Projet Scala

Delers , Novaretti , Collas

Parse Arguments

- Création d'un constructeur de parser:
  - Bibliothèque qui facilite la création de parser pour les arguments dans la CLI
- Définition du parser :
  - Définit la sequence de param attendu (opt , arg)
- Actions sur les paramètres:
  - Chaque paramètre a une action associée qui est exécutée lorsqu'il est trouvé dans les arguments de la ligne de commande
- Description des paramètres:
  - Chaque paramètre a un texte descriptif qui explique son but.
- Exécution du parser:
  - la fonction exécute le parser avec OParser.parse(parser, args, Config())

```scala
def parseArguments(args: Array[String]): Option[Config] = {
  val builder = OParser.builder[Config]
  val parser = {
    import builder._
    OParser.sequence(
      programName("WikiStats"),
      opt[Int]('l', "limit")
        .action((value, config) => config.copy(limit = value))
        .text("Limit of the number of pages to retrieve"),
      arg[String]("<keyword>")
        .required()
        .action((value, config) => config.copy(keyword = value))
        .text("Keyword to search for")
    )
  }

  OParser.parse(parser, args, Config())
}
```

getPages

- Exécution de la requête HTTP :
  - Effectue une requête HTTP à l'URL fournie en utilisant la méthode Http(url).asString. Récupère la réponse sous forme de chaîne de caractères.

- Vérification du code de statut HTTP :
  - Vérifie le code de statut de la réponse HTTP. Si le code est 200 (requête a réussi), retourne le corps de la réponse dans un Right.

- Gestion des erreurs :
  - Si le code de statut n'est pas 200 (erreur lors de la requête), retourne le code de statut dans un Left.

```
def getPages(url: String): Either[Int, String] = {
  val result = Http(url).asString
  if (result.code == 200) Right(result.body)
  else Left(result.code)
}
```

```
def getPages(url: String)(implicit httpUtils: HttpUtils): Either[Int, String] = {
  val result = httpUtils.parse(url).asString
  if (result.code == 200) Right(result.body)
  else Left(result.code)
}
```

formatUrl

- Construction de l'URL de l'API Wikipedia :
  - Construit l'URL de l'API en utilisant une chaîne de caractères avec des interpolations de variables. Intègre le mot-clé recherché et la limite de résultats dans l'URL.

- Encodage du mot-clé :
  - Avant d'intégrer le mot-clé dans l'URL, la fonction l'encode en utilisant l'interpolation $keyword. Cela garantit que le mot-clé est correctement encodé pour être inclus dans une URL.

- Renvoi de l'URL formatée :

```
def formatUrl(keyword: String, limit: Int): String = {
  val encodedKeyword = s"https://en.wikipedia.org/w/api.php?action=query&format=json&prop=&sroffset=0&list=search&srsearch=$keyword&srlimit=$limit"
  encodedKeyword
}
```

parseJson

- Parsing du JSON brut :
  - Analyse le JSON brut en utilisant la méthode Json.parse. Cela convertit la chaîne de caractères JSON en une structure JSON utilisable.

- Extraction des pages JSON :
  - Extrait la partie du JSON contenant les informations sur les pages recherchées. Accede à la clé "query", puis à la clé "search". Obtient un JsArray contenant les objets JSON.

- Transformation des pages JSON en objets WikiPage :
  - Via map, Transforme chaque objet JSON en un objet WikiPage. Extrait le titre et le nombre de mots en utilisant l'opérateur \. Elle crée ensuite une instance de WikiPage avec ces informations.

- Renvoi de la séquence de WikiPage :

```
def parseJson(rawJson: String): Seq[WikiPage] = {
  val json = Json.parse(rawJson)
  val pagesJson = (json \ "query" \ "search").as[JsArray].value

  pagesJson.map(pageJson =>
    WikiPage(
      title = (pageJson \ "title").as[String],
      words = (pageJson \ "wordcount").as[Int]
    )
  )
}
```

totalWords

- Agrégation des mots totaux :
  - Utilise la méthode foldLeft pour agréger le nombre total de mots de toutes les pages. Commence à 0 et itère sur chaque page de la séquence pages. À chaque itération, elle ajoute le nombre de mots de la page courante au total.

- Renvoi du nombre total de mots :
  - Renvoie le nombre total de mots obtenus.

```
  def totalWords(pages: Seq[WikiPage]): Int = {
    pages.foldLeft(0)((total, page) => total + page.words)
  }
}
```

run

- Récupération de l'URL :
  - Créer l'URL en utilisant via formatUrl

- Exécution de la requête et traitement des résultats :
  - Utilise la fonction getPages pour effectuer la requête à l'API. En fonction du résultat retourné, elle effectue différentes actions :
    - Si la requête a réussi (Right), extrait les informations des pages. Elle affiche le nombre de pages trouvées, chaque page une par une, ainsi que des statistiques telles que le nombre total de mots et le nombre moyen de mots par page.

    - Si la requête a échoué (Left), elle affiche un message d'erreur contenant le code d'erreur.

```scala
def run(config: Config): Unit = {
  val url = formatUrl(config.keyword, config.limit)
  getPages(url) match {
    case Right(body) =>
      val pages = parseJson(body)
      println(s"Number of pages found: ${pages.length}")
      pages.foreach(page => println(page))

      val total = totalWords(pages)
      println(s"Total number of words across all pages: $total")

      val average = if (pages.nonEmpty) total.toDouble / pages.length else 0
      println(s"Average number of words per page: $average")
    case Left(errorCode) => println(s"An error occurred: $errorCode")
  }
}
```

```scala
import org.scalatest.flatspec.AnyFlatSpec
import scalaj.http.Http

object MockHttpUtils extends HttpUtils {
  override def parse(url: String): HttpRequest = {
    new HttpRequest {
      override def getStatusCode(url: String): Int = 200
    }
  }
}
```

```scala
class MainSpec extends AnyFlatSpec {
  "formatUrl" should "return the formatted URL" in {
    val keyword = "Scala"
    val limit = 10
    val expectedUrl = "https://en.wikipedia.org/w/api.php?action=query&format=json&prop=&sroffset=0&list=search&srsearch=Scala&srlimit=10"

    val actualUrl = Main.formatUrl(keyword, limit)

    assert(actualUrl == expectedUrl)
  }
```

```
"parseJson" should "return a list of WikiPage objects" in {
  val rawJson = "{\"query\": {\"search\": [{\"title\": \"Page 1\", \"wordcount\": 100}, {\"title\": \"Page 2\", \"wordcount\": 200}]}}"
  val expectedPages = Seq(WikiPage("Page 1", 100), WikiPage("Page 2", 200))

  val actualPages = Main.parseJson(rawJson)

  assert(actualPages == expectedPages)
}
```

```
"totalWords" should "return 0 for an empty list of pages" in {
  val pages = Seq.empty[WikiPage]

  val total = Main.totalWords(pages)

  assert(total == 0)
}

it should "return the total number of words for a non-empty list of pages" in {
  val pages = Seq(WikiPage("Page 1", 100), WikiPage("Page 2", 200))
  val expectedTotal = 300

  val total = Main.totalWords(pages)

  assert(total == expectedTotal)
}
```

```
"parseArguments" should "return Some(config) for valid arguments" in {
  val args = Array("--limit", "20", "Scala")
  val expectedConfig = Config(20, "Scala")

  val actualConfig = Main.parseArguments(args)

  assert(actualConfig.contains(expectedConfig))
}

it should "return None for invalid arguments" in {
  val args = Array("--limit", "abc", "Scala")

  val actualConfig = Main.parseArguments(args)

  assert(actualConfig.isEmpty)
}
```

```
"getPages" should "return Right with the page content for a successful HTTP request" in {
  val mockHttpUtils = new MockHttpUtils
  val url = "https://example.com/"
  val responseBody = "Page content"
  mockHttpUtils.setResponse(url, 200, responseBody)

  val result = Main.getPages(url)(mockHttpUtils)

  assert(result == Right(responseBody))
}

it should "return Left with the error code for a failed HTTP request" in {
  val mockHttpUtils = new MockHttpUtils
  val url = "https://example.com/"
  val errorCode = 404
  mockHttpUtils.setResponse(url, errorCode, "")

  val result = Main.getPages(url)(mockHttpUtils)

  assert(result == Left(errorCode))
}
```

All packages	43.28%
<empty>	43.28%

SCoverage generated at Tue Jun 20 11:34:38 CEST 2023

Lines of code:	92	Files:	1	Classes:	2	Methods:	8
Lines per file:	92.00	Packages:	1	Classes per package:	2.00	Methods per class:	4.00
Total statements:	67	Invoked statements:	29	Total branches:	4	Invoked branches:	0
Ignored statements:	0						
Statement coverage:	43.28 %			Branch coverage:	0.00 %		

Class	Source file	Lines	Methods	Statements	Invoked	Coverage		Branches	Invoked	Coverage	
HttpUtilsImpl	Main.scala	18	1	1	0		0.00 %	0	0		0.00 %
Main	Main.scala	92	7	66	29		43.94 %	4	0		0.00 %