symbian Academy

Symbian OS Types and Declarations

Introduction

Symbian OS defines a set of fundamental types and class types to:

- Provide compiler independence using **`typedefs`**

  Symbian OS code is built with a number of different compilers

- Describe the main properties and behavior of common Symbian classes

  For example, whether they may be created on the heap, on the stack or on either (and how they should be cleaned up).

- Clarify the creation, use and destruction of objects of each class type

Symbian OS uses a simple naming convention which prefixes the class name with a letter (T, C, R or M).

- The only classes which don't adopt one of these prefixes are those which possess only static member functions.

symbian Academy

Symbian OS Types and Declarations

The Fundamental Symbian OS Types

- Know how the fundamental Symbian OS types relate to native built-in C++ types
- Understand that the fundamental types should always be used in preference to the native built-in C++ types (`bool, int, float, etc.`) because they are compiler-independent

The Fundamental Symbian OS Types

The `TInt`, `TUint`, `TInt64`, `TReal`, `TAny*` and `TBool` `typedef`s

- **`TIntX`** and **`TUintX`** (where X = 8, 16 and 32) are used for 8-, 16- and 32-bit **`signed`** and **`unsigned`** integers respectively

  The non-specified (X) **`TInt`** or **`TUint`** types correspond to signed and unsigned 32-bit integers, respectively

- **`TInt64`** and **`TUInt64`** are **`typedef`**'d to **`long long`** and use the available native 64-bit support

- **`TReal32`** and **`TReal64`** are used for single- and double-precision floating-point numbers, equivalent to **`float`** and **`double`** respectively

  `TReal` equates to **`TReal64`**

- **`TAny*`** is used in in place of **`void*`**, effectively replacing it with a **`typedef`**'d "pointer to anything"

  Note: it is not necessary to replace the native **`void`** type with **`TAny`**

- **`TBool`** is used for Boolean types and is equivalent to **`int`**

  Symbian OS **`typedef`**s values of **`ETrue`** (= 1) and **`EFalse`** (= 0)

symbian Academy

Symbian OS Types and Declarations

T Classes

- Know the purpose of a T class, what types of member data it may and may not own, and that it must never have a destructor
- Know what types of function a T class may have
- Understand that a T class may be created on the heap or stack
- Understand that a T class may be used as an alternative to the traditional C/C++ **`struct`**
- Know that the T prefix is also used to define an **`enum`**.

T Class Properties and Behavior

T classes behave, and may be used, like the C++ built-in types

- Thus have the same 'T' prefix as the fundamental Symbian types (`TInt` etc)
- Like built-in types, T classes do not have a destructor

  Note: C++ does require classes to have implicit destructors but the class user should not rely on it being called for clean-up

- T classes should not own pointers or handles to resources

  But can contain pointers and handles to objects and resources owned by other objects

T classes contain all their data internally

- Built-in types and objects of other T classes
- Pointers and references with a "uses-a" relationship rather than a "has-a" relationship, which would imply ownership

T Class Properties and Behavior

T classes may be created on the heap

- The pointer to the object should be pushed onto the cleanup stack prior to calling code with the potential to leave. In the event of a leave, the memory for the T class object is deallocated, but no destructor call is made

T classes are also often defined without default constructors

- Typically to allow a T class consisting only of built-in types to initialize in a similar manner as a **`struct`**

```
TMyPODClass local = {2000, 2001, 2003};
```

T Class Properties and Behavior

T classes typically have simple APIs

- Simply a C-style **`struct`** consisting only of public data

There are some T classes with more complex APIs

- Such as the lexical analysis class **`TLex`** and the descriptor base classes **`TDesC`** and **`TDes`**

The T prefix is used for enumerations too, since these are simple types

```
enum TMonth {EJanuary = 1, EFebruary = 2, ..., EDecember = 12};
```

A Note on Leaves & Exceptions

Removing the reliance on a destructor for clean up meant an object of a T class could be created on the stack and would be cleaned up correctly when the scope of that function exits, either through a normal return *or a leave*...

- Historically (pre-Symbian OS v9.2) a Symbian OS leave was a long jump that would not unwind the stack (anything created on the stack would be disregarded i.e. destructors would not be called) ...
  - ... Potentially orphaning pointers to heap objects, thus creating a memory leak - hence the requirement for not allowing T Classes to own pointers
- Symbian OS now supports standard C++ exceptions which unwind the stack and calls the destructors of all stack objects

symbian Academy

Symbian OS Types and Declarations

C Classes

- Recognize that a C class always derives from **`CBase`**
- Know the purpose of a C class, and what types of data it may own
- Understand that a C class must always be instantiated on the heap
- Know that a C class uses two-phase construction and has its member data zero-filled when it is allocated on the heap
- Understand the destruction of C classes via the virtual destructor defined in **`CBase`**

C Class Properties and Behavior

C classes contain member data that needs initialization at construction that could fail

- Such as allocation of memory or opening a resource handle

C classes are suitable for objects that need to be allocated on the heap

- To contain and/or own pointers to large objects

C class objects are frequently large objects in their own right

- i.e. unsuitable for creation on the stack

C class usually need to have a destructor defined

- They have member variables which need cleaning up in a destructor

All C classes derive from `CBase`

C classes must ultimately derive from class `CBase`

- Defined in `e32base.h`

`CBase` has private copy constructor and assignment operators

- Without these, a compiler would generate implicit versions that simply perform shallow copies of any member data
- Their private declaration thus prevents calling code from accidentally performing invalid copy operations on C classes
- A C class can still define its own a copy constructor and assignment operator

`CBase` has two key implementation characteristics

- safe construction and destruction, and zero initialization

Safe Construction and Destruction

Correct inheritance destruction order

- **`CBase`** has a virtual destructor ensuring that C++ calls the destructors of the derived class(es) in the correct order (starting from the most derived class and calling up the inheritance hierarchy)

Two-phase construction

- To allow safe initialization code at construction time (for example, memory allocation) that could otherwise potentially leave and leak memory
- Two-phase construction is a pattern characterized by making all constructors **`private`** or **`protected`**
- A static factory function is provided, usually called **`NewL()`** or **`NewLC()`**
- Any construction code which may leave is called within the factory method allowing an 'atomic' transaction roll back
- This will become clearer later in course when leaves and two-phase construction are explored in more detail

Zero Initialization

A characteristic of **CBase**, and hence its derived classes, is that it overloads **operator new** to zero-initialize an object

- This means that all member data in a **CBase**-derived object will be zero-filled when it is first created, so this does not need to be done explicitly in the constructor.

Zero-initialization only occurs when an object is first allocated on the heap...not when it is deleted

Zero initialization will not occur for stack objects, which is why all C classes must be created on the heap.

symbian Academy

Symbian OS Types and Declarations

R Classes

- Know the purpose of an R class, to own a resource
- Understand that an R class can be instantiated on the heap or the stack
- Understand the separate construction and initialization of R classes
- Understand the separate cleanup and destruction of R classes, and the consequences of forgetting to call the `Close()` or `Reset()` method before destruction

R Class Properties and Behavior

The "R" which prefixes an R class indicates that it owns an external resource handle

- For example a handle to a server session

R classes are often small

- Usually containing no other member data besides a resource handle

It is rare for an R class to have a destructor

- Cleanup is usually performed in a `Close()` method

R classes may exist as:

- class members
- variables on the stack
- occasionally on the heap

R Classes & Resource Handles

The types of resource handles owned by R classes vary

- Ownership of a file server session (class **`RFs`**)
- Ownership of memory allocated on the heap (class **`RBuf`** or **`RArray`**).

R classes are typically used when implementing a client–server framework

- R classes are usually used to store client-side handles to server sessions
- More on the client-server framework idiom later in the course

The resource handle will not be initialized in its constructor:

- Initialization may fail and a constructor cannot return an error or leave
- Unlike for C classes there is no equivalent ***`RBase`*** class
- a typical R class will just have a simple constructor which sets the resource handle to zero

Open and Closing a Resource

To initialize the R-class object the class typically has a function such as **`Open()`**, **`Connect()`** or **`Initialize()`**

- Called after construction to set up the associated resource and store its handle as a member variable of the R-class object.

An R class also has a corresponding **`Close()`** or **`Reset()`** cleanup method

- Used to release the resource and reset the resource handle
  - Although in theory the this function can be named anything, by convention it is almost always called **`Close()`**

Note a common mistake when using R classes is to forget to call **`Close()`**

- This can lead to serious memory leaks
- Do not assume that there is a destructor which cleans up the owned resource
- Something to look for when performing a code review.

symbian Academy

Symbian OS Types and Declarations

M Classes

- Know the purpose of an M class, to define an interface
- Understand the use of M classes for multiple inheritance, and the order in which to derive an implementation class from C and M classes
- Know that an M class should never contain member data and does not have constructors
- Know what types of function an M class may include and the circumstances where it is appropriate to define their implementation
- Understand that an M class cannot be instantiated

M Class Properties and Behavior

The "M" prefix stands for "mixin"

- A term originating from an early object-oriented programming system
- M Classes are usually referred to as mixins or interfaces

An M class is an abstract interface class

- Declares pure virtual functions

A concrete class deriving from a mixin typically inherits from **`CBase`**

- And from one or more mixins and implements the interface functions

The correct class-derivation order is always to put the **`CBase`** class first:

```
class CCat : public CBase, public MDomesticAnimal{...};
```

And not

```
class CCat : public MDomesticAnimal, public CBase{...};
```

M Class Properties and Behavior

M classes are often used to define callback interfaces or observer classes

- The calling object only requires the interface and does not need to know about the implementation

The use of multiple interface inheritance is the only form of multiple inheritance encouraged on Symbian OS.

- Other forms of multiple inheritance can introduce significant levels of complexity

An M class must have no member data

An M class is never instantiated and has no member data

- So there is no need for an M class to have a constructor

M Classes & Destructors

Should an M class have a destructor?

- A destructor places a restriction on how the mixin is inherited, forcing it to be implemented only by a CBase-derived class (as against T or R since these don't have destructors)
- Having a destructor also means that delete must be called to clean up, which in turn demands that the object cannot reside on the stack
- Thus a destructor forces deriving classes to always be heap-based C classes

It may be preferable to provide a pure virtual `Release()` method

- The owner can just say "I'm done" - it is up to the implementing code to decide the cleanup method
- This is a more flexible interface because the implementing class can be stack- or heap-based, and can perform reference counting, special cleanup or other tasks

symbian Academy

Symbian OS Types and Declarations

Static Classes

- Know that static classes do not have a prefix letter
- Understand that static classes cannot be instantiated because they contain only static functions

Classes containing only static methods

Symbian OS static classes are effectively used as namespaces

- Symbian OS pre-dates standard C++ namespaces

Symbian OS static classes take no prefix letter

- They provide utility code through a set of static member functions
- For example **`User`**, **`Math`** and **`Mem`**

The classes themselves cannot be instantiated

- The functions are invoked using the scope-resolution operator

```
User::After(1000); // Suspends the current thread for 1000 microseconds

Mem::FillZ(&targetData, 12); // Zero-fills 12-byte block starting
                             // from &targetData
```

A static class is sometimes implemented to act as a factory class

symbian Academy

Symbian OS Types and Declarations

Factors to Consider when Creating a Symbian OS Class

- Know the important factors to consider when creating a new class, and how this determines the choice of Symbian OS class type

Considerations when creating a new class

Classes that do not contain any member data

- An interface class for inheritance only
- Usually defined as an M class
- Sometimes a C class which has pure virtual or virtual functions only

Classes containing only static functions (or factory classes)

- Will be assigned no prefix at all i.e. a static class

If the member data has no destructor and needs no special cleanup

- That is, contains only native types, T classes, or pointers and references to objects owned elsewhere
- It will typically be defined as a T class

Considerations when creating a new class

If the size of data the class contains will be large (over 512 bytes)

- Stack space is limited, thus avoid large stack-based (T class) objects
- It will typically be a C class i.e. created on the heap
- A T class can be defined but it should be made clear that it should always be used on the heap

If the class owns data that needs to be cleaned up

- Such as heap-based buffers, C-class objects or R-class objects
- It will be a C class i.e. destructor

symbian Academy

Symbian OS Types and Declarations

Why Is the Symbian OS Naming Convention Important?

- Understand that the use of a class prefix makes it clear to anyone wishing to use a class how it should be instantiated, used and destroyed safely.
- Recognize that the naming convention forces a class designer to think about the factors described (below) and, having decided on the fundamental behavior, can concentrate on the role of the class, knowing that leave-safe construction, destruction and ownership are already handled.

Why is the Symbian OS Naming Convention Important?

The class types simplify the design process

- Providing a consistence implementation pattern
- The required behavior of a class can be matched to the definitions of the basic Symbian OS types
- Leaving the purpose of the class design (a little less) hindered by support implementation issues

The class types simplify usage

- Following the naming convention a user of an unfamiliar class can be confident in how to:
  - Instantiate an object
  - Use it
  - Destroy it in a leave-safe way

symbian Academy

Symbian OS Types and Declarations

- ✓ The Fundamental Symbian OS Types
- ✓ T Classes
- ✓ C Classes
- ✓ R Classes
- ✓ M Classes
- ✓ Static Classes
- ✓ Factors to Consider when Creating a Symbian OS Class
- ✓ Why is the Symbian OS Naming Convention Important?