Classes - 2

Data Processing Course, I. Hrivnacova, IPN Orsay

- OOP, Classes – Reminder
- Requirements for a Class
- Class Development
- Constructor
- Access Control Modifiers
- “Getters”, “Setters”
- Keyword this
- const Member Functions
- Destructor
Object-Oriented Programming (OOP)

- The basic unit of OOP is a class
- A class is an implementation of an abstract data type:
  - It describes both the attributes (data) of an object and its operations (methods)
- The OOP languages let you think in the problem space, and use software objects to represent and abstract entities of the problem space to solve the problem.

Chua Hock-Chuan: Programming Notes
A class can be visualized as a three-compartment box, as illustrated:

- **Classname** (or identifier): identifies the class.
- **Data Members** (or variables, attributes, states, fields):
  - contain the static attributes of the class.
- **Member Functions** (or methods, behaviors, operations):
  - contain the dynamic operations of the class.
- In other words, a class encapsulates the static attributes (data) and dynamic behaviors (operations that operate on the data) in a box.
- **Class Members**: The data members and member functions are collectively called class members.
Example of a Class, Instances

- A **class** is a definition of objects of the same kind.
  - **Circle**

- An **instance** is a realization of a particular item of a class.
  - c1, c2, c3

---

Chua Hock-Chuan: Programming Notes
Requirements for a Class

• Before a class is implemented, two issues must be clarified:
  – What should we be able to do with objects of a class?
  – Which data represent objects of this class?

• Requirements for the **Circle** class:
  – It represents a circle of a given size and a color
  – We can get its size (radius, area) and its color
  – Data contained: radius (the area can be computed), color
// The Circle class (All source codes in one file)
#include <iostream>    // using IO functions
#include <string>      // using string
using namespace std;

class Circle {
private:
    double mRadius;   // Data member (Variable)
    string mColor;    // Data member (Variable)

public:
    Circle() {   // Constructor
        mRadius = 0;
        mColor = "red";
    }
    double getRadius() { // Member function (Getter)
        return mRadius;
    }
    string getColor() { // Member function (Getter)
        return mColor;
    }
    double getArea() {   // Member function
        return mRadius * mRadius * 3.1416;
    }
}; // need to end the class declaration with a semi-colon
Using
Class Circle

// 'testCircle' main function
int main() {
    // Construct an instance of Circle c1
    Circle c1;
    cout << "Radius=" << c1.getRadius() << " Area=" << c1.getArea() << " Color=" << c1.getColor() << endl;

    c1.setRadius(2.1);    // Change radius and color of c1
    c1.setColor("blue");
    cout << "Radius=" << c1.getRadius() << " Area=" << c1.getArea() << " Color=" << c1.getColor() << endl;

    // Construct another instance
    Circle c2;
    cout << "Radius=" << c2.getRadius() << " Area=" << c2.getArea() << " Color=" << c2.getColor() << endl;
}

I. Hrivnacova @ Data Processing Course 2019
Using Class Circle

// 'testCircle' main function
int main() {
    // Construct an instance of Circle c1
    Circle c1;
    cout << "Radius= " << c1.getRadius() << " Area= " << c1.getArea()
         << " Color= " << c1.getColor() << endl;

    c1.setRadius(2.1); // Change radius and color of c1
    c1.setColor("blue");
    cout << "Radius= " << c1.getRadius() << " Area= " << c1.getArea()
         << " Color= " << c1.getColor() << endl;

    // Construct another instance using t
    Circle c2;
    cout << "Radius= " << c2.getRadius() << " Area= " << c2.getArea()
         << " Color= " << c2.getColor() << endl;
}

Program output:
> ./testCircle
Radius=0 Area=0 Color=red
Radius=2.1 Area=13.8545 Color=blue
Radius=0 Area=0 Color=red
Constructor

// Constructor
Circle() {
    mRadius = 0;
    mColor = "red";
}

// Construct a Circle instance
Circle c;
    // Take note that there is
    // no empty bracket ()

- A constructor is a special member function that is used to construct the object
- A constructor function is different from an ordinary function in the following aspects:
  - The name of the constructor is the same as the class name.
  - Constructor has no return type. Hence, no return statement is allowed inside the constructor's body.
  - Constructor can only be invoked once to initialize the instance constructed.
- A class may have several constructor
  - The constructor without arguments is called default constructor
// Default constructor
Circle() {
    mRadius = 0;
    mColor = "red";
}

// Constructor with arguments
Circle(double radius, std::string color) {
    mRadius = radius;
    mColor = color;
}

// Construct a Circle instance
Circle c1(1.2, "blue");
Circle c2; // default radius and color
    // Take note that there is
    // no empty bracket ()

**Constructor (2)**

- A class may have several constructor
  - The constructor without arguments is called **default constructor**
Initialization List

- The data members values can be set using so called initialization list
  - It includes all the members of the class, following the same order as their declaration in the class definition

- It is not mandatory, but if it is not done, it is done by the compiler using random values

```cpp
// Constructor with values assignment
Circle(double radius, std::string color) {
    mRadius = radius;
    mColor = color;
}

// Constructor with initialization list
Circle(double radius, string color) :
    mRadius(radius),
    mColor(color)
{
}
```
Access Control Modifiers

- Used to control the visibility of a data member or a member function within a class
  - **public**: the member is accessible and available to all in the system.
  - **private**: The member is accessible and available within this class only.

- You cannot use "c1.mRadius" to refer to c1's radius in main():
  - Try inserting the statement "cout << c1.mRadius;" in main() and observe the error message:
  - error: 'radius' is a private member of 'Circle'

- The “getRadius()” function is declared **public**, hence, it can be invoked in the main()

- Data encapsulation (or information hiding):
  - Objects communicate with each others using well-defined interfaces (public functions).
  - Objects are not allowed to know the implementation details of others. The implementation details are hidden or encapsulated within the class.
  - Information hiding facilitates reuse of the class.

- **Rule of Thumb**: Do not make any data member public, unless you have a good reason.
Class Getters

To allow other to read the value of a private data member, eg. radius in our class Circle, you shall provide a get function, eg. `getRadius()`

- A getter need not expose the data in raw format.
- It can process the data and limit the view of the data others will see.
- Getters shall not modify the data member.

```cpp
// Class getter
double getRadius() {
    return mRadius;
}

// Using getter in main()
Circle c1(1.2, "blue");
cout << "Radius=" << c1.getRadius << endl;
```
Class Setters

// Class setter
void setColor(string color) {
    mColor = color;
}

// Using setter in main()
Circle c1(1.2, "blue");
c1.setColor("red");

• There is no way you can change the radius or color of a Circle object, after it is constructed in main(), as they are declared as private.

• To allow other classes to modify the value of a private data member, eg. color in our class Circle, you shall provide a set function, eg. setColor()
  – A setter could provide data validation (such as range checking), and transform the raw data into the internal representation.
Data Members Naming

- Using an agreed prefix, eg. m, with the data member names
  - Makes data members "visible" withing the class implementation
  - Avoids the ambiguity between the names of data member and function parameter.

```cpp
class Circle {
private:
    string mColor;  // Member variable
...
public:
    void setColor(string color) {
        mColor = color;
    }
...}
```
Keyword "this"

```cpp
class Circle {
private:
    string color; // Member variable called "color"
...
public:
    void setColor(string color) {
        // Function's argument also called "color"
        this->color = color;
        // "this->color" refers to this instance's member variable
        // "color" refers to the function's argument.
    }
    ...
}
```

- One can use keyword "this" to refer to this instance inside a class definition.
  - It can be used when we need to refer to the class object.
- "this" is actually a pointer to the object of the class.
"const" Member Functions

- Member function declared with a **const** keyword cannot modify any data member of this object.
- The getters can be always declared constant.

```cpp
double getRadius() const {  // const member function
    mRadius = 0;
    // error: cannot assign to non-static data member
    // within const member function 'getRadius'
    return mRadius;
}

double getArea() const {  // const member function
    return mRadius*mRadius*3.1416;
}
```
Destructor

- A destructor is a special function that has the same name as the classname, with a prefix \(~\), e.g., \(~\text{Circle}()\).
- Destructor is called implicitly when an object is destroyed.
- If you do not define a destructor, the compiler provides a default, which does nothing.
- If your class contains data members which are dynamically allocated (via \text{new} or \text{new[]} operator), you need to free the storage via delete or delete[].

```cpp
class Circle {
    public:
        ... 
        // destructor
        ~Circle() {}
    ...
};
```
Separating Header and Implementation

- We shall "separate the interface and implementation" by placing the codes in 3 files:
  - Circle.h: defines the public interface of the Circle class.
  - Circle.cxx: provides the implementation of the Circle class.
  - testCircle.cxx: a test driver program for the Circle class.
#include <string>
using namespace std;

// Circle class declaration
class Circle {
    public:
        // Constructor with arguments
        Circle(double radius, string color);
        // Default constructor
        Circle();

        // Public getters & setters for private data members
        double getRadius() const;
        void setRadius(double radius);
        string getColor() const;
        void setColor(string color);

        // Public member Function
        double getArea() const;

    private:
        // private data members (variables)
        double mRadius;
        string mColor;
};
// user-defined header in the same directory
#include "Circle.h"

// Constructor with arguments
Circle::Circle(double radius, string color)
  : mRadius(radius),
    mColor(color)
{
  ...

// Public getter for private data member mRadius
double Circle::getRadius() const {
  return mRadius;
}

// Public setter for private data member mRadius
void Circle::setRadius(double radius) {
  mRadius = radius;
  }

// A public member function
double Circle::getArea() const {
  return mRadius*mRadius*3.14159265;
}
```cpp
#include <iostream>
#include "Circle.h"   // using Circle class
using namespace std;

// 'testCircle' main function
int main() {
    // Construct an instance of Circle c1
    Circle c1;
    cout << "Radius=" << c1.getRadius() << " Area=" << c1.getArea()
         << " Color=" << c1.getColor() << endl;

    c1.setRadius(2.1);   // Change radius and color of c1
    c1.setColor("blue");
    cout << "Radius=" << c1.getRadius() << " Area=" << c1.getArea()
         << " Color=" << c1.getColor() << endl;

    // Construct another instance using the default constructor
    Circle c2;
    cout << "Radius=" << c2.getRadius() << " Area=" << c2.getArea()
         << " Color=" << c2.getColor() << endl;
}
```