

CSC 211: Computer Programming

Classes, Header Files, Constructors

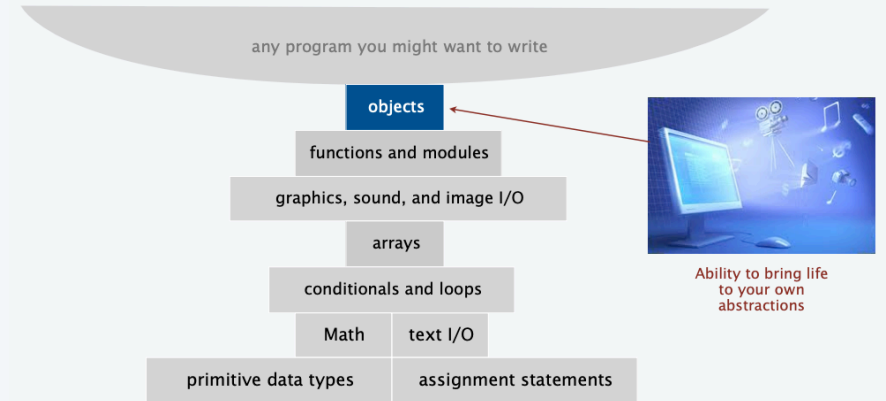
Michael Conti

Department of Computer Science and Statistics
University of Rhode Island

Spring 2025



Basic building blocks



<https://introcs.cs.princeton.edu/java/lectures/keynote/CS9.CreatingDTIs.pdf>

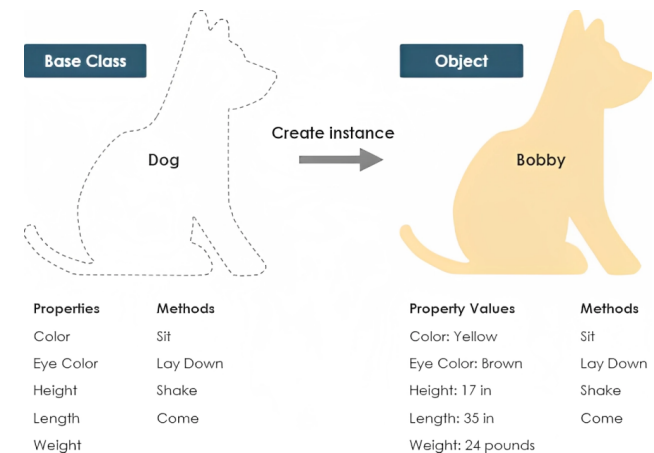
2

Classes

- In object-oriented programming (OOP), a **class** is an extensible “datatype” for creating **objects** (“variables”)
- Examples of classes you have already used
 - `std::string`, `std::istream`, `std::ostream`
- A class can define **member variables** and behavior (called **member functions** or **methods**)
- When an object is created, the resulting object is also called an **instance of the class**

3

Classes



4

C++ Classes

- A class in C++ is a user-defined type declared with the keyword **class**
- A class can define data members and member functions
 - three levels of access: **private** (default), **protected**, or **public**
- **Private members** are not accessible outside the class
 - only through methods of the class
- **Public members** form an interface to the class and are accessible outside the class

5

Class declaration

- Similar to structs, however level of access must be specified

```
class MyClass {  
    public:  
        int myNum;  
        string myString;  
};
```

6

Declaration and dot operator

```
#include <iostream>  
#include <string>  
  
class MyClass {  
    // access specifier  
    public:  
        // data members  
        int myNum;  
        std::string myString;  
};  
  
int main() {  
    // creating an object  
    MyClass object;  
  
    // using the dot operator  
    object.myNum = 10;  
    object.myString = "My Message";  
    std::cout << object.myNum << std::endl;  
    std::cout << object.myString << std::endl;  
  
    return 0;  
}
```

7

Methods (member functions)

- Methods must be declared inside the class
 - definition of methods must identify the class they belong to
 - **::** is the **scope resolution operator**

```
class Date {  
    public:  
        int month;  
        int year;  
        int day;  
  
        void print();  
};  
  
void Date::print() {  
    std::cout << month << '-'  
        << day << '-'  
        << year << std::endl;  
}  
  
// declaration  
// definition
```

8

Example

```
#include <iostream>

class Date {
public:
    int month;
    int year;
    int day;

    void print();
};

void Date::print() {
    std::cout << month << '-'
               << day << '-'
               << year << std::endl;
}

int main() {
    Date today;

    today.day = 16;
    today.month = 07;
    today.year = 2024;

    today.print();

    return 0;
}
```

Must include the object

9

Improving the class declaration

- Making changes to the internal representation of **Date** requires changes to the entire program
- A better declaration of the class **Date** would allow for changes to the class without requiring changes to the program(s) that use **Date**

don't allow the program to directly reference or access member variables

10

Example

```
#include <iostream>

class Date {
public:
    int mymonth;
    int year;
    int day;

    void print();
};

void Date::print() {
    std::cout << month << '-'
               << day << '-'
               << year << std::endl;
}

int main() {
    Date today;

    today.day = 12;
    today.month = 07;
    today.year = 2023;

    today.print();

    return 0;
}
```

Internal change to date broke this line

11

```
#include <iostream>

class Date {
public:
    int month;
    int year;
    int day;

    void set(int m, int d, int y);
    void print();
};
```

12

```

void Date::print() {
    std::cout << month << '-'
               << day << '-'
               << year << std::endl;
}

void Date::set(int m, int d, int y) {
    month = m;
    day = d;
    year = y;
}

```

When accessing data members or methods inside the class, the object name is not required

13

```

int main() {
    Date today;

    today.set(07, 12, 2023);
    today.print();

    return 0;
}

```

Now changes to the date class will not require changes to main (programs that use date)

14

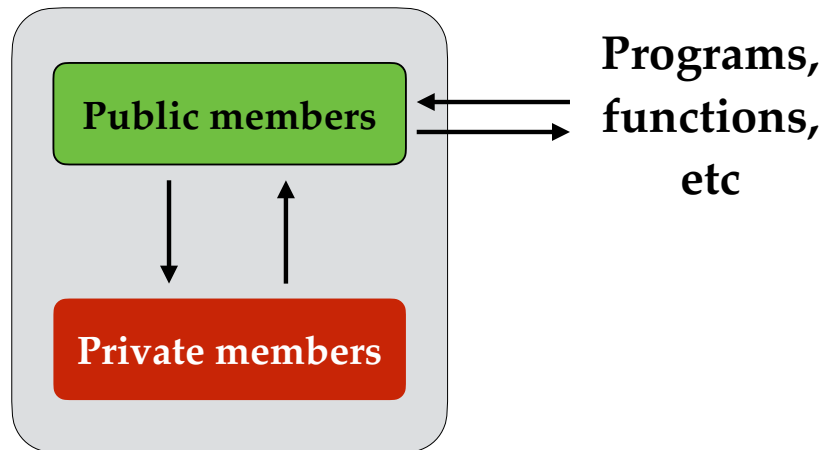
Encapsulation

- **Encapsulation** is one of the most fundamental concepts of OOP
- In OOP, encapsulation is used to hide the values or state of a structured data object inside a class. It is implemented as a:
 - ✓ language construct that **facilitates the bundling of data with the methods** (or other functions) operating on that data
 - ✓ language mechanism for **restricting direct access** to some of the object's components

Public vs Private

- C++ helps us restrict the program from directly referencing member variables
- Private members of a class can **only be referenced** within member functions
 - ✓ otherwise, the compiler gives an error message
- The keyword **private** identifies the members of a class that can be accessed only by member functions
- The keyword **public** identifies the members of a class that can be accessed from outside the class

Object



17

```
class Date {  
    private:  
        int month;  
        int year;  
        int day;  
  
    public:  
        void set(int m, int d, int y);  
        void print();  
};
```

18

Assignment operator

- Objects and structures can be assigned values using the = operator

```
int main() {  
    Date today;  
    Date due;  
  
    today.set(07, 12, 2023);  
    due = today;  
    today.print();  
    due.print();  
  
    return 0;  
}
```

19

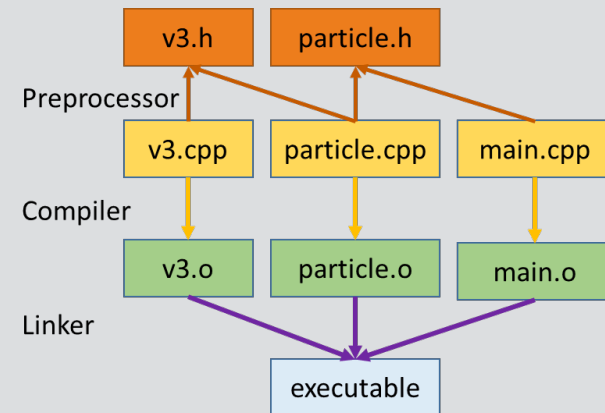
Header Files

Separate compilation

- Source code can be divided into multiple files
 - ✓ source files can be compiled separately
 - ✓ enterprise code files can take hours to compile
 - Source code separation eliminates the need to compile everything, all the time
- Classes can be implemented in their own files
 - ✓ allows reusing codes in multiple programs
 - ✓ source files including class methods and function definitions
 - ✓ header files including declarations and global constants

21

Compiling multiple files



```
g++ v3.cpp particle.cpp main.cpp -o executable
```

<https://devblogs.nvidia.com/separate-compilation-linking-cuda-device-code/>

22

#include

- Used for including header files
 - ✓ usually contains class declarations, function prototypes, or global constants
- When used with < >
 - ✓ The **preprocessor searches** in an implementation dependent manner, normally in search **directories pre-designated by the compiler/IDE**. This method is normally used to include standard library header files
- When used with " "
 - ✓ The **preprocessor searches first in the same directory** as the file containing the directive, and then follows the search path used for the #include <filename> form. This method is normally used to include programmer-defined header files.
- Cannot compile header files directly!

23

Multiple declarations of classes

- With large projects, multiple declaration of classes must be prevented

- Use #ifndef

```
#ifndef DATE_H
#define DATE_H
```

```
class Date {
    // ...
};
```

```
#endif
```

24

Multiple declarations of classes

- Do header guards need to be capital or use an underscore instead of a dot?
 - ✓ Preprocessor definitions have to use valid identifiers. Dots are not valid in identifiers.
 - ✓ Convention that preprocessor definitions (especially preprocessor macros) use all-uppercase names, to distinguish them from non-preprocessor identifiers.
- Not a hard and fast rule, just convention

25

Constructors

Constructors

- Special `methods` used to initialize data members when objects are created
- A constructor ...
 - ✓ ... is a member function (usually `public`)
 - ✓ ... must have the same name as its class
 - ✓ ... is automatically called when an object is created
 - ✓ ... does not have a return type (not even `void`)

constructors cannot be called as other methods

27

Example

```
class Date {  
    private:  
        int month;  
        int year;  
        int day;  
  
    public:  
        Date();  
        // ...  
};
```

No return
value

28

Example: Date

```
#ifndef DATE_H
#define DATE_H
class Date {
private:
    int month;
    int year;
    int day;

public:
    Date();
    void print();
};
#endif
```

```
#include "date.h"

int main() {
    Date mydate;

    mydate.print();
}
```

```
#include "date.h"
#include <iostream>

Date::Date() {
    month = 1;
    day = 1;
    year = 1970;
}

void Date::print() {
    std::cout << month << '-' <<
    day << '-' << year << '\n';
}
```

```
g++ date.cpp main.cpp -o exec
```

29

Overloading constructors

- A constructor with no parameters is also known as the **default constructor**
- Classes may have multiple constructors
 - constructors are **overloaded** by defining constructors with different parameter lists

```
Date();
Date(int m, int d, int y);
```

30

Synthesized default constructor

- If you don't define any constructor, C++ will define one default constructor for you
- If you define at least one constructor, C++ will not add any other (not even the default constructor)

31

Initialization lists

- C++ allows for optional initialization lists as part of the constructor definition

```
Date::Date(int _d, int _m, int _y) {
    day = _d;
    month = _m;
    year = _y;
    // more statements
}
```

Same as...

```
Date::Date(int _d, int _m, int _y) : day(_d), month(_m), year(_y) {
    // more statements
}
```

32