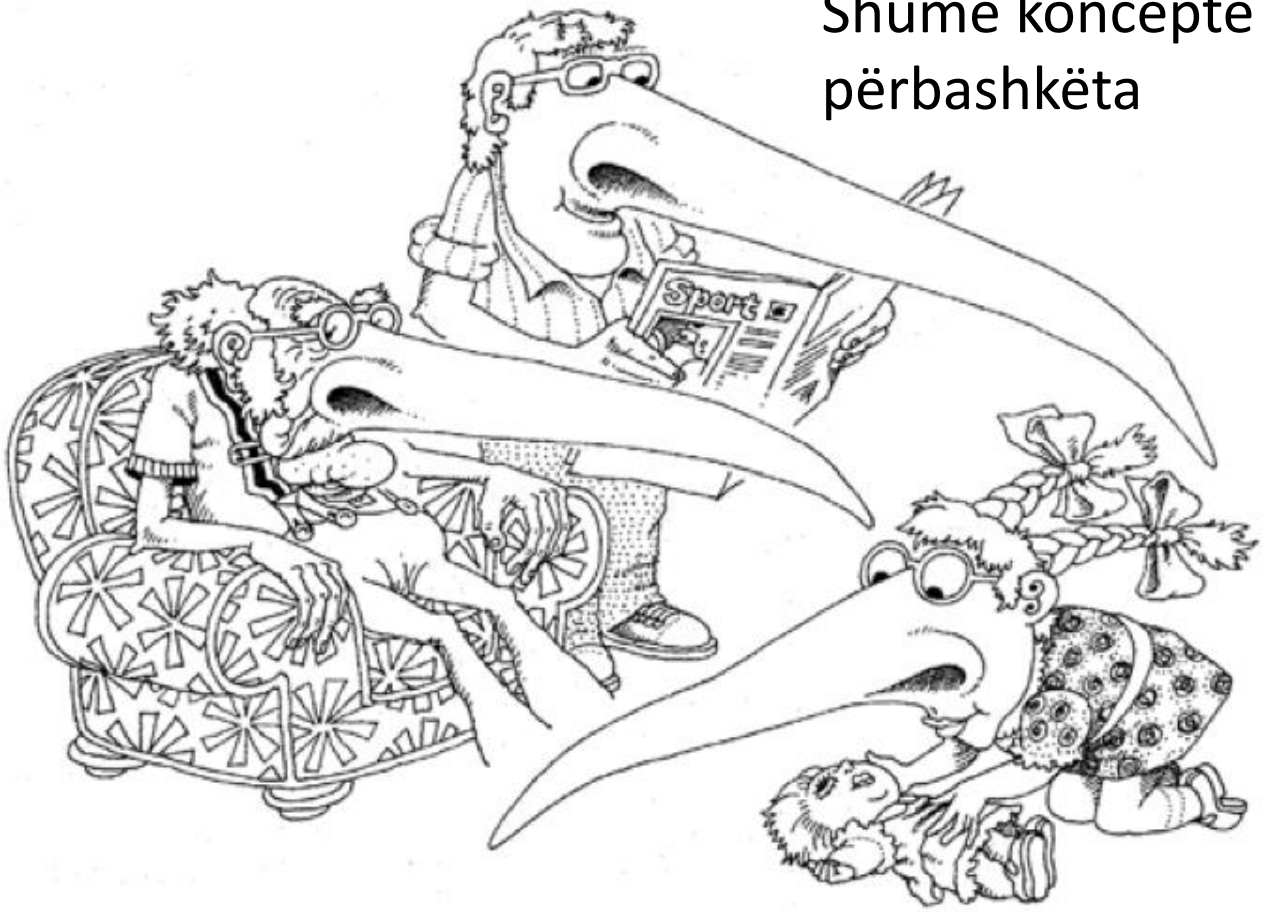
Trashegimia

Lidhjet midis klasave

- Klasat përmbledhin atribute dhe funksione në një njësi të vetme
- Klasat nuk funksionojne të vetme në një program OO
- Dy mënyrat më të përdorura për të lidhur klasat me njëra tjetrën janë:
  - Trashëgimia
  - Kompozimi (Agregimi)

Trashëgimia

Disa koncepte janë të përgjithshëm
Të tjerë janë më specifikë
Shumë konceptë ndajnë tipare të përbashkëta

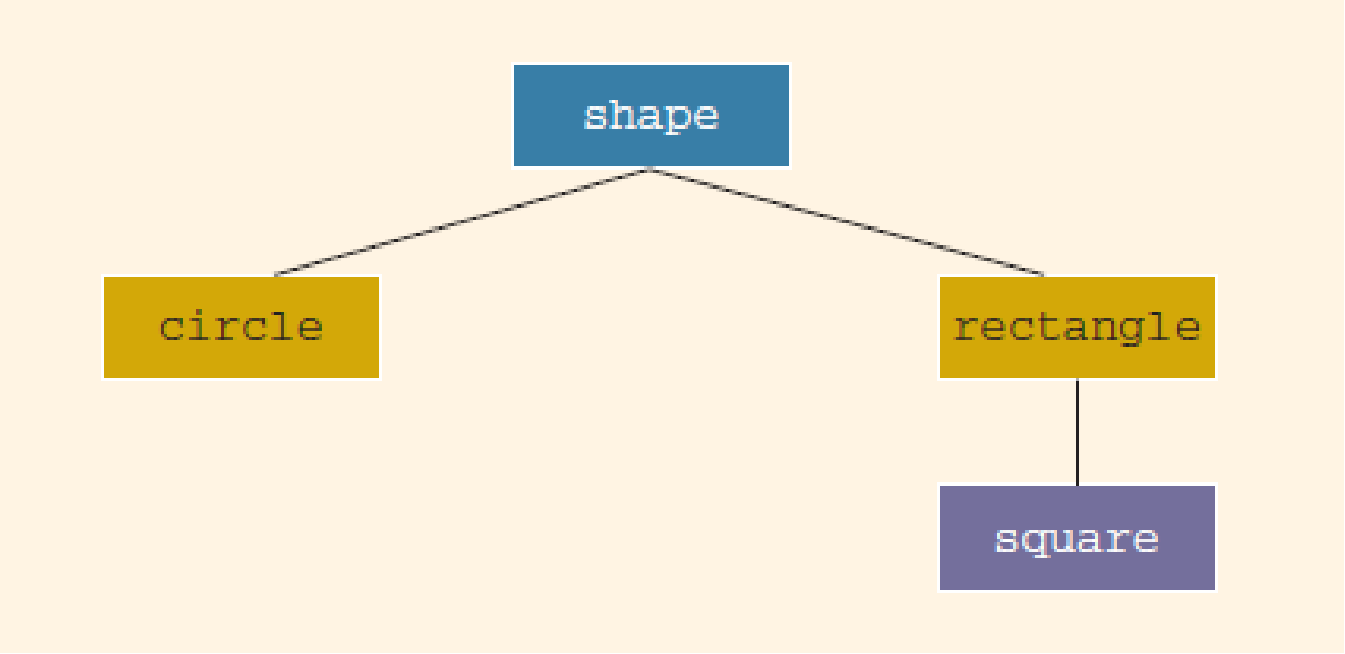


Trashëgimia

- Trashëgimia shpreh lidhjet semantike përgjithësuese/specializuese midis koncepteve
- Në gjuhët OO është e mundur të deklarohen klasa të cilat specializojnë klasa më të përgjithshme
- Klasa e përgjithshme quhet **klasë bazë** ose **superklasë**
- Klasa e specializuar quhet **klasë e derivuar**
- Klasat e derivuara trashëgojnë cilësitë e superklasave

Trashëgimia

- Trashëgimia krijon një strukturë hierarkike midis klasave
- Llojet e trashëgimisë
  - Trashëgimi e njëfishtë
  - Trashëgimi e shumëfishtë



Pse është e nevojshme trashëgimia?

- Klasat ashtu si objektet nuk gjenden të izoluara. Për një domain problemi specifik, abstraksionet gjenden të lidhura në shumë mënyra interesante që formojë edhe **strukturën e klasave**.
- Ka dy arsye pse krijohen lidhjet midis klasave:
  - **Një lidhje midis klasave mund të tregojë një lloj bashkëndarje informacioni midis klasave**. P.sh trëndafilat dhe zambakët janë lule, që do të thotë që të dy kanë petale, një ngjyrë dhe aromë të caktuar.
  - **Një lidhje midis klasave mund të tregojë një lidhje semantike midis klasave**. P.sh trëndafilat e bardhë dhe trëndafilat e kuq janë më të ngjashëm se sa trëndafilat e kuq dhe zambakët.
- Përfitime:
  - Ripërdorim i kodit
  - Imponim i lidhjeve midis koncepteve në nivel programimi

Trashëgimia në C++

1. Anëtarët e klasave bazë janë gjithashtu anëtarë të klasave të derivuara.

2. Anëtarët privatë të një klase bazë nuk mund të aksesohen direkt nga anëtarët e klasave të derivuara. Kur shkruhen implementimet e funksioneve të klasave të derivuara, nuk është e mundur të aksesohen anëtarët privatë të klasave bazë.

Trashëgimia në C++

3. Klasat e derivuara mund të përfshijnë anëtarë shtesë (të dhëna ose funksione)

4. Klasat e derivuara mund të riimplementojnë anëtarët publikë të klasës bazë. D.m.th. në klasën e derivuar mund të krijohet një anëtar funksion me të njëjtin emër dhe listë parametrash formalë si një funksion i klasës bazë. Riimplementimi aplikohet vetëm tek objektet e klasës së derivuar dhe jo objekteve të klasës bazë.

Trashëgimia në C++

```
class className: memberAccessSpecifier baseClassName
{
    member list
};
```

- memberAccessSpecifier:
  - private
  - public
  - Protected

```
class circle: public shape
{
    .
    .
    .
};
```

```
class circle: private shape
{
    .
    .
    .
};
```

Nqs nuk percaktohet asnje specifikator aksesi, si default trashegimia eshte private

Trashëgimia si public, private, protected

```
class B: memberAccessSpecifier A
{
     .
     .
     .
};
```

- Nqs memberAccessSpecifier është public atëherë:
  - Anëtarët **public** të A janë anëtarë **public** të B
  - Anëtarët **protected** të A janë anëtarë **protected** të B
  - Anëtarët **privatë** të A janë **te fshehur** ndaj anetareve të B. Ato mund te aksesohen nga B vetem nepermjet funksioneve public apo protected te A
- Nqs memberAccessSpecifier është protected atëherë:
  - Anëtarët **public** të A janë anëtarë **protected** të B
  - Anëtarët **protected** të A janë anëtarë **protected** të B
  - Anëtarët **privatë** të A janë **te fshehur** ndaj anetareve të B. Ato mund te aksesohen nga B vetem nepermjet funksioneve public apo protected te A
- Nqs memberAccessSpecifier është private atëherë:
  - Anëtarët **public** të A janë anëtarë **private** të B
  - Anëtarët **protected** të A janë anëtarë **private** të B
  - Anëtarët **privatë** të A janë **te fshehur** ndaj anetareve të B. Ato mund te aksesohen nga B vetem nepermjet funksioneve public apo protected te A

```
class bClass
{
public:
    void setData(double);
    void setData(char, double);
    void print() const;
    bClass(char ch = '*', double u = 0.0);

protected:
    char bCh;

private:
    double bX;
};
```

```
void bClass::setData(double u)
{
    bX = u;
}
void bClass::setData(char ch, double u)
{
    bCh = ch;
    bX = u;
}

void bClass::print() const
{
    cout << "Base class: bCh = " << bCh << ", bX = " << bX
         << endl;
}

bClass::bClass(char ch, double u)
{
    bCh = ch;
    bX = u;
}
```

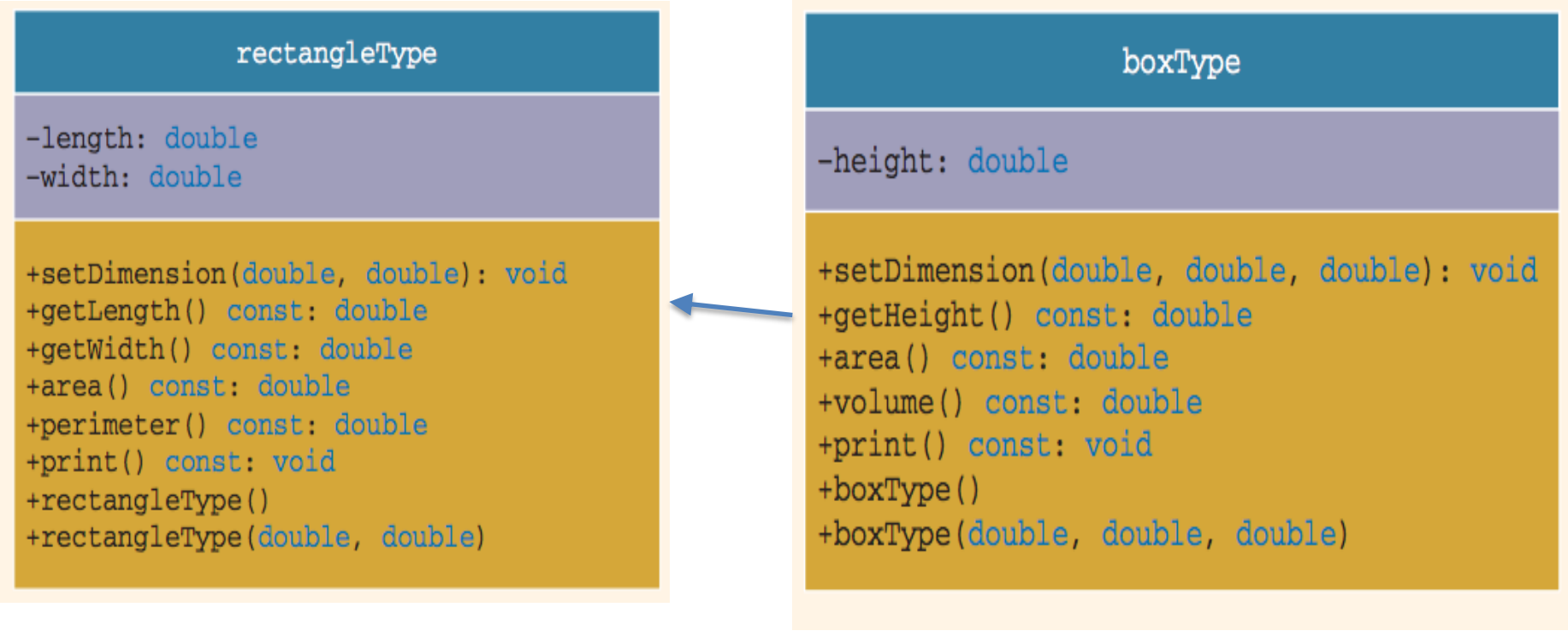
```
class dClass: public bClass
{
public:
    void setData(char, double, int);
    void print() const;

    dClass(char ch = '*', double u = 0.0, int x = 0);

private:
    int dA;
};
```

```
void dClass::setData(char ch, double v, int a)
{
    bClass::setData(v);

    bCh = ch;
    dA = a;
}
```

```
void dClass::print() const
{
    bClass::print();

    cout << "Derived class dA = " << dA << endl;
}
```

```
dClass::dClass(char ch, double u, int x)
   : bClass(ch, u)
{
    dA = x;
}
```

Shembull Trashegimie



#include rectangleType.h

```
class rectangleType
{
public:
    void setDimension(double l, double w);

    double getLength() const;

    double getWidth() const;

    double area() const;

    double perimeter() const;

    void print() const;

    rectangleType();

    rectangleType(double l, double w);

private:
    double length;
    double width;
};
```

```
void rectangleType::setDimension(double l, double w)
{
    if (l >= 0)
        length = l;
    else
        length = 0;

    if (w >= 0)
        width = w;
    else
        width = 0;
}
```

```
double rectangleType::getLength() const
{
    return length;
}

double rectangleType::getWidth() const
{
    return width;
}

double rectangleType::area() const
{
    return length * width;
}
```

```
double rectangleType::perimeter() const
{
     return 2 * (length + width);
}

void rectangleType::print() const
{
    cout << "Length = "  << length
         << "; Width = " << width;
}

rectangleType::rectangleType(double l, double w)
{
    setDimension(l, w);
}

rectangleType::rectangleType()
{
    length = 0;
    width = 0;
}
```

```
class boxType: public rectangleType
{
public:
    void setDimension(double l, double w, double h);

    double getHeight() const;

    double area() const;

    double volume() const;

    void print() const;

    boxType();

    boxType(double l, double w, double h);

 private:
     double height;
 };
```

Overriding – Riimplementimi i funksioneve të klasave bazë

- Nqs një klasë e derivuar riimplementon një funksion publik të një klase bazë, atëherë për të specifikuar thirrjen e funksionit publik të klasës bazë përdoret

klasa_bazë::funksioni_publik(liste_parametra)

- Nqs një funksion publik i klases bazë, nuk është riimplementuar, ai thirret vetem me emrin e tij dhe parametrat e nevojshem nga klasa e derivuar

```
void boxType::print() const
{
    rectangleType::print();
    cout << "; Height = " << height;
}
```

Function Overloading

- Klasa e derivuar mund të përmbajë gjithashtu funksione që kanë të njëjtin emër me funksione të klasës bazë, por me parametra të ndryshëm.
- Në këtë rast do të kishim të bënim me function overloading
- Një funksion publik i klasës bazë, që është bërë overload tek klasa e derivuar, mund të thirret nga kjo e fundit, duke përcaktuar para emrit të funksionit, emrin e klasës bazë të ndjekur nga ::

```
void boxType::setDimension(double l, double w, double h)
{
    rectangleType::setDimension(l, w);

    if (h >= 0)
        height = h;
    else
        height = 0;
}
```

Overriding – Riimplementimi i funksioneve të klasave bazë

```
double boxType::volume() const
{
    return rectangleType::area() * height;
}

double boxType::getHeight() const
{
    return height;
}

double boxType::area() const
{
    return  2 * (getLength() * getWidth()
               + getLength() * height
               + getWidth() * height);
}
```

Konstruktorët në trashëgimi

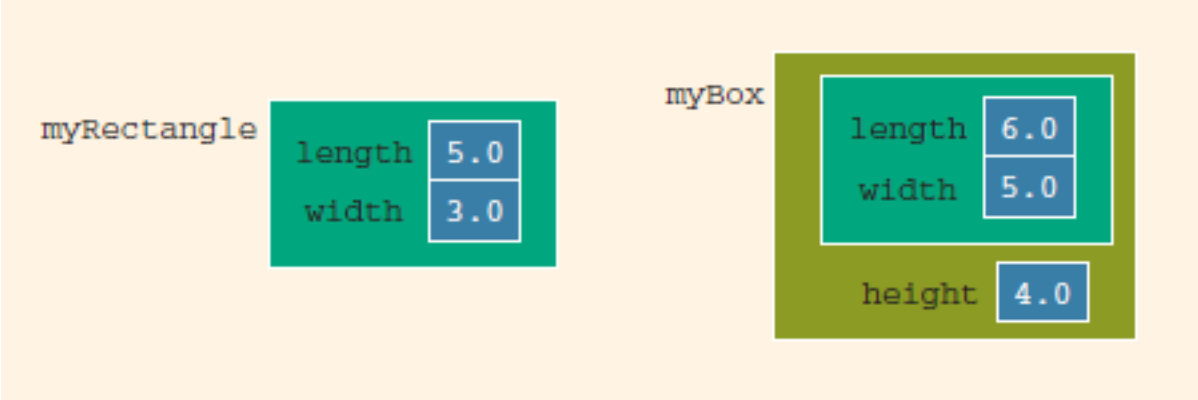
- Klasa e derivuar nuk mund të aksesojë anëtarët privatë të klasës bazë
- Megjithatë klasat e derivuara mund të kenë anëtarët e tyre privatë që duhet të inicializohen. Rrjedhimisht, kanë nevojë për konstruktorë.
- Konstruktorët e klasave të derivuara duhet të thërrasin një prej konstruktorëve të klasës bazë dhe më pas të specifikojnë më tej sjelljen e konstruktorit

```
boxType::boxType()
{
    height = 0.0;
}
```

```
boxType::boxType(double l, double w, double h)
          : rectangleType(l, w)
{
    if (h >= 0)
        height = h;
    else
        height = 0;
}
```

width dhe length janë inicializuar duke thirrur konstruktorin e klasës bazë

```
rectangleType myRectangle(5.0, 3.0);
boxType myBox(6.0, 5.0, 4.0);
```




```
myRectangle.print();
cout << endl;
myBox.print();
cout << endl;
```

OUTPUT

```
Length = 5.0; Width = 3.0

Length = 6.0; Width = 5.0; Height = 4.0
```

```
#include <iostream>
#include <iomanip>
#include "rectangleType.h"
#include "boxType.h"

using namespace std;

int main()
{
    rectangleType myRectangle1;
    rectangleType myRectangle2(8, 6);

    boxType myBox1;
    boxType myBox2(10, 7, 3);

    cout << fixed << showpoint << setprecision(2);

    cout << "Line 6: myRectangle1: ";
    myRectangle1.print();
    cout << endl;
    cout << "Line 9: Area of myRectangle1: "
         << myRectangle1.area() << endl;

    cout << "Line 10: myRectangle2: ";
    myRectangle2.print();
    cout << endl;
    cout << "Line 13: Area of myRectangle2: "
         << myRectangle2.area() << endl;
```

```
    cout << "Line 14: myBox1: ";
    myBox1.print();
    cout << endl;
    cout << "Line 17: Surface Area of myBox1: "
         << myBox1.area() << endl;
    cout << "Line 18: Volume of myBox1: "
         << myBox1.volume() << endl;

    cout << "Line 19: myBox2: ";
    myBox2.print();
    cout << endl;
    cout << "Line 22: Surface Area of myBox2: "
         << myBox2.area() << endl;
    cout << "Line 23: Volume of myBox2: "
         << myBox2.volume() << endl;

    return 0;
}
```

Ekzekutimi:

```
Line 6: myRectangle1: Length = 0.00; Width = 0.00
Line 9: Area of myRectangle1: 0.00
Line 10: myRectangle2: Length = 8.00; Width = 6.00
Line 13: Area of myRectangle2: 48.00
Line 14: myBox1: Length = 0.00; Width = 0.00; Height = 0.00
Line 17: Surface Area of myBox1: 0.00
Line 18: Volume of myBox1: 0.00
Line 19: myBox2: Length = 10.00; Width = 7.00; Height = 3.00
Line 22: Surface Area of myBox2: 242.00
Line 23: Volume of myBox2: 210.00
```

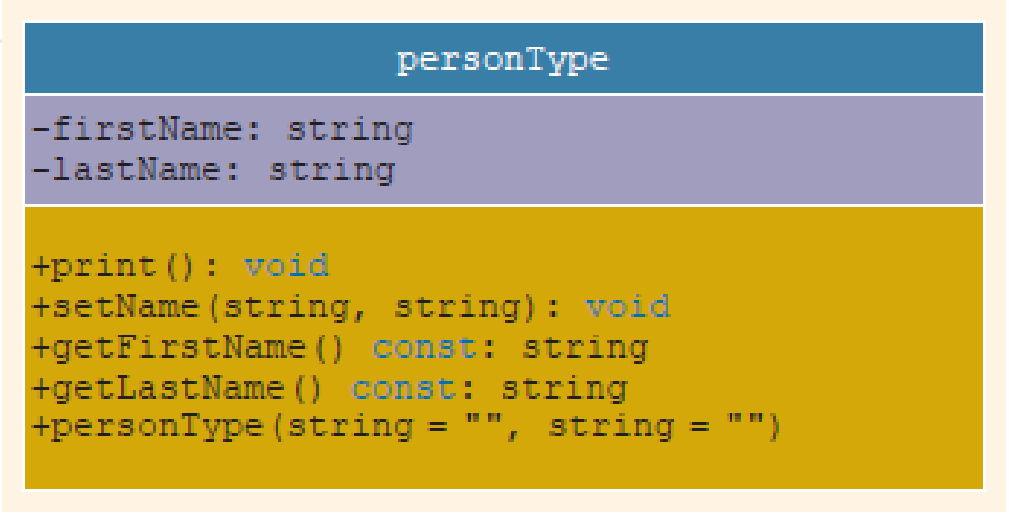
- Nje klase e derivuar gjithashtu mund te kete konstruktore me parametra default

```
class rectangleType
{
public:
   rectangleType(double l = 0, double w = 0);
      .
      .
      .
};

rectangleType::rectangleType(double l, double w)
{
    setDimension(l, w);
}
```

```
class boxType: public rectangleType
{
public:
    boxType(double l = 0, double w = 0, double h = 0);
        .
        .
        .
};

boxType::boxType(double l, double w, double h)
        : rectangleType(l, w)
{
    if (h >= 0)
        height = h;
    else
        height = 0;
}
```

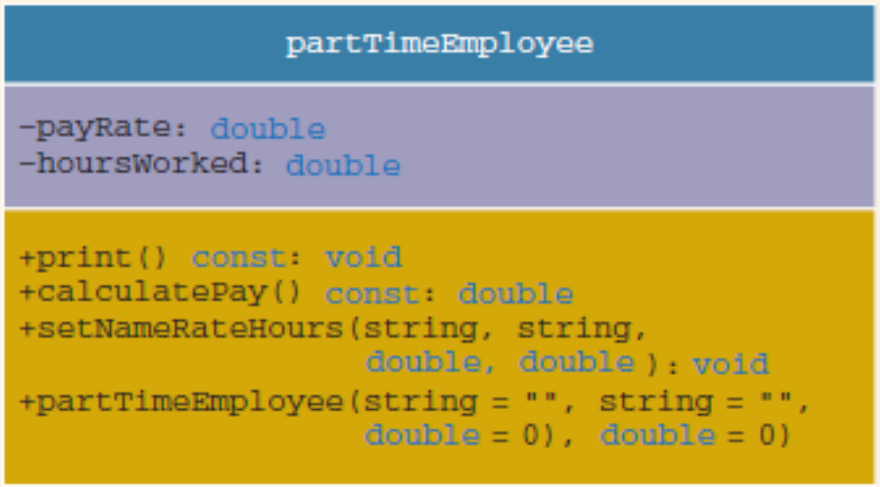
Shembull trashegimie

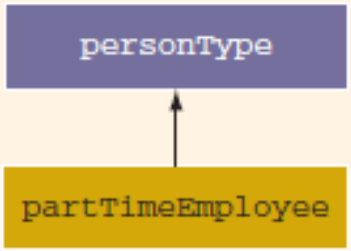
```
#include <string>

using namespace std;

class personType
{
public:
    void print() const;
    void setName(string first, string last);
    string getFirstName() const;
    string getLastName() const;
    personType(string first = "", string last = "");

private:
    string firstName;
    string lastName;
};
```



Implementimi

```
void personType::print() const
{
    cout << firstName << " " << lastName;
}

void personType::setName(string first, string last)
{
    firstName = first;
    lastName = last;
}

string personType::getFirstName() const
{
    return firstName;
}

string personType::getLastName() const
{
    return lastName;
}

    //constructor
personType::personType(string first, string last)
{
    firstName = first;
    lastName = last;
}
```






```
class partTimeEmployee: public personType
{
public:
    void print() const;

    double calculatePay() const;

    void setNameRateHours(string first, string last,
                          double rate, double hours);


private:
    double payRate;
    double hoursWorked;
};
```

```
void partTimeEmployee::print() const
{
    personType::print();
    cout << "'s wages are: $" << calculatePay() << endl;
}

double partTimeEmployee::calculatePay() const
{
    return (payRate * hoursWorked);
}

void partTimeEmployee::setNameRateHours(string first,
                         string last, double rate, double hours)
{
    personType::setName(first, last);
    payRate = rate;
    hoursWorked = hours;
}

partTimeEmployee::partTimeEmployee(string first, string last,
                                   double rate, double hours)
      : personType(first, last)
{
    if (rate >= 0)
        payRate = rate;
    else
        payRate = 0;

    if (hours >= 0)
        hoursWorked = hours;
    else
        hoursWorked = 0;
}
```

Destruktorët në trashëgimi

- Kur një klasë del jashtë shtrirjes së saj atëherë ekzekutohet automatikisht destruktori i saj
- Kur ekzekutohet destruktori i klasës së derivuar ai ekzekuton automatikisht edhe destruktorin e klasës bazë
- Destruktori i klasës bazë ekzekutohet i pari