BM102
Algoritma ve Programlama II

Çoklu İşlev

İçerik

1. Çoklu İşlevin Gerçekleştirilmesi
2. Saf Sanal Fonksiyonlar ve Soyut Sınıflar
3. Sanal Yıkıcı Fonksiyonlar
4. Statik ve Dinamik Bağlama

Çözümlü Sorular

Hedefler

- Çoklu işlev prensibinin amacını anlatma
- Çoklu işlev yönteminin kullanılmasına yönelik gereksinimleri listeleme
- Verilen program parçasında çoklu işlevin uygulandığı yeri gösterme
- Verilen programı, çoklu işlev yöntemi ile yeniden yazma
- Sanal fonksiyonlar ve saf sanal fonksiyonlar arasındaki farkı açıklama
- Verilen problem tanımından soyut sınıfları çıkarma
- Soyut ve somut sınıflar arasındaki farkları listeleme
- Soyut sınıflarına yönelik UML sınıf diyagramını çizme
- Sanal yıkıcı fonksiyon tanımlama
- Statik ve dinamik bağlama arasındaki farkları anlatma

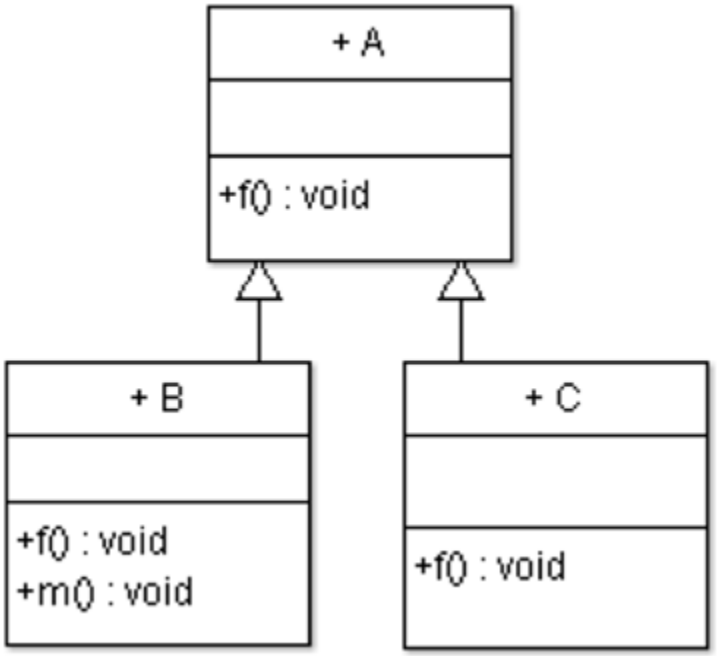
Çoklu İşlev

- *İng. Polymorphism*
- Nesne tabanlı programlamanın temel prensiplerinden biridir.
- Bir nesne göstergesinin veya referans değişkeninin farklı tipte nesneleri göstermesi ve gösterilen nesne tipine göre üye fonksiyonunu dinamik olarak çağırılmasıdır.
- Sınıflar arasında miras ilişkisi olması gerekir.
- Üst sınıfa tanımlı olan fonksiyonların alt sınıf üzerine yazılması yani yeniden tanımlanmış olması gerekir.

1. Çoklu İşlevin Gerçekleştirilmesi

Üst Sınıf Tipinde Bir Göstergenin Alt Sınıf Nesnelerini Göstermesi

Örnek:



1 Çoklu İşlevin Gerçekleştirilmesi…

```
//---------------- A.h ----------------
#ifndef A_H
#define A_H
class A        // Üst sınıf tanımı
{
    public:
        void f();
};
#endif
//---------------- A.cpp ----------------
#include "A.h"
#include <iostream>
using namespace std;
void A::f()
{
    cout<<"A"<<endl;
}
```

```
//---------------- B.h ----------------
#include "A.h"
class B:public A     // A sınıfından türemiş alt sınıf tanımı
{
    public:
        void f();
        void m();
};
//---------------- B.cpp ----------------
#include "B.h"
void B::f()
{
    cout<<"B"<<endl;
}
void B::m()
{
    cout<<"m fonksiyonu"<<endl;
}
```

1 Çoklu İşlevin Gerçekleştirilmesi…

```
//---------------- C.h ----------------
#include "A.h"
class C:public A                    // A sınıfından türemiş alt sınıf tanımı
{
    public:
        void f();
};
```

```
//---------------- C.cpp ----------------
#include "C.h"
void C::f()
{
    cout<<"C"<<endl;
}
```

1 Çoklu İşlevin Gerçekleştirilmesi…

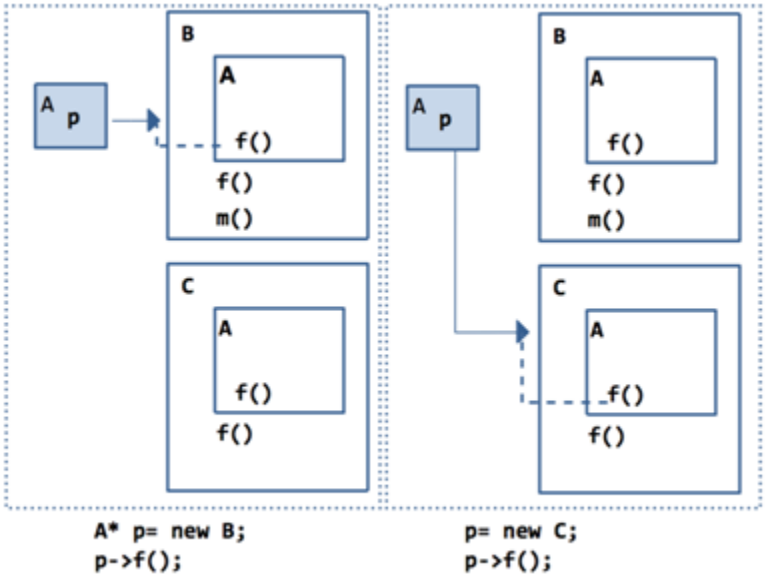
```
//---------------- Uygulama.cpp ----------------
#include "A.h"
#include "B.h"
#include "C.h"
int main(){
        A* p= new B;        // A tipindeki gösterge B tipindeki nesneyi gösterir
        p->f();             // A::f() çağırılır
        p= new C;           // A tipindeki gösterge C tipindeki nesneyi gösterir
        p->f();             // A::f() çağırılır
        delete p;
        return 0;
}
```

Çıktı

```
A
A
```

- Çoklu işlev yöntemi, çalışma zamanında bir göstergenin gösterdiği tipteki nesneye ait fonksiyonu çağırabilmesini sağlar. Ancak bu örnekte çoklu işlev uygulanamamışar. Bu yöntemin uygulanması için **sanal (virtual)** fonksiyon kullanımı gerekir.

1 Çoklu İşlevin Gerçekleştirilmesi…



A Tipinde p Göstergesi ve Çalışma Zamanındaki Davranışı

Üst sınıf tipinde tanımlanmış bir gösterge alt sınıf nesnesini gösterse bile sadece üst sınıf üyelerine erişme hakkına sahiptir.

```
A* p = new B;
p->m(); // X Hata
```

Yukarıdaki komutta, p göstergesi A tipinde tanımlanır. Ancak p göstergesi, B sınıfının üyesi olan m() fonksiyonuna erişmeye çalıştığı için geçersiz olacaktır.

1 Çoklu İşlevin Gerçekleştirilmesi...

- **Sanal Fonksiyonlar** (*İng. Virtual Functions*)
- Fonksiyonun *üzerine yazma* (overriding), üst sınıfta yer alan bir fonksiyonun, aynı isim ve parametre listesi ile alt sınıfta tekrar tanımlanmasıdır.
- Fonksiyon üzerine yazma işleminin gerçekleşebilmesi için bu fonksiyonların *sanal* (virtual) fonksiyonlar olarak tanımlanması gerekir.
- `virtual` anahtar kelimesi ile tanımlanır.

1 Çoklu İşlevin Gerçekleştirilmesi…

```
//---------------- A.h ----------------
#ifndef A_H
#define A_H
class A
{
    public:
        virtual void f(); // virtual fonksiyon prototipi
};
#endif
//---------------- A.cpp ----------------
#include "A.h"
#include <iostream>
using namespace std;
void A::f()
{
    cout<<"A"<<endl;
}
```

1 Çoklu İşlevin Gerçekleştirilmesi…

```
//---------------- B.h ----------------
#include "A.h"
class B:public A
{
    public:
        void f();
};
//---------------- B.cpp ----------------
#include "B.h"
#include <iostream>
using namespace std;
void B::f()
{
    cout<<"B"<<endl;
}
```

```
//---------------- C.h ----------------
#include "A.h"
class C:public A
{
    public:
        void f();
};
```

```
//---------------- C.cpp ----------------
#include "B.h"
void C::f()
{
    cout<<"C"<<endl;
}
```

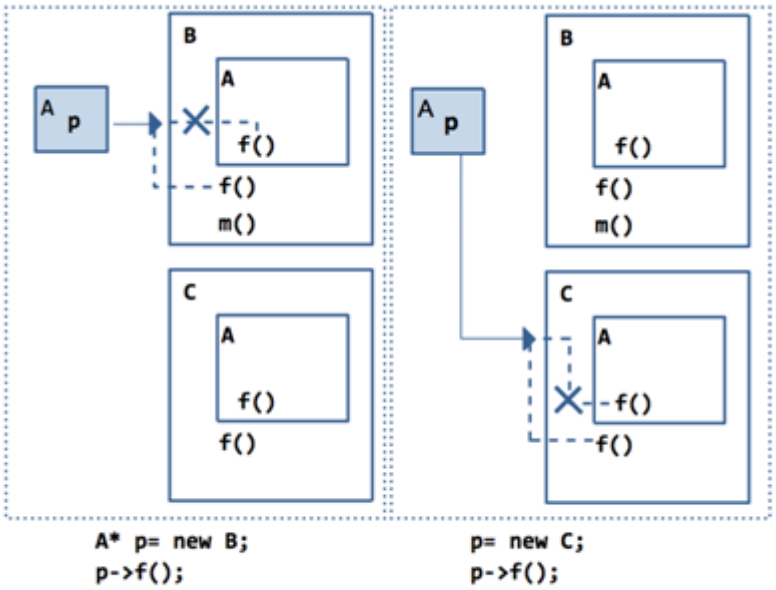
1 Çoklu İşlevin Gerçekleştirilmesi…

```
//---------------- Uygulama.cpp ----------------
#include "A.h"
#include "B.h"
#include "C.h"
int main()
{
    A* p= new B;      // A tipindeki gösterge B tipindeki nesneyi gösterir
    p->f();           // B::f() çağırılır
    p= new C;         // A tipindeki gösterge C tipindeki nesneyi gösterir
    p->f();           // C::f() çağırılır
    delete p;
}
```

Çıktı

B
C

1 Çoklu İşlevin Gerçekleştirilmesi…



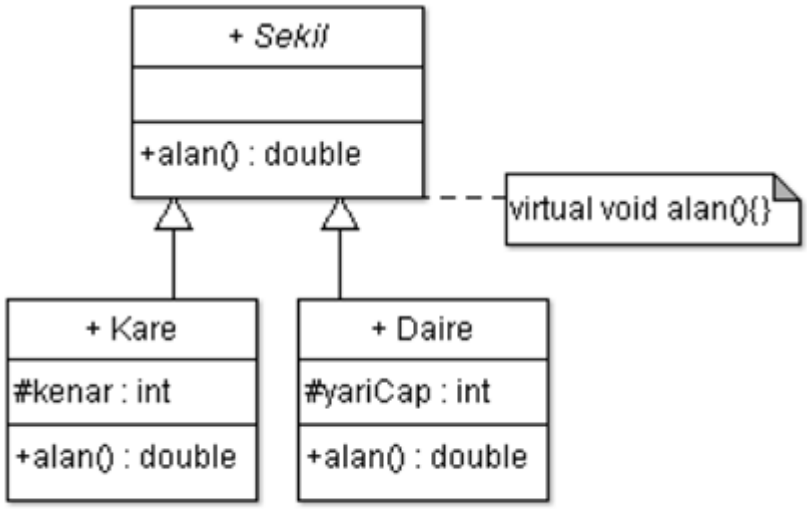
Sanal Fonksiyonlar Kullanıldığında A Tipindeki p Göstergesinin Çalışma Zamanındaki Davranışı

Çoklu işlev yönteminin uygulanması için aşağıdaki tanımlamaların yapılması gerekir:

- sınıflar arasında miras ilişkisi kurulması,
- üst sınıf tipinde bir gösterge veya referans değişkeni tanımı
- üst sınıfta sanal fonksiyon tanımı
- alt sınıfların bu sanal fonksiyonların üzerine yazması

1 Çoklu İşlevin Gerçekleştirilmesi…

- **Örnek**: Sekil, Daire, ve Kare sınıfları oluşturalım.
- Her şeklin bir alanı vardır, ancak şeklin türüne göre alan hesabı farklılaşır.
- Örneğin karenin alanı kenar^2 iken dairenin alanı ise πr^2'dir.



Sekil, Daire ve Kare sınıfları

1 Çoklu İşlevin Gerçekleştirilmesi…

```
//---------- Sekil.h ----------
#ifndef  _SEKIL_H
#define _SEKIL_H
class Sekil
{
    public:
        virtual double alan() { return 0;}   // virtual üye fonksiyon
};
#endif
//---------- Daire.h ----------
#include "Sekil.h"
class Daire: public Sekil
{
    protected:
        int yaricap;
    public:
         Daire(int _yaricap):yaricap(_yaricap){}
         double alan();
};
//---------- Daire.cpp ----------
#include "Daire.h"
double Daire::alan()        // Dairenin alanını hesaplayan fonksiyon
{
    return (yaricap*yaricap*3.14);
}
```

1 Çoklu İşlevin Gerçekleştirilmesi…

```
//---------------- Kare.h ----------------
#include "Sekil.h"
class Kare: public Sekil
{
    protected:
        int kenar;
    public:
        Kare(int _kenar): kenar(_kenar){}
        double alan();
};
//---------------- Kare.cpp----------------
#include "Kare.h"
double Kare::alan() // Karenin alanını hesaplayan fonksiyon
{
    return kenar*kenar;
}
```

1 Çoklu İşlevin Gerçekleştirilmesi…

```
//---------- Uygulama.cpp ----------
#include <iostream>
#include "Daire.h"
#include "Kare.h"
#include "Sekil.h"
using namespace std;
int main()
{
    Daire daire(10);
    Kare kare(5);
    Sekil* sekilGosterge = &daire;      // Sekil göstergesi daireyi gösteriyor
    cout<<"Alan (Daire) = "<<sekilGosterge->alan()<<endl;
    sekilGosterge = &kare;              // Sekil göstergesi kareyi gösteriyor
    cout<<"Alan (Kare) = "<<sekilGosterge->alan()<<endl;
    return 0;
}
```

Çıktı

```
Alan (Daire) = 314
Alan (Kare) = 25
```

1 Çoklu İşlevin Gerçekleştirilmesi…

Fonksiyon üzerine yazma (overriding) ile fonksiyon yükleme (overloading) kavramları arasındaki farkı vurgulamakta fayda vardır.

- Fonksiyon yükleme aynı fonksiyon isminin farklı parametrelerle yeniden tanımlanmasıdır. Fonksiyon yüklemede fonksiyonun sınıf içinde tanımlanması veya miras ilişkisi kurulması gerekli değildir.
- Fonksiyon üzerine yazma ise üst sınıfta tanımlanan `virtual` fonksiyonun alt sınıflarda aynı isim ve parametrelerle yeniden tanımlanmasıdır.

2 Saf Sanal Fonksiyonlar

- *İng. Pure virtual functions*
- Üst sınıfta prototipi verilen sanal fonksiyonların alt sınıflarda üzerine yazılmalarını şart koşmak için kullanılan fonksiyonlardır.
- Söz dizimi:

```
virtual VeriTipi fonksiyonAdi() = 0 ;
```

- =0 ifadesi bu fonksiyonun saf sanal fonksiyon olduğunu tanımlar.
- Bu fonksiyonların içerik kısmı ({…}) yoktur, sadece fonksiyonun prototipi belirtilir.

2 Saf Sanal Fonksiyonlar…

- **Soyut Sınıf** (*İng. Abstract class*)
- İçerisinde en az bir saf sanal fonksiyon bulunan sınıftır
- Soyut sınıflarda saf sanal fonksiyonlara ek olarak diğer veri ve fonksiyon üyeleri de tanımlanabilir.
- Bu sınıflarda tanımlanan saf sanal fonksiyonların içerikleri olamadığı için, bu sınıflardan nesne oluşturulmaz.
- Çoğunlukla oluşturulacak sınıf hiyerarşisinde üst sınıf olarak kullanılırlar.
- Bir sınıf, soyut bir sınıftan türediğinde, bu sınıfta yer alan saf sanal fonksiyonların üzerine yazması gerekir.
- Saf sanal fonksiyonların üzerine yazmadığı durumda alt sınıf sanal sınıf olarak kabul edilir.

2 Saf Sanal Fonksiyonlar…

```
//---------- Sekil.h ----------
#ifndef _SEKIL_H
#define _SEKIL_H
class Sekil                                  // Soyut sınıf tanımı
{
    public:
        virtual double alan() = 0;           // Saf sanal fonksiyon tanımı
};
#endif
//---------- Daire.h ----------
#include "Sekil.h"
class Daire: public Sekil
{
    protected:
        int yaricap;
    public:
         Daire(int _yaricap):yaricap(_yaricap){}
         double alan();
};
//---------- Daire.cpp ----------
#include "Daire.h"
double Daire::alan()
{
    return (yaricap*yaricap*3.14);
}
```

2 Saf Sanal Fonksiyonlar…

```
//---------------- Kare.h ----------------
#include "Sekil.h"
class Kare: public Sekil
{
    protected:
        int kenar;
    public:
        Kare(int kenar): kenar(_kenar){}
        double alan();
};
//---------------- Kare.cpp----------------
#include "Kare.h"
double Kare::alan()
{
    return kenar*kenar;
}
```

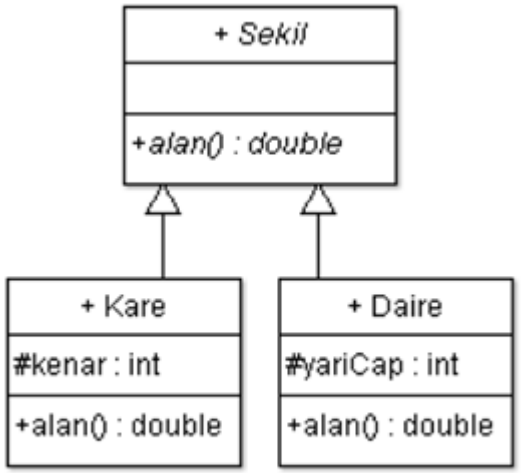
2 Saf Sanal Fonksiyonlar…

```
//---------------- Uygulama.cpp ----------------
#include "Daire.h"
#include "Kare.h"
#include "Sekil.h"
#include <iostream>
using namespace std;
int main()
{
    Daire daire(10);
    Kare kare(5);
    Sekil* sekilGosterge = &daire;
    cout<<"Alan (Daire) = "<<sekilGosterge->alan()<<endl;
    sekilGosterge = &kare;
    cout<<"Alan (Kare) = "<<sekilGosterge->alan()<<endl;
    return 0;
}
```

Çıktı

```
Alan (Daire) = 314
Alan (Kare) = 25
```

2 Saf Sanal Fonksiyonlar…



Soyut Sekil Sınıfı

Soyut sınıf ve saf sanal fonksiyonlar italic yazı formatı ile belirtilir

```
Sekil sekil;                    // X Hatalı tanım
Sekil* sekilGosterge;           // Geçerli tanım
Kare kare(5);
Sekil& sekilRef = kare;         // Geçerli tanım
```

İçinde içeriği tanımlanmamış saf sanal fonksiyon(lar) bulunduran sınıflara soyut sınıf adı verilir. Bu sınıflardan nesne yaratılamaz.
İçinde hiçbir saf sanal fonksiyon olmayan sınıflara ise ***somut sınıf*** (*concrete class*) adı verilir. Bu sınıflardan nesne yaratılabilir.

3 Sanal Yıkıcı Fonksiyonlar

- *İng. Virtual destructors*
- İki sınıf arasında bir miras ilişkisi var ise, her iki sınıfında yıkıcı fonksiyonlarıçağırılmalıdır.
- *Virtual yıkıcı fonksiyonlar* kullanarak alt sınıfın yıkıcı fonksiyonunun da çağırılmasını sağlanır.

3 Sanal Yıkıcı Fonksiyonlar…

```
//---------------- A.h ----------------
#ifndef A_H
#define A_H
#include <iostream>
using namespace std;
class A
{
    public:
        A(){ cout<<"A";}          // Yapıcı fonksiyon
         ~A(){ cout<<"A yok edildi.";}      // Yıkıcı fonksiyon
};
#endif
//---------------- B.h ----------------
#include "A.h"
class B: public A
{
    public:
        B(){ cout<<"B";}          // Yapıcı fonksiyon
         ~B(){ cout<<"B yok edildi.";}      // Yıkıcı fonksiyon
};
```

3 Sanal Yıkıcı Fonksiyonlar…

```
//---------------- Uygulama.cpp ----------------
#include "A.h"
#include "B.h"
int main()
{
    A  *aPtr = new B();   // A tipindeki gösterge B tipinde nesneyi gösterir
    delete aPtr;
    return 0;
}
```

Çıktı

```
A
B
A yok edildi.
```

3 Sanal Yıkıcı Fonksiyonlar…

- Virtual yıkıcı fonksiyonlar kullanarak alt sınıfın yıkıcı fonksiyonunun da çağırılmasını sağlayabiliriz.

```
//---------- A.h ----------
#ifndef A_H
#define A_H
#include <iostream>
using namespace std;
class A
{    public:
        A(){ cout<<"A";}
        virtual ~A(){cout<<"A yok edildi. "<<endl; // virtual yıkıcı fonksiyon
                     system("pause");}
};
#endif
//---------- B.h ----------
#include "A.h"
class B: public A
{    public:
        B(){ cout<<"B";}
        virtual ~B(){cout<<"B yok edildi. "<<endl; // virtual yıkıcı fonksiyon
                     system("pause");}
};
```

3 Sanal Yıkıcı Fonksiyonlar…

```
//----------------- Uygulama.cpp -----------------
#include "A.h"
#include "B.h"
int main()
{       A  *aPtr = new B();
        delete aPtr;
        return 0;
}
```

Çıktı

```
A
B
B yok edildi.
Press any key to continue...
A yok edildi.
Press any key to continue...
```

4 Statik ve Dinamik Bağlama

- ***Statik Bağlama*** (İng. Static binding)
- Derleyicinin, program içerisinde hangi fonksiyonun çağrılacağını derleme zamanında belirlemesine *statik bağlama* adı verilir.
  - Böylece, daha program çalışarılmadan program içerisinde çağrılacak olan fonksiyonlar belirlenmiş ve derleyici tarafından kod içerisine gömülmüş olur.
  - Bu da çalışma zamanında fonksiyon çağırmak için harcanan zamanı azaltır ve performansı arttırır.

4 Statik ve Dinamik Bağlama…

- ***Dinamik Bağlama*** *(İng. Dynamic binding)*
- Program içerisinde oluşturulan nesneler çalışma zamanında farklılaşırsa, çalışma zamanından önce bir ilişkilendirme yapılmaması gerekir.
- Programcı derleyiciye hangi fonksiyonun çalışma zamanında bağlanacağını belirtebilir.
- Çalışma zamanında yapılan ilişkilendirmeye ***dinamik bağlama*** adı verilir.
- Sanal olarak tanımlanan tüm fonksiyonlar derleyici tarafından çalışma zamanında bağlanmak üzere etiketlenir.
- Dinamik bağlama, nesne tabanlı programlama ile gelen çoklu işlev ve sarmalama kavramlarının gerçekleşmesine olanak sağlayan önemli bir kavramdır.
- Statik bağlamaya nazaran daha fazla zaman alsa da, çalışma zamanında değişen nesnelere göre dinamik olarak fonksiyon çağırımına olanak verdiği için programa esneklik kazandırır.

Çözümlü Sorular

Soru

2. Aşağıda verilen programın çıktısını yazınız.

```
//---------------- A.h ----------------
#ifndef A_H
#define A_H
#include <iostream>
using namespace std;
class A
{  public:
       A(){
         cout << "1. A" << endl;
       }
       ~A()
       {
         cout << "2. ~A" << endl;
       }
       virtual void f1();
       void f2();
};
#endif
```

Çözümlü Sorular

Soru…

```
//---------- A.cpp ----------
#include "A.h"
void A::f1()
{
    cout << "3. A.f1()" << endl;
}
void A::f2()
{
    cout << "4. A.f2()" << endl;
}
//---------- B.h ----------
#include "A.h"
class B: public A
{
   public:
      B()
      {
        cout << "5. B" << endl;
      }
      void f1();
      void f2();
};
```

```
//---------- B.cpp ----------
#include "B.h"
void B::f1()
{
    cout << "6. B.f1()" << endl;
}
void B::f2()
{
    cout << "7. B.f2()"<< endl;
}
//---------- Uygulama.cpp ----------
#include "A.h"
#include "B.h"
int main()
{
    A a;
    B b;
    A *aGosterge = new B;
    a.f1();
    a.f2();
    b.f1();
    b.f2();
    aGosterge->f1();
    aGosterge->f2();
    delete aGosterge;
    return 0;
}
```

Çözümlü Sorular

Cevap

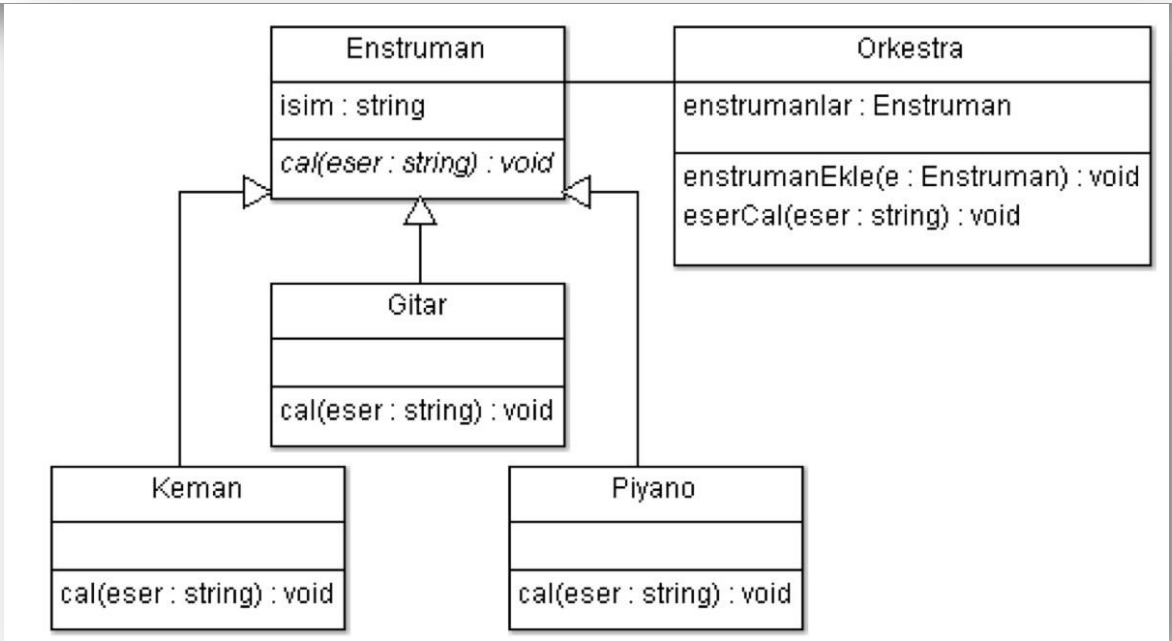
2.

```
1. A
1. A
5. B
1. A
5. B
3. A.f1()
4. A.f2()
6. B.f1()
7. B.f2()
6. B.f1()
4. A.f2()
2. ~A
```

Çözümlü Sorular

Soru

4. Bu soruda bir orkestra nesnesini modellemeniz beklenmektedir. Bir orkestraya herhangi bir enstrüman eklenebilmektedir. Orkestradan bir eseri çalması istendiğinde, orkestrada yer alan tüm enstrümanlar kendilerine özgü eseri çalmaya başlayacaklardır. Problem tanımına ek olarak UML sınıf diyagramı, `Enstruman` sınıfı, `main()` fonksiyonu ve örnek çıktı verilmiştir. Buna göre gerekli sınıf ve fonksiyon tanımlamalarını yapınız.



Çözümlü Sorular

Soru

```
class Enstruman
{
    protected:
        string isim;
    public:
        Enstruman(string _isim):isim(_isim){}
        virtual void cal() = 0;
};
     //...
int main()
{
    Orkestra orkestra;
    Keman keman;
    Piyano piyano;
    Gitar gitar;
    orkestra.enstrumanEkle(&keman);
    orkestra.enstrumanEkle(&piyano);
    orkestra.enstrumanEkle(&gitar);
    orkestra.eserCal("Pavane");
    return 0;
}
```

Örnek Çıktı

```
Keman caliyor…
Piyano caliyor…
Gitar caliyor…
```

Çözümlü Sorular

Cevap…

```
// ---------------------- Enstruman.h ----------------------
#ifndef ENSTRUMAN_H
#define ENSTRUMAN_H
#include <iostream>
using namespace std;
class Enstruman
{
    protected:
        string isim;
    public:
        Enstruman(string _isim):isim(_isim){}
        virtual void cal(string eser) = 0;
};
#endif
```

Çözümlü Sorular

Cevap…

```
// ---------- Gitar.h ----------
#include "Enstruman.h"
#include <iostream>
using namespace std;
class Gitar: public Enstruman
{
    public:
        Gitar():Enstruman("Gitar"){}
        void cal(string eser);
};
```

```
// ---------- Gitar.cpp ----------
#include "Gitar.h"
#include <iostream>
using namespace std;

void Gitar::cal(string eser)


{
    cout<<isim<<" caliyor..."<<endl;
}
```

Çözümlü Sorular

Cevap…

```
// ---------------------- Keman.h ----------------------
#include "Enstruman.h"
#include <iostream>
using namespace std;
class Keman: public Enstruman
{
    public:
        Keman():Enstruman("Keman"){}
        void cal(string eser);
};
// ---------------------- Keman.cpp ----------------------
#include "Keman.h"
#include <iostream>
using namespace std;
void Keman::cal(string eser)
{
    cout<<isim<<" caliyor..."<<endl;
}
```

```
// ---------------------- Piyano.h ----------------------
#include "Enstruman.h"
#include <iostream>
using namespace std;
class Piyano: public Enstruman
{
    public:
        Piyano():Enstruman("Piyano"){}
        void cal(string eser);
};
// ---------------------- Piyano.cpp ----------------------
#include "Piyano.h"
#include <iostream>
using namespace std;
void Piyano::cal(string eser)
{
    cout<<isim<<" caliyor..."<<endl;
}
```

Çözümlü Sorular

Cevap…

```
// ---------- Orkestra.h ----------
#include "Enstruman.h"
#include <iostream>
using namespace std;
class Orkestra
{
    Enstruman* enstrumanlar[5];

    static int x;
    public:
        void enstrumanEkle(Enstruman* e);
        void eserCal(string eserAdi);
};
```

```
// ---------- Orkestra.cpp ----------
#include "Enstruman.h"
#include "Orkestra.h"
#include <iostream>
using namespace std;

void Orkestra::enstrumanEkle(Enstruman* e)
{
    enstrumanlar[x] = e;
    x++;
}
void Orkestra::eserCal(string eserAdi)
{
    for(int i=0;i<x;i++){
        Enstruman* e = enstrumanlar[i];
        e->cal(eserAdi);
    }
}
int Orkestra::x = 0;
```

Çözümlü Sorular

Cevap…

```
// ---------------------- Uygulama.cpp ----------------------
#include <iostream>
#include "Orkestra.h"
#include "Keman.h"
#include "Piyano.h"
#include "Gitar.h"
using namespace std;
int main()
{
    Orkestra orkestra;
    Keman keman;
    Piyano piyano;
    Gitar gitar;
    orkestra.enstrumanEkle(&keman);
    orkestra.enstrumanEkle(&piyano);
    orkestra.enstrumanEkle(&gitar);
    orkestra.eserCal("Pavane");
    return 0;
}
```