CONSTRUCTOR (OBJECT ORIENTED PROGRAMMING)

Nishant malik

Abstract

In class-based object-oriented programming, a **constructor** (abbreviation: **ctor**) in a class is a special type of subroutine called to create an object. It prepares the new object for use, often accepting arguments that the constructor uses to set required member variables.

A constructor resembles an instance method, but it differs from a method in that it has no explicit return type, it is not implicitly inherited and it usually has different rules for scope modifiers. Constructors often have the same name as the declaring class. They have the task of initializing the object's data members and of establishing the invariant of the class, failing if the invariant is invalid. A properly written constructor leaves the resulting object in a *valid* state. Immutable objects must be initialized in a constructor.

1. Types

(a) Parameterized constructors

Constructors that can take arguments are termed as parameterized constructors. The number of arguments can be greater or equal to one(1). For example:

```
class Example
{
    int x, y;
   public:
    Example();
    Example(int a, int b); // Parameterized
constructor
};
Example :: Example()
{
}
Example :: Example(int a, int b)
{
    x = a;
    y = b;
    }
```

When an object is declared in a parameterized constructor, the initial values have to be passed as arguments to the constructor function. The normal way of object declaration may not work. The constructors can be called explicitly or implicitly. The method of calling the constructor implicitly is also called the *shorthand* method.

(b) Default constructors

If the programmer does not supply a constructor for an instantiable class, most languages will provide a *default constructor*.

The behavior of the default constructor is language dependent. It may initialize data members to zero or other same values, or it may do nothing at all.

Example:

```
#include <iostream>

struct not_default_constructible {
        not_default_constructible() = delete; //
delete default constructor
        not_default_constructible(int x) { std::cout
<< "Constructed with " << x << '\n'; }
};

int main() {
        not_default_constructible static_array[] =
                { 1, 2, 3 };
        not_default_constructible *dynamic_array
=
                new
not_default_constructible[3]{ 4, 5, 6 }; // C++11
}
```

(c) Copy constructor

Copy constructors define the actions performed by the compiler when copying class objects. A copy constructor has one formal parameter that is the type of the class (the parameter may be a reference to an object). It is used to create a copy of an existing object of the same class. Even though both classes are the same, it counts as a conversion constructor.

While copy constructors are usually abbreviated copy ctor or cctor, they have nothing to do with *class constructors* used in .NET using the same abbreviation.

2. Syntax

(a) Java, C++, C#, ActionScript, and PHP 4 have a naming convention in which constructors have the same name as the class of which they are associated with.

(b) In Objective-C, the constructor method is split across two methods, "alloc" and "init" with the alloc method setting aside (allocating) memory for an instance of the class, and the init method handling the bulk of initializing the instance. A call to the method "new" invokes both the alloc and the init methods, for the class instance.

3. Language details

(a) Java

In Java, constructors differ from other methods in that:

- Constructors never have an explicit return type.
- Constructors cannot be directly invoked (the keyword “new” invokes them).
- Constructors cannot be *synchronized*, *final*, *abstract*, *native*, or *static*.

(b) C++

In C++, the name of the constructor is the name of the class. It returns nothing. It can have parameters like any member function. Constructor functions are usually declared in the public section, but can also be declared in the protected and private sections, if the user wants to restrict access to them.

The constructor has two parts. First is the initializer list which follows the parameter list and before the method body. It starts with a colon and entries are comma-separated. The initializer list is not required, but offers the opportunity to provide values for data members and avoid separate assignment statements.

4. Reference

https://en.wikipedia.org/wiki/Constructor_(object-oriented_programming)