Jose Enrico B. Tiongson

TItle: ElegantSMS Framework / DragonSMS Application

Document: Project Proposal

ElegantSMS is a support annotation-based framework for easy querying of SMS-formatted messages, inspired by the querying style of the Spring-Hibernate framework. The project proposal will make use of Java 1.8 with an annotations processor, to keep SMS parsing easy and elegant for Java developers.

1. Annotation API

|  |  |
| --- | --- |
| **@SmsQuery** | |
| The core functionality of the ElegantSMS framework. Decorates a method with special format for SMS message dispatching. The order of arguments will determine the binding.  Important: All methods annotated by @SmsQuery must return a String.   1. <IDENTIFIER>   Denotes an identifier to a String variable that can be bounded. Delimited by the next non-whitespace token found after the pattern. If the next non-space token is a variable, this is delimited by whitespace.   1. <ARRAY...>   Denotes an identifier to a String[] array that can be bounded. Collects all Strings just before the non-whitespace token found after the pattern. If the next non-space token is a variable, this collects all delimited Strings until the end of the message. Delimited by what is provided by the @ArrayDelim annotation (see @ArrayDelim).   1. Literal   A trimmed String literal that is not an identifier or an array. Note that if the String literal is affixed with a space that is started with/ended with a word character (\\w), then the affixed space will not be stripped from the literal. This is case-insensitive by default. See @CaseSensitive annotation for the alternative. | |
| Retention | RetentionPolicy.RUNTIME |
| Target | ElementType.METHOD |
| Parameters | String value |
| @SmsQuery(**"REGISTER <NAME> <AGE>"**) *// accepts single words* String register(String name, String age); *// e.g. “REGISTER john 10”*  @SmsQuery(**"REGISTER <NAME> / <AGE>"**) *// accepts multiple words before slash* String register(String name, String age); *// e.g. “REGISTER john cruz / 10”*  @SmsQuery(**"REGISTER <NAME>"**) *// accepts only one word* String register(String name); *// e.g. not accept “REGISTER john cruz”* @SmsQuery(**"REGISTER <NAME>\n"**) *// accepts a whole line* String register(String name); *// e.g. accepts “REGISTER john cruz”*  *// get all parameters as an array split by a whitespace*  @SmsQuery(**"COMMAND <PARAMS...>"**) String command(String... params); | |

|  |  |
| --- | --- |
| **@ArrayDelim** | |
| Sets the delimiter of a parameter of an @SmsQuery method for splitting an <ARRAY...> parameter. Delimiters can be a regex expression. Uses whitespace(“\\s+”) by default. Empty delimiter will capture everything until the end of the message. | |
| Retention | RetentionPolicy.RUNTIME |
| Target | ElementType.PARAMETER |
| Parameters | String value |
| *// get all parameters as an array split by a whitespace* @SmsQuery(**"COMMAND <PARAMS...>"**) String command(@ArrayDelim(**"\\s+"**) String... params);  *// No need to bind delimiter because implicit from pattern* @SmsQuery(**"REGISTER <NAME> / <AGE> / <TELEPHONE NUMBER>"**) String command(String name, String age, String telephoneNumber);  *// Delimiter can sometimes be useful for smart splits* *// example: accepts “REGISTER Mark Jacobs / 20 / 09xx-xxxx-xxxx”*  *// name = {“Mark”, “Jacobs”}*  *// age = 20*  *// telephoneNumber = {“09xx”, “xxxx”, “xxxx”}* @SmsQuery(**"REGISTER <NAME...> / <AGE> / <TELEPHONE NUMBER...>"**) String command(@ArrayDelim(**"\\s+"**) String[] name, *// sep. by sapce* int age, *// gets age until a slash* @ArrayDelim(**"-"**) String[] telephoneNumber); *// sep. by dash  // get all parameters as an array split by slash surrounded by arbitrary whitespace*  *// example: accepts “REGISTER Mark Jacobs / 20 / 09xx-xxxx-xxxx*  *// params = {“Mark Jacobs”, “20”, “09xx-xxxx-xxxx”}* @SmsQuery(**"REGISTER <PARAMS...>"**) String command(@ArrayDelim(**"\\s\*/\\s\*"**) String... params); | |

|  |  |
| --- | --- |
| **@CaseSensitive** | |
| Marks if a query literals should be processed case-sensitive. | |
| Retention | RetentionPolicy.RUNTIME |
| Target | ElementType.METHOD |
| Parameters | int value |
| @CaseSensitive @SmsQuery(**"GO <ROOM#>”)** String go(String room); *// will not accept "go <ROOM#>"* | |

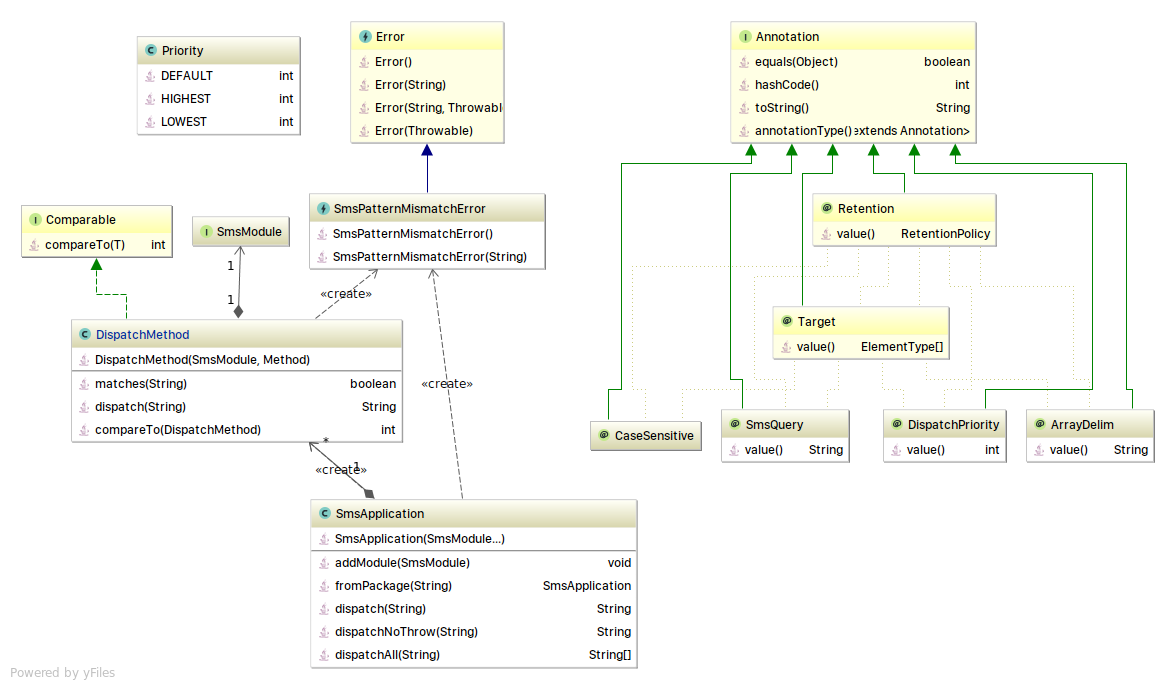
|  |  |
| --- | --- |
| **@DispatchPriority** | |
| Sets a priority order to a class or query method for when messages are dispatched. Useful for handling fallback queries. The higher the number, the higher the priority. Default value is Priority.DEFAULT == Integer.MIN\_VALUE / 2.  You may also annotate classes. Higher priority classes will dispatch first before lower priority classes. For tied class priorities, higher priority methods will dispatch first before lower priority methods. Tied method priorities will dispatched with a non-deterministic order.  Possible priorities are PriorityLOWEST, Priority.DEFAULT, and Priority.HIGHEST, or any int. | |
| Retention | RetentionPolicy.RUNTIME |
| Target | ElementType.TYPE, ElementType.METHOD |
| Parameters | int value |
| @DispatchPriority(Priority.***DEFAULT***) @SmsQuery(**"GO <ROOM>"**) String go(String room); *//makes sure to dispatch this first before fallbacks* @DispatchPriority(Priority.***LOWEST***) @SmsQuery(**"<COMMAND> <PARAMS...>"**) String fallback(String command, String... params); *// dispatch on fallback* | |

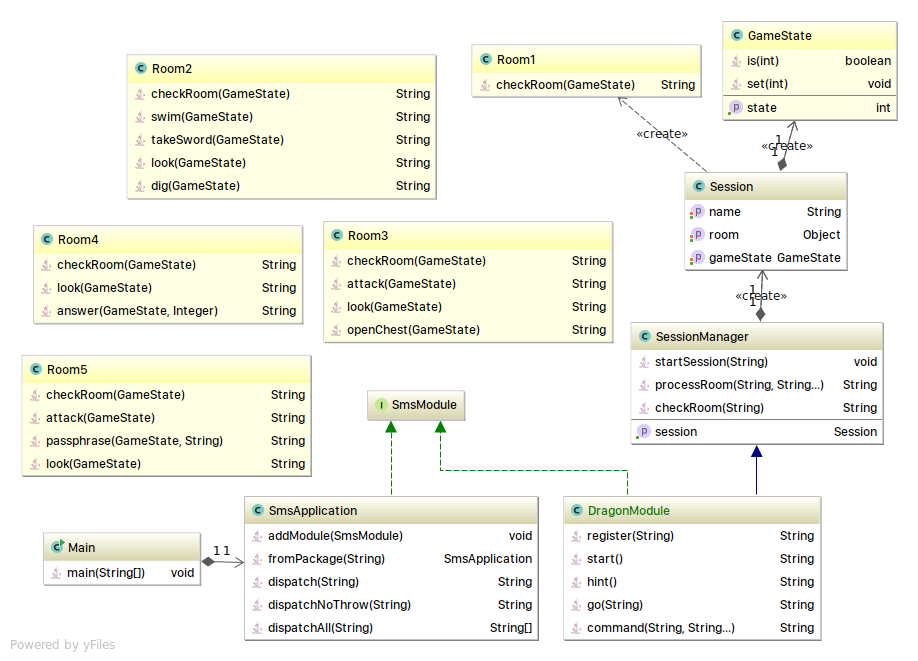
1. Technologies and External Libraries

|  |  |
| --- | --- |
| **Fast Classpath Scanner** | |
| Version | fast-classpath-scanner-2.0.17 |
| Description | A lightweight library optimized for scanning classpaths/packages in JVM. |
| Usage | This is used to load SMS Modules via package names (see SmsApplication#fromPackage) |

|  |  |
| --- | --- |
| **Objenesis** | |
| Version | objenesis-tck-2.5.1 |
| Description | An empty object instantiator library. |
| Usage | This is useful for cases SMS Modules have no default constructors. Since we plan for the SMS Application to instantiate modules dynamically through the default constructor, there will be issues when modules cannot be instantiated via the default Java reflections’ newInstance method. We use Objenesis to handle reflective construction for modules without default constructors, for seamless framework integration. |

1. Diagrams
2. ElegantSMS Framework UML Diagram

B. DragonSMS Application UML Diagram



1. Code Examples: DragonSMS with ElegantSMS

*// Example: DragonModule.java*  
**public class** DragonModule **implements** SmsModule {  
 **private String sessionId**; *// stores the name of user in the session* @SmsQuery(**"REGISTER <NAME>\n"**) *// end with “\n” to get full name*  
 **public String** register(**String** name) {  
 **this**.**sessionId** = name;  
 **return "Hello, "** + name + **", welcome to DragonSMS"**;  
 }  
  
 @SmsQuery(**"START"**)  
 **public String** start() {  
 startSession(**sessionId**); *// to be implemented* **return** this.go(**"Room1"**);}  
  
 @SmsQuery(**"GO <ROOM#>"**)  
 **public String** go(**String** roomName) {  
 **return** checkRoom(roomName); *// to be implemented*  
 }  
@DispatchPriority(Priority.***LOWEST***)  
 @SmsQuery(**"<COMMAND> <PARAMS...>"**)  
 **public String** command(**String** command, @ArrayDelim(**"\\s+"**) **String**... params){  
 **return** processRoom(command, params); *// to be implemented*  
 }

@SmsQuery(**"HINT"**)  
 **public String** hint() {  
 **return ""**; *// to be implemented*}  
  
}

*// Example: Main.java*

**public class** Main {  
  
 *// Load SMS application from module* **static SmsApplication** *dragonSMS* = **new** SmsApplication(**new** DragonModule());

*// Alternative: load application from package name  
 // static SmsApplication dragonSMS= SmsApplication.fromPackage("app.module");*

*// Driver program* **public static void** main(String[] args) {  
 **Scanner** sc = **new** Scanner(System.***in***);  
 **while** (sc.hasNextLine()) {  
 **try** {

*// Dispatch messages by using SmsApplication#dispatch*  
 **String** reply = *dragonSMS*.dispatch(sc.nextLine());  
 System.***out***.println(reply);  
 } **catch** (**SmsPatternMismatchException** e) {  
 e.printStackTrace();  
 System.***out***.println(**"invalid command"**);  
 }  
 }  
 }  
}