



# Orientación a Objetos 2

## Cuadernillo Semestral de Actividades

### - Frameworks -

**Actualizado: 27 de mayo de 2024**

El presente cuadernillo estará en elaboración durante el semestre y tendrá un compilado con todos los ejercicios que se usarán durante la asignatura respecto al tema Frameworks.

### Ejercicio 1: Java Logging

En las clases teóricas de frameworks se trabajó con el framework de [Logging de Java](#). Con lo visto en teoría y leyendo la documentación provista en el link anterior, resuelva los siguientes ejercicios.

**Parte A:** En este apartado utilizaremos el framework como usuarios, aprovechando las implementaciones ya provistas por éste.

i ) Tomando su implementación del ejercicio de protección para el acceso a una base de datos (Cuadernillo patrones, ejercicio 14), incorpore logging de mensajes en las siguientes situaciones:

- Acceso válido para búsquedas a la base de datos con nivel INFO.
- Acceso válido para inserciones a la base de datos con nivel WARNING.
- Acceso inválido a la base de datos con nivel SEVERE.
- 

ii ) Retomamos el ejercicio de Wallpost de Objetos 1, donde trabajamos con mensajes de una red social como Facebook o Twitter. Se cuenta con una clase Wallpost con los siguientes atributos: un texto que se desea publicar, cantidad de likes ("me gusta") y una marca que indica si es destacado o no.

Para realizar este ejercicio, descargue este [material adicional](#). La implementación provista consta de tres paquetes: uno destinado a la aplicación, otro al modelo y el tercero a la interfaz de usuario de la aplicación.

Nos piden implementar dos registros de eventos, uno destinado al modelo y otro destinado a las interacciones realizadas con la interfaz. El logger del modelo debe informar un mensaje con nivel warning cuando los dislikes hagan llegar la cantidad de likes a 0 y cuando la cantidad de likes llegue a 10. Además se pide realizar estos registros en un archivo de texto. Por otro lado, el logger de la parte visual debe registrar con nivel info todas las interacciones con la vista, tales como escribir el nombre del post, clicar en like o dislike. Además se solicita que registre un mensaje con nivel de información al iniciar la ejecución de la aplicación.

Asegúrese de completar todas las configuraciones de los loggers requeridas anteriormente dentro de la clase `Ejercicio1Application`

**Parte B:** A partir de este ejercicio, vamos a necesitar funcionalidad extra que el framework no provee y, por lo tanto, lo extenderemos para que se adapte a nuestras necesidades.

i ) Extienda el framework de logging de Java para poder formatear los mensajes de log de las siguientes maneras:

- Realizar el registro de un mensaje completamente en mayúscula, similar al ejercicio resuelto en teoría. Utilice este formato en los casos aplicados del inciso A.
- Realizar el registro de un mensaje en formato JSON. El resultado de hacer logging con este formato debería ser un String en formato JSON con los campos *message* y *level* y como valores el string registrado y el nivel de severidad. Utilice este formato en los casos aplicados del inciso A.

Por ejemplo:

```
logger.info("Logging with json");
```

Debería generar como salida el siguiente mensaje:

```
{ "message": "Logging with json", "level": "info" }
```

ii ) Realice las siguientes dos extensiones al framework:

- Agregar un Handler que aplique un filtro de palabras a ocultar antes de ejecutar un Handler existente. El mismo debe poder configurarse con una lista de palabras a ocultar y reemplazar cada aparición de alguna de éstas con el String `***`. Por ejemplo, si se configura este handler con la lista `["switch-statements"]` debería suceder que:

```
logger.info("I love switch-statements");
```

en realidad realice el log del String:

```
"I love ***"
```

- Mediante un handler posibilite la opción de enviar por correo electrónico los mensajes registrados por el framework. Al final del documento encontrará un anexo con una solución general al envío de mails desde una aplicación java. Si lo considera conveniente, puede seguir estos pasos para resolver el envío de mails, adaptando lo que considere necesario para que la funcionalidad se ejecute al momento de realizar el logging de un mensaje.
- Aplique ambos handlers en los escenarios de logging planteados en el inciso de la parte A.

## Ejercicio 2: Extensión de Frameworks

Dado el recurso de aprendizaje de “plantillas y ganchos” que se encuentra en la plataforma cátedras en [esta URL](#). Realice las siguientes tareas:

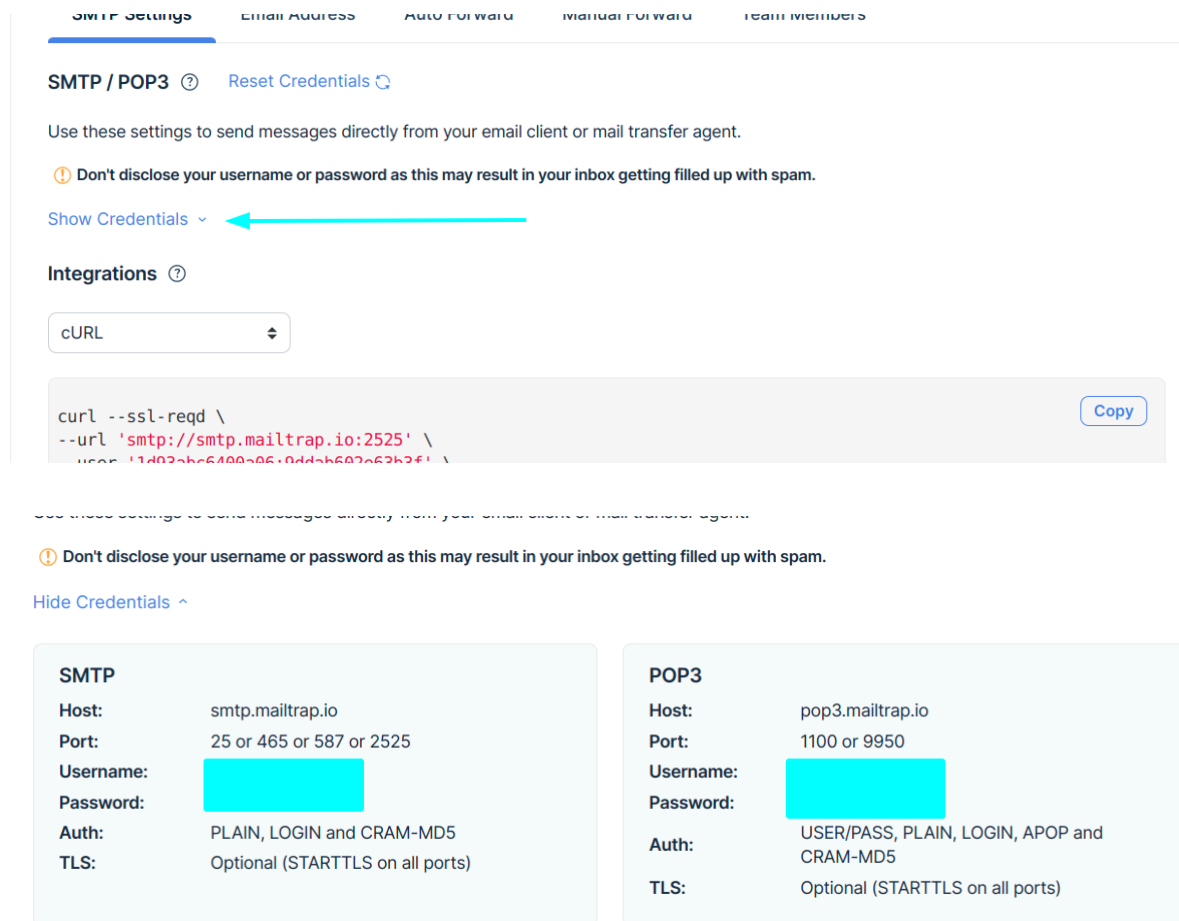
- Lea todo el material provisto en el recurso de aprendizaje
- Responda las preguntas correspondientes al ejercicio de plantillas y **herencia** (que a continuación se copian):
  - a. ¿Qué debo hacer si aparece una nueva fuente de energía (por ejemplo, paneles solares con baterías)? ¿Cuántas y cuáles clases debo agregar en caso de querer todas las variantes de robots posibles para este nuevo tipo de fuente de energía?
  - b. ¿Puedo cambiarle, a un robot existente, el sistema de armas sin tener que instanciar el robot de nuevo?
  - c. ¿Dónde almacenaría usted el nivel de carga de la batería? ¿Qué implicaría eso si antes de disparar el láser hay que garantizar que la fuente de energía puede satisfacer el consumo del arma?
- Implemente en Java lo necesario para satisfacer el punto a. Luego, agregue un nuevo ejemplo de uso del framework instanciando uno de los robots con la nueva fuente de energía.
- Responda las preguntas correspondientes al ejercicio de plantillas y **composición** (que a continuación se copian):
  - a. ¿Qué debo hacer si aparece una nueva fuente de locomoción (por ejemplo, motor con ruedas con tracción 4x4)? ¿Cuántas y cuáles clases debo agregar en caso de querer todas las variantes de robots posibles para este nuevo tipo de sistema de locomoción?
  - b. ¿Puedo cambiarle, a un robot existente, el sistema de armas sin tener que instanciar el robot de nuevo?
  - c. ¿Dónde almacenaría usted el nivel de carga de la batería? ¿Qué implicaría eso si antes de disparar el láser hay que garantizar que la fuente de energía puede satisfacer el consumo del arma?
- Implemente en Java lo necesario para satisfacer el punto a. Luego, agregue un nuevo ejemplo de uso del framework instanciando uno de los robots con la nueva forma de locomoción.
- Explique las ventajas y desventajas de las dos formas de extensión del framework (herencia y composición).

## Anexo Mailtrap

Mailtrap es una herramienta que implementa un “falso servidor SMTP”. Es ideal para pruebas, ya que nos permite recibir correos electrónicos en una bandeja de entrada en la nube. Al registrarnos, nos permite crear tantas inbox cómo queramos, y nos provee de las credenciales para poder enviar los mails desde nuestras aplicaciones:

1. Ingrese a <https://mailtrap.io/> y regístrese.
2. En el dashboard, presione el botón “Add project”. Nos va a pedir que indiquemos un nombre para crearlo.
3. Una vez creado el proyecto, presione el botón “Add inbox”, en donde nuevamente nos pedirá indicar el nombre.

4. En la tabla aparecerá la bandeja de entrada creada en el paso anterior, haga click sobre su nombre (o bien, en el botón “settings”).
5. Se nos presentan los datos del inbox; necesitamos obtener la configuración de conexión para poder incorporarla en nuestro proyecto Java. Para ver estas credenciales, haga click en “Show credentials”.



SMTP Settings | Email Address | Auto Forward | Manual Forward | Team Members

### SMTP / POP3 ? [Reset Credentials](#)

Use these settings to send messages directly from your email client or mail transfer agent.

⚠ Don't disclose your username or password as this may result in your inbox getting filled up with spam.

[Show Credentials](#) ▼

#### Integrations ?

cURL

```
curl --ssl-reqd \
--url 'smtp://smtp.mailtrap.io:2525' \
--user '11d023b76400a0610dd4b607a63b2f1' \
--pass '11d023b76400a0610dd4b607a63b2f1'
```

[Copy](#)

⚠ Don't disclose your username or password as this may result in your inbox getting filled up with spam.

[Hide Credentials](#) ^

#### SMTP

Host: smtp.mailtrap.io

Port: 25 or 465 or 587 or 2525

Username: [REDACTED]

Password: [REDACTED]

Auth: PLAIN, LOGIN and CRAM-MD5

TLS: Optional (STARTTLS on all ports)

#### POP3

Host: pop3.mailtrap.io

Port: 1100 or 9950

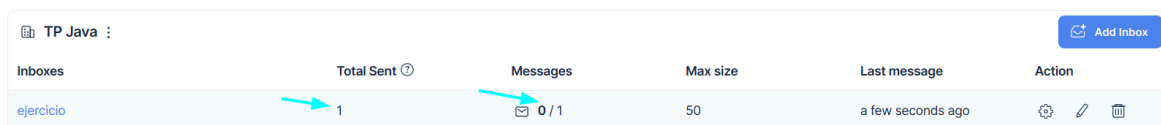
Username: [REDACTED]

Password: [REDACTED]

Auth: USER/PASS, PLAIN, LOGIN, APOP and CRAM-MD5

TLS: Optional (STARTTLS on all ports)

6. Con la información de las credenciales (username y password) modifique la clase **MailExample** que se encuentra en el proyecto provisto en el [material adicional](#).
7. Ejecute la clase anterior, y verifique que dentro de la bandeja aparece un correo electrónico con el texto y tema utilizado en el ejemplo.







Inboxes	Total Sent ?	Messages	Max size	Last message	Action
<a href="#">ejercicio</a>	1	0 / 1	50	a few seconds ago	

Dentro de la bandeja, se puede ver el mail:



Search...



Tema del mail

to: <destination@mail.com>      a minute ago

Tema del mail

From: Java logging mail <example@logger.com>

To: <destination@mail.com>

Show Headers

HTML

HTML Source

Text

Raw

Spam Analysis

Tech Info

Texto del mail

