The C++ Programming Language

- **C++** is an **imperative**, **object-oriented language** and is an extension of the C programming language.
- C++ has a **static type system** and is considered more **strongly typed** than C.
- **Semicolons** terminate statements while **curly braces** are used to group statements into blocks **(block-structured)**.
- Code is organized into **classes and objects** (which are instantiations of classes).
- Applications include operating systems, desktop applications, video games, servers, and performance-critical applications.

a general-purpose programming language good for ‘scaling up’

It also provides a way to create **user-defined types**.

A **class** is a user-defined type.

- A class is defined in C++ using the keyword `class` followed by the name of the class.
- The body of the class is defined inside the curly brackets and terminated by a semicolon.

```cpp
class Student
{
public:

    . . .

private:

    . . .
};
```

access specifier `public`, for the **interface** portion of the class (usually member functions)

access specifier `private`, for the **implementation** portion of the class (usually member variables)

Example of a class

```
class Student
{
public:
    Student() {};
    Student(string name, double GPA) {this->name = name; this->GPA=GPA;};
    string getName() {return name;};
    double getGPA() {return GPA;};
private:
    string name;
    double GPA;
};
```

a default (parameterless) constructor

```
int main(int argc, const char * argv[])
{
    Student s1("Jack", 3.2);

    cout << "Hello " << s1.getName() << endl;
    cout << "Your GPA is: " << s1.getGPA() << endl;

    return 0;
}
```

s1 is an object of type Student
s1 is an instance of the Student class

Another Example of a class

```
#include <iostream>

using namespace std;

class Rectangle
{
public:
    int area() {return width*height;}
    void set_width(int w) {width = w;};
    void set_height(int h) {height = h;};
private:
    int width, height;
};


int main ()
{
    Rectangle rect;
    rect.set_width(3);
    rect.set_height(4);

    cout << "area: " << rect.area() << endl;

    return 0;
}
```

no constructor is declared so the compiler will add a default constructor if needed by the program

```cpp
#include <iostream>

using namespace std;

class Rectangle
{
public:
    int area();
    void set_width(int w) {width = w;};
    void set_height(int h) {height = h;};
private:
    int width, height;
};

int Rectangle::area()
{
    return width*height;
}

int main ()
{
    Rectangle rect;
    rect.set_width(3);
    rect.set_height(4);

    cout << "area: " << rect.area() << endl;

    return 0;
}
```

the scope operator `::`, seen earlier in relation to namespaces, is used in the definition of a member of a class outside the class itself ctors

```
class Rectangle
{
public:
    Rectangle() {width = 0; height = 0;}
    Rectangle(int w, int h) {width = w; height = h;}
    int area();
    void set_width(int w) {width = w;};
    void set_height(int h) {height = h;};
private:
    int width, height;
};

int Rectangle::area()
{
    return width*height;
}
```

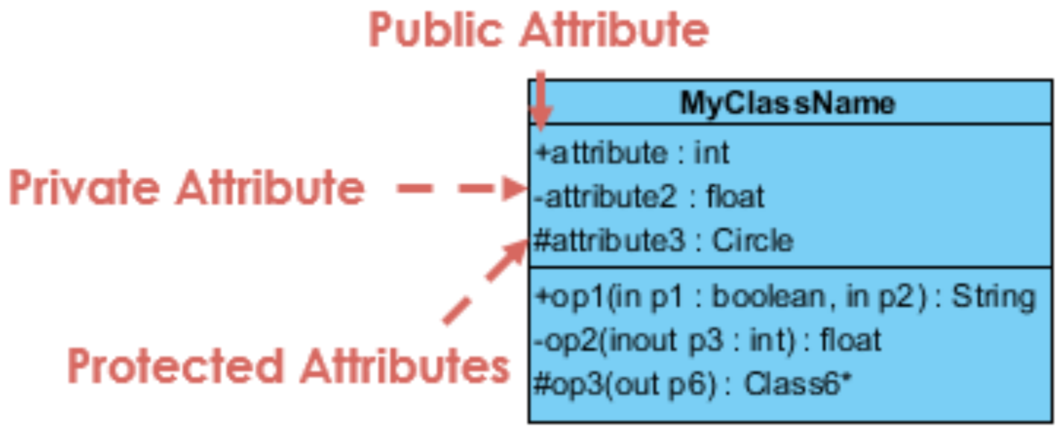
```
class Rectangle
{
public:
    Rectangle() {width = 0; height = 0;}
    Rectangle(int x, int y) : width(x), height(y) { }
    int area();
    void set_width(int w) {width = w;};
    void set_height(int h) {height = h;};
private:
    int width, height;
};

int Rectangle::area()
{
    return width*height;
}

int main ()
{
    Rectangle rect(3,4);

    cout << "area: " << rect.area() << endl;
```

using member initialization

Modeling the Application Domain

What classes do we need?
How should they be defined?

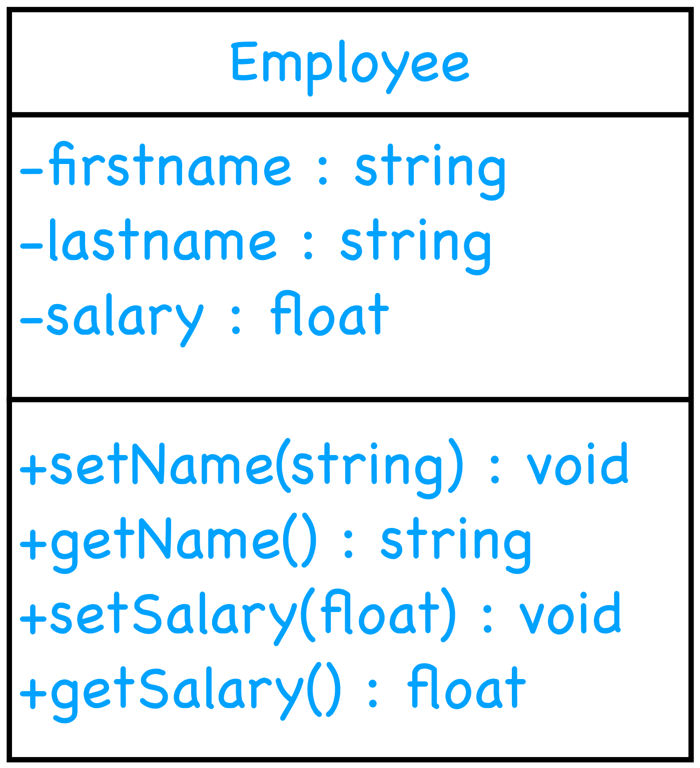
UML Class Diagrams



attributes
(member variables)

operations
(member functions)

How to model an employee?



attributes
(member variables)

operations
(member functions)

How will we use employee objects?

```
#include <iostream>
#include <string>
#include "Employee.hpp"

using namespace std;

int main()
{
    Employee e1("Jack", "Black", 35000.0);
    Employee e2("Tom", "Jones", 25000.0);
    Employee e3;

    cout << e1.GetName() << ' ' << e1.GetSalary() <<endl;
}
```

```cpp
#ifndef Employee_hpp
#define Employee_hpp

#include <string>
using namespace std;

class Employee
{
public:
    Employee(string fn, string ln, float sal):
                first_name(fn), last_name(ln), salary(sal) {};
    Employee():first_name(""), last_name(""), salary(0.0) {};
    string GetName() {return first_name + ' ' + last_name;}
    float GetSalary() {return salary;};
private:
    string first_name;
    string last_name;
    float salary;
};

#endif /* Employee_hpp */
```

```cpp
#include <iostream>
#include <string>
#include "Employee.hpp"

using namespace std;

int main()
{
    Employee e1("Jack", "Black", 35000.0);
    Employee e2("Tom", "Jones", 25000.0);
    Employee* e3 = new Employee("Henry", "Fielding", 88000.0);

    cout << e3->GetName() << ' ' << e3->GetSalary() <<endl;
}
```

where are these objects in memory?