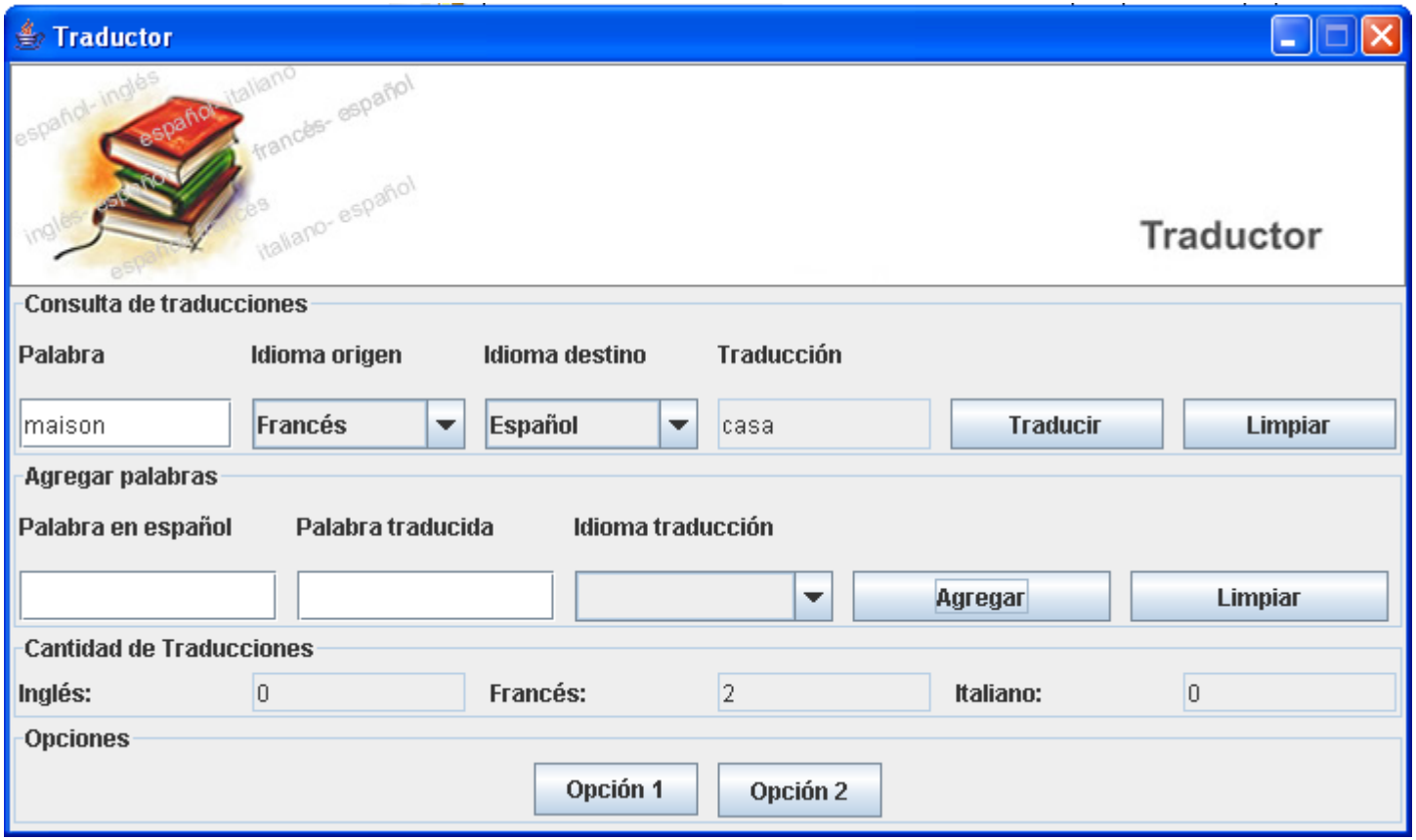
Búsqueda, ordenamiento y pruebas automáticas

Temas

1. Invariantes de clase
2. Pruebas Automáticas Unitarias
3. Algoritmo de Ordenamiento
4. Algoritmo de Búsqueda

Caso de Estudio: Traductor



Ejercicio

(Caso de Estudio: Traductor)

1. Lea el enunciado e identifique la información importante
2. Identifique los Requerimientos Funcionales
3. Dibuje el modelo del mundo del problema

Comprensión y Especificación del Problema

Requerimientos Funcionales (Caso de Estudio: Traductor)

Nombre	R1: Agregar una palabra en español a un diccionario
Resumen	
Entradas	
Resultados	

Comprensión y Especificación del Problema

Requerimientos Funcionales (Caso de Estudio: Traductor)

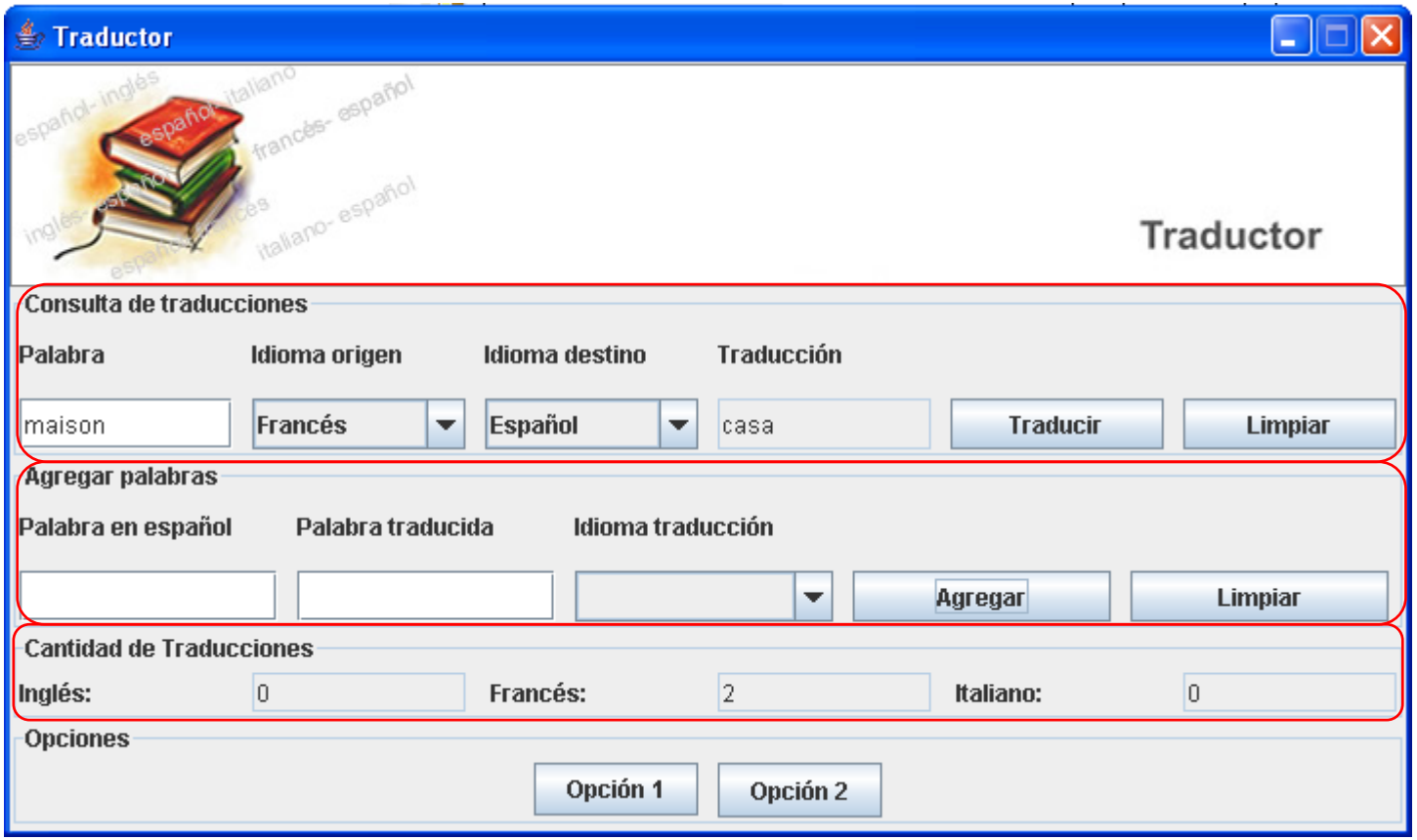
Nombre	R2: Consultar la traducción de una palabra
Resumen	Dada una palabra y el idioma en el que se encuentra, consultar su respectiva traducción en un idioma dado (español, inglés, italiano o francés)
Entradas	
Palabra a ser traducida	
Idioma en el que se encuentra la palabra	
Idioma en el que se quiere buscar la traducción de la palabra	
Resultados	
La palabra traducida	

Comprensión y Especificación del Problema

Requerimientos Funcionales (Caso de Estudio: Traductor)

Nombre	R3: Consultar el número de palabras de cada diccionario
Resumen	Para cada diccionario (español-inglés, español-italiano, español-francés) mostrar el número de palabras que hay
Entradas	
Ninguna	
Resultados	
Total de palabras que hay en cada diccionario del traductor	
Resultados	

Caso de Estudio: Traductor



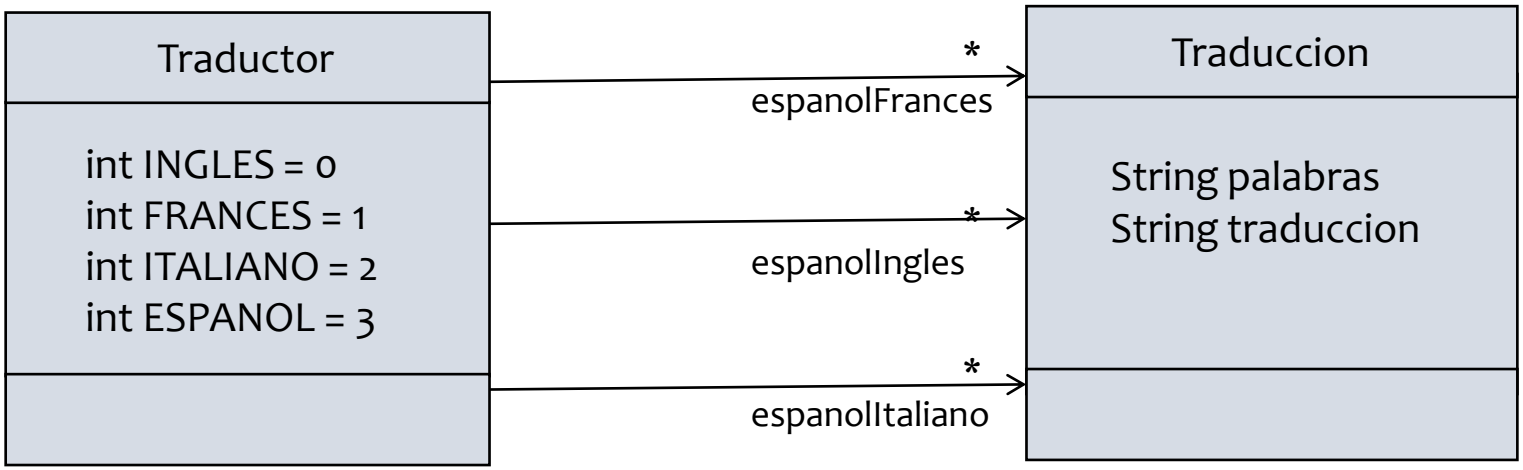
Análisis del problema

- Identificar entidades del mundo del problema
- Identificar atributos y sus tipos
- Identificar constantes
- Buscar relaciones entre entidades y sus propiedades
- Escribir el diagrama de clases en UML

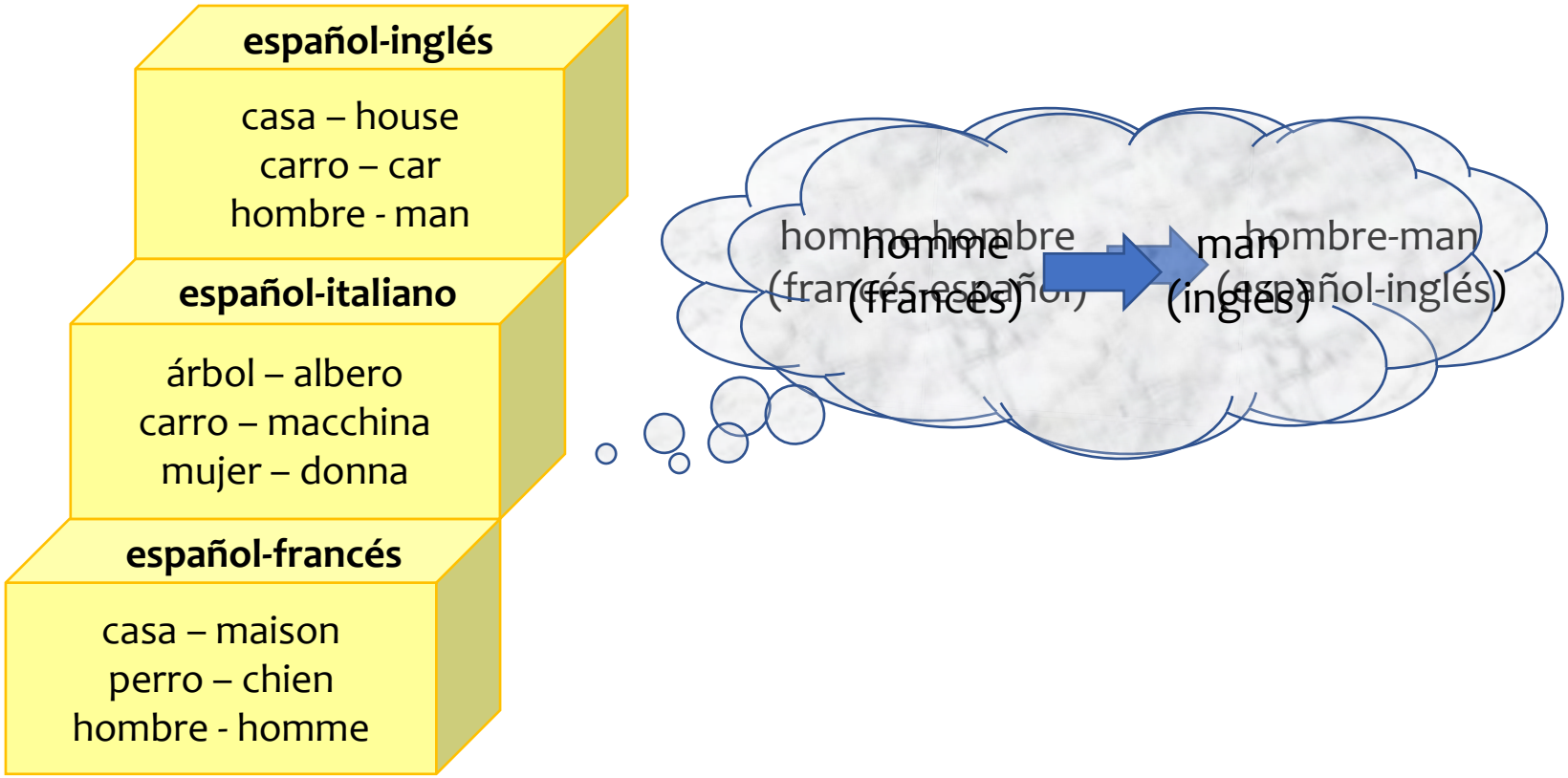
Análisis del Problema

Diagrama UML (Caso de Estudio: Traductor)

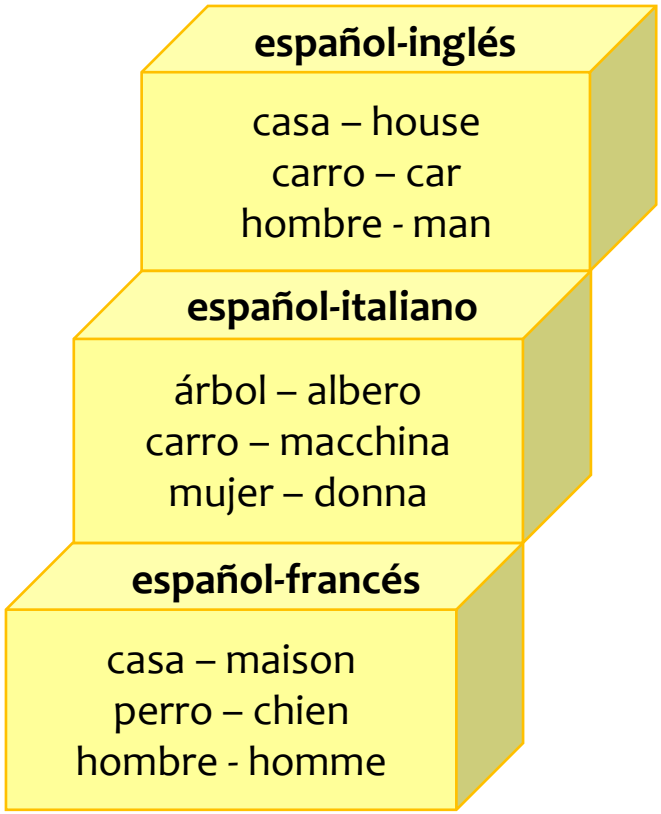
Identificar entidades del mundo del problema



Caso de Estudio: Traductor



Caso de Estudio: Traductor



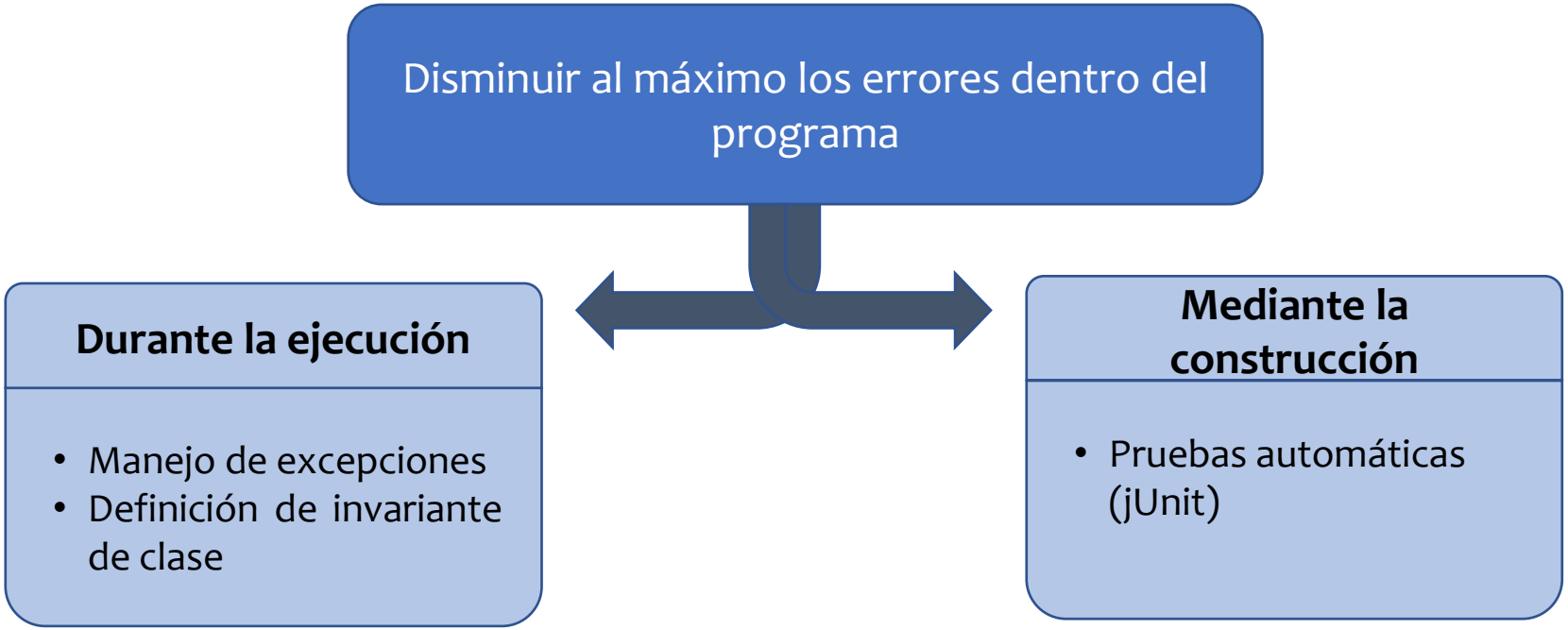
Restricciones

1. No hay dos palabras en español repetidas en ningún diccionario. Cada palabra tiene sólo una traducción en el otro idioma.

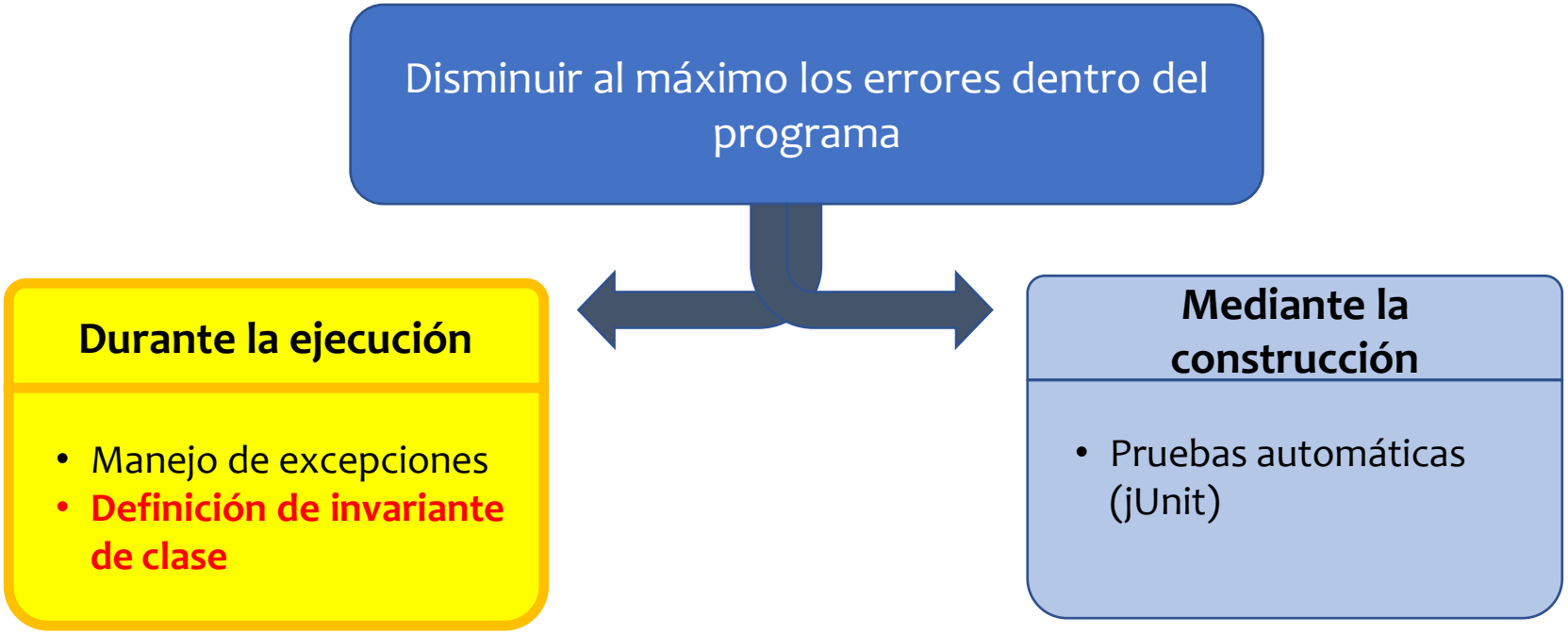
2. Las traducciones son únicas.

Verificación y Pruebas de programa

Verificación y Pruebas de programa



Verificación y Pruebas de programa



Invariante de clase

¿Qué información no quedó plasmada ni los requerimientos funcionales ni en el modelo del mundo?

- No puede haber palabras repetidas
- Las traducciones son únicas

Invariante de clase

Problemas a resolver

- Falta de documentación de las características del modelo del mundo
- Duplicación de la información en los contratos de los métodos
- Poca claridad sobre quién tiene la responsabilidad de verificar si las propiedades definidas para una clase se cumplen

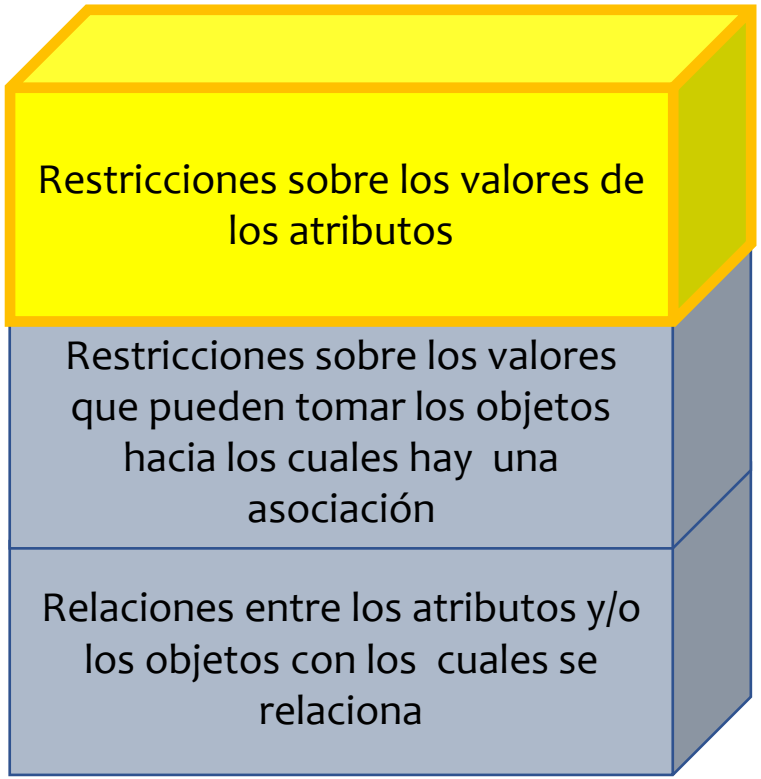
Invariante de clase

Invariante: Conjunto de afirmaciones (aserciones) que indican las propiedades que en todo momento deben cumplir las instancias de una clase.

Pueden usarse como suposiciones en todos los métodos y no aparecen en las precondiciones.

Invariante de clase

Invariante de clase

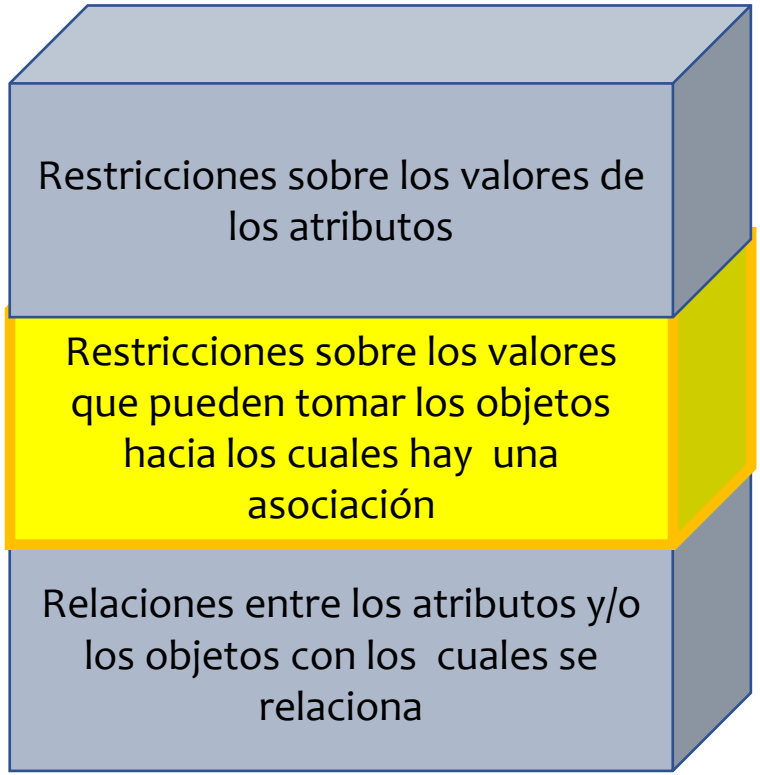


- palabra != null
- !palabra.equals(“”)
- traduccion != null
- !traduccion.equals(“”)

=

Una traducción es válida sólo si la palabra y la traducción no son nulas o vacías

Invariante de clase



- espanolIngles != null
- espanolFrances != null
- espanolItaliano != null

=

Los diccionarios deben estar inicializados

Invariante de clase

$$espanolIngles_i.palabra \; != \; espanolIngles_k.palabra, \; para\ todo\ i \; != \; k$$

=

En el vector espanolIngles no hay palabras repetidas

Invariante de clase

Todo método (que no sea un constructor) puede suponer al comienzo de su ejecución que se cumplen todas las afirmaciones que aparecen en el invariante de clase y en la precondición del contrato, y se compromete a que después de haber sido ejecutado sobre un objeto, éste cumple todas las afirmaciones del invariante y de la postcondición.

Invariante de clase

¿Cómo calcular el invariante?

1. Afirmaciones que vienen del análisis

Para que el programa funcione correctamente hay dos características que deben respetar los diccionarios. La primera es que no hay dos palabras en español repetidas en ningún diccionario (cada palabra tiene una sola traducción en cada idioma). La segunda es que no hay dos palabras en español con la misma traducción en ningún diccionario (cada palabra tiene una traducción que es única).

Invariante de clase

¿Cómo calcular el invariante?

- Los vectores que representan los diccionarios son distintos de null.
- En una traducción, tanto palabra como traducción no son cadenas vacías y son distintas de null.

2. Afirmaciones que vienen de decisiones de diseño

Invariante de clase

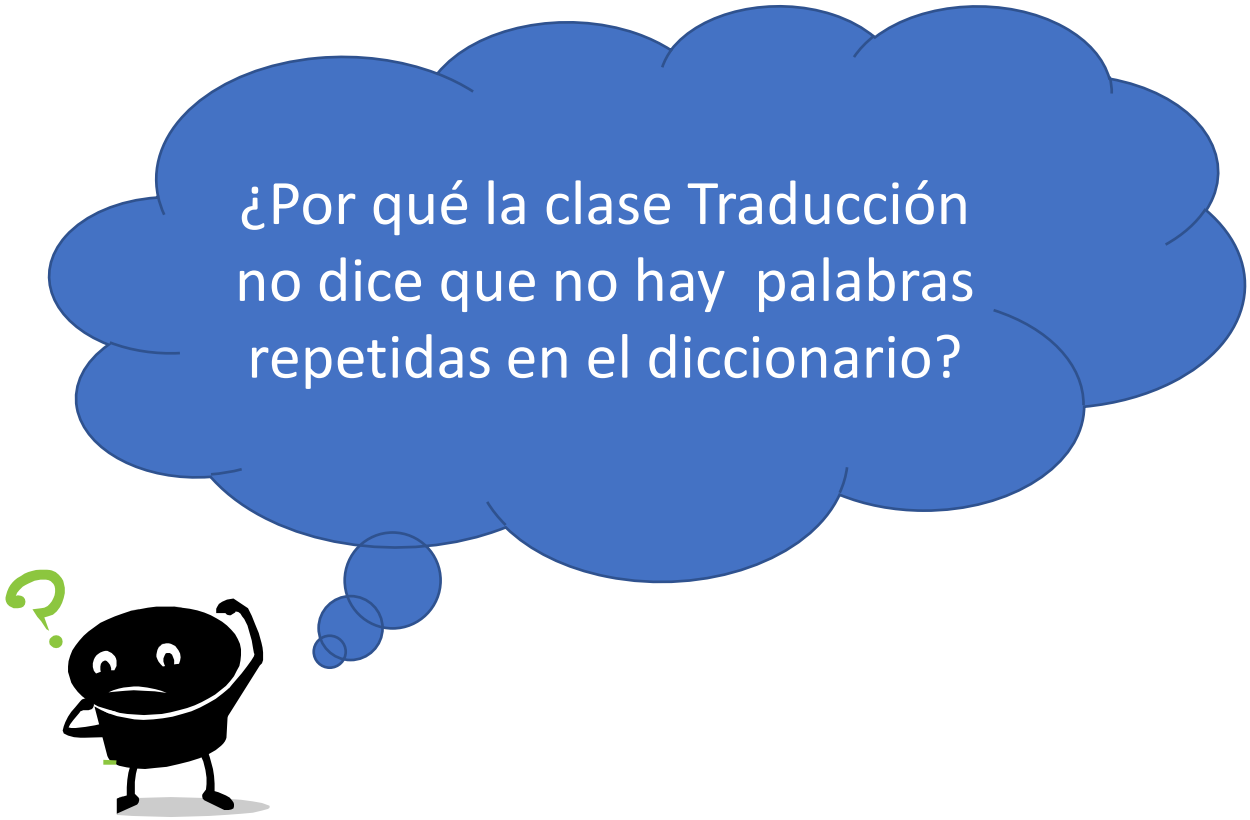
¿Cómo calcular el invariante?

1.Definir afirmaciones que vienen del análisis

2. Definir afirmaciones que vienen de decisiones de diseño

3. Decidir quién es el responsable de hacer las afirmaciones

Invariante de clase



Invariante de clase

¿Cómo se documenta?

```
/**
 * Descripción de la clase <br>
 * <b>inv:</b> <br>
 * ...
*/
public class MiClase
{

      ...
}
```

Las condiciones que componen el invariante, no se repiten en las precondiciones y las postcondiciones

Invariante de clase

```
/**
 * Traductor de palabras de español, inglés, francés e italiano. <br>
 * <b>inv: </b> <br>
 * espanolIngles != null <br>
 * espanolFrances != null <br>
 * espanolItaliano != null <br>
 *
 * En el vector espanolIngles no hay palabras repetidas <br>
 * En el vector espanolFrances no hay palabras repetidas <br>
 * En el vector espanolItaliano no hay palabras repetidas <br>
 *
 * En el vector espanolIngles no hay traducciones repetidas <br>
 * En el vector espanolFrances no hay traducciones repetidas <br>
 * En el vector espanolItaliano no hay traducciones repetidas
 */
public class Traduccion
{

    ...
}
```

Invariante de clase

```
/**
 * Representa una palabra y su traducción en otro idioma. <br>
 * <b>inv: </b> <br>
 * palabra != null <br>
 * !palabra.equals( "" ) <br>
 * traduccion != null <br>
 * !traduccion.equals( "" )
 */
public class Traduccion
{

    ...
}
```

Instrucción Assert

- Es una instrucción de java que verifica si algo se cumple o no.
- Se puede activar o desactivar
- Hay dos formas de usarla:

❑ assert expresion; ⟶ Lanza excepción de tipo AssertionError

❑ assert expresion1 : expresion2 ⟶ Lanza AssertionError con mensaje personalizado

Instrucción Assert

assert expresion;

Expresión booleana (verdadera o falsa)

Al evaluarla, si es verdadera se continúa la ejecución del programa. Si es falsa, se lanza una excepción de tipo AssertionError

Instrucción Assert

assert expresion1 : expresion2 ;

Expresión booleana (verdadera o falsa)

Al evaluarla, si es verdadera se continúa la ejecución del programa. Si es falsa, se lanza una excepción de tipo AssertionError , con el mensaje dado por la expresion2

Instrucción Assert

Expresiones simples

```
assert palabra != “ ” : “La palabra está vacía";
assert traduccion != null : “La traducción es inválida";
```

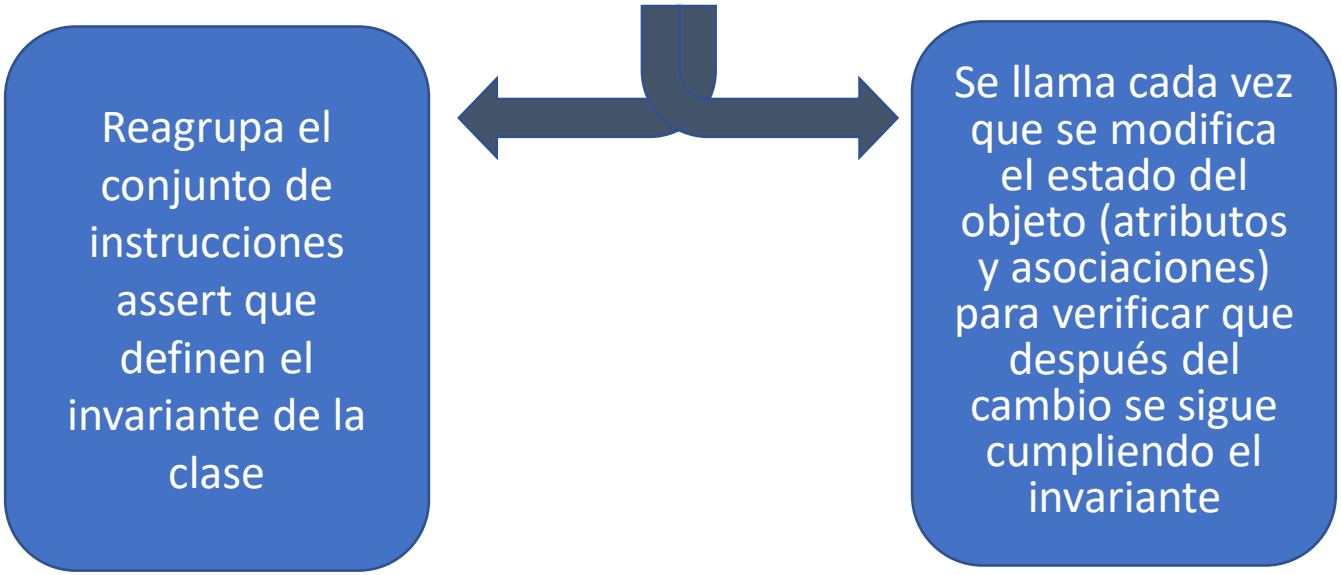
Expresiones compuestas / llamados a métodos privados

```
assert palabraEsValida ( ) : “La palabra es inválida";
assert traduccionEsValida ( ) : “La traducción es inválida";
```

Instrucción Assert

```
public class Traduccion
{
      …

     /** La palabra es válida si no está nula y si es diferente de la cadena vacía
      * @return true si la palabra es válida, false en caso contrario
      */
     private boolean palabraEsValida( )
     {
           return ( palabra != null && ! ( palabra.equals( "" ) ) );
      }
}
```

Método Verificar Invariante



Método Verificar Invariante

```
public class Traduccion
{
      ...

     /**
      * Verifica que el invariante de la clase se cumpla. Si algo falla, lanza un AssertError
      */
     private void verificarInvariante( )
     {
          assert palabraEsValida( ) : "La palabra es inválida";
          assert traduccionEsValida( ) : "La traducción es inválida";
      }
...
}
```

Método Verificar Invariante

```
public class Traduccion
{
      …

     /**
      * Asigna/cambia la traducción de la palabra
      * @param nuevaTraduccion - traducción de la palabra
      */
      public void asignarTraduccion( String nuevaTraduccion )
     {
           traduccion = nuevaTraduccion;
          // verifica el invariante
          verificarInvariante( );
     }
      …

}
```

Método Verificar Invariante

```
public class Traductor
{

    /**
     * Verifica que el invariante de la clase se cumpla. Si algo falla, lanza un AssertError
     */
    private void verificarInvariante( )
    {
        assert espanolIngles != null : "Diccionario español-ingles sin inicializar";
        assert espanolFrances != null : "Diccionario español-francés sin inicializar";
        assert espanolItaliano != null : "Diccionario español-italiano sin inicializar";

        assert !hayPalabrasRepetidas( INGLES ) : "Palabras repetidas en el diccionario de inglés";
        assert !hayPalabrasRepetidas( FRANCES ) : "Palabras repetidas en el diccionario de  francés";
        assert !hayPalabrasRepetidas( ITALIANO ) : "Palabras repetidas en el diccionario de italiano";

       assert !hayTraduccionesRepetidas(INGLES) : "Traducciones repetidas en el diccionario de inglés";
      assert !hayTraduccionesRepetidas(FRANCES) : "Traducciones repetidas en el diccionario de
      francés";
     assert !hayTraduccionesRepetidas(ITALIANO) : "Traducciones repetidas en el diccionario de
     italiano;
    }
...
}
```

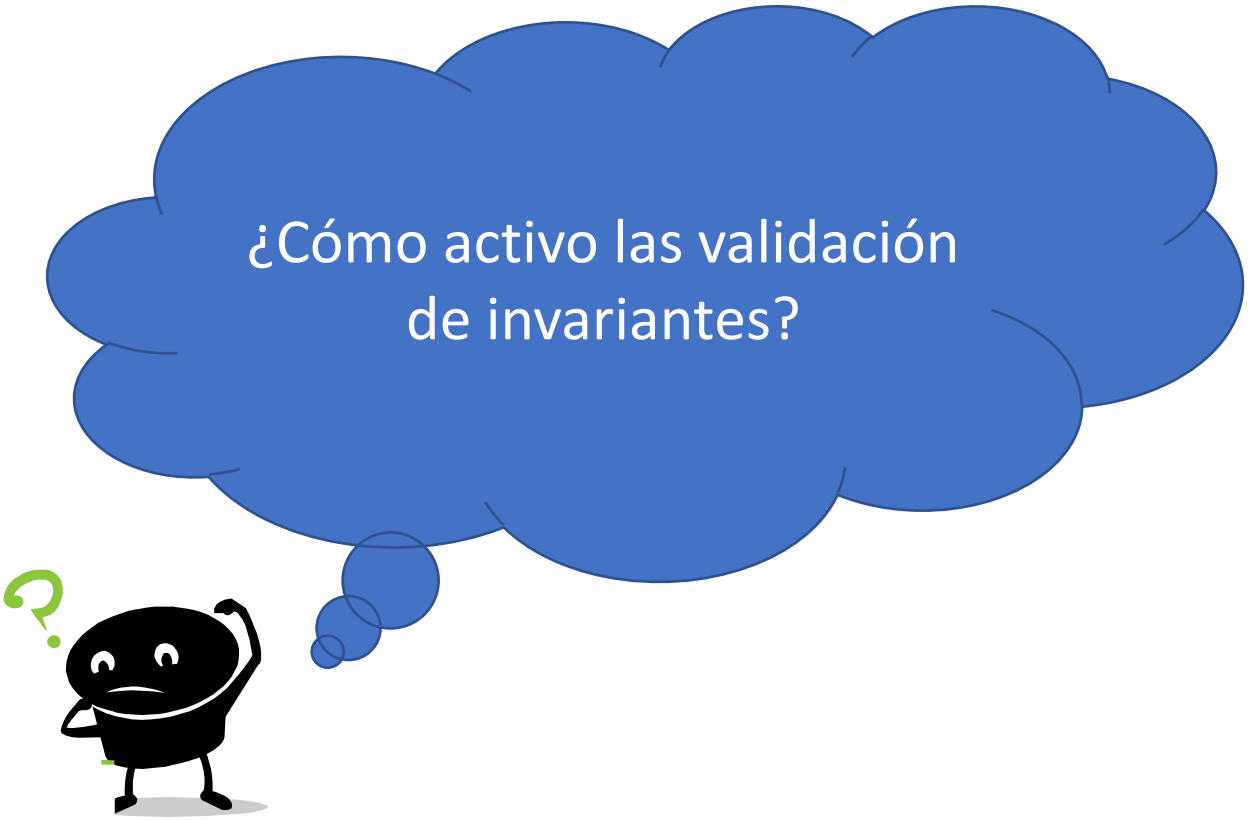
Ejercicio

```
/**
   * Indica si hay palabras con la misma traducción en el diccionario del idioma
     dado. <br>
   * @param idTrad es el idioma del diccionario - idTrad pertenece a {FRANCES,
                  INGLES, ITALIANO}
   * @return True si hay dos palabras con la misma traducción o false en caso
                  contrario.
   */
    private boolean hayPalabrasRepetidas( int idTrad )
   {









    }
```

Ejercicio

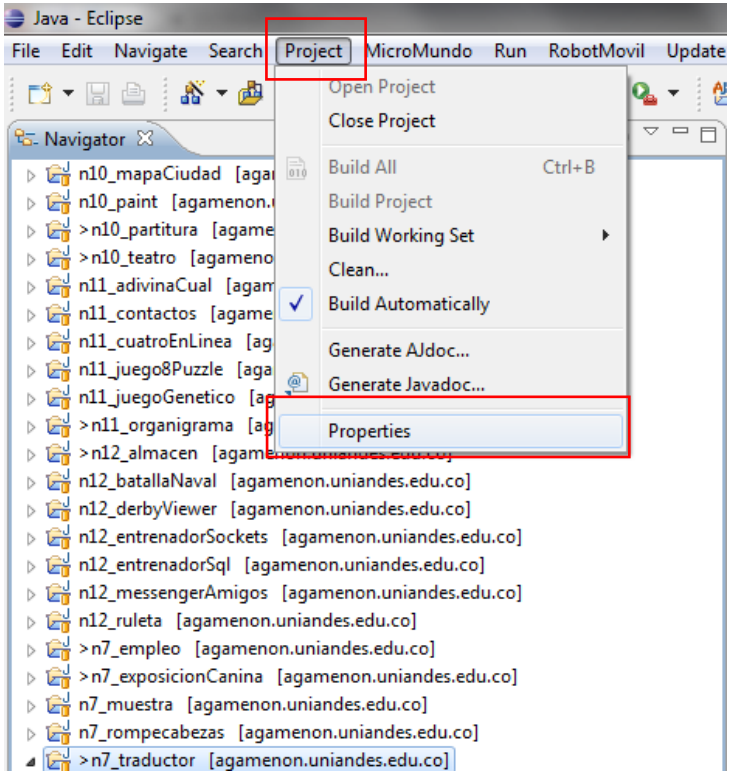
```
/**
   * Indica si hay palabras repetidas en el diccionario del idioma dado. <br>
   * @param idTrad es el idioma del diccionario - idTrad pertenece a { FRANCES,
                  INGLES, ITALIANO }
   * @return True si hay al menos una palabra repetida o false en caso
                  contrario.
   */
    private boolean hayPalabrasRepetidas( int idTrad )
   {









    }
```

Configuración Eclipse



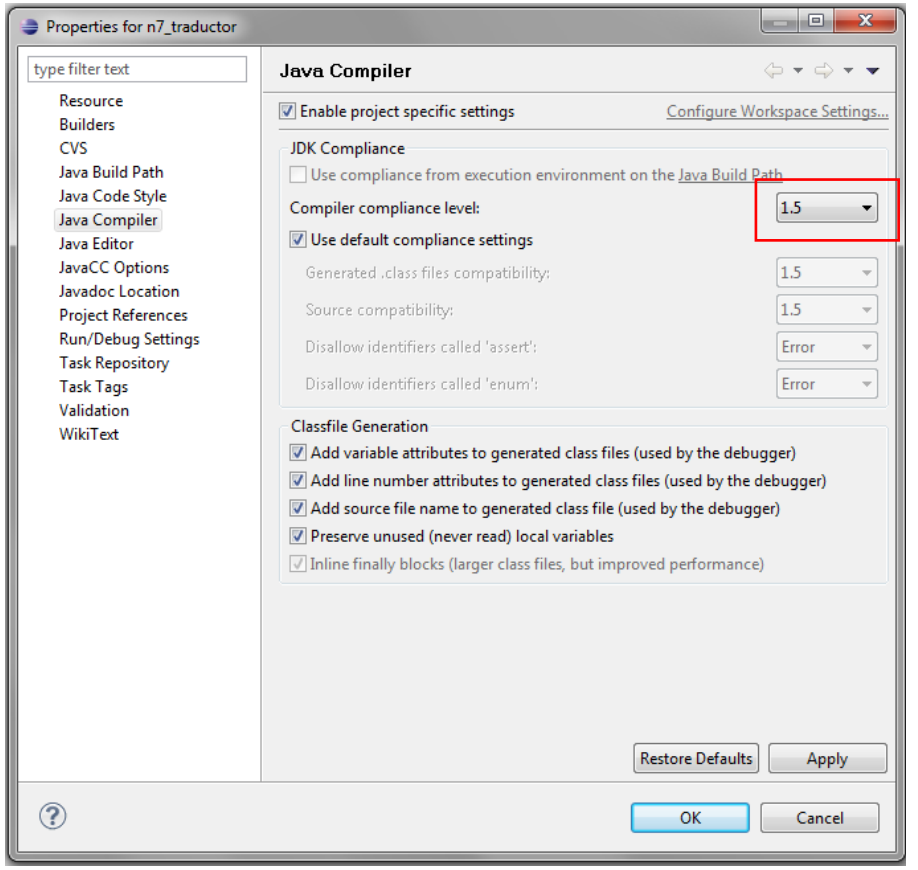
Configuración Eclipse

1 Entrar a las propiedades del proyecto

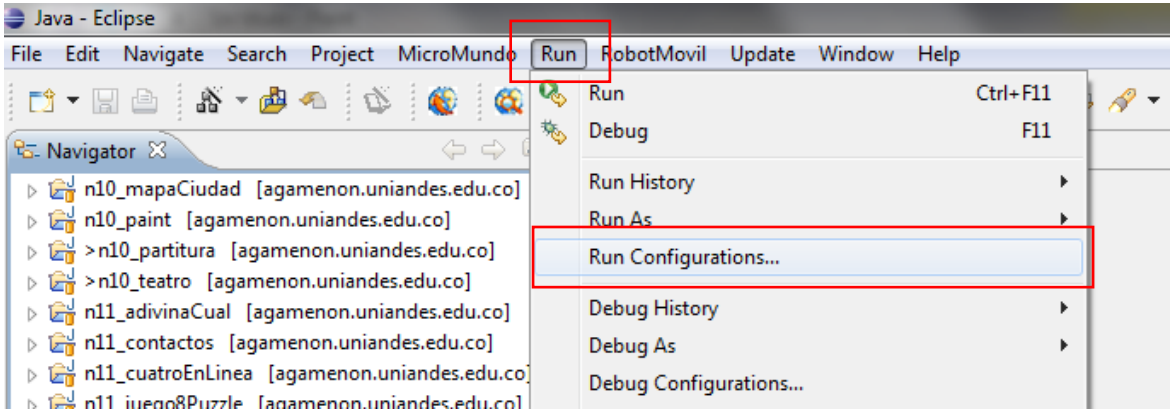


Configuración Eclipse

2 Cambiar el nivel del compilador a 1.5 o un nivel mayor



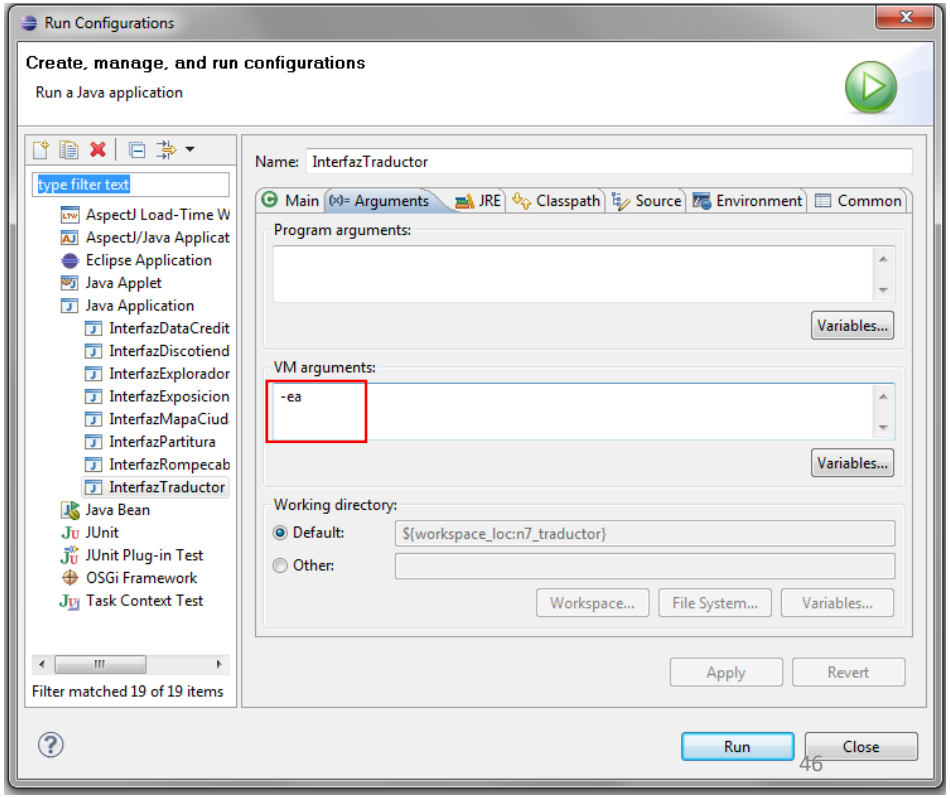
Configuración Eclipse



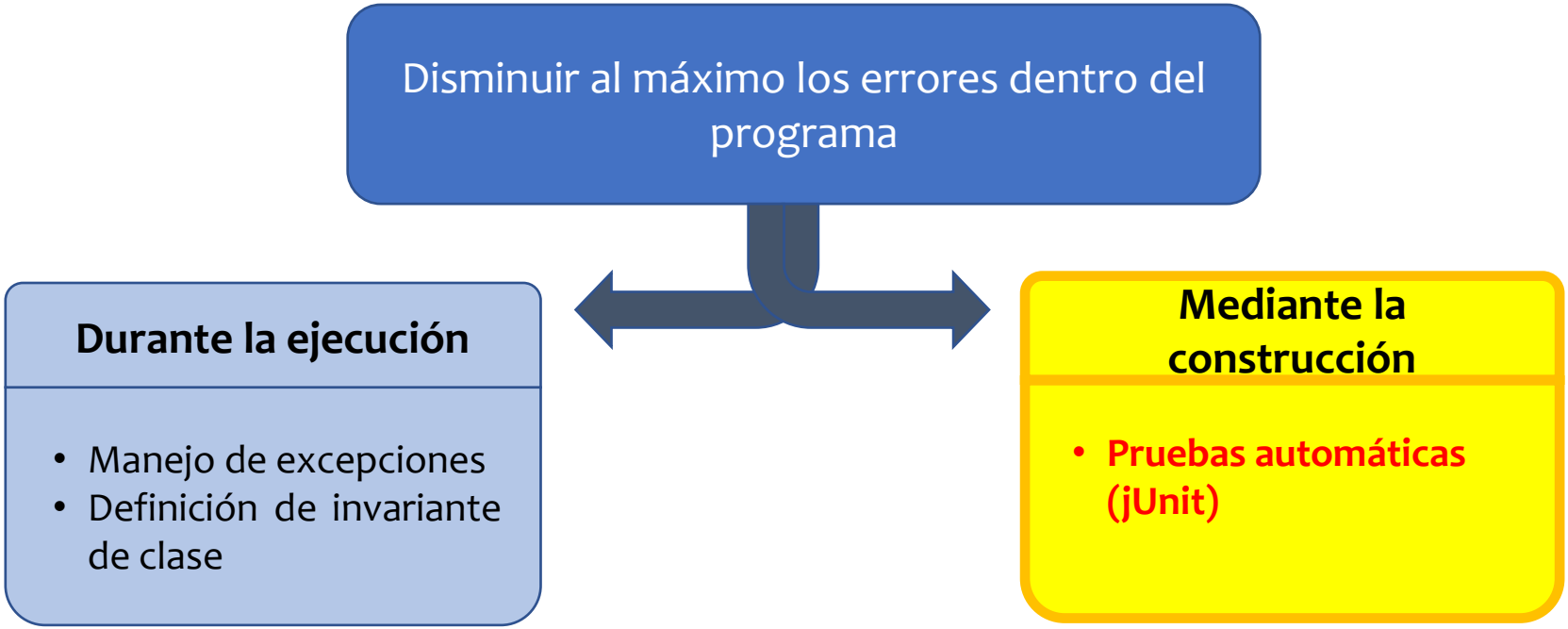
3 Entrar a las configuraciones de ejecución

Configuración Eclipse

4 Cambiar el argumento de la máquina virtual a -ea



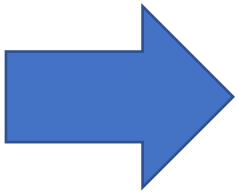
Verificación y Pruebas de programa



Pruebas unitarias automáticas

Pruebas unitarias automáticas

Desarrollo Completo



- Pruebas completas sobre interfaz de usuario
- Basadas en guiones de prueba para verificación de RF

Pruebas unitarias automáticas

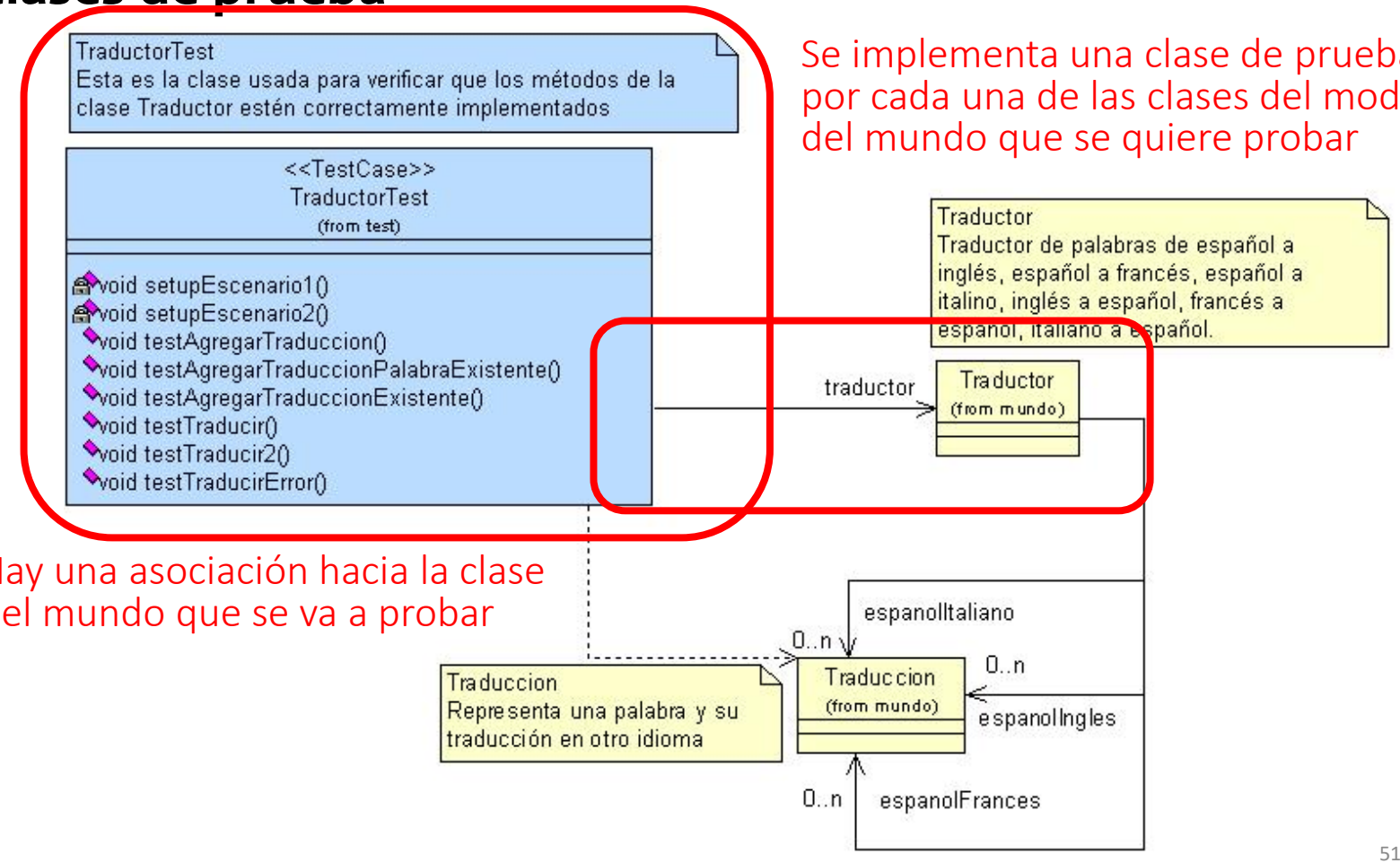
- No se prueba el programa completo
- Se prueba cada clase individualmente

Clase de prueba

- Si algo falla, se sabe en qué clase fue y en qué método específicamente

- Trabaja sobre un conjunto predefinido de escenarios y casos

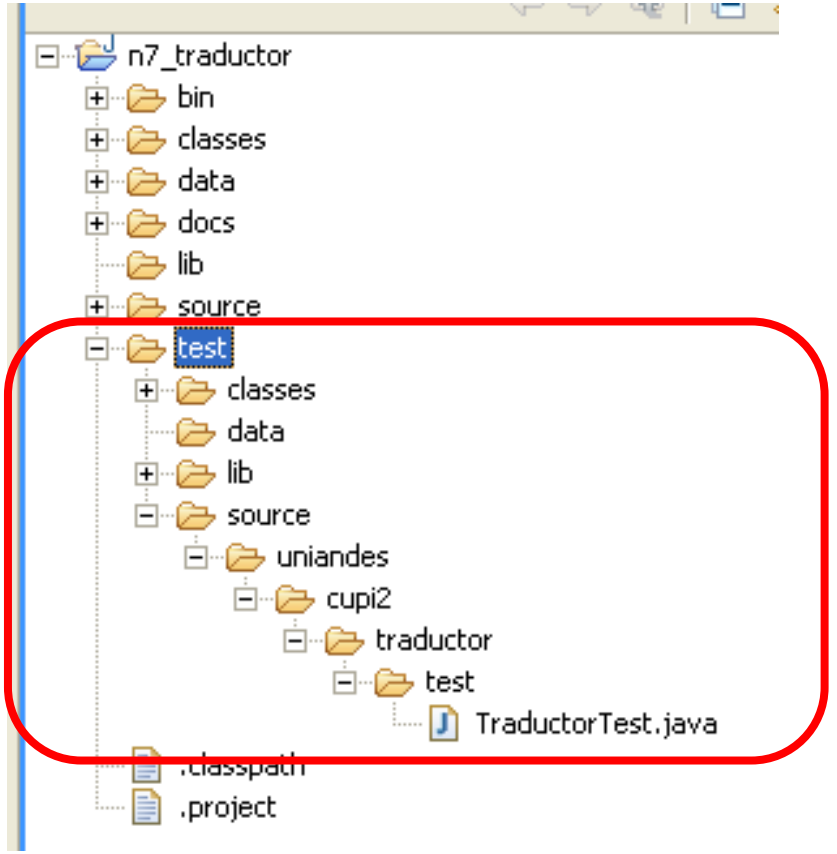
Clases de prueba



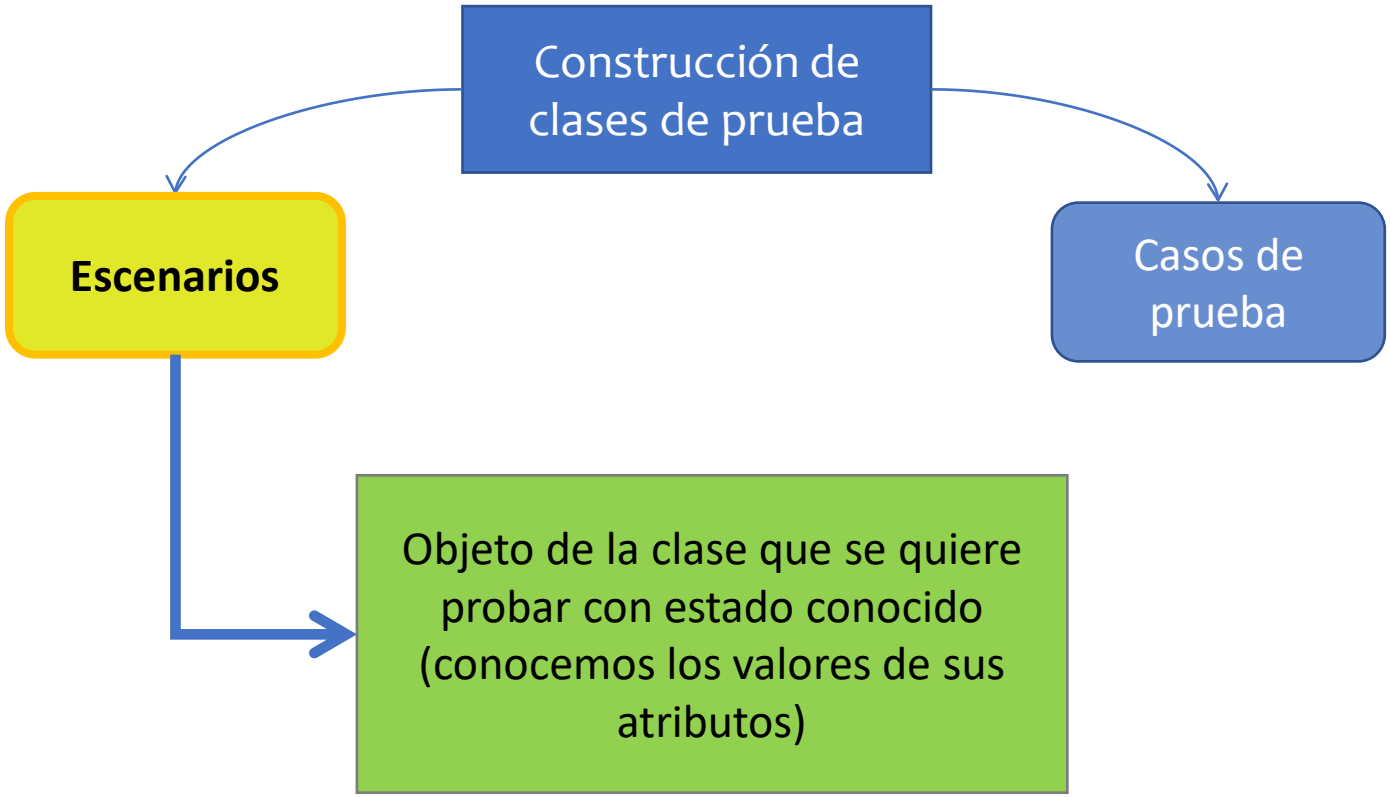
Clases de prueba

Las clases de prueba se almacenan:

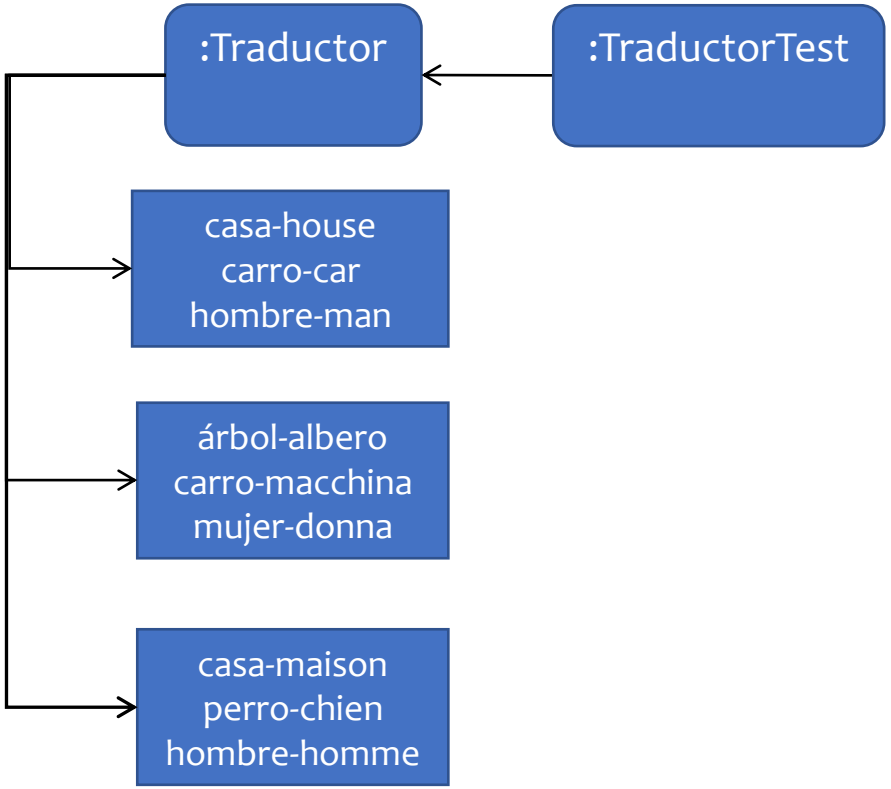
- En un paquete distinto (test)
- En un directorio aparte



Clases de prueba



Clases de prueba (Escenarios)



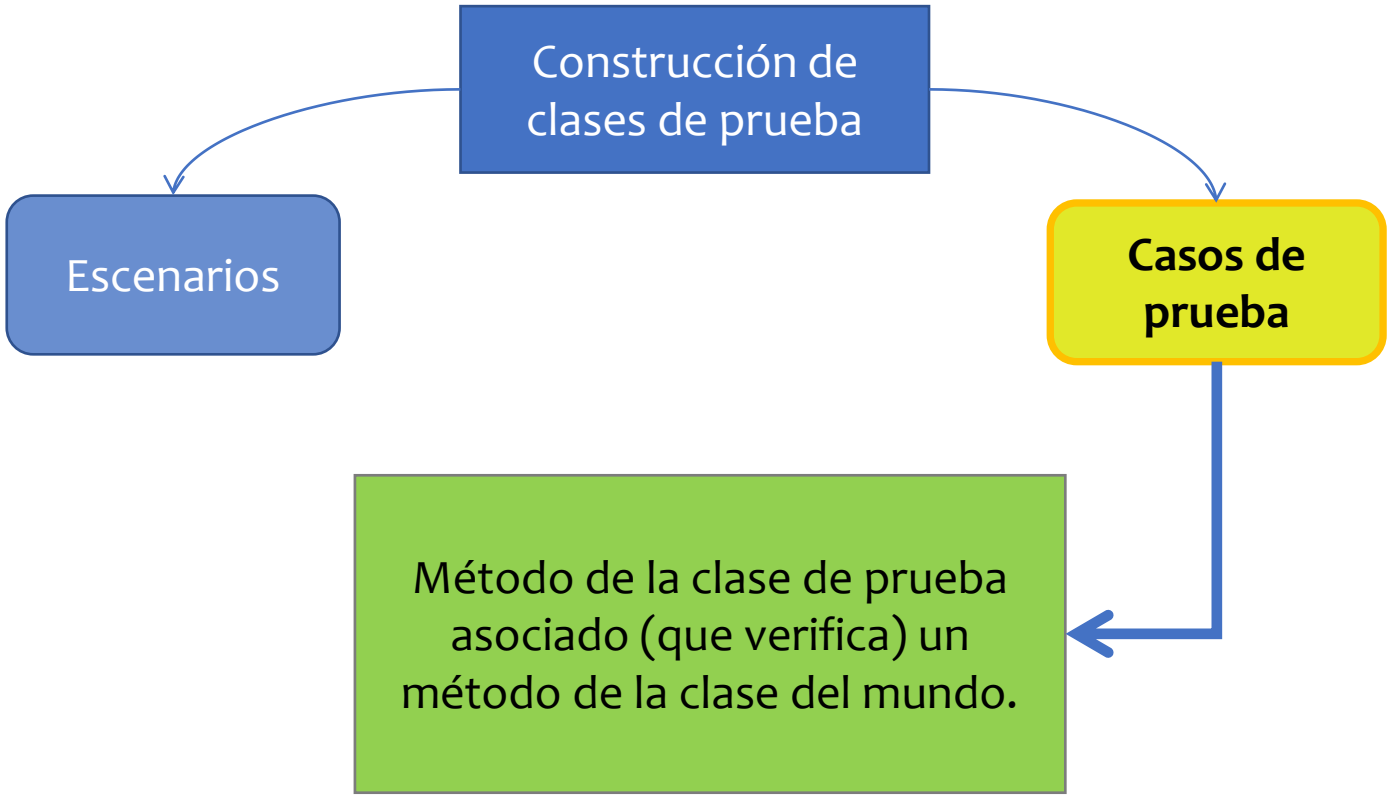
Clases de prueba (Escenarios)

español	inglés
perro	dog
lápiz	pencil
amor	love
ratón	mouse

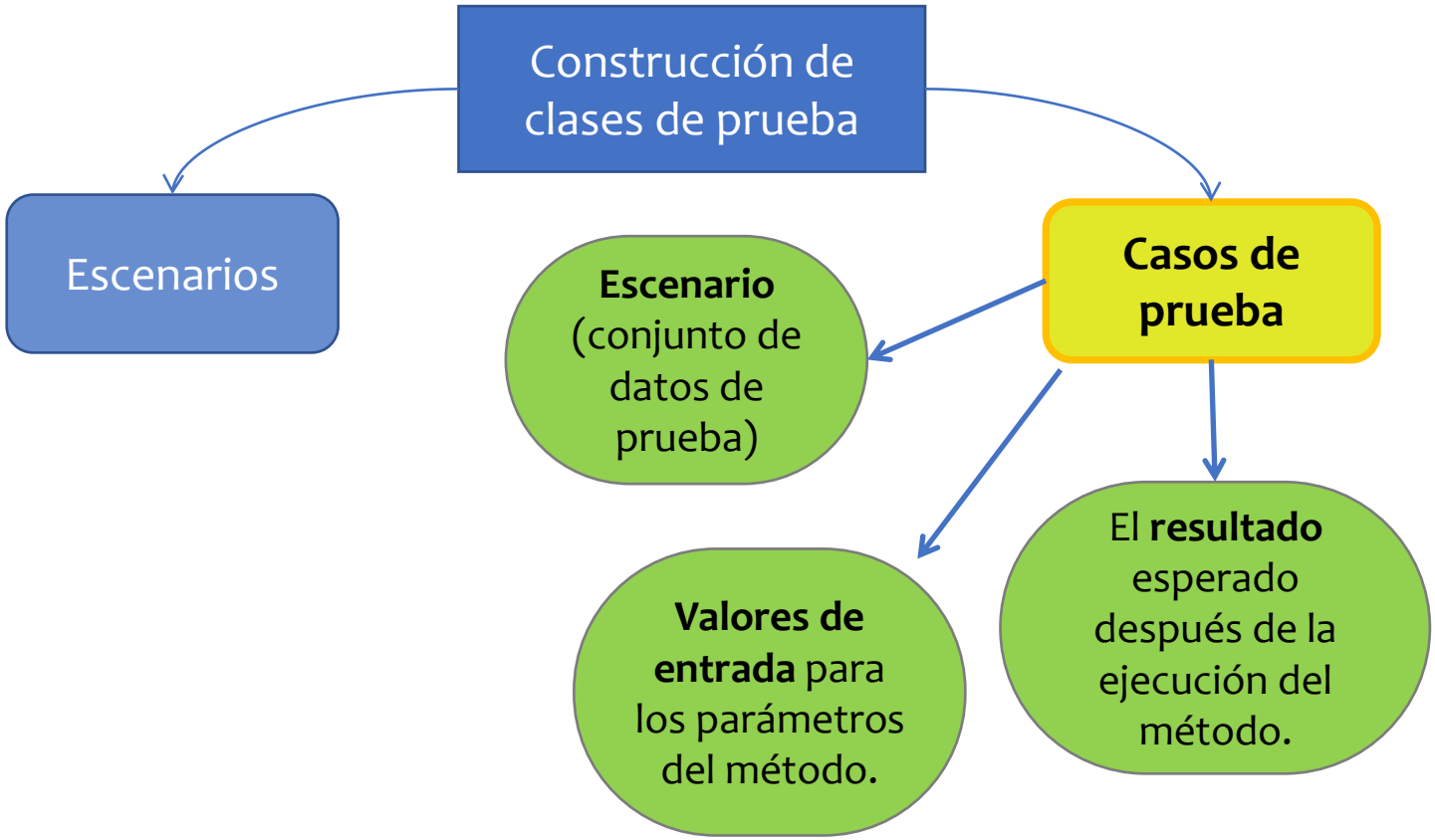
español	francés
azul	bleu
libro	livre
lápiz	crayon
teléfono	téléphone
mesa	table

español	italiano
mesa	tavolo
lápiz	rossetto

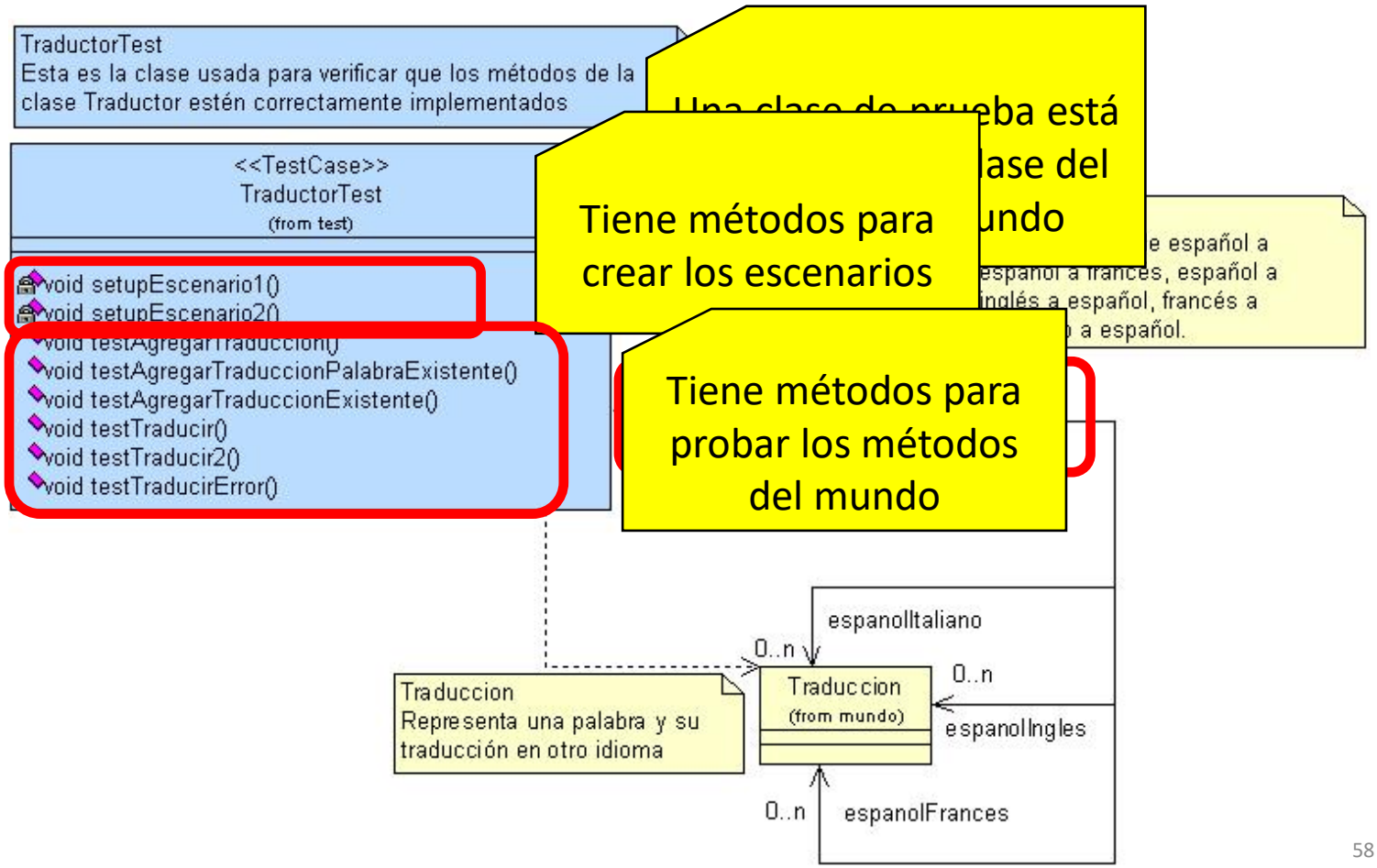
Clases de prueba



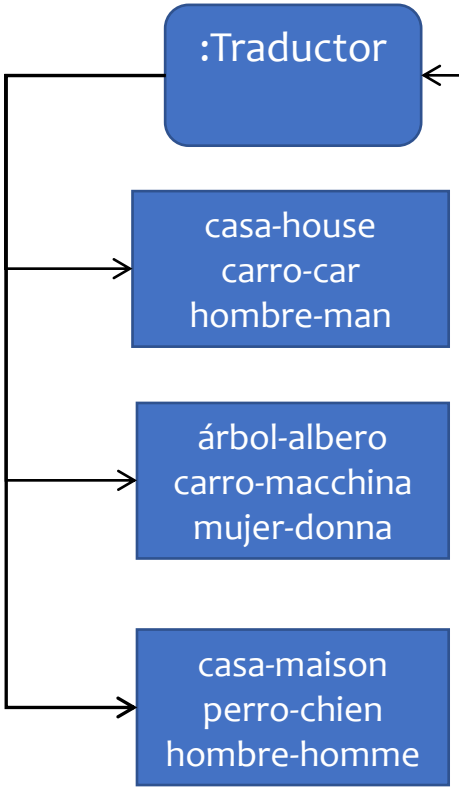
Clases de prueba



Clases de prueba (Casos de prueba)



Clases de prueba (Casos de prueba)



Método	Valores de entrada	Resultado
agregarTraduccion()	pal=azul trad=bleu idDestino = FRANCES	Verdadero
agregarTraduccion()	pal=casa trad=maison idDestino = FRANCES	Falso
agregarTraduccion()	pal=coche trad=car idDestino = INGLES	Falso

Ejercicio

(Casos de prueba)

Definir las pruebas para el método traducir( ) de la clase Traductor

Prueba 1

Objetivo: Probar que el método es capaz de encontrar correctamente la traducción de una palabra del español a cualquiera de los otros idiomas.

Prueba 2

Objetivo: Probar que el método es capaz de encontrar correctamente la traducción de una palabra de un idioma distinto a español a cualquiera de los otros idiomas.

Prueba 3

Objetivo: Probar que el método no encuentra la traducción de palabras que no están en el diccionario.

Ejercicio

(Casos de prueba)

```
/**
 Dada una palabra, el idioma en el que está y el idioma al que se quiere traducir, este
 método retorna  la traducción correspondiente. Para que la traducción exista entre
 dos idiomas diferentes es necesario que la traducción de la palabra exista en los dos
 idiomas implicados. Así por ejemplo para traducir una palabra de francés a inglés es
 necesario que exista la traducción en el diccionario de español-francés y que la
 traducción de la palabra en español exista el diccionario español-inglés. <br>
 * @param pal es la palabra - pal != null
 * @param idOrigen es el idioma de origen - idOrigen pertenece a {FRANCES, INGLES,
ITALIANO, ESPANOL}
 * @param idDestino es el idioma destino - idDestino pertenece a {FRANCES, INGLES,
ITALIANO, ESPANOL}
 * @return Traducción de la palabra en el idioma destino. Si no existe, retorna null.
 */
 public Traduccion traducir(String pal, int  idOrigen, int idDestino )
```

Ejercicio

(Casos de prueba)

español	inglés
perro	dog
lápiz	pencil
amor	love
ratón	mouse
español	francés
azul	bleu
libro	livre
lápiz	crayon
teléfono	téléphone
mesa	table
español	italiano
mesa	tavlo
lápiz	rossetto

Prueba 1

Objetivo: Probar que el método es capaz de encontrar correctamente la traducción de una palabra del español a cualquiera de los otros idiomas.

Valores de entrada	Resultado

Ejercicio

(Casos de prueba)

español	inglés
perro	dog
lápiz	pencil
amor	love
ratón	mouse
español	francés
azul	bleu
libro	livre
lápiz	crayon
teléfono	téléphone
mesa	table
español	italiano
mesa	tavlo
lápiz	rossetto

Prueba 2

Objetivo: Probar que el método es capaz de encontrar correctamente la traducción de una palabra de un idioma distinto a español a cualquiera de los otros idiomas.

Valores de entrada	Resultado

Ejercicio

(Casos de prueba)

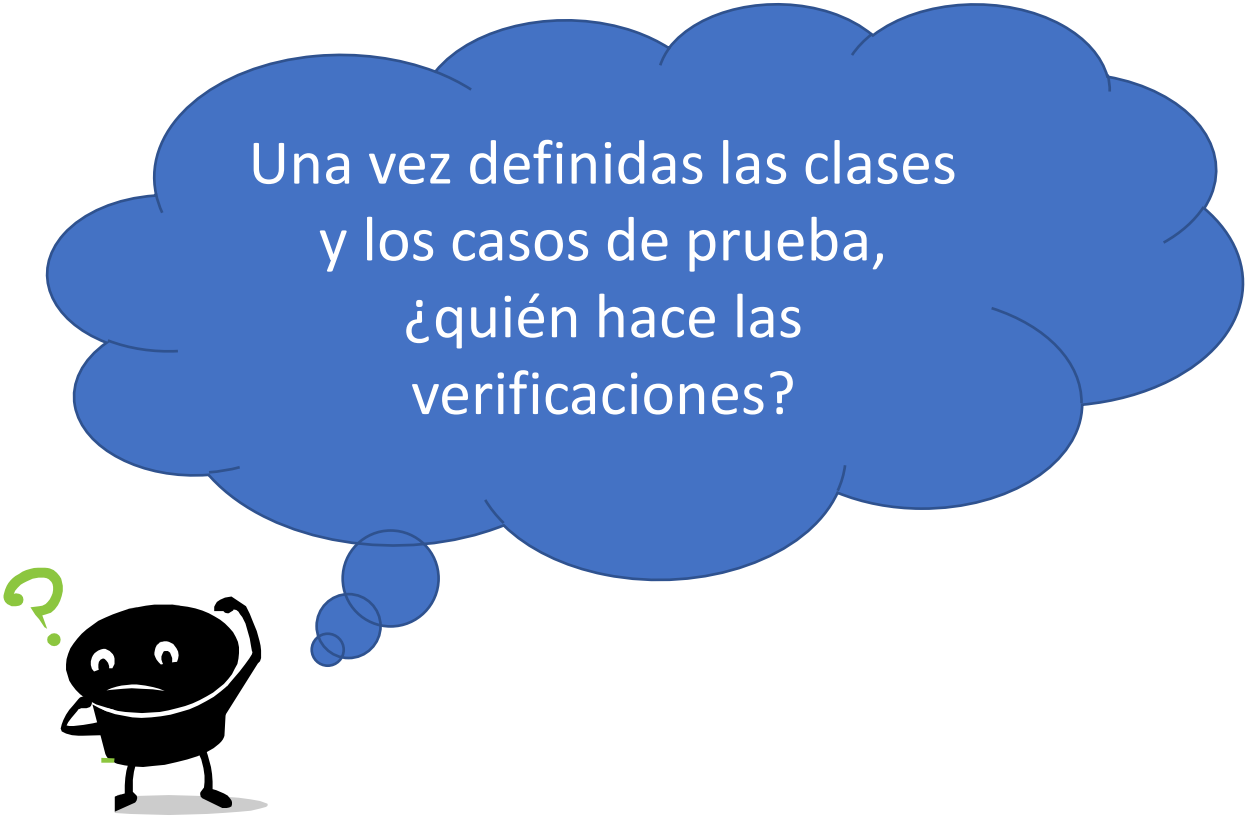
español	inglés
perro	dog
lápiz	pencil
amor	love
ratón	mouse
español	francés
azul	bleu
libro	livre
lápiz	crayon
teléfono	téléphone
mesa	table
español	italiano
mesa	tavlo
lápiz	rossetto

Prueba 3

Objetivo: Probar que el método no encuentra la traducción de palabras que no están en el diccionario.

Valores de entrada	Resultado

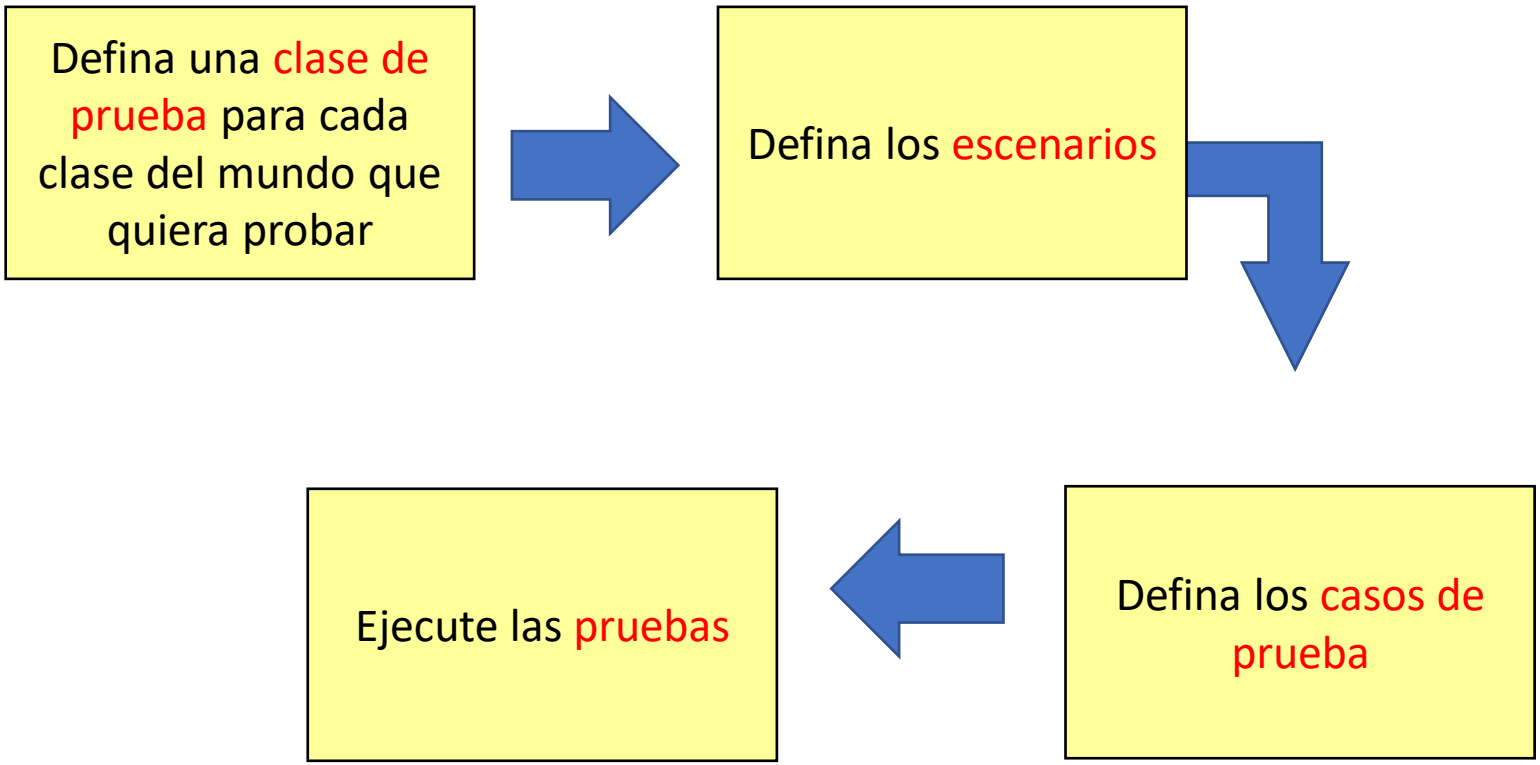
Pruebas unitarias automáticas



JUnit

- JUnit es una herramienta que permite probar las clases de las aplicaciones construidas en Java.
- Consta de:
  - Clases básicas de java que se utilizan para crear las clases de prueba propias de la aplicación
  - Un ejecutor de pruebas: ejecuta la pruebas y presenta sus resultados de forma gráfica o textual.
- Sitio http://junit.org

Junit (Procedimiento General)



Junit (Procedimiento General)

1. Defina una clase de prueba para cada clase del mundo que quiera probar

```
package uniandes.cupi2.traductor.test;

import junit.framework.TestCase;

/**
 * Esta es la clase usada para verificar que los métodos de la clase Traductor estén
 * correctamente implementados
 */
public class TraductorTest extends TestCase
{
    // -----------------------------------------------------------------
    // Atributos
    // -----------------------------------------------------------------

    /**
     * Es la clase donde se harán las pruebas
     */
    private Traductor traductor;

    // -----------------------------------------------------------------
    // Métodos
    // -----------------------------------------------------------------
```

Se deben importar las clases del paquete junit.framework

Debe declararse con un extends TestCase en el encabezado

Nombre la clase de la siguiente manera: <Nombre clase a probar>Test

Hay una asociación (atributo) de la clase del mundo que va a probar

Junit (Procedimiento General)

2. Defina los escenarios

```
/**
 * Crea al traductor con 12 traducciones de cada idioma
 */
private void setupEscenario1( )
{
    traductor = new Traductor( );

    // Agrega 12 palabras con traducción en inglés
    traductor.agregarTraduccion( "perro", "dog", Traductor.INGLES );
    traductor.agregarTraduccion( "blanco", "white", Traductor.INGLES );
    traductor.agregarTraduccion( "casa", "house", Traductor.INGLES );
    traductor.agregarTraduccion( "cielo", "sky", Traductor.INGLES );
    traductor.agregarTraduccion( "amor", "love", Traductor.INGLES );
    traductor.agregarTraduccion( "anillo", "ring", Traductor.INGLES );
    traductor.agregarTraduccion( "torre", "tower", Traductor.INGLES );
    traductor.agregarTraduccion( "león", "lion", Traductor.INGLES );
    traductor.agregarTraduccion( "basura", "garbage", Traductor.INGLES );
    traductor.agregarTraduccion( "bello", "beautiful", Traductor.INGLES );
    traductor.agregarTraduccion( "ratón", "mouse", Traductor.INGLES );
    traductor.agregarTraduccion( "lápiz", "pencil", Traductor.INGLES );

    // Agrega 12 palabras con traducción en francés
    traductor.agregarTraduccion( "casa", "maison", Traductor.FRANCES );
```

Comienzan con el prefijo "setupEscenario" y un número

Invocan los métodos necesarios para llevarlo al estado que se definió en el diseño

Son métodos privados

Se encargan de crear una instancia del modelo del mundo

Junit (Procedimiento General)

3. Defina los casos de prueba

- Defina los casos a partir de los métodos relevantes de la clase a probar y sus contratos, validando que para las diferentes situaciones iniciales que pueden tenerse, los resultados sean los esperados.
- Un caso de prueba es un método cuyo nombre comienza con test:

```
public void test<nombre caso de prueba>
```

- Utilice el escenario indicado para preparar los datos de la prueba.
- Invoque los métodos necesarios para probar la funcionalidad.
- Defina las aserciones o condiciones que deben ser válidas al ejecutar la funcionalidad que está probando

Junit (Procedimiento General)

- Aserción
  assertTrue(mensaje, condicion): evalúa la condicion. Si es verdadera, continúa la ejecución. Si es falsa, lanza la excepción AssertionFailedError con el mensaje mensaje.
  assertFalse(mensaje, condicion): lanza excepción si la condición NO es falsa.
  assertNull(mensaje, objeto): lanza excepción si el objeto NO es null.
  assertNotNull:(mensaje, objeto): lanza excepción si el objeto es null.
  assertEquals(mensaje, esperado, actual): lanza excepción si esperado != actual (de tipo int, long o char).
  assertSame: valida que dos referencias sean iguales
  assertNotSame: valida que dos referencias sean distintas

- No confundir con **assert** de Java

- Falla
  Fail(mensaje): Hace fallar una prueba. Este método SIEMPRE lanza una excepción. Sólo se debe situar en puntos de las pruebas en donde es un error que el programa llegue.

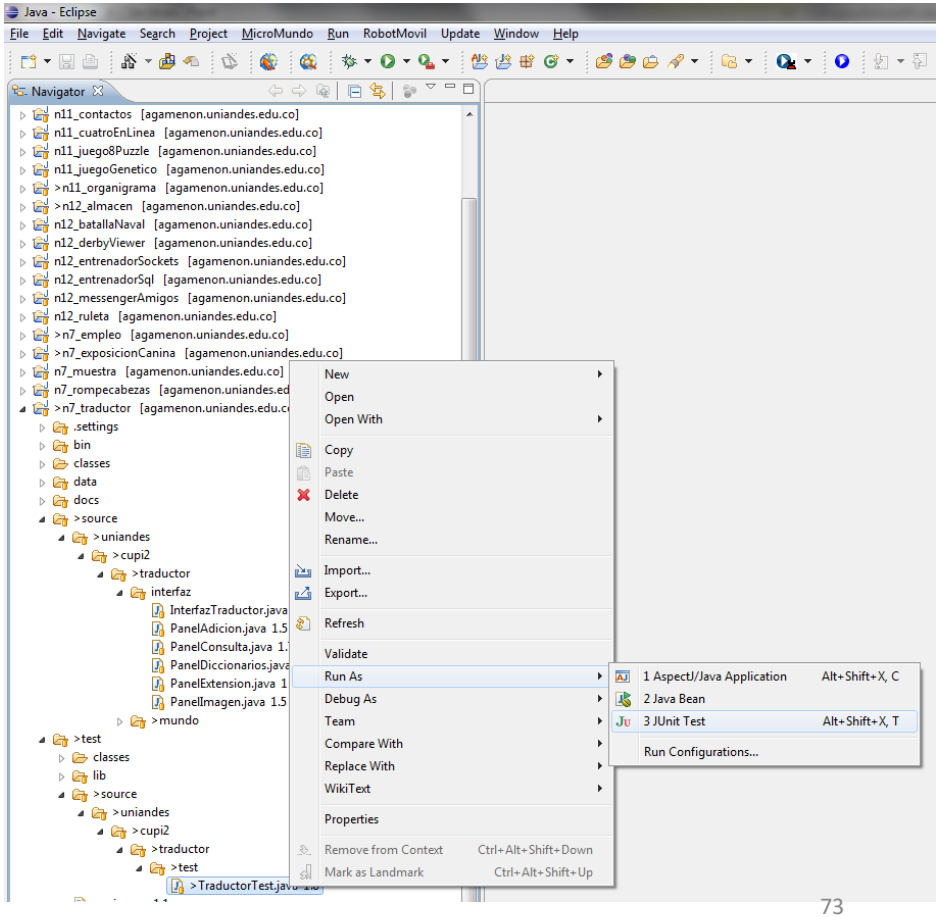
Junit (Procedimiento General)

3. Defina los casos de prueba

```
public void testTraducir1( )
{
    // Configura los datos de prueba
    setupEscenario1( );

    // Realizar traducciones de español a inglés
    Traduccion traduccion = traductor.traducir( "casa", Traductor.ESPANOL, Traductor.INGLES );
    assertEquals( "La palabra original está mal", "casa", traduccion.darPalabra( ) );
    assertEquals( "La palabra traducida está mal", "house", traduccion.darTraduccion( ) );

    traduccion = traductor.traducir( "anillo", Traductor.ESPANOL, Traductor.INGLES );
    assertEquals( "La palabra original está mal", "anillo", traduccion.darPalabra( ) );
    assertEquals( "La palabra traducida está mal", "ring", traduccion.darTraduccion( ) );
```

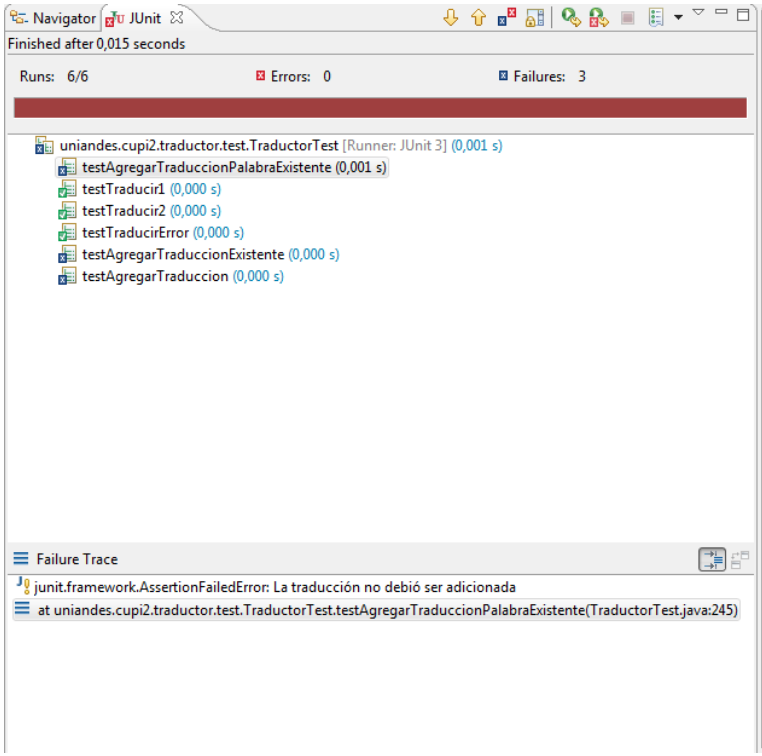
Junit (Procedimiento General)

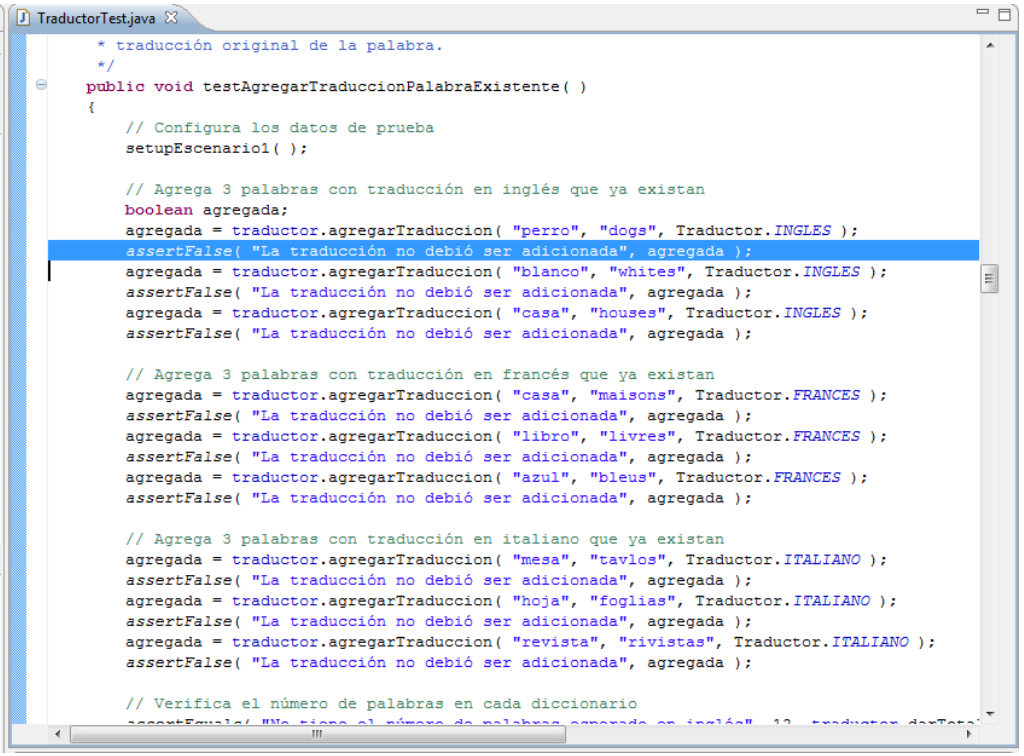
3. Ejecute los casos de prueba



Junit (Procedimiento General)

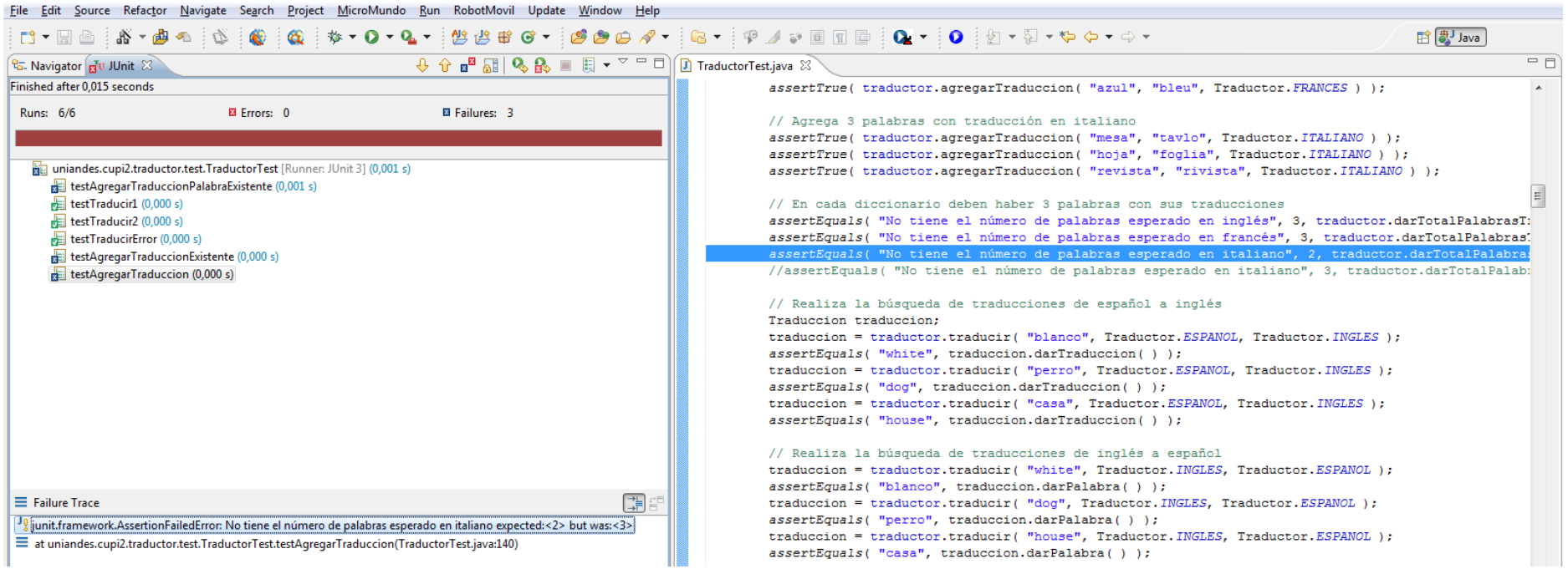
3. Ejecute los casos de prueba





Junit (Procedimiento General)

3. Ejecute los casos de prueba



Caso de Estudio: Notas del curso

```
public class Curso
{
        //Constantes
         public final static int NUM_NOTAS = 12;

        //Atributos
        private double notas[ ];
}
```

notas =

0	1	2	3	4	5	6	7	8	9	10	11
3.5	5.0	4.5	2.5	4.5	2.5	3.5	4.0	2.0	1.5	3.5	5.0

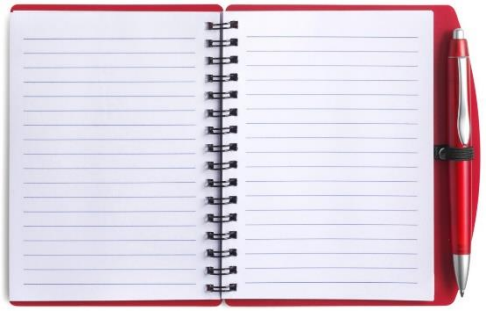
Algoritmos de Búsqueda

Algoritmos de Búsqueda

Se pueden presentar dos casos

Arreglo ordenado

Arreglo no ordenado



Búsqueda Secuencial en Arreglo no Ordenado

Problema: Buscar un elemento X en un arreglo desordenado.

Solución: Hacer una búsqueda secuencial.

Arreglo no ordenado

	0	1	2	3	4	5	6	7	8	9	10	11
notas =	3.5	5.0	4.5	2.5	4.5	2.5	3.5	4.0	2.0	1.5	3.5	5.0

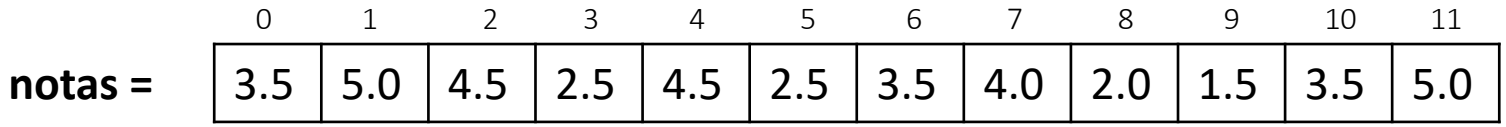
Búsqueda Secuencial en Arreglo no Ordenado

Comenzar a partir de la posición 0 del arreglo e ir recorriendo las diversas casillas hasta:

- Encontrar el elemento

  o

- Llegar al final del arreglo.



Búsqueda Secuencial en Arreglo no Ordenado

Tarea No 13. (Pag. 52): Buscar un elemento en un arreglo desordenado

```
public boolean buscarSecuencial( double valor )
{



}
```

notas =

0	1	2	3	4	5	6	7	8	9	10	11
3.5	5.0	4.5	2.5	4.5	2.5	3.5	4.0	2.0	1.5	3.5	5.0

Algoritmos de Búsqueda

Se pueden presentar dos casos

Arreglo ordenado

Arreglo no ordenado

Búsqueda en Arreglo Ordenado

Arreglo ordenado

Problema: Buscar un elemento X en un arreglo ordenado (ascendentemente o descendentemente).

Solución:

- Búsqueda secuencial.
- Búsqueda binaria

	0	1	2	3	4	5	6	7	8	9	10	11
notas =	1.2	1.3	2.0	2.1	2.5	2.7	3.0	3.3	3.4	4.0	4.2	4.4

Búsqueda Secuencial en Arreglo Ordenado

- Comenzar a partir de la posición 0 del arreglo e ir recorriendo las diversas casillas hasta:
  - Encontrar el elemento, o
  - Estar seguro de que no está.

- ¿Cómo saber que no está?
  - Si orden ascendente: Cuando encuentra un elemento superior
  - Si orden descendente: Cuando encuentra un elemento inferior
  - Cuando llega al final del arreglo

	0	1	2	3	4	5	6	7	8	9	10	11
notas =	1.2	1.3	2.0	2.1	2.5	2.7	3.0	3.3	3.4	4.0	4.2	4.4

Búsqueda Secuencial en Arreglo Ordenado

Ejemplos:

- Buscar la posición de la nota 2.3
- Buscar la posición de la nota 3.3
- Buscar la posición del la nota 4.6

notas =

0	1	2	3	4	5	6	7	8	9	10	11
1.2	1.3	2.0	2.1	2.5	2.7	3.0	3.3	3.4	4.0	4.2	4.4

Búsqueda Secuencial en Arreglo Ordenado

Tarea No 13. (Pag. 52): Buscar un elemento en un arreglo ordenado ascendentemente.

```
public boolean buscarSecuencial( double valor )
{

}
```

notas =

0	1	2	3	4	5	6	7	8	9	10	11
1.2	1.3	2.0	2.1	2.5	2.7	3.0	3.3	3.4	4.0	4.2	4.4

Búsqueda Binaria en Arreglo Ordenado

- La búsqueda binaria sigue el mismo principio
- Se mira el elemento de la mitad
- Si ese es el elemento que se está buscando, se termina y se retorna esa posición
- Si ese elemento es mayor que el que se está buscando
  - Necesariamente se encuentra en la primera mitad, por lo que la parte del arreglo que nos interesa se encuentra entre la posición 0 y la mitad-1
- Si ese elemento es menor que el que se está buscando
  - Necesariamente se encuentra en la segunda mitad, por lo que la parte del arreglo que nos interesa se encuentra entre la posición mitad+1 hasta el último elemento del arreglo
- Se repite la división por mitades en el nuevo rango de búsqueda

Búsqueda Binaria en Arreglo Ordenado

```
public boolean buscarBinario( double nota )
 {
    int inicio = 0;
    int fin = notas.length - 1;
    boolean encontre = false;
    while( inicio <= fin && !encontre )
    {
       int medio = ( inicio + fin ) / 2;
       if( notas[ medio ] == nota )
          encontre = true;
       else if( notas[ medio ] > nota )
          fin = medio - 1;
       else
          inicio = medio + 1;
    }
    return encontre;
 }
```

Se definen los índices inicio y fin. Como se va dividiendo el rango de búsqueda por mitades, se termina cuando se cruzan los índices

Se mira el elemento que se encuentra en la mitad del rango. Si es el que se está buscando, termina

Si es mayor que el que se está buscando, se limita el rango a la primera mitad

Si es menor que el que se está buscando, se limita el rango a la segunda mitad

Búsqueda Binaria en Arreglo Ordenado

Ejemplos:

1) Buscar la nota 2.1
2) Buscar la nota 4.4

notas =

0	1	2	3	4	5	6	7	8	9	10	11
1.2	1.3	2.0	2.1	2.5	2.7	3.0	3.3	3.4	4.0	4.2	4.4

Búsqueda Secuencial en Arreglo Ordenado

Tarea No 15. (Pag. 55): Contar el número de veces que aparece un valor en el arreglo de notas ordenado

```
public int contarOcurrencias ( double valor )
{
}
```

notas =

0	1	2	3	4	5	6	7	8	9	10	11
1.2	1.3	2.0	2.5	2.5	2.7	3.3	3.3	3.3	4.2	4.2	4.4

Búsqueda Secuencial en Arreglo Ordenado

Tarea No 15. (Pag. 56): Contar el número de veces que aparece un valor en el arreglo de notas ordenado

```
public int contarValoresDistintos ( )
{
}
```

notas =

0	1	2	3	4	5	6	7	8	9	10	11
1.2	1.3	2.0	2.0	2.5	2.7	3.3	3.3	3.3	4.0	4.4	4.4

Búsqueda Secuencial en Arreglo Ordenado

Tarea No 15. (Pag. 56): Contar el número de elementos que hay en un rango de valores (incluidos los extremos)

```
public int contarElementosEnRango ( double valorInicial, double valorFinal)
{
}
```

notas =

0	1	2	3	4	5	6	7	8	9	10	11
1.2	1.3	2.0	2.1	2.5	2.7	3.0	3.3	3.4	4.0	4.2	4.4

Búsqueda Secuencial en Arreglo Ordenado

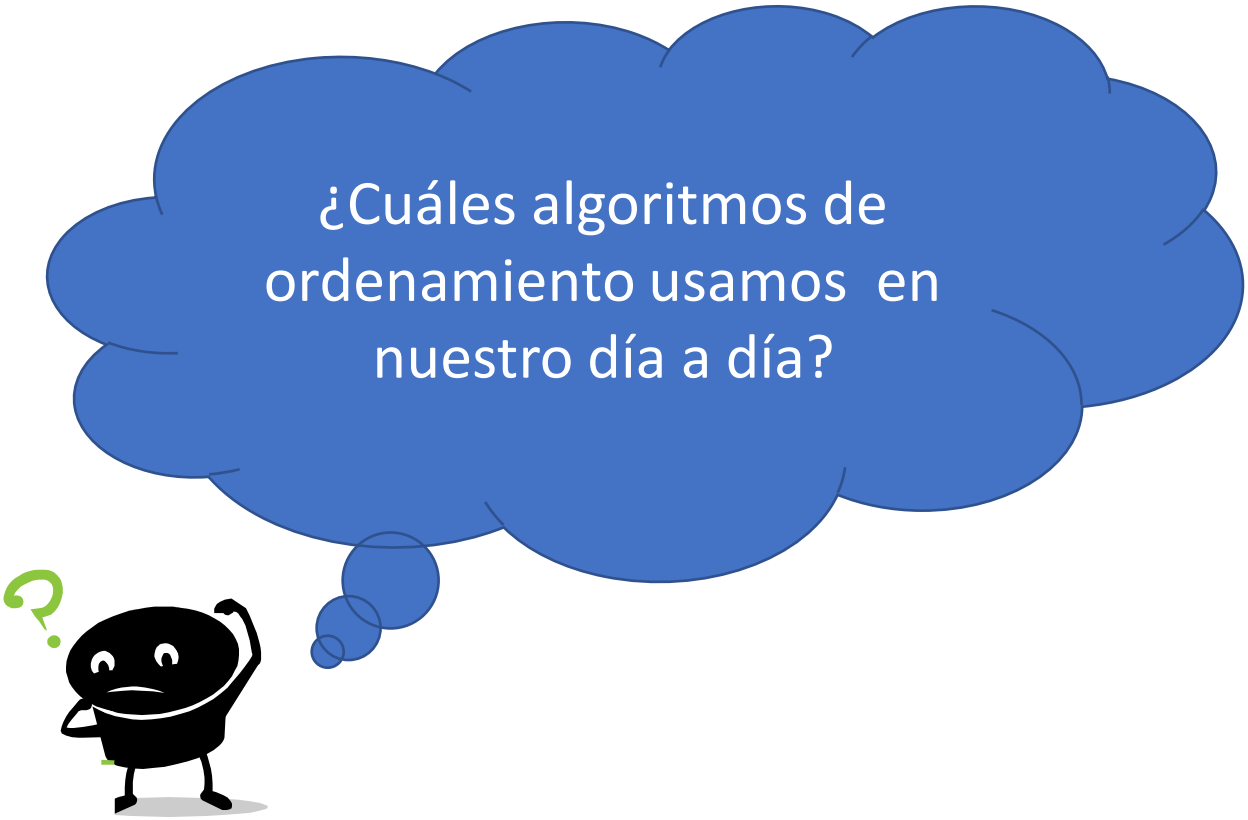
Tarea No 15. (Pag. 56): Contar el valor mas frecuente en el arreglo de notas ordenado. Si hay varias notas con la misma frecuencia, retorna la menor de ellas.

```
public double darNotaMasFrecuente( )
{
}
```

notas =

0	1	2	3	4	5	6	7	8	9	10	11
1.2	1.3	2.1	2.1	2.5	2.5	3.0	3.3	3.4	4.0	4.0	4.0

Algoritmos de Ordenamiento



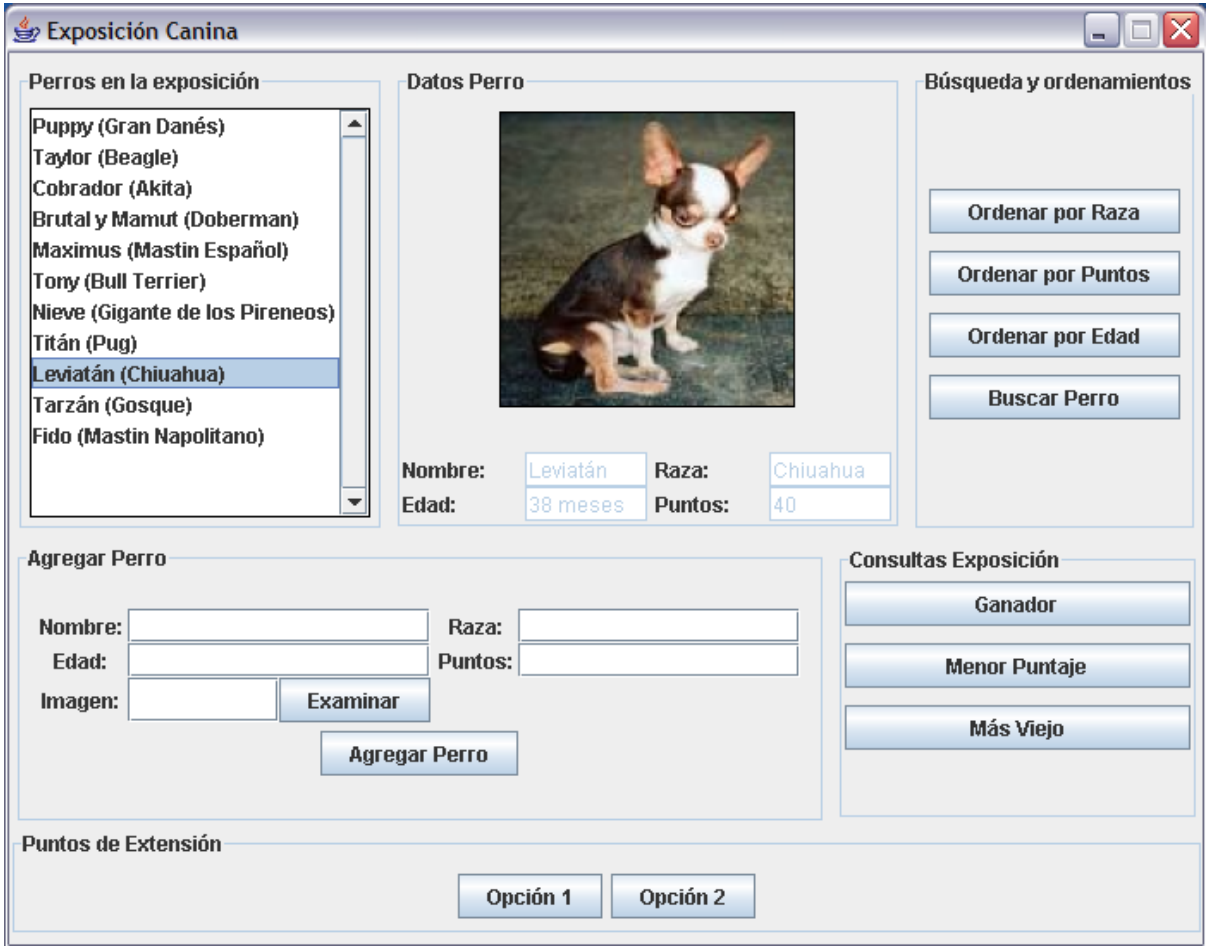
Algoritmos de Ordenamiento (para estudiar)

- Por Selección
- Por Inserción
- Por Intercambio (Burbuja)
- MergeSort

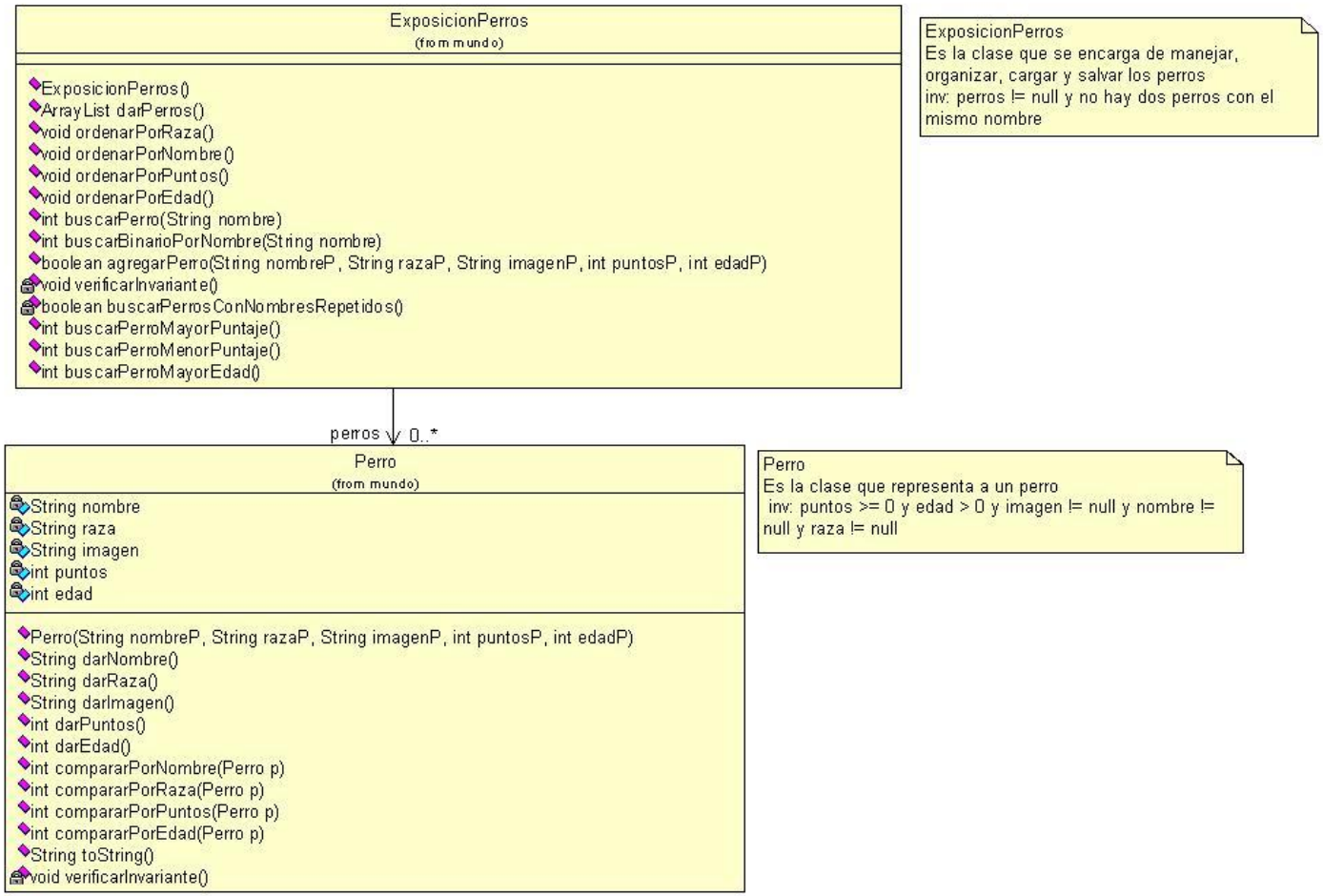
Como ordenamos realmente

```
/* Sorting of arraylist using Collections.sort*/
Collections.sort(arraylist);
```

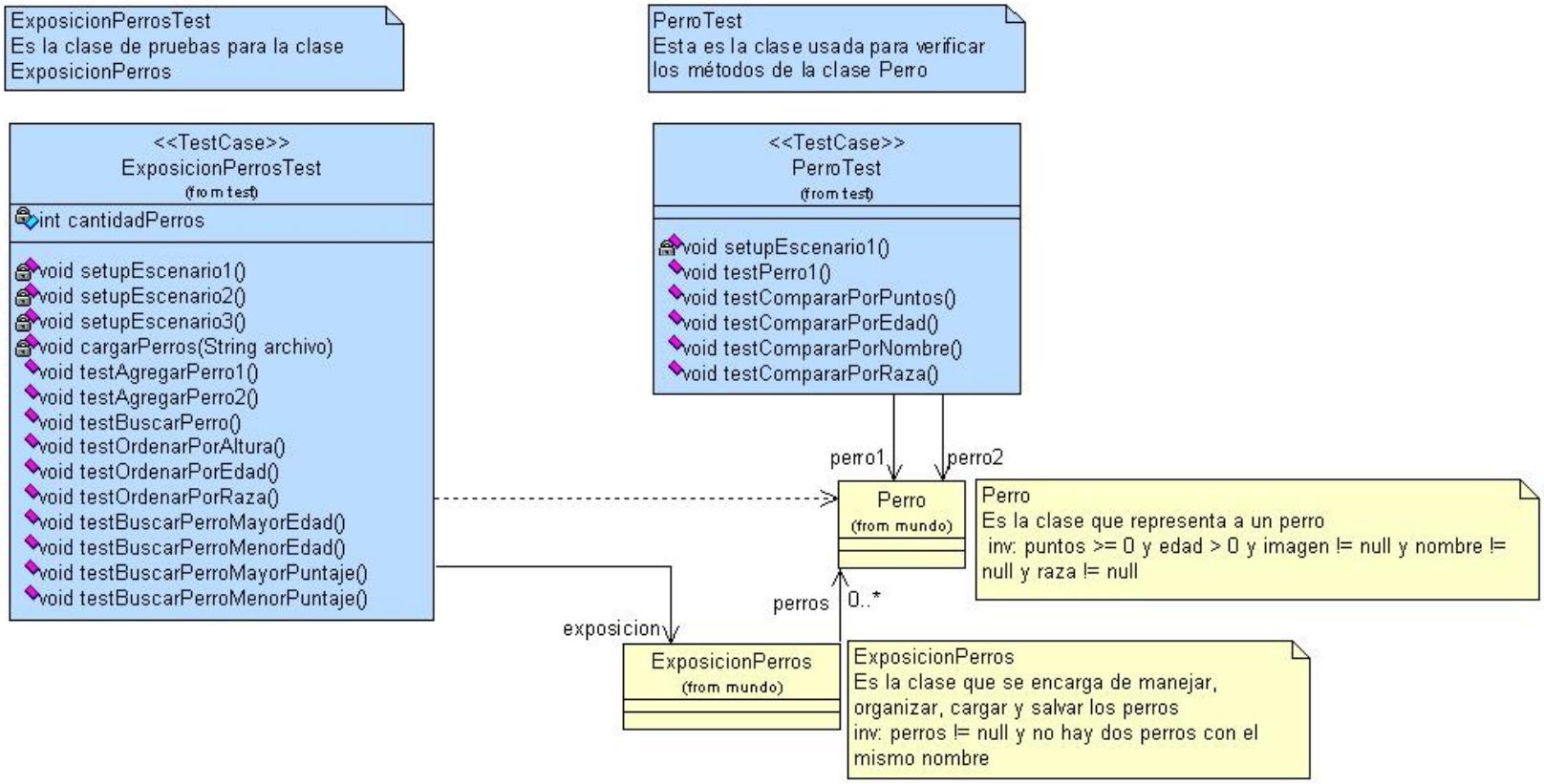
Caso de Estudio: La Exposición Canina



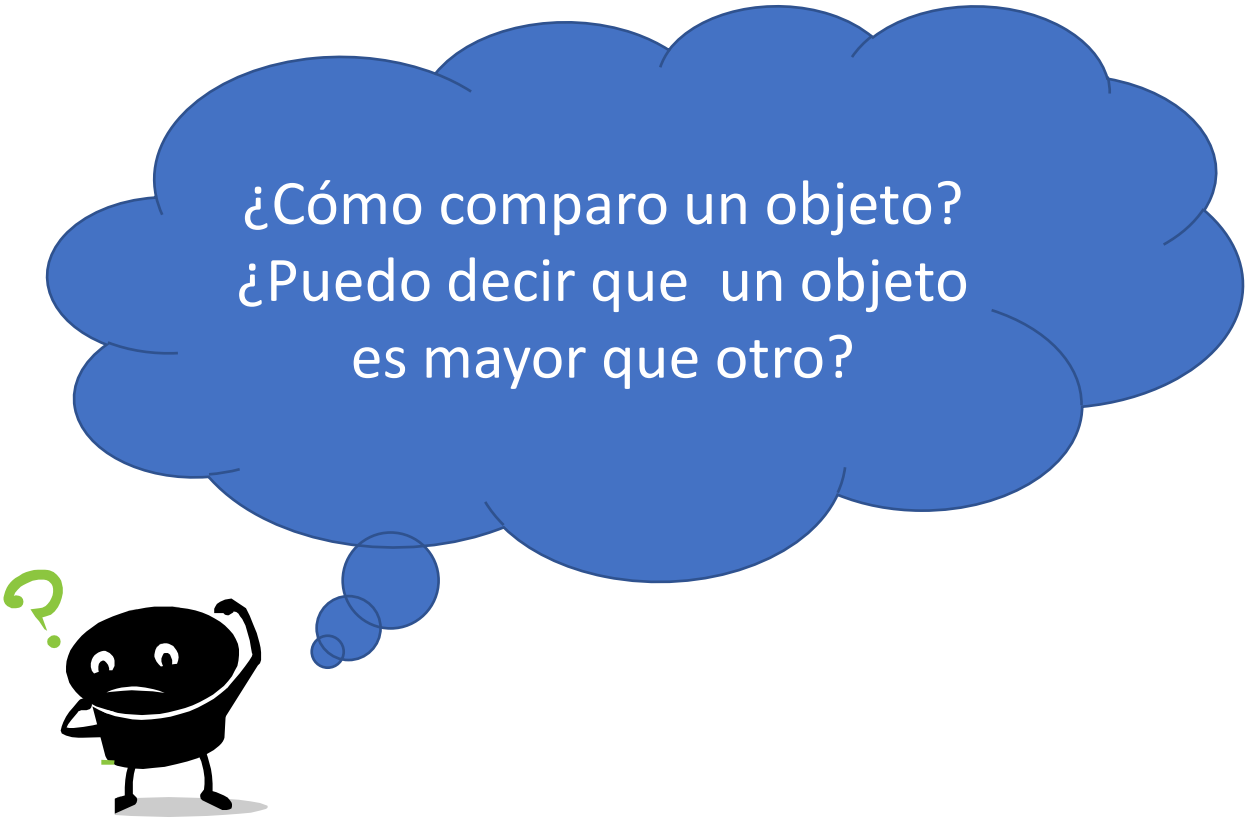
Caso de Estudio: La Exposición Canina



Caso de Estudio: La Exposición Canina



Métodos compararPorX



Métodos compararPorX

- Son métodos que permiten comparar objetos de la clase donde se declara el método con otros objetos del mismo tipo, según criterios (atributos) diferentes.

- Ejemplos en la clase Perro de la exposición canina:
  - compararPorEdad
  - compararPorAltura
  - compararPorRaza
  - compararPorNombre

Métodos compararPorX

Retorna:

0 → si los dos objetos (el actual (this) y con el que se está comparando) son iguales (el atributo que se está comparando tiene el mismo valor en los dos objetos)

1 → si el objeto actual (this) es mayor que el objeto con el cual se está comparando (el atributo que se está comparando tiene un valor mas alto en el objeto actual que en el otro)

-1 → si el objeto actual (this) es menor que el objeto con el cual se está comparando (el atributo que se está comparando tiene un valor mas bajo en el objeto actual que en el otro)

Métodos compararPorX

```
public int compararPorAltura( Perro p )
{
      if ( p.darAltura( ) == altura )
            return 0;
      else if ( p.darAltura( ) < altura )
            return 1;
       else
            return -1;
}
```

Métodos compararPorX

```
public int compararPorRaza( Perro p )
{
        return raza. compareTo( p.darRaza( ) );
}
```

Método compareTo de la clase String. Retorna:

- 0 si los dos strings son iguales
- Un valor mayor a 0 si la raza del objeto actual es “mayor” que la raza de p
- Un valor menor a 0 si la raza del objeto actual es “menor” que la raza de p

Métodos compararPorX

```
public int compararPorRaza( Perro p )
{
      return raza. toLowerCase( ).compareTo( p.darRaza( ).toLowerCase( ) );
}
```

Método toLowerCase de la clase String.
Se utiliza para pasar antes las dos cadenas a minúsculas para evitar problemas de comparación entre mayúsculas y minúsculas

Métodos compararPorX

```
public int compararPorRaza( Perro p )
  {
     return raza.compareToIgnoreCase( p.darRaza( ) );
  }
```

Método compareToIgnoreCase de la clase String.
Funciona igual que el compareTo, pero ignora si está en mayúscula o minúscula