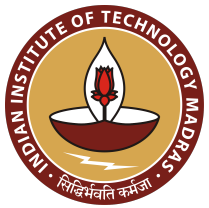
Object-Oriented Programming
The Big Picture

Saurav Samantaray

Department of Mathematics

Indian Institute of Technology Madras

July 22, 2024



Texts and References

1. Objected-Oriened Programming in C++, Robert Lafore.
2. Sams Teach Yourself C++ in one Hour a Day, Siddhartha Rao.
3. C++ Primer, Stanley B. Lipmann et. al.
4. Guide to Scientific Computing in C++, Joe Pitt-Francis, Jonathan Whiteley.

Programming Inovations

- In the old days, 40 or so years ago, programmers starting a project would sit down almost immediately and start writing code.
- As programming projects became large and more complicated, it was found that this approach did not work very well (problem was complexity).
- Large programs are prone to error, and software errors can be expensive and even life threatening (in air traffic control, for example).
- Three major innovations in programming have been devised to cope with the problem of complexity:
  1. Object-oriented programming (OOP)
  2. The Unified Modeling Language(UML)
  3. Improved software development processes

Object-Oriented Programming (OOP)

OOP offers a new and powerful way to cope with complexity.

- Instead of viewing a program as a series of steps to be carried out, it views it as a group of objects that have certain properties and can take certain actions.

The Unified Modeling Language

The Unified Modeling Language (UML) is a graphical language consisting of many kinds of diagrams.

- It helps program analysts figure out what a program should do, and helps programmers design and understand how a program works.
- The UML is a powerful tool that can make programming easier and more effective.

C++ Is Not the Same as C

- C and C++ are entirely separate languages.
- It's true that their syntax is similar, and C is actually a subset of C++.
- But the similarity is largely a historical accident.
- The basic approach in a C++ program is radically different from that in a C program.

What is an object !!! and in-turn object oriented programming

Procedural Languages

- The software is decomposed into various functional components.
- The program is written as a collection of functions, which are implemented in particular order to achieve the envisioned results.

Object-based Programming

The approach is to organise the software into a collection of components, called objects, that group together:

- Related items of data, known as properties.
- Operations that are to be performed on the data, which are known as methods.

Procedural Languages

- Examples: C, Pascal, FORTRAN, etc.
- A program in a procedural language is a list of instructions.
- For very small programs, no other organising principle is needed.

Division into Functions:

- When programs become larger, few programmers can comprehend a program unless it is broken down into smaller units.
- Owing to this reason the *function* was adopted as a way to make programs more comprehensible to their human creators.
- A procedural program is divided into functions.
- Each function has a clearly defined purpose and a clearly defined interface to the other functions in the program.
- Breaking a program into functions can be further extended by grouping a number of functions together into a larger entity called a *module* (file).

Problems with Structured Programming

- As programs grow ever larger and more complex, even the structured programming approach begins to show signs of strain.
- What are the reasons for these problems with procedural languages?

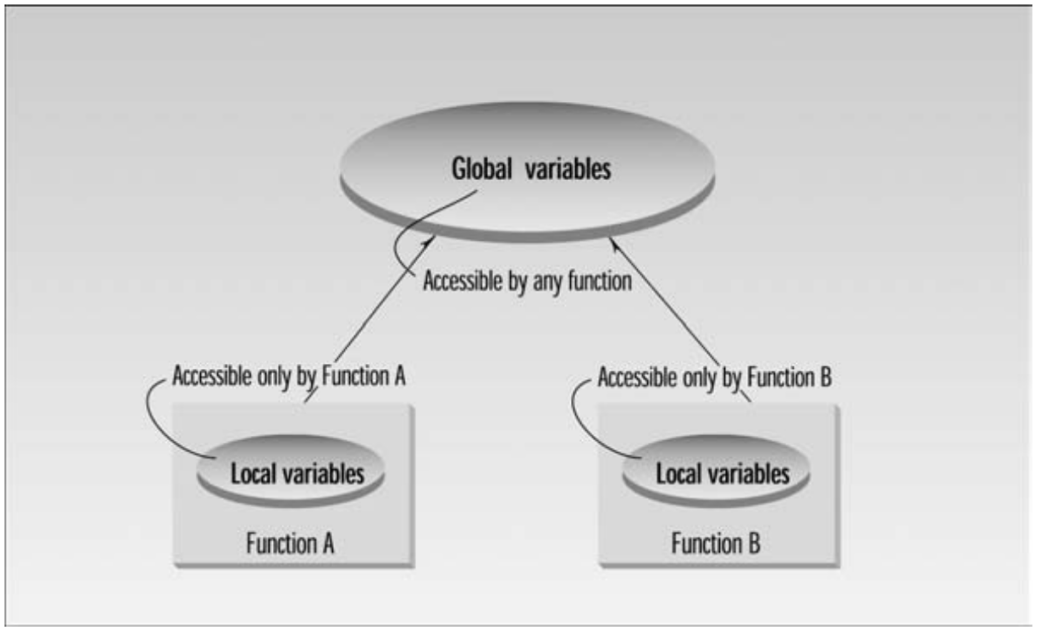
Two related problems:

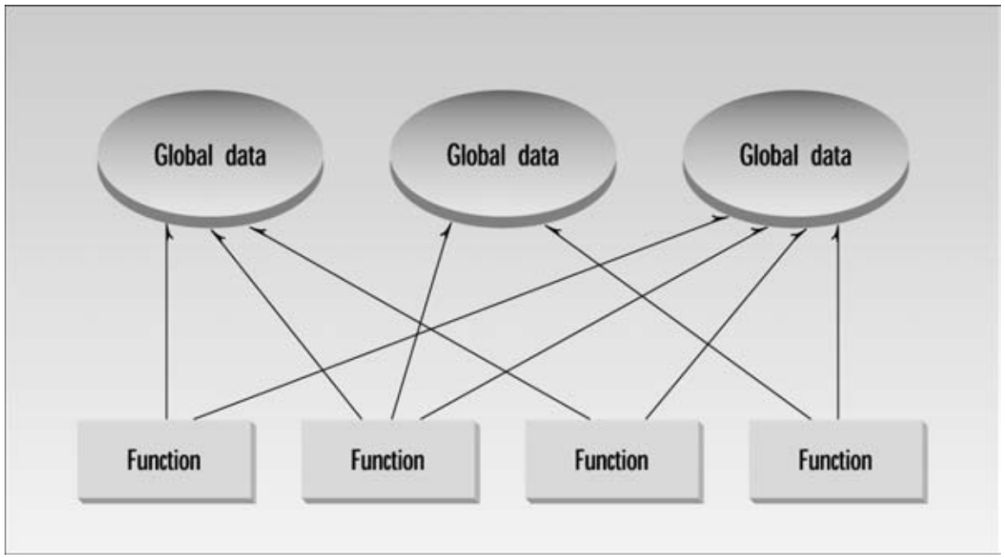
1. functions have unrestricted access to global data.
2. unrelated functions and data, the basis of the procedural paradigm, provide a poor model of the real world.

Unrestricted Access

- In a procedural program, one written in C for example, there are two kinds of data:
  1. *Local data*: hidden inside a function, and is used exclusively by the function. Local data is closely related to its function and is safe from modification by other functions.
  2. *Global data*: when two or more functions must access the same data. Global data can be accessed by any function in the program.
- Large programs have many functions and many global data items.
- The problem with the procedural paradigm is that this leads to an even larger number of potential connections between functions and data.

Global and Local Access





Problems with structured Programming

- Large number of connections causes problems in several ways:
  1. it makes a program's structure difficult to conceptualise.
  2. it makes the program difficult to modify.
- A change made in a global data item may necessitate rewriting all the functions that access that item.
- When data items are modified in a large program it may not be easy to tell which functions access the data.
- Modifications to the functions may cause them to work incorrectly with other global data items.
- Everything is related to everything else, a modification has far-reaching, and often unintended, consequences.

- The second—and more important—problem with the procedural paradigm: its arrangement of separate data and functions does a poor job of modeling things in the real world.
- Real world objects such as cars, people, etc. have both *attributes* and *behaviour*.
- **Attributes**:for people,
  - eye color and
  - job title;

  and, for cars,
  - horsepower and
  - number of doors.

  Attributes in the real world are equivalent to data in a program.
- **Behaviour**: Behavior is something a real-world object does in response to some stimulus. *Ask your boss for a raise, she will generally say yes or no.*
- Behavior is like a function: you call a function to do something (display the inventory, for example) and it does it.

So neither data nor functions, by themselves, model real-world objects effectively.

- Combine into a single unit both *data* and the *functions* that operate on that data.
- Such a unit is called an *object*.
- Typically the only way to access the data of an *object* is via the *functions* in the *object* called *member functions*.
- The data is hidden, so it is safe from accidental alteration.
- *Data* and its *functions* are said to be *encapsulated* into a single entity.
- *Data encapsulation* and *data hiding* are key terms in the description of object-oriented languages.

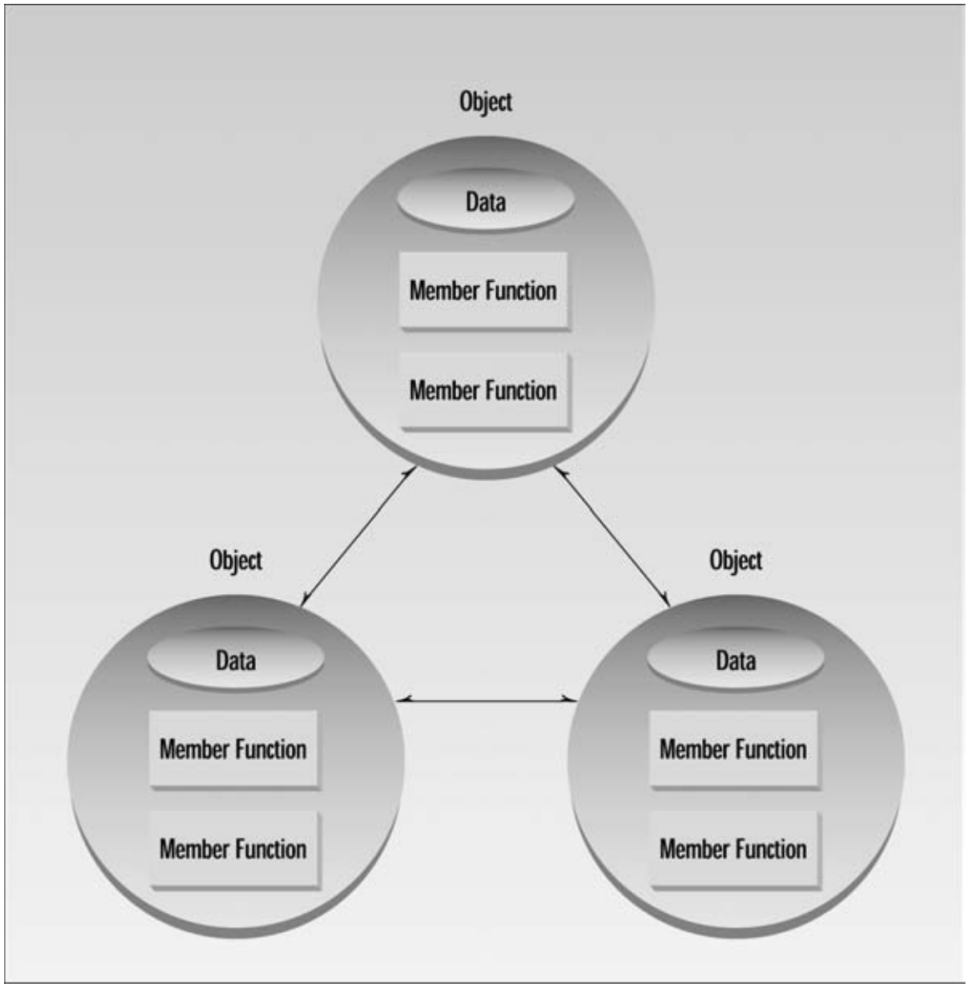


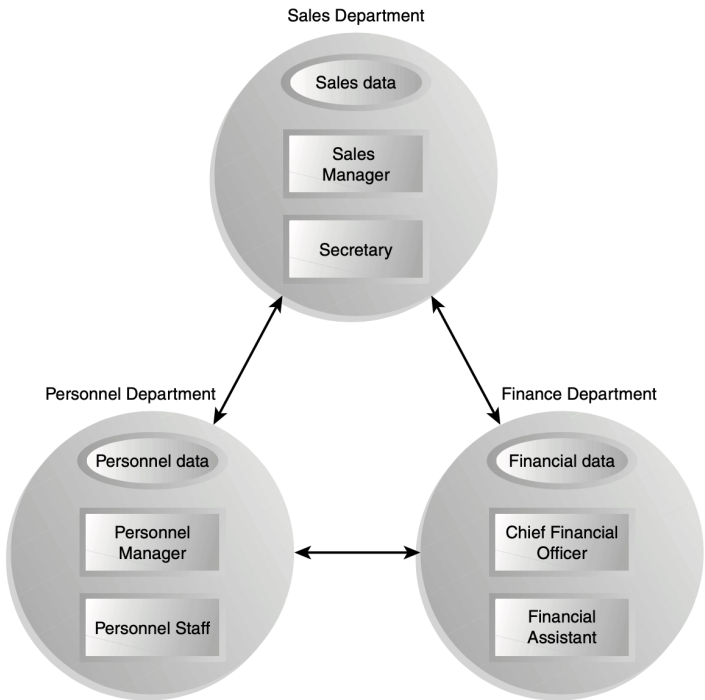
Figure 1: A C++ program typically consists of a number of objects, which communicate with each other by calling one another's member functions.

An Analogy

- Think of objects as departments—such as -> sales, -> accounting, -> personnel, and so on—in a company.
- In most companies, people don't work on personnel problems one day, the payroll the next, and then go out in the field as salespeople the week after.
- Each department has its own personnel, with clearly assigned duties.
- It also has its own data: the accounting department has payroll figures, the sales department has sales figures, the personnel department keeps records of each employee, and so on.
- Dividing the company into departments makes it easier to comprehend and control the company's activities, and helps maintain the integrity of the information used by the company.

An Analogy

- If you need to know the total of all the salaries paid in the southern region in July, send a memo to the appropriate person in the department, then wait for that person to access the data and send you a reply with the information you want.
- Ensures that the data is accessed accurately and that it is not corrupted by inept outsiders.



Example of an Object in real life

Triangle is a geometric object

- Can be defined by its vertices.
- Can have functions associated to it such as:
  1. area;
  2. centroid;
  3. length of sides;
  4. mid-points of sides; et cetera

OOP: An Approach to Organization

- Object-oriented programming is not primarily concerned with the details of program operation.
- Instead, it deals with the overall organization of the program.
- Most individual program statements in C++ are similar to statements in procedural languages, and many are identical to statements in C.
- When one looks at the larger context that one can determine whether a statement or a function is part of a procedural C program or an object-oriented C++ program.

Objects

- *How the problem will be divided into objects, but not into functions?*
- *Thinking in terms of objects, rather than functions, has a helpful effect on how easily programs can be designed!!*

What kinds of things become objects in object-oriented programs?

Answer: Limited only by your imagination.

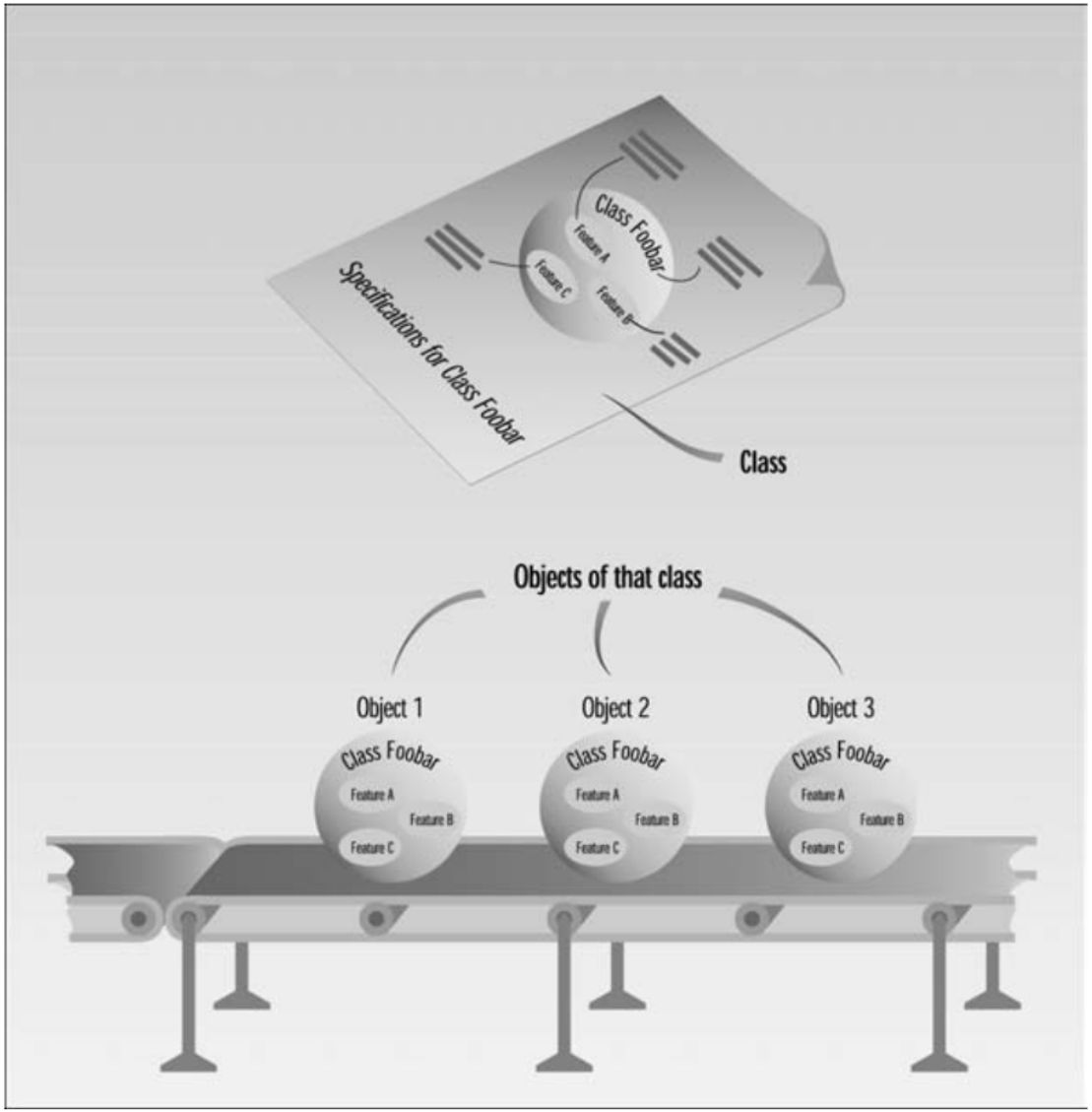
- Physical objects: Automobiles in a traffic-flow simulation; Electrical components in a circuit-design program
- Data-storage constructs: Customized arrays; Stacks ; Linked lists
- Human entities: Employees; Students; Customers

- In OOP objects are members of *classes*.
- Almost all computer languages have built-in data types.
- Declare as many variables of type `int` as you need in your program:

```
int day;
int count;
int divisor;
int answer;
```

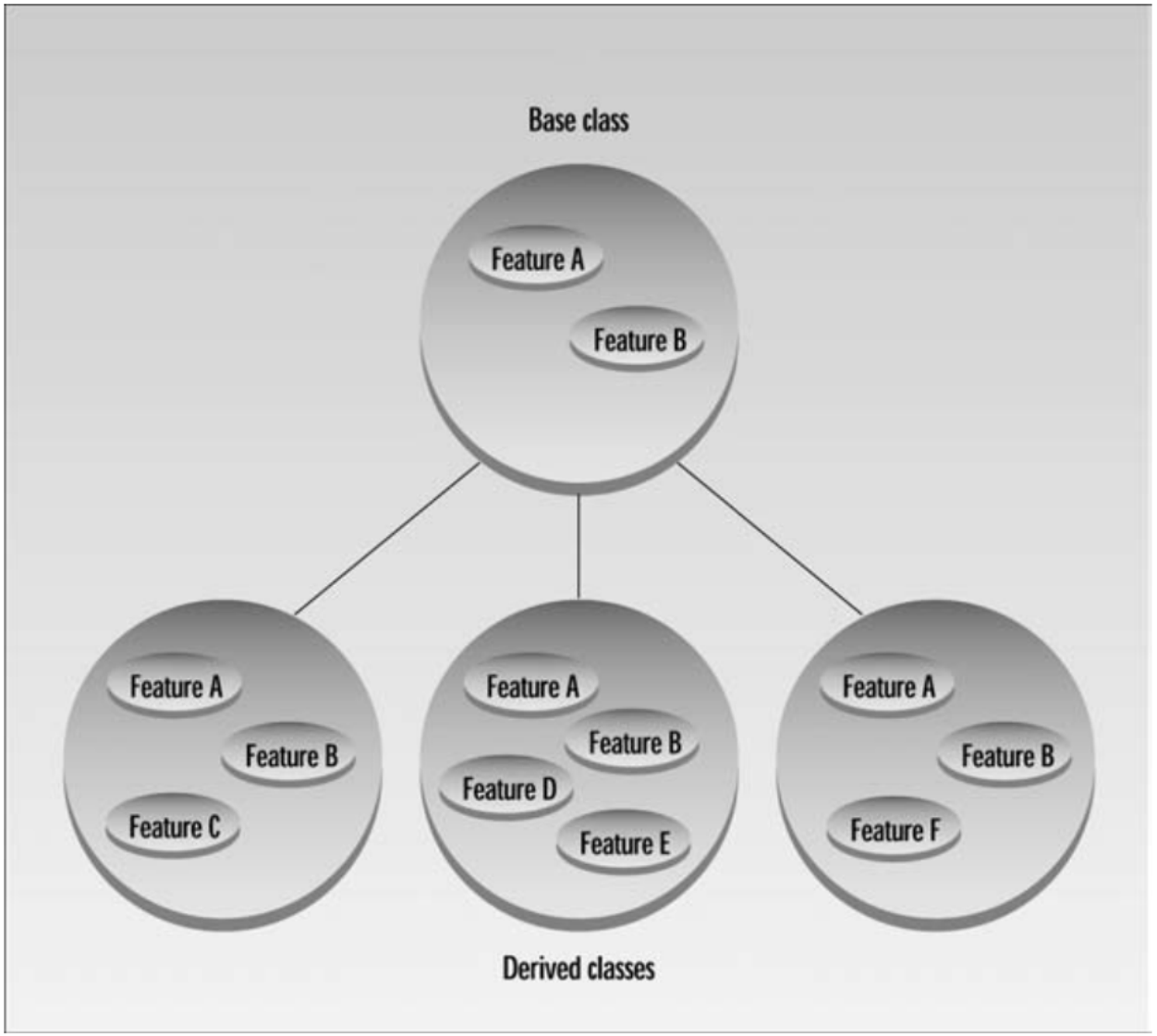
- In a similar way, one can define many objects of the same class.
- A class serves as a plan, or blueprint.
- It specifies what data and what functions will be included in objects of that class.
- Defining the class doesn't create any objects.
- An object is often called an "instance" of a class.

Classes


Class Foobar
Feature A
Feature C
Feature B
Specifications for Class Foobar
Class
Objects of that class
Object 1
Class Foobar
Feature A
Feature B
Feature C
Object 2
Class Foobar
Feature A
Feature B
Feature C
Object 3
Class Foobar
Feature A
Feature B
Feature C

- A vehicle class can be divided into cars, trucks, buses, motorcycles, and so on.
- Each subclass shares common characteristics with the class from which it's derived.
- Cars, trucks, buses, and motorcycles all have wheels and a motor; these are the defining characteristics of vehicles.
- In addition to the characteristics shared with other members of the class, each subclass also has its own particular characteristics.
- An OOP class can become a parent of several subclasses.
- In C++ the original class is called the base class; other classes can be defined that share its characteristics, but add their own as well. These are called derived classes.

Inheritance



- Once a class has been written, created, and debugged, it can be distributed to other programmers for use in their own programs.
- By deriving a new class from the existing one, a programmer can take an existing class and, without modifying it, add additional features and capabilities to it.
- For example, a class that creates a menu system, such as that used in Windows or other Graphic User Interfaces (GUIs).
- This class works fine, and you don't want to change it, but you want to add the capability to make some menu entries flash on and off.
- The ease with which existing software can be reused is an important benefit of OOP.

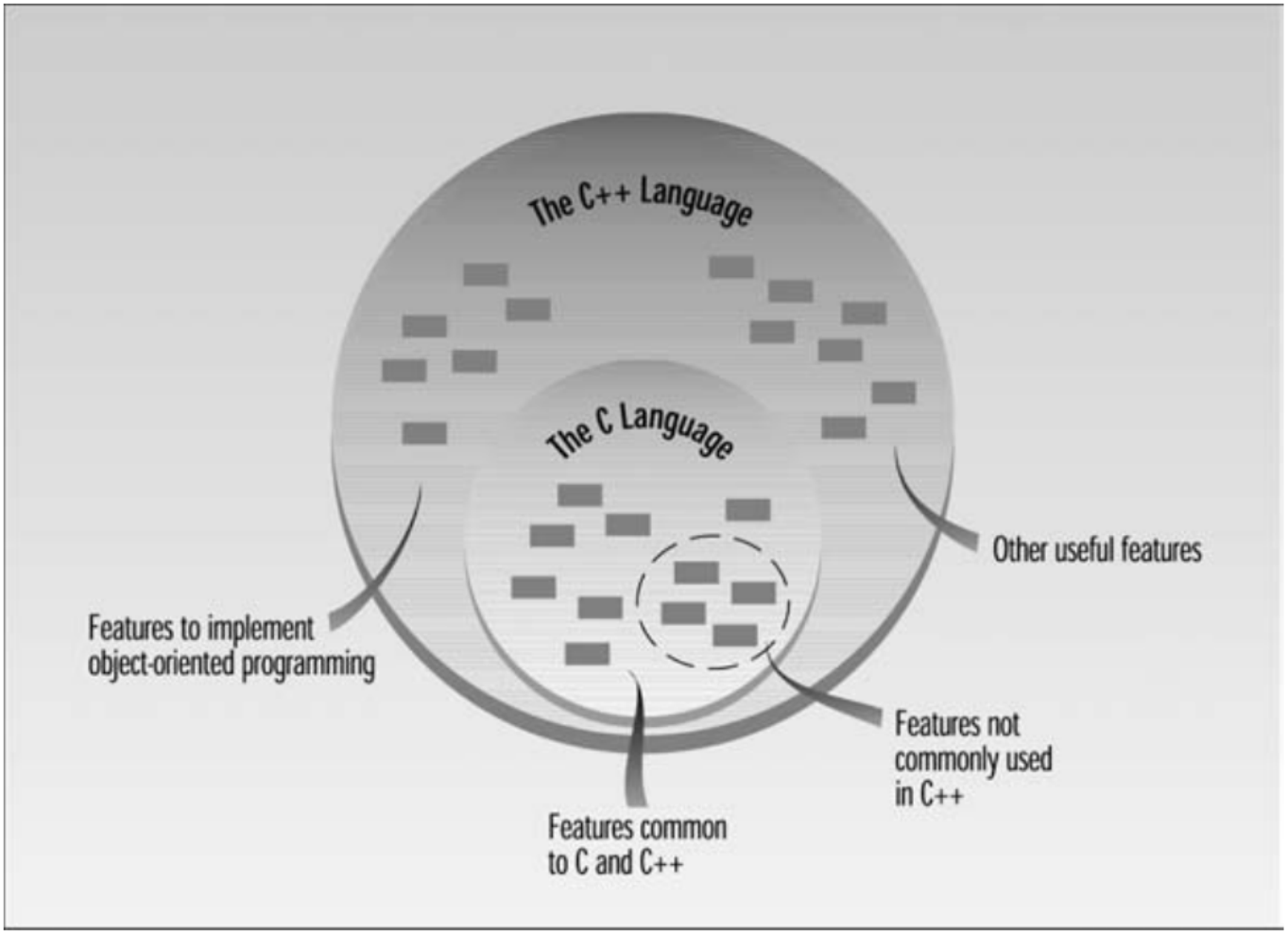
Creating New Data Types

- A convenient way to construct new data types.
- Two-dimensional positions, such as x and y coordinates, or latitude and longitude.
- Express operations on these positional values with normal arithmetic operations, such as:
  `position1 = position2 + origin`
- The variables `position1`, `position2`, and `origin` each represent a pair of independent numerical quantities.
- Creating a class that incorporates these two values, and declaring `position1`, `position2`, and `origin` to be objects of this class, we can, in effect, create a new data type.

Polymorphism and Overloading

- = (equal) and + (plus) operators, used in the position arithmetic shown above, don't act the same way they do in operations on built-in types such as `int`.
- How do the = and + operators know how to operate on objects?
- We can define new behaviours for these operators.
- These operations will be member functions of the `Position` class.
- Using operators or functions in different ways, depending on what they are operating on, is called *polymorphism*.
- When an existing operator, such as + or =, is given the capability to operate on a new data type, it is said to be *overloaded*.
- Overloading is a kind of polymorphism.

- C++ is derived from the C language.
- Almost every correct statement in C is also a correct statement in C++, although the reverse is not true.



The Unified Modeling Language (UML)

- The UML is a graphical "language" for modeling computer programs.
- "Modeling" means to create a simplified representation of something, as a blueprint models a house.
- The UML provides a way to visualize the higher-level organization of programs without getting mired down in the details of actual code.
- The UML is not a software development process.
- The UML is simply a way to look at the software being developed.
- The UML is especially attuned to OOP.

Why do we need the UML?

- In a large computer program it's often hard to understand, simply by looking at the code, how the parts of the program relate to each other.
- Though object-oriented programming is a vast improvement over procedural program the trouble with code is that it's very detailed
- It would be nice if there were a way to see a bigger picture, one that depicts the major parts of the program and how they work together.
- The UML answers this need.
- The most important part of the UML is a set of different kinds of diagrams:
  - Class diagrams show the relationships among classes;
  - object diagrams show how specific objects relate;
  - sequence diagrams show the communication among objects over time;
  - use case diagrams show how a program's users interact with the program, and so on.

Summary

- OOP is a way of organising programs.
- In particular, OOP programs are organised around objects, which contain both data and functions that act on that data.
- A class is a template for a number of objects.
- Inheritance allows a class to be derived from an existing class without modifying it.
- Inheritance makes possible re-usability, or using a class over and over in different programs.
- The UML is a standardised way to visualise a program's structure and operation using diagrams.