**ПРИЛОЖЕНИЕ А**

(Обязательное)

**Листинг классов**

**package** org.telegram;  
  
**import** org.telegram.telegrambots.ApiContextInitializer;  
**import** org.telegram.telegrambots.TelegramBotsApi;  
**import** org.telegram.telegrambots.exceptions.TelegramApiException;  
**import** org.telegram.telegrambots.logging.BotLogger;  
**import** org.telegram.telegrambots.logging.BotsFileHandler;  
**import** org.telegram.updateshandlers.\*;  
  
**import** java.io.IOException;  
**import** java.util.logging.ConsoleHandler;  
**import** java.util.logging.Level;  
  
*/\*\*  
 \* TelegramBot class to create all bots  
 \*/***public class** TelegramBot {  
 **private static final** String ***LOGTAG*** = **"MAIN"**;  
  
 **public static void** main(String[] args) {  
 BotLogger.*setLevel*(Level.***ALL***);  
 BotLogger.*registerLogger*(**new** ConsoleHandler());  
 **try** {  
 BotLogger.*registerLogger*(**new** BotsFileHandler());  
 } **catch** (IOException e) {  
 BotLogger.*severe*(***LOGTAG***, e);  
 }  
  
 **try** {  
 ApiContextInitializer.*init*();  
 TelegramBotsApi telegramBotsApi = *createTelegramBotsApi*();  
 **try** {  
 telegramBotsApi.registerBot(**new** GomelGuideHandlers());  
 } **catch** (TelegramApiException e) {  
 BotLogger.*error*(***LOGTAG***, e);  
 }  
 } **catch** (Exception e) {  
 BotLogger.*error*(***LOGTAG***, e);  
 }  
 }  
  
 **private static** TelegramBotsApi createTelegramBotsApi() **throws** TelegramApiException {  
 TelegramBotsApi telegramBotsApi;  
 telegramBotsApi = createLongPollingTelegramBotsApi();  
 **return** telegramBotsApi;  
 }  
  
 */\*\*  
 \** ***@brief*** *Creates a Telegram Bots Api to use Long Polling (getUpdates) bots.  
 \** ***@return*** *TelegramBotsApi to register the bots.  
 \*/* **private static** TelegramBotsApi createLongPollingTelegramBotsApi() {  
 **return new** TelegramBotsApi();  
 }  
  
}

**package** org.telegram.updateshandlers;  
  
**import** org.telegram.BotConfig;  
**import** org.telegram.services.\*;  
**import** org.telegram.telegrambots.api.methods.send.SendMessage;  
**import** org.telegram.telegrambots.api.objects.Message;  
**import** org.telegram.telegrambots.api.objects.Update;  
**import** org.telegram.telegrambots.bots.TelegramLongPollingBot;  
**import** org.telegram.telegrambots.exceptions.TelegramApiException;  
  
**public class** GomelGuideHandlers **extends** TelegramLongPollingBot {  
 **private static final** String ***LOGTAG*** = **"GOMELGUIDEHANDLERS"**;  
  
 @Override  
 **public** String getBotToken() {  
 **return** BotConfig.***GOMELGUIDE\_TOKEN***;  
 }  
  
 @Override  
 **public** String getBotUsername() {  
 **return** BotConfig.***GOMELGUIDE\_USER***;  
 }  
  
 @Override  
 **public void** onUpdateReceived(Update update) {  
 Message message = update.getMessage();  
 **if** (message != **null** && message.hasText()) {  
 sendMsg(message, GomelGuideService.*getAnswer*(message.getText()));  
 }  
 }  
  
 **private void** sendMsg(Message message, String text) {  
 SendMessage sendMessage = **new** SendMessage();  
 sendMessage.enableMarkdown(**true**);  
 sendMessage.setChatId(message.getChatId().toString());  
 sendMessage.setText(text);  
 **try** {  
 sendMessage(sendMessage);  
 } **catch** (TelegramApiException e) {  
 e.printStackTrace();  
 }  
 }  
  
 **private void** replyToMsg(Message message, String text) {  
 SendMessage sendMessage = **new** SendMessage();  
 sendMessage.enableMarkdown(**true**);  
 sendMessage.setChatId(message.getChatId().toString());  
 sendMessage.setReplyToMessageId(message.getMessageId());  
 sendMessage.setText(text);  
 **try** {  
 sendMessage(sendMessage);  
 } **catch** (TelegramApiException e) {  
 e.printStackTrace();  
 }  
 }  
}

**package** org.telegram.services;  
  
**import** org.apache.http.HttpEntity;  
**import** org.apache.http.HttpResponse;  
**import** org.apache.http.NameValuePair;  
**import** org.apache.http.client.entity.UrlEncodedFormEntity;  
**import** org.apache.http.client.methods.HttpPost;  
**import** org.apache.http.impl.client.CloseableHttpClient;  
**import** org.apache.http.impl.client.HttpClients;  
**import** org.apache.http.message.BasicNameValuePair;  
**import** org.apache.http.util.EntityUtils;  
  
**import** javax.net.ssl.HttpsURLConnection;  
**import** java.io.\*;  
**import** java.net.MalformedURLException;  
**import** java.net.ProtocolException;  
**import** java.net.URL;  
**import** java.net.URLDecoder;  
**import** java.util.ArrayList;  
**import** java.util.List;  
  
**public class** GomelGuideService {  
  
 **private static final** String ***USER\_AGENT*** = **"Mozilla/5.0"**;  
  
 *// HTTP GET request* **public static** String sendGet(String parameter) {  
  
 String url = **"https://localhost:8080/chat/post"**;  
 **try** {  
 URL obj = **new** URL(url);  
 HttpsURLConnection con = (HttpsURLConnection) obj.openConnection();  
  
 *// optional default is GET* con.setRequestMethod(**"GET"**);  
  
 *//add request header* con.setRequestProperty(**"User-Agent"**, ***USER\_AGENT***);  
  
 **int** responseCode = con.getResponseCode();  
 System.***out***.println(**"\nSending 'GET' request to URL : "** + url);  
 System.***out***.println(**"Response Code : "** + responseCode);  
  
 BufferedReader in = **new** BufferedReader(  
 **new** InputStreamReader(con.getInputStream()));  
 String inputLine;  
 StringBuffer response = **new** StringBuffer();  
  
 **while** ((inputLine = in.readLine()) != **null**) {  
 response.append(inputLine);  
 }  
 in.close();  
  
 *//print result* System.***out***.println(response.toString());  
 **return** response.toString();  
 } **catch** (MalformedURLException e) {  
 e.printStackTrace();  
 } **catch** (ProtocolException e) {  
 e.printStackTrace();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return** parameter;  
 }  
  
 *// HTTP POST request* **public static** String getAnswer(String message) {  
 **try** {  
 CloseableHttpClient httpclient = HttpClients.*createDefault*();  
 HttpPost httppost = **new** HttpPost(**"http://localhost:8080/chat"**);  
  
 List<NameValuePair> params = **new** ArrayList();  
 params.add(**new** BasicNameValuePair(**"message"**, message));  
 httppost.setEntity(**new** UrlEncodedFormEntity(params, **"UTF-8"**));  
  
 HttpResponse response = httpclient.execute(httppost);  
 HttpEntity entity = response.getEntity();  
  
 **if** (entity != **null**) {  
 InputStream instream = entity.getContent();  
 **try** {  
 **return** URLDecoder.*decode*(EntityUtils.*toString*(entity), **"UTF-8"**);  
 } **finally** {  
 instream.close();  
 }  
 }  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 **return** message;  
 }  
}

**package** by.gstu.chatbot.core;  
  
**import** com.sun.jersey.api.client.Client;  
**import** com.sun.jersey.api.client.ClientResponse;  
**import** com.sun.jersey.api.client.WebResource;  
**import** org.apache.log4j.Logger;  
**import** org.json.simple.JSONArray;  
**import** org.json.simple.JSONObject;  
**import** org.json.simple.parser.JSONParser;  
**import** org.json.simple.parser.ParseException;  
  
**import** java.io.UnsupportedEncodingException;  
**import** java.net.URLEncoder;  
**import** java.util.\*;  
  
**public class** YandexSpellChecker {  
  
 **final static** Logger ***LOG*** = Logger.*getLogger*(YandexSpellChecker.**class**);  
  
 **private** Client **client**;  
  
 **public** YandexSpellChecker() {  
 **client** = Client.*create*();  
 }  
**public** String check(String text) {  
 **try** {  
 ClientResponse response = sendRequest(text);  
 JSONArray array = getJSON(response);  
 **if**(array.isEmpty()){  
 **return** text;  
 }  
 Map incorrectWords = getIncorrectWordsMap(array);  
 String correctedText = replaceIncorrectWords(text, incorrectWords);  
 **return** correctedText;  
 } **catch** (SpellCheckerException e) {  
 ***LOG***.debug(e.getMessage());  
 **return** text;  
 }  
 }  
  
 **private** JSONArray getJSON(**final** ClientResponse response) **throws** SpellCheckerException {  
 String output = response.getEntity(String.**class**);  
 JSONParser parser = **new** JSONParser();  
 Object obj;  
 **try** {  
 obj = parser.parse(output);  
 } **catch** (ParseException e) {  
 **throw new** SpellCheckerException(TypeError.***PARSING\_ERROR***, **"PARSING\_ERROR"**);  
 }  
  
 JSONArray array = (JSONArray) obj;  
 **if** (array == **null**) {  
 **throw new** SpellCheckerException(TypeError.***PARSING\_ERROR***, **"PARSING\_ERROR"**);  
 }  
 **return** array;  
 }  
  
 **private** String replaceIncorrectWords(**final** String text, **final** Map<String, List<String>> replacementWords) **throws** SpellCheckerException {  
 String correctText = text;  
 **for** (Map.Entry<String, List<String>> entry : replacementWords.entrySet()) {  
 **if**(!entry.getValue().isEmpty()) {  
 correctText = correctText.replace(entry.getKey(), entry.getValue().get(0));  
 }  
 }  
 **return** correctText;  
 }  
  
 **private** Map getIncorrectWordsMap(**final** JSONArray array) **throws** SpellCheckerException {  
 Map<String, List<String>> incorrectWords = **new** LinkedHashMap();  
 Iterator<Object> it = array.iterator();  
 **while** (it.hasNext()) {  
 JSONObject jo = (JSONObject) it.next();  
 String incorrectWord = (String) jo.get(**"word"**);  
 List<String> correctWords = getCorrectWords((JSONArray) jo.get(**"s"**));  
 incorrectWords.put(incorrectWord, correctWords);  
 }  
 **return** incorrectWords;  
 }  
  
 **private** List<String> getCorrectWords(**final** JSONArray array) **throws** SpellCheckerException {  
 List<String> correctWords = **new** ArrayList<>();Iterator<Object> sit = array.iterator();  
 **while** (sit.hasNext()) {  
 String v = (String) sit.next();  
 correctWords.add(v);  
 }  
 **return** correctWords;  
 }  
  
 **private** ClientResponse sendRequest(**final** String text) **throws** SpellCheckerException {  
 String encodedText;  
 **try** {  
 encodedText = URLEncoder.*encode*(text, **"UTF8"**);  
 } **catch** (UnsupportedEncodingException e) {  
 **throw new** SpellCheckerException(TypeError.***ENCODING\_ERROR***, **"ENCODING\_ERROR"**);  
 }  
  
 WebResource webResource = **client** .resource(**"http://speller.yandex.net/services/spellservice.json/checkText"** + **"?text="** + encodedText);  
  
 ClientResponse response = webResource.accept(**"application/json"**)  
 .get(ClientResponse.**class**);  
  
 **if** (response.getStatus() != 200) {  
 **throw new** SpellCheckerException(TypeError.***HTTP\_ERROR***, **"HTTP\_ERROR"**);  
 }  
 **return** response;  
 }  
}  
  
**class** SpellCheckerException **extends** Exception {  
 TypeError **result**;  
 **public** SpellCheckerException(**final** TypeError result, **final** String message) {  
 **super**(result.name());  
 }  
}  
  
**enum** TypeError {  
 ***ENCODING\_ERROR***,  
 ***PARSING\_ERROR***,  
 ***HTTP\_ERROR***;  
}

**package** by.gstu.chatbot.core.datalayer;  
  
**import** java.sql.\*;  
**import** java.util.ArrayList;  
**import** java.util.List;  
**import** java.util.Properties;  
  
**public class** DataBase {  
 *// init database constants* **private static final** String ***DATABASE\_DRIVER*** = **"com.mysql.jdbc.Driver"**;  
 **private static final** String ***DATABASE\_URL*** = **"jdbc:mysql://localhost:3306/gomel"**;  
 **private static final** String ***USERNAME*** = **"root"**;  
 **private static final** String ***PASSWORD*** = **"1234"**;  
 **private static final** String ***MAX\_POOL*** = **"250"**;  
  
 *// init connection object* **private** Connection **connection**;  
 *// init properties object* **private** Properties **properties**;  
  
 **public** DataBase() {  
 connect();  
 }  
  
 *// create properties* **private** Properties getProperties() {  
 **if** (**properties** == **null**) {  
 **properties** = **new** Properties();  
 **properties**.setProperty(**"user"**, ***USERNAME***);  
 **properties**.setProperty(**"password"**, ***PASSWORD***);  
 **properties**.setProperty(**"MaxPooledStatements"**, ***MAX\_POOL***);  
 }  
 **return properties**;  
 }  
  
 *// connect database* **public** Connection connect() {  
 **if** (**connection** == **null**) {  
 **try** {  
 Class.*forName*(***DATABASE\_DRIVER***);  
 **connection** = DriverManager.*getConnection*(***DATABASE\_URL***, getProperties());  
 } **catch** (ClassNotFoundException | SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 **return connection**;  
 }  
  
 *// disconnect database* **public void** disconnect() {  
 **if** (**connection** != **null**) {  
 **try** {  
 **connection**.close();  
 **connection** = **null**;  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
 }  
 }  
  
 **public** List<String> sentQuery(**final** String query) {  
 List<String> response = **new** ArrayList<>();  
 Statement statement = **null**;  
 **try** {  
 statement = **connection**.createStatement();  
 ResultSet resultSet = statement.executeQuery(query);  
 **while** (resultSet.next()) {  
 response.add(resultSet.getString(1));  
 }  
 statement.close();  
 } **catch** (SQLException e) {  
 e.printStackTrace();  
 }  
  
 **return** response;  
 }  
  
 @Override  
 **protected void** finalize() **throws** Throwable {  
 **this**.disconnect();  
 **super**.finalize();  
 }  
}

**package** by.gstu.chatbot.core.datalayer;  
  
  
**import** com.mysql.jdbc.StringUtils;  
  
**import** java.util.List;  
  
**public class** DataBaseService {  
  
 **private final static** String ***startKey*** = **"{"**;  
 **private final static** String ***finishKey*** = **"}"**;  
 **private static** DataBase *dataBase* = **new** DataBase();  
  
 **public static** String formatData(**final** String key) {  
 **final** List<String> data = *dataBase*.sentQuery(key);  
 **final** StringBuilder response = **new** StringBuilder();  
 **for** (**int** i = 0; i < data.size() - 1; i++) {  
 response.append(data.get(i)).append(**", "**);  
 }  
 response.append(data.get(data.size() - 1));  
 **return** response.toString();  
 }  
  
 **public static** String fillResponse(String response) {  
 **if** (!StringUtils.*isEmptyOrWhitespaceOnly*(response) && *isNeedFillFromDB*(response)) {  
 response = *replaceKey*(response);  
 }  
 **return** response;  
 }  
  
 **public static** String replaceKey(String response) {  
 **final** String key = *getKey*(response);  
 **return** response.replace(***startKey*** + key + ***finishKey***, *formatData*(key));  
 }  
  
 **private static boolean** isNeedFillFromDB(**final** String response) {**return** response.contains(***startKey***) && response.contains(***finishKey***);  
 }  
  
 **private static** String getKey(**final** String response) {  
 String keyword = response.substring(response.indexOf(***startKey***) + 1, response.indexOf(***finishKey***));  
 **return** keyword;  
 }  
}

**package** by.gstu.chatbot.controllers;  
  
@Controller  
@RequestMapping(**"/chat"**)  
**public class** ChatController {  
  
 **private final** ChatRepository **chatRepository**;  
 **private static** MyBot *bot*;  
 **private final** Map<DeferredResult<List<String>>, Integer> **chatRequests** =  
 **new** ConcurrentHashMap<DeferredResult<List<String>>, Integer>();  
 YandexSpellChecker **spellChecker** = **new** YandexSpellChecker();  
  
 @Autowired  
 **public** ChatController(ChatRepository chatRepository) {  
 **this**.**chatRepository** = chatRepository;  
 DataParser dp = **new** DataParser();  
 *bot* = **new** MyBot(**"0"**, dp);  
 **this**.**chatRepository**.addMessage(**"[bot]"** + *bot*.getMessage());  
 }  
  
 @RequestMapping(method = RequestMethod.***GET***)  
 @ResponseBody  
 **public** DeferredResult<List<String>> getMessages(@RequestParam **int** messageIndex) {  
  
 **final** DeferredResult<List<String>> deferredResult = **new** DeferredResult<>(**null**, Collections.*emptyList*());  
 **this**.**chatRequests**.put(deferredResult, messageIndex);  
  
 deferredResult.onCompletion(**new** Runnable() {  
 @Override  
 **public void** run() {  
 **chatRequests**.remove(deferredResult);  
 }  
 });  
  
 List<String> messages = **this**.**chatRepository**.getMessages(messageIndex);  
 **if** (!messages.isEmpty()) {  
 deferredResult.setResult(messages);  
 }  
  
 **return** deferredResult;  
 }  
  
 @RequestMapping(method = RequestMethod.***POST***)  
 @ResponseBody  
 **public void** postMessage(@RequestParam String message, HttpServletResponse rsp) {  
 **this**.**chatRepository**.addMessage(message);  
 String userMessage = **spellChecker**.check(message.substring(message.indexOf(**']'**) + 1));  
 String response = DataBaseService.*fillResponse*(*bot*.send(userMessage));  
 **if** (response.length() == 0) {  
 response = DataBaseService.*fillResponse*(*bot*.getMessage());  
 }  
 **this**.**chatRepository**.addMessage(**"[bot]"** + response);  
  
 **for** (Entry<DeferredResult<List<String>>, Integer> entry : **this**.**chatRequests**.entrySet()) {  
 List<String> messages = **this**.**chatRepository**.getMessages(entry.getValue());  
 entry.getKey().setResult(messages);  
 }  
  
 **try** {  
 PrintWriter out = rsp.getWriter();  
 out.println(URLEncoder.*encode*(response, **"UTF-8"**));  
 out.close();  
 } **catch** (IOException e) {  
 e.printStackTrace();  
 }  
 }  
@RequestMapping(**"/reload"**)  
 **public void** reload() {  
 (*bot*.setParser(**new** DataParser());  
 }  
}