Project 1

Qinzhi Peng

Task 1

**Screen shots**

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

**Code snippets** of computation of each hash

public String computeHashesHex(String text, String hashChoice) throws IOException {  
 String hash\_hex = "";  
 try {  
 // access MessageDigest class for SHA-265 or MD5  
 MessageDigest md = MessageDigest.*getInstance*(hashChoice);  
 // compute the digest and convert to hex  
 md.update(text.getBytes("UTF-8"));  
 hash\_hex = jakarta.xml.bind.DatatypeConverter.*printHexBinary*(md.digest());  
 } catch (NoSuchAlgorithmException e) {  
 System.*out*.println("No Algorithm available" + e);  
 }  
 return hash\_hex;  
}  
  
public String computeHashesBase64(String text, String hashChoice) throws IOException {  
 String hash\_base = "";  
 try {  
 // access MessageDigest class for SHA-265 or MD5  
 MessageDigest md = MessageDigest.*getInstance*(hashChoice);  
 // compute the digest and convert to base64  
 md.update(text.getBytes("UTF-8"));  
 hash\_base = jakarta.xml.bind.DatatypeConverter.*printBase64Binary*(md.digest());  
 } catch (NoSuchAlgorithmException e) {  
 System.*out*.println("No Algorithm available" + e);  
 }  
 return hash\_base;  
}

Task 2

**Screen shots**

input page

Graphical user interface, text, application, email

Description automatically generated

drop-down menu

Graphical user interface, text, application, email

Description automatically generated

output page for Collie and Whippet

A picture containing graphical user interface

Description automatically generatedA picture containing text, mammal, dog

Description automatically generated

**Code snippets**

scraping of intelligence

*/\*\*  
 \* Return a list of a dog information  
 \*  
 \** ***@param*** *searchTag The tag of the dog to be searched for.  
 \*/*public List<String> searchDogInfo(String searchTag) throws IOException {  
 searchTag = URLEncoder.*encode*(searchTag, "UTF-8");  
  
 // Create a String List to store information for selected dog  
 List<String> dogInfo = new ArrayList<String>();  
  
 // Create a URL for the dog information page to be screen scraped  
 String dogInfoUrl = "https://dogtime.com/dog-breeds/" + searchTag;  
  
 // Load a Document from a URL  
 Document doc = Jsoup.*connect*(dogInfoUrl).get();  
  
 // Get star ratings for Friendly and Intelligence  
 String firStar = getStar(doc, "All Around Friendliness");  
 String intelSubStar = getSubStar(doc, "Intelligence");  
 System.*out*.println("Friendly: " + firStar);  
 System.*out*.println("Intelligence: " + intelSubStar);  
  
 // Get vital stat for height, weight, and lifespan  
 String height = getStat(doc, "height");  
 String weight = getStat(doc, "weight");  
 String lifeSpan = getStat(doc, "life span");  
 System.*out*.println("Height: " + height);  
 System.*out*.println("Weight: " + weight);  
 System.*out*.println("Life Span: " + lifeSpan);  
  
 // Store information into dogInfo List  
 dogInfo.add(firStar);  
 dogInfo.add(intelSubStar);  
 dogInfo.add(height);  
 dogInfo.add(weight);  
 dogInfo.add(lifeSpan);  
  
 return dogInfo;  
}

/\*  
 \* Get dog sub star rating  
 \*  
 \* Arguments  
 \* @param Document  
 \* @return star string  
 \*/  
private String getSubStar(Document doc, String searchWord) {  
 // Find all elements by class "characteristic-title"  
 Elements contents = doc.getElementsByClass("characteristic-title");  
  
 String starClass = "";  
 for (Element content : contents) {  
 // Find star Tag by searchWord  
 if (content.text().equalsIgnoreCase(searchWord)) {  
 Elements subContent = content.parent().getElementsByClass("characteristic-star-block");  
 // Extract the class names  
 starClass = subContent.get(0).children().attr("class");  
 }  
 }  
 // Format the star string  
 String star = starClass.substring(starClass.length() - 1) + " Stars";  
 return star;  
}

scraping of weight

/\*  
 \* Get dog vital stat  
 \*  
 \* Arguments  
 \* @param Document  
 \* @return stat string  
 \*/  
private String getStat(Document doc, String searchWord) {  
 String stat = "";  
 Elements contents = doc.getElementsByClass("vital-stat-box");  
 for (Element content : contents) {  
 // Extract the text  
 if (content.text().toLowerCase().contains(searchWord.toLowerCase())) {  
 stat = content.text().substring(content.text().lastIndexOf(":") + 2);  
 }  
 }  
 return stat;  
}

api call for the picture, including choosing one at random.

*/\*\*  
 \* Return a URL of a dog image  
 \*  
 \** ***@param*** *searchTag The tag of the photo to be searched for.  
 \*/*public String doDogSearch(String searchTag) throws IOException {  
 searchTag = URLEncoder.*encode*(searchTag, "UTF-8");  
 String response = "";  
  
 // Create a URL for the Json page to be screen scraped  
 String dogJsonUrl = "https://dog.ceo/api/breed/" + searchTag + "/images";  
  
 // Fetch the page  
 response = fetch(dogJsonUrl);  
  
 // Convert json record to java object  
 DogPictures dogPictures = json2object(response);  
  
 // Get a URL from all URLs  
 String dogPictureUrl = chooseUrl(dogPictures);  
 System.*out*.println("pictureURL= " + dogPictureUrl);  
 return dogPictureUrl;  
}

// Create a DogPictures Object  
class DogPictures {  
 List<String> message = new ArrayList<String>();  
 String status = "";  
}  
  
/\*  
 \* Convert json record to java object  
 \*  
 \* Arguments  
 \* @param dogJsonUrl The Json String  
 \* @return The dogPicture object  
 \*/  
private DogPictures json2object(String response) throws IOException {  
 // Creating a Gson Object  
 Gson gson = new Gson();  
 DogPictures dogPictures = gson.fromJson(response, DogPictures.class);  
 return dogPictures;  
}  
  
/\*  
 \* Randomly choose a URL from all URLs for pictures of dogs  
 \*  
 \* Arguments  
 \* @param dogPicture The dogPictures object  
 \* @return A URL of a dog's picture chosen randomly  
 \*/  
private String chooseUrl(DogPictures dogPictures) {  
 Random rd = new Random();  
 int randomIndex = rd.nextInt(dogPictures.message.size());  
 String dogPictureUrl = dogPictures.message.get(randomIndex);  
 return dogPictureUrl;  
}  
  
/\*  
 \* Make an HTTP request to a given URL  
 \*  
 \* @param urlString The URL of the request  
 \* @return A string of the response from the HTTP GET. This is identical  
 \* to what would be returned from using curl on the command line.  
 \*/  
private String fetch(String urlString) {  
 String response = "";  
 try {  
 URL url = new URL(urlString);  
 /\*  
 \* Create an HttpURLConnection. This is useful for setting headers  
 \* and for getting the path of the resource that is returned (which  
 \* may be different than the URL above if redirected).  
 \* HttpsURLConnection (with an "s") can be used if required by the site.  
 \*/  
 HttpURLConnection connection = (HttpURLConnection) url.openConnection();  
 // Read all the text returned by the server  
 BufferedReader in = new BufferedReader(new InputStreamReader(connection.getInputStream(), "UTF-8"));  
 String str;  
 // Read each line of "in" until done, adding each to "response"  
 while ((str = in.readLine()) != null) {  
 // str is one line of text readLine() strips newline characters  
 response += str;  
 }  
 in.close();  
 } catch (IOException e) {  
 System.*out*.println("read mistake, an exception");  
 // Do something reasonable. This is left for students to do.  
 }  
 return response;  
}

Task 3

**Screen shots**

input page

Graphical user interface, text, application, email

Description automatically generated

(mobile)

Graphical user interface, application

Description automatically generated

output page (one vote)

Graphical user interface, text, application

Description automatically generated

(mobile)

Graphical user interface, application

Description automatically generated

results page

Graphical user interface, text, application, email

Description automatically generated

Graphical user interface, text, application, email

Description automatically generated

(mobile)

Graphical user interface, text, application

Description automatically generatedGraphical user interface, text, application

Description automatically generated

**Code snippets** from the Java code that produces the output page and the results page.

produces the output page

<body>  
 <h1>Distributed Systems Class Clicker</h1>  
 <% if (request.getAttribute("currentChoice") == null) { %>  
 <p>Submit your answer to the current question:</p>  
 <% } else { %>  
 <p>Your "<%= request.getAttribute("currentChoice")%>" response has been registered</p>  
 <p>Submit your answer to the next question:</p>  
 <% } %>  
 <form action="submit" method="post">  
 <input type="radio" name="choice" id="A" value="A"/><label for="A"> A </label><br/>  
 <input type="radio" name="choice" id="B" value="B"/><label for="B"> B </label><br/>  
 <input type="radio" name="choice" id="C" value="C"/><label for="C"> C </label><br/>  
 <input type="radio" name="choice" id="D" value="D"/><label for="D"> D </label><br/><br/>  
 <input type="submit" value="Submit"/>  
 </form>  
</body>

produces the results page

<body>  
 <h1>Distributed Systems Class Clicker</h1>  
 <% if (request.getAttribute("countA") != null || (request.getAttribute("countB") != null) || (request.getAttribute("countC") != null) || (request.getAttribute("countD") != null)) { %>  
 <%--Show the result if it exists--%>  
 <p>The results from the survey are as follows:</p>  
 <% if (request.getAttribute("countA") != null) { %>  
 A : <%= request.getAttribute("countA")%><br/>  
 <% } %>  
 <% if (request.getAttribute("countB") != null) { %>  
 B : <%= request.getAttribute("countB")%><br/>  
 <% } %>  
 <% if (request.getAttribute("countC") != null) { %>  
 C : <%= request.getAttribute("countC")%><br/>  
 <% } %>  
 <% if (request.getAttribute("countD") != null) { %>  
 D : <%= request.getAttribute("countD")%><br/>  
 <% } %>  
 <p>These results has now been cleared.</p>  
 <%--If all count are zero then show no result --%>  
 <% } else { %>  
 <p>There are currently no results.</p>  
 <% } %>  
</body>