C++

Ders 6

Çoklu Benzeşim

Prof. Dr. M. Serdar ÇELEBİ, İTÜ

C++

Çoklubenzeşim

- Farklı sınıflardaki nesnelerin aynı fonksiyon çağrılmasına farklı şekilde cevap verebilmesini sağlar. Diğer bir deyişle benzer durumlara genel bir arabirimin uygulanma işlemidir. Böylece, “bir arabirim çok metod” felsefesini tamamlar.
- Sanal fonksiyonlar aracılığıyla uygulanır.
  - Kök-sınıftan bir pointer (veya referans) bir sanal fonksiyon çağırır
  - C++ nesnedeki doğru tekrar tanımlanmış fonksiyonu seçer
- Eğer birden fazla sınıfta sanal olmayan üye fonksiyon tanımlanmış ve kök-sınıf pointer aracılığıyla çağrılmışsa, o zaman kök-sınıf fonksiyonu çağrılır.
  - Eğer türetilmiş sınıf pointer’ ı tarafından çağrılmışsa türetilmiş sınıftaki fonksiyon kullanılır
- print fonksiyonunun sanal fonksiyon olmadığını düşünün

```
Employee e, *ePtr = &e;
HourlyWorker h, *hPtr = &h;
ePtr->print();    // base-sınıf print fonksiyonu
hPtr->print();    // derived-sınıf print fonksiyonu
ePtr=&h;          // dönüşüm mümkündür
ePtr->print();    // ama hala base-sınıf print fonksiyonunu çağırır
```

Employee Kök sınıfını,

HourlyWorker Türetilmiş-sınıfı göstermektedir. Kök-sınıfındaki print() türetilmiş sınıflarada açıktır, fakat bunu açık bir şekilde yapmak gerekir, söyleki;

hPtr -> Employee :: print ();

C++

Örnek (Çoklubenzeşim)

```
// Fig. 10.1: employ2.h
// Abstract base class Employee
#ifndef EMPLOY2_H
#define EMPLOY2_H

class Employee {
public:
   Employee( const char *, const char * )
   ~Employee();   // destructor reclaims
   const char *getFirstName() const;
   const char *getLastName() const;

   // Pure virtual function makes Employe
   virtual double earnings() const = 0;   // pure virtual
```

earnings saf(pure) ve **virtual** belirtiliyor, çünkü uygulama **earnings** fonksiyonun hangi türetilmiş sınıfta kullanılacağına bağlı

Employee abstract base sınıf

C++

Örnek (devamı)

```
   virtual void print() const;            // virtual
private:
   char *firstName;
   char *lastName;
};

#endif
// Fig. 10.1: employ2.cpp
// Member function definitions for
// abstract base class Employee.
// Note: No definitions given for pure virtual functions.
#include <iostream>

using std::cout;

#include <cstring>
#include <cassert>
#include "employ2.h"
```

C++

Örnek (devamı)

```

// Constructor dynamically allocates space for the
// first and last name and uses strcpy to copy
// the first and last names into the object.
Employee::Employee( const char *first, const char *last )
{
   firstName = new char[ strlen( first ) + 1 ];
   assert( firstName != 0 );    // test that new worked
   strcpy( firstName, first );

   lastName = new char[ strlen( last ) + 1 ];
   assert( lastName != 0 );     // test that new worked
   strcpy( lastName, last );
}

// Destructor deallocates dynamically allocated memory
Employee::~Employee()
{
   delete [] firstName;
   delete [] lastName;
}
```

C++

Örnek (devamı)

```cpp

// Return a pointer to the first name
// Const return type prevents caller from modifying private
// data. Caller should copy returned string before destructor
// deletes dynamic storage to prevent undefined pointer.
const char *Employee::getFirstName() const
{
   return firstName;   // caller must delete memory
}

// Return a pointer to the last name
// Const return type prevents caller from modifying private
// data. Caller should copy returned string before destructor
// deletes dynamic storage to prevent undefined pointer.
```

C++

Örnek (devamı)

```
const char *Employee::getLastName() const
{
   return lastName;   // caller must delete memory
}

// Print the name of the Employee
void Employee::print() const
   { cout << firstName << ' ' << lastName; }
// Fig. 10.1: boss1.h
// Boss class derived from Employee
#ifndef BOSS1_H
#define BOSS1_H
#include "employ2.h"

```

C++

Örnek (devamı)

```cpp
class Boss : public Employee {
public:
   Boss( const char *, const char *, double = 0.0 );
   void setWeeklySalary( double );
   virtual double earnings() const;
   virtual void print() const;
private:
   double weeklySalary;
};

#endif
```

C++

Örnek (devamı)

```
// Fig. 10.1: boss1.cpp
// Member function definitions for class Boss
#include <iostream>

using std::cout;

#include "boss1.h"

// Constructor function for class Boss
Boss::Boss( const char *first, const char *last, double s )
   : Employee( first, last )  // call base-class constructor
{ setWeeklySalary( s ); }

// Set the Boss's salary
void Boss::setWeeklySalary( double s )
   { weeklySalary = s > 0 ? s : 0; }

```

C++

Örnek (devamı)

```
// Get the Boss's pay
double Boss::earnings() const { return weeklySalary; }

// Print the Boss's name
void Boss::print() const
{
   cout << "\n             Boss: ";
   Employee::print();
}
// Fig. 10.1: commis1.h
// CommissionWorker class derived from Employee
#ifndef COMMIS1_H
#define COMMIS1_H
#include "employ2.h"

class CommissionWorker : public Employee {
```

Tekrar yazılmış (overriden) **earnings** ve **print** fonksiyonları kök sınıfta **virtual** olarak bildirildiler.

C++

Örnek (devamı)

```
public:
   CommissionWorker( const char *, const char *,
                     double = 0.0, double = 0.0,
                     int = 0 );
   void setSalary( double );
   void setCommission( double );
   void setQuantity( int );
   virtual double earnings() const;
   virtual void print() const;
private:
   double salary;       // base salary per week
   double commission;   // amount per item sold
   int quantity;        // total items sold for week
};

#endif
```

C++

Örnek (devamı)

```
// Fig. 10.1: commis1.cpp
// Member function definitions for class CommissionWorker
#include <iostream>

using std::cout;

#include "commis1.h"

// Constructor for class CommissionWorker
CommissionWorker::CommissionWorker( const char *first,
        const char *last, double s, double c, int q )
   : Employee( first, last )  // call base-class constructor
{
   setSalary( s );
```

C++

Örnek (devamı)

```
   setCommission( c );
   setQuantity( q );
}

// Set CommissionWorker's weekly base salary
void CommissionWorker::setSalary( double s )
   { salary = s > 0 ? s : 0; }

// Set CommissionWorker's commission
void CommissionWorker::setCommission( double c )
   { commission = c > 0 ? c : 0; }

// Set CommissionWorker's quantity sold
void CommissionWorker::setQuantity( int q )
```

C++

Örnek (devamı)

```
   { quantity = q > 0 ? q : 0; }

// Determine CommissionWorker's earnings
double CommissionWorker::earnings() const
   { return salary + commission * quantity; }

// Print the CommissionWorker's name
void CommissionWorker::print() const
{
   cout << "\nCommission worker: ";
   Employee::print();
}
```

Tekrar yazılmış **earnings** ve **print** fonksiyonları kök sınıfta **virtual** olarak bildirildiler.

C++

Örnek (devamı)

```
// Fig. 10.1: piece1.h
// PieceWorker class derived from Employee
#ifndef PIECE1_H
#define PIECE1_H
#include "employ2.h"

class PieceWorker : public Employee {
public:
   PieceWorker( const char *, const char *,
                double = 0.0, int = 0);
   void setWage( double );
   void setQuantity( int );
   virtual double earnings() const;
```

C++

Örnek (devamı)

```
   virtual void print() const;
private:
   double wagePerPiece; // wage for each piece output
   int quantity;        // output for week
};

#endif
// Fig. 10.1: piece1.cpp
// Member function definitions for class PieceWorker
#include <iostream>

using std::cout;

#include "piece1.h"

// Constructor for class PieceWorker
PieceWorker::PieceWorker( const char *first, const char *last,
                          double w, int q )
```

C++

Örnek (devamı)

```
   : Employee( first, last )  // call base-class constructor
{
   setWage( w );
   setQuantity( q );
}

// Set the wage
void PieceWorker::setWage( double w )
   { wagePerPiece = w > 0 ? w : 0; }

// Set the number of items output
void PieceWorker::setQuantity( int q )
   { quantity = q > 0 ? q : 0; }

// Determine the PieceWorker's earnings
double PieceWorker::earnings() const
   { return quantity * wagePerPiece; }

// Print the PieceWorker's name
void PieceWorker::print() const
{
   cout << "\n     Piece worker: ";
   Employee::print();
}
```

Bir kez daha tekrar yazılmış **earnings** ve **print** fonksiyonları kök sınıfta **virtual** olarak bildirildiler.

C++

Örnek (devamı)

```
// Fig. 10.1: hourly1.h
// Definition of class HourlyWorker
#ifndef HOURLY1_H
#define HOURLY1_H
#include "employ2.h"

class HourlyWorker : public Employee {
public:
   HourlyWorker( const char *, const char *,
                 double = 0.0, double = 0.0);
   void setWage( double );
   void setHours( double );
   virtual double earnings() const;
```

C++

Örnek (devamı)

```
   virtual void print() const;
private:
   double wage;    // wage per hour
   double hours;   // hours worked for week
};

#endif
// Fig. 10.1: hourly1.cpp
// Member function definitions for class HourlyWorker
#include <iostream>

using std::cout;

#include "hourly1.h"
```

C++

Örnek (devamı)

```

// Constructor for class HourlyWorker
HourlyWorker::HourlyWorker( const char *first,
                            const char *last,
                            double w, double h )
   : Employee( first, last )   // call base-class constructor
{
   setWage( w );
   setHours( h );
}

// Set the wage
void HourlyWorker::setWage( double w )
   { wage = w > 0 ? w : 0; }
```

C++

Örnek (devamı)

```cpp

// Set the hours worked
void HourlyWorker::setHours( double h )
   { hours = h >= 0 && h < 168 ? h : 0; }

// Get the HourlyWorker's pay
double HourlyWorker::earnings() const
{
   if ( hours <= 40 ) // no overtime
      return wage * hours;
   else                  // overtime is paid at wage * 1.5
      return 40 * wage + ( hours - 40 ) * wage * 1.5;
}

// Print the HourlyWorker's name
```

Overridden fonksiyonlar

C++

Örnek (devamı)

```
void HourlyWorker::print() const
{
   cout << "\n    Hourly worker: ";
   Employee::print();
}
// Fig. 10.1: fig10_01.cpp
// Driver for Employee hierarchy
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::ios;
```

C++

Örnek (devamı)

```
using std::setiosflags;
using std::setprecision;

#include "employ2.h"
#include "boss1.h"
#include "commis1.h"
#include "piece1.h"
#include "hourly1.h"

void virtualViaPointer( const Employee * );
void virtualViaReference( const Employee & );

int main()
{
   // set output formatting
   cout << setiosflags( ios::fixed | ios::showpoint )
        << setprecision( 2 );

```

C++

Örnek (devamı)

```
   Boss b( "John", "Smith", 800.00 );
   b.print();
   cout << " earned $" << b.earnings();
   virtualViaPointer( &b );           // use
   virtualViaReference( b );          // use

   CommissionWorker c( "Sue", "Jones", 200
   c.print();
   cout << " earned $" << c.earnings();
   virtualViaPointer( &c );           // us
   virtualViaReference( c );          // us

   PieceWorker p( "Bob", "Lewis", 2.5, 20
   p.print();
   cout << " earned $" << p.earnings();
   virtualViaPointer( &p );           // u
   virtualViaReference( p );          // u

   HourlyWorker h( "Karen", "Price", 13.7  40 );
                    ned $" << h.ea
```

```
Boss: John Smith earned $800.00
Boss: John Smith earned $800.00
Boss: John Smith earned $800.00
```

Kendi nesnesini kullanarak print fonksiyonu çağırmak.

Bir base-sınıfı pointeri kullanarak print fonksiyonu çağırmak.

Bu virtual fonksiyonları kullanır ve dinamik ömürlüdür.

Bir base-sınıf referansı kullanarak print fonksiyonu çağırmak.

Bu virtual fonksiyonu kullanılır ve dinamik ömürlüdür.

```
Piece worker: Bob Lewis earned $500.00
Piece worker: Bob Lewis earned $500.00
Piece worker: Bob Lewis earned $500.00
```

```
Commission worker: Sue Jones earned $650.00
Commission worker: Sue Jones earned $650.00
Commission worker: Sue Jones earned $650.00
```

Örnek (devamı)

```
   virtualViaPointer( &h );
   virtualViaReference( h );
   cout << endl;
   return 0;
}

// Make virtual function calls off
// using dynamic binding.
void virtualViaPointer( const Employee *baseClassPtr )
{
   baseClassPtr->print();
   cout << " earned $" << baseClassPtr->earnings(
}

// Make virtual function calls off a base-class reference
// using dynamic binding.
void virtualViaReference( const Employee &baseClassRef )
{
   baseClassRef.print();
   cout << " earned $" << baseClassRef.earning
}
```

```
Hourly worker: Karen Price earned
$550.00

    Hourly worker: Karen Price earned
$550.00

    Hourly worker: Karen Price earned
$550.00
```

Bir base-sınıfı pointer ile print virtual fonksiyonu çağırmak.

Bir base-sınıfı reference ile print virtual fonksiyonu çağırmak.

C++

Program çıktısı

```
Boss: John Smith earned $800.00
             Boss: John Smith earned $800.00
             Boss: John Smith earned $800.00
Commission worker: Sue Jones earned $650.00
Commission worker: Sue Jones earned $650.00
Commission worker: Sue Jones earned $650.00
     Piece worker: Bob Lewis earned $500.00
     Piece worker: Bob Lewis earned $500.00
     Piece worker: Bob Lewis earned $500.00
    Hourly worker: Karen Price earned $550.00
    Hourly worker: Karen Price earned $550.00
    Hourly worker: Karen Price earned $550.00
```

C++

Yeni Sınıflar ve Dinamik Bağlantı

- Çoklubenzeşim ve sanal (virtual) fonksiyonlar
  - Bütün sınıflar önceden bilinmediğinde iyi çalışır.
  - Yeni sınıfları bir sisteme yerleştirirken dinamik bağlantı kullanır.
- Dinamik bağlantı (geç bağlantı)
  - Bir sanal fonksiyon için nesnenin türünün derleme zamanında bilinmesi gerekmez.
  - Sanal fonksiyon çağırma çalışma zamanında eşleştirilir.

C++

Sanal destructor

- Problem:
  - Eğer türetilmiş bir nesneye işaret eden bir kök-sınıf işaretçisi silinirse kök-sınıf destructor’ u nesne üzerinde etkili olacaktır.
- Çözüm:
  - Uygun destructor’ un çağrılacağını garanti etmek için bir sanal kök-sınıf destructor tanımlanır.

C++

Örnek: Miras Arayüzü

Point, circle, cylinder hiyerarşisini genişletmek için:

- Hiyerarşinin başı olarak Shape soyut (abstract) kök-sınıfı kullanıldı:
  - İki saf sanal fonksiyon **printShapeName** ve **print**
  - Diğer iki sanal fonksiyon **volume** ve **area**
- Point, Shape sınıfından türetildi ve miras olarak bu yapıyı aldı.

C++

Örnek (Sanal destructor)

```
// Fig. 10.2: shape.h
// Definition of abstract base class Shape
#ifndef SHAPE_H
#define SHAPE_H

class Shape {
public:
   virtual double area() const { return 0.0; }
   virtual double volume() const { return 0.0; }

   // pure virtual functions overridden in derived classes
   virtual void printShapeName() const = 0;
   virtual void print() const = 0;
};

#endif
// Fig. 10.2: point1.h
// Definition of class Point
#ifndef POINT1_H
#define POINT1_H

#include <iostream>
```

Not :Her bir sınıf tarafından tekrar yazılan virtual fonksiyonlar.

C++

Örnek (devamı)

```

using std::cout;

#include "shape.h"

class Point : public Shape {
public:
   Point( int = 0, int = 0 );  // default constru
   void setPoint( int, int );
   int getX() const { return x; }
   int getY() const { return y; }
   virtual void printShapeName() const { cout << "Point: "; }
   virtual void print() const;
private:
   int x, y;   // x and y coordinates of Point
};

#endif
```

Point abstract base sınıfından miras alıyor.

C++

Örnek (devamı)

```cpp
// Fig. 10.2: point1.cpp
// Member function definitions for class Point
#include "point1.h"

Point::Point( int a, int b ) { setPoint( a, b ); }

void Point::setPoint( int a, int b )
{
   x = a;
   y = b;
}

void Point::print() const
   { cout << '[' << x << ", " << y << ']'; }
```

C++

Örnek (devamı)

```
// Fig. 10.2: circle1.h
// Definition of class Circle
#ifndef CIRCLE1_H
#define CIRCLE1_H
#include "point1.h"

class Circle : public Point {
public:
   // default constructor
   Circle( double r = 0.0, int x = 0, int y = 0 );

   void setRadius( double );
   double getRadius() const;
   virtual double area() const;
   virtual void printShapeName() const { cout << "Circle: "; }
   virtual void print() const;
```

Circle, Point sınıfından miras alıyor.

C++

Örnek (devamı)

```
private:
   double radius;   // radius of Circle
};

#endif
// Fig. 10.2: circle1.cpp
// Member function definitions for class Circle
#include <iostream>

using std::cout;

#include "circle1.h"

Circle::Circle( double r, int a, int b )
   : Point( a, b )  // call base-class constructor
{ setRadius( r ); }
```

C++

Örnek (devamı)

```cpp

void Circle::setRadius( double r ) { radius = r > 0 ? r : 0; }

double Circle::getRadius() const { return radius; }

double Circle::area() const
   { return 3.14159 * radius * radius; }

void Circle::print() const
{
   Point::print();
   cout << "; Radius = " << radius;
}
```

C++

Örnek (devamı)

```
// Fig. 10.2: cylindr1.h
// Definition of class Cylinder
#ifndef CYLINDR1_H
#define CYLINDR1_H
#include "circle1.h"

class Cylinder : public Circle {
public:
   // default constructor
   Cylinder( double h = 0.0, double r = 0.0,
             int x = 0, int y = 0 );

   void setHeight( double );
   double getHeight();
   virtual double area() const;
```

Cylinder Circle’ den miras alıyor.

C++

Örnek (devamı)

```
   virtual double volume() const;
   virtual void printShapeName() const { cout << "Cylinder: "; }
   virtual void print() const;
private:
   double height;   // height of Cylinder
};

#endif
// Fig. 10.2: cylindr1.cpp
// Member and friend function definitions for class Cylinder
#include <iostream>

using std::cout;

#include "cylindr1.h"

Cylinder::Cylinder( double h, double r, int x, int y )
   : Circle( r, x, y )  // call base-class constructor
{ setHeight( h ); }
```

C++

Örnek (devamı)

```

void Cylinder::setHeight( double h )
   { height = h > 0 ? h : 0; }

double Cylinder::getHeight() { return height; }

double Cylinder::area() const
{
   // surface area of Cylinder
   return 2 * Circle::area() +
          2 * 3.14159 * getRadius() * height;
}

double Cylinder::volume() const
   { return Circle::area() * height; }

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

C++

Örnek (devamı)

```cpp
// Fig. 10.2: fig10 02.cpp
// Driver for shape, point, circle, cylinder hierarchy
#include <iostream>

using std::cout;
using std::endl;

#include <iomanip>

using std::ios;
using std::setiosflags;
using std::setprecision;

#include "shape.h"
#include "point1.h"
#include "circle1.h"
#include "cylindr1.h"

void virtualViaPointer( const Shape * );
void virtualViaReference( const Shape & );

```

C++

Örnek (devamı)

```
int main()
{
   cout << setiosflags( ios::fixed | ios::showpoint )
        << setprecision( 2 );

   Point point( 7, 11 );
   Circle circle( 3.5, 22, 8 );          // create a circle
   Cylinder cylinder( 10, 3.3, 10, 10 );  // create a Cylinder

   point.printShapeName();    // static binding
   point.print();             // static binding
   cout << '\n';

   circle.printShapeName();   // static binding
   circle.print();
   cout << '\n';

   cylinder.printShapeName(); // static binding
   cylinder.print();          // static binding
   cout << "\n\n";

```

```
Point: [7, 11]
Circle: [22, 8]; Radius = 3.50
Cylinder: [10, 10]; Radius = 3.30; Height = 10.00
```

Nesnenin kendisini kullanarak Print çağırılıyor.

Örnek (devamı)

```cpp
   Shape *arrayOfShapes[ 3 ];  // array of base

   // aim arrayOfShapes[0] at derived-class Poi
   arrayOfShapes[ 0 ] = &point;

   // aim arrayOfShapes[1] at derived-class Ci
   arrayOfShapes[ 1 ] = &circle;

   // aim arrayOfShapes[2] at derived-class Cy
   arrayOfShapes[ 2 ] = &cylinder;

   // Loop through arrayOfShapes and c
   // to print the shape name, attributes, area, a
   // of each object using dynamic binding.
   cout << "Virtual function calls made off "
        << "base-class pointers\n";

   for ( int i = 0; i < 3; i++ )
      virtualViaPointer( arrayOfShapes[ i ] )

   // Loop through
   // to print the
   // of each obje
```

Base sınıf işaretçileri dizisi oluşturuluyor. Bunlara nesneleri atayıp, base sınıf işaretçileriyle print fonksiyonunu tekrar çağırılınca uygun virtual fonksiyonlar çağrılacak.

```
Virtual function calls made off
base-class pointers
```

```
Point: [7, 11]
Area = 0.00
Volume = 0.00
```

```
Circle: [22, 8];
Radius = 3.50
Area = 38.48
Volume = 0.00
```

```
Cylinder: [10, 10]; Radius
= 3.30; Height = 10.00
Area = 275.77
Volume = 342.12
```

C++

Örnek (devamı)

```
   cout << "Virtual function calls made off "
        << "base-class references\n";

   for ( int j = 0; j < 3; j++ )
      virtualViaReference( *arrayOfShapes[ j ] );

   return 0;
}

// Make virtual function calls off a base-class pointer
// using dynamic binding.
void virtualViaPointer( const Shape *baseClassPtr )
{
   baseClassPtr->printShapeName();
   baseClassPtr->print();
   cout << "\nArea = " << baseClassPtr->area()
        << "\nVolume = " << baseClassPtr->volume() << "\n\n";
}

```

base-sınıfı referansları kullanılarak işlemler tekrarlanıyor.

C++

Örnek (devamı)

```cpp
// Make virtual function calls off a base-class reference
// using dynamic binding.
void virtualViaReference( const Shape &baseClassRef )
{
   baseClassRef.printShapeName();
   baseClassRef.print();
   cout << "\nArea = " << baseClassRef.area()
        << "\nVolume = " << baseClassRef.volume() << "\n\n";
}
```

```
Virtual function calls made off base-class references
Point: [7, 11]
Area = 0.00
Volume = 0.00
Circle: [22, 8]; Radius = 3.50
Area = 38.48
Volume = 0.00
Cylinder: [10, 10]; Radius = 3.30; Height = 10.00
Area = 275.77
Volume = 342.12
```

C++

Örnek (devamı)

```
Point: [7, 11]
Circle: [22, 8]; Radius = 3.50
Cylinder: [10, 10]; Radius = 3.30; Height = 10.00
Virtual function calls made off base-class pointers
Point: [7, 11]
Area = 0.00
Volume = 0.00
Circle: [22, 8]; Radius = 3.50
Area = 38.48
Volume = 0.00
Cylinder: [10, 10]; Radius = 3.30; Height = 10.00
Area = 275.77
Volume = 342.12
Virtual function calls made off base-class references
Point: [7, 11]
Area = 0.00
Volume = 0.00
Circle: [22, 8]; Radius = 3.50
Area = 38.48
Volume = 0.00
Cylinder: [10, 10]; Radius = 3.30; Height = 10.00
Area = 275.77
Volume = 342.12
```

C++

Çoklubenzeşim, sanal fonksiyonlar ve dinamik bağlama

- Ne zaman Çoklubenzeşim kullanılır?
  - Çoklubenzeşim bir çok ek yük gerektirir
  - Çoklubenzeşim performansı iyileştirmek için STL' de (Standard Template Library) kullanılmamıştır.
- sanal fonksiyon tablosu (vtable):
  - Bir sanal fonksiyona sahip her sınıf bir vtable sahibidir.
  - Her sanal fonksiyona ait vtable uygun fonksiyona işaret eden bir işaretçiye sahiptir
  - Eğer türetilmiş sınıf kök-sınıfla aynı fonksiyona sahipse fonksiyon işaretçisi kök-sınıf fonksiyonuna işaret eder.