

Laboratorio S03: Clases e Interacción entre Objetos

Proyecto EHUMail

(PARTE 1) Implementar, depurar y documentar clases-objeto

Objetivos:

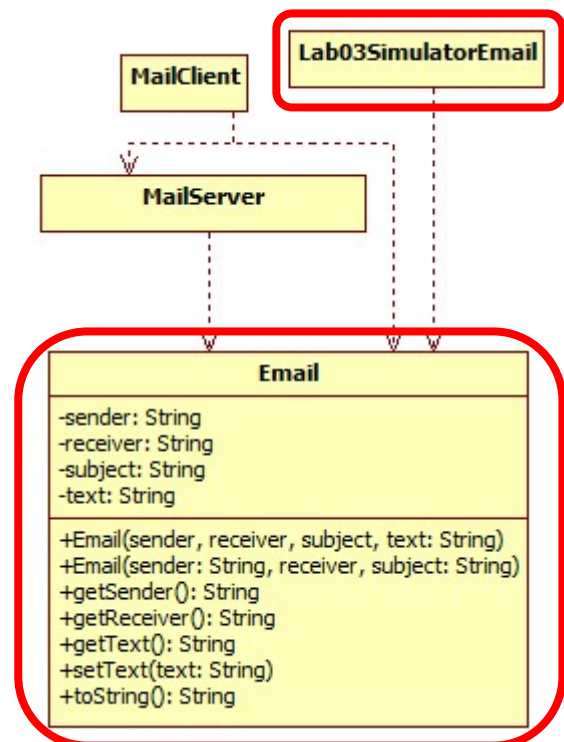
- Entender que la definición de una clase define un nuevo tipo de dato
- Entender los conceptos de abstracción y modularización
- Ser capaz de instanciar o crear múltiples objetos: el operador **new**
 - Ser capaz de definir variables de tipo referencia
 - Ser capaz de construir objetos utilizando distintos constructores
 - Ser capaz de asignar objetos a variables
 - Entender que la asignación de objetos es copiar referencias y no objetos
- Entender que cada objeto es independiente y cada uno tiene su propio estado
- Entender que los valores de las variables de tipo clase son referencias
- Invocar métodos de un objeto:
 - Ser capaz de invocar métodos de un objeto concreto
 - Ser capaz de pasar parámetros a los métodos.
 - Ser capaz de diferenciar entre llamadas internas y externas a métodos
- Uso del debugger de Eclipse
 - Ser capaz de analizar el valor de las variables y los estados de los objetos de un programa
- Documentación con JavaDoc

Proyecto y herramientas que vamos a utilizar:

- Proyecto: *EHUMail*
- Herramientas:
 - entorno de desarrollo Eclipse,
 - depurador (*debugger*) de Eclipse y
 - documentación con JavaDoc

- **Contexto del proyecto:**

El proyecto a desarrollar simulará un sistema de correo electrónico sencillo. El proyecto se irá implementando, documentando y depurando poco a poco. El sistema de correo electrónico permite que sus clientes (**MailClient**) se envíen mensaje. El sistema de envíos entre clientes se gestiona mediante un servidor de correo (**MailServer**). En esta primera parte sólo nos centraremos en la clase que representa el correo electrónico (**Email**).



Tareas a realizar:

Las tareas a realizar en este laboratorio forman parte de un proyecto más amplio que se completará en las próximas semanas. Tareas a realizar esta semana:

1. **Crea un nuevo proyecto Java en Eclipse, llámalo LabS03EHUMail.** A la hora de crear el proyecto, crea un **package mailsystem** (todo en minúsculas).
2. Crea una clase **Email**, dentro del paquete *mailsystem*.

- a. Documenta la clase expresando qué representa y cuál es su utilidad. Indica además tu nombre y la fecha de creación. Para cada método expresa su funcionalidad, los parámetros y el valor devuelto si procede. Para que JavaDoc sea capaz de generar documentación automáticamente han de seguirse estas reglas:

- La documentación para JavaDoc ha de incluirse entre símbolos de comentario que han de empezar con una barra y doble asterisco, y terminar con un asterisco y barra simple.

```
/**  
 * Esto es un comentario ejemplo para JavaDoc  
 */
```

- La ubicación le define a JavaDoc qué representa el comentario: si está incluido justo antes de la declaración de clase se considerará un comentario de clase, y si está incluido justo antes de la signatura de un constructor o método se considerará un comentario de ese constructor o método.
- Para alimentar JavaDoc dentro de los símbolos de comentario se usan ciertas palabras reservadas (tags) precedidas por el carácter "@".).
- En la tabla siguiente mostramos las palabras reservadas (tags) que utilizaremos (ya las iremos viendo poco a poco):

TAG	DESCRIPCIÓN	COMPRENDE
@author	Nombre del desarrollador.	Nombre del autor o autores
@param	Definición de un parámetro de un método, es requerido para todos los parámetros del método.	Nombre de parámetro y descripción (entre el parámetro y la descripción sólo un espacio en blanco. No se debe poner ,)
@return	Informa de lo que devuelve el método, no se aplica en constructores o métodos "void".	Descripción del valor de retorno
@version	Versión del método o clase.	Versión
@throws	Informa de la posible elevación de una Exception . Es necesario uno por cada excepción que se pueda elevar.	Nombre de excepción y condiciones en la que se eleva.

- Por ejemplo, para la clase Email podríamos poner:

```
/**
 * Email represents ...
 * @author XXXX, ¿? february 2020-2021
 *
 */
public class Email {...}
```

- b. Declara los atributos de la clase. Éstos representan las características que tendrán los objetos. Son parecidos a los campos de un registro en Ada (recuerda el laboratorio de la semana pasada), pero en java la orientación a objetos es mucho más. Para evitar malos usos de sus atributos se declaran de tipo **private**. Un email se caracteriza por:

- el nombre del que envía el mensaje (atributo sender tipo String),
- el que debe recibir el mensaje (atributo receiver de tipo String),
- el asunto del mensaje (atributo subject String)
- el texto propiamente dicho (text String).

Implementa los siguientes constructores y métodos dentro de la clase:

- c. **Constructoras**, que serán las que hagan explícitamente la reserva de memoria para los atributos y los inicialicen dependiendo de los parámetros:

- Una constructora con tres parámetros para **sender**, **receiver** y **subject**.
Recuerda documentar constructoras y métodos. Para que se escriba automáticamente el esquema de la documentación de cualquier método, incluidas las constructoras, antes debes escribir la cabecera del método/constructora. Sitúate justo en la línea superior a la misma, escribir **/**** y pulsa salto de línea. Eclipse hará el resto. La documentación podría ser la siguiente (se ha dejado subrayado lo que eclipse ha generado automáticamente):

```
/**
 * Create and initialize the sender, receiver and subject of the email
 * @param sender the sender of the email
 * @param receiver the receiver of the email
 * @param subject the subject of the email
 */
public Email(String sender, String receiver, String subject) {...}
```

El tag **@param** permite establecer el cometido de cada uno de los parámetros del método. Es necesario siempre definirlos.

- Otra constructora con cuatro parámetros.
- d. Todos los **Getters** de los atributos privados que se han definido. Haz uso de la funcionalidad de eclipse para implementarlos automáticamente. Pulsa pestaña Source y selecciona Generate Getters and Setters... En la ventana que se abre, selecciona los atributos de los que quieres que se genere el getter. Una vez seleccionado todo lo deseado, pulsa el botón Generate.
- e. El **Setter** sólo del atributo **text**.
- f. El método **toString** devolverá en formato String el contenido de un email (¡OJO! El método toString no tiene ningún parámetro). Para conseguir que el String contenga saltos de línea para poder luego imprimirlo en varias líneas se pondrá **"\n"**:

```
"Email:\nSender: "+ sender + "\nReceiver: "+receiver+"\nSubject: "+subject+ "\nText: "+ text
```

2020/2021

Por ejemplo, ese String podría ser imprimido mediante `System.out.println` quedando el mensaje en pantalla como sigue: Email:

```
Sender: maite.urretavizcaya@ehu.eus
Receiver: julian.gutierrez@ehu.eus
Subject: Lab S3
Text: Ya están colgados en eGela el enunciado y clases del laboratorio S3.
```

- g. Descárgate el fichero `Lab03SimulatorEmail.java` y cópialo en la carpeta `src` del proyecto dentro de un package `simulators`.
- h. Ejecuta el simulador. Te debe dar como resultado algo así:

```
email1 attributes (by getters):
Sender: maite.urretavizcaya@ehu.eus
Receiver: julian.gutierrez@ehu.eus
Subject: Lab S3
Text: Ya está colgado en eGela el enunciado y clases del laboratorio S3.

email2 attributes (by getters):
Sender: julian.gutierrez@ehu.eus
Receiver: maite.urretavizcaya@ehu.eus
Subject: Lab S3
Text: null

Text: De acuerdo, ¡gracias!
Julian

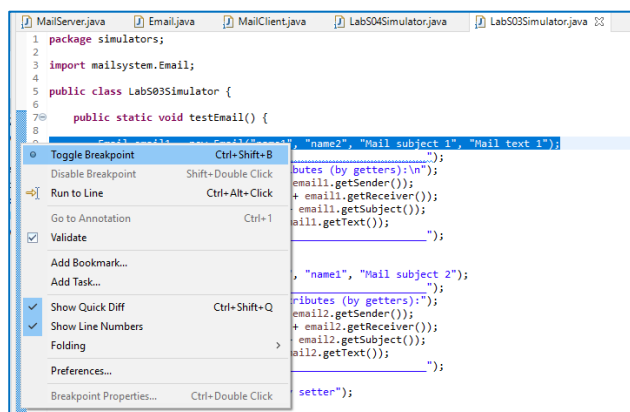
Sender: maite.urretavizcaya@ehu.eus
Receiver: julian.gutierrez@ehu.eus
Subject: Lab S3
Text: Ya está colgado en eGela el enunciado y clases del laboratorio S3.

Email:
Sender: julian.gutierrez@ehu.eus
Receiver: maite.urretavizcaya@ehu.eus
Subject: Lab S3
Text: De acuerdo, ¡gracias!
Julian
```


3. Ahora vamos a hacer una ejecución especial para observar paso a paso como se va ejecutando el simulador. En Eclipse, abre la clase `Lab03SimulatorEmail`.

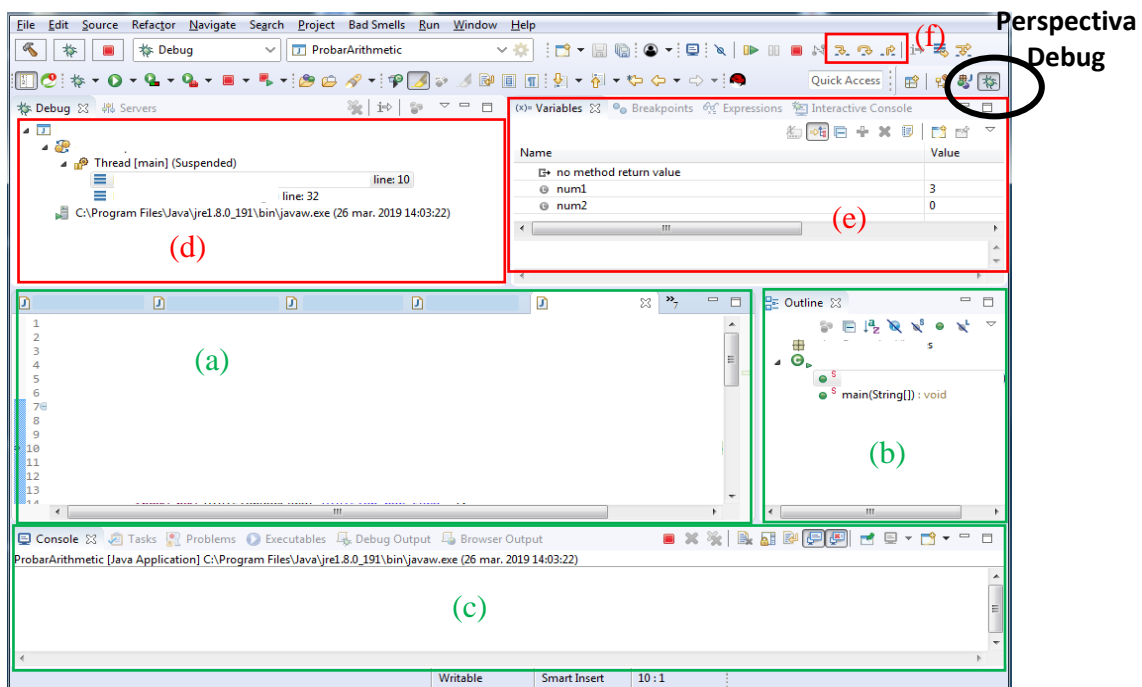
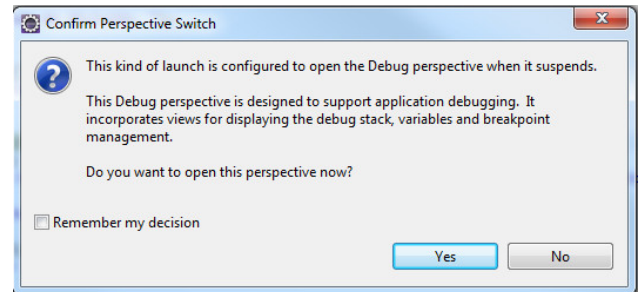
- a. Pon un punto de interrupción en la primera línea de código del método `testEmail`. Sitúate en la zona izquierda delante del número de la primera línea, pulsa el botón izquierdo del ratón y selecciona opción `Toggle Breakpoint`.

- b. En vez de ejecutar el programa de la forma habitual, vamos a hacerlo con el debugger (depurador del código). El depurador te permite una ejecución paso a paso de cada sentencia del programa, visualizando los valores de las variables y el estado de los objetos. En eGela pestaña Software tienes un documento que te explica la información

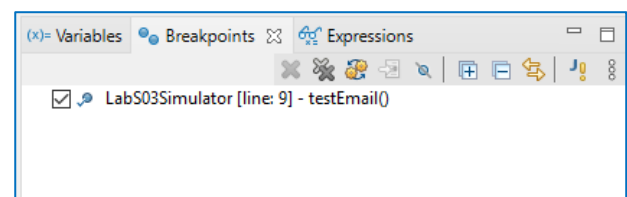


básica del uso del Debugger en Eclipse, ábrelo y síguelo para tener más información. Aquí te pasamos un resumen del mismo Es importante que comprendas y aprendas a utilizarlo. Te ayudará en la depuración de errores.

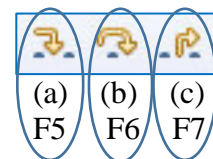
- **Para ejecutar el software** y asegurar que se vaya parando a partir de un determinado punto es necesario poner un Breakpoint.
- Para usar el debugger de Eclipse, en lugar de ejecutar la clase correspondiente con “Run As ...” usar “Debug As ...” o pulsar  Java, se solicita cambio de perspectiva. Pulsa Yes.
- Aparece la **perspectiva Debug** con varias ventanas/iconos significativos. La ventana central con las clases abiertas (**a**), Outline que incluye las características la clase seleccionada en a (**la ventana b**) y Console (**la ventana c**), y, son las habituales de java. Pero aparecen tres partes importantes más.
 - La **ventana (d)** ofrece información del/los métodos que se están ejecutando y en qué línea esta parada la ejecución. Al comenzar la ejecución en modo Debug ésta se detiene en ese punto (Breakpoint).



- La **ventana (e)**:
 - pestaña **Variables**, muestra el estado de las variables locales/parámetros del método que se está ejecutando.
 - pestaña **Breakpoints**, te señalará todos los puntos de los programas donde se tiene breakpoints. Se pueden activar y desactivar los mismos dependiendo de las necesidades de revisión del software. Una vez revisado todo el software para dejarlo limpio te permite eliminar uno a uno cada breakpoint o todos a la vez.



- **Los botones (f)**, permiten navegar paso a paso por las instrucciones del método y observar claramente qué hace cada una de ellas y cómo modifica los valores de las variables. La funcionalidad de cada botón es:
 - a) Estando en una instrucción que corresponde con un método de usuario, permite ir paso a paso dentro de ese método para visualizar su comportamiento.
 - b) Estando en una instrucción que corresponde con un método de usuario, permite seguir en el método actual sin introducirse en el método de usuario.
 - c) Estando dentro de un método que ha sido llamado por otro, le indica que salga de ese método sin revisar paso a paso las instrucciones que quedan por ejecutar (pero el sistema sí las ejecuta).



Algunos métodos complementarios de la clase Email:

- **showEmail**: Muestra por pantalla la información de un Email. Haz uso del método toString para implementarlo (reutiliza código. No repitas código).
- **equalsReceiver**: Recibe como parámetro un objeto de tipo Email e indica (boolean) si el receiver del email actual es igual al receiver del email pasado como parámetro.
- **equals**. Indica si el mensaje pasado por parámetro es igual al actual. Para ello debe coincidir el sender, receiver, subject y text.

Tras la realización de este laboratorio se tiene que comprender:

- Los conceptos de clase, objeto, atributo (private), constructoras, métodos (void y los que devuelven un valor) y método main.
- La documentación de clases y métodos. Controlar el modo de documentar un programa.
- El uso del debugger para poder entender el proceso de ejecución secuencial y en su caso detectar posibles errores o malfuncionamientos de los programas.

4. Termina todo el Laboratorio y súbelo a eGela.

Terminadas todas las tareas debes exportar el proyecto. Para saber cómo exportar un proyecto, es importante que revises la documentación que está disponible en la pestaña **Software** del curso de eGela. Es necesario que subas a eGela el proyecto exportado.

Tienes en eGela la tarea correspondiente para subir tu trabajo (**fecha límite de entrega viernes 12 de febrero de 2021**).