Orientación a Objetos 2
Cuadernillo Semestral de Actividades
- Frameworks -

Actualizado: 26 de mayo de 2025

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: SingleThreadTCPFramework

A partir del código compartido en teoría de SingleThreadTCPFramework.

i. Refactorizar el método SingleThreadTCPFramework::handleClient(Socket) para convertirlo en un Template Method. Este método debe incluir métodos hook opcionales, es decir, métodos hooks que pueden ser implementados por las subclases o no.

ii. Extienda el framework para permitir que la “palabra” que produce el cierre de la sesión con un cliente sea configurable. Evalúe las siguientes cuatro alternativas e implemente la que considere más adecuada:
   a. Una variable en SingleThreadTCPServer que se configura desde el método main() de las subclases.
   b. Un método (hook) que retorna un booleano resultado de evaluar la condición.
   c. Un método (hook) que retorna un String que es la palabra de término de sesión.
   d. Una jerarquía de Strategies que implementan cada una de las condiciones de cierre de sesión.

Para ayudarlo en definir cuál sería la alternativa que considera más apropiada, puede usar los siguientes aspectos (pero no limitarse a):

- Esfuerzo de implementación dentro del framework
- Facilidad de uso para los programadores usuarios del framework
- Limitaciones de la solución; es decir, que tan flexible es la alternativa elegida, ¿que casos de uso permite abarcar y cuales no podría?

iii. Implemente un servidor PasswordServer. Este servidor debe generar una password a partir de los tres argumentos que recibe en el mensaje enviado por el cliente.
- Arg[0]: cadena de caracteres (letras) permitidas para utilizar en la password
- Arg[1]: cadena de caracteres (números de 0 a 9) permitidos para utilizar en la password
- Arg[2]: cadena de caracteres especiales permitidos para utilizar en la password

Las passwords deben ser generadas de forma aleatoria y complir con las siguientes reglas:

Tener una longitud de 8 caracteres
Contener letras, al menos un número y un solo carácter especial

iv. Implemente un servidor RepeatServer. Este servidor debe repetir un string a partir de los argumentos que recibe en el mensaje enviado por el cliente:
- Arg[0]: es el string a repetir. Este argumento es requerido y no puede ser nulo o vacío.
- Arg[1]: es la cantidad de veces que debe repetir el string. Este argumento es requerido y debe ser un número entero mayor a 0.
- Arg[2]: es un carácter que se utiliza para delimitar los strings repetidos. Este argumento es opcional, y en caso de que el cliente no lo especifique, es un espacio en blanco.

Notas:
Para repetir un String, puede utilizar el método repeat que ya tienen los objetos String
Para ver sí un String está vacío, puede utilizar el método isEmpty
Para convertir un String a int, puede utilizar el método estático parseInt de la clase Integer:
Recuerde que debe manejar la excepción NumberFormatException

Ejercicio 2: 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 17), 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, clickear 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 para realizar instanciaciones similares a las realizadas en el 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 para realizar instanciaciones similares a las realizadas en el 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 para realizar instanciaciones similares a las realizadas en el inciso A.

Ejercicio 3: Extensión de Frameworks

Dado el recurso de aprendizaje de “plantillas y ganchos” que se encuentra en la plataforma cátedras en esta URL. Lea todo el material provisto en el recurso de aprendizaje

1) Respecto a la sección **Plantillas y herencia** del recurso de aprendizaje

- Responda las siguientes preguntas:
  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 sí 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.

2) Respecto a la sección **Plantillas y composición** del recurso de aprendizaje

- Responda las siguientes preguntas:
    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 sí 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.

3) Explique las ventajas y desventajas de las dos formas de extensión del framework (herencia y composición).

Ejercicio 4: tcp.server.reply (B)

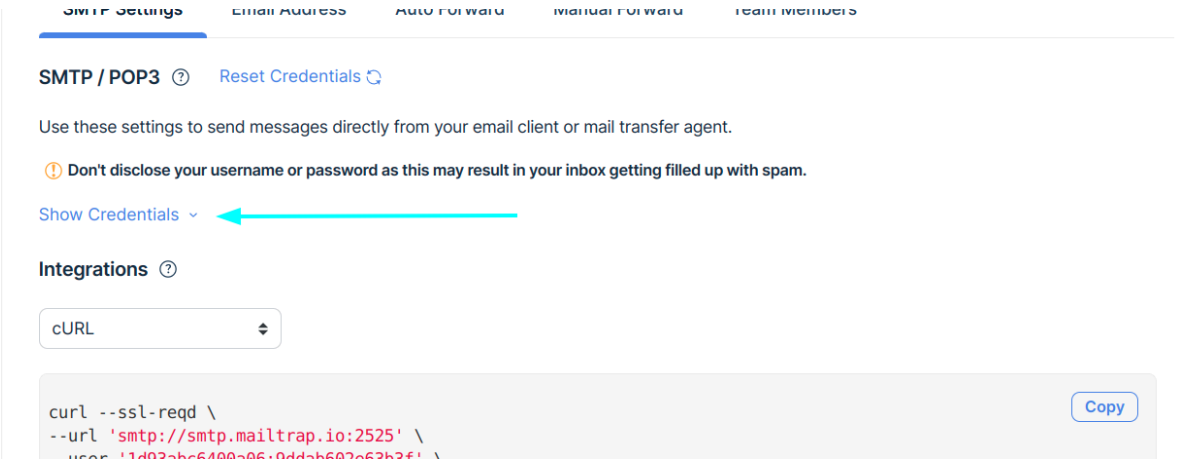
A partir del código compartido en teoría de tcp.server.reply.

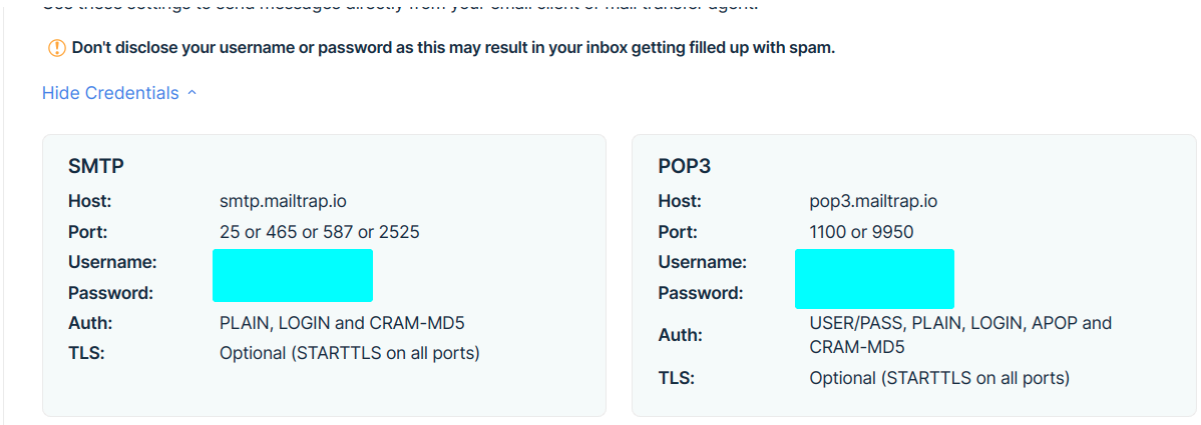
i. Reimplementar el servidor PasswordServer empleando un enfoque basado en composición de objetos utilizando los componentes del framework `tcp.server.reply` vistos en teoría
ii. Modifique el framework para que la condición de cierre de una conexión sea configurable con strings provistos por las instanciaciones.
iii. Respecto a las dos formas vistas para implementar los servidores (PasswordServer ejercicio 1) iii) y 4) i) :
    - ¿Qué debe hacer un desarrollador para extender el framework en cada una de las formas? Especifique qué clases debe subclasificar o implementar, qué métodos debe definir.
    - ¿Cuánto conocimiento necesita tener el desarrollador sobre la estructura interna del framework para instanciarlo? ¿Y para extenderlo?
    - ¿Qué técnica usarías si tuviera que ofrecer muchas configuraciones posibles para el servidor? ¿Por qué?
    - Identifique los hotspots y frozen spots en cada una de las implementaciones.
    - Considerando las dos formas de implementación del servidor PasswordServer, los programadores pueden asegurar que hay inversión de control? Justifique su respuesta identificando en qué parte se produce la inversión de control en cada uno de los casos.

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:

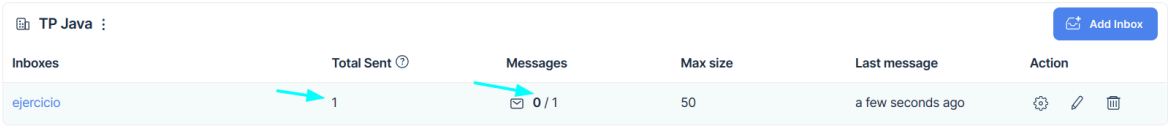
1. Ingrese a https://mailtrap.io/ y regístrese.
2. En el dashboard, presione el botón “Add project”. Nos va a pedir que indiquemos un nombre para crearlo.
3. Una vez creado el proyecto, presione el botón “Add inbox”, en donde nuevamente nos pedirá indicar el nombre.
4. En la tabla aparecerá la bandeja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





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.



Dentro de la bandeja, se puede ver el mail: