Práctica 5

Colecciones, genericidad, expresiones lambda y patrones de diseño

Inicio: Semana del 17 de Abril
Duración: 3 semanas
Entrega: 5 de Mayo, 23:55h (todos los grupos)
Peso de la práctica: 30%

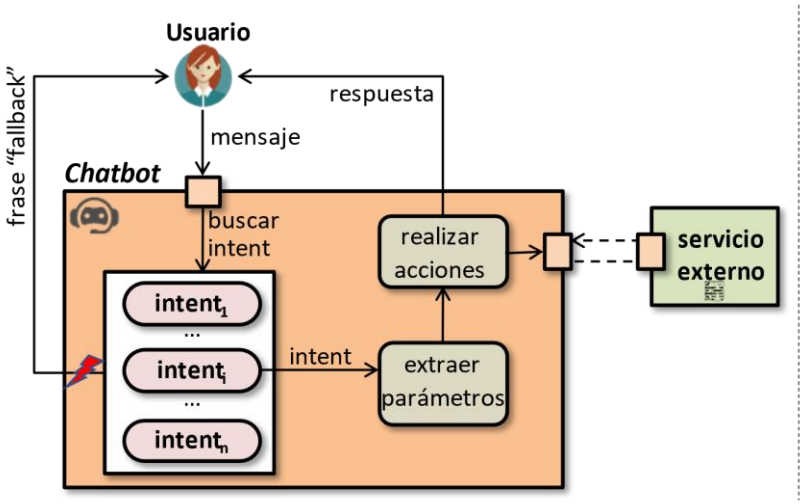
El objetivo de esta práctica es diseñar programas genéricos que usen colecciones avanzadas, sean capaces de adaptarse a tipos paramétricos, empleando expresiones lambda y patrones de diseño de manera práctica. Para ello, construiremos una librería (muy simplificada) para la construcción de chatbots para tareas específicas.

Introducción

Los chatbots son programas que dan acceso a servicios informáticos mediante conversación en lenguaje natural. A diferencia de los chatbots para dominios abiertos, como ChatGPT, los chatbots para tareas específicas se construyen para resolver una tarea concreta, como reservar un billete de avión, resolver un problema con la conexión a internet, o pedir una pizza. Estos chatbots se pueden integrar en páginas web, redes sociales como Telegram, o bien en altavoces inteligentes. Existen numerosas tecnologías para este propósito, como Dialogflow (de Google), Lex (de Amazon), o Rasa (de código abierto).

En este tipo de chatbots, se diseñan las posibles conversaciones explícitamente, definiendo "*intents*" (intenciones) que pueda tener el usuario en la conversación. Por ejemplo, en un chatbot para una cafetería, los *intents* incluirían la obtención de la lista de bebidas y pedir bebidas. Los *intents* se describen mediante conjuntos de *frases de entrenamiento*. Dada una frase del usuario, el chatbot comprueba cada intent con el objetivo de encontrar una frase que concuerde. Si es así, el chatbot produce la respuesta asociada al intent. Mientras que los sistemas reales utilizan sistemas avanzados de procesamiento del lenguaje natural (con redes neuronales) para detectar concordancias entre la frase del usuario y las frases de entrenamiento, nosotros simplemente compararemos las cadenas. Finalmente, los intents pueden esperar parámetros. Por ejemplo, en un intent para pedir un café, podemos esperar el tipo de café (espresso, capuccino, con leche), el tamaño del café (pequeño, mediano, grande), y el número (un entero).

Más abajo tienes un esquema de funcionamiento de los chatbots, así como un ejemplo de conversación de un usuario con un chatbot de una cafetería. En un sistema real, el chatbot estaría conectado con un servicio externo, para, por ejemplo, consultar en una base de datos, o realizar peticiones a un sistema de información. Nuestro diseño estará muy simplificado, así que no consideraremos servicios externos.



Apartado 1: Frases para los intents (2.5 puntos)

En este apartado, comenzaremos definiendo la estructura de las frases para los intents. Las frases estarán construidas por cadenas de texto, que podrán contener parámetros (con nombre) de cualquier tipo. Así, por ejemplo podremos definir frases como "*I'd like a coffee*" (sin parámetros), o bien "*I'd like a* [coffee-type:CoffeeType(CAPUCCINO)]" (con un parámetro llamado coffee-type cuyo tipo es `CoffeeType` – una enumeración, y su valor por defecto es CAPUCCINO), o "*Can I have* [coffee-number:Integer(2)] [coffee-type:CoffeeType(ESPRESSO)] *please?*" (con dos parámetros, el primero el número de cafés, y el segundo el tipo).

En nuestro diseño, construiremos las frases a base de concatenar literales de texto y parámetros genéricos (ambos representados como objetos), como puedes ver en el siguiente ejemplo. El método `with` de `StructuredPhrase` añade a la frase en construcción bien un literal de texto, o bien un parámetro, con un valor. El método `setting`, permite añadir valores por defecto a los parámetros que no se mencionan explícitamente en la frase.

```
public class TextInputMain {
  public static void main(String[] args) {
       StructuredPhrase[] phrases = createPhrases();

       for (StructuredPhrase sp: phrases) {
              System.out.println("'"+sp+"' with parameters:");
              System.out.println("  + coffee-type   = "+sp.getValue("coffee-type"));
              System.out.println("  + coffee-number = "+sp.getValue("coffee-number"));
       }
  }
  public static StructuredPhrase[] createPhrases() {
       StructuredPhrase phrases[] = {
              new StructuredPhrase()
                     .with("I'd like a")
                     .with("coffee-type", CoffeeType.CAPUCCINO)
                     .setting("coffee-number", 1),     // coffee-number vale 1
              new StructuredPhrase()
                     .with("I'd like a coffee")
                     .setting("coffee-number", 1)      // coffee-number vale 1
                     .setting("coffee-type", CoffeeType.ESPRESSO),  // coffee-type vale espresso
              new StructuredPhrase()
                     .with("Can I have")
                     .with("coffee-number", 2)
                     .with("coffee-type", CoffeeType.ESPRESSO)
                     .with("please?")};
       return phrases;
  }
}
```

Salida esperada:

```
'I'd like a [coffee-type:CoffeeType(CAPUCCINO)]' with parameters:
  + coffee-type   = CAPUCCINO
  + coffee-number = 1
'I'd like a coffee' with parameters:
  + coffee-type   = ESPRESSO
  + coffee-number = 1
'Can I have [coffee-number:Integer(2)] [coffee-type:CoffeeType(ESPRESSO)] please?' with parameters:
  + coffee-type   = ESPRESSO
  + coffee-number = 2
```

Donde CoffeeType es un enumerado: `public enum CoffeeType { CAPUCCINO, AMERICANO, ESPRESSO }`

Nota:
Para imprimir el tipo del parámetro, puede serte útil el método `getClass()` de `Object`, así como el método `getSimpleName()` de la clase `Class`.

Apartado 2: Intents (4 puntos)

A continuación, crearemos clases para definir *intents*. Distinguiremos intents que no tienen parámetros (útil para detectar frases simples de bienvenida o despedida) de los que sí los tienen (clase genérica `ContextIntent`). En ambos casos, un intent se define mediante un nombre, una lista de objetos de tipo `StructuredPhrase` y una frase con la que contesta el chatbot cuando se detecta el intent. En el caso de `ContextIntent`, esta frase puede tener "huecos" para incorporar los valores de los parámetros detectados (mediante el formato #<nombre-parametro>#).

La clase `ContextIntent` se parametriza con el tipo del objeto que se debe producir cuando se detecta el intent. Para ello, el método `withParameter` especifica cuándo se considera que un parámetro concuerda con un texto, y cómo obtener el valor del parámetro a partir del texto. Por ejemplo, la llamada:

```
withParameter("coffee-number",
              s -> s.matches("\\d+"), // matchea con un String que contiene dígitos
              s -> Integer.valueOf(s))// el valor del parámetro es el entero que hay en el String
```

especifica mediante expresiones lambda que el parámetro `coffee-number` hace match con una cadena que contiene dígitos, y que el valor del parámetro es el entero codificado en el `String` (en ambas expresiones, `s` es de tipo `String`).

Adicionalmente el método `resultObject` especifica cómo crear el objeto resultado a partir de la información recabada de la frase. Por ejemplo, la llamada:

```
resultObject((ContextIntent<CoffeeOrder> c) ->
                    new CoffeeOrder(c.<Integer>getParam("coffee-number"),
                                    c.<CoffeeType>getParam("coffee-type")))
```

especifica que el objeto `CoffeeOrder` se crea obteniendo los dos parámetros del intent (`coffee-number` y `coffee-type`) e invocando al constructor. Esto se ha realizado mediante una expresión lambda que recibe el propio `ContextIntent` como argumento, para poder llamar a `getParam` y recuperar el valor de los parámetros del intent. El método `getParam` es genérico, para poder convertir adecuadamente los valores almacenados al tipo correspondiente.

El siguiente programa ilustra el uso de los intents. El método `match` devuelve si el intent concuerda o no con la frase, y el método `process`, procesa la frase, extrayendo el valor de los parámetros (y devolviendo de nuevo el intent). El método `getReply` devuelve la frase de respuesta del intent, incorporando los valores de los parámetros detectados, si es necesario. Finalmente, el método `getObject` de `ContextIntent` devuelve el objeto resultado.

```
public class IntentTest {
  public static void main(String[] args) {
       Intent greeting = greetingIntent();
       ContextIntent<CoffeeOrder> order = orderIntent();

       System.out.println(greeting.matches("Hello"));
       System.out.println(greeting.process("Hello").getReply());
       System.out.println(order.matches("I'd like a capuccino"));
       System.out.println(order.process("I'd like a capuccino").getReply());
       CoffeeOrder coffeeOrder = order.getObject();
       System.out.println(coffeeOrder);
       System.out.println(order instanceof Intent);   // ContextIntents are Intents
  }
  public static ContextIntent<CoffeeOrder> orderIntent() {
       return new ContextIntent<CoffeeOrder>("Coffee Order",
                    List.of(TextInputMain.createPhrases()))
                 .withParameter("coffee-number",
                              s -> s.matches("\\d+"),
                              s -> Integer.valueOf(s))
                 .withParameter("coffee-type",
                              s -> IntentHelper.containsIgnoreCase(s, CoffeeType.values()),
                              s -> CoffeeType.valueOf(s.toUpperCase()))
                 .resultObject(c -> new CoffeeOrder(c.<Integer>getParam("coffee-number"),
                                                    c.<CoffeeType>getParam("coffee-type")))
                 .replies("All right, you ordered #coffee-number# #coffee-type#");
       }
  public static Intent greetingIntent() {
       return new Intent("Greetings",
                    List.of(new StructuredPhrase().with("Hello")))
              .replies("Welcome to Java Cafe, how can I help you?");
  }
}
```

Salida esperada:

```
true
Welcome to Java Cafe, how can I help you?
true
All right, you ordered 1 CAPUCCINO
CoffeeOrder[1, CAPUCCINO]
true
```

Donde la clases `IntentHelper` y `CoffeeOrder` están definidas como:

```java
public class IntentHelper { // clase de utilidad para la definición de Intents
  public static boolean containsIgnoreCase(String s, Object[] values) {
       return List.of(values).stream().anyMatch(e -> s.toUpperCase().equals(e.toString()));
  }
}

public class CoffeeOrder {
       private int num;
       private CoffeeType ct;

       public CoffeeOrder(int num, CoffeeType ct) {
              this.num = num;
              this.ct = ct;
       }
       @Override public String toString() { return "CoffeeOrder[" + num + ", " + this.ct +"]"; }
}
```

Apartado 3: Chatbots (1.5 puntos)

A continuación, definiremos una clase `Chatbot`, que agregue intents. La clase será genérica, y se parametrizará con el tipo de objeto que se debe producir como resultado de la interacción del usuario y el chatbot. La clase `chatbot` tendrá un constructor que recibe el nombre del chatbot, y se construirá llamando a dos métodos:

- `withIntent`: que recibe un intent que se añadirá al chatbot
- `withFallback`: que recibe como argumento la frase que contesta el chatbot cuando ningún intent concuerda con la frase del usuario

La clase chatbot tendrá dos métodos adicionales:

- `reactTo`: que recibe una frase del usuario, busca un intent que concuerde, y devuelve la frase de salida del intent, o bien la frase *fallback* si no concuerda ninguno.
- `getObject`: que devuelve el objeto resultado de la interacción.

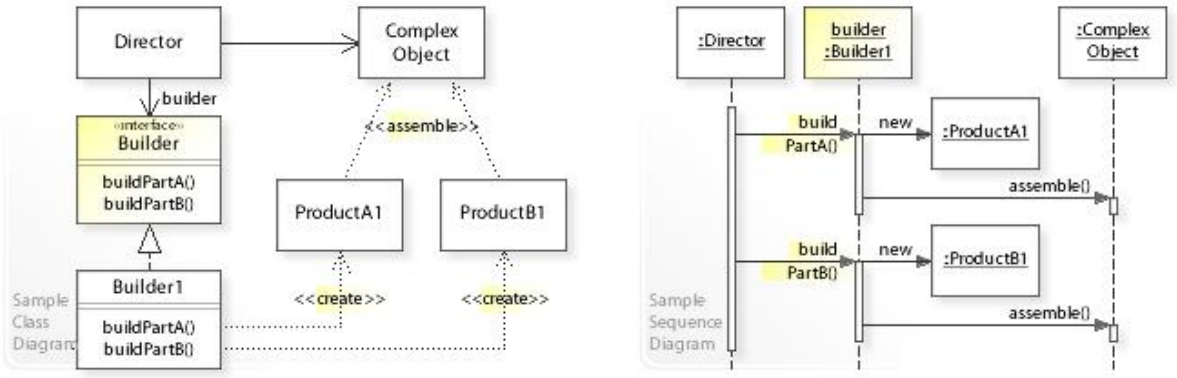
```java
public class ChatbotTest {
  public static void main(String[] args) {
    Chatbot<CoffeeOrder> coffeeShop = createChatbot();

    coffeeShop.reactTo("Hello")
              .reactTo("How are you?")
              .reactTo("I'd like a coffee");

    CoffeeOrder co = coffeeShop.getObject();
    System.out.println("Returned object: "+co);
  }
  public static Chatbot<CoffeeOrder> createChatbot() {
    Chatbot<CoffeeOrder> coffeeShop =
       new Chatbot<CoffeeOrder>("Java Cafe")
              .withIntent(IntentTest.greetingIntent())
              .withIntent(IntentTest.orderIntent())
              .withFallback("Sorry, I did not understand that, but I may help you ordering coffee");

       return coffeeShop;
  }
}
```

Salida esperada:

```
User> Hello
Java Cafe> Welcome to Java Cafe, how can I help you?
User> How are you?
Java Cafe> Sorry, I did not understand that, but I may help you ordering coffee
User> I'd like a coffee
Java Cafe> All right, you ordered 1 ESPRESSO
Returned object: CoffeeOrder[1, ESPRESSO]
```

Apartado 4: Plantillas para la creación de chatbots: el patrón *Builder* (2 puntos)

En este último apartado, crearemos clases que faciliten la creación de chatbots, y permitan definir plantillas, que añadan automáticamente intents con frases y respuestas predefinidas. Para ello, usaremos el patrón de diseño Builder, un patrón de creación cuyo esquema se muestra más abajo. El patrón define una interfaz con métodos para crear objetos (que puede ser una interfaz Java o una clase, quizá abstracta), así como un método `build` que devuelve el objeto creado. Mediante distintos tipos builders, podremos crear distintas variantes del objeto a crear, reutilizando el mismo programa.



El programa de más abajo usa esta idea para la creación de chatbots. La clase `ChatbotBuilder` define un builder básico, que simplemente permite crear un chatbot llamando a los métodos `withIntent` y `withFallback`. El método `build` devuelve el chatbot creado. La clase `ChitChatChatbotBuilder` es una plantilla que añade *intents* de bienvenida (que entiende frases como "*Hello*" o "*Hi!*"), de conversación informal "*chit-chat*" (que entiende frases como "*How are you?*") y de despedida (que entiende frases como "*Bye bye*"), así como una frase de fallback por defecto.

```
public class ChatbotBuilderTest {
  public static void main(String[] args) {
    Chatbot<CoffeeOrder> chatbot = createChatbot(new ChatbotBuilder<CoffeeOrder>("Java Cafe"),
                          "Sorry, I did not understand that, but I may help you ordering coffee.");
    doInteraction(chatbot);
    System.out.println("==============================");
    chatbot = createChatbot(new ChitChatChatbotBuilder<CoffeeOrder>("Java Cafe"), null);
    doInteraction(chatbot);
  }
  private static void doInteraction(Chatbot<CoffeeOrder> cb) {
       cb.reactTo("Hello")
         .reactTo("How are you?")
         .reactTo("I'd like a coffee")
         .reactTo("Bye bye!");
       System.out.println("Returned object: "+ cb.getObject());
  }
  private static Chatbot<CoffeeOrder> createChatbot(ChatbotBuilder<CoffeeOrder> builder,
                                                    String fallBack)
  {
       builder.withIntent(IntentTest.orderIntent());
       if (fallBack != null) builder.withFallback(fallBack);
       return builder.build();
  }
}
```

Salida esperada

```
User> Hello
Java Cafe> Sorry, I did not understand that, but I may help you ordering coffee.
User> How are you?
Java Cafe> Sorry, I did not understand that, but I may help you ordering coffee.
User> I'd like a coffee
Java Cafe> All right, you ordered 1 ESPRESSO
User> Bye bye!
Java Cafe> Sorry, I did not understand that, but I may help you ordering coffee.
Returned object: CoffeeOrder[1, ESPRESSO]
==============================
User> Hello
Java Cafe> Welcome to Java Cafe, how can I help you?
User> How are you?
Java Cafe> I'm good, what can I do for you today?
User> I'd like a coffee
Java Cafe> All right, you ordered 1 ESPRESSO
User> Bye bye!
Java Cafe> Thank you, please call again!
Returned object: CoffeeOrder[1, ESPRESSO]
```

Normas de Entrega. Se deberá entregar:

- Un directorio **src** con el código Java de todos los apartados, incluidos los datos de prueba y **testers adicionales** que hayas desarrollado en los apartados que lo requieren (puedes usar JUnit).
- Un directorio **doc** con la documentación generada.
- Una **memoria** en formato **PDF** con una pequeña descripción de las decisiones del diseño adoptadas para cada apartado, y **con el diagrama de clases** de tu diseño.

Se debe entregar un único fichero ZIP con todo lo solicitado, que deberá llamarse de la siguiente manera: GR<numero_grupo>_<nombre_estudiantes>.zip. Por ejemplo Marisa y Pedro, del grupo 2213, entregarían el fichero: GR2213_MarisaPedro.zip, de manera que cuando se extraiga haya una carpeta con el mismo nombre, y dentro de la misma, el directorio src/, el doc/ y el PDF. El **incumplimiento** de la entrega en este formato supondrá una **penalización en la nota de la práctica**.