Capítulo 5

C++ Programación orientada a objeto

Como ya se ha comentado en el capítulo anterior C++ es uno de los lenguajes de desarrollo en el que se pueden desarrollar aplicaciones Symbian. C++ es un lenguaje que proviene de C pero con la gran diferencia de que es orientado a objetos. C++ es más rápido en la compilación que Java y consigue un mejor aprovechamiento de los recursos y capacidades del sistema operativo Symbian. Con C++ se encuentran accesibles muchas de las características de los smartphones, disponiendo de librerías para manejar cada uno de estas.

A pesar de estas ventajas de C++ frente a Java, se tiene el handicap de ser un lenguaje mucho más complejo en la realización de nuevas tareas.

Aunque se ha estado generalizado al describir C++ como el lenguaje utilizado en la implementación de aplicaciones en Symbian, no es exactamente el lenguaje usado, ya que lo que se usa es una versión modificada. La principal razón por la que se usa una modificación de C++ es porque C++ estándar no contiene librerías de interfaz de usuario y no dispone de mecanismos de gestión de hilos.

A continuación se describen algunas de las característica principales de C++ [10] y más tarde se comentan algunas otras características específicas de Symbian C++.

5.1 Conceptos generales de la programación orientada a objetos

- Clase: es una plantilla que define la estructura de un conjunto de objetos, que al ser creados se llamarán las instancias de la clase. Esta estructura está compuesta por la definición de los atributos y la implementación de las operaciones (métodos).

- Objeto: es la implementación de una instancia de clase, es decir, una ocurrencia de esta, que tiene los atributos definidos por la clase, y sobre la que se puede ejecutar las operaciones definidas en ella.

- Identidad: característica de cada objeto que los diferencia de los demás, incluyendo de aquellos que pudieran pertenecer a la misma clase y tener los mismos valores en sus atributos.

- Herencia: es la propiedad que tienen las clases para heredar propiedades y métodos de otras clases.

5.1.1 El concepto de clase

Los objetos en C++ son abstraídos mediante una clase. Según el paradigma de la programación orientada a objetos, un objeto consta de:

- Métodos o funciones.
- Atributos o variables miembro.

5.1.2 Constructores

Son unos métodos especiales que se ejecutan automáticamente al crear un objeto de la clase. En su declaración no se especifica el tipo de dato que devuelven, y poseen el mismo nombre de la clase a la que pertenecen. Al igual que otros métodos, puede haber varios constructores sobrecargados, aunque no pueden existir constructores virtuales.

5.1.3 Destructores

Son funciones especiales llamadas automáticamente en la ejecución del programa, y que por tanto no deben ser llamadas explícitamente por el programador. Su cometido es liberar los recursos computacionales que el objeto de la clase haya adquirido en tiempo de ejecución al expirar este.

Los destructores son invocados automáticamente al alcanzar el flujo del programa el fin del ámbito en el que está declarado el objeto.

5.1.4 Funciones Miembro

Son aquellas declaradas en el ámbito de la clase. Son similares a las funciones habituales, excepto en que el compilador cambia el nombre de la función añadiendo un identificador de la clase en la que está declarada. Además las funciones miembro reciben un parámetro adicional, el puntero this, que hace referencia al objeto que ejecuta la función.

5.1.5 Clases abstractas

Una clase abstracta está diseñada como clase padre de la cual se derivan clases hijas. Se usan para representar entidades o métodos que después se implementarán en las clases derivadas. Aunque la clase abstracta en sí no tiene ninguna implementación, sólo representa los métodos que se deben implementar.

En C++ los métodos de las clases abstractas se definen como funciones virtuales puras.

5.1.7 Herencia

Existen varios tipos de herencias entre clases:

Herencia simple

La herencia es un mecanismo creado para facilitar, y mejorar el diseño de clases de un programa. Con ella se pueden crear nuevas clases a partir de clases ya hechas, siempre y cuando tengan un tipo de relación especial.

En la herencia, las clases derivadas heredan los datos y las funciones miembro de las clases base, pudiendo las clases derivadas redefinir estos comportamientos (polimorfismo) y añadir comportamientos nuevos propios de las clases derivadas.

Las declaraciones realizadas en la definición de una clase se agrupan en tres apartados diferentes que se inician con las siguientes palabras clave:

- public: las declaraciones realizadas en este grupo serán accesibles por otros programas o módulos.

- private: en este grupo se incluyen todos aquellos miembros que no se ponen a disposición de terceros.

- protected: sólo serán accesibles por los objetos derivados del inicial.

Existen tres tipos de herencia, que se diferencian en el modo de manejar la visibilidad de los componentes de la clase resultante:

- Herencia pública: con este tipo de herencia se respetan las visibilidades originales de la clase base en la clase heredada.

- Herencia privada: todo componente de la clase base será privado en la clase derivada.

- Herencia protegida: los componentes públicos y protegidos en la clase base serán protegidos en la clase derivada y los privados seguirán siendo privados.

Herencia múltiple

Es el mecanismo que permite al programador hacer clases derivadas a partir de no sólo una clase base, sino de varias.

5.2 Extensiones en Symbian C++

Symbian Os crea un conjunto de directrices, que sirven de guía pero que no producen errores de compilación al no seguirlos, aunque si podrán producirse errores de ejecución o de ineficiencia de las aplicaciones [11].

- Convenciones de nomenclatura: tienen significado especial en relación con la funcionalidad de clases, objetos, o la limpieza en la escritura. Están diseñados para evitar errores de programación y es recomendable ajustarse a ellos.

- Manejo de cadenas y descriptores.

- Reglas para la herencia: en Symbian se tiene sus propias reglas específicas para la herencia de clases. Como por ejemplo, clases derivadas obligatoriamente de otras o restricciones en la herencia múltiple.

- Soporte de limpieza: cuando se dan lugar abandonos que pueden producir fugas de memoria o de recursos, Symbian proporciona un mecanismo de limpieza de la pila.

- Constructores en dos fases: la primera fase se corresponde con el constructor de C++ estándar.

- Modelo cliente-Servidor: la arquitectura de Symbian se basa en un micronúcleo que realiza las mínimas tareas, siendo un el sistema operativo con servidores el que se encarga de ofrecer los servicios.

- El planificador activo y los objetos activos.

5.3 Objetos activos en Symbian

En Symbian se tienen una arquitectura cliente-servidor con dos tipos de métodos fundamentales: síncronos y asíncronos.

Cuando se llama a un método síncrono el cliente del hilo tiene que esperar, siendo bloqueado, a que el servidor del hilo le notifique que la solicitud emitida ha sido completada. En muchos casos, el cliente necesita actualizar datos de la interfaz de usuario o procesar eventos de entrada. Estos no podrán ser tratados si

el cliente se encuentra en espera de solicitudes durante algún tiempo, por lo que el usuario de la aplicación se verá afectado por este retraso.

Si se usan en el programa métodos asíncronos, el cliente podrá actualizar la interfaz de usuario y responder a los eventos de entrada sin necesidad de esperar a que la petición emitida se haya completado. Cuando se ha terminado de procesar la petición se notifica al cliente que procesará los resultados como con cualquier otro evento. En este caso la experiencia del usuario no se ve afectada puesto que son respondidas sus actuaciones en el sistema.

El uso de métodos síncronos es más sencillo que de métodos asíncronos, ya que se deben recordar cada método asíncrono llamado y procesar los resultados al finalizar, mientras que cuando finaliza un método síncrono los resultados se procesan automáticamente. Para facilitar el uso de los métodos asíncronos se crean en Symbian los objetos activos.

La mayoría de las aplicaciones Symbian se desarrollan en un único hilo y los objetos activos se utilizan para el tratamiento de multitareas. El gestor del hilo tiene un solo planificador activo que es responsable del orden en el que se tratan los eventos.

Sin embargo, se puede tener uno o más objetos activos que son responsables de emitir peticiones y manejar eventos, cuando se presenta la solicitud. El planificador tiene un papel fundamental en la gestión de objetos activos, ya que es responsable de las llamadas a las funciones RunL () de los objetos activos.

Los objetos activos utilizan un proveedor de servicios de clase asíncrona para prestar el servicio. El objeto activo trata de esconder el servicio asíncrono, ofrecen solicitudes y cancelan las funciones para el proveedor de servicios.

Todos los objetos activos se derivan de la clase CActive. Esto proporciona objetos activos derivados de:

- iStatus, un objeto de tipo TRequestStatus, que se pasa a la función solicitada del servicio asíncrono. Por lo tanto la función solicitada del objeto activo no incluye una variable TRequestStatus entre sus parámetros (excepto cuando el objeto activo actúa como un segundo proveedor de servicios asíncronos).

- iActive, que se utiliza para indicar que una solicitud se ha emitido. El planificador comprueba esta bandera cuando maneja la finalización de una espera de cualquier solicitud. El objeto activo la solicitud de funciones debe

establecer este indicador cuando emiten una solicitud llamando a la función SetActive ().

- La función Cancel (), que anula una solicitud pendiente y se restablece la bandera iActive. Si no hay ninguna solicitud pendiente Cancel () no hace nada, de lo contrario, se llama a DoCancel (), una función virtual pura que debe ser proporcionada por las clases derivadas y que se encarga de la cancelación específica del servicio que se está proporcionando. Cancel () espera la finalización de dicha solicitud.

- TPriority define el conjunto de prioridades que un objeto puede tener. La prioridad del objeto activo de se establece durante la construcción. Cuando el planificador de la espera finaliza, se comprueba el objeto activo con más alta prioridad. Cuando los objetos activos tienen la misma prioridad, el orden de control no está definido.

Las clases derivadas de CActive deben:

- Proporcionar su propio servicio asíncrono.

- Proporcionar una o más funciones de petición, que a su vez pasarán solicitudes a un proveedor de servicios asíncronos.

- Implementar el método DoCancel (), que pasa una solicitud de cancelación del servicio asíncrono.

- Implementar el método RunL (), que es llamado por el planificador cuando se detecta que la solicitud de un objeto activo ha finalizado.

- Proporcionar una función RunError (), que es llamada por el planificador si el método RunL () del objeto activo abandona. Una clase derivada puede utilizar la implementación por defecto.