Proyecto final Informática I:

"Concesionario de naves espaciales de segunda mano."

Autor: Jorge Pastor García
NIA:100315924
Gr:24

Índice

Introducción

En este curso se propone realizar un programa que simule la gestión de un concesionario de naves espaciales. Para ello se requiere tener unos conceptos solidos de programación en lenguajes de alto nivel, así como conceptos bien asentados en lo que a programación orientada a objetos se refiere.

Para desarrollar este programa aplicaremos conceptos estudiados en clase aplicado al desarrollo de una aplicación, como puede ser la herencia, polimorfismo, asignación de memoria dinámica, sobrecarga de operadores… entre otros.

Adicionalmente deberemos estar familiarizados con el entorno del sistema operativo de Linux (Ubuntu), así como con la herramienta Qt Creator. Para esta familiarización, las clases prácticas nos han aportado un extra de conocimientos, así como soltura en estos entornos mencionados.

La realización de esta practica nos ha ayudado a asentar esos conocimientos aprendidos, así como a ampliarlos.

Funcionalidades requeridas

La práctica realizada tiene como finalidad la realización de un programa de gestión de venta de naves espaciales de segunda mano. Dicho proyecto tiene como objetivo implementar todas las funcionalidades pedias de manera correcta y de la forma más óptima posible.

Estas funcionalidades son entre otras:

a) La correcta gestión de propietarios:
   Se tendrá que poder dar de alta, dar de baja y modificar un usuario ya registrado en el sistema, así como poder mostrar los usuarios registrados en el sistema cuando el usuario los desee.

b) La correcta gestión de naves espaciales:
   Se podrá que poder dar de alta cuatro tipos diferentes de naves espaciales, todas con atributos comunes, y también con atributos exclusivos de cada tipo. Dar de baja una nave ya registrada, o modificar características de las naves registradas, así como mostrar las naves existentes en el sistema.

c) Permitir realizar compras:
   Se debe poder realizar la compra de naves espaciales por parte de los dos tipos de usuarios existentes (humanos y extraterrestres). Además, se tendrá que evitar la compra de estaciones espaciales por parte de extraterrestres, y de destructores por parte de humanos.

d) Gestionar un registro de vehículo vendidos:
Se deberá de poder consultar un registro de ventas de naves espaciales donde se mostrarán todas las ventas realizadas en la aplicación. Adicional mente se deberá de poder hacer la misma consulta de ventas, pero en una fecha introducida por el usuario.

e) Guardar datos:
Se deberá poder guardar los datos al salir de la aplicación, así como recuperarlos al entrar de nuevo.

Teniendo en cuenta todas las exigencias pedidas se ha diseñado para hacer el programa lo más fácil posible de usar para el usuario y blindándolo para salidas inesperadas del programa como puede ser la incorrecta utilización del programa por parte del usuario, o la introducción de un tipo de dato erróneo (por ejemplo, introducir un carácter en lugar de un entero).

Atendiendo a estas características mínimas que tiene que tener nuestro programa se ha intentado realizar su implementación de la mejor manera posible.

Diagrama de clases actualizado y descripción de clases

Diagrama de clases base:

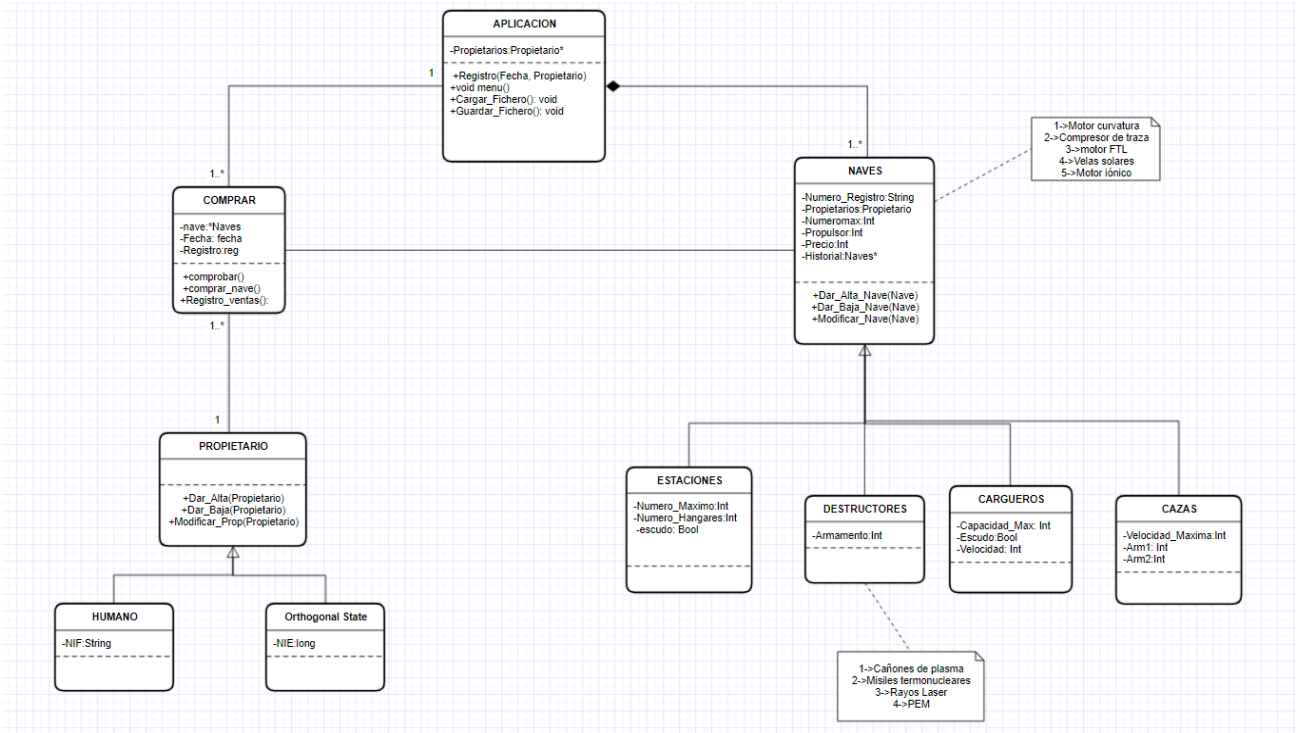
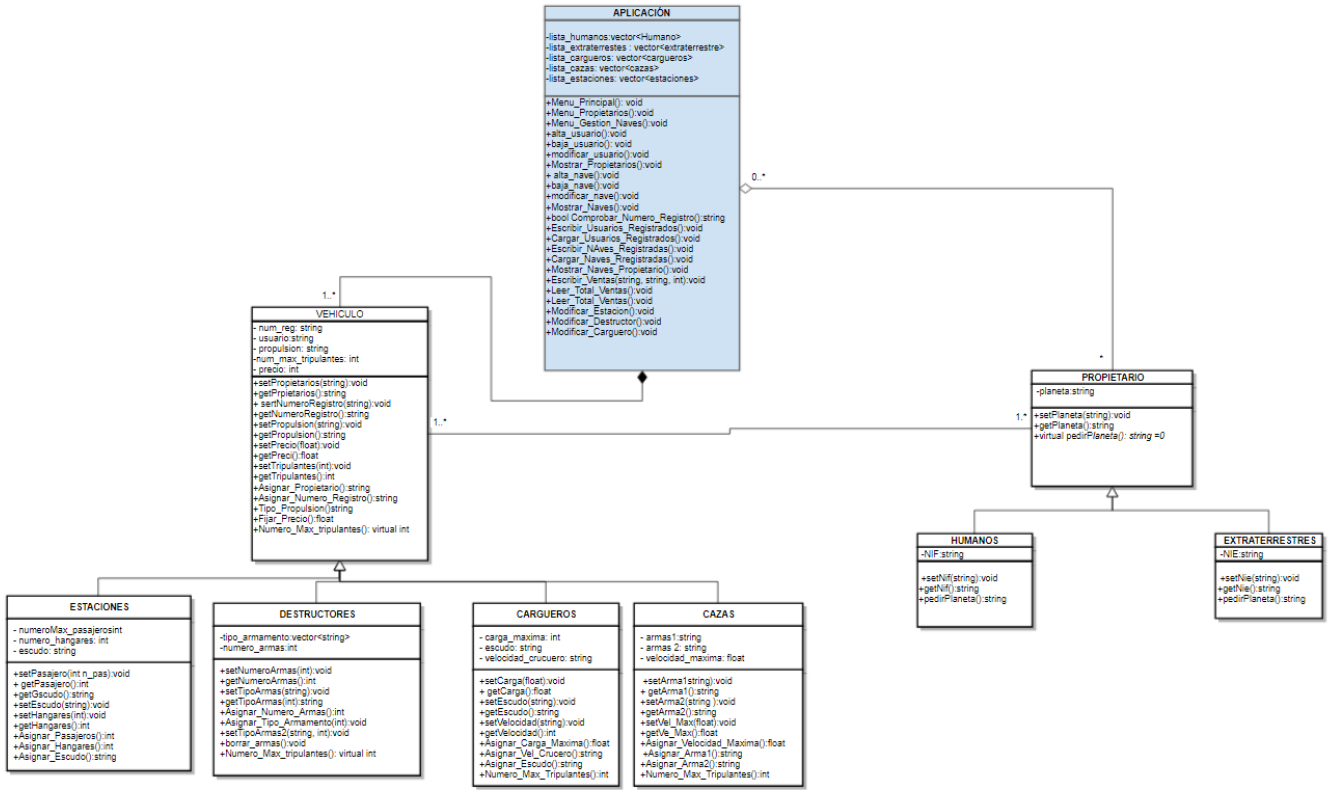


Diagrama de clases actualizado:



- **Clases principales**

1. **Aplicación:**

Atributos: Se almacenan los datos de la aplicación

```
private:

    vector <humanos> lista_humanos;
    vector <extraterrestres> lista_extraterrestres;
    vector <estaciones> lista_estaciones;
    vector <cargueros> lista_cargueros;
    vector <cazas> lista_cazas;
    vector <destructores> lista_destructores;
```

Métodos:

```
public:
    aplicacion();
    void Menu_Principal();
    void Menu_Propietarios();
    void Menu_Gestion_Naves();
    void alta_propietario();
    void baja_propietario();
    void modificar_propietario();
    void Mostrar_Propietarios();
    void alta_nave();
    void baja_nave();
    void modificar_nave();
    void Mostrar_Naves();
    void Comprar_Nave();
```

```
bool Comprobar_Numero_Registro(string);
void Escrbir_Usuarios_Registro();
void Cargar_Usuarios_Registro();
void Escribir_Naves_Registradas();
void Cargar_Naves_Registradas();
void Mostrar_Naves_Propietarios();
void Escribir_Ventas(string, string, int);
void Leer_Total_Ventas();
void Leer_Total_Ventas_Fecha();
void Modificar_Estacion();
void Modificar_Destructor();
void Modificar_Carguero();
void Modificar_Caza();
~aplicacion();
```

- Funciones Menús: Se muestran el menú principal, que llama a otros submenús como el de gestión de naves, como el de propietarios.

- Alta_Propietario (): se piden los datos de los propietarios (NIE y NIF) así como el planeta de origen. Atendiendo al formato de identificación pedido.

- Baja Propietario (): Se podrá elegir un usuario para borrar del sistema.

- Modificar_Propietario(): Recupera el usuario seleccionado, recupera los datos necesarios, lo elimina del sistema y añade el usuario en la misma posición del vector en la que estaba el propietario anterior.

- Mostrar_propietarios(): Se recorren todos los vectores en los que estén guardados los propietarios registrados, y los imprime por pantalla.

- Alta_Nave(): Se crean los objetos de las naves y se inicializan atendiendo a los requisitos pedidos (formatos de códigos de identificación, cantidad de atributos…) una vez creado el objeto e inicializados sus atributos se guardará en el vector del tipo de nave correspondiente.

- Baja_nave(): Se recorren los vectores de naves, se imprimen por pantalla, y se borra la nave seleccionada del vector correspondiente a su tipo de nave.

- Modificar_Nave(): Es un menú que dependiendo de la opción seleccionada, llamará a Modificar_Estacion/Destructor/Carguero/Caza().

- Mostrar_Naves(): Se recorren los vectores donde están guardas las diferentes naves espaciales, y se imprimen sus atributos inicializados por pantalla.

- Comprar_Naves(): Consta de dos menús, a los cuales se podrá acceder con un determinado numero de usuario, si el usuario introducido es humano, irá a su menú, de caso contrario se irá al menú extraterrestres, se imprimen las naves del tipo seleccionado para la compra, se elige una nave. Esta función llamará a otra de escritura en un fichero, la cual escribirá la nave seleccionada en un fichero .txt registrando automáticamente la fecha de compra, y el comprador de la nave. Cuando acaba la escritura de la nave en el fichero, se elimina la nave seleccionada del sistema.

- Comprobar_Numero_Registro(string): Verifica que ninguna nave tenga asignado el numero de registro que se desea introducir.

- Escribir_Usuarios_Registro(): Esta función escribe en los ficheros de texto "lista_humanos.txt" y "lista_extraterrestres.txt" los vectores de humanos y extraterrestes. El formato es el siguiente:

```
12345678A;Planeta Tierra
87654321A;Planeta Tierra
05961375M;Planeta Tierra
```

```
1234567890;Marte
0987654321;Jupiter
```

Cada atributo está separado por un punto y como para poder recuperarlos más adelante.

- Cargar_Usuarios_Registro(): Esta función lee el fichero línea a línea hasta el final del fichero, se usa la sentencia strtok() que separa las palabras entre dos puntos y comas para poder inicializarlos a continuación. Esta función es llamada desde el main para tener los datos cargados desde el principio de la ejecución.

- Escribir_Naves_Registradas(): Se recorren los vectores de naves y se escriben los atributos en sus ficheros de texto correspondientes. El formato es común para todas las naves y es el siguiente:

```
1234567890;D1234ABC;Motor iónico;234;21;Misiles termonucleares;Rayos laser;Rayos laser
05961375M;D5678ABC;Velas solares;12.56;9;PEM;Rayos laser;Misiles termonucleares
```

- Cargar_Naves_Registradas(): Lee los ficheros correspondientes línea a línea y con la sentencia strtok() se recuperan los atributos para inicializarlos posteriormente. Esta función es llamada desde el main para tener los datos cargados desde el principio de la ejecución.

- Mostrar_Naves_Propietarios(): Se pide una cadena a introducir, y se compara en un bucle con el atributo propietario de los objetos guardados en los vectores de las naves. Si estos coinciden, se imprimirán las naves asociadas a su nombre.

- Escribir_Ventas(string, string, int): A esta función se le pasarán tres parámetros, uno que es un numero del 1 al 4 que indica el tipo de nave, el comprador, y la posición en la que se encuentra en el vector. Por último, la escribe en el formato de texto en modo añadir al final del fichero. Guarda los datos en el siguiente formato:

```
4;10/12/2017;12345678A;C5678ABC;Motor FTL;12.4;1;3456;Cañones plasma;PEM
```

Donde el primer dato antes del punto y coma es el tipo de nave, esto indicará como leer esta línea.

- Leer_Total_Ventas(): Esta función lee el fichero “lista_ventas.txt” línea a línea, el formato de lectura está implementado para que dependiendo del primer numero antes del primer punto y coma que se encuentre, lea de una manera u otra. Si encuentra un 1, lee en formato estación y muestra datos, si es un 2, destructores, 3 para cargueros, y 4 para cazas.

- Leer_Total_Ventas_Fecha(): Esta función realiza los mismo que la función Leer_Total_Ventas() con la uni ca diferencia de que si el segundo campo escrito, que es la fecha, coincide con la fecha introducida por el usuario, imprime los datos de la nave vendida esa fecha.

- Modificar_Estación/Destructor/Caza_(): Estas funciones crean un nuevo objeto, el cual se inicializa con los datos de la nave seleccionada si no se desea modificar un atributo, o los inicializa con nuevos datos introducidos por el usuario. Finalmente borra la posición de la nave seleccionada, e inserta una nueva nave en la posición donde estaba la anterior nave.

2. Clase naves:

Atributos: Serán heredados por los demás tipos de naves.

```
private:
    string propietario;
    string numero_registro;
    string tipo_propulsion;
    float precio;
    int numero_tripulantes;
```

Métodos: Serán heredados por los demás tipos de naves.

```
public:
    naves();
    virtual int Numero_Max_Tripulantes()=0;
    void setPropietario(string);
    string getPropietario();
    void setNumeroRegistro(string);
    string getNumeroRegistro();
    void setPropulsion(string);
    string getPropulsion();
    void setPrecio(float);
    float getPrecio();
    float getPrecio();
    void setTripulantes(int);
    int getTripulantes();
    string Asignar_Propietario();
    string Asignar_Numero_Registro();
    string Tipo_Propulsion();
    float Fijar_Precio();
    ~naves();
};
```

- Getters y Setters: Inicializan y recuperan los atributos de la clase.

- Asignar propietario(): Esta función asigna un propietario a una nave que se quiera dar de alta o modificar, para ello comprobará que el usuario introducido esté dado de alta en el sistema.

- Asignar Numero Registro(): Se pedirá introducir un numero de registro en el formato correcto, una vez introducido, se comprobará que no exista una nave registrada en el sistema con el mismo número introducido.

- Asignar_Tipo_Propulsion(): Muestra los diferentes motores para registrar, se selecciona uno, y se le asigna a la nave.

- Fijar_Precio(): Se introduce una cadena, se comprueba que sea numérica, y lo convierte a tipo entero devolviendo ese valor.

- Virtual int Numero_Max_Tripulantes(): Se declara el método virtual a heredar.

3. Clase estaciones:

Atributos: Hereda los de naves más los suyos propios.

```
private:

    string escudo;
    int numeroMax_pasajeros;
    int numero_hangares;
```

Métodos: Hereda de naves más los suyos propios.

```
public:
    estaciones();
    void setEscudo(string);
    string getEscudo();
    void setPasajeros(int);
    int getPasajeros();
    void setHangares(int);
    int getHangares();
    int Asignar_Pasajeros();
    int Asignar_Hangares();
    string Asignar_Escudo();
    int Numero_Max_Tripulantes();
    ~estaciones();
```

- Getters y Setters: Inicializan y recuperan los atributos de la clase.

- Asignar_Pasajeros(): Se pide el número de pasajeros en forma de cadena, se comprueba que sea una cadena numérica para evitar salidas del programa al introducir un carácter alfabético se convierte a entero, y retorna el valor.

- Asignar_Hangares(): Se pide el número de pasajeros en forma de cadena, se comprueba que sea una cadena numérica para evitar salidas del programa al introducir un carácter alfabético se convierte a entero, y retorna el valor.

- Asignar_Escudo(): Se pregunta si la nave posee escudo, si se pulsa 's' devuelve "Si", si se pulsa 'n' devuelve "No".

- Numero_Maximo_Tripulantes: Se pide asignar el numero de tripulantes, se ha implementado como método virtual para usar polimorfismo. Retorna el número de tripulantes asignado.

4. Clase destructores:

Atributos: Hereda los de naves más los suyos propios.

```
private:

    int numero_armas;
    vector <string> tipo_armamento;
```

Métodos: Hereda de naves más los suyos propios.

```
public:
    destructores();
    void setNumeroArmas(int);
    int getNumeroArmas();
    void setTipoArmas(string);
    string getTipoArmas(int);
    int Asignar_Numero_Armas();
    void Asignar_Tipo_Armamento(int);
    void setTipoArmas2(string, int);
    void borrar_armas();
    int Numero_Max_Tripulantes();
    ~destructores();
```

- Getters y Setters: Inicializan y recuperan los atributos de la clase.
- Asignar_Numero_Armas(): Pide introducir el número de armas retornando su valor.
- Asignar_Tipo_Armamento(int): Recibe por parámetros el numero de armas, muestra el tipo de armamento que se puede asignar, pide asignar a cada arma un tipo introduciéndolo en el vector tipo_armamento.
- Borrar_Armas(): Esta función se utiliza cuando queremos inicializar dos objetos en una misma función, y solo creamos un objeto, sirve para eliminar el tipo de armamento y poder asignárselo de nuevo.
- Numero_Maximo_Tripulantes: Se pide asignar el número de tripulantes, se ha implementado como método virtual para usar polimorfismo. Retorna el número de tripulantes asignado.

5. Clase cargueros:

Atributos: Hereda los de naves más los suyos propios.

```
private:
    float carga_maxima;
    string velocidad_crucero;
    string escudo;
```

Métodos: Hereda de naves más los suyos propios.

```
public:
    cargueros();
    void setCarga(float);
    float getCarga();
    void setVelocidad(string);
    string getVelocidad();
    void setEscudo(string);
    string getEscudo();
    float Asignar_Carga_Maxima();
    string Asignar_Vel_Crucero();
    string Asignar_Escudo();
    int Numero_Max_Tripulantes();
    ~cargueros();
```

- Getters y Setters: Inicializan y recuperan los atributos de la clase.

- Asignar_Carga_Maxima(): Se pide introducir la carga para asignarla a la nave, se introduce una cadena, se comprueba que sea numérica y que pueda tener un punto en la cadena introducida (evita que salga del programa al introducir otros caracteres), y se convierte la cadena a float.

- Asignar_Vel_Crucero(): Se pregunta si la nave posee velocidad de crucero equipada, si se pulsa 's' devuelve "Si", si se pulsa 'n' devuelve "No".

- Asignar_Escudo(): Se pregunta si la nave posee escudo, si se pulsa 's' devuelve "Si", si se pulsa 'n' devuelve "No".

- Numero_Maximo_Tripulantes: Se pide asignar el número de tripulantes, se ha implementado como método virtual para usar polimorfismo. Retorna el número de tripulantes asignado.

6. Clase cazas:

Atributos: Hereda los de naves más los suyos propios.

```
private:

    string arma1;
    string arma2;
    float velocidad_maxima;
```

Métodos: Hereda de naves más los suyos propios.

```
public:
    cazas();
    void setArma1(string);
    string getArma1();
    void setArma2(string);
    string getArma2();
    void setVel_Max(float);
    float getVel_Max();
    float Asignar_Velocidad_Maxima();
    string Asignar_Arma1();
    string Asignar_Arma2();
    int Numero_Max_Tripulantes();
    ~cazas();
```

- Getters y Setters: Inicializan y recuperan los atributos de la clase.

- Asignar_Velocidad_Maxima(): Se pide introducir la velocidad máxima para asignarla a la nave, se introduce una cadena, se comprueba que sea numérica y que pueda tener un punto en la cadena introducida (evita que salga del programa al introducir otros caracteres), y se convierte la cadena a float.

- Asignar_Arma1()/Asignar_Arma2(): Se muestran por pantalla las armas disponibles para asignar, se selecciona el arma que se desea equipar, y la retorna.

- Numero Maximo Tripulantes: Se pide asignar el número de tripulantes, se ha implementado como método virtual para usar polimorfismo. Retorna el número de tripulantes asignado.

7. Clase Propietarios:

Atributos:

```
private:
    string planeta;
```

Métodos:

```
public:
    propietarios();
    virtual string pedirPlaneta() = 0;
    void setPlaneta(string);
    string getPlaneta();
    ~propietarios();
```

- Getters y Setters: Inicializan y recuperan los atributos de la clase.

- pedirPlaneta(): Se declara el método virtual para poder realizar polimorfismo.

8. Clase humanos y extraterrestres

Atributos: Heredan de propietarios.

```
class humanos : public propietarios {

private:
    string NIF;
```

```
class extraterrestres : public propietarios
private:

    string NIE;
```

Métodos

```
public:
    humanos();
    void setNIF(string);
    string getNIF();
    string pedirPlaneta();
    ~humanos();
```

```
public:

    extraterrestres();
    void setNIE(string);
    string getNIE();
    string pedirPlaneta();
    ~extraterrestres();
```

- Getters y Setters: Inicializan y recuperan los atributos de la clase.

- pedirPlaneta(): Como los humanos siempre pertenecerán al planeta tierra se asigna el planeta automáticamente, en el caso de los extraterrestres, se asigna el planeta por teclado.

Soluciones adoptadas desde la POO

Para la correcta realización del programa se han de realizar distintos conceptos de programación orientada a objetos para un mejor orden y eficiencia del programa, así como una mejor comprensión. El código implementado consta de distintos conceptos de POO como pueden ser herencia, polimorfismo, encapsulamiento… entre otros. A continuación, se procederá a explicar como y por que se han implementado dichos conceptos:

a) Herencia:

Se ha decidido utilizar herencia que el programa gane eficiencia, orden y evitar la repetición de implementar funciones redundantes.

El programa consta de dos clases padres, propietarios y naves, en cada una de ella se implementan los métodos comunes para las clases hijas que heredan de esta.

La clase padre naves tendrá cuatro clases hijas (estaciones, destructores, cargueros y cazas) mientras que, de la clase propietarios, heredaran dos clases, extraterrestres y humanos, cada una con sus métodos y atributos correspondientes.

```
class cazas : public naves
{

private:

    string arma1;
    string arma2;
    float velocidad_maxima;

public:
    cazas();
    void setArma1(string);
    string getArma1();
    void setArma2(string);
    string getArma2();
    void setVel_Max(float);
    float getVel_Max();
    float Asignar_Velocidad_Maxima();
    string Asignar_Arma1();
    string Asignar_Arma2();
    int Numero_Max_Tripulantes();
    ~cazas();
};
```

```
class humanos : public propietarios {

private:

    string NIF;

public:
    humanos();
    void setNIF(string);
    string getNIF();
    string pedirPlaneta();
    ~humanos();
};
```

b) Encapsulamiento:

Se ha tenido en cuenta en realizar el mejor encapsulamiento posible, haciendo todos los atributos de las clases privados, incluso los de las clases padres, evitando asi que sean de tipo protected. Por ello se ha optado por inicializar los atributos con getters y setters.

```
class naves{

private:

    string propietario;
    string numero_registro;
    string tipo_propulsion;
    float precio;
    int numero_tripulantes;
```

```
class propietarios {

private:

    string planeta;
```

c) Clases abstractas:

Tanto las clases padres naves y propietarios serán clases abstractas, ya que se declaran métodos virtuales puros para realizar polimorfismo.

d) Polimorfismo:

Algunos de los requisitos del programa conviene implementarlos usando polimorfismo, ya que algunas clases realizan la misma acción, pero de maneras diferentes, como puede ser el asignar numero de tripulantes en las naves, como asignar el planeta de origen a los propietarios.

Para asignar el número de tripulantes a las naves, a los cazas solo se les podrá asignar un tripulante, mientras que al resto de naves los que deseemos como se puede observar en la implentación.

Clase abstracta con método virtual:

```
class naves{

private:
    string propietario;
    string numero_registro;
    string tipo_propulsion;
    float precio;
    int numero_tripulantes;

public:
    naves();
    virtual int Numero_Max_Tripulantes()=0;
```

```
class propietarios {

private:
    string planeta;

public:
    virtual string pedirPlaneta() = 0;
```

Las diferentes formas de asigna tripulantes son:

Para cazas: Solo se podrá asignar un tripulante

```
int cazas::Numero_Max_Tripulantes(){

    int numero_tripulantes;
    string cadena;
    bool numero = true;

    do{
        numero=true;
        cout << "Introduzca el numero máximo de tripulantes de la nave: ";
        cin >> cadena;

        for(unsigned i=0; i<cadena.length(); i++){
            if(!isdigit(cadena.at(i))){numero=false;
            }
        }
        if(numero){numero_tripulantes = stoi(cadena);}
    }while(numero_tripulantes!=1);

    return numero_tripulantes;
}
```

Para el resto de naves: Se podrán asignar los propietarios que desee el usuario.

```
int cargueros::Numero_Max_Tripulantes(){

    int numero_tripulantes;
    string cadena;
    bool numero = true;

    do{
        numero=true;
        cout << "Introduzca el numero máximo de tripulantes de la nave: ";
        cin >> cadena;

        for(unsigned i=0; i<cadena.length(); i++){

            if(!isdigit(cadena.at(i))){numero=false;}
        }

        if(numero){numero_tripulantes = stoi(cadena);}

    }while(numero==false);

    return numero_tripulantes;
}
```

Las diferentes formas de asignar planeta de origen son:

Para humanos: Se asigna el "Planeta Tierra automáticamente".

```
string humanos::pedirPlaneta(){

    string planeta_procedencia = "Planeta Tierra";

    cout << "\nSu planeta de procedencia es 'El Planeta Tierra." << endl;

    return planeta_procedencia;
}
```

Para extraterrestres: Se tendrá que introducir el planeta por teclado.

```
//Metodo viertual para asignar el planet a extraterrestres
string extraterrestres::pedirPlaneta(){

    string planeta_procedencia;

    cout << "Introduzca su planeta de procedencia: ";
    cin >> planeta_procedencia;
    cout << "\nSu planeta de procedencia es " << planeta_procedencia << "." << endl;

    return planeta_procedencia;
}
```

e) Sobrecarga de operadores:

Se ha optado por no realizar sobrecarga de operadores ya que es más simple inicializar y recuperar los atributos por getters y setters, esto además permite hacer un mejor encapsulamiento de las clases padres haciendo los atributos privados.

Pruebas realizadas.

Las pruebas realizadas una vez concluida la practica han sido varias:

En primer lugar, se realizaron las pruebas propuestas para la defensa, probando su correcto funcionamiento y evitar posibles errores o implementaciones mal desarrolladas.

Posteriormente se comprobó que el formato de impresión fuera incorrecto y no se mostraran mensajes donde no debía y cuidar de que la impresión de los mensajes fuera cómoda de entender para el usuario.

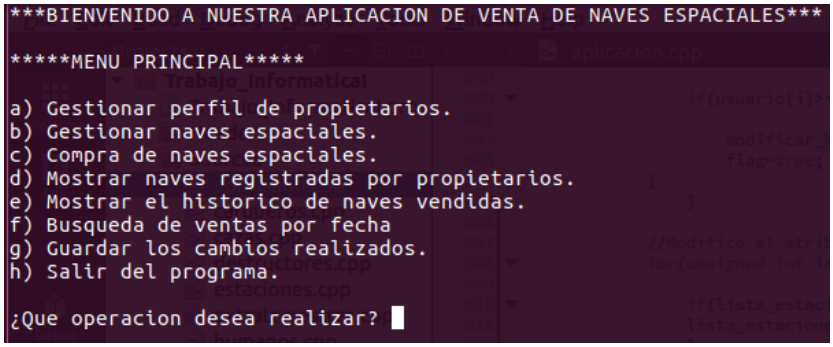
También se han buscado salidas inesperadas del programa, como puede ser la salida repentinas del programa, o la entrada en bucle por introducir un carácter donde se debía de introducir un número, intentándolo blindar de estas salidas.

Se ha probado también que un se llene el buffer de entrada al meter bastantes caracteres por teclado, limpiándolo en puntos críticos del código de este.

Finalmente se han hecho pruebas aleatorias probando el programa para encontrar posibles errores adicionales.
Creo que se han conseguido solventar todos estos problemas de manera satisfactoria.

Manual de usuario

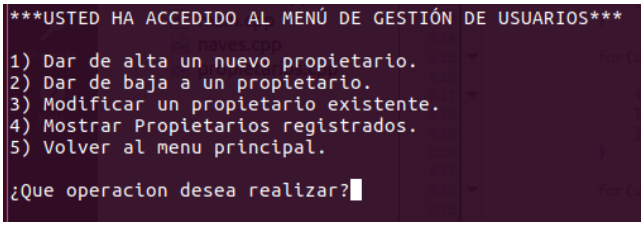
En primer lugar, el programa empieza mostrando un menú principal donde se ofrecen las distintas opciones que el usuario puede realizar:



Dentro de cada opción se mostrarán distintos submenús o funcionalidades dentro del programa que se desglosan a continuación:

a) Gestionar perfil de propietarios:

Dentro de este submenú se podrán realizar las distintas operaciones pediddas.



A cada una de las opciones del submenú solo se podrá acceder si hay usuarios registrados en el programa (salvo la opción de dar de alta un propietario), de lo contrario se muestra un mensaje informativo de que no hay usuarios registrados.

Opción 1:

Pide introducir un numero de registro en formato [NNNNNNNNNN] ó [NNNNNNNNL] hasta que sea el formato correcto y el usuario por registrar no esté registrado en el sistema. La introducción del usuario no es sensible a mayúsculas ni minúsculas. Posteriormente se pedirá introducir un planeta de origen a cada usuario. Si el usuario registrado es un humano se le asignará automáticamente el Planeta Tierra.

Opión2:

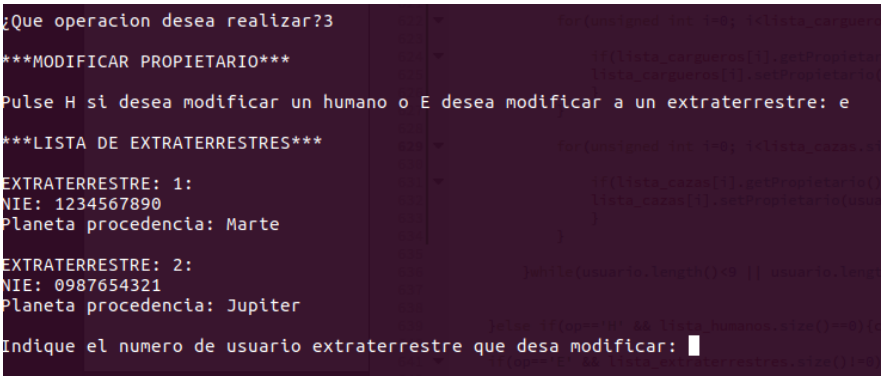
En la opción dar de baja a un propietario se pregunta a que tipo de usuario se quiere dar de baja (humanos o extraterrestres), según la elección se muestra una lista con los usuarios registrados de ese tipo de usuarios y se pide que numero de usuario se quiere dar de baja. Una vez seleccionado el usuario ya estará dado de baja del sistema.

Si se da de baja un usuario se darán de baja, junto con él, las naves espaciales asignadas a su número de registro.

Opción 3:

Para modificar un usuario, en primer lugar, el sistema pide que tipo de usuario se quiere modificar. Se muestra por consola todos los usuarios registrados de ese tipo y se selecciona el numero de usuario que desea modificar.

Se pide el nuevo numero de usuario, y se le vuelve a asignar el planeta asociado que ya tenía. Una vez modificado el usuario, se modifica el número de propietario de sus naves asociadas.



Opción 4:

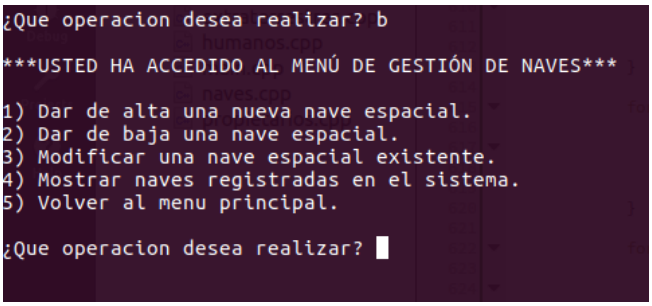
Muestra todos los usuarios de cada tipo registrados en el sistema, separándolos entre humanos y extraterrestres con su numero de registro y su planeta asociado.

Opción 5:

Vuelve al menú principal para realizar otras operaciones a gusto del usuario.

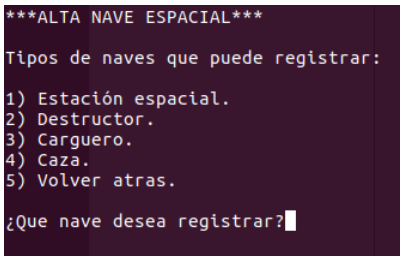
b) Gestionar naves espaciales:

Dentro de este menú se podrán realizar diferentes gestiones respecto a las diferentes naves espaciales disponibles.

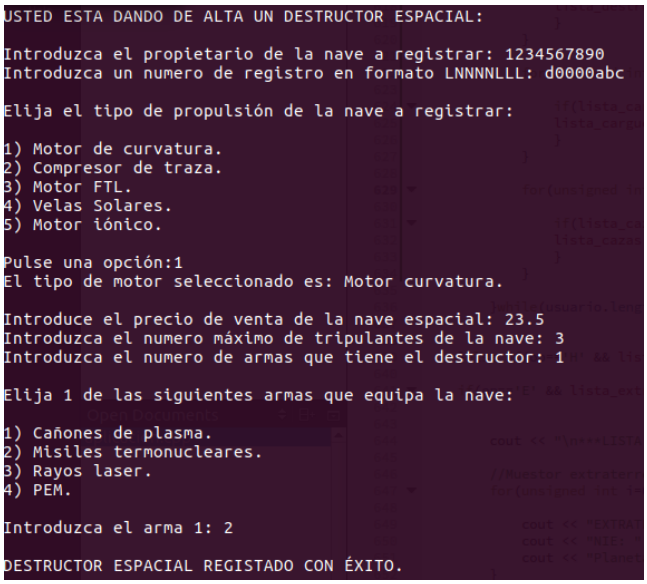


Opción 1:

En esta opción se ofrece un menú al usuario para elegir el tipo de nave espacial para dar de alta.



Una vez seleccionada una opción se procede a pedir los distintos datos a introducir de la nave espacial a registrar.



Como se puedo observar se asocian los distintos datos a la nave.

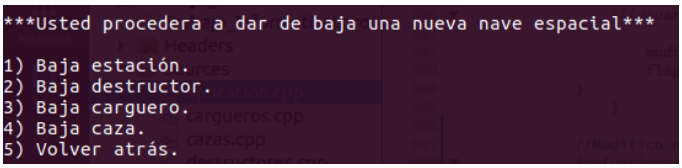
- **Como particularidades:**

Para introducir un propietario debe de estar registrado en el sistema para registrar el campo de propietario.

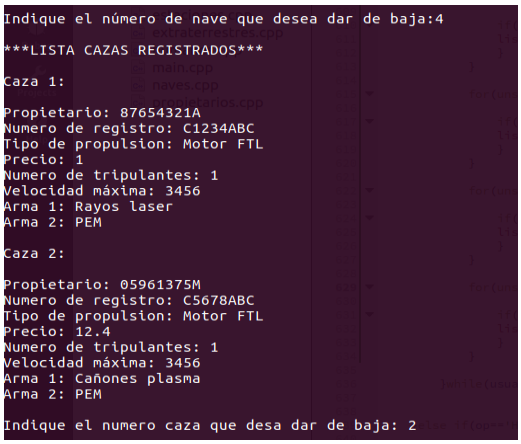
El numero de registro de la nave no puede ser uno que ya este asignado a una nave ya registrada.

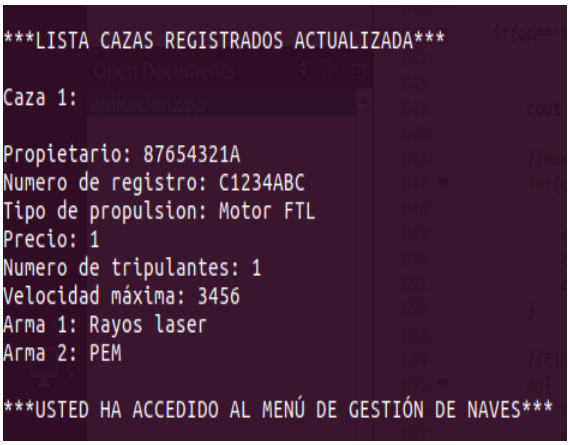
Opción 2:

Para dar de baja una nave espacial primero se pide que tipo de nave se desea dar de baja del sistema. Vez seleccionada el tipo de nave a dar de baja, se muestran todas las naves registradas de ese tipo.



Por último, se selecciona el numero de nave del listado que se quiere eliminar, y finalmente se elimina del sistema y se muestra la lista de ese tipo de naves actualizada, sin la nave espacial dada de baja del sistema y volviendo al menú de gestión de naves.



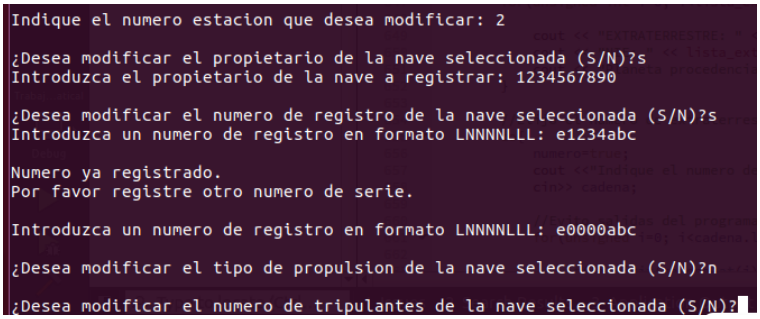


Si no hay usuarios registrados en el sistema nos se podrá acceder a esta opción ya que tampoco habría naves espaciales asociadas en el sistema.

Opción 3:

Para modificar una nave espacial primero se muestran los tipos de naves que se pueden modificar. Si se selecciona un tipo de nave que no tenga ninguna en su registro volverá a pedir otro tipo diferente.

Una vez seleccionado el tipo de nave se mostrará una lista similar que en la opción 2 (dar de baja nave espacial). Se selecciona el tipo de nave que se desea modificar, y se procede a pedir si se quieren modificar los distintos campos que la componen.



Los parámetros que se deseen modificar se vuelven a pedir y a introducir, y los parámetros que no se deseen modificar se recuperan y se guardan.

Opción 4:

Cuando se selecciona la opción de mostrar naves espaciales, se imprimen por pantalla las distintas naves espaciales registradas en el sistema ordenadas por tipos.

c) Compra de naves espaciales:

En primer lugar, se pide que se introduzca un numero de usuario. Si el usuario no está registrado se muestra un mensaje por pantalla indicado que ningún usuario corresponde a esa identificación. Si no hay ningún usuario registrado.

```
***COMPRAR NAVE ESPACIAL***

Introduce su numero de usuario: 0000000000

Es necesario estar registrado en el sistema para adquirir una nave.

Introduce su numero de usuario: 1234567890

Usted ha accedido al sistema como humano como extraterrestre.

***Usted procederá a comprar una nave espacial***


1) Destructores.
2) Cargueros.
3) Cazas.
4) Volver al menú principal.


¿Que tipo de nave desea comprar?
```

Una vez que se identifica el usuario correctamente se accederá a un menú diferente para cada tipo de usuario, los extraterrestres acceden a uno con los tipos de naves que tienen la posibilidad de comprar, y los humanos con las suyas.

Una vez en este menú, se selecciona el tipo de nave que se desea comprar, si no hay naves disponibles de ese tipo, se mostrará un mensaje diciendo que no existen naves registradas, si no, se mostrarán todas las naves disponibles para comprar.

```
¿Que tipo de nave desea comprar? 2

***LISTA CARGUEROS REGISTRADOS***

Estacion 1:

Numero de registro: R1234ABC
Tipo de propulsion: Compresor traza
Precio: 12
Numero de tripulantes: 13
Carga máxima admitida: 45
Velocidad máxima: Si
Escudo: Si

Estacion 2:

Numero de registro: R5678ABC
Tipo de propulsion: Compresor traza
Precio: 234.4
Numero de tripulantes: 27
Carga máxima admitida: 55
Velocidad máxima: Si
Escudo: Si

Seleccione el carguero espacial que desea comprar:2

El extraterrestre de NIE 1234567890 ha comprado el carguero 2.
```

Finalmente se selecciona la nave que se quiere comprar, se elimina del sistema para que no vuelva a estar disponible, y se registra la venta en un fichero de ventas, la fecha de compra se registra automáticamente en el fichero.

d) Mostrar naves registradas por propietarios:

Esta funcionalidad sirve para mostrar por pantalla las naves asociadas aun usuario registrado en el sistema. Esta función se ha implementado por si en la defensa se requiriera, realizarla con más comodidad.

Simplemente se pide un numero de usuario, si está registrado, muestra sus propiedades por pantalla, si no, muestra un mensaje por pantalla y vuelve al menú principal.

```
***BUSQUEDA DE NAVES REGISTRADAS POR USUARIOS***

¿De que usuario desea mostrar las naves registradas en el sistema?: 0000000000

El usuario 0000000000 no esta registrado en el sistema.
```

```
¿De que usuario desea mostrar las naves registradas en el sistema?: 1234567890

DESTRUCTOR 1:

Propietario: 1234567890
Numero de registro: D1234ABC
Tipo de propulsion: Motor iónico
Precio: 234
Numero de tripulantes: 21
Arma numero 1: Misiles termonucleares
Arma numero 3: Rayos laser
```

e) Mostrar histórico de naves vendidas:

Al seleccionar esta opción se pueden producir dos sucesos, que el registro de ventas esté vacío, caso en el que se muestra un mensaje por pantalla para indicar este suceso, o que no los esté, caso en el cual se mostrarán por pantalla las ventas.

```
*****USTED HA ACCEDIDO AL REGISTRO VENTAS DE LA APLICACIÓN*****

NAVE NUMERO 1: (Caza)

Fecha de compra: 8/12/2017
Comprador: 12345678A
Número de registro: C1234ABC
Tipo de motor: Motor FTL
Precio de venta: 1
Número de tripulantes: 1
Velocidad máxima: 3456
Arma 1: Rayos laser
Arma 2: PEM

NAVE NUMERO 2: (Carguero espacial)

Fecha de compra: 10/12/2017
Comprador: 1234567890
Número de registro: R5678ABC
Tipo de motor: Compresor traza
Precio de venta: 234.4
Número de tripulantes: 27
Carga máxima: 55
Velocidad crucero: Si
Escudo: Si
```

```
4;8/12/2017;12345678A;C1234ABC;Motor FTL;1;1;3456;Rayos laser;PEM
3;10/12/2017;1234567890;R5678ABC;Compresor traza;234.4;27;55;Si;Si
```

f) Búsqueda de naves por fecha:

Se pide al usuario que introduzca una día, mes y año para buscar las naves compradas en esa fecha. Si en esa fecha no hay ninguna nave registrada, se muestra un mensaje de indicando que esa fecha no se realizó ninguna venta, si no, muestra las naves vendidas durante ese periodo.

```
¿Que operacion desea realizar? f

*****USTED HA ACCEDIDO A LA BUSQUEDA DE NAVES POR FECHA DE COMPRA*****

Las ventas de que fecha desea mostrar.
Formato: d/m/yyyy:
Día: 8
Mes: 12
Año: 2017

***VENTAS REALIZADAS EN LA FECHA 8/12/2017***

NAVE NUMERO 1: (Caza)

Comprador: 12345678A
Número de registro: C1234ABC
Tipo de motor: Motor FTL
Precio de venta: 1
Número de tripulantes: 1
Velocidad máxima: 3456
Arma 1: Rayos laser
Arma 2: PEM
```

```
4;8/12/2017;12345678A;C1234ABC;Motor FTL;1;1;3456;Rayos laser;PEM
3;10/12/2017;1234567890;R5678ABC;Compresor traza;234.4;27;55;Si;Si
```

g) Guardar los cambios realizados:

Esta poción permite al usuario salir del programa haciendo permanentes las operaciones realizadas durante la ejecución del programa. Si se selecciona se actualizarán los ficheros para que en la próxima ejecución volvamos al punto anterior.

Implementación de posibles mejoras

Como aspectos a mejorar, uno de ellos puede ser el implementar menos métodos en la clase aplicación, como puede ser por ejemplo el dar de lata una nueva nave, o modificar propietarios, entre otros…

Se podrían realizar mejoras de ampliación del programa propuestos, o realizar menús de administrador o de usuario.

Como funcionalidades adicionales se ha implementado una búsqueda de propiedades de un propietario introducido previamente registrado en el sistema.