Miras, Kalıtım (Inheritance)

Dr. Metin Özkan

Bilgi

- Mevcut bir sınıftan yeni bir sınıf türeterek, mevcut sınıfın üyelerinin kullanılmasına Miras(Inheritance) denir.

- Yazılımın tekrar kullanılabilmesi (Software reusability) mümkün olmaktadır.
  - *Mevcut sınıftan yeni bir sınıfın türetilmesi*
    - *Mevcut sınıfın verileri ve fonksiyonları kullanılır*
    - *Yeni yetenekler eklenir (veriler ve fonksiyonlar)*
  - *Türetilen sınıf (Derived class), taban sınıftan(base class) miras alır.*
    - *Türetilen sınıf(Derived class)*

      Daha özelleşmiş nesneler yaratır.

      Bazı fonksiyonları, taban sınıftan miras kalır.

      Bu fonksiyonlar, ihtiyaca göre özelleştirilebilir

      İlave fonksiyonlara sahip olur.

Bilgi

- Miras (Inheritance),yazılım yeniden kullanımının (software reuse) bir formudur. Öyle ki, yaratılan bir sınıf, mevcut bir sınıfın veri ve fonksiyonlarına sahip olup, yeni fonksiyon ve veriler ile geliştirilmektedir.

- Mevcut sınıf, taban sınıf (base class) olarak ve yeni sınıf türetilen sınıf(derived class) olarak adlandırılmaktadır.

- Türetilen sınıf (derived class), özelleşmiş nesneleri belirtir.

- Türetilen sınıf(derived class), taban sınıfın (base class) üyeleri yanında yeni eklenmiş üyeleri kapsar.

- Türetilen sınıf(derived class), miras yoluyla sahip olduğu üye fonksiyonları kendisine göre özelleştirebilir.

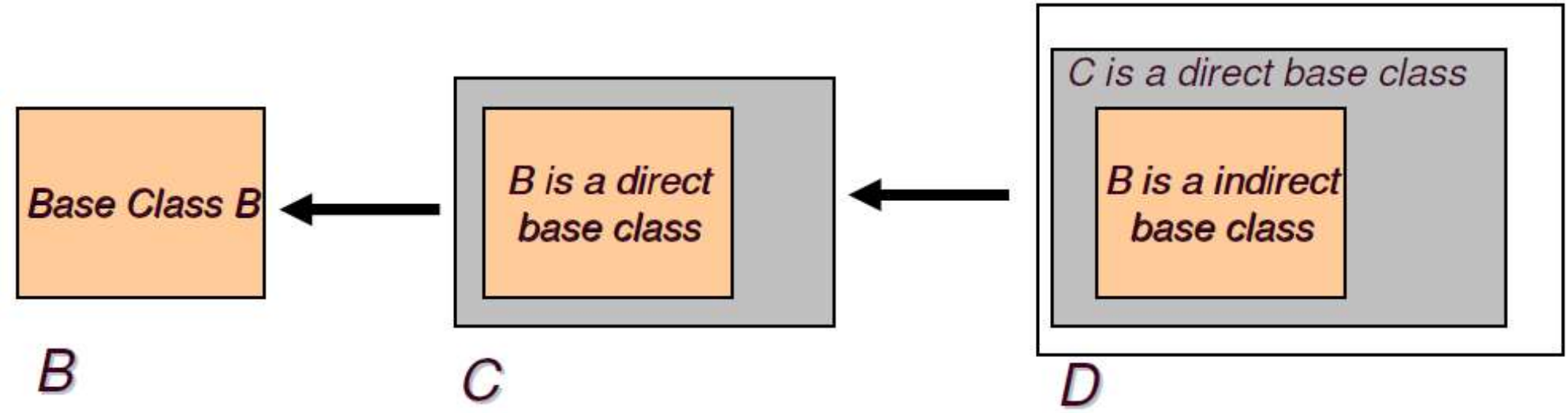
Sınıf Sıradüzeni (Class hierarchy)

- Sınıf sıradüzeni
  - *Doğrudan taban sınıfı (Direct base class)*
    - *Bari olarak miras alma vardır. (bir üst seviye sıradüzeni)*
  - *Dolaylı taban sınıfı (Indirect base class)*
    - *Miras alma iki ya da daha fazla üst seviye sıradüzeni ile gerçekleşir.*
  - *Tekil Miras (Single inheritance)*
    - *Sadece tek bir sınıftan miras alma söz konusudur.*
  - *Çoklu Miras (Multiple inheritance)*
    - *Birden çok sınıftan miras alınır.*

Doğrudan taban sınıfı (Direct base class) ile Dolaylı taban sınıfı (Indirect base class) Farkı

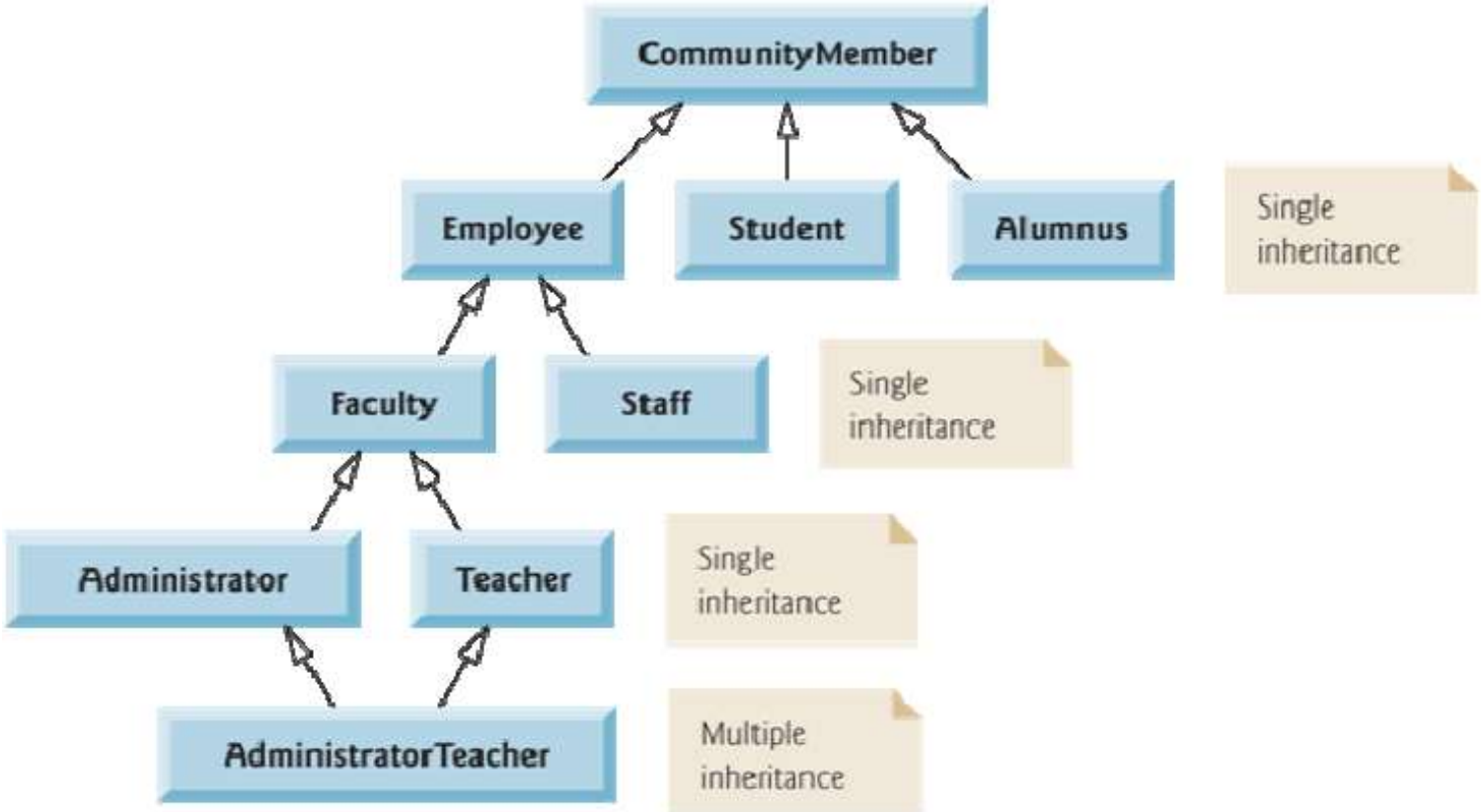
- Bir doğrudan taban sınıfı (direct base class), doğrudan miras alımı söz konusu olduğunda ifade edilir.

- Bir dolaylı taban sınıfı (indirect base class), sıradüzeninde iki ya da daha çok seviyeden miras taşındığında belirtilir.



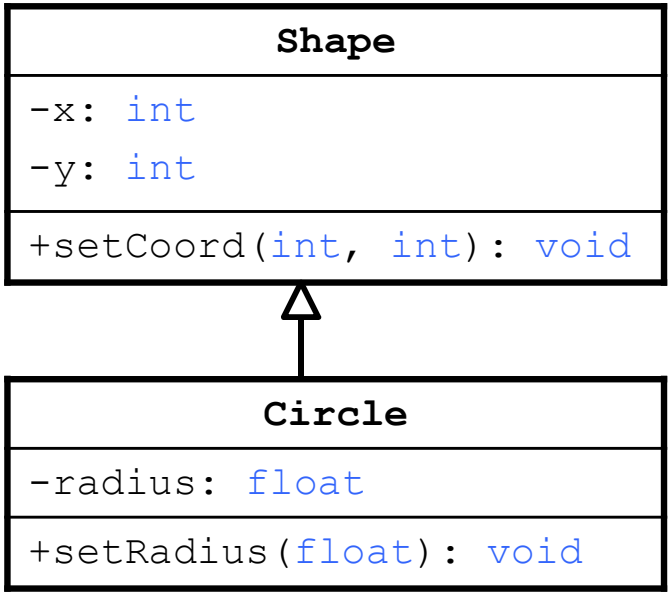
Tekil Miras (Single inheritance) ile Çoklu Miras (Multiple inheritance) Farkı

- Tekil Miras(single inheritance) durumunda, bir sınıf sadece bir sınıftan miras almaktadır.
- Çoklu Miras(Multiple inheritance) durumunda, bir sınıf birden çok sınıftan miras almaktadır.



Miras (Inheritance)

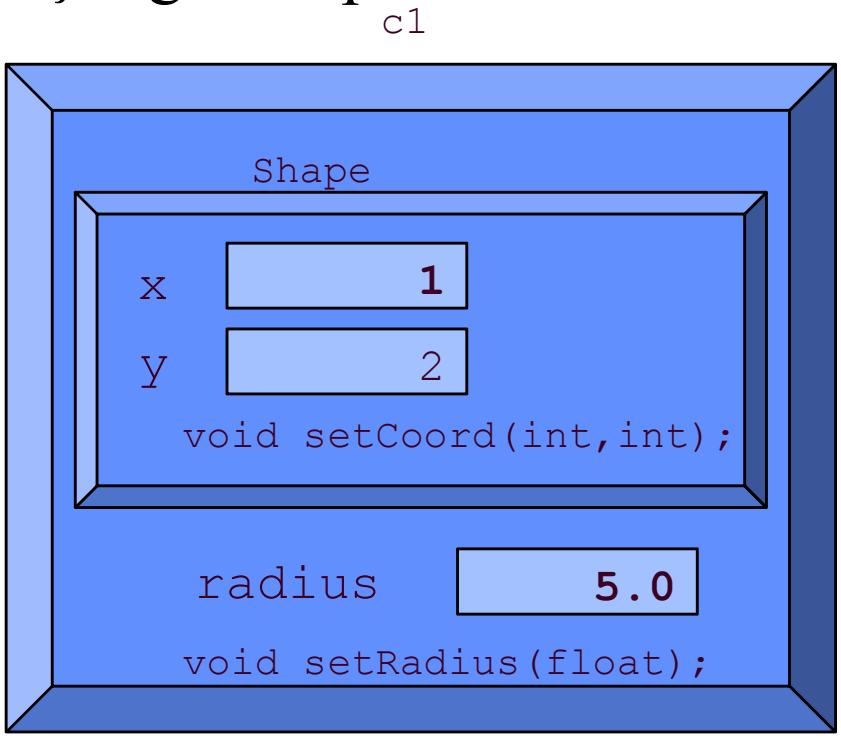
- Yanda verilen programda, Shape sınıfı taban sınıf (base class) ve Circle sınıfı ise türetilen sınıf (Derived class) olarak tanımlanmıştır.
- UML ile gösterimi şu şekildedir.




```
#include <iostream>
using namespace std;
class Shape{
            int x;
            int y;
public:
            Shape(int X=0,int Y=0)
            :x(X),y(Y){}
            void setCoord(int X,int Y){
                        x=X;
                        y=Y;
            }
};
class Circle:public Shape{
            float radius;
public:
            Circle(int X=0,int Y=0,float r=0)
            :Shape(X,Y),radius(r){}
            void setRadius(float r){
                        radius=r;
            }
};
int main(){
            Circle c1(1,2,5.0);
            c1.setCoord(3,2);
            c1.setRadius(7);
            return 0;
}
```

Miras (Inheritance)

- `Circle c1(1,2,5.0);` komutu ile yaratılan c1 nesnesi aşağıdaki gibi bir içeriğe sahiptir.




```
#include <iostream>
using namespace std;
class Shape{
            int x;
            int y;
public:
            Shape(int X=0,int Y=0)
            :x(X),y(Y){}
            void setCoord(int X,int Y){
                        x=X;
                        y=Y;
            }
};
class Circle:public Shape{
            float radius;
public:
            Circle(int X=0,int Y=0,float r=0)
            :Shape(X,Y),radius(r){}
            void setRadius(float r){
                        radius=r;
            }
};
int main(){
            Circle c1(1,2,5.0);
            c1.setCoord(3,2);
            c1.setRadius(7);
            return 0;
}
```

Miras (Inheritance)

- Üç tip miras tanımlanmaktadır: (`class Circle:public Shape{... )`
  - `public`
    - *Her türetilen sınıfa (derived class) ait nesne, aynı zamanda taban sınıfa (base class) ait bir nesnedir.*

      Taban sınıf (Base-class) nesneleri, türetilen sınıf (derived classes) nesnesi değildir.

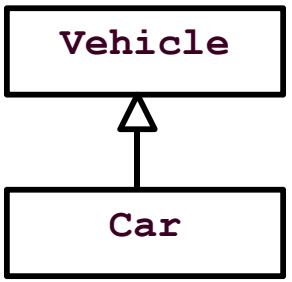
      Örneğin: Bütün otomobiller (cars), bir araçtır (vehicles), ancak bütün araçlar otomobil değildir.
    - *Türetilen sınıfta (derived class), taban sınıfın (base class) private tanımlı üyelerine doğrudan erişilemez.*

      `private` *taban sınıf üyelerine (base-class members) erişmek için, private olmayan üye fonksiyonlar kullanılmalıdır.*
  - `private`
    - *Bu, bileşim (composition) için alternatif bir yol oluşturur. (composition: an objects has another object.- "has a" relationship)*
  - `protected`
    - *Çok seyrek kullanılır.*

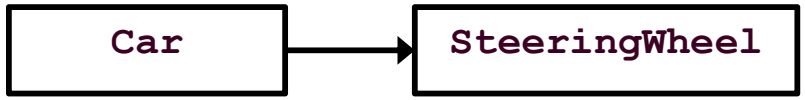
Introduction

- Soyutlama (Abstraction)
  - *Miras (inheritance) ile, nesnelerin ortak özelliklerine odaklanılır.*
- -dir “is-a” ile sahiptir “has-a” nesne ilişkisi
  - *-dir "is-a”*
    - *Miras (Inheritance)*
    - *Türetilen sınıf (Derived class) nesnesi, aynı zamanda taban sınıf (base class) nesnesi muamelesi görür.*
    - *Örneğin: Otomobil, bir araçtır. (Car is a vehicle.)*

      Araç (Vehicle) özellikleri ve davranışları, bir otomobil (car) için de geçerlidir.

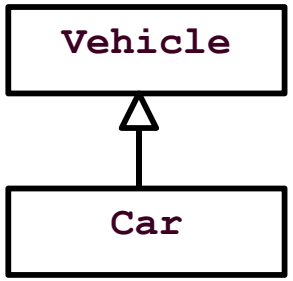


  - *sahiptir “has-a”*
    - *Bileşim (Composition)*
    - *Nesne, diğer sınıfın bir ya da daha fazla nesnesini üyesi olarak kapsar.*
    - *Örneğin: Otomobil, bir direksiyona sahiptir. (Car has a steering wheel)*



Introduction

- -dir “is-a” ile sahiptir “has-a” nesne ilişkisi
  - *-dir "is-a”*
    - *Miras (Inheritance)*



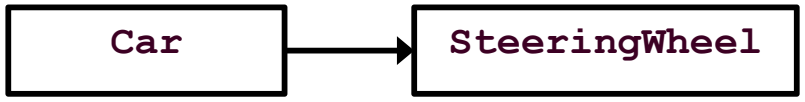
Tek bir Car nesnesi yaratılır. Car nesnesi Vehicle altındaki üyelere de sahiptir.

```
#include <iostream>
using namespace std;
class Vehicle{

};
class Car:public Vehicle{

};
int main(){
        Car car;
        return 0;
}
```

  - *sahiptir“has-a”*
    - *Bileşim (Composition)*



Bir Car nesnesi ve bir de SteeringWheel nesnesi yaratılır. SteeringWheel nesnesi, Car nesnesinin bir üyesidir.

```
#include <iostream>
using namespace std;
class SteeringWheel{

};
class Car{
     SteeringWheel wheel;
};
int main(){
        Car car;
        return 0;
}
```

Taban Sınıflar (Base Classes) ve Türetilen Sınıflar (Derived Classes)

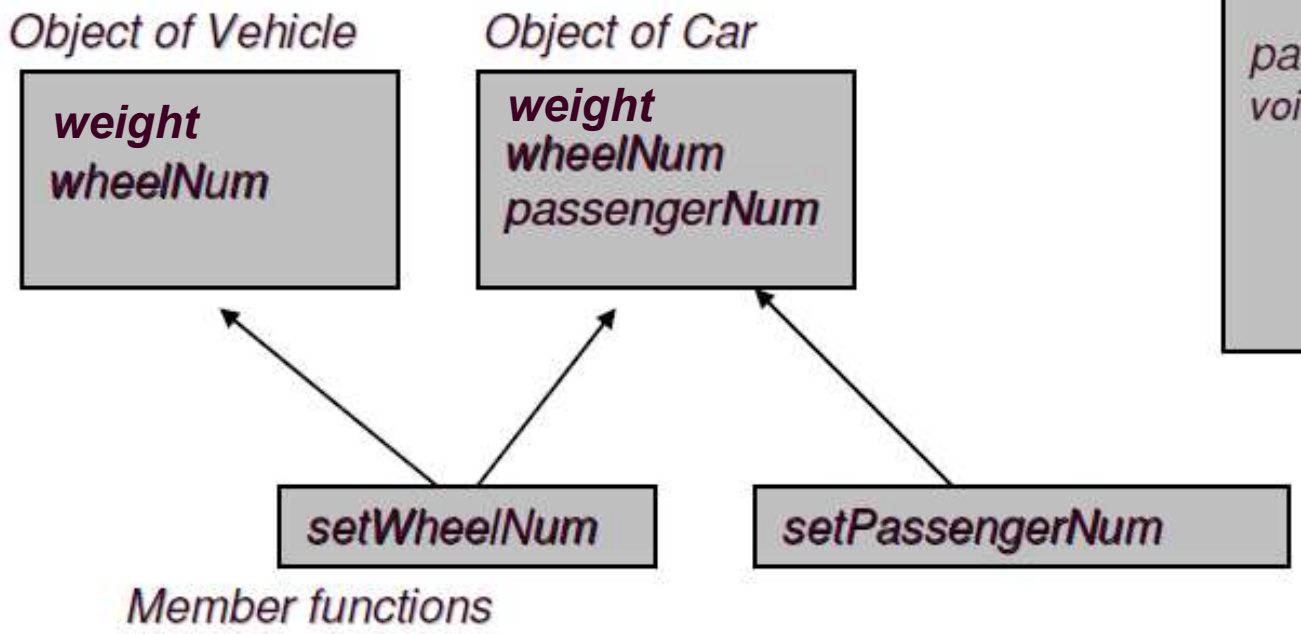
- Bir örnek
  - *Bir araç sınıfına sahibiz (Vehicle) ve bu sınıfı kullanarak otomobil sınıfı (Car) modellenebilir. Çünkü, otomobil bir araçtır (a car is a vehicle).*

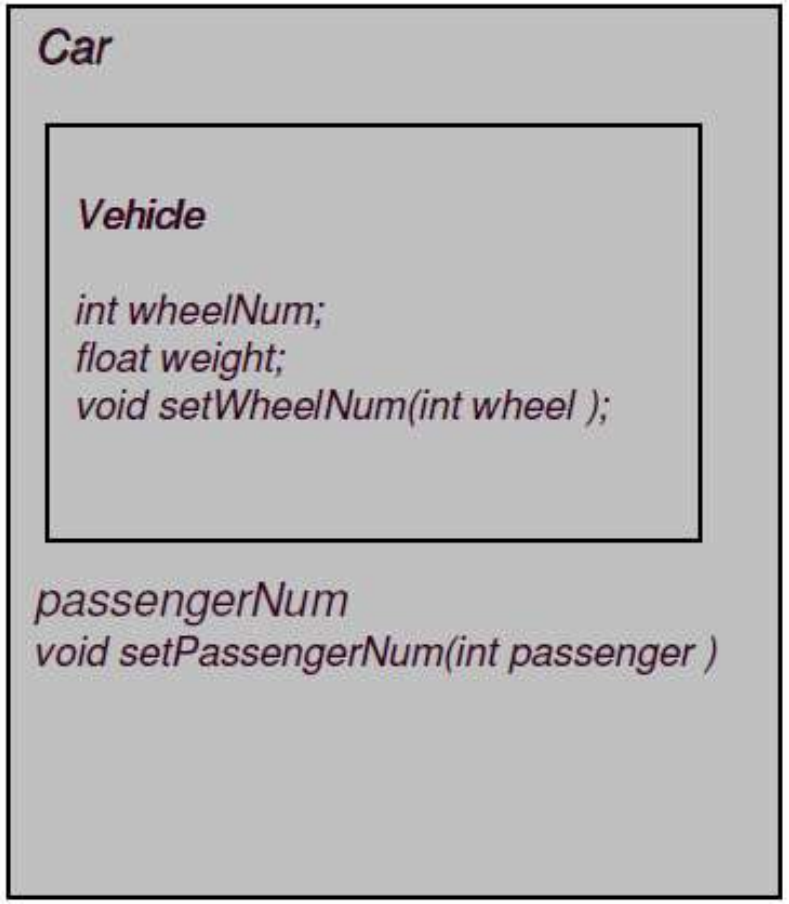
```
class Vehicle{ //base class
protected:
   int wheelNum;
   float weight;
public:
   void setWheelNum(int wheel ){wheelNum=wheel;}
};

class Car:public Vehicle{ //derived class
   int passengerNum;
public:
   void setPassengerNum(int passenger ){passengerNum=passenger;}
};
```

Taban Sınıflar (Base Classes) ve Türetilen Sınıflar (Derived Classes)

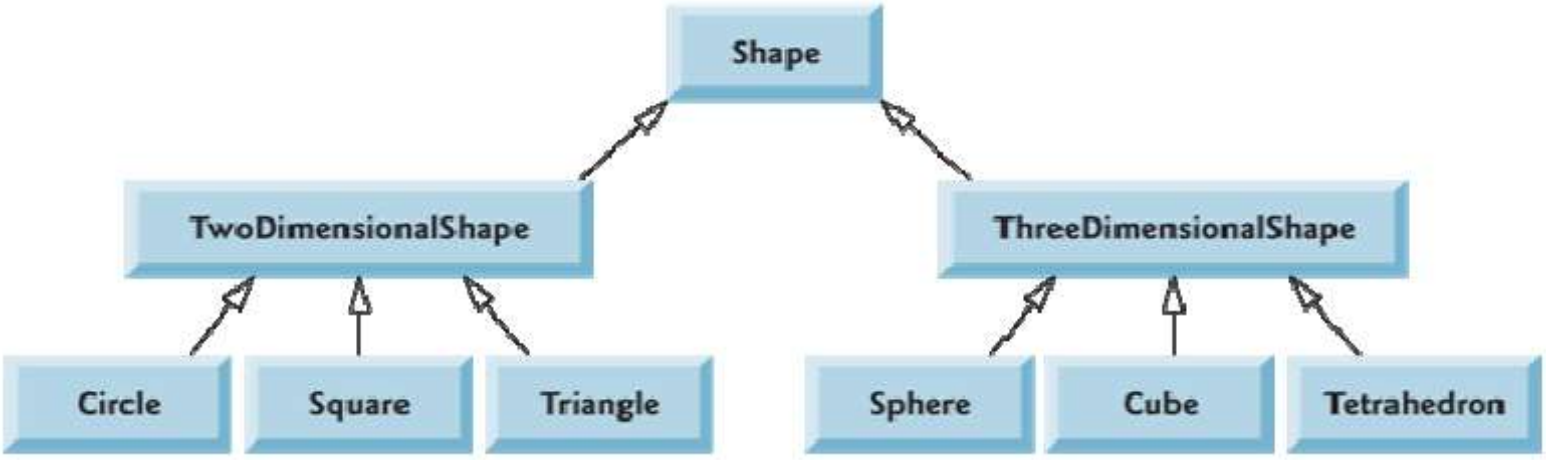
```
int main(){
        Vehicle vehicle1;
        Car car1;
        car1.setWheelNum(4);
        vehicle1.setWheelNum(0);
        car1.setPassengerNum(5);
        return 0;
}
```





Taban Sınıflar (Base Classes) ve Türetilen Sınıflar (Derived Classes)

- Bir şekil miras sıradüzenini inceleyelim. (`Shape` inheritance hierarchy)
- Şekil taban sınıfı ile başlanır (base class `Shape`).
- İki boyutlu ve üç boyutlu şekiller birer sınıf olarak Şekil sınıfından türetilir (Classes `TwoDimensionalShape` and `ThreeDimensionalShape`)
- Üçüncü seviyede, iki boyutlu ve üç boyutlu şekillerin daha özel sınıf tipleri bulunmaktadır.
- -dir ilişkisini (*is-a relationships*) belirlemek için, sıradüzende diyagramın en alttından başlayıp ok yönünde yukarı doğru ilerlenebilir.
  - *Örneğin, Daire, bir iki boyutlu şekildir ve aynı zamanda bir şekildir. (Circle is a two dimensional shape and is also a shape)*



Mirasta (Inheritance) Erişim Denetimi (Access Control)

	Accessible by own class (Vehicle)	Accessible by derived class (Car)	Accessible by objects (vehicle1 or car1)
public	yes	yes	yes
protected	yes	yes	no
private	yes	no	no

```
class Vehicle{ //base class
private:
   int wheelNum; //only base class can access
protected:
   float weight; //Base and derived class can access
public:
   void setWheelNum(int wheel ){wheelNum=wheel;} //All can access
};
class Car:public Vehicle{ //derived class
   int passengerNum;
public:
   void setWeight(float w){weight=w;} //legal
   int getWheelNum(){return wheelNum;} //Illegal
   void setPassengerNum(int passenger ){passengerNum=passenger;}
};
```

Mirasta (Inheritance) Erişim Denetimi (Access Control)

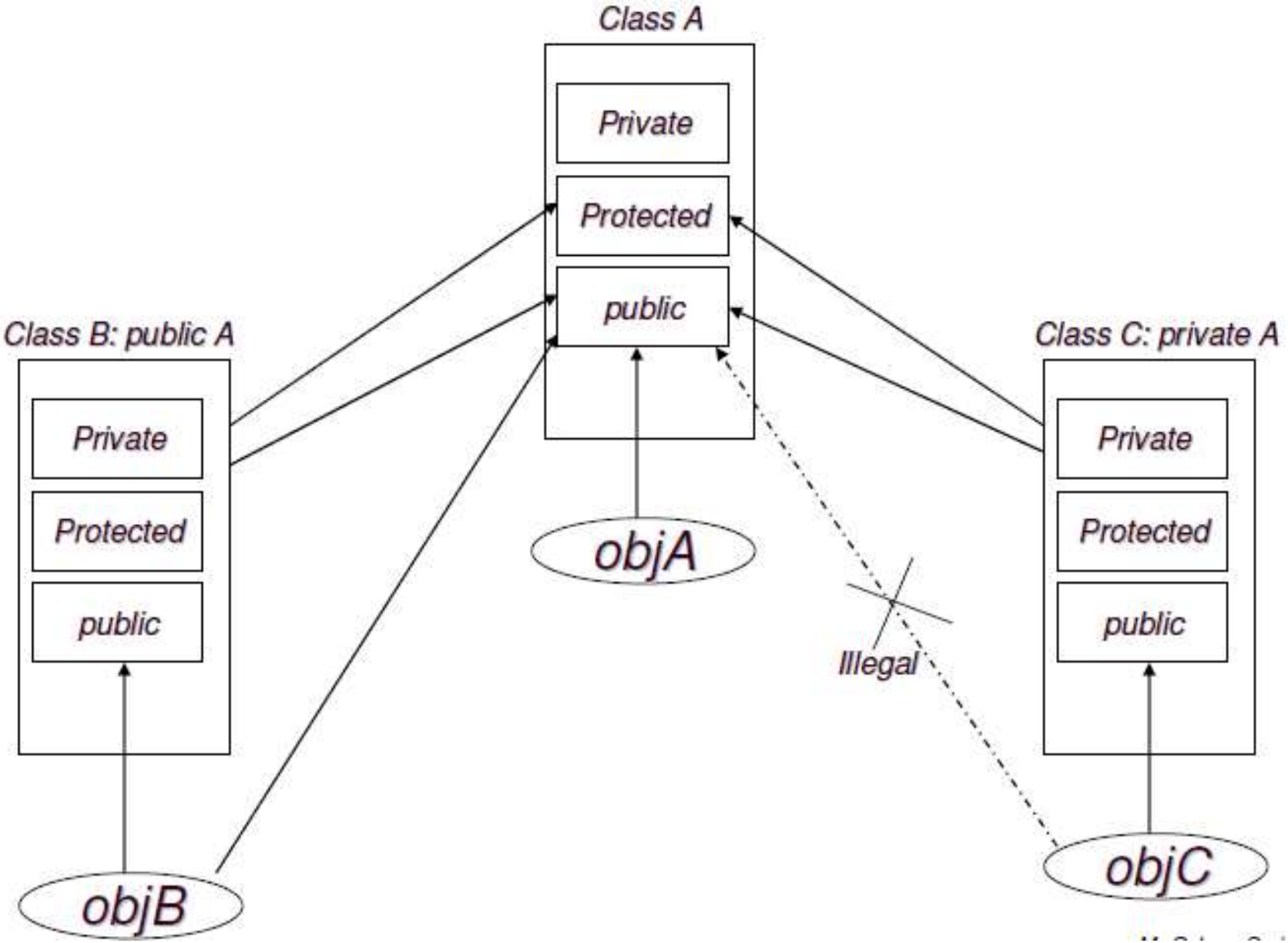
```
int main(){
        Vehicle vehicle1;
        Car car1;
        car1.setWheelNum(4);
        vehicle1.setWheelNum(0); //legal
        car1.setPassengerNum(5); //legal
        car1.wheelNum=3; //Illegal
        return 0;
}
```

Mirasta (Inheritance) Erişim Denetimi (Access Control)

- Türetilen bir sınıfta, taban sınıf üyelerine erişilebilirlik özeti

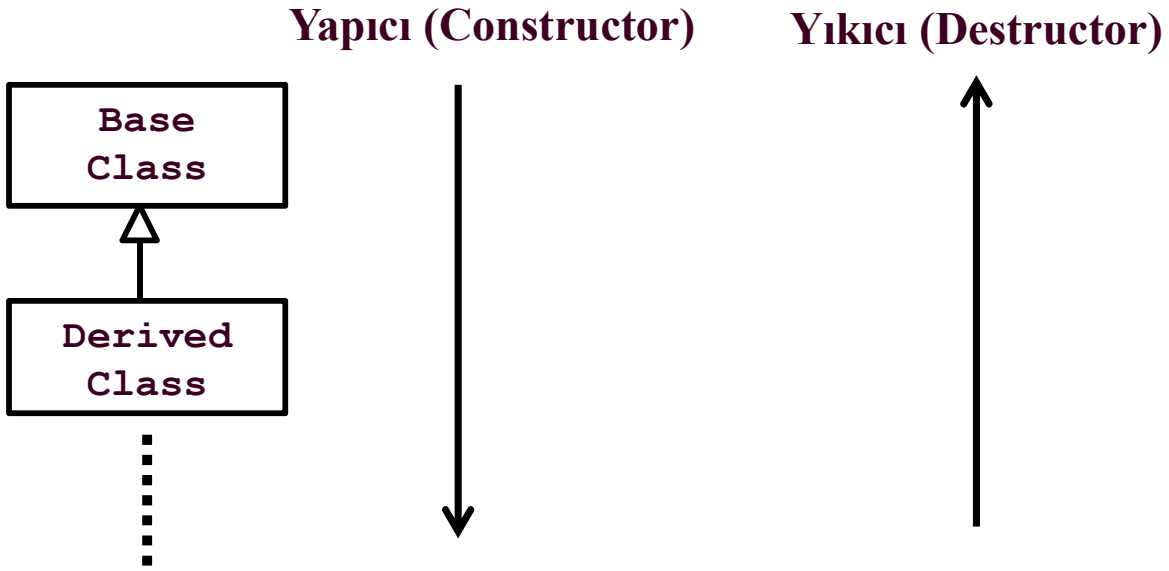
Base-class member-access specifier	Type of inheritance		
	`public` inheritance	`protected` inheritance	`private` inheritance
`public`	`public` in derived class. Can be accessed directly by member functions, `friend` functions and nonmember functions.	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`private` in derived class. Can be accessed directly by member functions and `friend` functions.
`protected`	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`protected` in derived class. Can be accessed directly by member functions and `friend` functions.	`private` in derived class. Can be accessed directly by member functions and `friend` functions.
`private`	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.	Hidden in derived class. Can be accessed by member functions and `friend` functions through `public` or `protected` member functions of the base class.

Mirasta (Inheritance) Erişim Denetimi (Access Control)



Mirasta (Inheritance) Yapıcı Fonksiyon (Constructor) ve Yıkıcı Fonksiyon (Destructor) İşleyişi

- Miras (Inheritance) yapısı içerisinde, hiyerarşide bulunan tüm sınıfların kendine ait yapıcı ve yıkıcı fonksiyonu vardır.
- Hiyerarşide bulunan herhangi bir sınıftan nesne yaratıldığında, o sınıfın ve üstünde bulunan tüm sınıflardaki yapıcı ve yıkıcı fonksiyonlar sırayla çağrılır.
- Yapıcı (Constructor) ve yıkıcı (destructor) fonksiyonların çağrılma sırası şöyledir.



Mirasta (Inheritance) Yapıcı Fonksiyon (Constructor) ve Yıkıcı Fonksiyon (Destructor) İşleyişi

```
class Vehicle{ //base class
private:
        int wheelNum; //only base class can access
        float weight; // only base class can access
public:
        Vehicle(int); // constructor
        void setWheelNum(int wheel ){wheelNum=wheel;} //All can access
        void print(){cout<<"wheelNum:"<<wheelNum<<endl;} //All can access
        void setWeight(float w){weight=w;} //All can access
};

class Car:public Vehicle{ //derived class
        int passengerNum;
public:
        Car(int); // constructor
        void setPassengerNum(int passenger ){passengerNum=passenger;}
        void print(){Vehicle::print(); //overrides print()
                cout<<"wheelNum:"<<passengerNum<<endl;}
};
```

Mirasta (Inheritance) Yapıcı Fonksiyon (Constructor) ve Yıkıcı Fonksiyon (Destructor) İşleyişi

```
Vehicle::Vehicle(int wheel)
{
        wheelNum=wheel;
}
Car::Car(int wheel, int passen):Vehicle(wheel)
{
        passengerNum=passen;
}
```

Mirasta (Inheritance) Yapıcı Fonksiyon (Constructor) ve Yıkıcı Fonksiyon (Destructor) İşleyişi

- Türetilen sınıfın bir nesnesi, kapsam dışına çıkarken (goes out of scope), yıkıcı fonksiyonlar çağrılır. İlk, türetilen sınıftaki, sonra taban sınıftaki yıkıcı fonksiyon çağrılır.

```
class Vehicle{ //base class
        . . .
        ~Vehicle(){cout<<"Destructor of Vehicle"<<endl;} //destructor
        . . .
};

class Car:public Vehicle{ //derived class
        . . .
        ~Car(){cout<<"Destructor of Car"<<endl;} //destructor
        . . .
};

int main(){
        Car car1;
}

OUTPUT:
        Destructor of Car
        Destructor of Vehicle
```

Türetilen Sınıfta (Derived Class) Taban Sınıf (Base Class) Metotlarını Geçersiz Kılma (Overriding)

- Türetilen sınıfta (Derived Class), Taban sınıfta (Base Class) bulunan bir üye fonksiyon aynı ad ile tekrar tanımlanırsa, türetilen sınıftaki fonksiyon geçerli olacaktır.

```
class Vehicle{ //base class
private:
        int wheelNum; //only base class can access
protected:
        float weight; //Base and derived class can access
public:
        void setWheelNum(int wheel ){wheelNum=wheel;} //All can access
        void print(){cout<<"wheelNum:"<<wheelNum<<endl;}
};
class Car:public Vehicle{ //derived class
        int passengerNum;
public:
        void setWeight(float w){weight=w;} //legal
        int getWheelNum(){return wheelNum;} //Illegal
        void setPassengerNum(int passenger ){passengerNum=passenger;}
        void print(){Vehicle::print(); //overrides print()
                cout<<"passengerNum:"<<passengerNum<<endl;}
}
```

Bir Örnek – Bir Firmanın Maaş Bordro Uygulaması (A Company's Payroll Application)

- Komisyonlu çalışanlara (Commission employees) (Taban sınıfı – Base Class- oluşturacak), yaptıkları satıştan komisyon ödenmektedir.

- Taban ücretli ve komisyonlu çalışanlar (Base-salaried commission employees) (Türetilmiş sınıfı – Derived Class- oluşturacak), yaptıkları satıştan komisyon artı taban ücret almaktadır.

Bir Örnek – Bir Firmanın Maaş Bordro Uygulaması (A Company's Payroll Application) : Base Class

```
// CommissionEmployee.h
// CommissionEmployee class definition represents a commission employee.
#ifndef COMMISSION_H
#define COMMISSION_H

#include <string> // C++ standard string class
using namespace std;

class CommissionEmployee
{
public:
   CommissionEmployee( const string &, const string &, const string &,
   double = 0.0, double = 0.0 );

   void setFirstName( const string & ); // set first name
   string getFirstName() const; // return first name

   void setLastName( const string & ); // set last name
   string getLastName() const; // return last name

   void setSocialSecurityNumber( const string & ); // set SSN
   string getSocialSecurityNumber() const; // return SSN
```

Bir Örnek – Bir Firmanın Maaş Bordro Uygulaması (A Company's Payroll Application) : Base Class

```
        void setGrossSales( double ); // set gross sales amount
        double getGrossSales() const; // return gross sales amount

        void setCommissionRate( double ); // set commission rate (percentage)
        double getCommissionRate() const; // return commission rate

        double earnings() const; // calculate earnings
        void print() const; // print CommissionEmployee object
private:
        string firstName;
        string lastName;
        string socialSecurityNumber;
        double grossSales; // gross weekly sales
        double commissionRate; // commission percentage
}; // end class CommissionEmployee

#endif
```

Bir Örnek – Bir Firmanın Maaş Bordro Uygulaması (A Company's Payroll Application) : Base Class

```
// CommissionEmployee.cpp
// Class CommissionEmployee member-function definitions.
#include <iostream>
#include "CommissionEmployee.h" // CommissionEmployee class definition
using namespace std;

// constructor
CommissionEmployee::CommissionEmployee(
const string &first, const string &last, const string &ssn,
double sales, double rate )
{
        firstName = first; // should validate
        lastName = last; // should validate
        socialSecurityNumber = ssn; // should validate
        setGrossSales( sales ); // validate and store gross sales
        setCommissionRate( rate ); // validate and store commission rate
 } // end CommissionEmployee constructor

// set first name
void CommissionEmployee::setFirstName( const string &first )
{
         firstName = first; // should validate
} // end function setFirstName
```

Bir Örnek – Bir Firmanın Maaş Bordro Uygulaması (A Company's Payroll Application) : Base Class

```
// return first name
string CommissionEmployee::getFirstName() const
{
         return firstName;
} // end function getFirstName

// set last name
void CommissionEmployee::setLastName( const string &last )
{
         lastName = last; // should validate
 } // end function setLastName

// return last name
string CommissionEmployee::getLastName() const
{
         return lastName;
} // end function getLastName

// set social security number
void CommissionEmployee::setSocialSecurityNumber( const string &ssn )
{
         socialSecurityNumber = ssn; // should validate
} // end function setSocialSecurityNumber
```

Bir Örnek – Bir Firmanın Maaş Bordro Uygulaması (A Company's Payroll Application) : Base Class

```
// return social security number
string CommissionEmployee::getSocialSecurityNumber() const
{
         return socialSecurityNumber;
} // end function getSocialSecurityNumber

// set gross sales amount
void CommissionEmployee::setGrossSales( double sales )
{
         grossSales = ( sales < 0.0 ) ? 0.0 : sales;
} // end function setGrossSales

// return gross sales amount
double CommissionEmployee::getGrossSales() const
{
         return grossSales;
} // end function getGrossSales

// set commission rate
void CommissionEmployee::setCommissionRate( double rate )
{
         commissionRate = ( rate > 0.0 && rate < 1.0 ) ? rate : 0.0;
} // end function setCommissionRate
```

Bir Örnek – Bir Firmanın Maaş Bordro Uygulaması (A Company's Payroll Application) : Base Class

```
// return commission rate
double CommissionEmployee::getCommissionRate() const
{
        return commissionRate;
} // end function getCommissionRate

// calculate earnings
double CommissionEmployee::earnings() const
{
        return commissionRate * grossSales;
} // end function earnings

// print CommissionEmployee object
void CommissionEmployee::print() const
{
        cout << "commission employee: " << firstName << ' ' << lastName
                << "\nsocial security number: " << socialSecurityNumber
                << "\ngross sales: " << grossSales
                << "\ncommission rate: " << commissionRate;
} // end function print
```

Bir Örnek – Bir Firmanın Maaş Bordro Uygulaması (A Company's Payroll Application) : Application

```
// main.cpp
// Testing class CommissionEmployee.
#include <iostream>
#include <iomanip>
#include "CommissionEmployee.h" // CommissionEmployee class definition
using namespace std;

int main()
{
   // instantiate a CommissionEmployee object
   CommissionEmployee employee("Sue", "Jones", "222-22-2222", 10000, .06 );

   // set floating-point output formatting
   cout << fixed << setprecision( 2 );

   // get commission employee data
   cout << "Employee information obtained by get functions: \n"
        << "\nFirst name is " << employee.getFirstName()
        << "\nLast name is " << employee.getLastName()
        << "\nSocial security number is "
        << employee.getSocialSecurityNumber()
        << "\nGross sales is " << employee.getGrossSales()
        << "\nCommission rate is " << employee.getCommissionRate()<<endl;
```

Bir Örnek – Bir Firmanın Maaş Bordro Uygulaması (A Company's Payroll Application) : Application

```
         employee.setGrossSales( 8000 ); // set gross sales
         employee.setCommissionRate( .1 ); // set commission rate

         cout<<"\nUpdated employee information output by print function: \n"<<endl;
         employee.print(); // display the new employee information

         // display the employee's earnings
         cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;
} // end main
```

```
Employee information obtained by get functions:

First name is Sue
Last name is Jones
Social security number is 222-22-2222
Gross sales is 10000.00
Commission rate is 0.06

Updated employee information output by print function:

commission employee: Sue Jones
social security number: 222-22-2222
gross sales: 8000.00
commission rate: 0.10

Employee's earnings: $800.00
```

Bir Örnek – Bir Firmanın Maaş Bordro Uygulaması (A Company's Payroll Application) : Derived Class

```
// BasePlusCommissionEmployee.h
// BasePlusCommissionEmployee class derived from class
// CommissionEmployee.
#ifndef BASEPLUS_H
#define BASEPLUS_H
#include <string> // C++ standard string class
#include "CommissionEmployee.h" // CommissionEmployee class declaration
using namespace std;

class BasePlusCommissionEmployee : public CommissionEmployee
{
public:
        BasePlusCommissionEmployee( const string &, const string &,
                const string &, double = 0.0, double = 0.0, double = 0.0 );
        void setBaseSalary( double ); // set base salary
        double getBaseSalary() const; // return base salary

        double earnings() const; // calculate earnings
        void print() const; // print BasePlusCommissionEmployee object
private:
        double baseSalary; // base salary
}; // end class BasePlusCommissionEmployee

#endif
```

Bir Örnek – Bir Firmanın Maaş Bordro Uygulaması (A Company's Payroll Application) : Derived Class

```
// BasePlusCommissionEmployee.cpp
// Class BasePlusCommissionEmployee member-function definitions.
#include <iostream>
#include "BasePlusCommissionEmployee.h" // class definition
using namespace std;

// constructor
BasePlusCommissionEmployee::BasePlusCommissionEmployee(
         const string &first, const string &last, const string &ssn,
         double sales, double rate, double salary )
         // explicitly call base-class constructor
         : CommissionEmployee( first, last, ssn, sales, rate )
{
         setBaseSalary( salary ); // validate and store base salary
} // end BasePlusCommissionEmployee constructor

// set base salary
void BasePlusCommissionEmployee::setBaseSalary( double salary )
{
         baseSalary = ( salary < 0.0 ) ? 0.0 : salary;
} // end function setBaseSalary
```

Bir Örnek – Bir Firmanın Maaş Bordro Uygulaması (A Company's Payroll Application) : Derived Class

```
// return base salary
double BasePlusCommissionEmployee::getBaseSalary() const
{
         return baseSalary;
} // end function getBaseSalary

// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
         // derived class cannot access the base class's private data
         return baseSalary + ( commissionRate * grossSales );
} // end function earnings

// print BasePlusCommissionEmployee object
void BasePlusCommissionEmployee::print() const
{
         // derived class cannot access the base class's private data
         cout << "base-salaried commission employee: " << firstName << ' '
                  <<lastName<<"\nsocial security number: " << socialSecurityNumber
                  << "\ngross sales: " << grossSales
                  << "\ncommission rate: " << commissionRate
                  << "\nbase salary: " << baseSalary;
} // end function print
```

Bir Örnek – Bir Firmanın Maaş Bordro Uygulaması (A Company's Payroll Application) : Compiler Error

```
C:\cpphtp7_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(33) :
   error C2248: 'CommissionEmployee::commissionRate' :
   cannot access private member declared in class 'CommissionEmployee'

C:\cpphtp7_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(33) :
   error C2248: 'CommissionEmployee::grossSales' :
   cannot access private member declared in class 'CommissionEmployee'

C:\cpphtp7_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(40) :
   error C2248: 'CommissionEmployee::firstName' :
   cannot access private member declared in class 'CommissionEmployee'
```

```
C:\cpphtp7_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(41) :
   error C2248: 'CommissionEmployee::lastName' :
   cannot access private member declared in class 'CommissionEmployee'

C:\cpphtp7_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(41) :
   error C2248: 'CommissionEmployee::socialSecurityNumber' :
   cannot access private member declared in class 'CommissionEmployee'

C:\cpphtp7_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(42) :
   error C2248: 'CommissionEmployee::grossSales' :
   cannot access private member declared in class 'CommissionEmployee'

C:\cpphtp7_examples\ch12\Fig12_10_11\BasePlusCommissionEmployee.cpp(43) :
   error C2248: 'CommissionEmployee::commissionRate' :
   cannot access private member declared in class 'CommissionEmployee'
```

Bir Örnek – Bir Firmanın Maaş Bordro Uygulaması (A Company's Payroll Application) : Derived Class

- CommissionEmployee, sınıfındaki private tanımlı üyeler protected olarak tanımlanabilir.

Bir Örnek – Bir Firmanın Maaş Bordro Uygulaması (A Company's Payroll Application) : Application

```
// main.cpp
// Testing class BasePlusCommissionEmployee.
#include <iostream>
#include <iomanip>
#include "BasePlusCommissionEmployee.h" // class definition
using namespace std;

int main()
{
   // instantiate BasePlusCommissionEmployee object
   BasePlusCommissionEmployee
   employee( "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );

   // set floating-point output formatting
   cout << fixed << setprecision( 2 );

   // get commission employee data
   cout << "Employee information obtained by get functions: \n"
        << "\nFirst name is " << employee.getFirstName()
        << "\nLast name is " << employee.getLastName()
        << "\nSocial security number is "
        << employee.getSocialSecurityNumber()
        << "\nGross sales is " << employee.getGrossSales()
        << "\nCommission rate is " << employee.getCommissionRate()
        << "\nBase salary is " << employee.getBaseSalary() << endl;
```

Example (Derived Class)

```
    employee.setBaseSalary( 1000 ); // set base salary

    cout << "\nUpdated employee information output by print function: \n"<<endl;
    employee.print(); // display the new employee information

    // display the employee's earnings
    cout << "\n\nEmployee's earnings: $" << employee.earnings() << endl;
} // end main
```

```
Employee information obtained by get functions:

First name is Bob
Last name is Lewis
Social security number is 333-33-3333
Gross sales is 5000.00
Commission rate is 0.04
Base salary is 300.00

Updated employee information output by print function:

base-salaried commission employee: Bob Lewis
social security number: 333-33-3333
gross sales: 5000.00
commission rate: 0.04
base salary: 1000.00

Employee's earnings: $1200.00
```

Bir Örnek – Bir Firmanın Maaş Bordro Uygulaması (A Company's Payroll Application)

- Taban sınıfta, üye verileri `protected` olarak tanımlamak performansı biraz arttırabilir. Çünkü, üye verilere doğrudan ulaşılmakta, set ve get üye fonksiyonlar kullanılmamaktadır.
- Birçok durumda, veri üyeleri `private` olarak tanımlamak yazılım mühendisliği için önemlidir. Böylece, kod optimizasyonu derleyiciye bırakılabilir.

Miras (Inheritance) ile Yazılım Mühendisliği

- Mevcut bir sınıftan, miras (inheritance) kullanarak yeni bir sınıf yaratınca, yeni sınıf mevcut sınıfın üye fonksiyonları ve verilerini miras alır.

- İlave üyelerle ve taban sınıfın (base class) üyelerini yeniden tanımlayarak, yeni sınıfı ihtiyaçları karşılamak üzere özelleştirebiliriz.

- Türetilen sınıfın (derived-class) programcısı, taban sınıfın (base class) kaynak kodlarına erişmeksizin bunları yapabilir.

- Türetilen sınıf (derived class), taban sınıfın (base class) object koduna bağlantı (link) oluşturur.

Miras (Inheritance) ile Yazılım Mühendisliği

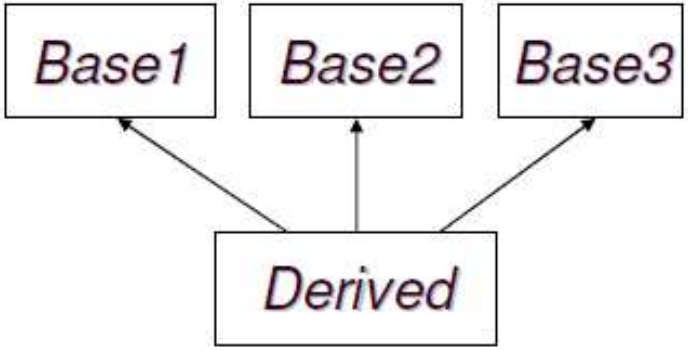
- Bağımsız yazılım tedarikçileri (Independent Software Vendor, ISV), satış ya da lisans için tescilli sınıflar (classes) geliştirebilir ve bu sınıfları kullanıcılara object-kod formatında sunabilir.

- Kullanıcılar, bu kütüphane sınıflarından hızlıca ve bağımsız yazılım tedarikçilerinin tescilli kaynak kodlarına erişmeksizin yeni sınıflar türetebilir.

- Bütün bağımsız yazılım tedarikçileri, başlık dosyaları ile birlikte object kodları vermesine ihtiyaç vardır.

Çoklu Miras (Multiple Inheritance)

- Bir sınıfın, birden fazla sınıftan türetilmesine çoklu miras (multiple inheritance) denir.



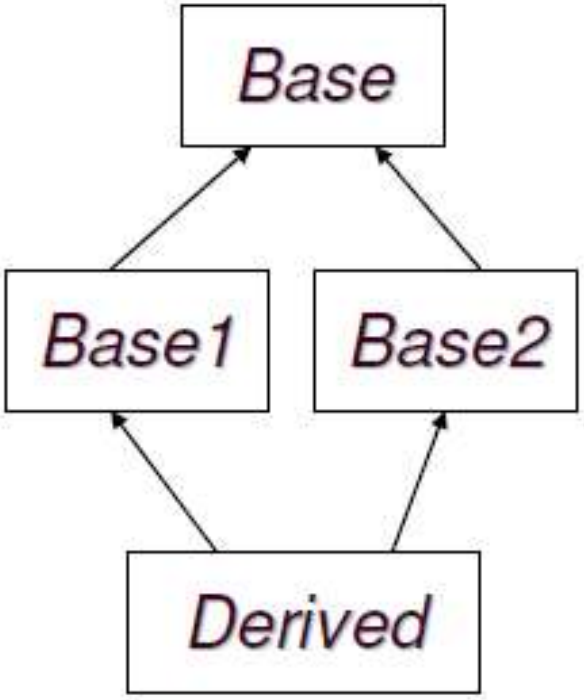

```
class Base1{
public:
        void f1(){}
};
class Base2{
public:
        void f2() {}
};
class Base3{
public:
        void f3() {}
};
class Derived:public Base1, public Base2, public Base3{
public:
        void f2() {}
        void f3(int) {}
};
```

Çoklu Miras (Multiple Inheritance)

```
int main(){
        Derived a;
        a.f1(); //Base1::f1
        a.f2(); //overrides Base2
        a.f3(); //Error: Functions are not overloaded in inheritance.
        a.Base3::f3(); //Base3::f3
}
```

Çoklu Miras (Multiple Inheritance) (belirsizlik)

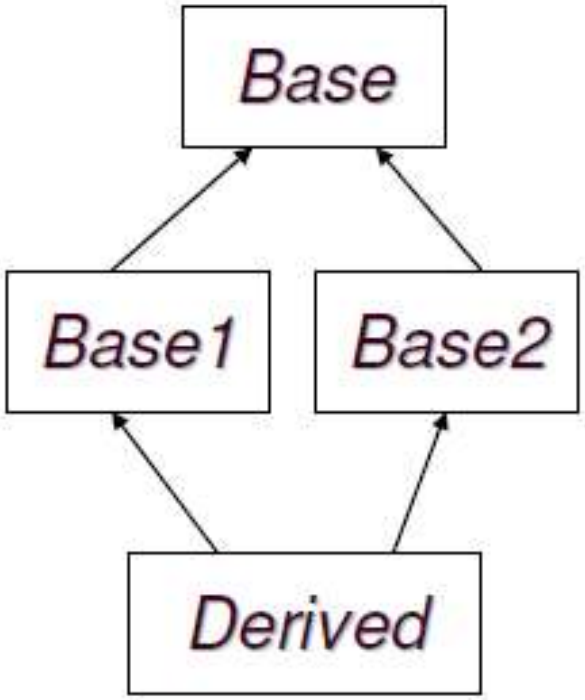
- Türetilen sınıf (Derived class), taban sınıfın (Base class) üyelerini iki kere içermektedir. Taban sınıftaki üyelere erişimde, belirsizlik oluşmaktadır.




```
class Base {
protected:
        int a;
};
class Base1:public Base{
        ...
};
class Base2:public Base{
        ...
};
class Derived:public Base1, public Base2{
public:
        void f(){a++;} //Error: ambiguity
};
```

Çoklu Miras (Multiple Inheritance) (belirsizlik)-> Çözüm

- virtual anahtar kelimesi kullanılır. Bu anahtar kelime derleyiciye sadece bir üyeyi miras almasını söyler.
- Bunula birlikte, çoklu mirastan mümkün olduğunca kaçınmak gereklidir.




```
class Base {
protected:
        int a;
};
class Base1:virtual public Base{
        ...
};
class Base2:virtual public Base{
        ...
};
class Derived:public Base1, public Base2{
public:
        void f(){a++;}
};
```

Kaynaklar

- T.C. Lethbridge and R. Laganiere, Object-Oriented Software Engineering - Practical software development using UML and Java, McGraw Hill, Second Edition, 2005.
- H.M.Deitel and P.J.Deitel, C++ How To Program, 9E, Pearson Press, 2014.
- B. Stroustrup, The C++ Programming Language, 3rd Edition, Special Edition, Addison Wesley, 2000.
- Dr. Feza Buzluca, Ders Notları.
- Ç. Turhan ve F.C. Serçe, C++ Dersi: Nesne Tabanlı Programlama, 2nci Baskı, 2014.