

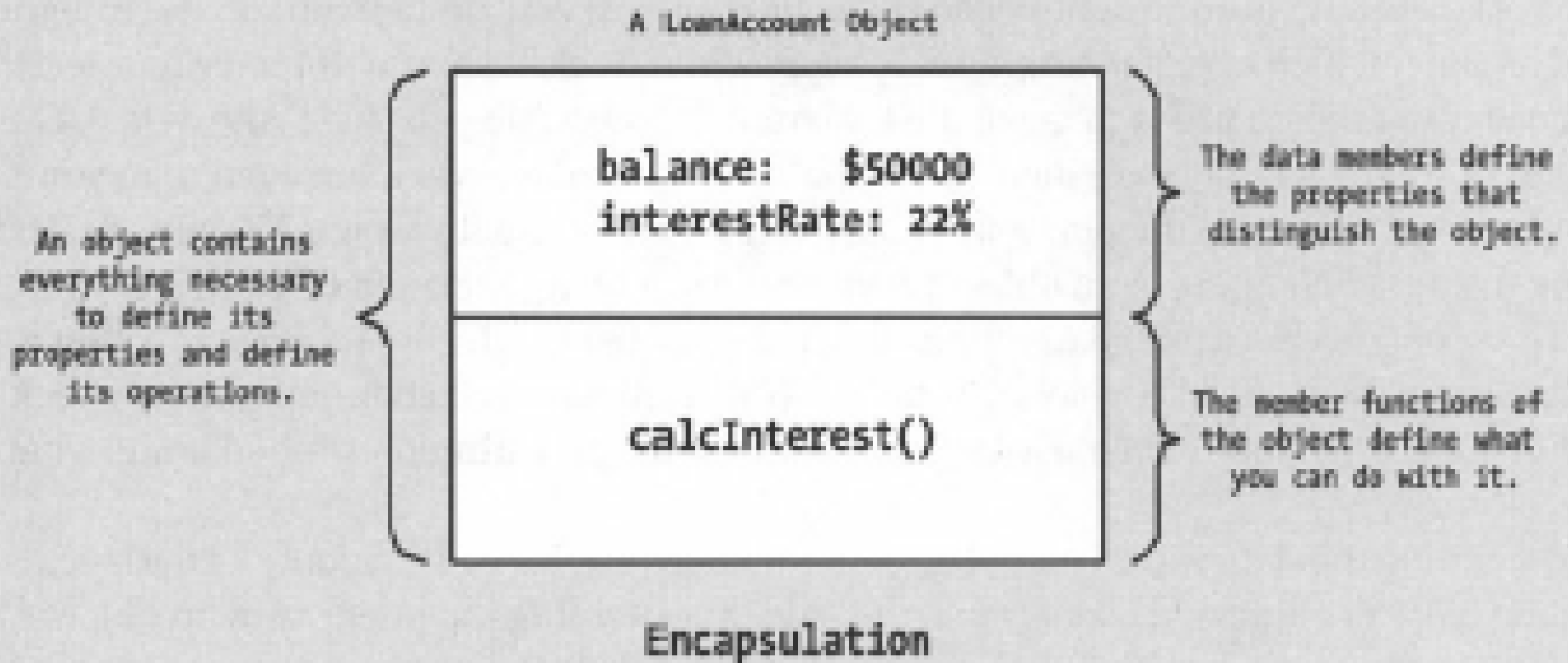
Classes: Defining Your Own Data Types

- Basic principles in OOP
- Define a new data type as a class and use objects of a class
- Member Functions
 - Constructors
 - Default constructors
 - Copy constructor
- Friend class/function
- The 'this' pointer

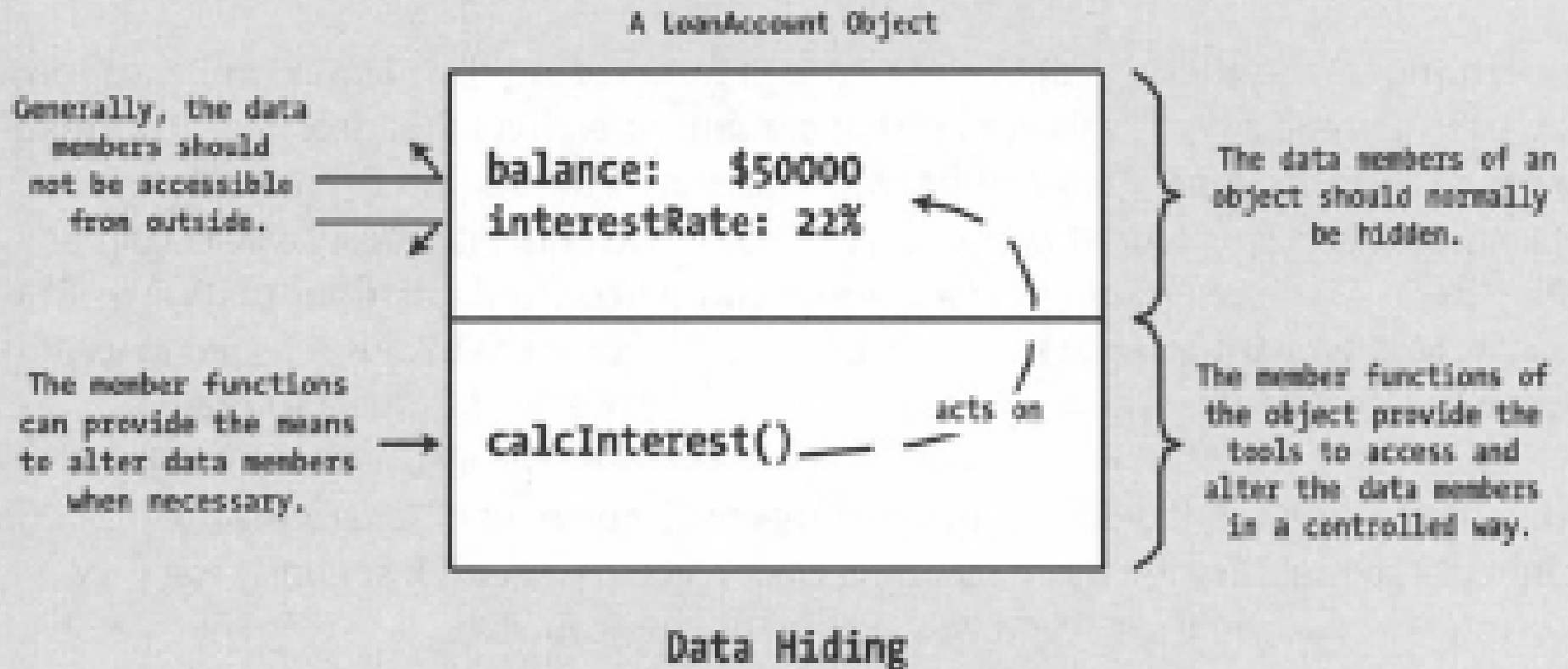
Object-Oriented Programming

- Encapsulation
- Data hiding
 - Data Member \Rightarrow state, feature
 - Member function \Rightarrow interface, method
- Inheritance
 - Base class
 - Derived class
- Polymorphism

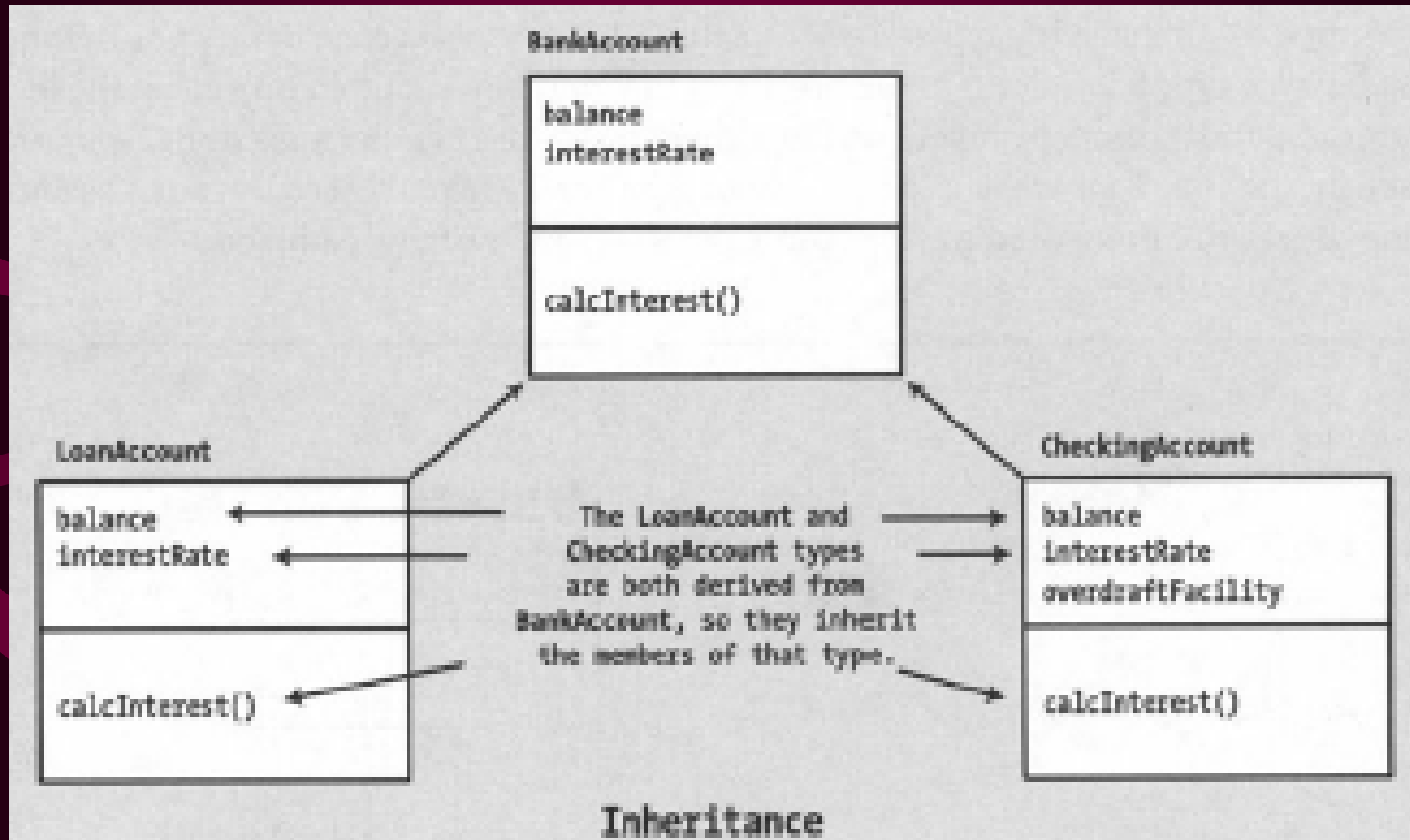
Encapsulation



Data Hiding

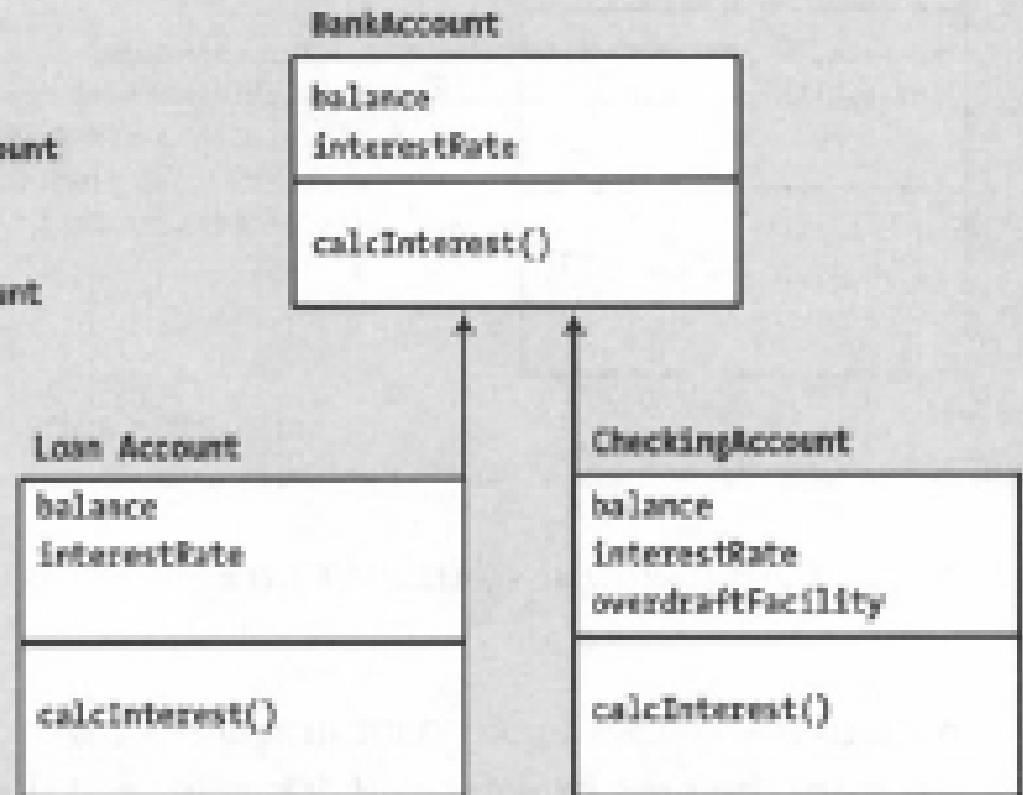


Inheritance



Polymorphism

```
BankAccount* pAcc;  
LoanAccount debt;  
CheckingAccount cash;  
  
pAcc = &cash;  
pAcc->calcInterest(); // Credits the account  
  
pAcc = &debt;  
pAcc->calcInterest(); // Debits the account
```



Polymorphism

Terminology

- A **class** is a user-defined data type
- Variables of a class are called **data members**, and the functions are called **member functions** which is referred to **methods**
- One declares variables of the class type also called **instances** of the class. Each instance is an **object** of the class.

Terminology

- Defining an instance of a class is sometimes referred to as **instantiation**
- Object-oriented programming involves the ideas of **encapsulation** of data, **polymorphism**

Defining a Class

- Access specifier
 - private
 - public
 - protected
- class and struct
- Program 12.1

Constructors

- Memory space generating
- Program 12.2
- Rewrite program 12.2a
 - `box.h`, `box.cpp`, `use_box.cpp`

Default Constructor

- If you don't define a constructor for your class, then the compiler supplies a **default constructor** that is used to create objects of your class

```
int main()
{
    Box smallBox(80.0, 50.0, 40.0);
    Box mediaBox;
    // compiler error, constructor already specified
    ...
}
```

- Program 12.3

Default Value

```
class Box
{
    ...
    Box();
    Box(double length = 1.0,
        double breadth = 1.0, double height = 1.0);
};

int main()
{
    Box smallBox; // Compiler error, ambiguity
    ... }
}
```

Default Initialization Values Using **initializer list**

// Constructor Definition

```
Box::Box(double la, double lb, double h):  
    length(la), breadth(lb), height(h)  
{  
    cout << "Box Constructor called" << endl;  
}
```

The explicit Keyword

- Prevent automatically data type cast
- Program 12.4a

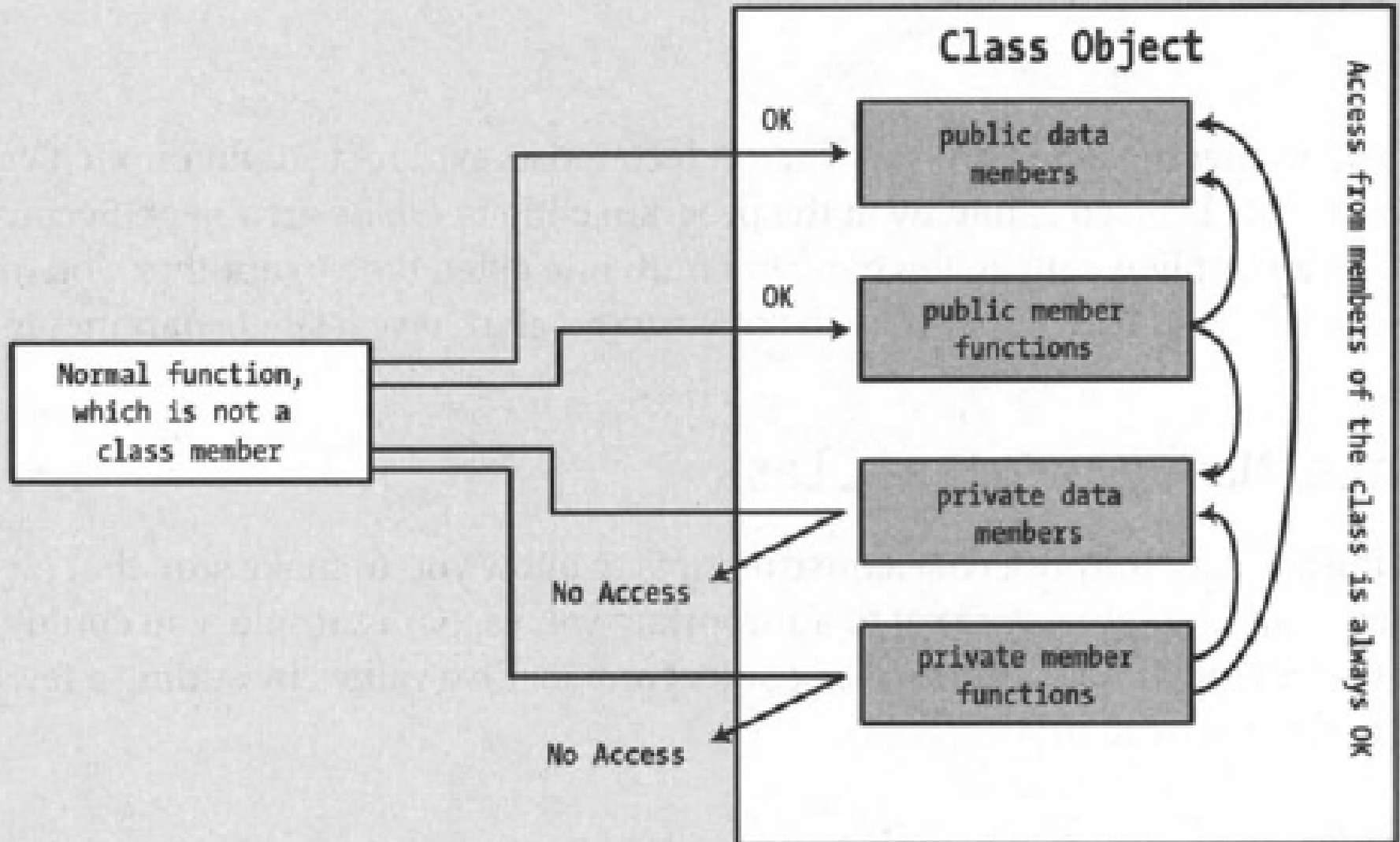
Private Members of a Class

- Proper initialized list
- Prevent improper modification

```
Box theBox(10.0, 10.0, 5.0);  
theBox.length = -20.0;
```

- Private members are not accessible
- Program 12.4

Private Members



Accessing Private Class Members

- The private members can be accessed by public member functions, **accessor**
- Program 12.5a

The Default Copy Constructor

- If you don't define copy constructor, the compiler supplies a default constructor that allows an object to be created
- Copy constructor: create a new instance by copying the exist one
- Program 12.5

Friends

- Friend classes – a whole class can be specified as a friend of a class
- Friend functions – an individual function can be specified as a friend of a class

The Friend Functions of a Class

- A function that is not a member of a class but nonetheless can access all its members is called a **friend function**
- Friend function is declared in one class
- Friend function can be declare as a global function or a member of a class
- Program 12.6

The Pointer Named 'this'

// usually 'this' pointer is hidden

```
double Box::volume()
```

```
{ return length * breadth * height; }
```

// Same as

```
double Box::volume()
```

```
{ return this->length * this->breadth *  
  this->height; }
```

- Program 12.7

const Objects and const Member Functions

```
class Box
{
int compareVolume(const Box& otherBox) const;
}

// in main()
Box theBox(17.0, 11.0, 4.4), otherBox(8.0, 4.0, 20.0);
if (theBox.compareVolume(otherBox))
{ ... }
```

- Program 12.8a

Mutable Data Members of a Class

- If you declare an object as `const`, you can't change the values of the data members of the object because they'll also be effectively `const`. However, you may find situations in which you need to allow certain, selected data members of a class to be altered, even if the object was declared as `const`
- How?
 - By declaring a data member as **mutable**

Mutable Data Members of a Class

```
class SecureAccess
{
public:  ...
    bool isLocked() const;
private: ...
    mutable int time;
};

bool SecureAccess::isLocked() const // definition
{ time = getCurrentTime(); return lockStatus(); }

const SecureAccess mainDoor; // in main()
bool doorState = mainDoor.isLocked(); // calling
```


Casting Away const

- Very rare situations to use it
- A function is dealing with a const object, either passed as an argument of the object pointed to by this, and it is necessary to make it non-const
- Make const type to non-const, the expression must be a const Type
`const_cast<Type>(expression)`

Arrays of Objects of a Class

- Construct every element in an object array by calling default constructor
- Program 12.8

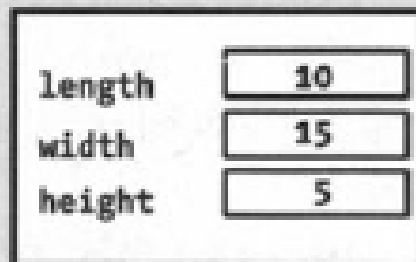
Boundary Alignment

- For the reasons of efficiency, a 2-byte and a 4-byte variables must be placed at the addresses that is a multiple of two, and four respectively.
- The size needed for variable is larger than it truly used
- Program 12.9

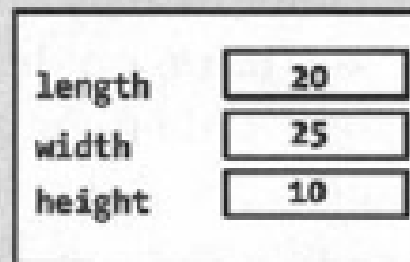
Static Data Members of a Class

```
class Box {  
    private:  
        static int objectCount;  
        double length;  
        double width;  
        double height;  
        ...  
}
```

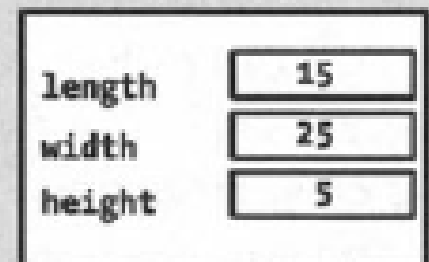
boxObject1



boxObject2



boxObject3



One copy of each static data member is shared
between all objects of a class.

objectCount

Defining a static Member

```
class Box
{
private:
    static int objectCount;
    double length;
    double breadth;
    double height;
};
// should be define in global region
int Box::objectCount = 0;
```

- Program 12.10

Access Static Data Members

- Access the static variable in a class
- Program 12.10a (in case doesn't exist, the following two snaps can create 12.10a)

A Static Data Member of the Same Type As the Class

```
class Box {
public:
    // Constructors
    Box();
    Box(double lengthValue, double widthValue, double heightValue);

    // Function to calculate the volume of a box
    double volume() const;

    // Function to compare two Box objects
    int compareVolume(const Box& otherBox) const;

private:
    const static Box refBox;           // Standard reference box
    double length;
    double width;
    double height;
};
```

```
class Box {
public:
    // Constructors
    Box();
    Box(double lengthValue, double widthValue, double heightValue);

    // Function to calculate the volume of a box
    double volume() const;

    // Function to compare two Box objects
    int compareVolume(const Box& otherBox) const;

    static int getObjectCount() {return objectCount;}

private:
    static int objectCount;           // Count of objects in exist
    double length;
    double width;
    double height;
};
```



```
const Box Box::refBox(10.0, 10.0, 10.0);
```

- A static member function is a full member of the class in terms of access privileges

```
static double sum(Box theBox) {  
    return theBox.length + theBox.width + theBox.height;  
}
```