MİMAR SİNAN FINE ARTS UNIVERSITY

Bilgisayar Programlama II
Ders 6
MSGSU Fizik Bölümü
Ferhat ÖZOK

- **Kullanılacak kaynaklar:**
  http://www.cplusplus.com/doc/tutorial/
  Published by Juan Soulié

- **C++ ile ileri programlama**
  Paul Deitel
  Harvey Deitel

MİMAR SİNAN FINE ARTS UNIVERSITY

Bilgisayar Programlama II
Ders 6
MSGSU Fizik Bölümü
Ferhat ÖZOK

Polymorphism

Pointers to base class

Sınıf kalıtımının(inheritance) en önemli özelliklerinden biri, türetilmiş bir sınıfın işaretçisinin temel sınıfa işaretçi ile tür uyumlu olmasıdır. Polimorfizm, bu basit ama güçlü ve çok yönlü özelliğin avantajlarından yararlanma sanatıdır.

Daha önceki derslerimizde gördüğümüz Dikdörtgen ve üçgen sınıfları ile ilgili örnek, bu özelliği dikkate alan işaretçileri kullanarak yeniden yazılabilir:

```
// pointers to base class
#include <iostream>
using namespace std;
class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
};
class Rectangle: public Polygon {
  public:
    int area()
      { return width*height; }
};
class Triangle: public Polygon {
  public:
    int area()
      { return width*height/2; }
};
int main () {
  Rectangle rect;
  Triangle trgl;
  Polygon * ppoly1 = &rect;
  Polygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  cout << rect.area() << '\n';
  cout << trgl.area() << '\n';
  return 0;
}
```

MİMAR SİNAN FINE ARTS UNIVERSITY

Bilgisayar Programlama II
Ders 6
MSGSU Fizik Bölümü
Ferhat ÖZOK

Main fonksiyonunda, Polygon'a iki işaretçiyi bildirilr (ppoly1 ve ppoly2 olarak adlandırılmış nesneler). Bunlara sırasıyla dikdörtgen ve üçgenli nesneler olan rect ve trgl adresleri atanır. Hem Rectangle hem de Triangle, Polygon'dan türetilmiş sınıflar olduğundan, bu atamalar geçerlidir.

Ayrıştırma ppoly1 ve ppoly2 (* ppoly1 ve * ppoly2 ile) geçerlidir ve işaret edilen nesnelerinin üyelerine erişmemize izin verir. Örneğin, aşağıdaki iki ifade önceki örnekte eşdeğer olacaktır:
ppoly1->set_values (4,5);
rect.set_values (4,5);

Ancak, ppoly1 ve ppoly2'nin türü, Çokgen'e (ve Dikdörtgen'e işaretçi ya da Üçgen'e işaretçi olmadığından) işaretçi olmadığından sadece Çokgen'den devralınan üyelere erişilebilir.

MİMAR SİNAN FINE ARTS UNIVERSITY

Bilgisayar Programlama II
Ders 6
MSGSU Fizik Bölümü
Ferhat ÖZOK

Virtual members

Sanal üye, çağrı özelliklerini referanslar vasıtasıyla koruyarak türetilmiş bir sınıfta yeniden tanımlanabilen bir üye işlevdir. Sanal olabilmesi için bir işlevin sözdizimi, bildiriminin öncesinde virtual anahtar sözcüğüyle gelmektir:
Bu örnekte üç sınıfın (Çokgen, Dikdörtgen ve Üçgen) aynı üyeleri vardır: genişlik, yükseklik ve fonksiyonlar set_values ve area.

Türetilmiş sınıfların her birinde daha sonra yeniden tanımlandığından üye area fonksiyonu taban sınıfında sanal olarak bildirilmiştir. Sanal olmayan üyeler türetilmiş sınıflarda da yeniden tanımlanabilir, ancak türetilmiş sınıfların sanal olmayan üyelerine temel sınıfın bir referansı yoluyla erişilemez:
ör. yandaki örnekte alanın bildiriminden sanal çıkarılırsa, üç çağrı da area sıfır döndürür, çünkü her durumda, temel sınıfın sürümü yerine çağrılırdı.

Dolayısıyla, esas olarak, sanal anahtar kelimenin yaptığı şey, temel sınıftaki bir sınıfla aynı ada sahip türetilmiş bir sınıf üyesine, bir işaretçiden uygun şekilde çağrılabilmesine izin vermektir; daha kesin olarak ise, işaretçinin türü, yandaki örnekte olduğu gibi türetilmiş sınıfın bir nesnesini işaret eden temel sınıf.

Sanal bir fonksiyon beyan eden veya devralan bir sınıf, çok biçimli (polymorphic) bir sınıf olarak adlandırılır.

Polygon, üyelerinden birinin sanallığına rağmen, düzenli bir sınıftır; bu nesnenin bir nesnesi bile (poly) oluşturulur ve her zaman 0 değerini alan area fonksiyonu tanımlanır.

```
// virtual members
#include <iostream>
using namespace std;
class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area ()
      { return 0; }
};
class Rectangle: public Polygon {
  public:
    int area ()
      { return width * height; }
};
class Triangle: public Polygon {
  public:
    int area ()
      { return (width * height / 2); }
};
int main () {
  Rectangle rect;
  Triangle trgl;
  Polygon poly;
  Polygon * ppoly1 = &rect;
  Polygon * ppoly2 = &trgl;
  Polygon * ppoly3 = &poly;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  ppoly3->set_values (4,5);
  cout << ppoly1->area() << '\n';
  cout << ppoly2->area() << '\n';
  cout << ppoly3->area() << '\n';
  return 0;
}
```

Bilgisayar Programlama II
Ders 6
MSGSU Fizik Bölümü
Ferhat ÖZOK

Abstract base classes

Özet(abstract) taban sınıfları, önceki örnekteki Poligon sınıfına çok benzer bir şeydir. Bunlar yalnızca taban sınıfları olarak kullanılabilen sınıflardır ve dolayısıyla tanım olmadan sanal üye işlevlerine sahip olmalarına izin verilir (saf sanal işlevler olarak bilinir). Sözdizimi tanımlarını = 0 (eşittir işareti ve sıfır) ile değiştirmektir:

```
// abstract class CPolygon

class Polygon {

  protected:

    int width, height;

  public:

    void set_values (int a, int b)

      { width=a; height=b; }

    virtual int area () =0;

};
```

area hiçbir tanımının olmadığına dikkat edin; bu onu, saf bir sanal fonksiyon haline getiren = 0 ile değiştirildi. En az bir saf sanal fonksiyon içeren sınıflar soyut temel sınıflar olarak bilinir.
Özet temel sınıflar, nesneleri oluşturmak için kullanılamaz.
Bu nedenle, Polygon'ın bu son özet taban sınıfı sürümü gibi nesneleri bildirmek için kullanılamaz:
Polygon mypolygon; // özet taban sınıfı koşulunda bu çalışmaz
Aşağıdaki kullanım geçerlidir:
Polygon * ppoly1;
Polygon * ppoly2;

MİMAR SİNAN FINE ARTS UNIVERSITY

Bilgisayar Programlama II
Ders 6
MSGSU Fizik Bölümü
Ferhat ÖZOK

```
// abstract base class
#include <iostream>
using namespace std;
class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area (void) =0;
};
class Rectangle: public Polygon {
  public:
    int area (void)
      { return (width * height); }
};
class Triangle: public Polygon {
  public:
    int area (void)
      { return (width * height / 2); }
};
int main () {
  Rectangle rect;
  Triangle trgl;
  Polygon * ppoly1 = &rect;
  Polygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  cout << ppoly1->area() << '\n';
  cout << ppoly2->area() << '\n';
  return 0;
}
```

Bilgisayar Programlama II
Ders 6
MSGSU Fizik Bölümü
Ferhat ÖZOK

```
// pure virtual members can be called
// from the abstract base class
#include <iostream>
using namespace std;
class Polygon {
  protected:
    int width, height;
  public:
    void set_values (int a, int b)
      { width=a; height=b; }
    virtual int area() =0;
    void printarea()
      { cout << this->area() << '\n'; }
};
class Rectangle: public Polygon {
  public:
    int area (void)
      { return (width * height); }
};
class Triangle: public Polygon {
  public:
    int area (void)
      { return (width * height / 2); }
};
int main () {
  Rectangle rect;
  Triangle trgl;
  Polygon * ppoly1 = &rect;
  Polygon * ppoly2 = &trgl;
  ppoly1->set_values (4,5);
  ppoly2->set_values (4,5);
  ppoly1->printarea();
  ppoly2->printarea();
  return 0;
}
```

Bu örnekte, farklı fakat ilişkili türdeki nesnelere tek bir işaretçi türü (Polygon *) kullanılarak atıfta bulunulur ve uygun üye fonksiyonları her zaman sanal oldukları için çağrılır. Bu, bazı durumlarda gerçekten yararlı olabilir. Örneğin, özet taban sınıfı Poligon üyesi, Poligonun kendisinin bu fonsiyon için hiçbir uygulaması olmadığı halde, uygun sanal üyelere erişmek için bu özel işaretçinin kullanılmasını mümkün kılar

MİMAR SİNAN FINE ARTS UNIVERSITY

Bilgisayar Programlama II
Ders 6
MSGSU Fizik Bölümü
Ferhat ÖZOK

```
// dynamic allocation and polymorphism
#include <iostream>
using namespace std;
class Polygon {
  protected:
    int width, height;
  public:
    Polygon (int a, int b) : width(a), height(b) {}
    virtual int area (void) =0;
    void printarea()
      { cout << this->area() << '\n'; }
};
class Rectangle: public Polygon {
  public:
    Rectangle(int a,int b) : Polygon(a,b) {}
    int area()
      { return width*height; }
};
class Triangle: public Polygon {
  public:
    Triangle(int a,int b) : Polygon(a,b) {}
    int area()
      { return width*height/2; }
};
int main () {
  Polygon * ppoly1 = new Rectangle (4,5);
  Polygon * ppoly2 = new Triangle (4,5);
  ppoly1->printarea();
  ppoly2->printarea();
  delete ppoly1;
  delete ppoly2;
  return 0;
}
```

MİMAR SİNAN FINE ARTS UNIVERSITY

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

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

   void setCommissionRate( double ); // set commission rate
   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
```

MİMAR SİNAN FINE ARTS UNIVERSITY

CommissionEmployee.cpp:

```
#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 ), lastName( last ), socialSecurityNumber( ssn )
{
   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

// 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

// 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

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

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

// print CommissionEmployee object
void CommissionEmployee::print() const
{
   cout << "commission employee: "
      << getFirstName() << ' ' << getLastName()
      << "\nsocial security number: " << getSocialSecurityNumber()
      << "\ngross sales: " << getGrossSales()
      << "\ncommission rate: " << getCommissionRate();
} // end function print
```

BasePlusCommissionEmployee.hh :

```
#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
```

BasePlusCommissionEmployee.cpp

```
#include <iostream>
#include "BasePlusCommissionEmployee.h"
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
// return base salary
double BasePlusCommissionEmployee::getBaseSalary() const
{
   return baseSalary;
} // end function getBaseSalary
// calculate earnings
double BasePlusCommissionEmployee::earnings() const
{
   return getBaseSalary() + CommissionEmployee::earnings();
} // end function earnings
// print BasePlusCommissionEmployee object
void BasePlusCommissionEmployee::print() const
{
   cout << "base-salaried ";

   // invoke CommissionEmployee's print function
   CommissionEmployee::print();

   cout << "\nbase salary: " << getBaseSalary();
} // end function print
```

MİMAR SİNAN FINE ARTS UNIVERSITY

Ders6_poly_ornek.cpp

```
#include <iostream>
#include <iomanip>
#include "CommissionEmployee.h"
#include "BasePlusCommissionEmployee.h"
using namespace std;

int main()
{
   // create base-class object
   CommissionEmployee commissionEmployee(
      "Sue", "Jones", "222-22-2222", 10000, .06 );
   // create base-class pointer
   CommissionEmployee *commissionEmployeePtr = 0;
   // create derived-class object
   BasePlusCommissionEmployee basePlusCommissionEmployee(
      "Bob", "Lewis", "333-33-3333", 5000, .04, 300 );
   // create derived-class pointer
   BasePlusCommissionEmployee *basePlusCommissionEmployeePtr = 0;
   // set floating-point output formatting
   cout << fixed << setprecision( 2 );
   // output objects commissionEmployee and basePlusCommissionEmployee
   cout << "Print base-class and derived-class objects:\n\n";
   commissionEmployee.print(); // invokes base-class print
   cout << "\n\n";
   basePlusCommissionEmployee.print(); // invokes derived-class print
      // aim base-class pointer at base-class object and print
   commissionEmployeePtr = &commissionEmployee; // perfectly natural
   cout << "\n\n\nCalling print with base-class pointer to "
      << "\nbase-class object invokes base-class print function:\n\n";
   commissionEmployeePtr->print(); // invokes base-class print
   // aim derived-class pointer at derived-class object and print
   basePlusCommissionEmployeePtr = &basePlusCommissionEmployee; // natural
   cout << "\n\n\nCalling print with derived-class pointer to "
      << "\nderived-class object invokes derived-class "
      << "print function:\n\n";
   basePlusCommissionEmployeePtr->print(); // invokes derived-class print
   // aim base-class pointer at derived-class object and print
   commissionEmployeePtr = &basePlusCommissionEmployee;
   cout << "\n\n\nCalling print with base-class pointer to "
      << "derived-class object\ninvokes base-class print "
      << "function on that derived-class object:\n\n";
   commissionEmployeePtr->print(); // invokes base-class print
   cout << endl;
} // end main
```

MİMAR SİNAN FINE ARTS UNIVERSITY

Programı Derlemek için:

```
g++   CommissionEmployee.cpp BasePlusCommissionEmployee.cpp Ders6_poly_ornek.cpp -o Ders6
```