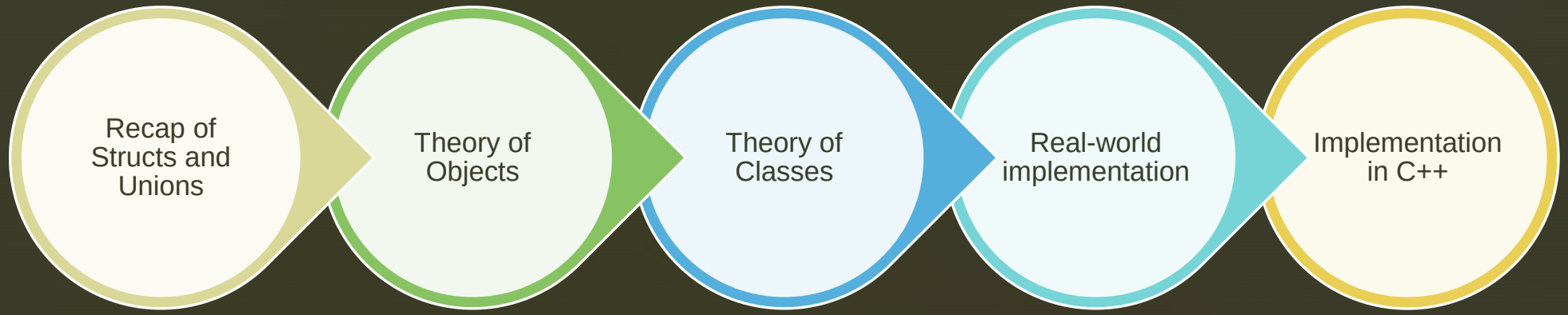
DR. (MRS) T-S.M.A. ADJAIDOO

05 Classes and Objects

Today's Lesson



Learning Outcomes

1. To understand the need of classes and objects
2. To properly understand how to capture real-world situations as classes and objects
3. To create classes and objects in Python Programming language

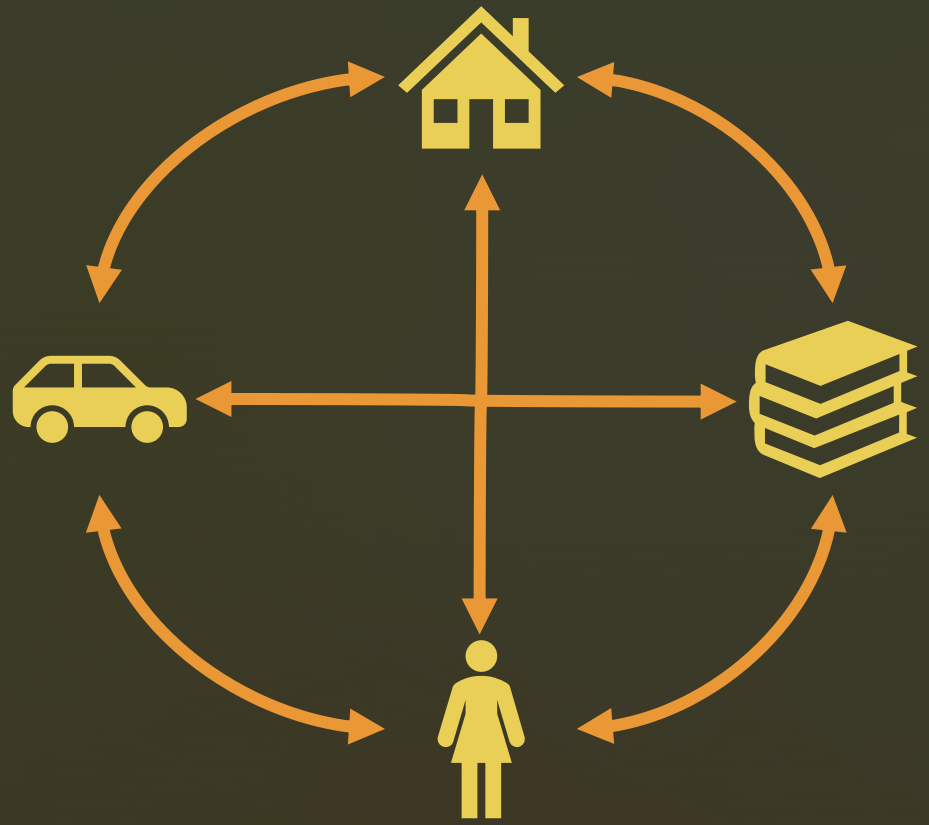
Recap of structs and unions

- Recall your previous lessons in C Programming
  - What are structures?
  - When are structures needed?
  - What are unions?
  - When are unions needed?
- What are the limitations of structures and unions?

This Photo by Unknown Author is licensed under CC BY-NC

Theory of Objects

- What are OBJECTS?
- Objects are the building blocks of an OO program.
- A program that uses OO technology is basically a collection of objects messaging each other
- Each object consists of some data and behaviour



Theory of Objects

Object Data

- The data stored within an object represents the state of the object.
- In OO programming terminology, this data is called attributes
- Let's illustrate using a school system:
  - Objects that represent students
    - Attributes such as name, index number, date of birth, gender etc.
- The attributes contain the information that differentiates between the various objects

Theory of Objects

Object Data: Examples of student attributes

Name: Felix Arthur
Gender: Male
Date of Birth: 26th October 2001
Student No: 123456

Name: Kwadjo Baako
Gender: Male
Date of Birth: 15th April 2002
Student No: 678910

Name: Leslie Osman
Gender: Female
Date of Birth: 12th March 2000
Student No: 111213

Name: Phoebe Ansah
Gender: Female
Date of Birth: 1st July 2001
Student No: 141516

Theory of Objects

Object Behaviours

- The behaviour of an object represents what the object can do.
- In procedural languages, the behaviour is defined by procedures, functions, and subroutines.
- In OO programming terminology, these behaviours are contained in methods, and you invoke a method by sending a message to it.

Theory of Objects

Object Behaviours

- In our student example, consider that one of the behaviours required of a student object is to set and return the values of the various attributes.
- Therefore, each attribute would have corresponding methods, such as setGender() and getGender() etc.
- These are often called getters and setters in programming languages

Student Object

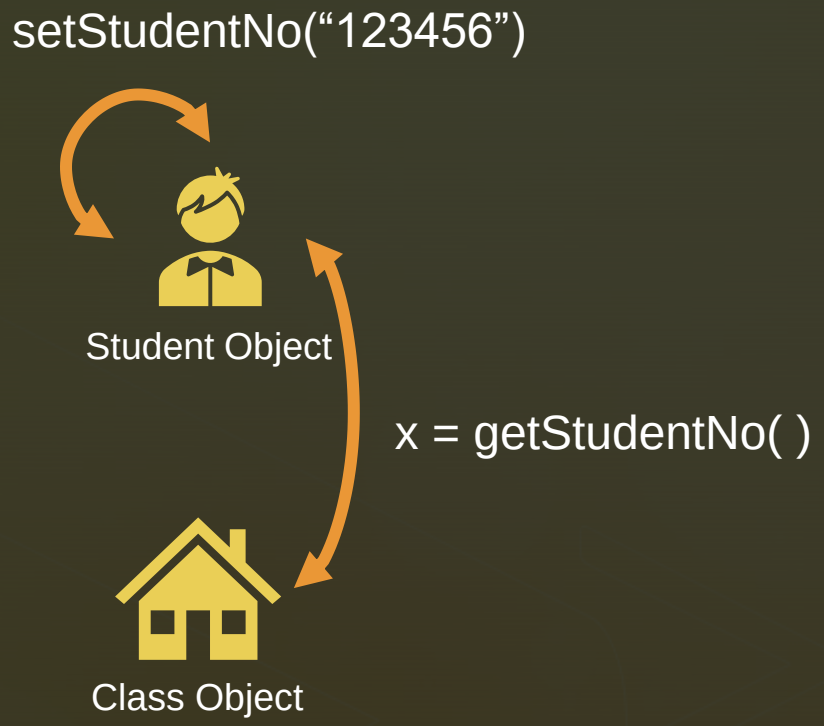
Attributes
Name: Felix Arthur
Gender: Male
DateofBirth: 26th October 2001
StudentNo: 123456

# Behaviours
setName(name)
getName()
setGender(gender)
getGender()
setDateofBirth(dob)
getDateofBirth()
setStudentNo(studentno)
getStudentNo()

Theory of Objects

Object Behaviours

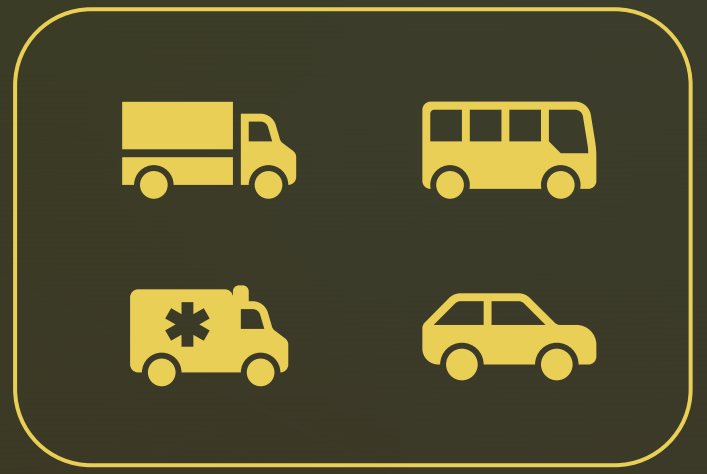
- The following information is all a user needs to know to effectively use a methods:
  - The name of the method
  - The parameters passed to the method
  - The return type of the method



Theory of Classes

- What are classes?
- A class is a blueprint for an object.
- When you instantiate an object, you use a class as the template for how the object is built.
- An object cannot be instantiated without a class

vehicles



Tyre count, window count, body shape, body colour, Engine type etc

Theory of Classes

- Let us use an example from the relational database world.
- In a database table, the definition of the table itself (fields, description, and data types used) would be a class (metadata), and the objects would be the rows of the table (data).

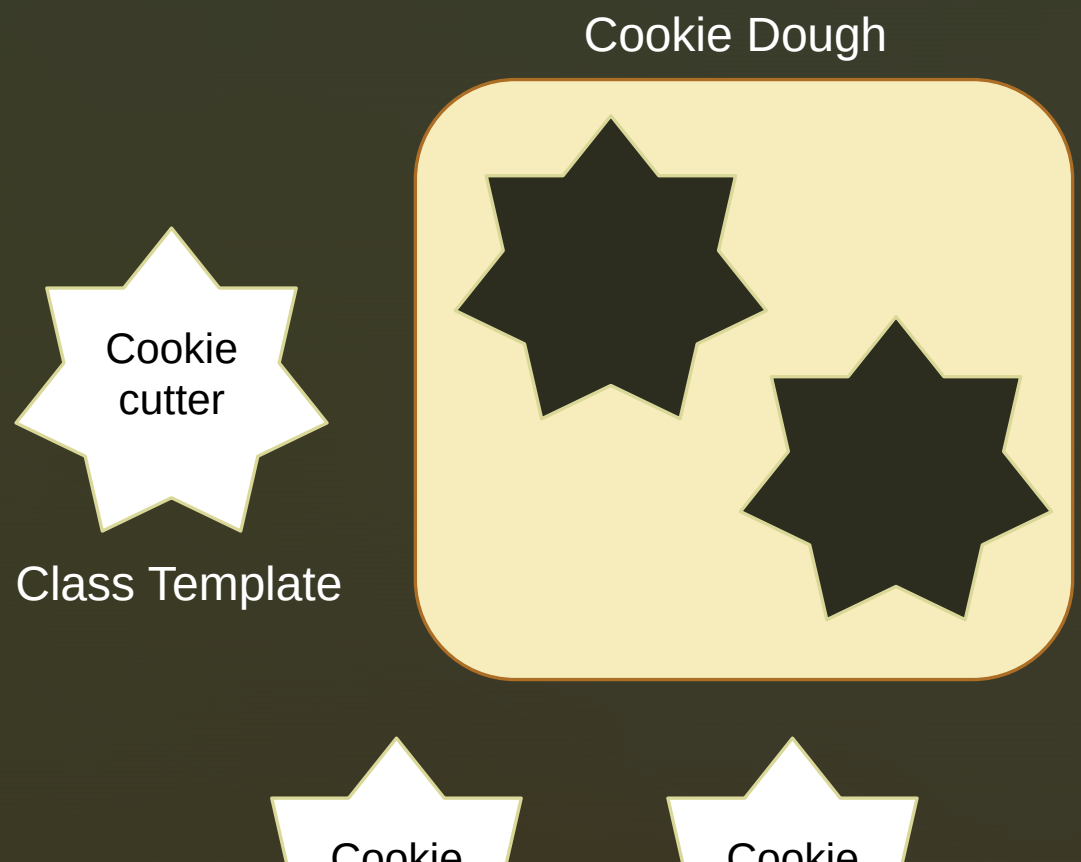
	Student No	Name	Gender	Date of Birth
Object 1	123456	Felix Arthur	Male	26th October 2001
Object 2	678910	Kwadjo Baako	Male	15th April 2002
Object 3	111213	Leslie Osman	Female	12th March 2000
Object 4	141516	Phoebe Ansah	Female	1st July 2001

Student Class

Theory of Classes

Creating Objects

- Classes can be thought of as the templates, or cookie cutters, for objects.
- A class is used to create an object.



Theory of Classes

- A class can be thought of as a sort of higher-level data type.
  For example, just as we create an integer or a float:
  - x = 5
  - y = 10.5
- We can also create an object by using a predefined class:
  - myObject = MyClass( )
- Remember that each object has its own attributes (data) and behaviours (functions or routines).
- A class defines the attributes and behaviours that all objects created with this class will possess.

Real-world Implementation

Identifying classes in the real world

- Classes are to be a grouping of conceptually-related state (attributes/features) and behaviour
- One popular way to group them is using grammar
  - Noun → Object
  - Adjective → Attribute
  - Verb → Method

- For example:
- "A multiple-choice quiz program that asks questions and checks the answers are correct"

Real-world Implementation

Try this on your own:

- Identify the possible objects, methods and attributes in this scenario:
  - A digital library management program which checks-in books, displays them for users to borrow and also checks-out books

C++ Implementation

```
// demonstrates a small, simple object
#include <iostream>
using namespace std;
    class smallobj //define a class
    {
        private:
            int somedata; //class data
        public:
            void setdata(int d) //member function to set data
                { somedata = d; }
            void showdata() //member function to display data
                { cout << "Data is "<< somedata << endl; }
    };
```

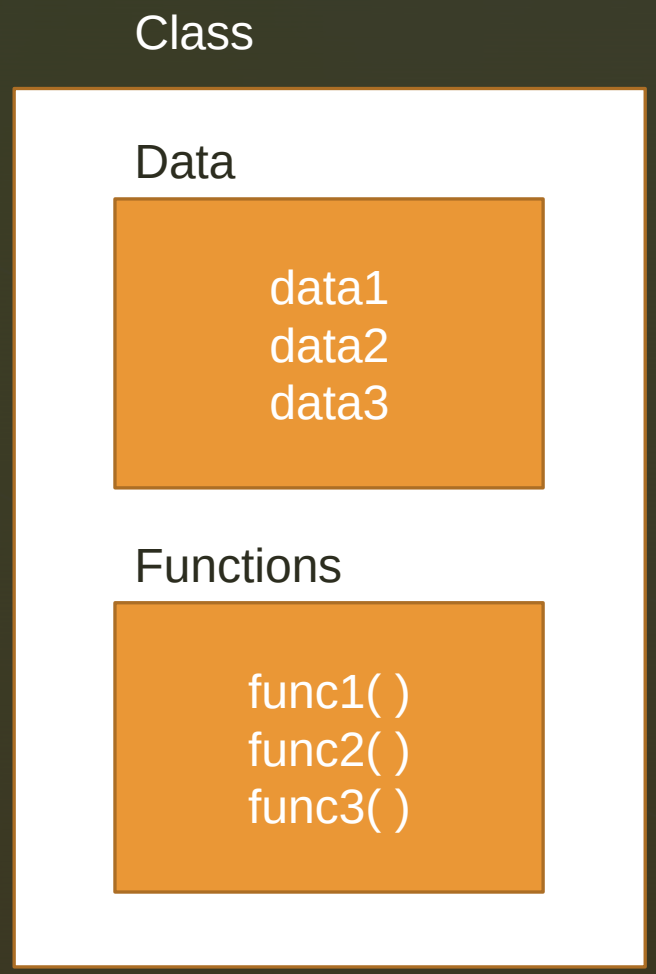
C++ Implementation

```
int main()
    {
        smallobj s1, s2; //define two objects of class smallobj
        s1.setdata(23); //call member function to set data
        s2.setdata(12);
        s1.showdata(); //call member function to display data
        s2.showdata();
        return 0;
}
```

C++ Implementation

- The class smallobj defined in this program contains one data item and two member functions.
- The two member functions provide the only access to the data item from outside the class.
- The first member function sets the data item to a value, and the second displays the value.
- Placing data and functions together into a single entity is a central idea in object-oriented programming.

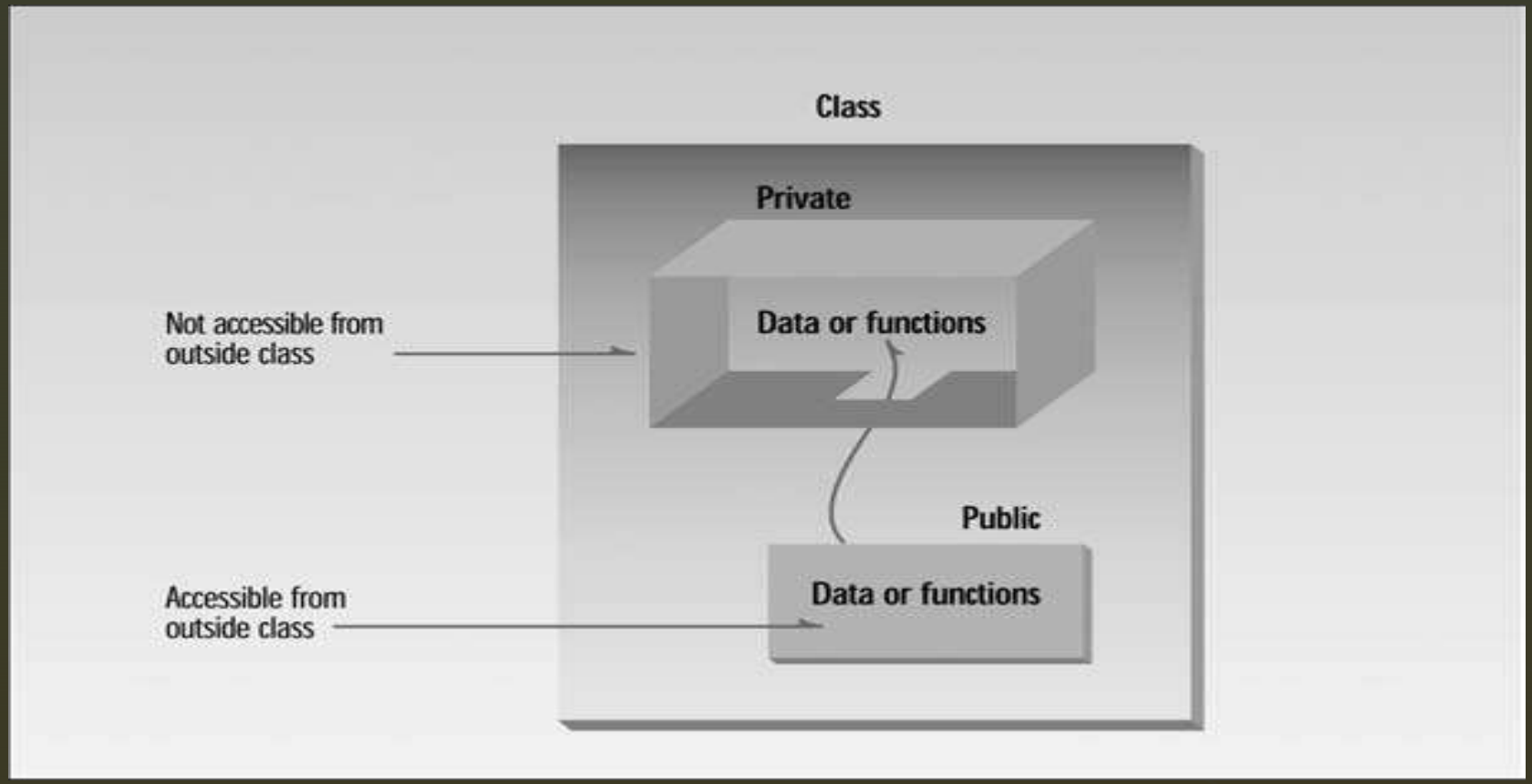
C++ Implementation



Defining the Class

- The definition starts with the keyword class, followed by the class name.
- The body of the class is delimited by braces and terminated by a semicolon.
- The body of the class contains two keywords: private and public.
- Private data or functions can only be accessed from within the class.
- Public data or functions are accessible from outside the class.

C++ Implementation



C++ Implementation

- The data items within a class are called data members (or sometimes member data).
- There can be any number of data members in a class.
- Member functions are functions that are included within a class.

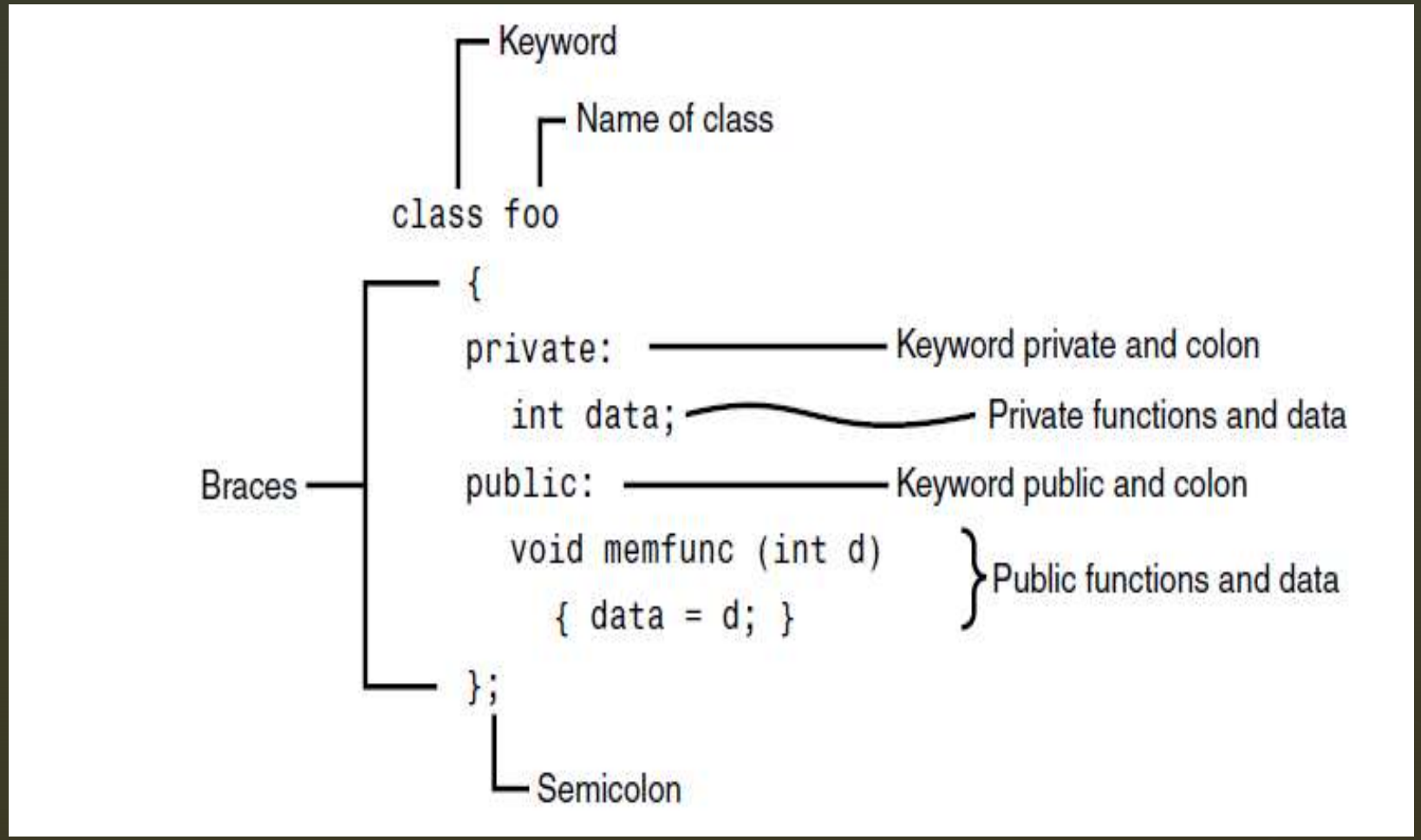
C++ Implementation

- A key feature of object-oriented programming is *data hiding.*
- It means that data is concealed within a class so that it cannot be accessed mistakenly by functions outside the class.
- Data hiding, on the other hand, means hiding data from parts of the program that don’t need to access it.
- One class’s data is hidden from other classes.
- Data hiding is designed to protect well-intentioned programmers from honest mistakes.

C++ Implementation

- Usually the data within a class is private and the functions are public.
- The data is hidden so it will be safe from accidental manipulation, while the functions that operate on the data are public so they can be accessed from outside the class.
- However, there is no rule that says data must be private and functions public; in some circumstances you may find you'll need to use private functions and public data.

Syntax of a class definition



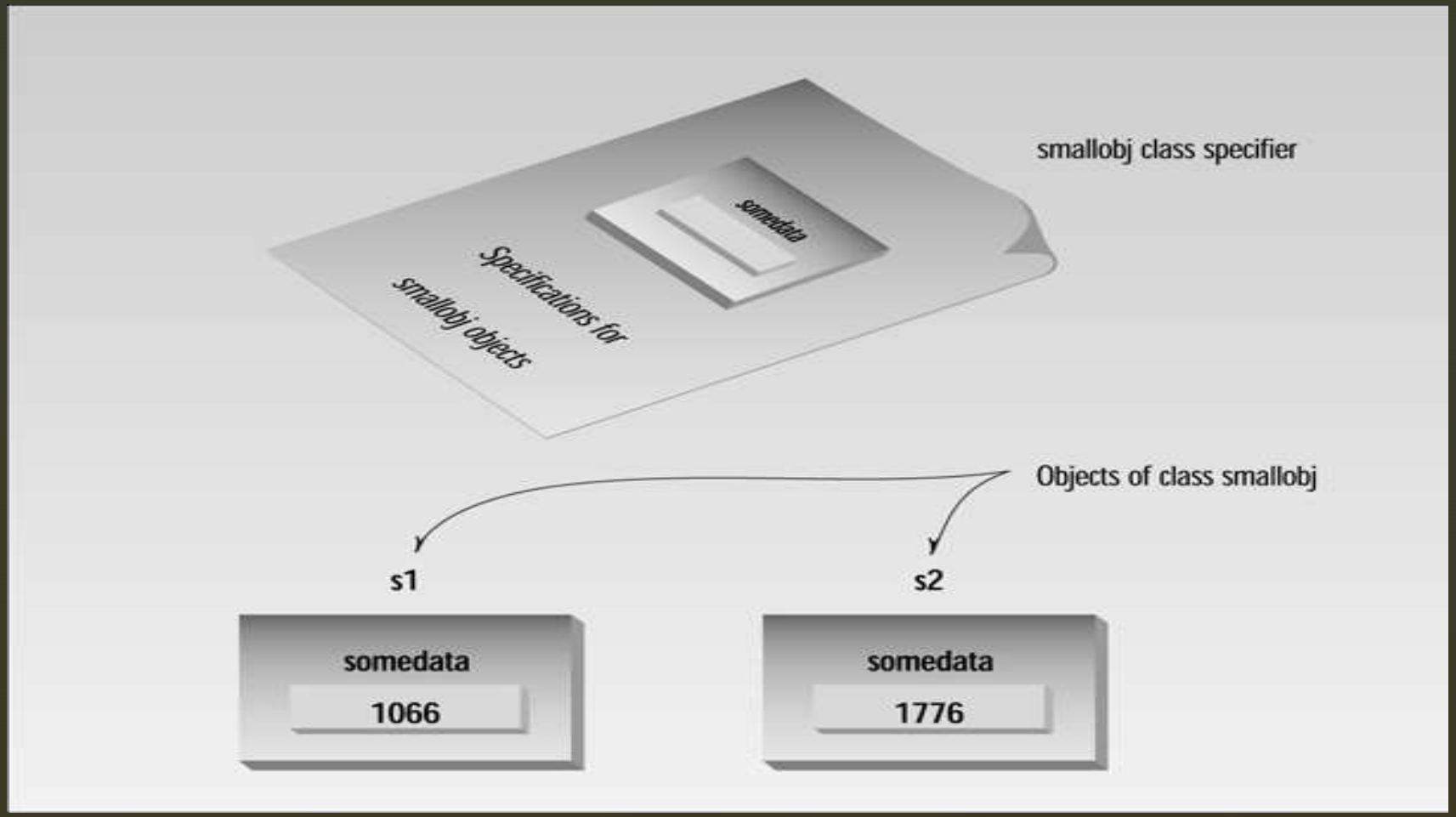
Defining Objects

- Defining an object is similar to defining a variable of any data type: Space is set aside for it in memory.
- Defining objects in this way means creating them. This is also called instantiating them.
- The term instantiating arises because an instance of the class is created.
- An object is an instance (that is, a specific example) of a class. Objects are sometimes called instance variables.
- The first statement in main() smallobj s1, s2; defines two objects called s1 and s2, of class smallobj.

Calling Member Functions

- To use a member function, the dot operator connects the object name and the member function.
- The dot operator is also called the *class member access operator.*
- The first call to setdata(), s1.setdata(23); executes the setdata() member function of the s1 object.
- This function sets the variable somedata in object s1 to the value 23.
- The second call s2.setdata(12); causes the variable somedata in s2 to be set to 12.

C++ Implementation



Messages

- Some object-oriented languages refer to calls to member functions as *messages.*
- Thus the call

  s1.showdata();

  can be thought of as sending a message to s1 telling it to show its data.

C++ Implementation

```
// widget part as an object
#include <iostream>
using namespace std;

class part //define class
    {
    private:
        int modelnumber; //ID number of widget
        int partnumber; //ID number of widget part
        float cost; //cost of part
    public:
        void setpart(int mn, int pn, float c) //set data
        {
            modelnumber = mn;
            partnumber = pn;
            cost = c;
        }
    void showpart() //display data
        {
            cout << "Model " << modelnumber;
            cout << ", part " << partnumber;
            cout << ", costs GHC" << cost << endl;
        }
    };
```

C++ Implementation

```
int main()
{
    part part1; //define object  of class part
    part1.setpart(6244, 373, 217.55F); //call member function
    part1.showpart(); //call member function
    return 0;
}
```

Constructors

- Automatic initialization is carried out using a special member function called a *constructor.*
- *A constructor is a* member function that is executed automatically whenever an object is created.

C++ Implementation

```
// object represents a counter variable
#include <iostream>
using namespace std;
    class Counter
    {
        private:
            unsigned int count; //count
        public:
            Counter() //constructor
                { count = 0; }
            void inc_count() //increment count
                { count++; }
            int get_count() //return count
                { return count; }
    };
```

C++ Implementation

```
int main()
{
    Counter c1, c2; //define and initialize
    cout << "\nc1=" << c1.get_count(); //display
    cout << "\nc2=" << c2.get_count();

    c1.inc_count(); //increment c1
    c2.inc_count(); //increment c2
    c2.inc_count(); //increment c2

    cout << "\nc1=" << c1.get_count(); //display again
    cout << "\nc2=" << c2.get_count();
    cout << endl;
    return 0;
}
```

Destructors

- A destructor has the same name as the constructor (which is the same as the class name) but is preceded by a tilde.
- It is called automatically when an object is destroyed.
- Like constructors, destructors do not have a return value.
- They also take no arguments (the assumption being that there's only one way to destroy an object).
- The most common use of destructors is to de-allocate memory that was allocated for the object by the constructor.

C++ Implementation

```
class Foo
   {
   private:
      int data;
   public:
      Foo() : data(0)       //constructor (same name as class)
         {  }
      ~Foo()                //destructor (same name with tilde)
         {  }
   };
```

Objects as Function Arguments

```
#include <iostream>
using namespace std;
    class Distance //English Distance class
    {
        private:
            int feet;
            float inches;
        public:
            //constructor (no args)
            Distance() : feet(0), inches(0.0)
            { }
            //constructor (two args)
            Distance(int ft, float in) : feet(ft), inches(in)
            { }
            void getdist() //get length from user
            {
                cout << "\nEnter feet: "; cin >> feet;
                cout << "Enter inches: "; cin >> inches;
            }
            void showdist() //display distance
            { cout << feet << "\'-" << inches << '\"'; }
            void add_dist( Distance, Distance ); //declaration
    };
```

C++ Implementation

Distance Function

```
117    //add lengths d2 and d3
118    void Distance::add_dist(Distance d2, Distance d3)
119    {
120        inches = d2.inches + d3.inches; //add the inches
121        feet = 0; //(for possible carry)
122        if(inches >= 12.0) //if total exceeds 12.0,
123        {
124            //then decrease inches
125            inches -= 12.0; //by 12.0 and
126            feet++; //increase feet by 1
127        }
128        feet += d2.feet + d3.feet; //add the feet
129    }
```

C++ Implementation

Main Function

```
130   int main()
131   {
132       Distance dist1, dist3; //define two lengths
133       Distance dist2(11, 6.25); //define and initialize dist2
134
135       dist1.getdist(); //get dist1 from user
136
137       //dist3 = dist1 + dist2  //display all lengths
138       dist3.add_dist(dist1, dist2);
139
140       cout << "\ndist1 = "; dist1.showdist();
141       cout << "\ndist2 = "; dist2.showdist();
142       cout << "\ndist3 = "; dist3.showdist();
143       cout << endl;
144       return 0;
145   }
```

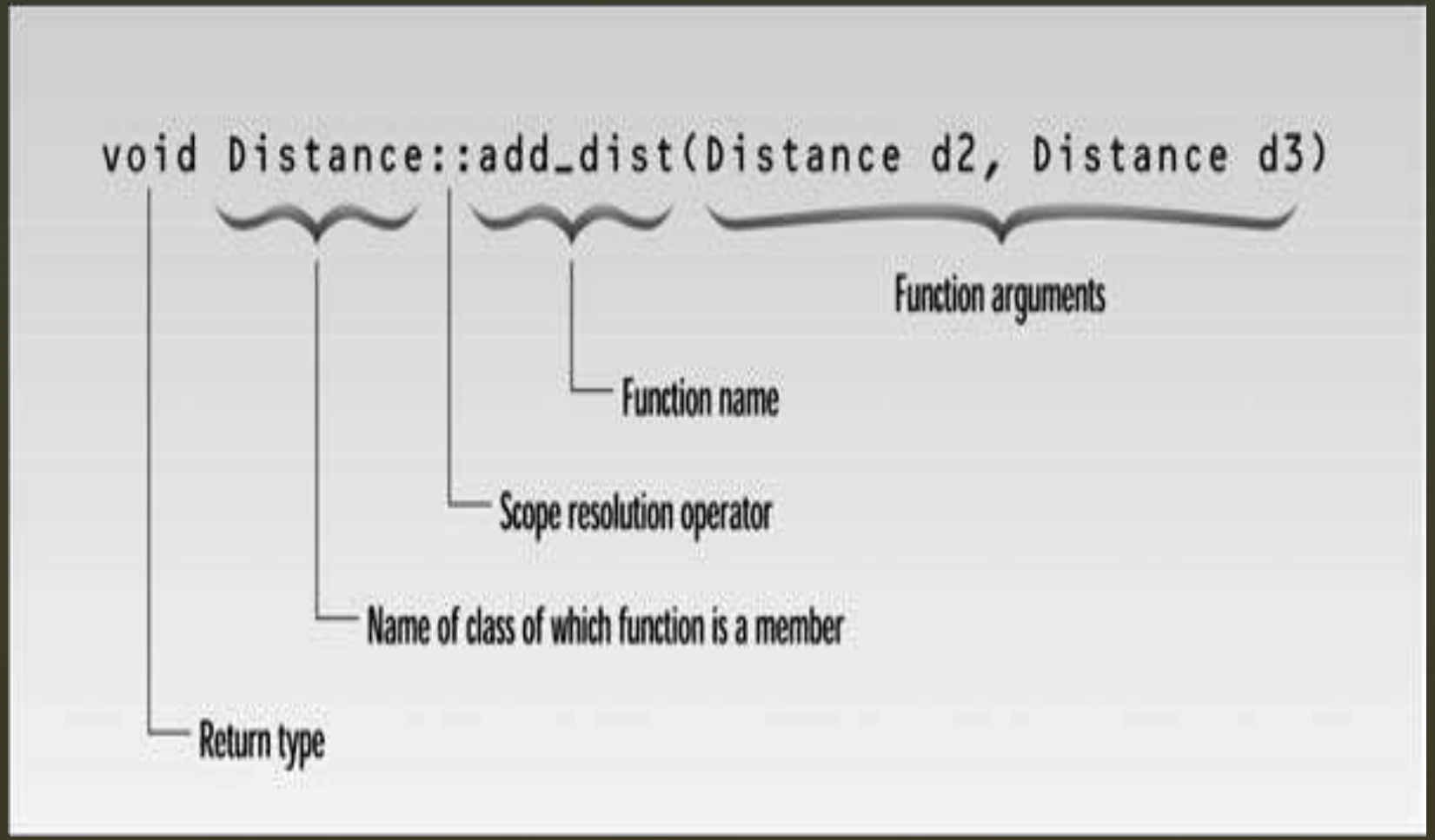
C++ Implementation

- add_dist(), that is not defined within the Distance class definition.
- It is only *declared inside the class, with the statement*

  void add_dist(Distance, Distance);

- This tells the compiler that this function is a member of the class but that it will be defined outside the class declaration, someplace else in the listing.
- The function name, add_dist(), is preceded by the class name, Distance, and a new symbol—the double colon (::).
- This symbol is called the scope resolution operator. It is a way of specifying what class something is associated with. In this situation, Distance::add_dist() means “the add_dist() member function of the Distance class.”

C++ Implementation



The Default Copy Constructor

- A no-argument constructor can initialize data members to constant values.
- A multi-argument constructor can initialize data members to values passed as arguments.
- Another way to initialize an object is to initialize it with *another object of the same type.*
- You don’t need to create a special constructor for this; one is already built into all classes. It’s called the *default copy constructor.*
- *It’s a one argument* constructor whose argument is an object of the same class as the constructor.

The Default Copy Constructor

```
148  // initialize objects using default copy constructor
149  #include <iostream>
150  using namespace std;
151
152  class Distance //English Distance class
153  {
154      private:
155          int feet;
156          float inches;
157      public:
158          //constructor (no args)
159          Distance() : feet(0), inches(0.0)
160              { }
161          //Note: no one-arg constructor
162          //constructor (two args)
163          Distance(int ft, float in) : feet(ft), inches(in)
164              { }
165          void getdist() //get length from user
166          {
167              cout << "\nEnter feet: "; cin >> feet;
168              cout << "Enter inches: "; cin >> inches;
169          }
170          void showdist() //display distance
171          { cout << feet << "\'-" << inches << '\"'; }
172  };
```

The Default Copy Constructor

```
174  int main()
175  {
176      Distance dist1(11, 6.25); //two-arg constructor
177      Distance dist2(dist1); //one-arg constructor
178      Distance dist3 = dist1; //also one-arg constructor
179
180      //display all lengths
181      cout << "\ndist1 = "; dist1.showdist();
182      cout << "\ndist2 = "; dist2.showdist();
183      cout << "\ndist3 = "; dist3.showdist();
184      cout << endl;
185      return 0;
186  }
```

The Default Copy Constructor

- dist1 is initialized to the value of 11’-6.25” using the two-argument constructor.
- Two more objects of type Distance, dist2 and dist3 are defined, initializing both to the value of dist1. These definitions both use the default copy constructor.
- The object dist2 is initialized in the statement

  Distance dist2(dist1);

- This causes the default copy constructor for the Distance class to perform a member-by-member copy of dist1 into dist2.

Static Class Data

- If a data item in a class is declared as static, only one such item is created for the entire class, no matter how many objects there are.
- A static data item is useful when all objects of the same class must share a common item of information.
- A member variable defined as static has characteristics similar to a normal static variable:
  - It is visible only within the class, but its lifetime is the entire program.
  - It continues to exist even if there are no objects of the class.
- While a normal static variable is used to retain information between calls to a function, static class member data is used to share information among the objects of a class.

Static Class Data

```
// static class data
#include <iostream>
using namespace std;

class foo
{
    private:
        static int count; //only one data item for all objects
        //note: "declaration" only!
    public:
        foo() //increments count when object created
    { count++; }
        int getcount() //returns count
    { return count; }
};
```

Static Class Data

```
int foo::count = 0; //*definition* of count

int main()
{
    foo f1, f2, f3; //create three objects

    cout << "count is " << f1.getcount() << endl; //each object
    cout << "count is " << f2.getcount() << endl; //sees the
    cout << "count is " << f3.getcount() << endl; //same value
    return 0;
}
```

Static Class Data

- The class foo in this example has one data item, count, which is type static int.
- The constructor for this class causes count to be incremented.
- In main() three objects of class foo are defined.
- Since the constructor is called three times, count is incremented three times.
- Another member function, getcount(), returns the value in count.

Static Class Data

- Here's the output:

```
count is 3 ← static data
count is 3
count is 3
```

Static Class Data

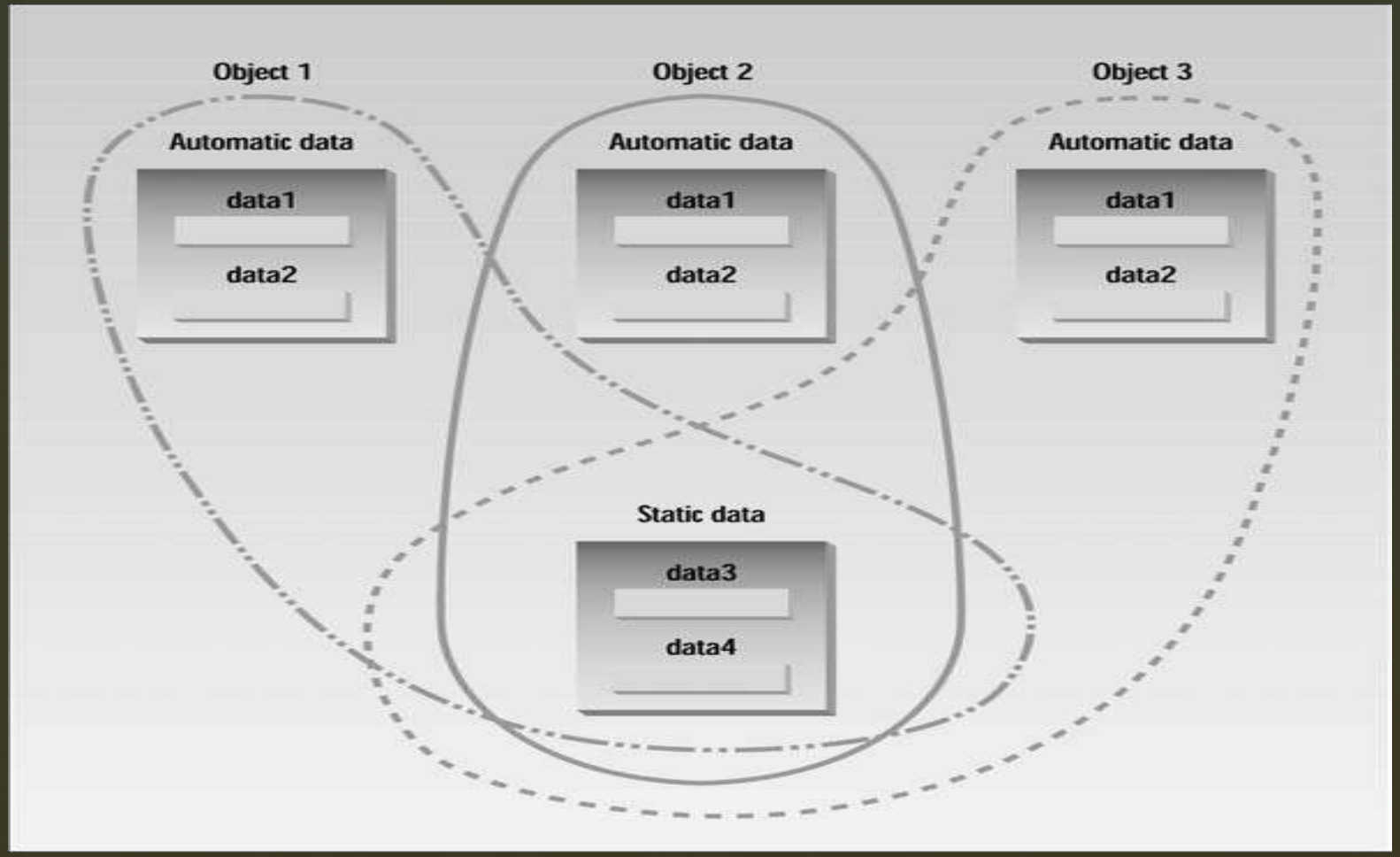
- If we had used an ordinary automatic variable—as opposed to a static variable—for count, each constructor would have incremented its own private copy of count once, and the output would have been

count is 1 ← automatic data

count is 1

count is 1

Static Class Data



const Member Functions

- A const member function guarantees that it will never modify any of its class's member data.
- A function is made into a constant function by placing the keyword const after the declaration but before the function body.
- If there is a separate function declaration, const must be used in both declaration and definition.
- Member functions that do nothing but acquire data from an object are obvious candidates for being made const, because they don't need to modify any data.

Exercise

Imagine a tollbooth at a bridge. Cars passing by the booth are expected to pay a 50 pesewas toll. Mostly they do, but sometimes a car goes by without paying. The tollbooth keeps track of the number of cars that have gone by, and of the total amount of money collected.

Model this tollbooth with a class called tollBooth. The two data items are a type unsigned int to hold the total number of cars, and a type double to hold the total amount of money collected. A constructor initializes both of these to 0.

A member function called payingCar() increments the car total and adds 0.50 to the cash total.

Another function, called nopayCar(), increments the car total but adds nothing to the cash total.

Finally, a member function called display() displays the two totals. Make appropriate member functions const.

Include a program to test this class. This program should allow the user to push one key to count a paying car, and another to count a nonpaying car. Pushing the Esc key should cause the program to print out the total cars and total cash and then exit.

Any Questions?

The End

Contact: tsadjaidoo@knust.edu.gh

Office: Caesar Building, Room 413