Orientación a Objetos II 2025

Explicación de práctica
Semana del 19 de mayo

Ejercicios de la semana

En el cuadernillo de Frameworks

- Ejercicio 1: SingleThreadTCPFramework
  - Ejercicio ii - Extension del framework
  - Ejercicio iii - PasswordServer
  - Ejercicio iv - Repeat server
- Ejercicio 2: Java Logging
- Ejercicio 3: Extensión de Frameworks

Ejercicio 1: SingleThreadTCPFramework

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

```java
private final void handleClient(Socket clientSocket) {
    try (
            PrintWriter out = new PrintWriter(clientSocket.getOutputStream(), true);
            BufferedReader in = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));) {
        String inputLine;
        while ((inputLine = in.readLine()) != null) {
            System.out.println("Received message: " + inputLine + " from "
                    + clientSocket.getInetAddress().getHostAddress() + ":" + clientSocket.getPort());

            if ("".equalsIgnoreCase(inputLine)) {
                break; // Client requested to close the connection
            }
            this.handleMessage(inputLine, out);
        }
        System.out.println("Connection closed with " + clientSocket.getInetAddress().getHostAddress() + ":"
                + clientSocket.getPort());
    } catch (IOException e) {
        System.err.println("Problem with communication with client: " + e.getMessage());
    } finally {
        try {
            clientSocket.close();
        } catch (IOException e) {
            System.err.println("Error closing socket: " + e.getMessage());
        }
    }
}
```

Ejercicio 1: SingleThreadTCPFramework

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

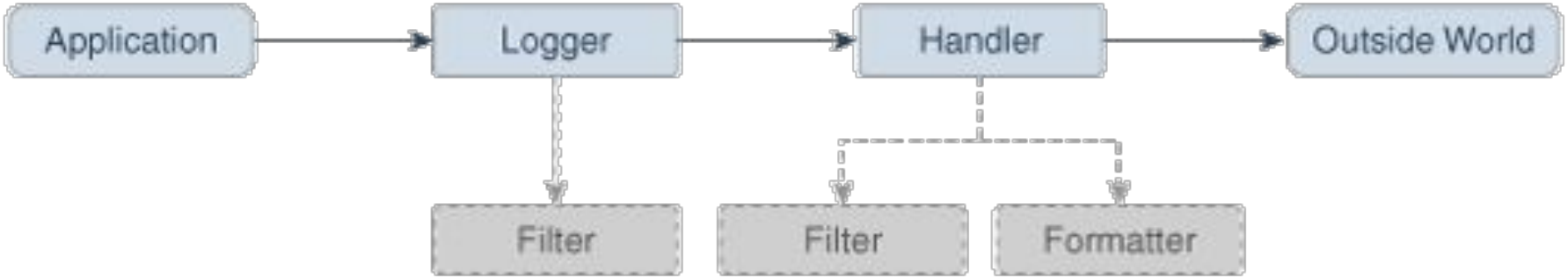
Ejercicio 1: SingleThreadTCPFramework

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.

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.

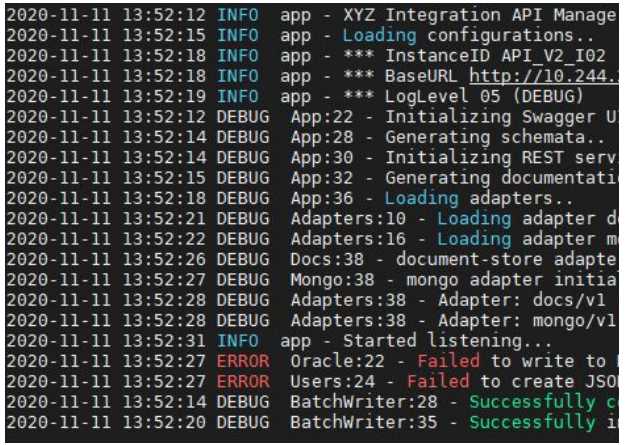


Ejercicio 2: Java Logging

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.



Ejercicio 2: Java Logging

Parte B:

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" }
  ```

Ejercicio 2: Java Logging

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 ***"
```

Frameworks - Logging

ii) Realice las siguientes dos extensiones al framework:

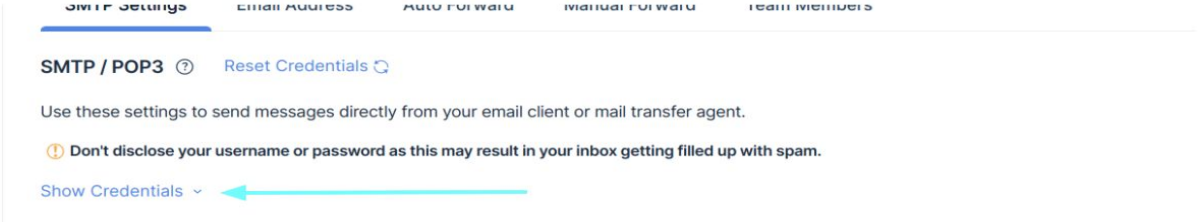
- Mediante un handler posibilite la opción de enviar por correo electrónico los mensajes registrados por el framework. Al final del documento encontrará un anexo con una solución general al envío de mails desde una aplicación java. Si lo considera conveniente, puede seguir estos pasos para resolver el 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.

Frameworks - Anexo Mailtrap

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 querramos, 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”.

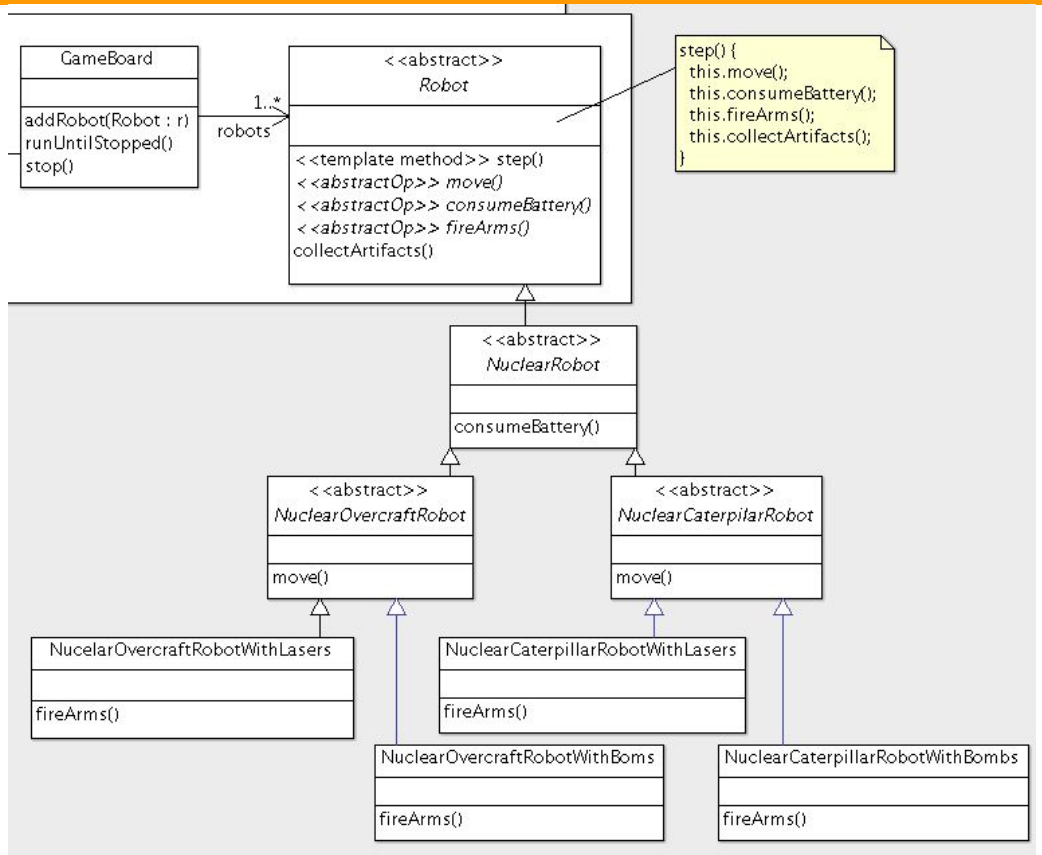


Ejercicio 3: Extensión de Frameworks

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.

Ejercicio 3: Extensión de Frameworks

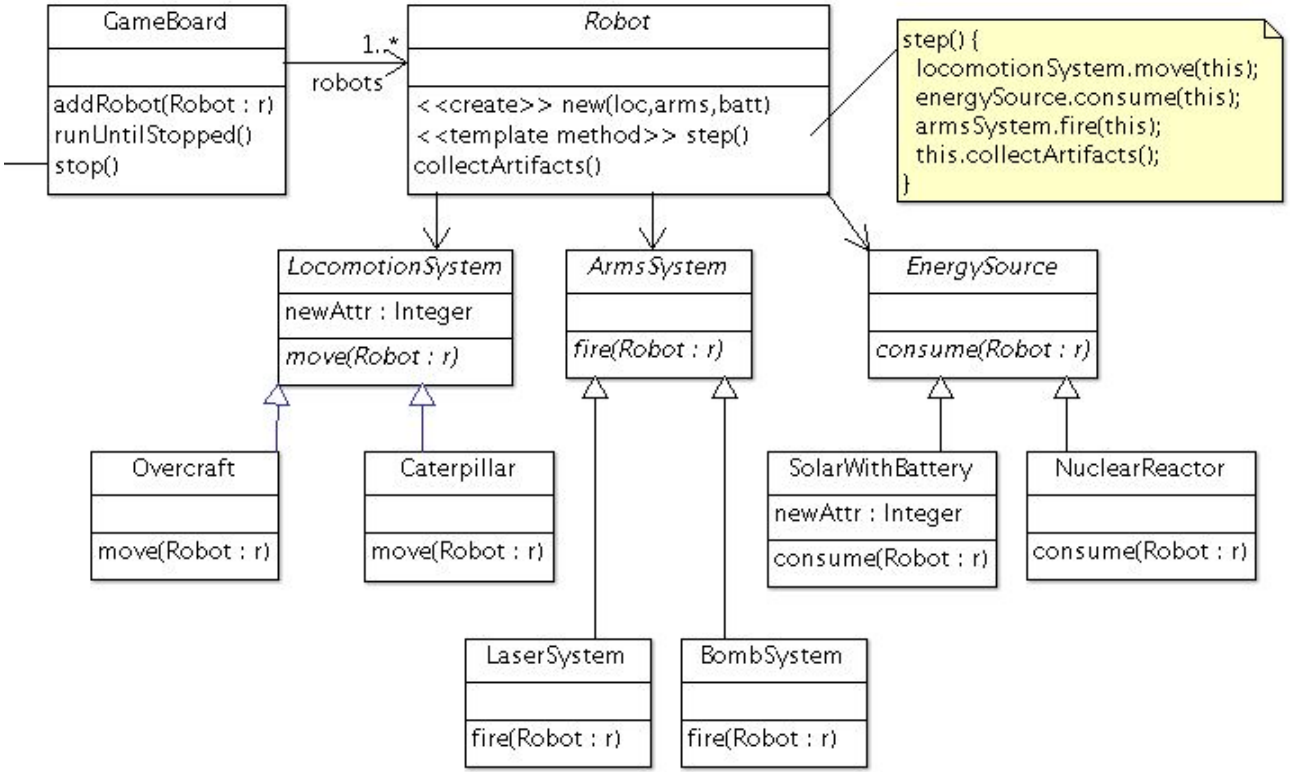


Ejercicio 3: Extensión de Frameworks

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.

Ejercicio 3: Extensión de Frameworks



Examen Parcial

- **Primera fecha**: **Sábado 7 de junio, 13,00 horas**
  aula 5, 4, 9, 10a y 11

- 1er Recuperatorio: Sábado 28 de junio, 13,00 horas
  aula 5, 4, 9, 10a y 11

- 2do Recuperatorio: Sábado 12 de julio, 13,00 horas
  aula 5, 4, 10a y 11
  + (aula a definir) + promoción