Lecture 11_6.1

Constructors, Destructors and Abstract Classes

Alessandro SAVINO
Politecnico di Torino (Italy)

alessandro.savino@polito.it

www.testgroup.polito.it

License Information

Disclaimer

- **We disclaim any warranties or representations as to the accuracy or completeness of this material.**
- **Materials are provided "as is" without warranty of any kind, either express or implied, including without limitation, warranties of merchantability, fitness for a particular purpose, and non-infringement.**
- **Under no circumstances shall we be liable for any loss, damage, liability or expense incurred or suffered which is claimed to have resulted from use of this material.**

Goal

- This lecture presents a deeper view about C++ classes and objects

Prerequisites

– **A basic knowledge about classes**

Homework

– **None**

Outline

- **Constructor and Destructor**
- **Abstract Classes**

Constructor

- **Definition:**
- **A constructor is a special member function whose task is to initialize the objects of its class.**
  - **It is special because its name is same as the class name.**
  - **The constructor is invoked whenever an object of its associated class is created.**
  - **It is called constructor because it should construct the values of data members of the class.**

Constructor

- **Example:**

```
class Rectangle {
public:
     Rectangle();
     ...
private:
     double _width;
     double _length;
};
```

- It is special because its name is same as the class name.

Constructor

- **Example:**

```
Rectangle::Rectangle()
{
      _width = 1;
      _length = 1;
}
```

- It is called constructor because it should construct the values of data members of the class.

Constructor

- **Example:**

```
int main() {
      Rectangle rect;
      ...
}
```

```
Rectangle::Rectangle()
{
      _width = 1;
      _length = 1;
}
```

- The constructor is invoked whenever an object of its associated class is created.

Constructor

- **Properties:**
  - **There is no need to write any statement to invoke the constructor function.**
    - **If a 'normal' member function is defined for zero initialization, we would need to invoke this function for each of the objects separately.**
  - **A constructor that accepts no parameters is called the default constructor.**
  - **If you write a class with no constructor at all, C++ will write a default constructor for you, one that does nothing.**

Constructor

- **Characteristics:**
  - **They should be declared in the public section.**
  - **They do not have return types, not even void and they cannot return values.**
  - **They cannot be inherited, though a derived class can call the base class constructor.**
  - **Like other C++ functions, Constructors can have default arguments.**
  - **Constructors can not be virtual.**

Constructor

- **Parameters:**
  - **To create a constructor that takes arguments:**
    1. **indicate parameters in prototype (.h).**

```
class Rectangle {
public:
     Rectangle(double w, double l);
     ...
```


Constructor

- **Parameters:**
  - **To create a constructor that takes arguments:**
    1. **indicate parameters in prototype (.h).**
    2. **Use parameters in the implementation (.cpp).**

```
Rectangle::Rectangle(double w, double l)
{
    _width = w;
    _length = len;
}
```


Constructor

- **Parameters:**
  - **To create a constructor that takes arguments:**
    1. **indicate parameters in prototype (.h).**
    2. **Use parameters in the implementation (.cpp).**
    3. **Pass the arguments to the constructor when you create an object.**

```
int main() {
     Rectangle rect(5.4, 7.8);
     ...
}
```


Constructor

- **Parameters:**
  - **If all constructor's parameters have default arguments, then you defined a default constructor**

```
class Rectangle {
public:
     Rectangle(double w=1, double l=1);
     ...
```

```
int main() {
     Rectangle rect(5.4, 7.8);
     Rectangle rect2;
     ...
```


Multiple Constructors

- **Multiple constructors:**
  - **You may need different kind of constructors**
  - **You can create as many as you like**
    - **They must differ in parameters!**

```
class Rectangle {
public:
     Rectangle();
     Rectangle(double w, double l);
     Rectangle(double w_l);
     ...
```


Destructor

- **A destructor is used to destroy the objects that have been created by a constructor.**
- **Like constructor, the destructor is a member function whose name is the same as the class name but is preceded by a tilde.**

```
class Rectangle {
public:
     Rectangle();
     ~Rectangle();
     ...
```


Destructor

- **A destructor never takes any argument nor does it return any value.**
- **It will be invoked implicitly by the compiler upon exit from the program – or block or function as the case may be – to clean up storage that is no longer accessible.**

```
class Rectangle {
public:
     Rectangle();
     ~Rectangle();
     ...
```


Destructor

- **Example:**

```
class Rectangle {
public:
     Rectangle();
     ~Rectangle();
     ...
private:
     double width, length;
};
...
Rectangle::~Rectangle() {}
```


Destructor

- **Example:**

```
class Re
public:
      Re

pri

};
.
angle::~Rectangle() {}
```

If you do not have a dynamic memory management in your class, you need an empty destructor...

Outline

- Constructor and Destructor
- Abstract classes

Abstract Classes

- **The goal of object-oriented programming is to divide a complex problem into small sets. This helps understand and work with problem in an efficient way.**
- **C++, you can create an abstract class that cannot be instantiated (you cannot create object of that class). Abstract classes are the base class which cannot be instantiated.**
- **A class containing pure virtual function is known as abstract class.**

Virtual Keyword

- **A virtual function or virtual method is an inheritable and override-able function or method.**

Virtual Keyword

- **Concepts:**

```
class Rectangle {
public:
     ...
     double getArea();
     double getPerimeter();
private:
     double width, length;
};
```


Virtual Keyword

- **Concepts:**

```
class Circle {
public:
     ...
     double getArea();
     double getPerimeter();
private:
     double radius;
};
```


Virtual Keyword

- **Concepts:**

```
class                {
public:
     ...
     double getArea();
     double getPerimeter();
private:
                    
};
```

This part is generic... but works in different ways!

Virtual Keyword

- **Concepts:**

```
class Shape {
public:
     ...
     virtual double getArea() = 0;
     virtual double getPerimeter() = 0;
};
```


Virtual Keyword

- **Concepts:**

```
class Shape {
public:
     ...
     virtual double getArea() = 0;
     virtual double getPerimeter() = 0;
};
```

It defines a pure virtual function

Abstract Classes - Usages

- **Usage (.h):**

```
class Rectangle : public Shape{
public:
     ...
     double getArea();
     double getPerimeter();
     ...
};
```


Abstract Classes - Usages

- **Usage (.h):**

```
class Rectangle : public Shape{
public:
      ...
      double getArea();
      double getPerimeter();
      ...
};
```

This way I'm avoiding classes that will derive from Shape that do not implement those methods... compiler gives you errors if you forget the implementation!

Abstract Classes - Usages

- **Usage (.cpp):**

```
double Rectangle::getArea() {
...
}
double Rectangle::getPerimeter() {
...
}
```



Малые Автюхи, Калинковичский район, Республики Беларусь