CS-1201 Object-Oriented Programming

Classes, Objects, and Constructors

Arbish Akram

Department of Computer Science
Government College University

Class

- A class is a *user-defined data type* that acts as a blueprint for creating objects.
- It *encapsulates* data for the object and methods to manipulate that data.
- A class typically defines:
  - Attributes: variables that hold the *state* of an object.
  - Methods: functions that define the *behavior* of the object.
- Classes do not occupy memory until an object is created.
- Think of a class as a plan or template.

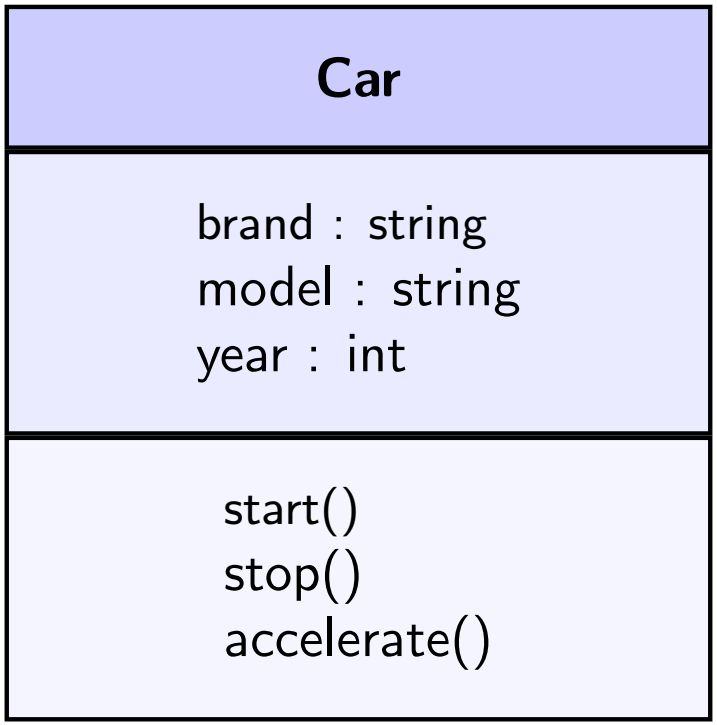
Objects

- An object is an instance of a class, meaning it is created based on the class definition.
- Every object contains its *own* set of data, following the class structure, with values specific to that object.
- Each object can have unique values for its attributes.
- Key characteristics:
  - State: defined by the values of the object's attributes.
  - Behavior: actions the object can perform.
  - Identity: each object is distinct, even if its state and behavior are identical to another object.
- Think of an object as a real-world entity created from a blueprint (class).

Class

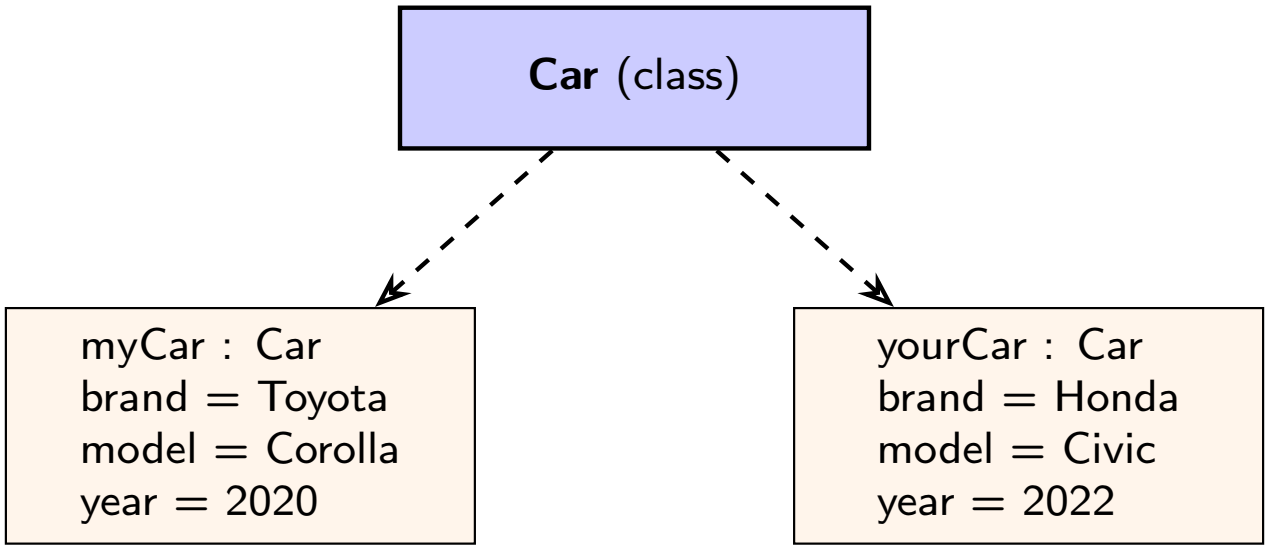
- A class can be visualized as a three-compartment box.
  1. Class name: identifies the class.
  2. Data Members: the attributes of the class.
  3. Member Functions: the operations of the class.

Class



- Top: the name of the class.
- Middle: data members (attributes) that hold state.
- Bottom: member functions (methods) that define behavior.

Class



- One class, `Car`, is used as the template for many objects.
- Every object built from `Car` has the same attributes (`brand`, `model`, `year`), but each can hold different values.
- Objects are independent: changing `myCar` has no effect on `yourCar`.

Syntax of a Class

```
class ClassName {
private:
    // data members: hidden from outside code
    dataType attribute1;
    dataType attribute2;

public:
    // member functions: the public interface
    returnType method1(parameters);
    returnType method2(parameters);
};
```

- `class` starts the definition, followed by the class name.
- Everything between `{` and `};` belongs to the class. Do not forget the final semicolon.
- `private` and `public` are access specifiers, covered next.

Access Specifiers

- Access specifiers control which parts of a class can be reached from outside code.
- C++ provides three levels:
  - `public`: accessible from anywhere the object is visible.
  - `private`: accessible only from within the class itself. This is the default if no specifier is written.
  - `protected`: accessible within the class and its derived classes (used with inheritance, discussed in a later lecture).
- Encapsulation, in practice, means keeping data members `private` and exposing controlled access through `public` methods.

Example: Class With Encapsulation

```
1  class Car {
2  private:
3      string brand, model;
4      int year;
5
6  public:
7      void setYear(int y) {
8          if (y >= 1886) year = y;  // simple validation
9      }
10     int getYear() { return year; }
11     void start() { cout << "Car started!" << endl; }
12 };
```

- `brand`, `model`, and `year` are now `private`: no outside code can touch them directly.
- `setYear()` and `getYear()` are the only doors in and out of `year`, and `setYear()` can reject invalid values.
- This is encapsulation in practice: the data is protected, and access happens only through a controlled interface.

Object

- An object is an instance of a class.
- Objects are created using the class as a blueprint, with their own distinct values for attributes.
- Objects can perform actions using the methods defined in their class.

```
1  int main() {
2      Car myCar;  // Create an object of the Car class
3      myCar.brand = "Toyota";
4      myCar.model = "Corolla";
5      myCar.year = 2020;
6
7      myCar.start();  // Call the start method on the object
8      return 0;
9  }
```

Constructors

- A constructor is a special *member function* of a class.
- It is automatically called when an object of the class is created.
- Constructors are used to *initialize objects*.
- The name of the constructor is the same as the class name.
- Constructors do not have a return type, not even `void`.
- A class can have more than one constructor.

Types of Constructors

- **Default Constructor**: takes no parameters. The compiler provides one automatically if you define no constructors at all.
- **Parameterized Constructor**: takes arguments to initialize an object with specific values.
- **Copy Constructor**: creates a new object as a copy of an existing object of the same class.

Example: Default and Parameterized Constructors

```
1  class Car {
2  private:
3      string brand, model;
4      int year;
5  public:
6      Car() : brand("Unknown"), model("Unknown"), year(0) {}
7      Car(string b, string m, int y) : brand(b), model(m), year(y) {}
8
9      void show() {
10         cout << brand << " " << model << " " << year << endl;
11     }
12 };
13 int main() {
14     Car car1;
15     Car car2("Toyota", "Corolla", 2020);
16     car1.show();
17     car2.show();
18     return 0;
19 }
```

Predict the Output

Using the Car class from the previous slide:

```
1  Car car1;
2  Car car2("Honda", "Civic", 2022);
3  Car car3("Suzuki", "Alto", 2015);
4
5  car1.show();
6  car3.show();
7  car2.show();
```

Constructor: Initialization List

- Constructors often initialize data members.
  - Instead of:

    ```
    Car() { year = 0; }
    ```

  - Preferred approach:

    ```
    Car() : year(0) { }
    ```

- Why use an initialization list?
  - Members are initialized *before* the constructor body executes, not assigned afterward.
  - It is the only way to initialize `const` members and *reference* members.
  - Members are separated by commas in the list.

Initialization Order

```
1  class Foo {
2      int a, b;    // NOTE: a is declared before b
3  public:
4      Foo(int x) : b(x), a(b + 1) { }   // list written b, then a
5  };
6
7  int main() {
8      Foo f(10);
9      // what is f's value of a?
10 }
```

1. The initialization list is written as `b(x), a(b + 1)`. Does that mean `b` is initialized first?
2. Data members are always initialized in the order they are *declared* in the class, not the order they appear in the list.

Default Constructor

- If a class has *no constructor* at all, C++ automatically generates a *default constructor*.
- The compiler-generated default constructor does nothing it does not set members to zero or any particular value.
- As soon as you write *any* constructor yourself, the compiler stops providing this automatic one.

Constructor Overloading

- Constructors can be overloaded, just like ordinary functions.
- Overloaded constructors share the same name (the class name) but differ in the number and/or type of parameters.
- The compiler picks the matching constructor based on the arguments passed at object-creation time.

Example: Overloaded Constructors

```
1  class Person {
2  private:
3      string name;
4      int age;
5
6  public:
7      Person() : name("Unknown"), age(0) { }
8      Person(string n) : name(n), age(0) { }
9      Person(string n, int a) : name(n), age(a) { }
10     void show() { cout << name << ", " << age << endl; }
11 };
12
13 int main() {
14     Person p1;
15     Person p2("Ali");
16     Person p3("Sara", 21);
17 }
```

Member Functions Defined Outside the Class

- A member function can be defined inside or outside the class body.
- Declaring functions inside the class keeps the interface clear; defining them outside keeps implementation details separate.
- Definitions outside the class make the class declaration itself shorter and easier to scan.

Member Functions Defined Outside the Class

```
1  class Distance {
2  private:
3      int feet;
4      float inches;
5
6  public:
7      Distance();                 // constructor declaration
8      void show_dist();           // member function declaration
9  };
10
11 // definitions outside the class
12 Distance::Distance() : feet(0), inches(0.0) { }
13 void Distance::show_dist() {
14     cout << feet << "'-" << inches << "\"" << endl;
15 }
```

- `Distance::` is the scope resolution operator, it says "this definition belongs to the `Distance` class."
- Notice the constructor itself is defined outside the class too.

Case Study: Library Management System

A university wants a system to manage its library. Members can borrow and return books. The library needs to track which books are available, who borrowed which book, and when each book is due back.

The library also has different types of members. Students and staff can both borrow books, but they may have different borrowing limits. A member has a name and a membership ID. Each book has a title, an author, and an ISBN. When a member borrows a book, the system creates a loan that records the borrowing date and the due date. The system should also allow a member to return a book and determine whether a loan is overdue.

Case Study: Library Management System - Finding Classes

A university wants a system to manage its library. Members can borrow and return books. The library needs to track which books are available, who borrowed which book, and when each book is due back.

The library also has different types of members. Students and staff can both borrow books, but they may have different borrowing limits. A member has a name and a membership ID. Each book has a title, an author, and an ISBN. When a member borrows a book, the system creates a loan that records the borrowing date and the due date. The system should also allow a member to return a book and determine whether a loan is overdue.

Now look only for the important classes.

Case Study: Library Management System - Finding Attributes

A university wants a system to manage its library. Members can borrow and return books. The library needs to track which books are available, who borrowed which book, and when each book is due back.

The library also has different types of members. Students and staff can both borrow books, but they may have different borrowing limits. A member has a name and a membership ID. Each book has a title, an author, and an ISBN. When a member borrows a book, the system creates a loan that records the borrowing date and the due date. The system should also allow a member to return a book and determine whether a loan is overdue.

Blue = Classes **Green = Attributes**

Case Study: Library Management System - Finding Behaviours

A university wants a system to manage its library. Members can borrow and return books. The library needs to track which books are available, who borrowed which book, and when each book is due back.

The library also has different types of members. Students and staff can both borrow books, but they may have different borrowing limits. A member has a name and a membership ID. Each book has a title, an author, and an ISBN. When a member borrows a book, the system creates a loan that records the borrowing date and the due date. The system should also allow a member to return a book and determine whether a loan is overdue.

Blue = Classes **Green = Attributes** **Red = Behaviours**

Class or Object?

For each item below, say whether it is acting as a **class** or as an **object**.

1. `Dog`
2. Your neighbor's dog, named Fido
3. `BankAccount`
4. The specific account with number 00219, balance $500
5. `Student`
6. You, as a student enrolled in this course

Summary

- A class is a blueprint; an object is a specific instance built from that blueprint.
- A class bundles data members (state) with member functions (behavior).
- Access specifiers (public, private, protected) are how encapsulation is actually implemented in C++.
- Constructors initialize objects automatically at creation time; a class can overload several of them (default, parameterized, copy).
- Initialization lists run before the constructor body, in **declaration order**, not the order written in the list.
- Member functions can be declared inside a class and defined outside it using `ClassName::`, keeping interface and implementation separate.
- Identifying classes, and their attributes from a written description is a design skill, not a mechanical step, and different reasonable designs can work.