BİL102 Nesne Yönelimli Programlama

Dr. Öğr. Üyesi Yavuz CANBAY

Kahramanmaraş Sütçü İmam Üniversitesi

Bilgisayar Mühendisliği Bölümü

Konu 6.

- Veri Soyutlaması (data abstraction)
- Veri Gizleme (data hiding)
- Veri Kapsülleme (data encapsulation)
- Çok biçimlilik (Polymorphism)

Veri soyutlama (data abstraction)

- Veri soyutlama, dış dünyaya sadece gerekli bilgileri sağlamayı ve arka plan ayrıntılarını gizlemeyi sağlar
- Yani ayrıntıları sunmadan programda gerekli bilgileri temsil etmeyi ifade eder.

- Veri soyutlama sınıf ve fonksiyonların içeriğine fazla bağımlı olmayan programlar yazılabilmesine imkan sağlar ve kullanım hatalarını önler.

Veri soyutlama - gerçek hayat örneği

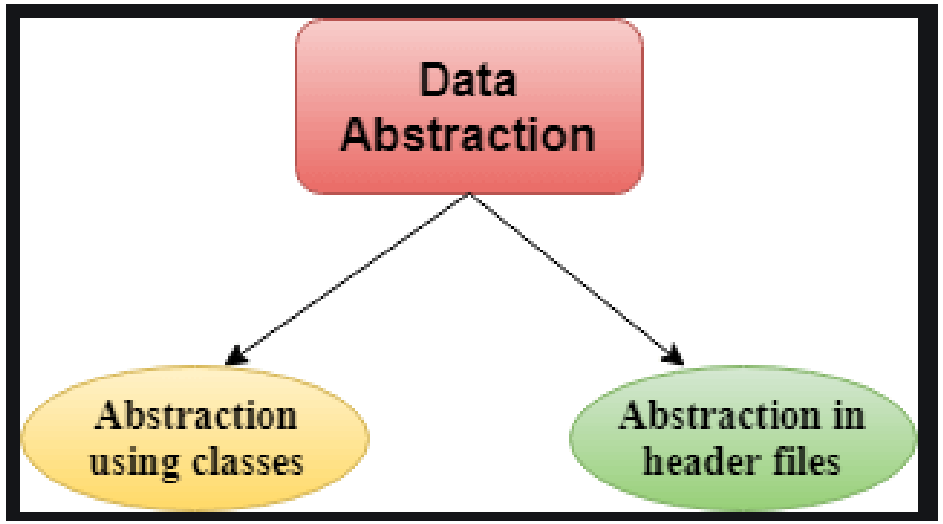
- Açıp kapatabileceğimiz bir dizüstü bilgisayar örneği alalım, interneti kullanabiliriz veya Filmler izleyebilir ve şarkıyı dinleyebiliriz.
- Ekstra klavye, USB Sürücü veya Ekstra Sabit disk gibi harici bileşenler ekleyebiliriz, müzik dinlerken sesi artırabilir ve azaltabiliriz.
- Ancak Dizüstü Bilgisayarın tam olarak nasıl çalıştığını bilmiyoruz,
- Bizi wifi veya DSL kullanarak İnternet'e nasıl bağladığını bilmiyoruz.
- Ve sadece bir kablo veya wifi üzerinden bir web sitesine nasıl eriştiğinizi bilmiyorsunuz.
- Bu tamamen Veri Soyutlama ile ilgilidir.

Veri soyutlama

- C ++ ile kendi soyut veri tiplerimizi (ADT) tanımlamak ve soyutlamayı sağlamak için **sınıf yapılarını** kullanıyoruz.
- Sınıf yapısının kullanılıyor olması veri soyutlama için bir örnektir.

Veri soyutlamanın faydaları

- Sınıf içerikleri, nesnenin durumunu değiştirebilecek kullanıcı seviyesi hatalardan korunurlar.
- Sınıf uygulaması, kullanıcı seviyesinde kodda herhangi bir değişikliğe gerek kalmadan zamanla değişen gereksinimlere bir cevap olarak gelişebilir ve değişebilir



Veri soyutlaması

› Public ve private üyelerle geliştirdiğiniz bir sınıfın ait olduğu bir C ++ programı veri soyutlamaya bir örnektir.

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

class Adder {
   public:
      // constructor
      Adder(int i = 0) {
         total = i;
      }

      // interface to outside world
      void addNum(int number) {
         total += number;
      }

      // interface to outside world
      int getTotal() {
         return total;
      };

   private:
      // hidden data from outside world
      int total;
};

int main() {
   Adder a;

   a.addNum(10);
   a.addNum(20);
   a.addNum(30);

   cout << "Total " << a.getTotal() <<endl;
   return 0;
}
```

Veri Gizleme

- Veri Gizleme, OOP'un gerçek hayattan esinlenen en önemli ilkelerinden biridir ve tüm bilgilerin herkes için erişilebilir olmaması gerektiğini söyler.
- Özel verilere yalnızca sahibinin erişebilmesi gerekir
- Veri gizleme, sınıf üyelerine kısıtlı veri erişimini garanti eder ve nesne bütünlüğünü korur

Veri gizlemeye hayattan örnekler

- İsmim ve kişisel bilgilerim beynimde saklanıyor, hiç kimse bu bilgilere doğrudan erişemiyor. Bu bilgiyi almak için bana sormanız gerekiyor ve sizinle ne kadar ayrıntı paylaşmak istediğim bana bağlı.
- Bir E-posta Sunucusu milyonlarca kişinin hesap bilgilerine sahip olabilir, ancak başka hesap bilgileri göndermesini istersem yalnızca hesap bilgilerimi benimle paylaşır, isteğim reddedilir.
- Facebook'un milyonlarca kullanıcı hesabı olabilir. Facebook tıpkı bir Sınıf gibidir,; private, public ve protected Üyeleri vardır. Private de, gelen kutusu, bazı kişisel Fotoğraf veya video olabilir. Publicde, bir gönderi veya doğum tarihi, yaşadığınız yer vb. gibi kullanıcı bilgileri olabilir ve Protected da yalnızca arkadaşınızın görebileceği ve herkesin gizlediği bir gönderi olabilir

Veri Gizleme

- Private, public and protected olmak üzere bir sınıf içinde üç tür koruma / erişim belirteci vardır.
- Genellikle, bir sınıftaki veriler private'dir ve fonksiyonlar ise public'tir
- Veriler gizlidir (private), böylece yanlışlıkla yapılan manipülasyondan korunur.

Veri Gizleme

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

    int num;  //by default private
    public:

    void read();
    void print();

};

void Base :: read(){
    cout<<"Enter any Integer value"<<endl; cin>>num;

}

void Base :: print(){
    cout<<"The value is "<<num<<endl;
}

int main(){
    Base obj;

    obj.read();
    obj.print();

    return 0;
}
```

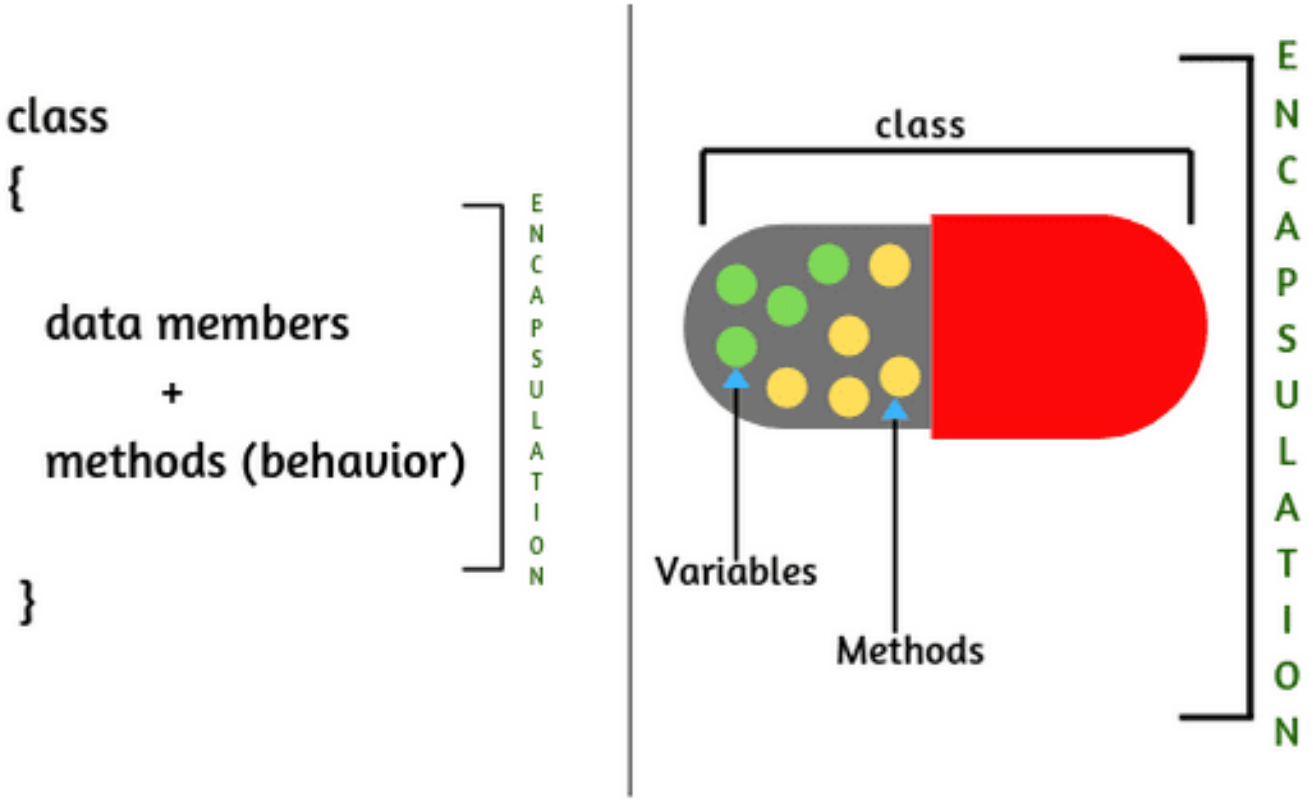
Veri Kapsülleme (encapsulation)

- Kapsülleme veya diğer adıyla veri kapsülleme işlemi, basitçe bir sınıfın (class) dışarıya karşı kapalı hale getirilmesidir.
- Bilindiği üzere, sınıflar, özelliklerden (properties) ve metotlardan (methods) oluşmaktadır.
- Veri kapsüllemesinin amacı, sınıfta bulunan özelliklerin erişimini kontrol altına almak ve sınıfın özelliklerinin dışarıdan erişimini engelleyerek, sınıftaki metotlar marifetiyle erişimi kontrol etmektir

Veri Kapsülleme (encapsulation)

- Sınıf özelliklerinizi private (olabildiğince sık) olarak beyan etmek iyi bir uygulama olarak kabul edilir.
- Kapsülleme verilerinizin daha iyi kontrol edilmesini sağlar, çünkü siz (veya diğerleri) kodun bir kısmını diğer kısımları etkilemeden değiştirebilirsiniz

Veri Kapsülleme



Kapsülleme vs Soyutlama

Abstraction	Encapsulation
1. Abstraction solves the problem in the design level.	1. Encapsulation solves the problem in the implementation level.
2. Abstraction is used for hiding the unwanted data and giving relevant data.	2. Encapsulation means hiding the code and data into a single unit to protect the data from outside world.
3. Abstraction lets you focus on what the object does instead of how it does it	3. Encapsulation means hiding the internal details or mechanics of how an object does something.
4. **Abstraction**- Outer layout, used in terms of design. For Example:- Outer Look of a Mobile Phone, like it has a display screen and keypad buttons to dial a number.	4. **Encapsulation**- Inner layout, used in terms of implementation. For Example:- Inner Implementation detail of a Mobile Phone, how keypad button and Display Screen are connect with each other using circuits.

Kapsülleme örneği

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

class Employee {
  private:
    // Private attribute
    int salary;

  public:
    // Setter
    void setSalary(int s) {
      salary = s;
    }
    // Getter
    int getSalary() {
      return salary;
    }
};

int main() {
  Employee myObj;
  myObj.setSalary(50000);
  cout << myObj.getSalary();
  return 0;
}
```

Çok Biçimlilik

- Polimorfizm "çoklu benzeşim veya çok biçimlilik" anlamına gelir
- Kalıtım yoluyla birbiriyle ilişkili birçok sınıfımız olduğunda ortaya çıkar.

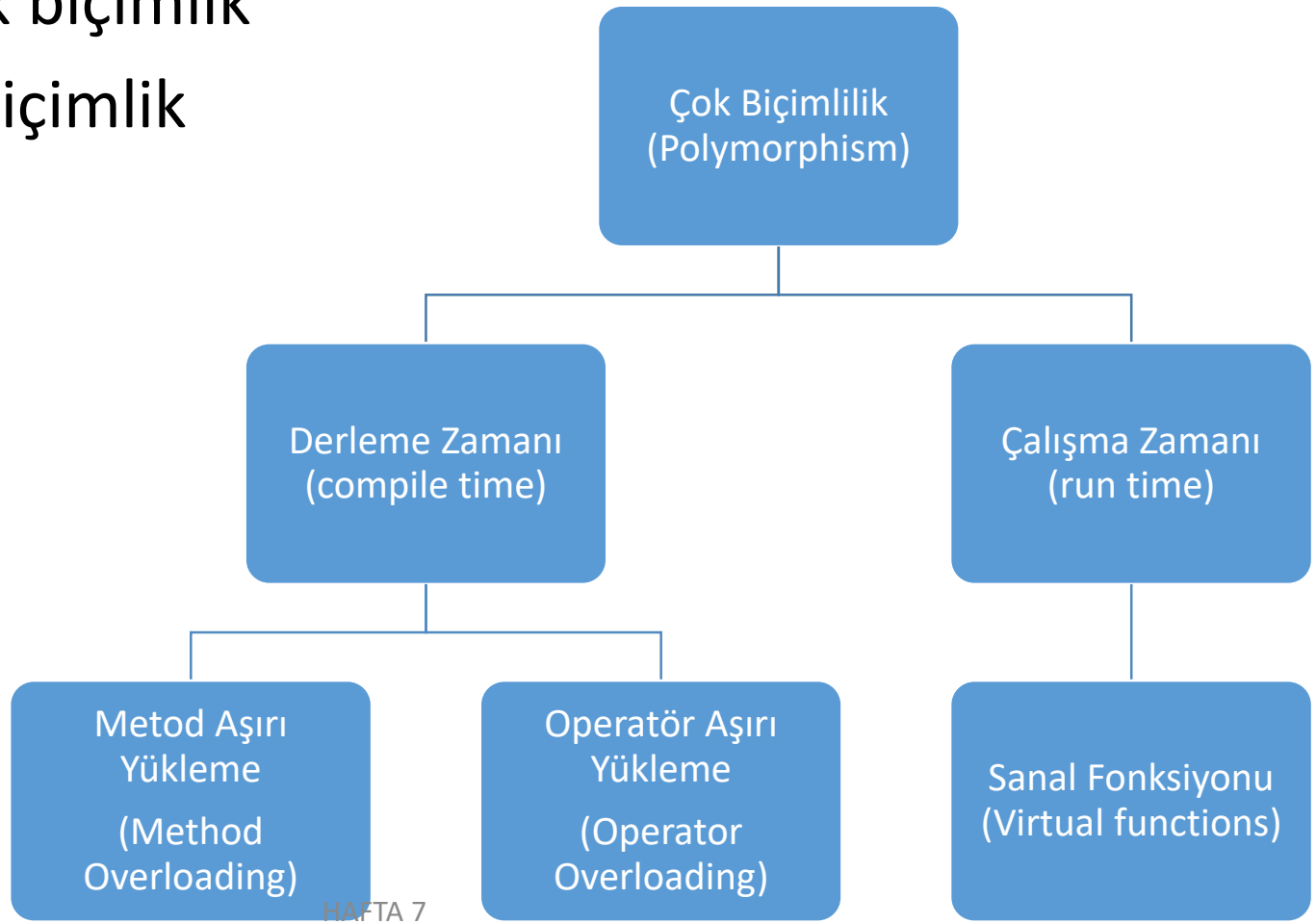
- Önceki bölümde belirttiğimiz gibi; Kalıtım, **nitelikleri** ve **yöntemleri** başka bir sınıftan devralmamızı sağlar.
- Çok biçimlilik ise farklı görevleri yerine getirmek için bu **yöntemleri** kullanır.
- Bu, tek bir işlemi farklı şekillerde gerçekleştirmemizi sağlar.

Çok Biçimlilik

› C++ da çok biçimlik temelde ikiye ayrılmıştır bunlar;

1-derlenme zamanı çok biçimlik

2-çalışma zamanı çok biçimlik



Derlenme zamanı çok biçimlik

- Bu çok biçimlik çeşidi, fonksiyon aşırı yükleme veya operatör aşırı yükleme olarak iki şekilde yapılabilir.
- Fonksiyon aşırı yükleme :
  - Aynı isime ancak farklı parametrelere sahip birden fazla fonksiyon olduğunda, bu fonksiyonların aşırı yüklendiği söylenir.
  - Fonksiyonlar, argüman sayısındaki değişiklik ve / veya argüman türünde değişiklik ile aşırı yüklenebilir.

Fonksiyon aşırı yükleme örneği

› Geeks sınıfında 3 aynı isimli ama farklı

Parametreli fonksiyonlara dikkat!!!

```
using namespace std;
class Geeks
{
    public:

    // function with 1 int parameter
    void func(int x)
    {
        cout << "value of x is " << x << endl;
    }

    // function with same name but 1 double parameter
    void func(double x)
    {
        cout << "value of x is " << x << endl;
    }

    // function with same name and 2 int parameters
    void func(int x, int y)
    {
        cout << "value of x and y is " << x << ", " << y << endl;
    }
};

int main() {

    Geeks obj1;

    // Which function is called will depend on the parameters passed
    // The first 'func' is called
    obj1.func(7);

    // The second 'func' is called
    obj1.func(9.132);

    // The third 'func' is called
    obj1.func(85,64);
    return 0;
}
```

Derlenme zamanı çok biçimlik

- Operatör aşırı yükleme :
  - C ++ ayrıca aşırı operatör yükleme seçeneği de sunar.
  - Örneğin, string birleştirme için (+) operatörünü kullanarak iki stringi birleştirebiliriz.
  - Aslında bu operatörün iki sayının toplamında kullanıldığını biliyoruz.
  - Dolayısıyla, tamsayılar arasına yerleştirildiğinde tek bir '+' operatörü iki tamsayıyı eklerken; stringler arasına yerleştirdiğimizde bu stringleri birleştirir.

Operatör aşırı yükleme örneği

```
// CPP program to illustrate
// Operator Overloading
#include<iostream>
using namespace std;

class Complex {
private:
    int real, imag;
public:
    Complex(int r = 0, int i =0)  {real = r;   imag = i;}

    // This is automatically called when '+' is used with
    // between two Complex objects
    Complex operator + (Complex const &obj) {
         Complex res;
         res.real = real + obj.real;
         res.imag = imag + obj.imag;
         return res;
    }
    void print() { cout << real << " + i" << imag << endl; }
};

int main()
{
    Complex c1(10, 5), c2(2, 4);
    Complex c3 = c1 + c2; // An example call to "operator+"
    c3.print();
}
```

Çalışma zamanı çok biçimlik

- Bu çok biçimlik çeşidi sadece sanal fonksiyon tekrar yazımı (virtual function overriding) ile yapılabilir.
- Fonksiyon tekrar yazımı:
  - Türetilmiş sınıf temel sınıfın üye fonksiyonlarından birine sahipse bu durumda gerçekleşir.
  - Bu durumda temel sınıf fonksiyonu override yani tekrar yazılmış olur.

Çalışma zamanı çok biçimlik örnek

- Hem base sınıfında hemde derived
- sınıfında aynı isimli print fonksiyonu
- olduğuna dikkat edin

- Kodda base class içerisindeki
- Virtual anahtar kelimesine dikkat edin

```
#include <bits/stdc++.h>
using namespace std;

class base
{
public:
    virtual void print ()
    { cout<< "print base class" <<endl; }

    void show ()
    { cout<< "show base class" <<endl; }
};

class derived:public base
{
public:
    void print () //print () is already virtual function in derived class
    { cout<< "print derived class" <<endl; }

    void show ()
    { cout<< "show derived class" <<endl; }
};

//main function
int main()
{
    base *bptr;
    derived d;
    bptr = &d;

    //virtual function, binded at runtime (Runtime polymorphism)
    bptr->print();

    // Non-virtual function, binded at compile time
    bptr->show();

    return 0;
}
```

Virtual Fonksiyon

- Virtual (sanal) fonksiyon; virtual anahtar kelimesi kullanılarak temel sınıf içerisinde bildirilen bir fonksiyondur

- Temel sınıfta bir virtual fonksiyon belirlemek, ki bu fonksiyon türetilmiş sınıfta da farklı bir versiyonla mevcuttur, bu fonksiyon için compiler yani derleyiciden statik bir bağlama (static linkage) istemediğimizi gösterir. (Not: statik bağlamada bir fonksiyon bir kere çalıştırılır ve tekrar çağrıldığında bir önceki içeriğini kopyalar olarak düşünebilirsiniz)

- Aslında istediğimiz, programın herhangi bir noktasında çağrılacak fonksiyonun, çağrıldığı nesne türüne göre seçilmesidir.

Virtual fonksiyonunun farkını anlamak için bir örnek

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

class Shape {
   protected:
      int width, height;

   public:
      Shape( int a = 0, int b = 0){
         width = a;
         height = b;
      }
      int area() {
         cout << "Parent class area :" <<endl;
         return 0;
      }
};
class Rectangle: public Shape {
   public:
      Rectangle( int a = 0, int b = 0):Shape(a, b) { }

      int area () {
         cout << "Rectangle class area :" <<endl;
         return (width * height);
      }
};

class Triangle: public Shape {
   public:
      Triangle( int a = 0, int b = 0):Shape(a, b) { }

      int area () {
         cout << "Triangle class area :" <<endl;
         return (width * height / 2);
      }
};
```

```
// Main function for the program
int main() {
   Shape *shape;
   Rectangle rec(10,7);
   Triangle  tri(10,5);

   // store the address of Rectangle
   shape = &rec;

   // call rectangle area.
   shape->area();

   // store the address of Triangle
   shape = &tri;

   // call triangle area.
   shape->area();

   return 0;
}
```

Bu kodun çıktısı:

```
Parent class area :
Parent class area :
```

- Burada Shape den türetilen Rect ve Tria sınıfları vardır.
- Bu iki sınıftan türetilen nesnelerin alanları area() fonksiyonu ile çağırılmaktadır.
- Ama istenen çıktı ekrana basılmamıştır.
- Çünkü area() fonksiyonu ilk olarak temel sınıfda derlendiği için compiler artık bunu derlemez ve bu duruma önceki sayfada yer alan static linkage denilmektedir.

Virtual fonksiyonunun farkını anlamak için bir örnek-devam

```
class Shape {
   protected:
      int width, height;

   public:
      Shape( int a = 0, int b = 0) {
         width = a;
         height = b;
      }
      virtual int area() {
         cout << "Parent class area :" <<endl;
         return 0;
      }
};
```

Bu durumda kodun çıktısı:

```
Rectangle class area
Triangle class area
```

› Bir önceki kodda sadece soldaki kısımda ilgili alana virtual yazılmıştır.

› Bir önceki sunumdaki hatayı gidermek için Virtual anahtar kelimesinin kullandığına dikkat ediniz. Bu şekilde dynamic linkage yapılır ve her area() fonksiyonu tekrar derlenir.

Çok Biçimlilik--ÖRNEK

```
// Base class
class Animal {
  public:
    void animalSound() {
    cout << "The animal makes a sound \n" ;
  }
};

// Derived class
class Pig : public Animal {
  public:
    void animalSound() {
    cout << "The pig says: wee wee \n" ;
  }
};

// Derived class
class Dog : public Animal {
  public:
    void animalSound() {
    cout << "The dog says: bow wow \n" ;
  }
};
```

Çok Biçimlilik-ÖRNEK(devam)

```
// Base class
class Animal {
  public:
    void animalSound() {
    cout << "The animal makes a sound \n" ;
  }
};

// Derived class
class Pig : public Animal {
  public:
    void animalSound() {
    cout << "The pig says: wee wee \n" ;
   }
};

// Derived class
class Dog : public Animal {
  public:
    void animalSound() {
    cout << "The dog says: bow wow \n" ;
  }
};

int main() {
  Animal myAnimal;
  Pig myPig;
  Dog myDog;

  myAnimal.animalSound();
  myPig.animalSound();
  myDog.animalSound();
  return 0;
}
```