AAB University

Faculty of Computer Sciences

Programimi i Orientuar në Objekte

Java 10:

Trashëgimia

Doc. Dr. **Mentor Hamiti**

mentor.hamiti@universitetiaab.com

- Trashëgimia
- Klasa bazë dhe e derivuar
- Kontrolli i qasjes dhe tipet e trashëgimisë
- Konstruktorët dhe trashëgimia
- Mbingarkesa
- Klasa virtuele bazë

Trashëgimia

- Një nga konceptet më të rëndësishme të POO është padyshim **trashëgimia**
  - *Trashëgimia lejon definimin e klasëve të reja të bazuara në ata paraprake*
  - *Kjo lejon mundësinë e rishfrytëzimt të kodit parakrak dhe siguron implementim të shpejtë*

Trashëgimia

- Trashëgimia paraqet mënyrë të rishfrytëzimit të softuerit në çrast krijohen klasa që shfrytëzojnë të dhëna nga klasat egzistuese dhe vetitë e tyre dhe pasurojnë ata me mundësi të reja

  - Klasa ekzistuese është e njohur si **klasa bazë**, kurse klasa e re që i referohet asaj është e njohur si **klasë e derivuar**
  - Klasa e derivuar zakonisht është më e specializuar se klasa bazë

- C++ ofron trashëgimi të tipit publik, të mbrojtur dhe private

Trashëgimia

- **Klasa bazë** tenton të jetë **më e përgjithshme** kurse **klasa e derivuar** zkonisht tenton të jetë **më e specializuar**

Base class	Derived classes
Student	GraduateStudent, UndergraduateStudent
Shape	Circle, Triangle, Rectangle, Sphere, Cube
Loan	CarLoan, HomeImprovementLoan, MortgageLoan
Employee	Faculty, Staff
Account	CheckingAccount, SavingsAccount

Trashëgimia

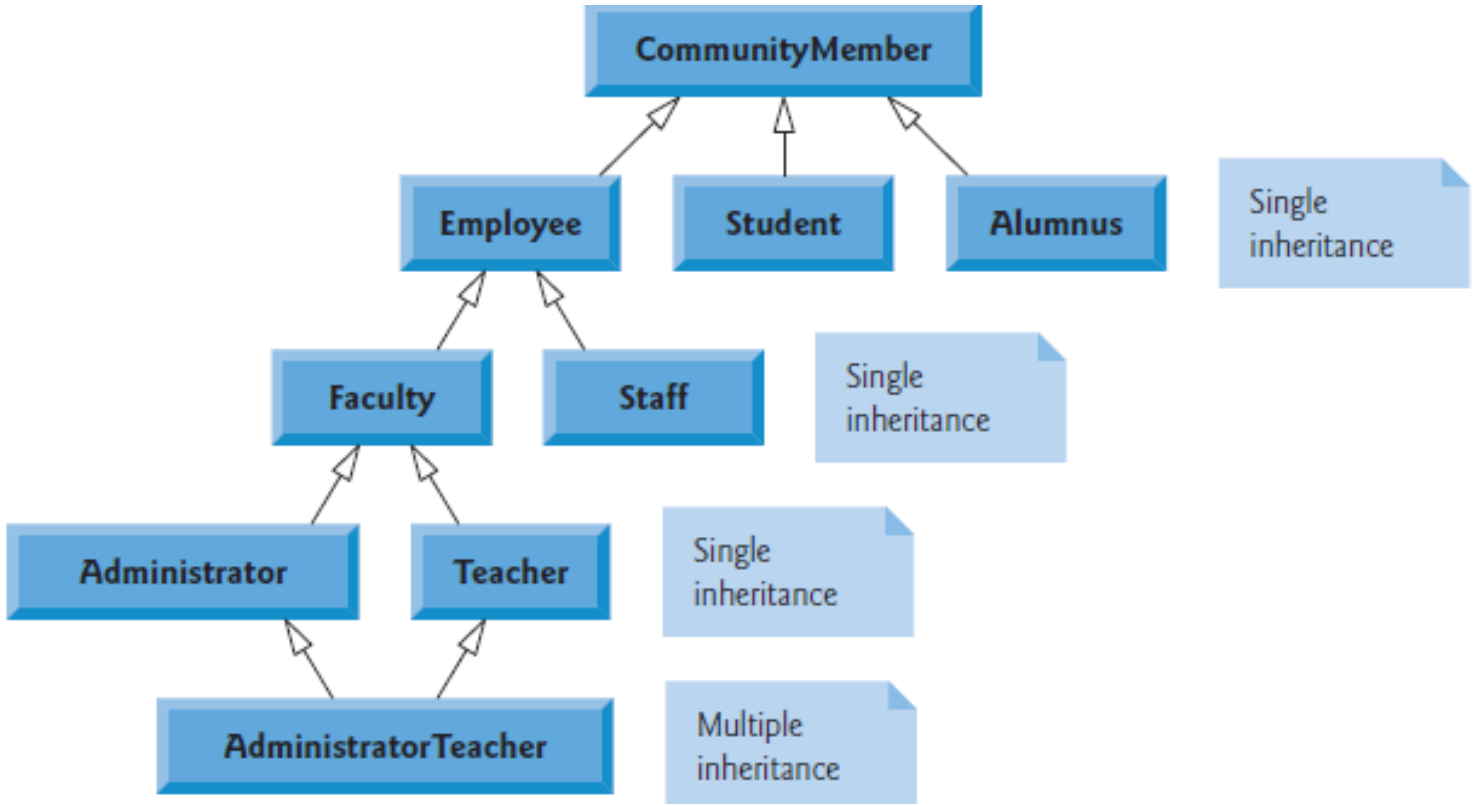
- Klasa bazë tenton të jetë më e përgjithshme kurse klasa e derivuar zkonisht tenton të jetë më e specializuar
- Shembull:

```
//The class Animal contains information and functions

class Animal
{
  public:
  Animal();
  ~Animal();
  void eat();
  void sleep();
  void drink();

private:
  int legs;
  int arms;
  int age;
};
```

```
//The class Cat is derived class

class Cat : public Animal
{
  public:
  int fur_color;
  void purr();
  void fish();
  void markTerritory();
};
```

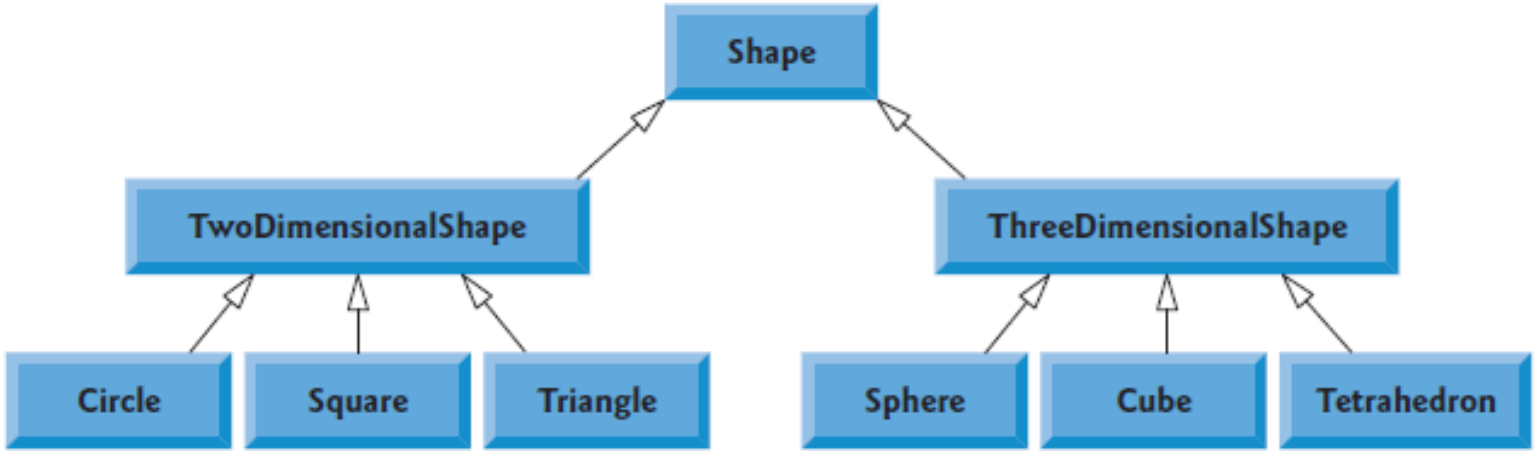
Trashëgimia

- Shembull:
  - Hierarkia e trashëgimisë



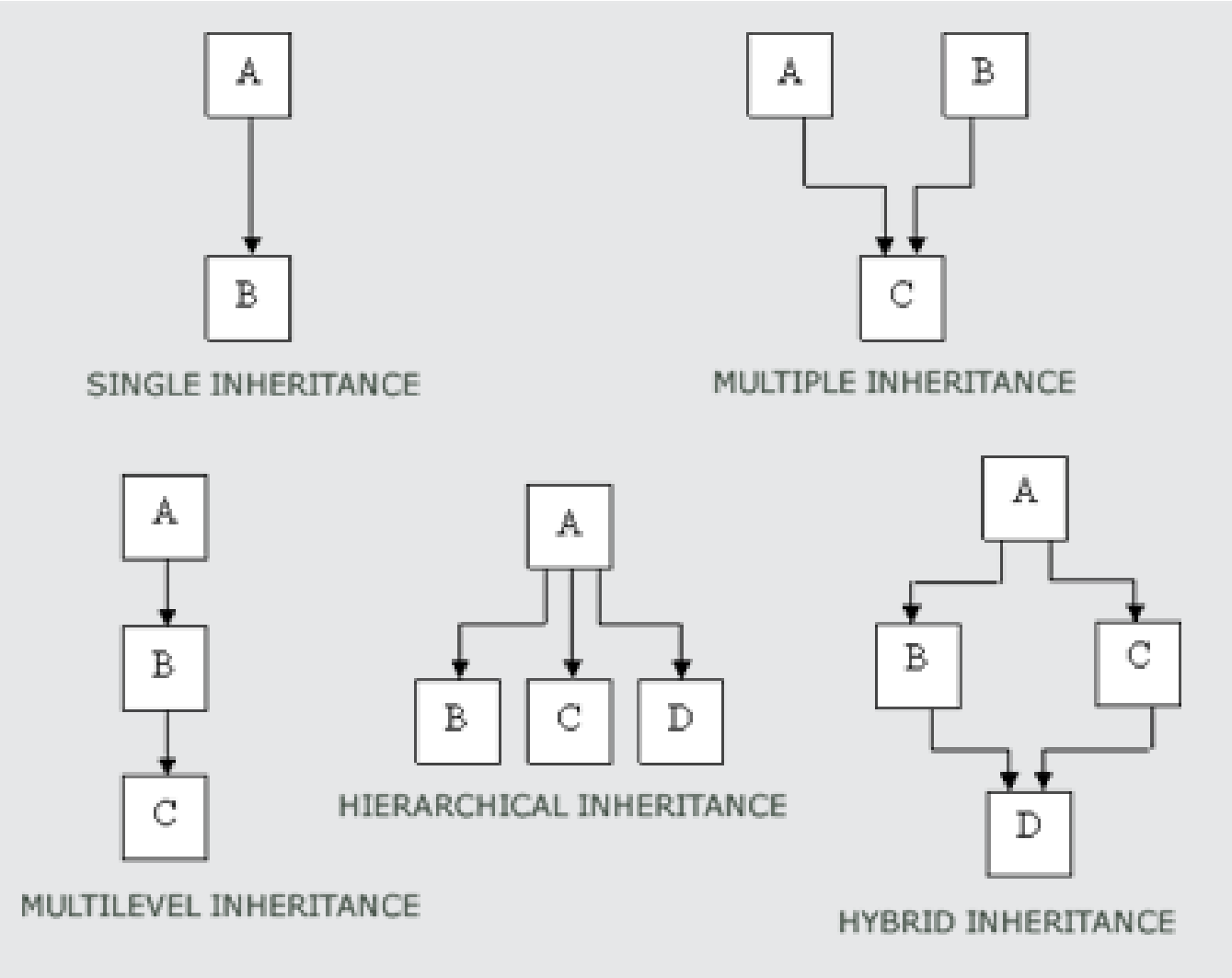
Trashëgimia

- Shembull:
  - Hierarkia e trashëgimisë



Trashëgimia

- Mënyrat e trashëgimisë



Klasa bazë

- Example 1:


```
# include <iostream>
using namespace std;

# define PI 3.14

class figura
{
private:
    float rrezja;

public:
    void perimetri (float r)
    {
        rrezja=r;
        float P;
        P = 2 * PI * rrezja;
        cout<<P;
    };
};
```

```
int main()
{
    figura rrethi;
    float r;

    cout<<"Lexo vleren e rrezes se rrethit: ";
    cin>>r;

    cout<<"\nPerimetri i llogaritur i rrethit: ";
    rrethi.perimetri(r);

    cin.get(); cin.get();
    return 0;
}
```

```
Lexo vleren e rrezes se rrethit: 10

Perimetri i llogaritur i rrethit: 62.8
```

Klasa bazë dhe e derivuar

- **Klasa** mund të derivohet nga një ose më tepër klasa
- Për të derivuar një klasë, shfrytëzohet shprehja vijuese:

class derived-class: access-specifier **base-class**

- Ku specifikuesi i qasjes “access-specifier” mund të jetë i tipit public, protected, ose private

Kontrolli i qasjes dhe trashëgimia

- **Klasa e derivuar** ka qasje në të gjitha anëtarët joprivat të klasës bazë
  - *Andaj anëtarët që nuk dëshirojmë të trashëgohen deklarohen si privat në kuadër të klasës bazë*

- Tipet e trashëgimisë:

Access	public	protected	private
Same class	yes	yes	yes
Derived classes	yes	yes	no
Outside classes	yes	no	no

Kontrolli i qasjes dhe trashëgimia

- Example 2: A base class Shape and its derived class Rectangle

```
#include <iostream>
using namespace std;

// Base class
class Shape
{
   public:
      void setWidth(int w)
      {
         width = w;
      }
      void setHeight(int h)
      {
         height = h;
      }
   protected:
      int width;
      int height;
};
```

```
// Derived class
class Rectangle: public Shape
{
   public:
      int getArea()
      {
         return (width * height);
      }
};

int main(void)
{
   Rectangle Rect;

   Rect.setWidth(5);
   Rect.setHeight(7);

   // Print the area of the object.
   cout << "Total area: " << Rect.getArea() << endl;

   cin.get(); return 0;
}
```

```
Total area: 35
```

Tipet e trashëgimisë

- Kur derivohet një klasë nga klasa bazë, deklarimi mund të bëhet publik, protected ose privat
  - *Deklarimi bëhet përmes parametrit të njohur si specifikues i qasjes "access-specifier"*

- Zakonisht më rrallë përdoret trashëgimia protected ose private, kurse më e shpeshtë është ajo e tipit public

- Konform trashëgimive të ndryshme, vlejnë rregullat vijuese:

Tipet e trashëgimisë

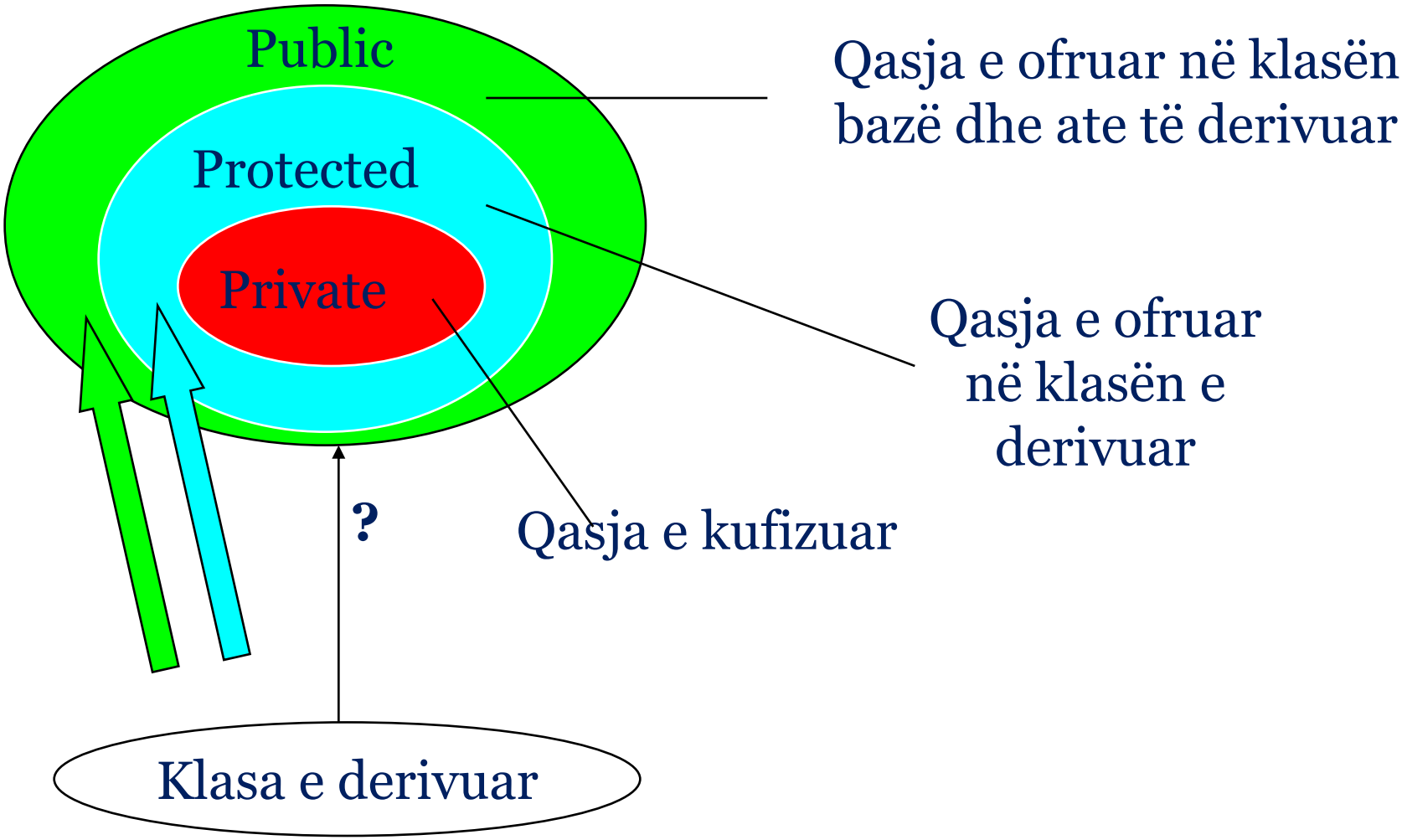
- **Trashëgimia public**: Kur klasa derivohet nga klasa bazë publike, anëtarët public të klasës bazë sërish janë public edhe në klasën e derivuar kurse anëtarët protected sërish janë protected edhe në klasën e derivuar. Vetëm në anëtarët e deklaruar private në klasën bazë asnjëherë nuk ofrohet qasje në klasën e derivuar

- **Trashëgimia protected**: Kur klasa derivohet nga klasa bazë protected, anëtarët public dhe protected të klasës bazë bëhen anëtarë protected në klasën e derivuar

- **Trashëgimia private**: Kur klasa derivohet nga klasa bazë privat, anëtarët public dhe protected të klasës bazë bëhen anëtarë private në klasën e derivuar

Tipet e trashëgimisë

- Anëtarët e klasës



Trashëgimia e shumëfishtë

- Në **C++** klasa mund të trashëgohet nga më shumë se një klasë e vetme

```
class derived-class:  access baseA, access baseB....
```

- Qasja sërish mund të jetë **public**, **protected**, ose **private** dhe caktohet për çdo klasë bazë në veçanti, të ndara me “**,**” mes veti

Trashëgimia e shumëfishtë

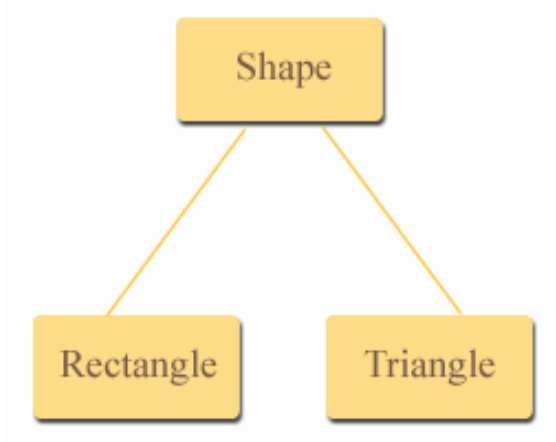
- Shembull 3:

```
#include <iostream>
using namespace std;

// Base class Shape
class Shape
{
   public:
      void setWidth(int w)
      {
         width = w;
      }
      void setHeight(int h)
      {
         height = h;
      }
   protected:
      int width;
      int height;
};

// Base class PaintCost
class PaintCost
{
   public:
      int getCost(int area)
      {
         return area * 70;
      }
};

// Derived class
class Rectangle: public Shape, public PaintCost
{
   public:
      int getArea()
      {
         return (width * height);
      }
};

int main(void)
{
   Rectangle Rect;
   int area;

   Rect.setWidth(5);
   Rect.setHeight(7);

   area = Rect.getArea();

   // Print the area of the object.
   cout << "Total area: " << Rect.getArea() << endl;

   // Print the total cost of painting
   cout << "Total paint cost: $" << Rect.getCost(area) << endl;

   cin.get(); return 0;
}
```

```
Total area: 35
Total paint cost: $2450
```

Trashëgimia

- Shembull 4:




```
# include <iostream>
using namespace std;

class Shape
{
protected:
    float width, height;
public:
    void set_data (float a, float b)
    {
        width = a;
        height = b;
    }
};
```

```
class Rectangle: public Shape
{
public:
    float area ()
    {
        return (width * height);
    }
};

class Triangle: public Shape
{
public:
    float area ()
    {
        return (width * height / 2);
    }
};

int main ()
{
    Rectangle rect;
    Triangle tri;
    rect.set_data (5,3);
    tri.set_data (2,5);
    cout << rect.area() << endl;
    cout << tri.area() << endl;
    cin.get(); return 0;
}
```

```
15
5
```

Trashëgimia

- Shembull 5:

```
class Shape
{
  protected:
       int width, height;
  public:
       void setDims (int a, int b){
       width=a; height=b;}
};
```

```
class Triangle: public Shape
{
  public:
       int area ( ) {
       return (width * height/2); }
};
```

```
class Rectangle: public Shape
{
   public:
       int area ( ) {
       return (width * height); }
};
```

```
class Square: public Rectangle
{
   public:
       void setDims (int a){
       width=a; height=a;}
};
```

Trashëgimia

- Shembull 5:

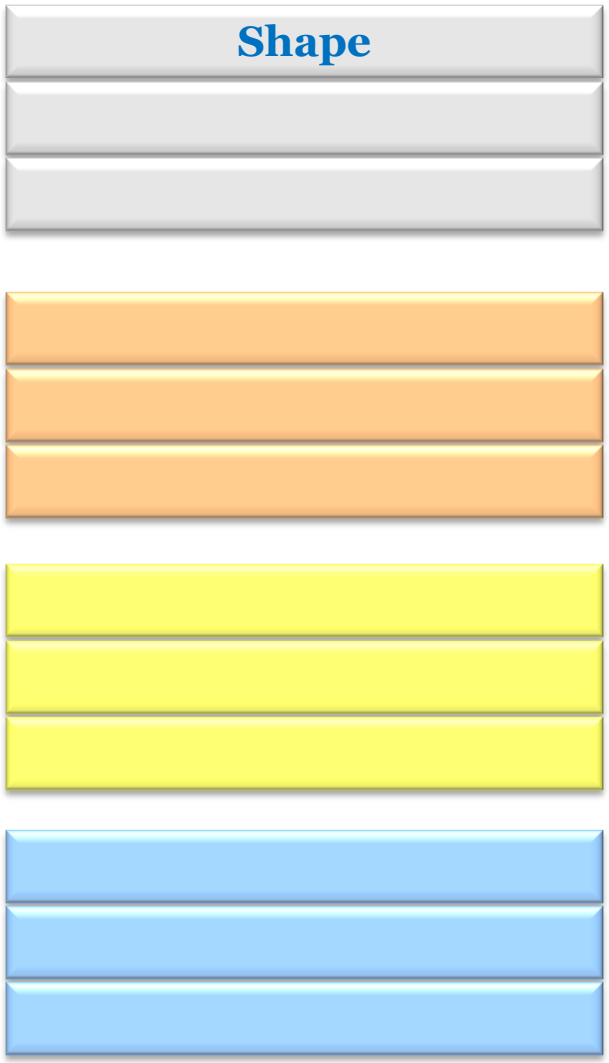
```
class Shape
{
  protected:
      int width, height;
  public:
       void setDims (int a, int b){
       width=a; height=b;}
};
```

```
class Rectangle: public Shape
{
   public:
       int area ( ) {
       return (width * height); }
};
```

```
class Triangle: public Shape
{
  public:
       int area ( ) {
       return (width * height/2); }
};
```

```
class Square: public Rectangle
{
   public:
       void setDims (int a){
       width=a; height=a;}
};
```

UML class diagram



Trashëgimia

- Shembull 6:

```
#include <iostream>
using namespace std;

class Shape
{
   protected:
         int width, height;
   public:
          void setDims (int a, int b){
          width=a; height=b;}
};

class Rectangle: public Shape  {
    public:
          int area ( ) {
          return (width * height); }
};

class Triangle: public Shape  {
   public:
         int area ( ) {
         return (width * height/2); }
};

class Square: public Rectangle  {
    public:
         void setDims (int a){
         width=a; height=a;}
};
```

```
int main(void)
{
   Rectangle Rect;

   Rect.setDims (4, 5);

  // Print the area of the object
  cout << "Total area: " << Rect.area() << endl;

   cin.get();
   return 0;
}
```

```
Total area: 20
```

Konstruktorët dhe trashëgimia

- Kompajleri në mënyrë automatike egzekuton konstruktorin e klasës bazë para se të bëhet ekzekutimi i konstruktorit të klasës së derivuar

- Në këtë rast, në mënyrë eksplicite mund të specifikohet cili konstruktor do të ekzekutohet nga kompajleri

http://www.cppforschool.com/tutorial/overriding-inheritance.html

Konstruktorët dhe trashëgimia

- Shembull 6:

```
# include <iostream>
using namespace std;

class Rectangle
{
private :
    float length;
    float width;
public:
    Rectangle ()
    {
        length = 0;
        width = 0;
    }

    Rectangle (float len, float wid)
    {
        length = len;
        width = wid;
    }

    float area()
    {
        return length * width ;
    }
};
```

```
class Box : public Rectangle
{
private :
    float height;
public:
    Box ()
    {
        height = 0;
    }

    Box (float len, float wid, float ht) : Rectangle(len, wid)
    {
        height = ht;
    }

    float volume()
    {
        return area() * height;
    }
};

int main ()
{
    Box bx;
    Box cx(4,8,5);
    cout << bx.volume() << endl;
    cout << cx.volume() << endl;
    cin.get(); return 0;
}
```

```
0
160
```


http://www.cppforschool.com/tutorial/overriding-inheritance.html

Mbingarkesa

- Funksione të ndryshme me emra të njejtë mund të egzistojnë edhe në klasën e derivuar edhe në ate bazë
  - *Në këtë rast, funksioni i klasës bazë thuhet se është I* ***mbingarkuar*** *nga klasa e derivuar*

http://www.cppforschool.com/tutorial/overriding-inheritance.html

- Shembull 7:

```
# include <iostream>
using namespace std;

class mother
{
public:
    void display ()
    {
        cout << "mother: display function\n";
    }
};

class daughter : public mother
{
public:
    void display ()
    {
        cout << "daughter: display function\n\n";
    }
};
```

```
int main ()
{
    daughter rita;
    rita.display();
    return 0;
}
```

```
daughter: display function
```

Mbingarkesa

- Përmes operatorit “scope resolution” specifikohet në mënyrë eksplicite se cili funksion do të ekzekutohet

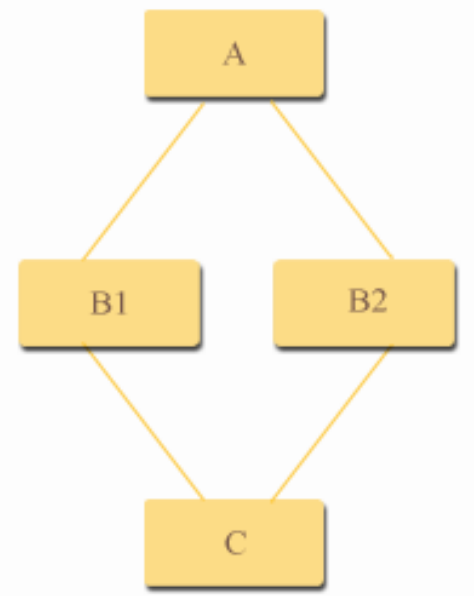
```
class daughter : public mother
{
public:
    void display ()
    {
        cout << "daughter: display function\n\n";

        mother::display();
    }
};
```

```
daughter: display function

mother: display function
```


http://www.cppforschool.com/tutorial/overriding-inheritance.html

Klasat bazë virtuele

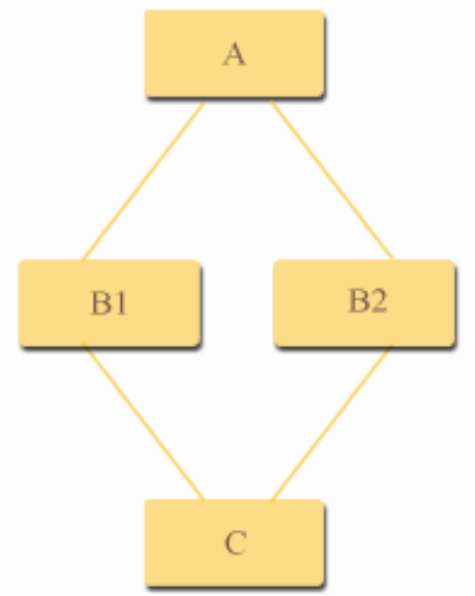
- Trashëgimia me lidhje të ndërthurura nganjëherë mund të inijon **dyfishimin** e të dhënave të trashëguara nga klasa bazë
  - *Nëse trashëgimia deklarohet si virtuale nga klasa bazë, atëherë C++ kujdeset që të egziston vetëm nga një kopje e të dhënave të trashëguara nga lasa bazë*



http://www.cppforschool.com/tutorial/overriding-inheritance.html

Klasat bazë virtuele

- Shembull:




```
class A
{
    .....
    .....
};

class B1 : virtual public A
{
    .....
    .....
};

class B2 : virtual public A
{
    .....
    .....
};

class C : public B1, public B2
{
    ..... // only one copy of A
    ..... // will be inherited
};
```

- Pyetje eventuale?!

mentor.hamiti@universitetiaab.com