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.

I. Annotation API

@SmsQuery	
The core functionality of the ElegantSMS framework. Decorates a method with special format for SMS message dispatching. Binding identifiers will be done per order. 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. **2.** <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). **3.** 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”
```

@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
@SmsQuery("REGISTER <NAME...> / <AGE> / <TELEPHONE NUMBER...>")
String command(@ArrayDelim("\\s+") String[] name,        // sep. by sapce
            String 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
@SmsQuery("REGISTER <PARAMS...>")
String command(@BindDelim("\\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.

Higher priority classes will dispatch first before lower priority classes. For tied priorities, higher priority methods will dispatch first before lower priority methods. Tied priorities will give a non-deterministic order.

Possible priorities are Priority.LOWEST, Priority.DEFAULT, and Priority.HIGHEST.

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
```

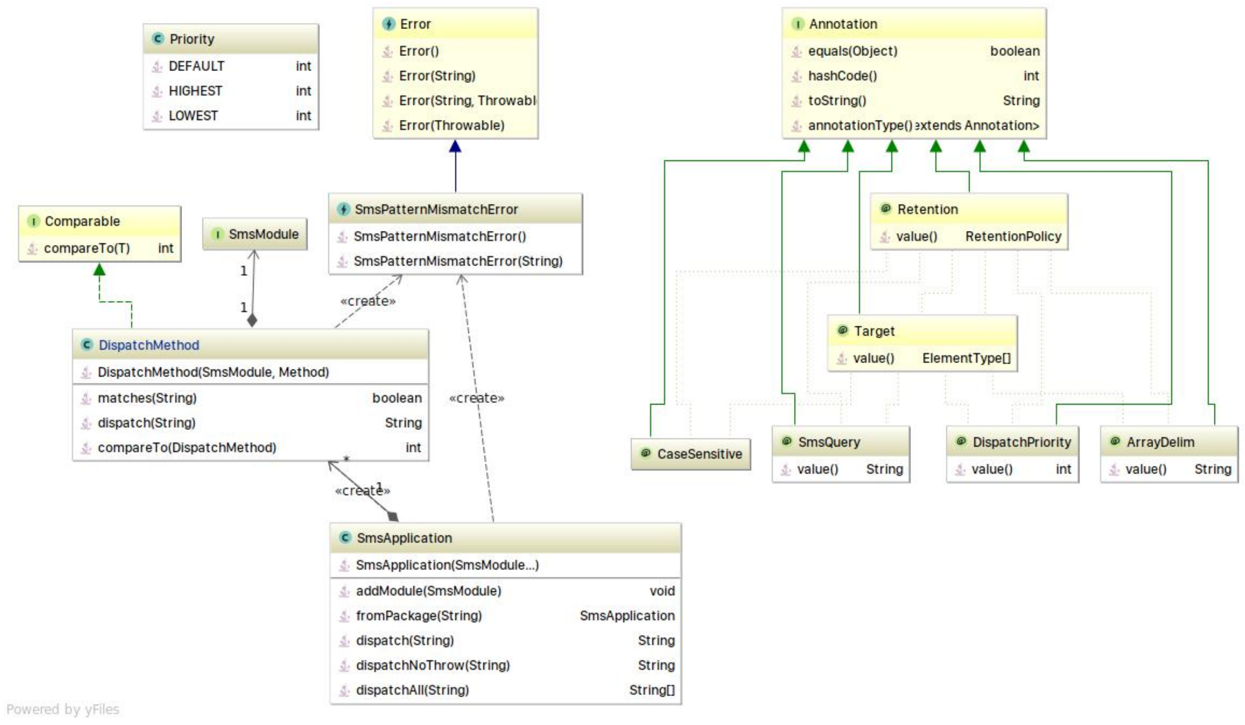
II. Technologies and External Libraries

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)

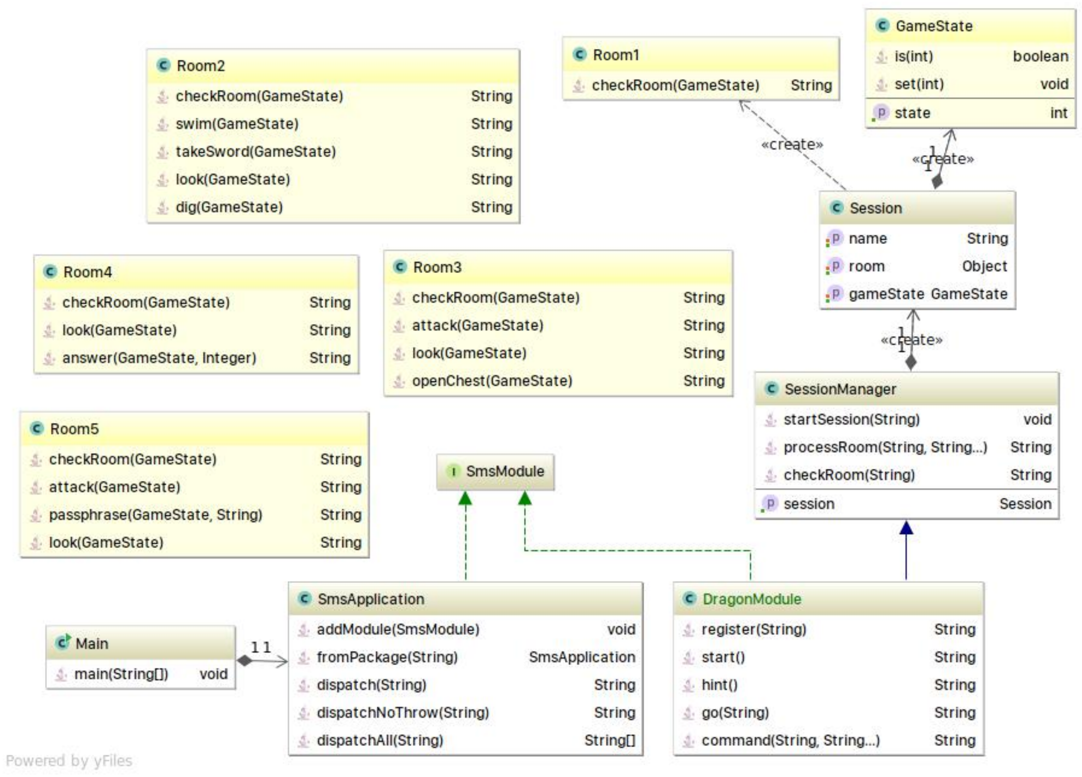
Objenesis	
Version	objenesis-tck-2.5.1
Description	An empty object instantiator library.
Usage	This is useful for cases SMS Modules have no default constructors. Since we plan for the SMS Application to instantiate modules dynamically through the default constructor, there will be issues when modules 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.

III. Diagrams

A. ElegantSMS Framework UML Diagram



B. DragonSMS Application UML Diagram



IV. Code Examples: DragonSMS with ElegantSMS

```java
// Example: DragonModule.java
public class DragonModule extends SessionManager 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 + 1)
   @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
   }

}
```

```java
// Example: Main.java
public class Main {

    // Load application from package
    static SmsApplication dragonSMS = new SmsApplication(new DragonModule());

    // Driver program
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        while (sc.hasNextLine()) {
            try {
                String reply = dragonSMS.dispatch(sc.nextLine());
                System.out.println(reply);
            } catch (SmsPatternMismatchException e) {
                e.printStackTrace();
                System.out.println("invalid command");
            }
        }
    }

}
```