THALASSA CODECHIP™

Object Oriented Programming by C

Overview

This document discuss the way to implement:

- Inheritance
- Polymorphism

And `Duck` simple project is used to show how OOP can be accomplished by C.

Overview of this sample

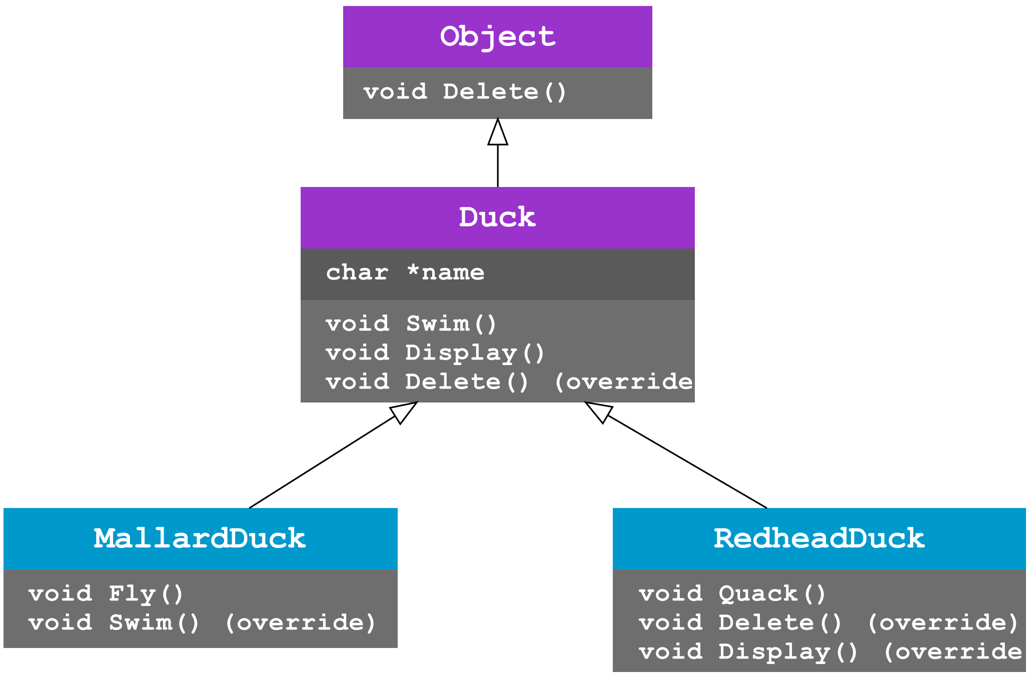

Figure 2-1 - Duck class diagram

We implement classes depicted in above class diagram. In this diagram:

- The `Object` class is the top abstract class that defines all of common methods that one object should have. And in this example, just the `Delete` method is defined and its purpose is to delete an object.
- The `Duck` class is also an abstract class to abstract all of generic attributes and methods that one duck should have. This class may have private attributes(`name`, for example), so it overrides `Delete` method of `Object` class which it extends from to delete additional private data when it is deleted. Otherwise, memory leak will happen.
- The `Duck` class also defines two additional methods: `Swim()` and `Display()`.
- `MallardDuck` defines new method `Fly()` and overrides `Swim()` of `Duck`.
- `RedheadDuck` defines new method `Quack()` and override `Display()` of `Duck`. It also override `Delete()` of `Object` class. This is to show how to implement overriding of more than one levels.

How classes are used

```
#include <stdio.h>
#include <string.h>

#include "MallardDuck.h"
#include "RedheadDuck.h"

static void TestDuckWithName(Duck duck, const char *duckName)
    {
    DuckNameSet(duck, (char*)duckName, strlen(duckName));

    /* Introduce */
    printf("\n============================================================\n");
    printf("Duck name: %s\n", DuckNameGet(duck, NULL));
    printf("============================================================\n");

    /* Run */
    DuckSwim(duck);
    DuckDisplay(duck);
    }

static void TestMallardDuck(MallardDuck duck)
    {
    MallardDuckFly(duck);
    }

static void TestRedheadDuck(RedheadDuck duck)
    {
    RedheadDuckQuack(duck);
    }

int main(int argc, char ** argv)
    {
    Duck duck;

    /* Mallard duck */
    duck = (Duck)MallardDuckNew();
    TestDuckWithName(duck, "MallardDuck");
    TestMallardDuck((MallardDuck)duck);
    ObjectDelete((Object)duck);

    /* Redhead duck */
    duck = (Duck)RedheadDuckNew();
    TestDuckWithName(duck, "RedheadDuck");
    TestRedheadDuck((RedheadDuck)duck);
    ObjectDelete((Object)duck);

    return 0;
    }
```

The **TestDuckWithName** is the function used to test a **Duck** object. It does not care whether a **Duck** is **MallardDuck** or **RedheadDuck**. So it can be used to test **MallardDuck** and **RedheadDuck** because they extends from **Duck** class. This is to show inheritance. That mean, all of codes for **Duck** are applicable for **MallardDuck** and **RedheadDuck**.

In the test codes, only one generic **Duck** variable is declared and its behaviors will be changed every time it is assigned to any **Duck** which can be **MallardDuck** and **RedheadDuck**. This is called polymorphism, that means behavior of object is changed at runtime.

Let see how that program outputs:

```
============================================================
Duck name: MallardDuck
============================================================
(MallardDuck.Swim): Swim in different way
(Duck.Display): My name is: MallardDuck
```

```
(MallardDuck.Fly): Fly
(Duck.Delete): Delete name
(Object.Delete): Delete

============================================================
Duck name: RedheadDuck
============================================================
(Duck.Swim): Swim
(RedheadDuck.Display): My name is RedheadDuck and I can quack
(RedheadDuck.Quack): Quack
(RedheadDuck.Delete): Delete
(Duck.Delete): Delete name
(Object.Delete): Delete
```

Every **Duck** introduces its name, the **name** attribute is defined at **Duck** class and available for all subclasses.

The **Duck** has its own default implementation. It override the **Delete** method of **Object** class so it deletes its internal data and call super implementation of **Delete()** to fully delete itself. So we have the output of **Delete()**.

```
(Duck.Delete): Delete   --> Delete internal data
(Object.Delete): Delete --> Output of super
```

The **MallardDuck** overrides **Swim()** of **Duck** and it produces different output. It also supports new method **Fly()**. What different between its output and **Duck**'s output are:

```
(MallardDuck.Swim): Swim in different way --> Overriding
(MallardDuck.Fly): Fly                    --> New method
```

MallardDuck inherits implementation of **Delete** of **Duck** so output of **Delete()** is the same as **Duck**.

The **RedheardDuck** is more complex, it override methods in **Duck** and **Object** class.

```
(RedheadDuck.Display): My name is RedheadDuck and I can quack --> Overriding
(RedheadDuck.Quack): Quack                                    --> New method
```

It overrides **Delete** method of **Object** class, its **Delete()** implementation will first delete internal data and then call super implement to fully destroy itself. So we have the output:

```
(RedheadDuck.Delete): Delete --> Delete internal data

// Output of super
(Duck.Delete): Delete
(Object.Delete): Delete
```

Recommend implementation

How class is represented

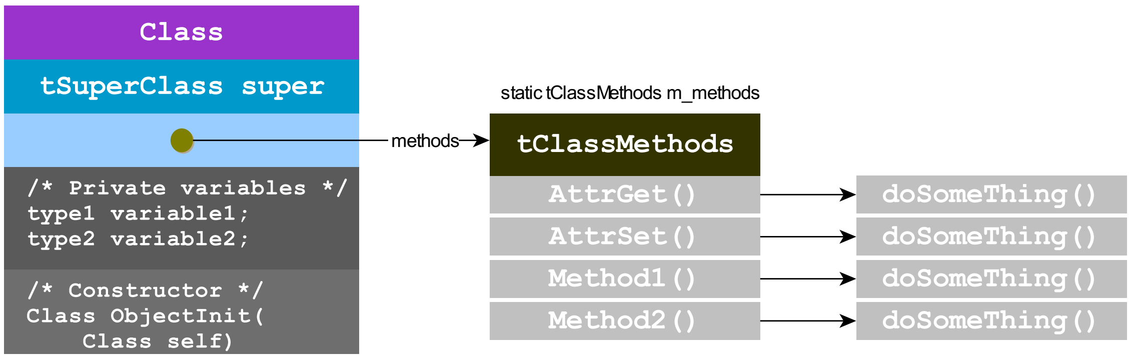

Figure 4-1 - Class representation

All of attributes must be encapsulated and accessed via SET/GET function.

A class is declared by using C structure, it stores a pointer pointing to implementation of class. So, all of instances of one class will have the same behavior. This structure also stores private variables which are only visible for this class but not for subclasses.

A class has one constructor **`ObjectInit(Class self)`** which is to initialize private variables and make the **`methods`** pointer point to correct **`ClassMethods`**.

How class is implemented

One class should be implemented with 3 files:

Class.h - Interface with the outside

```
/*-----------------------------------------------------------------------------
 *
 * COPYRIGHT (C) 2014 Arrive Technologies Inc.
 *
 * The information contained herein is confidential property of Arrive Technologies.
 * The use, copying, transfer or disclosure of such information
 * is prohibited except by express written agreement with Arrive Technologies.
 *
 * Module      : XXX
 *
 * File        : Class.h
 *
 * Created Date: MMM dd, yyyy
 *
 * Description : XXX
 *
 * Notes       :
 *----------------------------------------------------------------------------*/

#ifndef _CLASS_H_
#define _CLASS_H_

/*--------------------------- Includes ----------------------------------------*/
#include "SuperClass.h" /* To have all super class interfaces */

/*--------------------------- Define ------------------------------------------*/

/*--------------------------- Macros ------------------------------------------*/

/*--------------------------- Typedefs ----------------------------------------*/
typedef struct tClass * Class;
typedef void * ReturnType;
typedef void * AttrType;
typedef void * Type1;

/*--------------------------- Forward declarations ---------------------------*/

/*--------------------------- Entries ----------------------------------------*/
/* New instance of Class. */
Class ClassNew();

/* Accessors of attributes */
ReturnType ClassAttrSet(Class self, AttrType attrValue);
AttrType ClassAttrGet(Class self);

/* Methods */
ReturnType ClassMethod1(Class self, Type1 param1, Type1 param2);
ReturnType ClassMethod2(Class self, Type1 param);

#endif /* _CLASS_H_ */
```

ClassInternal.h - Class representation

Its common content is:

```
/*----------------------------------------------------------------------------
 *
 * COPYRIGHT (C) 2014 Arrive Technologies Inc.
 *
 * The information contained herein is confidential property of Arrive Technologies.
 * The use, copying, transfer or disclosure of such information
 * is prohibited except by express written agreement with Arrive Technologies.
 *
 * Module      : XXX
 *
 * File        : ClassInternal.h
 *
 * Created Date: MMM dd, yyyy
 *
 * Description : XXX
 *
 * Notes       :
 *----------------------------------------------------------------------------*/

#ifndef _CLASSINTERNAL_H_
#define _CLASSINTERNAL_H_

/*--------------------------- Includes ---------------------------------------*/
#include "SuperClassInternal.h" /* To have super representation */
#include "Class.h"

/*--------------------------- Define -----------------------------------------*/

/*--------------------------- Macros -----------------------------------------*/

/*--------------------------- Typedefs ---------------------------------------*/
/* Methods */
typedef struct tClassMethods
    {
    /* Accessors of attributes */
    ReturnType (*AttrSet)(Class self, AttrType attrValue);
    AttrType (*AttrGet)(Class self);

    /* Methods */
    ReturnType (*Method1)(Class self, Type1 param1, Type1 param2);
    ReturnType (*Method2)(Class self, Type1 param);
    }tClassMethods;

/* Class representation */
```

```
typedef struct tClass
    {
    /* Meta */
    tSuperClass super;
    const tClassMethods *methods;

    /* Private variables */
    AttrType attr1;
    AttrType attr2;

    /* So on... */
    }tClass;

/*--------------------------- Forward declarations ---------------------------*/

/*--------------------------- Entries ----------------------------------------*/
/* Constructor */
Class ClassObjectInit(Class self);

#endif /* _CLASSINTERNAL_H_ */
```

Class.c - Class implementation

Its common content looks like.

```
/*-----------------------------------------------------------------------------
 *
 * COPYRIGHT (C) 2014 Arrive Technologies Inc.
 *
 * The information contained herein is confidential property of The Arrive Technologies
 * The use, copying, transfer or disclosure of such information is prohibited
 * except by express written agreement with Arrive Technologies.
 *
 * Module      : XXX
 *
 * File        : Class.c
 *
 * Created Date: MMM dd, yyyy
 *
 * Description : XXX
 *
 * Notes       :
 *----------------------------------------------------------------------------*/

/*--------------------------- Include files ----------------------------------*/
#include <stdlib.h>
#include <string.h>
#include "ClassInternal.h"

/*--------------------------- Define -----------------------------------------*/

/*--------------------------- Macros -----------------------------------------*/

/*--------------------------- Local typedefs ---------------------------------*/

/*--------------------------- Global variables -------------------------------*/

/*--------------------------- Local variables --------------------------------*/
/* Implementation of this class */
static tClassMethods m_methods;
static char m_methodsInit = 0;

/* Override */
static tSuperClass1Methods m_SuperClass1Override;
static tSuperClass2Methods m_SuperClass2Override;

/* Save super implementation */
static const tSuperClass1Methods *m_SuperClass1Methods = NULL;
static const tSuperClass2Methods *m_SuperClass2Methods = NULL;
```

```
/*--------------------------- Forward declarations ---------------------------*/

/*--------------------------- Implementation ---------------------------------*/
static ReturnType AttrSet(Class self, AttrType attrValue)
    {
    /* Do some thing */
    return NULL;
    }

static AttrType AttrGet(Class self)
    {
    /* Do some thing */
    return NULL;
    }

static ReturnType Method1(Class self, Type1 param1, Type1 param2)
    {
    /* Do some thing */
    return NULL;
    }

static ReturnType Method2(Class self, Type1 param)
    {
    /* Do some thing */
    return NULL;
    }

static void Method1OfSuperClass1(SuperClass1 self)
    {
    /* Do some thing... */

    /* And reuse super implementation */
    m_SuperClass1Methods->Method1OfSuperClass1(self);
    }

static void Method1OfSuperClass2(SuperClass2 self)
    {
    /* Do some thing... */

    /* And reuse super implementation */
    m_SuperClass2Methods->Method1OfSuperClass2(self);
    }

static void OverrideSuperClass1(Class self)
    {
    SuperClass1 class1 = (SuperClass1)self;

    if (!m_methodsInit)
        {
        m_SuperClass1Methods = class1->methods;

        /* Copy to reuse implementation of super class. But override some methods */
        memcpy(&m_SuperClass1Override, m_SuperClass1Methods, sizeof(m_SuperClass1Override));
        m_SuperClass1Override.Method1OfSuperClass1 = Method1OfSuperClass1;
        }

    /* Change behavior of super class */
    class1->methods = &m_SuperClass1Override;
    }

static void OverrideSuperClass2(Class self)
    {
    SuperClass2 class2 = (SuperClass2)self;

    if (!m_methodsInit)
        {
        /* Save reference to super implementation */
        m_SuperClass2Methods = class2->methods;

        /* Copy to reuse implementation of super class. But override some methods */
        memcpy(&m_SuperClass2Override, m_SuperClass2Methods, sizeof(m_SuperClass2Override));

        m_SuperClass2Override.Method1OfSuperClass2 = Method1OfSuperClass2;
        }

    /* Change behavior of super class */
    class2->methods = &m_SuperClass2Override;
    }

static void Override(Class self)
    {
    OverrideSuperClass1(self);
    OverrideSuperClass2(self);
    }

static void MethodsInit(Class self)
    {
    if (!m_methodsInit)
        {
        memset(&m_methods, 0, sizeof(m_methods));

        m_methods.AttrGet = AttrGet;
        m_methods.AttrSet = AttrSet;
        m_methods.Method1 = Method1;
        m_methods.Method2 = Method2;
        }

    self->methods = &m_methods;
    }

static int ObjectSize()
    {
    return sizeof(tClass);
    }

Class ClassObjectInit(Class self)
    {
    memset(self, 0, ObjectSize());

    if (SuperClassObjectInit((SuperClass)self) == NULL)
        return NULL;

    /* Setup class */
    Override(self);
    MethodsInit(self);
    m_methodsInit = 1;

    /* Private data may be setup in this context */

    return self;
    }

Class ClassNew()
    {
    Class newObject = malloc(ObjectSize());
    return ClassObjectInit(newObject);
    }

ReturnType ClassAttrSet(Class self, AttrType attrValue)
    {
    if (self)
        return self->methods->AttrSet(self, attrValue);
    return NULL;
    }

AttrType ClassAttrGet(Class self)
    {
    if (self)
        return self->methods->AttrGet(self);
    return NULL;
    }
```

```
ReturnType ClassMethod1(Class self, Type1 param1, Type1 param2)
    {
    if (self)
        return self->methods->Method1(self, param1, param2);
    return NULL;
    }

ReturnType ClassMethod2(Class self, Type1 param)
    {
    if (self)
        return self->methods->Method2(self, param);
    return NULL;
    }
```

Implement Duck sample project

Object - Root class

Object.h - Publish header file

```
/*-----------------------------------------------------------------------------
 *
 * COPYRIGHT (C) 2012 Arrive Technologies Inc.
 *
 * The information contained herein is confidential property of Arrive Tecnologies.
 * The use, copying, transfer or disclosure of such information
 * is prohibited except by express written agreement with Arrive Technologies.
 *
 * Module      : OOP
 *
 * File        : Object.h
 *
 * Created Date: Aug 16, 2012
 *
 * Author      : namnn
 *
 * Description : Root class
 *
 * Notes       :
 *----------------------------------------------------------------------------*/

#ifndef _OBJECT_H_
#define _OBJECT_H_

/*--------------------------- Includes ---------------------------------------*/

/*--------------------------- Define -----------------------------------------*/

/*--------------------------- Macros -----------------------------------------*/

/*--------------------------- Typedefs ---------------------------------------*/
typedef struct tObject * Object;

/*--------------------------- Forward declarations ---------------------------*/

/*--------------------------- Entries ----------------------------------------*/
void ObjectDelete(Object self);

#endif /* _OBJECT_H_ */
```

ObjectInternal.h - Class representation (private)

```
/*-----------------------------------------------------------------------------
 *
 * COPYRIGHT (C) 2012 Arrive Technologies Inc.
 *
 * The information contained herein is confidential property of Arrive Tecnologies.
 * The use, copying, transfer or disclosure of such information
 * is prohibited except by express written agreement with Arrive Technologies.
 *
 * Module      : TODO module name
 *
 * File        : ObjectInternal.h
 *
 * Created Date: Aug 16, 2012
 *
 * Author      : namnn
 *
 * Description : TODO Description
 *
 * Notes       :
 *---------------------------------------------------------------------------*/

#ifndef _OBJECTINTERNAL_H_
#define _OBJECTINTERNAL_H_

/*--------------------------- Includes ---------------------------------------*/
#include "Object.h"

/*--------------------------- Define -----------------------------------------*/

/*--------------------------- Macros -----------------------------------------*/
#define mImplement(object) object->implement
#define mSuper(object)  (&(object->super))

/*--------------------------- Typedefs ---------------------------------------*/
/* Implementation of Object class. */
typedef struct tObjectImplement
    {
    void (*Delete)(Object self);
    }tObjectImplement;

/* Object class representation */
typedef struct tObject
    {
    const tObjectImplement *implement;
    }tObject;

/*--------------------------- Forward declarations ---------------------------*/

/*--------------------------- Entries ----------------------------------------*/
/* Constructor */
Object ObjectInit(Object self);

#endif /* _OBJECTINTERNAL_H_ */
```

Object.c - Implementation

```
/*----------------------------------------------------------------------------
 *
 * COPYRIGHT (C) 2012 Arrive Technologies Inc.
 *
 * The information contained herein is confidential property of The Arrive Technologies
 * The use, copying, transfer or disclosure of such information is prohibited
 * except by express written agreement with Arrive Technologies.
 *
 * Module      : OOP
 *
 * File        : Object.c
 *
 * Created Date: Aug 16, 2012
 *
 * Author      : namnn
 *
 * Description : Root class implementation
 *
 * Notes       :
 *---------------------------------------------------------------------------*/

/*--------------------------- Include files ----------------------------------*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>

#include "ObjectInternal.h"

/*--------------------------- Define -----------------------------------------*/

/*--------------------------- Macros -----------------------------------------*/
```

```
/*--------------------------- Global variables -------------------------------*/

/*--------------------------- Local variables --------------------------------*/
static tObjectMethods m_methods;
static char m_methodsInit = 0;

/*--------------------------- Forward declarations ---------------------------*/

/*--------------------------- Implementation ---------------------------------*/
static void Delete(Object self)
    {
    printf("(Object.Delete): Delete\n");
    free(self);
    }

static void MethodsInit(Object self)
    {
    if (!m_methodsInit)
        {
        memset(&m_methods, 0, sizeof(m_methods));

        mMethodOverride(m_methods, Delete);
        }

    mMethodsSet(self, &m_methods);
    }

static int ObjectSize()
    {
    return sizeof(tObject);
    }

Object ObjectInit(Object self)
    {
    memset(self, 0, ObjectSize());

    /* Setup class */
    MethodsInit(self);
    m_methodsInit = 1;

    return self;
    }

void ObjectDelete(Object self)
    {
    if (self)
        mMethodsGet(self)->Delete(self);
    }
```

Duck - A duck that can swim and display

Duck.h - Publish header file

```
/*-----------------------------------------------------------------------------
 *
 * COPYRIGHT (C) 2012 Arrive Technologies Inc.
 *
 * The information contained herein is confidential property of Arrive Tecnologies.
 * The use, copying, transfer or disclosure of such information
 * is prohibited except by express written agreement with Arrive Technologies.
 *
 * Module      : OOP
 *
 * File        : Duck.h
 *
 * Created Date: Aug 16, 2012
 *
 * Author      : namnn
 *
 * Description : Duck generic class
 *
 * Notes       :
 *----------------------------------------------------------------------------*/

#ifndef _DUCK_H_
#define _DUCK_H_

/*--------------------------- Includes ----------------------------------------*/
#include "Object.h"

/*--------------------------- Define ------------------------------------------*/

/*--------------------------- Macros ------------------------------------------*/

/*--------------------------- Typedefs ----------------------------------------*/
typedef struct tDuck * Duck;

/*--------------------------- Forward declarations --------------------------*/

/*--------------------------- Entries -----------------------------------------*/
Duck DuckNew();

/* Attributes */
void DuckNameSet(Duck self, char *name, int len);
char *DuckNameGet(Duck self, int *len);

/* Methods */
void DuckSwim(Duck self);
void DuckDisplay(Duck self);

#endif /* _DUCK_H_ */
```

DuckInternal.h - Class representation (private)

```
/*-----------------------------------------------------------------------------
 *
 * COPYRIGHT (C) 2012 Arrive Technologies Inc.
 *
 * The information contained herein is confidential property of Arrive Tecnologies.
 * The use, copying, transfer or disclosure of such information
 * is prohibited except by express written agreement with Arrive Technologies.
 *
 * Module      : OOP
 *
 * File        : DuckInternal.h
 *
 * Created Date: Aug 16, 2012
 *
 * Author      : namnn
 *
 * Description : Duck class representation
 *
 * Notes       :
 *----------------------------------------------------------------------------*/

#ifndef _DUCKINTERNAL_H_
#define _DUCKINTERNAL_H_

/*--------------------------- Includes ----------------------------------------*/
#include "ObjectInternal.h"
#include "Duck.h"

/*--------------------------- Define ------------------------------------------*/

/*--------------------------- Macros ------------------------------------------*/
```

```
/*--------------------------- Typedefs --------------------------------------*/
typedef struct tDuckMethods
    {
    void (*Swim)(Duck self);
    void (*Display)(Duck self);
    void (*NameSet)(Duck self, char *name, int len);
    char *(*NameGet)(Duck self, int *len);
    }tDuckMethods;

typedef struct tDuck
    {
    tObject super;
    const tDuckMethods *methods;

    /* Private */
    char *name;
    }tDuck;

/*--------------------------- Forward declarations ---------------------------*/

/*--------------------------- Entries ----------------------------------------*/
/* Constructor */
Duck DuckObjectInit(Duck self);

#endif /* _DUCKINTERNAL_H_ */
```

Duck.c - Implementation

```
/*-----------------------------------------------------------------------------
 *
 * COPYRIGHT (C) 2012 Arrive Technologies Inc.
 *
 * The information contained herein is confidential property of The Arrive Technologies
 * The use, copying, transfer or disclosure of such information is prohibited
 * except by express written agreement with Arrive Technologies.
 *
 * Module      : OOP
 *
 * File        : Duck.c
 *
 * Created Date: Aug 16, 2012
 *
 * Author      : namnn
 *
 * Description : Duck generic implementation
 *
 * Notes       :
 *----------------------------------------------------------------------------*/

/*--------------------------- Include files ----------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "DuckInternal.h"

/*--------------------------- Define -----------------------------------------*/
#define cDuckNameMaxLength 64

/*--------------------------- Macros -----------------------------------------*/

/*--------------------------- Global variables -------------------------------*/

/*--------------------------- Local variables --------------------------------*/
static char m_methodsInit = 0;
static tDuckMethods m_methods;

/* Override */
static tObjectMethods m_ObjectOverride;
```

```
/* Save super implementation */
static const tObjectMethods *m_ObjectMethods = NULL;

/*--------------------------- Forward declarations ---------------------------*/

/*--------------------------- Impl ---------------------------------*/
static void Swim(Duck self)
    {
    printf("(Duck.Swim): Swim\n");
    }

static void Display(Duck self)
    {
    printf("(Duck.Display): My name is: %s\n", mMethodsGet(self)->NameGet(self, NULL));
    }

static char *NameBuffer(Duck self, int *bufferLength)
    {
    char *nameBuffer;

    /* Allocated, just return */
    if (self->name)
        {
        if (bufferLength)
            *bufferLength = cDuckNameMaxLength;
        return self->name;
        }

    /* Allocate memory to hold name */
    nameBuffer = malloc(cDuckNameMaxLength);
    if (nameBuffer == NULL)
        {
        ObjectDelete((Object)self);
        return NULL;
        }
    memset(nameBuffer, 0, cDuckNameMaxLength);

    self->name = nameBuffer;
    if (bufferLength)
        *bufferLength = cDuckNameMaxLength;

    return nameBuffer;
    }

static void NameSet(Duck self, char *name, int len)
    {
    int bufferLength;
    char *nameBuffer = NameBuffer(self, &bufferLength);

    if (len > bufferLength)
        len = bufferLength;

    strncpy(nameBuffer, name, len);
    }

static char *NameGet(Duck self, int *len)
    {
    char *nameBuffer = NameBuffer(self, NULL);
    if (len)
        *len = strlen(nameBuffer);

    return nameBuffer;
    }

static void Delete(Object self)
    {
    /* Delete local resource */
    printf("(Duck.Delete): Delete name\n");
    free(((tDuck *)self)->name);
    /* Fully delete itself */
    m_ObjectMethods->Delete(self);
    }

static void MethodsInit(Duck self)
    {
    if (!m_methodsInit)
        {
        memset(&m_methods, 0, sizeof(m_methods));

        mMethodOverride(m_methods, Display);
        mMethodOverride(m_methods, Swim);
        mMethodOverride(m_methods, NameSet);
        mMethodOverride(m_methods, NameGet);
        }

    mMethodsSet(self, &m_methods);
    }

static void OverrideObject(Duck self)
    {
    Object object = (Object)self;

    if (!m_methodsInit)
        {
        m_ObjectMethods = mMethodsGet(object);
        memcpy(&m_ObjectOverride, m_ObjectMethods, sizeof(m_ObjectOverride));

        mMethodOverride(m_ObjectOverride, Delete);
        }

    mMethodsSet(object, &m_ObjectOverride);
    }

static void Override(Duck self)
    {
    OverrideObject(self);
    }

static int ObjectSize()
    {
    return sizeof(tDuck);
    }

Duck DuckObjectInit(Duck self)
    {
    memset(self, 0, ObjectSize());

    /* Call super class initialize */
    if (ObjectInit((Object)self) == NULL)
        return NULL;

    /* Setup class */
    Override(self);
    MethodsInit(self);
    m_methodsInit = 1;

    return self;
    }

void DuckNameSet(Duck self, char *name, int len)
    {
    if (self)
        mMethodsGet(self)->NameSet(self, name, len);
    }

char *DuckNameGet(Duck self, int *len)
    {
    if (self)
        return mMethodsGet(self)->NameGet(self, len);
    return NULL;
    }

void DuckSwim(Duck self)
    {
    if (self)
        mMethodsGet(self)->Swim(self);
    }

void DuckDisplay(Duck self)
    {
    if (self)
        mMethodsGet(self)->Display(self);
    }
```

MallardDuck - A duck that can fly and swim in different way

MallardDuck.h - Publish header file

```
/*-----------------------------------------------------------------------------
 *
 * COPYRIGHT (C) 2012 Arrive Technologies Inc.
 *
 * The information contained herein is confidential property of Arrive Tecnologies.
 * The use, copying, transfer or disclosure of such information
 * is prohibited except by express written agreement with Arrive Technologies.
 *
 * Module      : OOP
 *
 * File        : MallardDuck.h
 *
 * Created Date: Aug 16, 2012
 *
 * Author      : namnn
 *
 * Description : Mallard Duck concrete class
 *
 * Notes       :
 *----------------------------------------------------------------------------*/

#ifndef _MALLARDDUCK_H_
#define _MALLARDDUCK_H_

/*--------------------------- Includes ---------------------------------------*/
#include "Duck.h" /* Super class */

/*--------------------------- Define -----------------------------------------*/

/*--------------------------- Macros -----------------------------------------*/

/*--------------------------- Typedefs ---------------------------------------*/
typedef struct tMallardDuck * MallardDuck;

/*--------------------------- Forward declarations ---------------------------*/

/*--------------------------- Entries ----------------------------------------*/
MallardDuck MallardDuckNew();
void MallardDuckFly(MallardDuck self);

#endif /* _MALLARDDUCK_H_ */
```

MallardDuckInternal.h - Class representation (private)

```
/*-----------------------------------------------------------------------------
 *
 * COPYRIGHT (C) 2012 Arrive Technologies Inc.
 *
 * The information contained herein is confidential property of Arrive Tecnologies.
 * The use, copying, transfer or disclosure of such information
 * is prohibited except by express written agreement with Arrive Technologies.
 *
 * Module      : OOP
 *
 * File        : MallardDuckInternal.h
 *
 * Created Date: Aug 16, 2012
 *
 * Author      : namnn
 *
 * Description : Mallard Duck class represenation
 *
 * Notes       :
 *----------------------------------------------------------------------------*/

#ifndef _MALLARDDUCKINTERNAL_H_
#define _MALLARDDUCKINTERNAL_H_

/*--------------------------- Includes ---------------------------------------*/
#include "MallardDuck.h"
#include "DuckInternal.h"

/*--------------------------- Define -----------------------------------------*/

/*--------------------------- Macros -----------------------------------------*/

/*--------------------------- Typedefs ---------------------------------------*/
typedef struct tMallardDuckMethods
    {
    void (*Fly)(MallardDuck self);
    }tMallardDuckMethods;

/* MallardDuck representation */
typedef struct tMallardDuck
    {
    tDuck super;
    const tMallardDuckMethods *methods;

    /* Private data */
    }tMallardDuck;

/*--------------------------- Forward declarations ---------------------------*/

/*--------------------------- Entries ----------------------------------------*/
MallardDuck MallardDuckObjectInit(MallardDuck self);

#endif /* _MALLARDDUCKINTERNAL_H_ */
```

MallardDuck.c - Implementation

```
/*-----------------------------------------------------------------------------
 *
 * COPYRIGHT (C) 2012 Arrive Technologies Inc.
 *
 * The information contained herein is confidential property of The Arrive Technologies
 * The use, copying, transfer or disclosure of such information is prohibited
 * except by express written agreement with Arrive Technologies.
 *
 * Module      : OOP
 *
 * File        : MallardDuck.c
 *
 * Created Date: Aug 16, 2012
 *
 * Author      : namnn
 *
 * Description : Mallard Duck implementation
 *
 * Notes       :
 *--------------------------------------------------------------------------*/

/*--------------------------- Include files ---------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "MallardDuckInternal.h"

/*--------------------------- Define ----------------------------------------*/

/*--------------------------- Macros ----------------------------------------*/

/*--------------------------- Global variables ------------------------------*/

/*--------------------------- Local variables -------------------------------*/
static char m_methodsInit = 0;
static tMallardDuckMethods m_methods;

/* Override */
static tDuckMethods m_DuckOverride;

/*--------------------------- Forward declarations --------------------------*/

/*--------------------------- Implementation --------------------------------*/
static void Swim(Duck self)
    {
    printf("(MallardDuck.Swim): Swim in different way\n");
    }

static void Fly(MallardDuck self)
    {
    printf("(MallardDuck.Fly): Fly\n");
    }

static void MethodsInit(MallardDuck self)
    {
    if (!m_methodsInit)
        {
        memset(&m_methods, 0, sizeof(m_methods));

        mMethodOverride(m_methods, Fly);
        }

    mMethodsSet(self, &m_methods);
    }

static void OverrideDuck(MallardDuck self)
    {
    Duck duck = (Duck)self;

    if (!m_methodsInit)
        {
        memcpy(&m_DuckOverride, mMethodsGet(duck), sizeof(m_DuckOverride));

        mMethodOverride(m_DuckOverride, Swim);
        }

    mMethodsSet(duck, &m_DuckOverride);
    }

static void Override(tMallardDuck *duck)
    {
    OverrideDuck(duck);
    }

static int ObjectSize()
    {
    return sizeof(tMallardDuck);
    }

MallardDuck MallardDuckObjectInit(MallardDuck self)
    {
    memset(self, 0, ObjectSize());

    /* Call super class initialize */
    if (DuckObjectInit((Duck)self) == NULL)
        return NULL;

    /* Setup class */
    Override(self);
    MethodsInit(self);
    m_methodsInit = 1;

    return self;
    }

MallardDuck MallardDuckNew()
    {
    MallardDuck newDuck = malloc(ObjectSize());
    return MallardDuckObjectInit(newDuck);
    }

void MallardDuckFly(MallardDuck self)
    {
    if (self)
        mMethodsGet(self)->Fly(self);
    }
```

RedheadDuck - A duck that can quack and display in different way

RedheadDuck.h - Publish header file

```
/*-----------------------------------------------------------------------------
 *
 * COPYRIGHT (C) 2012 Arrive Technologies Inc.
 *
 * The information contained herein is confidential property of Arrive Tecnologies.
 * The use, copying, transfer or disclosure of such information
 * is prohibited except by express written agreement with Arrive Technologies.
 *
 * Module      : OOP
 *
 * File        : RedheadDuck.h
 *
 * Created Date: Aug 16, 2012
 *
 * Author      : namnn
 *
 * Description : Red Head Duck class
 *
 * Notes       :
 *----------------------------------------------------------------------------*/

#ifndef _REDHEADDUCK_H_
#define _REDHEADDUCK_H_

/*--------------------------- Includes ---------------------------------------*/

/*--------------------------- Define -----------------------------------------*/

/*--------------------------- Macros -----------------------------------------*/

/*--------------------------- Typedefs ---------------------------------------*/
typedef struct tRedheadDuck * RedheadDuck;

/*--------------------------- Forward declarations ---------------------------*/
```

```
/*--------------------------- Entries ----------------------------------------*/
RedheadDuck RedheadDuckNew();
void RedheadDuckQuack(RedheadDuck self);

#endif /* _REDHEADDUCK_H_ */
```

RedheadDuckInternal.h - Class representation (private)

```
/*--------------------------------------------------------------------------------
 *
 * COPYRIGHT (C) 2012 Arrive Technologies Inc.
 *
 * The information contained herein is confidential property of Arrive Tecnologies.
 * The use, copying, transfer or disclosure of such information
 * is prohibited except by express written agreement with Arrive Technologies.
 *
 * Module      : OOP
 *
 * File        : RedheadDuckInternal.h
 *
 * Created Date: Aug 16, 2012
 *
 * Author      : namnn
 *
 * Description : Red Head Duck class representation
 *
 * Notes       :
 *------------------------------------------------------------------------------*/

#ifndef _REDHEADDUCKINTERNAL_H_
#define _REDHEADDUCKINTERNAL_H_

/*--------------------------- Includes ---------------------------------------*/
#include "DuckInternal.h"
#include "RedheadDuck.h"

/*--------------------------- Define -----------------------------------------*/

/*--------------------------- Macros -----------------------------------------*/

/*--------------------------- Typedefs ---------------------------------------*/
/* RedheadDuck */
typedef struct tRedheadDuckMethods
    {
    void (*Quack)(RedheadDuck self);
    }tRedheadDuckMethods;

/* RedheadDuckQuack representation */
typedef struct tRedheadDuck
    {
    tDuck super;
    const tRedheadDuckMethods *methods;

    /* Additional variables go here */
    }tRedheadDuck;

/*--------------------------- Forward declarations ---------------------------*/

/*--------------------------- Entries ----------------------------------------*/
RedheadDuck RedheadDuckObjectInit(RedheadDuck self);

#endif /* _REDHEADDUCKINTERNAL_H_ */
```

RedheadDuck.c - Implementation

```c
/*---------------------------------------------------------------------------
 *
 * COPYRIGHT (C) 2012 Arrive Technologies Inc.
 *
 * The information contained herein is confidential property of The Arrive Technologies
 * The use, copying, transfer or disclosure of such information is prohibited
 * except by express written agreement with Arrive Technologies.
 *
 * Module      : OOP
 *
 * File        : RedheadDuck.c
 *
 * Created Date: Aug 16, 2012
 *
 * Author      : namnn
 *
 * Description : Read Head Duck class implementation
 *
 * Notes       :
 *----------------------------------------------------------------------------*/

/*--------------------------- Include files ---------------------------------*/
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

#include "RedheadDuckInternal.h"

/*--------------------------- Define -----------------------------------------*/

/*--------------------------- Macros -----------------------------------------*/

/*--------------------------- Global variables -------------------------------*/

/*--------------------------- Local variables --------------------------------*/
static tRedheadDuckMethods m_methods;
static char m_methodsInit = 0;

/* Override */
static tObjectMethods m_ObjectOverride;
static tDuckMethods   m_DuckOverride;

/* Save super implementation */
static const tObjectMethods *m_ObjectMethods = NULL;

/*--------------------------- Forward declarations --------------------------*/

/*--------------------------- Implementation --------------------------------*/
static void Display(Duck self)
    {
    printf("(RedheadDuck.Display): My name is %s and I can quack\n", DuckNameGet(self, NULL));
    }

static void Delete(Object self)
    {
    printf("(RedheadDuck.Delete): Delete\n");
    m_ObjectMethods->Delete((Object)self);
    }

static void Quack(RedheadDuck self)
    {
    printf("(RedheadDuck.Quack): Quack\n");
    }

static void OverrideObject(RedheadDuck self)
    {
    Object object = (Object)self;

    /* Override Delete method */
    if (!m_methodsInit)
```

```
        {
        m_ObjectMethods = mMethodsGet(object);
        memcpy(&m_ObjectOverride, m_ObjectMethods, sizeof(m_ObjectOverride));

        m_ObjectOverride.Delete = Delete;
        }

    mMethodsSet(object, &m_ObjectOverride);
    }

static void OverrideDuck(RedheadDuck self)
    {
    Duck duck = (Duck)self;

    /* Override Delete method */
    if (!m_methodsInit)
        {
        memcpy(&m_DuckOverride, mMethodsGet(duck), sizeof(m_DuckOverride));

        mMethodOverride(m_DuckOverride, Display);
        }

    mMethodsSet(duck, &m_DuckOverride);
    }

static void Override(RedheadDuck self)
    {
    OverrideDuck(self);
    OverrideObject(self);
    }

static void MethodsInit(RedheadDuck self)
    {
    if (!m_methodsInit)
        {
        memset(&m_methods, 0, sizeof(m_methods));

        mMethodOverride(m_methods, Quack);
        }

    mMethodsSet(self, &m_methods);
    }

static int ObjectSize()
    {
    return sizeof(tRedheadDuck);
    }

RedheadDuck RedheadDuckObjectInit(RedheadDuck self)
    {
    memset(self, 0, ObjectSize());

    /* Call super class initialize */
    if (DuckObjectInit((Duck)self) == NULL)
        return NULL;

    /* Setup class */
    Override(self);
    MethodsInit(self);
    m_methodsInit = 1;

    return self;
    }

RedheadDuck RedheadDuckNew()
    {
    RedheadDuck newDuck = malloc(ObjectSize());
    return RedheadDuckObjectInit(newDuck);
    }

void RedheadDuckQuack(RedheadDuck self)
```

```
{
if (self)
    mMethodsGet(self)->Quack(self);
}
```