TEMA 8 → INTERFACES o ADAPTADORES

Una interface posee propiedades y/o métodos, las propiedades son CONSTANTES, y los métodos pueden ser: abstractos, por defecto o estáticos. Son siempre public en las clases implementan la interface.

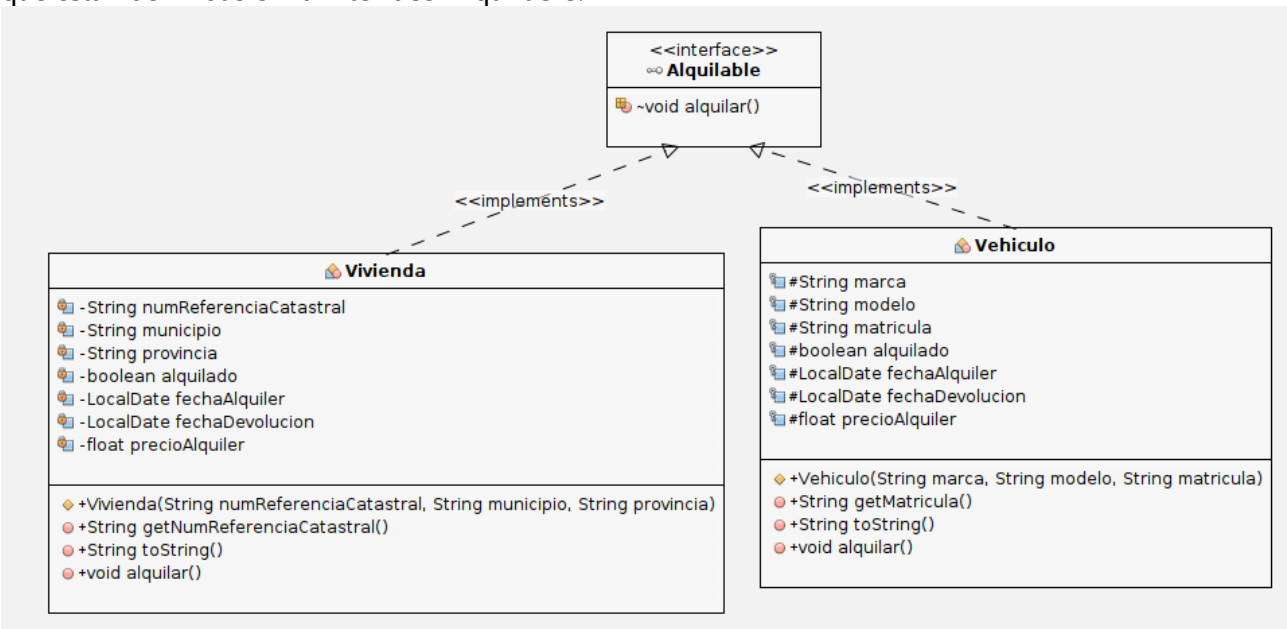
Ej,

```
public interface Alquilable {
    String EMPRESA = "LA EMPRENDEDORA, SA"; //nombre es una constante

    void alquilar(); //es un metodo abstracto
}
```

SUPUESTO1

Clase: Vivienda + Clase: Vehiculo + Interface: Alquilable

Se crearán objetos con estructuras distintas y compartirán métodos en común e incluso, constantes que están definidas en la interface Alquilable.



Output:

```
public class Ppal_Supuesto1 {

    public static void main(String[] args) {
        Vivienda vivienda1 = new Vivienda(numReferenciaCatastral: "Referencia1",municipio: "Murcia",provincia: "Murcia");
        System.out.println("He creado una vivienda para "+Vivienda.EMPRESA);

        Vehiculo vehiculo1 = new Vehiculo(marca: "Marca1",modelo: "Modelo1",matricula: "Matricula1");
        System.out.println("He creado una vehiculo para "+Vehiculo.EMPRESA);

        vivienda1.alquilar();
        vehiculo1.alquilar();
    }
}
```

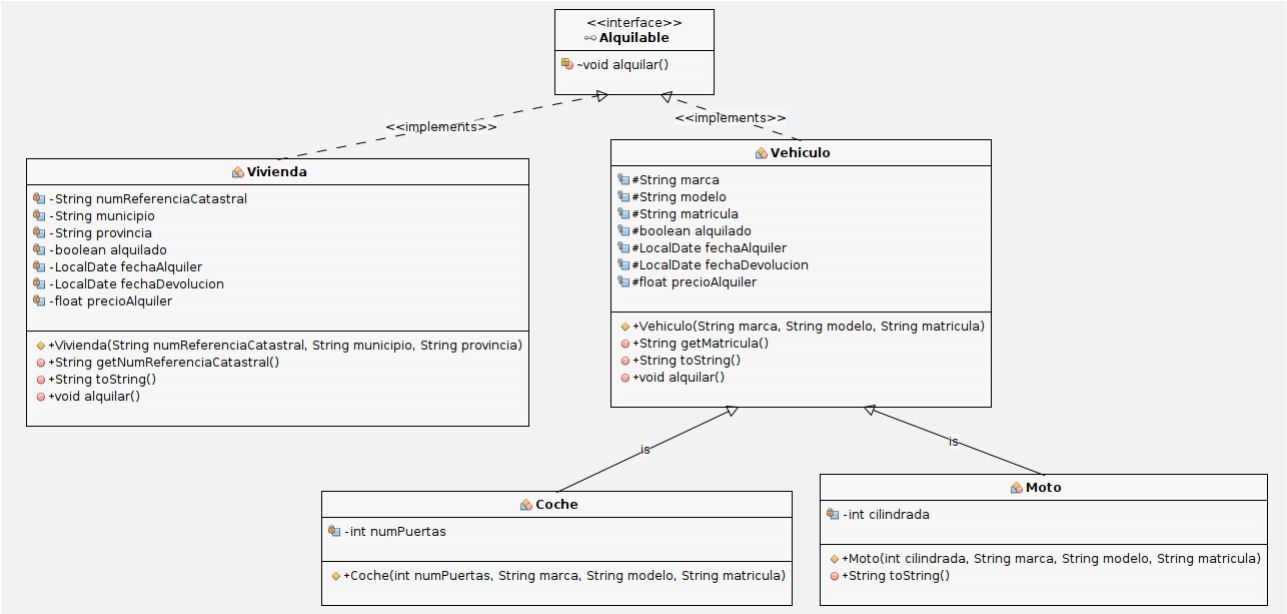
```
He creado una vivienda para LA EMPRENDEDORA, SA
He creado una vehiculo para LA EMPRENDEDORA, SA
Alquilando vivienda: Referencia1
Alquilando vehiculo: Matricula1
```

```java
public class Vivienda implements Alquilable{
    private String numReferenciaCatastral;
    private String municipio;
    private String provincia;
    private boolean alquilado=false;
    private LocalDate fechaAlquiler;
    private LocalDate fechaDevolucion;
    private float precioAlquiler=0;

    public Vivienda(String numReferenciaCatastral, String municipio, String provincia) {
        this.numReferenciaCatastral = numReferenciaCatastral;
        this.municipio = municipio;
        this.provincia = provincia;
    }

    public String getNumReferenciaCatastral() {...3 lines }

    @Override
    public String toString() {...3 lines }

    @Override
    public void alquilar()throws IllegalArgumentException{
        if (this.alquilado) throw new IllegalArgumentException( s: "Ya alquilado");
        System.out.println("Alquilando vivienda: "+this.numReferenciaCatastral);
    }
}
```

```java
public class Vehiculo implements Alquilable{
    protected String marca;
    protected String modelo;
    protected String matricula;
    protected boolean alquilado=false;
    protected LocalDate fechaAlquiler;
    protected LocalDate fechaDevolucion;
    protected float precioAlquiler=0;

    public Vehiculo(String marca, String modelo, String matricula) {
        this.marca = marca;
        this.modelo = modelo;
        this.matricula = matricula;
    }

    public String getMatricula() {...3 lines }

        @Override
    public String toString() {...3 lines }

    @Override
    public void alquilar()throws IllegalArgumentException{
        if (this.alquilado) throw new IllegalArgumentException( s: "Ya alquilado");
        System.out.println("Alquilando vehiculo: "+this.matricula);
    }
}
```

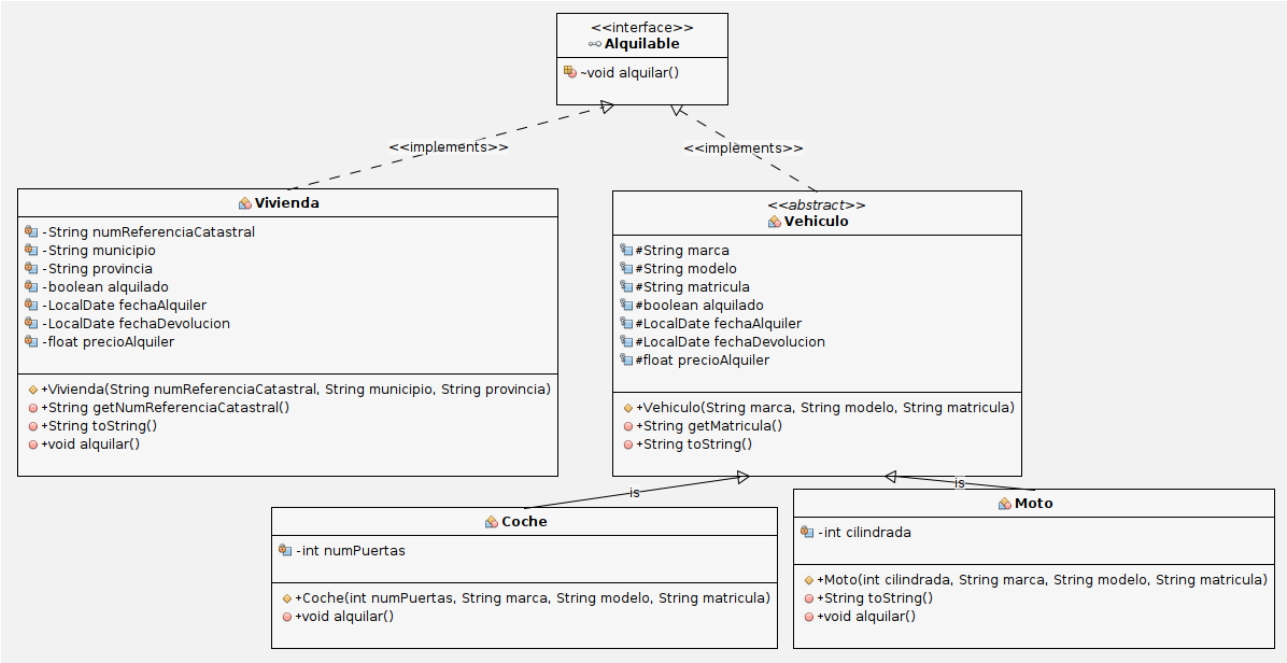
SUPUESTO2

Jerarquía de ClasesVehiculo + Clase Vivienda + Interface Alquilable
Se pueden alquilar objetos de tipo Vehiculo y de sus clases derivadas y de Vivienda.



SUPUESTO3

¿Qué ocurre si la clase Vehiculo hereda el método alquilar(), pero NO introduce código en el método? Respuesta: Hereda un método abstracto, por lo que la clase Vehiculo será abstracta, para poder crear objetos de tipo Moto y Coche, estas clases obligatoriamente deben codificar el método alquilar y no ser abstractas.



```
public class Coche extends Vehiculo{
    private int numPuertas;

    public Coche(int numPuertas, String marca, String modelo, String matricula) {...4 lines }
    @Override
    public void alquilar()throws IllegalArgumentException{
        if (this.alquilado) throw new IllegalArgumentException(s: "Ya alquilado");
        System.out.println("Alquilando coche: "+this.matricula+" con "+this.numPuertas+" puertas.");
    }
}
```

Clase Moto:

```
public class Moto extends Vehiculo{
    private int cilindrada;

    public Moto(int cilindrada, String marca, String modelo, String matricula) {...4 lines }

    @Override
    public String toString() {...3 lines }

    @Override
    public void alquilar()throws IllegalArgumentException{
        if (this.alquilado) throw new IllegalArgumentException(s: "Ya alquilado");
        System.out.println("Alquilando moto: "+this.matricula+" con "+this.cilindrada+" cilindrada.");
    }
}
```

Clase con main():

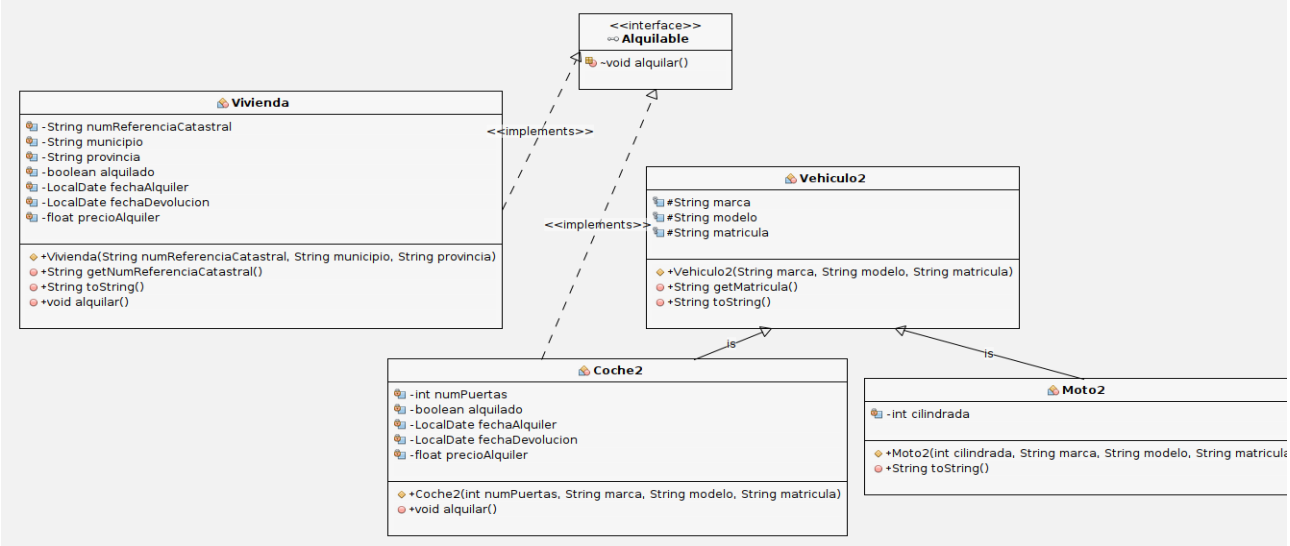
```
public class Ppal_Supuesto3 {

    public static void main(String[] args) {
        Vivienda vivienda1 = new Vivienda(numReferenciaCatastral: "Referencia1",municipio: "Murcia",provincia: "Murcia");

        Moto moto2 = new Moto(cilindrada: 400,marca: "Marca1",modelo: "Modelo1",matricula: "Matricula1");

        Coche coche2 = new Coche(numPuertas: 4,marca: "MarcaCoche2",modelo: "ModeloCoche2",matricula: "MatriculaCoche2");

        vivienda1.alquilar();
        moto2.alquilar();
        coche2.alquilar();
    }
}
```

Output:

```
Alquilando vivienda: Referencia1
Alquilando moto: Matricula1 con 400 cilindrada.
Alquilando coche: MatriculaCoche2 con 4 puertas.
```

SUPUESTO4

Jerarquía de clases Vehiculo2 + Clase Vivienda + Interface Alquilable
Solo se pueden alquilar objetos de tipo Coche2 y de Vivienda. No se alquilan motos.




```java
public class Coche2 extends Vehiculo2 implements Alquilable{
    private int numPuertas;
    private boolean alquilado=false;
    private LocalDate fechaAlquiler;
    private LocalDate fechaDevolucion;
    private float precioAlquiler=0;

    public Coche2(int numPuertas, String marca, String modelo, String matricula) {...4 lines }

    @Override
    public void alquilar()throws IllegalArgumentException{
        if (this.alquilado) throw new IllegalArgumentException(s: "Ya alquilado");
        System.out.println("Alquilando coche: "+this.matricula+" con "+this.numPuertas+" puertas.");
    }
}
```

Clase Vehiculo2

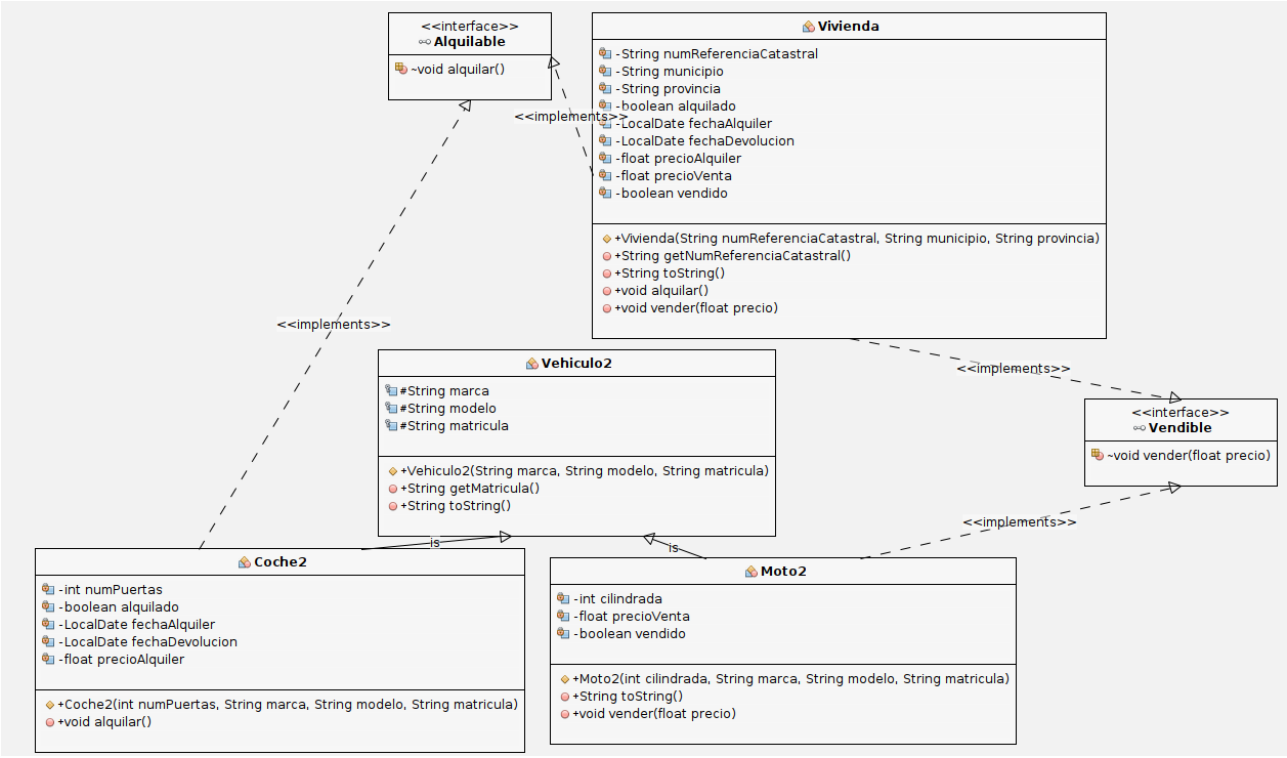
```java
public class Vehiculo2 {
    protected String marca;
    protected String modelo;
    protected String matricula;

    public Vehiculo2(String marca, String modelo, String matricula) {...5 lines }

    public String getMatricula() {...3 lines }

    @Override
    public String toString() {...3 lines }
}
```

Clase con main()

```java
public class Ppal_Supuesto4 {

    public static void main(String[] args) {
        Vivienda vivienda1 = new Vivienda(numReferenciaCatastral: "Referencia1",municipio: "Murcia",provincia: "Murcia");

        Moto2 moto2 = new Moto2(cilindrada: 400,marca: "Marca1",modelo: "Modelo1",matricula: "Matricula1");

        Coche2 coche2 = new Coche2(numPuertas: 4,marca: "MarcaCoche2",modelo: "ModeloCoche2",matricula: "MatriculaCoche2");

        vivienda1.alquilar();
        //moto2.alquilar(); NO se puede
        coche2.alquilar();
    }
}
```

SUPUESTO5

Jerarquía de clases Vehiculo2 + Clase Vivienda + Interface Alquilable + Interface Vendible
Solo se pueden alquilar objetos de tipo Coche2 y de Vivienda. No se alquilan motos.
Solo se pueden vender objetos de tipo Moto2 y de Vivienda. Los coches no se venden.



Una clase puede implementar más de una interface. La clase Vivienda implementa Alquilable y Vendible.
La clase Coche2 extiende de Vehiculo2 e implementa Alquilable
La clase Moto2 extiende de Vehiculo2 e implementa Vendible.
Una subclase también puede implementar varias interfaces.

```
public static void main(String[] args) {
    Vivienda vivienda1 = new Vivienda(numReferenciaCatastral: "Referencia1",municipio: "Murcia",provincia: "Murcia");
    Vivienda vivienda2 = new Vivienda(numReferenciaCatastral: "Referencia2",municipio: "Almeria",provincia: "Almeria");

    Moto2 moto2 = new Moto2(cilindrada: 400,marca: "Marca1",modelo: "Modelo1",matricula: "Matricula1");

    Coche2 coche2 = new Coche2(numPuertas: 4,marca: "MarcaCoche2",modelo: "ModeloCoche2",matricula: "MatriculaCoche2");

    vivienda1.alquilar();
    vivienda2.vender(precio: 120000);
    moto2.vender(precio: 2000);
    coche2.alquilar();
}
```

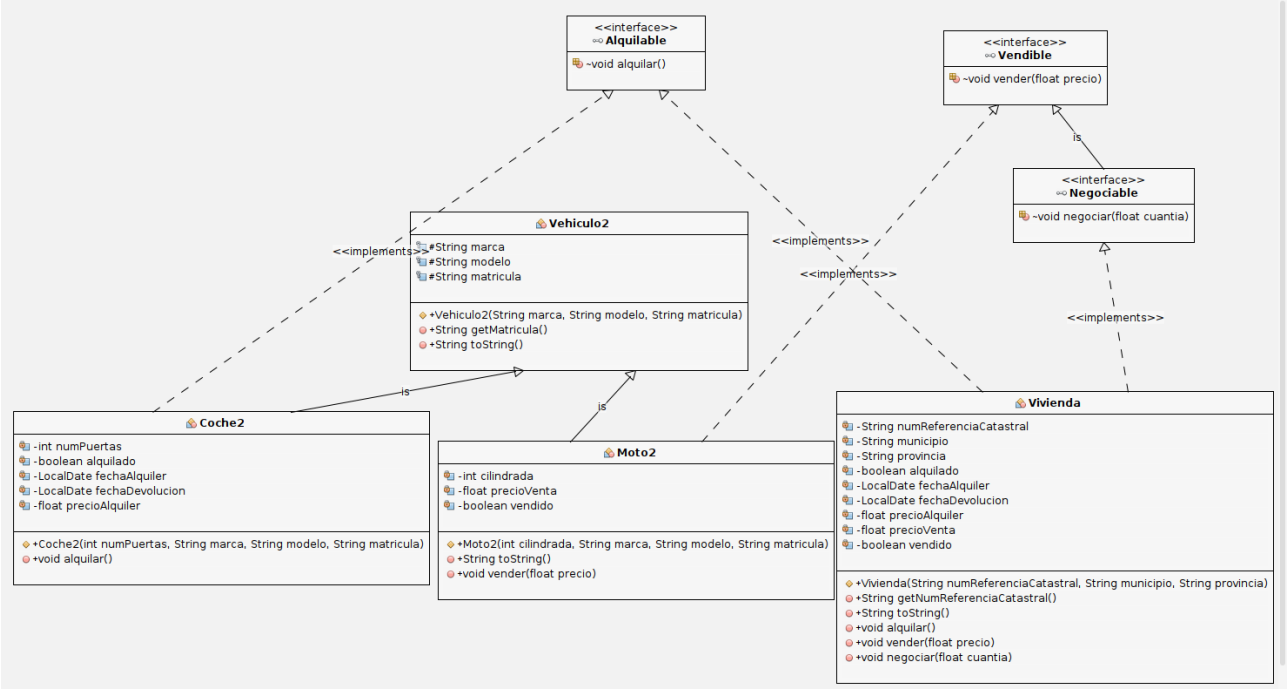
Output:

```
Alquilando vivienda: Referencia1
Vendiendo la vivienda Referencia2.... por 120000.0€
Vendiendo la moto Matricula1.... por 2000.0€
Alquilando coche: MatriculaCoche2 con 4 puertas.
-------------------------------------------------
```

SUPUESTO6

Jerarquía de clases Vehiculo2 + Clase Vivienda + Interface Alquilable + Jerarquía de Interfaces Vendible
Solo se pueden alquilar objetos de tipo Coche2 y de Vivienda. No se alquilan motos.
Solo se pueden vender objetos de tipo Moto2 y de Vivienda. Los coches no se venden.
Los objetos de tipo Vivienda se venden y algunas se negocian restando una cuantia en euros especial. Negociable es una interface hija de Vendible.

Se puede establecer una jerarquía de interfaces.
public interface Negociable extends Vendible{ …..}



```java
 */
public class Vivienda implements Alquilable, Negociable{
    private String numReferenciaCatastral;
    private String municipio;
    private String provincia;
    private boolean alquilado=false;
    private LocalDate fechaAlquiler;
    private LocalDate fechaDevolucion;
    private float precioAlquiler=0;
    private float precioVenta;
    private boolean vendido=false;

    public Vivienda(String numReferenciaCatastral, String municipio, String provincia) {
        this.numReferenciaCatastral = numReferenciaCatastral;
        this.municipio = municipio;
        this.provincia = provincia;
    }

    public String getNumReferenciaCatastral() {...3 lines }

    @Override
    public String toString() {...3 lines }

    @Override
    public void alquilar()throws IllegalArgumentException{
        if (this.alquilado) throw new IllegalArgumentException(s: "Ya alquilado");
        System.out.println("Alquilando vivienda: "+this.numReferenciaCatastral);
    }

    @Override
    public void vender(float precio) throws IllegalArgumentException{
        if (this.vendido) throw new IllegalArgumentException(s: "Error, la vivienda ya esta vendida");
        System.out.println("Vendiendo la vivienda .... por "+precio+"€");
    }

    @Override
    public void negociar(float cuantia)throws IllegalArgumentException{
        if (this.vendido) throw new IllegalArgumentException(s: "Error, la vivienda ya esta vendida");
        System.out.println("Negociando la vivienda con prestamo de ... "+cuantia+"€");
    }
}
```

```java
public static void main(String[] args) {
    Vivienda vivienda1 = new Vivienda(numReferenciaCatastral: "Referencia1",municipio: "Murcia",provincia: "Murcia");
    Vivienda vivienda2 = new Vivienda(numReferenciaCatastral: "Referencia2",municipio: "Almeria",provincia: "Almeria");
    Vivienda vivienda3 = new Vivienda(numReferenciaCatastral: "Referencia3",municipio: "Roquetas",provincia: "Almeria");

    Moto2 moto2 = new Moto2(cilindrada: 400,marca: "Marca1",modelo: "Modelo1",matricula: "Matricula1");

    Coche2 coche2 = new Coche2(numPuertas: 4,marca: "MarcaCoche2",modelo: "ModeloCoche2",matricula: "MatriculaCoche2");

    vivienda1.alquilar();
    vivienda2.vender(precio: 120000);
    vivienda3.vender(precio: 130000);
    vivienda3.negociar(cuantia: 10000);
    moto2.vender(precio: 2000);
    coche2.alquilar();
}
```

Output:

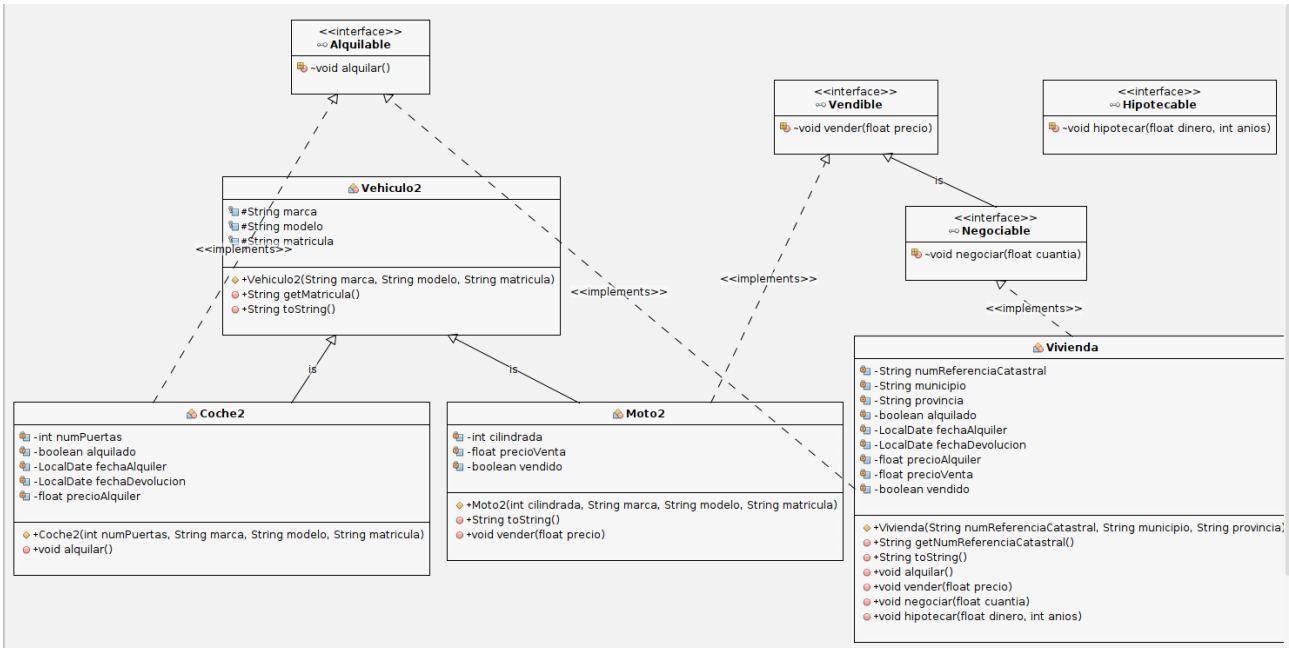
```
Alquilando vivienda: Referencia1
Vendiendo la viviendaReferencia2 .... por 120000.0€
Vendiendo la viviendaReferencia3 .... por 130000.0€
Negociando la vivienda Referencia3 con prestamo de ... 10000.0€
Vendiendo la motor Matricula1 .... por 2000.0€
Alquilando coche: MatriculaCoche2 con 4 puertas.
------------------------------------------------------------------------
```

SUPUESTO7

Jerarquía de clases Vehiculo2 + Clase Vivienda + Interface Alquilable + Jerarquía de Interfaces Vendible + Interface Hipotecable
Solo se pueden alquilar objetos de tipo Coche2 y de Vivienda. No se alquilan motos.
Solo se pueden vender objetos de tipo Moto2 y de Vivienda. Los coches no se venden.
Los objetos de tipo Vivienda se venden y algunas se negocian restando una cuantia en euros especial. Negociable es una interface hija de Vendible.
La interface Hipotecable tiene un método: hipotecar(float dinero, int anios) throws IllegalArgumentException

Se puede establecer una jerarquía de interfaces.
public interface Negociable extends Vendible{ …..}

Una interface puede heredar más de una interface (tiene Herencia múltiple)
public interface Negociable extends Vendible, Hipotecable {…..}




```
public interface Hipotecable {
    void hipotecar(float dinero, int anios) throws IllegalArgumentException;
}
```

```
public interface Negociable extends Vendible, Hipotecable {
    void negociar(float cuantia) throws IllegalArgumentException;
}
```

```java
public class Vivienda implements Alquilable, Negociable{
    private String numReferenciaCatastral,municipio,provincia;
    private boolean alquilado, vendido=false;
    private LocalDate fechaAlquiler, fechaDevolucion;
    private float precioAlquiler, precioVenta;

    public Vivienda(String numReferenciaCatastral, String municipio, String provincia) {...5 lines }
    public String getNumReferenciaCatastral() {...3 lines }
    @Override
    public String toString() {...3 lines }
    @Override
    public void alquilar()throws IllegalArgumentException{...4 lines }

    @Override
    public void vender(float precio) throws IllegalArgumentException{...4 lines }

    @Override
    public void negociar(float cuantia)throws IllegalArgumentException{...4 lines }

    @Override
    public void hipotecar(float dinero, int anios) throws IllegalArgumentException{
        if (this.vendido) throw new IllegalArgumentException(s: "Error, la vivienda ya esta vendida");
        System.out.println("Hipotecando la vivienda "+this.numReferenciaCatastral+" por ... "+dinero+"€ durante: "+anios+" años");
    }
}
```

```java
public static void main(String[] args) {
    Vivienda vivienda1 = new Vivienda(numReferenciaCatastral: "Referencia1",municipio: "Murcia",provincia: "Murcia");
    Vivienda vivienda2 = new Vivienda(numReferenciaCatastral: "Referencia2",municipio: "Almeria",provincia: "Almeria");
    Vivienda vivienda3 = new Vivienda(numReferenciaCatastral: "Referencia3",municipio: "Roquetas",provincia: "Almeria");
    Vivienda vivienda4 = new Vivienda(numReferenciaCatastral: "Referencia4",municipio: "Montealegre",provincia: "Teruel");

    Moto2 moto2 = new Moto2(cilindrada: 400,marca: "Marca1",modelo: "Modelo1",matricula: "Matricula1");

    Coche2 coche2 = new Coche2(numPuertas: 4,marca: "MarcaCoche2",modelo: "ModeloCoche2",matricula: "MatriculaCoche2");

    vivienda1.alquilar();
    vivienda2.vender(precio: 120000);
    vivienda3.vender(precio: 130000);
    vivienda3.negociar(cuantia:10000);

    vivienda4.hipotecar(dinero: 90000, anios: 4);

    moto2.vender(precio: 2000);
    coche2.alquilar();
}
```

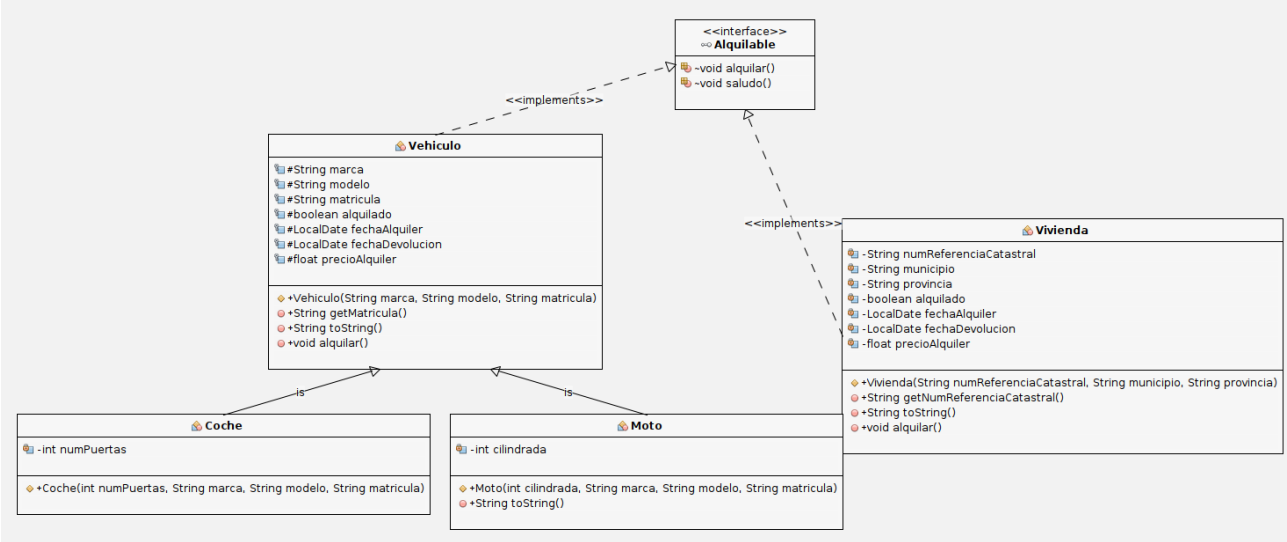
Output:

```
Alquilando vivienda: Referencia1
Vendiendo la viviendaReferencia2 .... por 120000.0€
Vendiendo la viviendaReferencia3 .... por 130000.0€
Negociando la vivienda Referencia3 con prestamo de ... 10000.0€
Hipotecando la vivienda Referencia4 por ... 90000.0€ durante: 4 años
Vendiendo la motor Matricula1 .... por 2000.0€
Alquilando coche: MatriculaCoche2 con 4 puertas.
```

SUPUESTO8

Jerarquía de ClasesVehiculo + Clase Vivienda + Interface Alquilable
Se pueden alquilar objetos de tipo Vehiculo y de Vivienda.

Hay otro tipo de métodos en interfaces: default, son métodos que tienen código y ese método ya se hereda codificado, con posibilidad de readaptar solo en la clase que lo implementa.




```java
public interface Alquilable {
    String EMPRESA = "LA EMPRENDEDORA, SA"; //nombre es una constante

    void alquilar() throws IllegalArgumentException; //es un metodo abstracto
    default void saludo(){
        System.out.println(x: "BIENVENIDOS, WELCOME!!!");
    }
}
```

Las clases Vehiculo y Vivienda no sobreescriben el método saludo de Alquilable en este ejemplo, aunque sí que lo pueden hacer.

Los métodos default NO se heredan en las clases derivadas de la clase que implementa la interface.

```java
    Moto moto2 = new Moto(cilindrada: 400,marca: "Marca1",modelo: "Modelo1",matricula: "Matricula1");

    Coche coche2 = new Coche(numPuertas: 4,marca: "MarcaCoche2",modelo: "ModeloCoche2",matricula: "MatriculaCoche2");

    vivienda1.alquilar();
    moto2.alquilar();
    coche2.alquilar();

    vivienda1.saludo();

    Vehiculo vehiculo1 = new Vehiculo(marca: "MarcaVehiculo1", modelo: "ModeloVehiculo1", matricula: "MatriculaVehiculo1");
    vehiculo1.saludo();

    //Los metodos default NO se heredan en moto ni coche, solo lo puede utilizar Vehiculo
    //moto2.saludo();
    //coche2.saludo();
}
```

Output:

```
Alquilando vivienda: Referencia1
Alquilando vehiculo: Matricula1
Alquilando vehiculo: MatriculaCoche2
BIENVENIDOS, WELCOME!!!
- BIENVENIDOS, WELCOME!!!
```

SUPUESTO9

SUPUESTO8 con la clase Vivienda readaptando el método saludo.

```
public class Vivienda implements Alquilable{
    private String numReferenciaCatastral;
    private String municipio;
    private String provincia;
    private boolean alquilado=false;
    private LocalDate fechaAlquiler;
    private LocalDate fechaDevolucion;
    private float precioAlquiler=0;

    public Vivienda(String numReferenciaCatastral, String municipio, String provincia) {...5 lines }

    public String getNumReferenciaCatastral() {...3 lines }

    @Override
    public String toString() {...3 lines }

    @Override
    public void alquilar()throws IllegalArgumentException{
        if (this.alquilado) throw new IllegalArgumentException(s: "Ya alquilado");
        System.out.println("Alquilando vivienda: "+this.numReferenciaCatastral);
    }

    @Override
    public void saludo(){
        System.out.println(x: "CIAO BENVENUTO");
    }
}
```

Clase con main():

```
public static void main(String[] args) {
    Vivienda vivienda1 = new Vivienda(numReferenciaCatastral: "Referencia1",municipio: "Murcia",provincia: "Murcia");

    Moto moto2 = new Moto(cilindrada: 400,marca: "Marca1",modelo: "Modelo1",matricula: "Matricula1");

    Coche coche2 = new Coche(numPuertas: 4,marca: "MarcaCoche2",modelo: "ModeloCoche2",matricula: "MatriculaCoche2");

    vivienda1.alquilar();
    moto2.alquilar();
    coche2.alquilar();

    vivienda1.saludo();

    Vehiculo vehiculo1 = new Vehiculo(marca: "MarcaVehiculo1", modelo: "ModeloVehiculo1", matricula: "MatriculaVehiculo1");
    vehiculo1.saludo();

    //Los metodos default NO se heredan en moto ni coche, solo lo puede utilizar Vehiculo
    //moto2.saludo();
    //coche2.saludo();
}
```

Output:

```
Alquilando vivienda: Referencia1
Alquilando vehiculo: Matricula1
Alquilando vehiculo: MatriculaCoche2
CIAO BENVENUTO
BIENVENIDOS, WELCOME!!!
```

SUPUESTO10

Jerarquía de ClasesVehiculo + Clase Vivienda + Interface Alquilable con una interface hija: Reservable

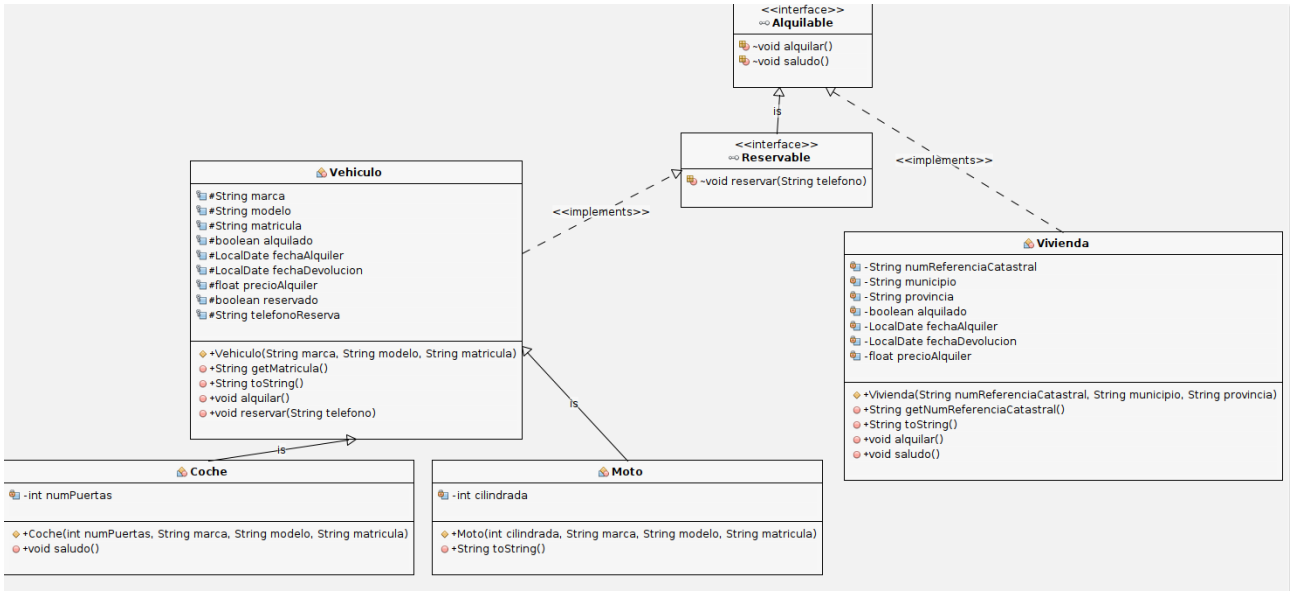
Se pueden alquilar objetos de tipo Vehiculo y de Vivienda. Y también reservar para su alquiler.
Los de tipo Vehiculo se alquilan para una semana máximo(7 días), y permite una sola reserva guardando el número de teléfono.
Los de tipo Vivienda NO se pueden reservar.

Método: default, son métodos que tienen código y ese método ya se hereda codificado, con posibilidad de readaptar solo en la clase que lo implementa.

Si una interface hereda de otra un método default:

a) Lo deja como está, y se hereda como default.
b) Reclara el método, con lo cual, lo deja abstracto
c) Lo puede redefinir o sobreescribir

Se elige la opción a) La interface **Reservable lo hereda** como default. * El método reservar() solo lo puede usar Vehiculo.




```java
public class Vehiculo implements Reservable{
    protected String marca;
    protected String modelo;
    protected String matricula;
    protected boolean alquilado=false;
    protected LocalDate fechaAlquiler;
    protected LocalDate fechaDevolucion;
    protected float precioAlquiler=0;
    protected boolean reservado=false;
    protected String telefonoReserva;

    public Vehiculo(String marca, String modelo, String matricula) {...5 lines }

    public String getMatricula() {...3 lines }

        @Override
    public String toString() {...3 lines }

    @Override
    public void alquilar()throws IllegalArgumentException{
        if (this.alquilado) throw new IllegalArgumentException(s: "Ya alquilado");
        System.out.println("Alquilando vehiculo: "+this.matricula);
        this.alquilado=true;
        this.fechaAlquiler=LocalDate.now();
        this.fechaDevolucion=this.fechaAlquiler.plusDays(daysToAdd: 7);
    }

    @Override
    public void reservar(String telefono)throws IllegalArgumentException{
        if (this.reservado) throw new IllegalArgumentException("Error, el vehiculo "+this.matricula+" ya esta reservado");
        System.out.println("Reservado el vehiculo: "+this.matricula);
        this.reservado=true;
        this.telefonoReserva=telefono;
    }
}
```

```java
public interface Reservable extends Alquilable{
    void reservar(String telefono)throws IllegalArgumentException;
}
```

Clase con main()

```java
public static void main(String[] args) {
    Vivienda vivienda1 = new Vivienda(numReferenciaCatastral: "Referencia1",municipio: "Murcia",provincia: "Murcia");

    Moto moto2 = new Moto(cilindrada: 400,marca: "Marca1",modelo: "Modelo1",matricula: "Matricula1");

    Coche coche2 = new Coche(numPuertas: 4,marca: "MarcaCoche2",modelo: "ModeloCoche2",matricula: "MatriculaCoche2");

    vivienda1.alquilar();
    vivienda1.saludo();

    moto2.alquilar();
    coche2.alquilar();

    Vehiculo vehiculo1 = new Vehiculo(marca: "MarcaVehiculo1", modelo: "ModeloVehiculo1", matricula: "MatriculaVehiculo1");
    vehiculo1.saludo();
    vehiculo1.alquilar();
    vehiculo1.reservar(telefono: "12345");

    //moto2.saludo(); No se puede
    //moto2.reservar(); NO se puede
}
```

Output:

```
Alquilando vivienda: Referencia1
CIAO BENVENUTO
Alquilando vehiculo: Matricula1
Alquilando vehiculo: MatriculaCoche2
BIENVENIDOS, WELCOME!!!
Alquilando vehiculo: MatriculaVehiculo1
Reservado el vehiculo: MatriculaVehiculo1
```

SUPUESTO11

Jerarquía de ClasesVehiculo + Clase Vivienda + Interface Alquilable con una interface hija: Reservable

Se pueden alquilar objetos de tipo Vehiculo y de Vivienda. Y también reservar para su alquiler.
Los de tipo Vehiculo se alquilan para una semana máximo(7 días), y permite una sola reserva guardando el número de teléfono.
Los de tipo Vivienda NO se pueden reservar.

Método: default, son métodos que tienen código y ese método ya se hereda codificado, con posibilidad de readaptar solo en la clase que lo implementa.

Si una interface hereda de otra un método default:

a) Lo deja como está, y se hereda como default.
b) Reclara el método, con lo cual, lo deja abstracto
c) Lo puede redefinir o sobreescribir

Se elige la opción b) La interface **Reservable redeclara dejándolo como abstracto**. * Los métodos saludo() y reservar() ya se puede usar en las clases derivadas de Vehiculo

```
public interface Reservable extends Alquilable{
    void reservar(String telefono)throws IllegalArgumentException;

    @Override
    void saludo();
}
```

En la clase Vehiculo se obliga a codificar el método saludo(), ya que si no se hace, la clase Vehiculo será abstracta y obligará a que las clases Moto y Coche lo codifiquen.

```
public class Vehiculo implements Reservable{
    protected String marca;
    protected String modelo;
    protected String matricula;
    protected boolean alquilado=false;
    protected LocalDate fechaAlquiler;
    protected LocalDate fechaDevolucion;
    protected float precioAlquiler=0;
    protected boolean reservado=false;
    protected String telefonoReserva;

    public Vehiculo(String marca, String modelo, String matricula) {...5 lines }

    public String getMatricula() {...3 lines }

        @Override
    public String toString() {...3 lines }

    @Override
    public void alquilar()throws IllegalArgumentException{...7 lines }

    @Override
    public void reservar(String telefono)throws IllegalArgumentException{...6 lines }

    @Override
    public void saludo(){
        System.out.println( x: "Paso de ser default en Alquilable a ser abstract en Reservable, ahora tengo cuerpo ;-)");
    }
}
```

Clase con main():

```
public class Ppal_Supuesto11 {
    public static void main(String[] args) {
        Vivienda vivienda1 = new Vivienda(numReferenciaCatastral: "Referencia1",municipio: "Murcia",provincia: "Murcia");

        Moto moto2 = new Moto(cilindrada: 400,marca: "Marca1",modelo: "Modelo1",matricula: "Matricula1");

        Coche coche2 = new Coche(numPuertas: 4,marca: "MarcaCoche2",modelo: "ModeloCoche2",matricula: "MatriculaCoche2");

        vivienda1.alquilar();
        vivienda1.saludo();

        moto2.alquilar();
        coche2.alquilar();

        Vehiculo vehiculo1 = new Vehiculo(marca: "MarcaVehiculo1", modelo: "ModeloVehiculo1", matricula: "MatriculaVehiculo1");
        vehiculo1.saludo();
        vehiculo1.alquilar();
        vehiculo1.reservar(telefono: "12345");

        moto2.saludo(); //ahora SI se puede
        moto2.reservar(telefono: "1233");// ahora SI se puede
    }
}
```

Output:

```
Alquilando vivienda: Referencia1
CIAO BENVENUTO
Alquilando vehiculo: Matricula1
Alquilando vehiculo: MatriculaCoche2
Paso de ser default en Alquilable a ser abstract en Reservable, ahora tengo cuerpo ;-)
Alquilando vehiculo: MatriculaVehiculo1
Reservado el vehiculo: MatriculaVehiculo1
Paso de ser default en Alquilable a ser abstract en Reservable, ahora tengo cuerpo ;-)
Reservado el vehiculo: Matricula1
```

SUPUESTO12

Jerarquía de ClasesVehiculo + Clase Vivienda + Interface Alquilable con una interface hija: Reservable
Se pueden alquilar objetos de tipo Vehiculo y de Vivienda. Y también reservar para su alquiler.
Los de tipo Vehiculo se alquilan para una semana máximo(7 días), y permite una sola reserva guardando el número de teléfono.
Los de tipo Vivienda NO se pueden reservar.

Método: default, son métodos que tienen código y ese método ya se hereda codificado, con posibilidad de readaptar solo en la clase que lo implementa.

Si una interface hereda de otra un método default:

a) Lo deja como está, y se hereda como default.
b) Reclara el método, con lo cual, lo deja abstracto
c) Lo puede redefinir o sobreescribir

Se elige la opción c) La interface **Reservable redefine el método saludo()**. * Los métodos saludo() y reservar() ya se puede usar en las clases derivadas de Vehiculo

```
public interface Reservable extends Alquilable{
    void reservar(String telefono)throws IllegalArgumentException;

    @Override
    default void saludo(){
        System.out.println(x: "Soy el metodo: default SALUDO @Override de Reservable");
    }
}
```

```
public class Ppal_Supuesto12 {
    public static void main(String[] args) {
        Vivienda vivienda1 = new Vivienda(numReferenciaCatastral: "Referencia1",municipio: "Murcia",provincia: "Murcia");

        Moto moto2 = new Moto(cilindrada: 400,marca: "Marca1",modelo: "Modelo1",matricula: "Matricula1");

        Coche coche2 = new Coche(numPuertas: 4,marca: "MarcaCoche2",modelo: "ModeloCoche2",matricula: "MatriculaCoche2");

        vivienda1.alquilar();
        vivienda1.saludo();

        moto2.alquilar();
        coche2.alquilar();

        Vehiculo vehiculo1 = new Vehiculo(marca: "MarcaVehiculo1", modelo: "ModeloVehiculo1", matricula: "MatriculaVehiculo1");
        vehiculo1.saludo();
        vehiculo1.alquilar();
        vehiculo1.reservar(telefono: "12345");

        moto2.saludo(); //ahora SI se puede
        moto2.reservar(telefono: "1233");// ahora SI se puede
    }
}
```

Output:

```
Alquilando vehiculo: Matricula1
Alquilando vehiculo: MatriculaCoche2
Soy el metodo: default SALUDO @Override de Reservable
Alquilando vehiculo: MatriculaVehiculo1
Reservado el vehiculo: MatriculaVehiculo1
Soy el metodo: default SALUDO @Override de Reservable
Reservado el vehiculo: Matricula1
```

SUPUESTO13

Jerarquía de ClasesVehiculo + Clase Vivienda + Interface Alquilable

Método: static, son métodos que tienen código y ese método NO se hereda, tan solo se puede acceder a ese método mediante una referencia a esa interface.

```
public interface Alquilable {
    String EMPRESA = "LA EMPRENDEDORA, SA"; //nombre es una constante

    void alquilar() throws IllegalArgumentException; //es un metodo abstracto

    static void saludo(){
        System.out.println(x: "BIENVENIDOS, WELCOME!!!");
    }
}
```

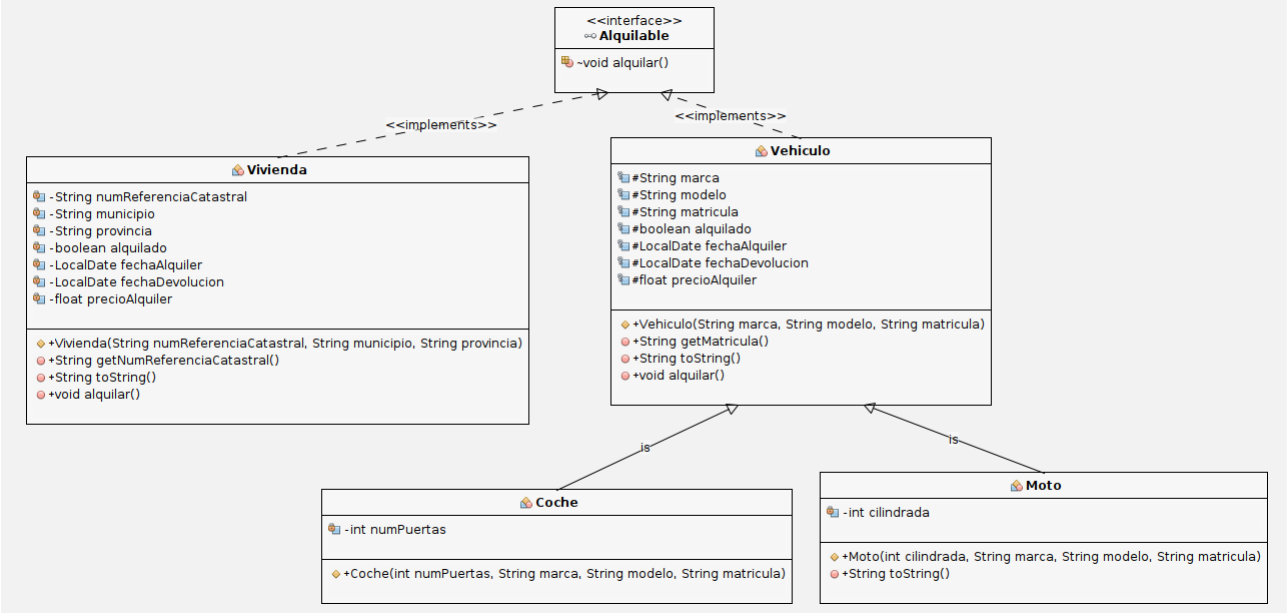
```
public class Ppal_Supuesto13 {

    public static void main(String[] args) {
        Vivienda vivienda1 = new Vivienda(numReferenciaCatastral: "Referencia1",municipio: "Murcia",provincia: "Murcia");

        Moto moto2 = new Moto(cilindrada: 400,marca: "Marca1",modelo: "Modelo1",matricula: "Matricula1");

        Coche coche2 = new Coche(numPuertas: 4,marca: "MarcaCoche2",modelo: "ModeloCoche2",matricula: "MatriculaCoche2");

        vivienda1.alquilar();
        moto2.alquilar();
        coche2.alquilar();

        Alquilable.saludo(); //es un metodo estatico y se puede ejecutar como tal
    }
}
```

Output:

```
Alquilando vivienda: Referencia1
Alquilando vehiculo: Matricula1
Alquilando vehiculo: MatriculaCoche2
BIENVENIDOS, WELCOME!!!
```

SUPUESTO14 -POLIMORFISMO, DOWNCASTING EXPLICITO

Jerarquía de ClasesVehiculo + Clase Vivienda + Interface Alquilable




```java
package Supuesto14;


import java.util.ArrayList;
public class Ppal_Supuesto14 {

    public static void main(String[] args) {
        Vivienda vivienda1 = new Vivienda(numReferenciaCatastral:"Referencia1",municipio: "Murcia",provincia: "Murcia");
        vivienda1.alquilar();
        System.out.println("El numero de Referencia catastral de vivienda1 es "+vivienda1.getNumReferenciaCatastral());
        System.out.println("Datos de vivienda1 :"+vivienda1);

        //Polimorfismo
        Alquilable alq = vivienda1; //alq apunta a vivienda1, se transforma en ivienda1
        alq.alquilar(); //tan solo puede ejecutar el metodo alquilar ya que es Alquilable, y tambien toString al ser de Object

        //Ademas tambien se puede hacer downcasting explicito
        System.out.println("Referencia catastral de vivienda1 "+((Vivienda)alq).getNumReferenciaCatastral());
        System.out.println("Datos de alq1 que apunta a vivienda1 "+ alq); //ejecuta el metodo toString de la Vivienda


        Vehiculo vehiculo1 = new Vehiculo(marca: "Marca1",modelo: "Modelo1",matricula: "Matricula1");
        vehiculo1.alquilar();
        System.out.println("Matricula de vehiculo1: "+vehiculo1.getMatricula());
        System.out.println("Datos completos de vehiculo1: "+vehiculo1);

        Moto moto2 = new Moto(cilindrada:400,marca: "Marca1",modelo: "Modelo1",matricula: "Matricula1");
        moto2.alquilar();
        System.out.println("Matricula de moto2: "+moto2.getMatricula());
        System.out.println("Datos completos de moto2: "+moto2);

        //Polimorfismo Ahora alq va a apuntar a moto2 -> se trasnforma en moto2
        alq = moto2;
        alq.alquilar(); //vehiculo1 se alquila por medio de alq
```

```java
    //DownCasting Explicito para alq apuntando a moto2
    System.out.println("Matricula de alq apuntando a moto2: "+((Moto)alq).getMatricula());
    System.out.println("Datos completos de alq apuntando a moto2: "+alq);//ejecuta el metodo toString de Moto

    //Polimorfismo  Ahora alq apunta a una zona de memoria sin nombre que guardainformacion de un coche
    alq = new Coche(numPuertas:4,marca: "MarcaCoche2",modelo: "ModeloCoche2",matricula: "MatriculaCoche2");
    alq.alquilar(); // se alquila el coche
    System.out.println("Matricula de alq apuntando a un coche: "+((Coche)alq).getMatricula());
    System.out.println("Datos completos de alq apuntando a un coche: "+alq);//ejecuta el metodo toString de Coche


    //Incluso se puede crear un ArrayList de Interface
    ArrayList <Alquilable> alquileres = new ArrayList <Alquilable>();
    alquileres.add(e:alq); //acabo de guardar el coche
    alquileres.add(e:vivienda1); //internamente un puntero de tipo Interface Alquilable va a apuntar a vivienda1
    alquileres.add(e:moto2);//internamente un puntero de tipo Interface Alquilable va a apuntar a vivienda1

    //Especificando el tipo
    System.out.println(x:"\n-- LISTADO DEL ARRAY--");
    for (Alquilable a: alquileres) {
        if (a instanceof Vivienda) {
            System.out.println("Vivienda :"+((Vivienda) a).getNumReferenciaCatastral());
        }else if (a instanceof Moto){
            System.out.println("Moto :"+((Moto) a).getMatricula());
        }else{
             System.out.println("Coche :"+((Coche) a).getMatricula());
        }
    }
```

```java
        //Otra forma
        System.out.println(x:"\n-- LISTADO DEL ARRAY--");
        for (Alquilable a: alquileres) {
            if (a instanceof Vivienda vivienda) {
                System.out.println("Vivienda :"+vivienda.getNumReferenciaCatastral());
            }else if (a instanceof Moto moto){
                System.out.println("Moto :"+moto.getMatricula());
            }else{
                 System.out.println("Coche :"+((Coche) a).getMatricula());
            }
        }


    }
}
```