***Framework:***

Un **framework** es un marco de trabajo que ofrece una serie de herramientas que facilitan la realización de tareas. Un **framework** tiene como objetivos tres cosas: normalizar, estandarizar y reutilizar la experiencia. Los **frameworks** no necesariamente están ligados a un lenguaje concreto.

Los elementos básicos de un Framework son:

* Frozen Spots: son aquellas partes del código provistas por el **framework**.
* Hot Spots: son las partes de código que coloca el programador e interactúan con el **framework**.
* *Inversión de Control:* El framework se encuentra andando mucho antes que el código del programador, pero el programador piensa que él llama al framework cuando es al revés. El framework controla el código del programador.

Las tres etapas principales del desarrollo del **framework** son:

* análisis del dominio: procura descubrir los requisitos del dominio y los posibles requerimientos futuros.
* diseño del **framework:** define sus abstracciones, se modelan los Frozen y Hot Spots.
* "instantiación" del **framework**.

Un **framework** se puede también clasificar según su extensibilidad:

* **Caja blanca:** se crean de nuevas clases por herencia o composición.
* **Caja negra**: se producen instancias usando código o scripts de configuración. No se requiere que el usuario sepa los detalles internos del framework

***.Net Framework:*** provee las herramientas necesarias en run-time y compile-time para construir y ejecutar aplicaciones basadas en .NET. Expone servicios de aplicaciones a través de clases de la **.NET Framework Class Library**.

Dos de los elementos más importantes en el .NET Framework son:

* **Common Language Runtime (CLR):** simplifica el desarrollo de aplicaciones, provee un entorno de ejecución robusto y seguro, soporta varios lenguajes y simplifica el despliegue y la administración. Los compiladores y herramientas exponen funcionalidades en tiempo de ejecución y permiten escribir código con el beneficio de un entorno de ejecución administrado. **CLR** administra la memoria utilizada por las aplicaciones, evitando perdidas de memoria, que podrían estar originadas por errores en el código escrito.
* **CTS (Common Type System).** Define la forma en la que los tipos deben ser declarados, utilizados y administrados por el runtime. Define las reglas que debe seguir un lenguaje, lo que asegura que distintos lenguajes puedan interactuar sin problemas. Provee una forma para que el runtime de soporte a la integración de varios lenguajes. Soporta dos categorías generales de tipos:
  + **Value types:** Directamente contienen sus datos
  + **Reference types:** Almacenan una referencia a una dirección de memoria con un valor.

***Administración de Memoria:***

El encargado de liberar memoria es el **Garbage Colector.** Libera memoria colocando como memoria disponible aquella que no tienen referencias que lo apunten.

Los objetos pertenecen a una generación. Al funcionar el recolector de basura los objetos que no son borrados sobreviven, esto causa que se eleven a la generación siguiente, esta puede oscilar entre 0 y 2

El **FINALIZE** es invocado por el **garbage colector** cuando se ejecuta, mientras que el **DISPOSE** lo invoca el programador

***Delegados:***

Los **delegados** son objetos que hacen referencia a métodos. Cuando se crea una instancia de un **delegado**, se puede asociar con cualquier método que posea una **firma** compatible y un tipo de devolución. Los delegados se utilizan para pasar métodos como argumentos a otros métodos. Los tipos de delegado están sellados, esto implica que no se pueden derivar de ellos



***Comunicación entre aplicaciones:***

Para que una comunicación se produzca se necesitan tres elementos

fundamentales:

* **Emisor:** Es quien envía los datos.
* **Canal:** medio por el cual se envían los datos, puede ser alámbrico, inalámbrico u óptico
* **Receptor:** Recibe lo enviado.

Para que el emisor y el receptor puedan transferirse datos debe existir un **protocolo** de comunicación.

**Protocolo**: conjunto de reglas que las computadoras que se comunican utilizan. En él se estandarizan los mensajes que se enviarán para poder entender lo que cada una envía. Permiten definir sintaxis, semántica y sincronización de lo transmitido.

* **TCP** (control de transmisiones). Trabaja en la capa de transporte. Se puede utilizar para crear conexiones seguras entre computadoras y enviar datos. Garantiza que los datos serán entregados y en el mismo orden. Puede distinguir distintas aplicaciones dentro de una misma máquina, a través del puerto.
* **UDP** (trabaja a nivel de transporte). Para el intercambio de datos utiliza datagramas. No realiza control de flujo, no hay confirmación de entrega o recepción.
* **IP** (trabaja a nivel de red). Los datos son enviados en bloques (paquetes). Las computadoras se identifican por una dirección IP.

Las aplicaciones informáticas pueden estar interconectadas con el objetivo de que

se transfieran información.

* **Modo batch/** por lotes: antes de realizar la transferencia de información acumula una determinada cantidad de datos, se gestiona la conexión, realiza la transferencia y finalmente se desconecta hasta la transferencia del próximo lote.
* **Modo online:** se gestiona la conexión y esta se mantiene, los datos se envían a medida que se producen.

Hoy en día la mayoría de las aplicaciones trabajan como cliente y servidor ya que son capaces de brindar ciertos servicios y datos, y en otro momento de solicitar colaboración o información remota.

***Sockets:***

Unión de una dirección IP + un puerto de la computadora.

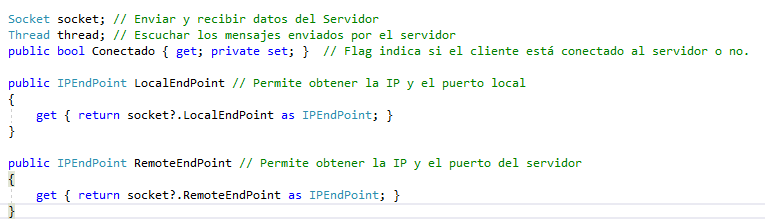
**Dirección IP**: número que identifica de manera única a una computadora. Números de 4 partes. Cada parte tiene un número que va de 0 a 255.

**Puerto**: interfaz lógica para la E/S de información. Se identifica con un número entero.

**Generación de aplicaciones:** es muy importante reconocer tres momentos muy marcados **la configuración**, **la conexión** y **el envío/recepción** de datos.

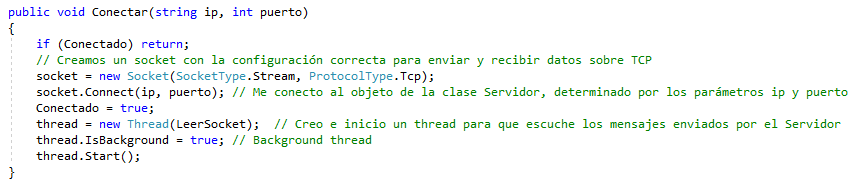
**Clase cliente**:

*Propiedades:*

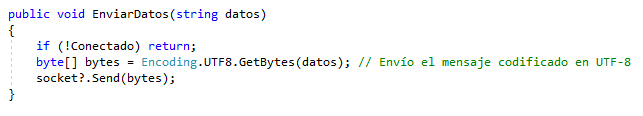


*Eventos:*



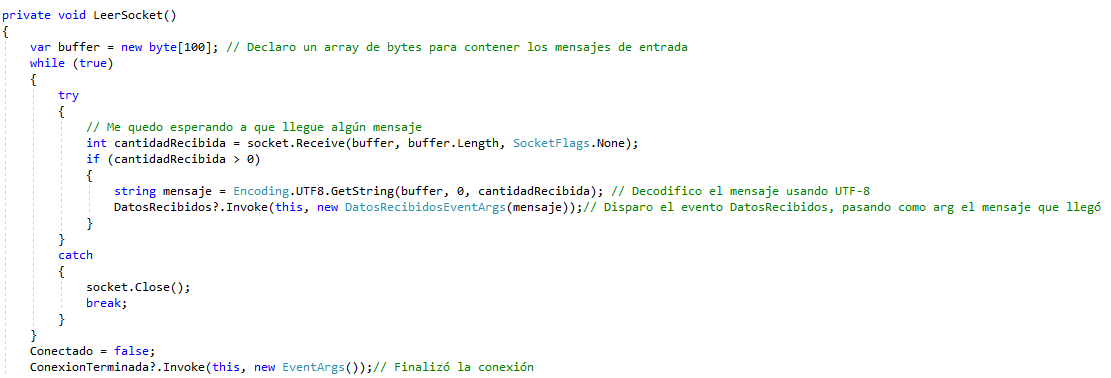
*Métodos:* Conectar**:** Envía una solicitud de conexión al servidor y, si la misma es exitosa, inicia un thread para escuchar los mensajes provenientes del mismo. 

EnviarDatos**:** Envía un mensaje al servidor



*FUNCIONES PRIVADAS:*

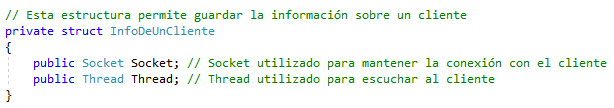
LeerSocket:Rutina que se queda esperando la recepción de datos.



**Clase servidor:**

ESTRUCTURAS: **InfoDeUnCliente** Estructura que permite guardar el socket y

el thread asociados a un cliente.

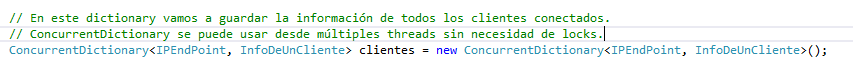


Propiedades:

**PuertoDeEscucha** Permite obtener el puerto de escucha del servidor





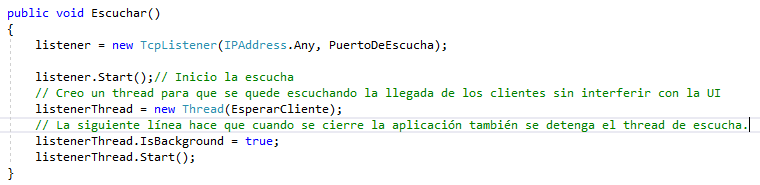


Eventos:

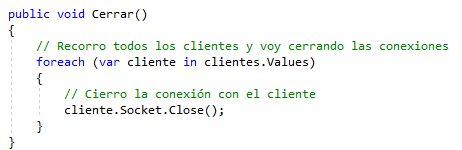


METODOS:

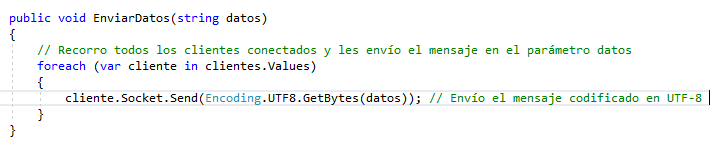
**Escuchar** Coloca al servidor en modo escucha esperando la conexión de un cliente.



**Cerrar** Cierra la conexión con todos los clientes conectados.

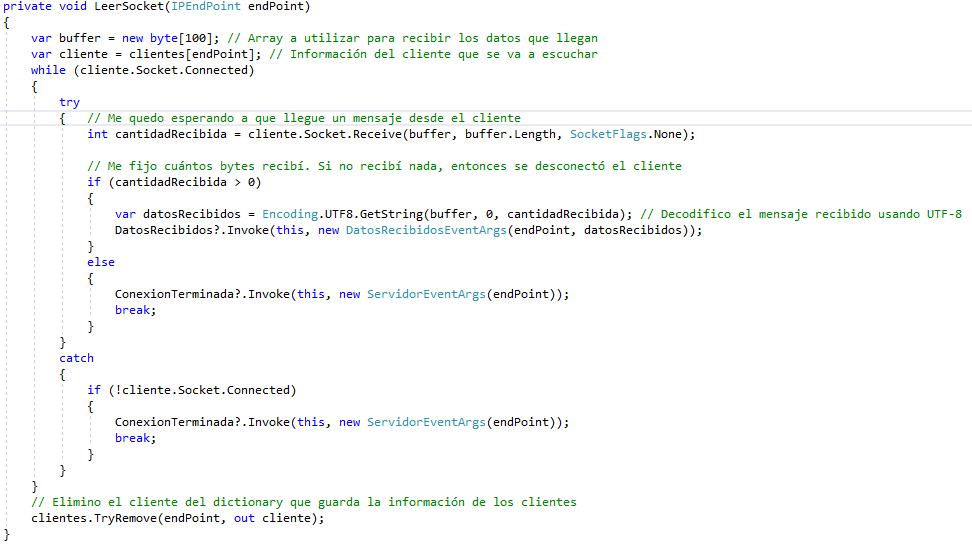


**EnviarDatos** Permite enviar datos a todos los clientes conectados.

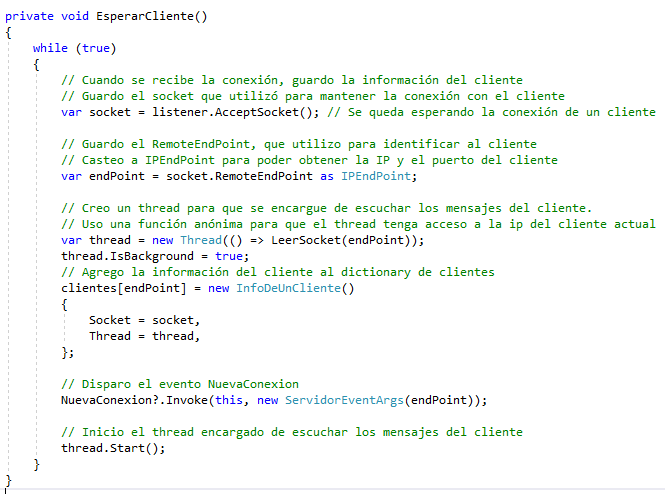


FUNCIONES PRIVADAS:

**LeerSocket** Rutina que se queda leyendo la recepción de datos por parte de un cliente.



**EsperarCliente** Rutina que se queda leyendo un puerto a la espera de que un cliente solicite conectarse



***Puertos en las computadoras***:

**PUERTO PARALELO**: Los sistemas paralelos utilizan ocho líneas de datos para transmitir un byte a la vez, y fueron utilizados para la conexión de impresoras. Podríamos adaptar este puerto para tener ocho señales y comandar otros dispositivos electrónicos.

Mandar ocho bits simultáneamente implica que se deba prestar especial atención a que deben llegar todos en un mínimo espacio de tiempo. Las distancias que se pueden cubrir con estos puertos son relativamente pequeñas. El hardware requerido está establecido por el estándar IEEE 1284. 3 tipos:

* El 1284 tipo A: es un conector hembra de 25 pines de tipo D (DB25-S).
* El 1284 tipo B que es un conector de 36 pines de tipo Centronics y encontrado en impresoras.
* El 1284 tipo C, similar al 1284 tipo B pero más pequeño, tiene mejores propiedades eléctricas y

mecánicas y es el recomendado para nuevos diseños.

**PUERTO SERIE:**  es utilizado para conectar equipamientos que esperan recibir la información bit a bit. Existen varios estándares para la comunicación serie por ejemplo RS-232, FireWire, Serial ATA y USB.

Debido a que lo hacen bit a bit los dispositivos deben saber cuando comienza y termina una trama de datos.

Existen varias formas:

* **Modo simplex:** comunicación unidireccional existiendo un emisor y un receptor.
* **Modo duplex**: ambos pueden ser emisor o receptor pero la información no puede fluir en ambos sentidos.
* **Modo fullduplex**: la información puede fluir en ambos sentidos.

***Detalles de la Guía:***

* El **sombreado** se caracteriza porque sobre escribe y oculta los otros métodos de la sobrecarga.
* **Resurrección de Objetos**: Cada vez que un objeto es eliminado del grafo, pasa a una cola de finalización en la que se le da una última oportunidad de ejecutar el código que tenga en el destructor.
* **This**: representa la instancia de la clase en la que me encuentro
* **Base**: representa la instancia de la clase padre
* Un **sistema domótico** es capaz de recoger información proveniente de unos sensores o entradas, procesarla y emitir órdenes a unos actuadores o salidas. El sistema puede acceder a redes exteriores de comunicación o información.