LivingObjects

COLLABORATORS

	TITLE : LivingObjects		
ACTION	NAME	DATE	SIGNATURE
WRITTEN BY		September 19, 2022	

REVISION HISTORY

NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

LivingObjects

1.1 Implementation notes

```
                        The A++ Living Object class
                --------------------------------
                  ($Date: 1994/05/09 21:25:07 $)
```

The Living Object class, brief LvObject, makes a C++ object become a child
task of the task that creates the object.

The created task executes a virtual method declared in LvObject that can
easily be overwritten with your special 'main' routine in which you have
access to all member data of your LvObject derived class.
The child task can be started after member initialisation within the
constructor with method call 'activate()'.

All member data is initialised by the constructor and usually not accessable
for the creator task since it is declared 'private' or 'protected'.
Access to exceptionally public data should be synchronized with semaphores.


```
            How to derive a specialized LvObject class

            How to create and delete a LvObject
            ----------------------------------------------------------
  -> Back to the root menu..
```

1.2 lvobject_1

Your derived class has to specify any additional class member variables along
with a constructor for their initialisation.

Let's have a closer look at this example:

```
    class MyLvObject : public LvObject
    {
```

```
    private:
        int data1;

    protected:
        long data2;
        void main()
        {
            cout << "I'm alive!\n";
            cout << data1 << data2;
            data2 = data1;
        }
    public:
        MyLvObject(int a, long b) : data1(a), data2(b) { /**/ }

};

void main()
{
    MyLvObject test;
    test.activate();
}
```

There are some important rules that have to be obeyed when defining your personal LvObject:

o Your constructor needs to initialise your additional member variables
   BEFORE the LvObject task is started with activate()! From this moment on
   the object belongs to this task. You MUST NEVER execute any code that
   writes object data or reads non-constant members!

o Public member methods are considered to be applicable for the creator
   task. Therefore any access to the object within these methods has to be
   synchronized to the LvObject's task via semaphores or message passing.

1.3 lvobject_2

Creating an LvObject hardly differs from creating any other object. But
notice the slightly changed semantics: The constructor arguments now function
as startup messages to the newly created task.

Consider the example before:

```
    main()
    {
        MyLvObject task1(2,80000), task2(4,67000);
        task1.activate(); task2.activate();
        // now two child task are created.

        // do something
        ....
    } // implicit destructor call before leaving this scope
```

But how do those child tasks terminate?
They terminate when the LvObject::main() method is left.

The creator task waits for their termination at the end of its own main() routine since the destructor of the LvObject waits for the LvObject task to finish before destructing itself.

So, the LvObject destructor will block until the LvObject::main() method is finished.