

FACULDADE DE INFORMÁTICA E ADMINISTRAÇÃO PAULISTA FIAP

RELATÓRIO TÉCNICO DO PROJETO DE INTEGRAÇÃO COM O TWITTER

Trabalho final para o módulo de
Fundamentos da Tecnologia Java e
Modelagem UML 2.0 para o curso de MBA
em Desenvolvimento de Aplicações Java,
SOA e Internet das coisas

Professor: Michel Pereira Fernandes

MARCOS ANTONIO SOUZA PINHEIRO

RM: 30366

github: https://github.com/marcos-pinheiro/27SCJ_AtividadeFinalTwitter.git

SÃO PAULO
2016

Sumário

1. Bibliotecas.....	3
2. Organização do sistema.....	4
3. Diagramas de classe.....	6
4. Diagramas de sequencia.....	7
5. Screenshots.....	9

1 Bibliotecas

1. Twitter4J

Neste projeto foi utilizado a biblioteca Twitter4j. Esta biblioteca possui classes que abstrai funcionalidades nos quais executam chamadas HTTP a API do Twitter. Podemos citar funcionalidades tais como: “Tweetar”, obter tweets em um determinado período, obter retweets em um determinado período, etc.

1. Stream API

Neste projeto também foi utilizado a API de stream do Java 8. Esta API que cria um fluxo em coleções de dados aplicando funcionalidades tais como: filtros, transformações, coletas, redução e etc.

2. Java Concurrent API

A API de concorrência do Java fornece uma série de classes e métodos para manipular, executar e reciclar threads em aplicações concorrentes.

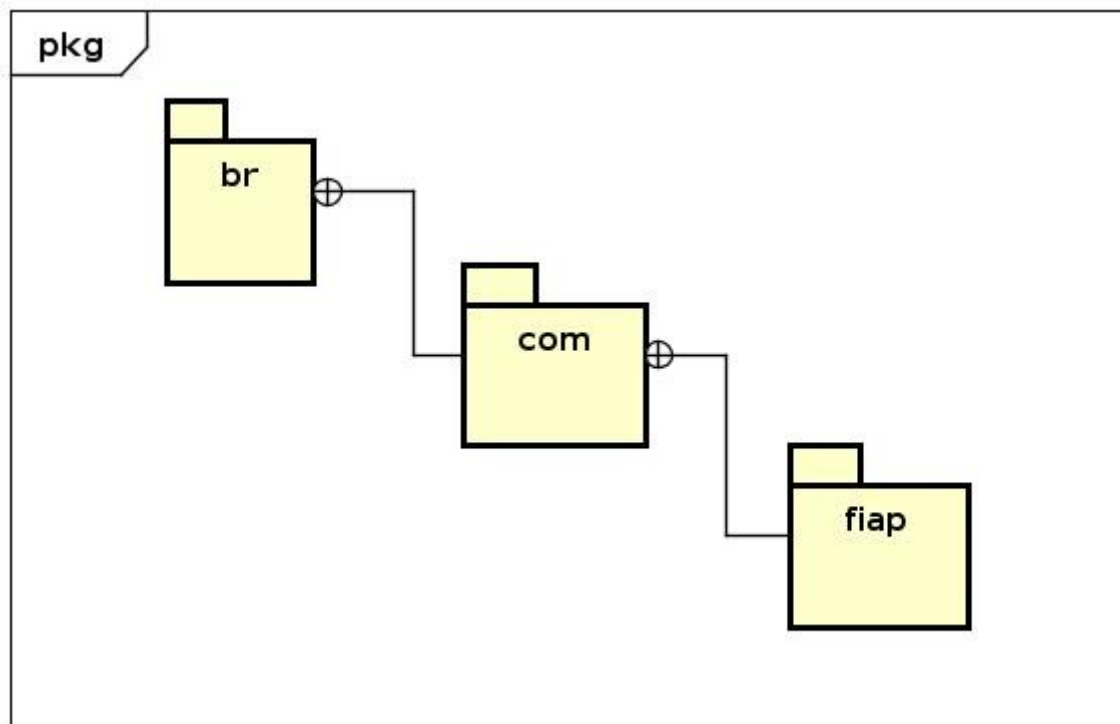
3. Collections API

Esta API fornece uma série de classes e interfaces para os mais variados tipos de estruturas de dados. Desde estruturas como Listas, Filas, Pilhas, coleções de conjuntos e chave-valor, threads-safe e non-threads-safe, entre outros.

2 Organização do sistema

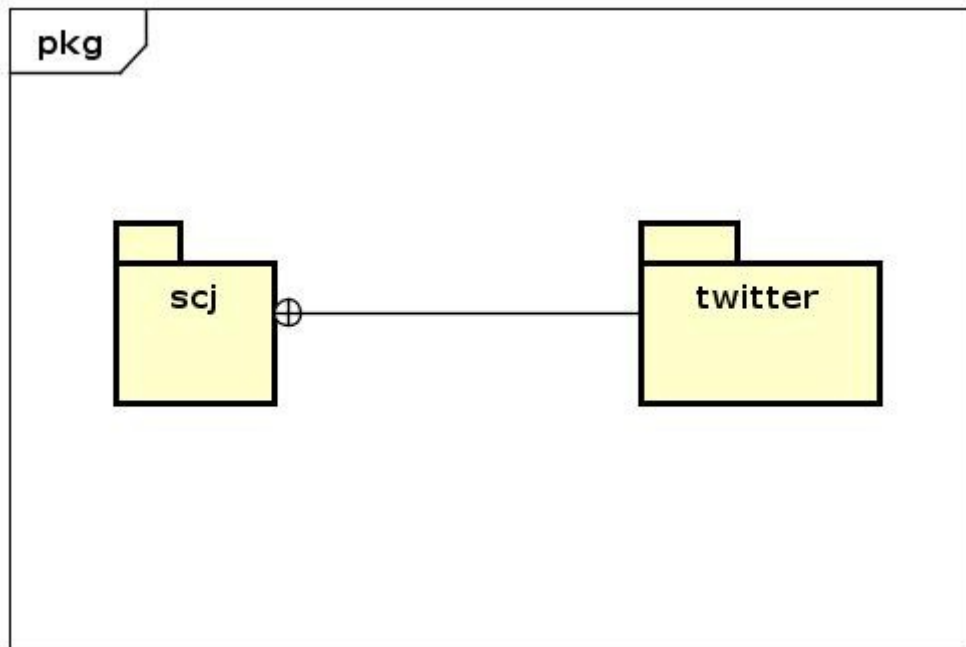
Para uma melhor organização das classes do projeto, em java utilizamos os pacotes/packages. Com os pacotes podemos agrupar um conjunto de classes que pertençam a um mesmo domínio ou objetivo.

Neste projeto organizamos nossos pacotes da seguinte maneira:

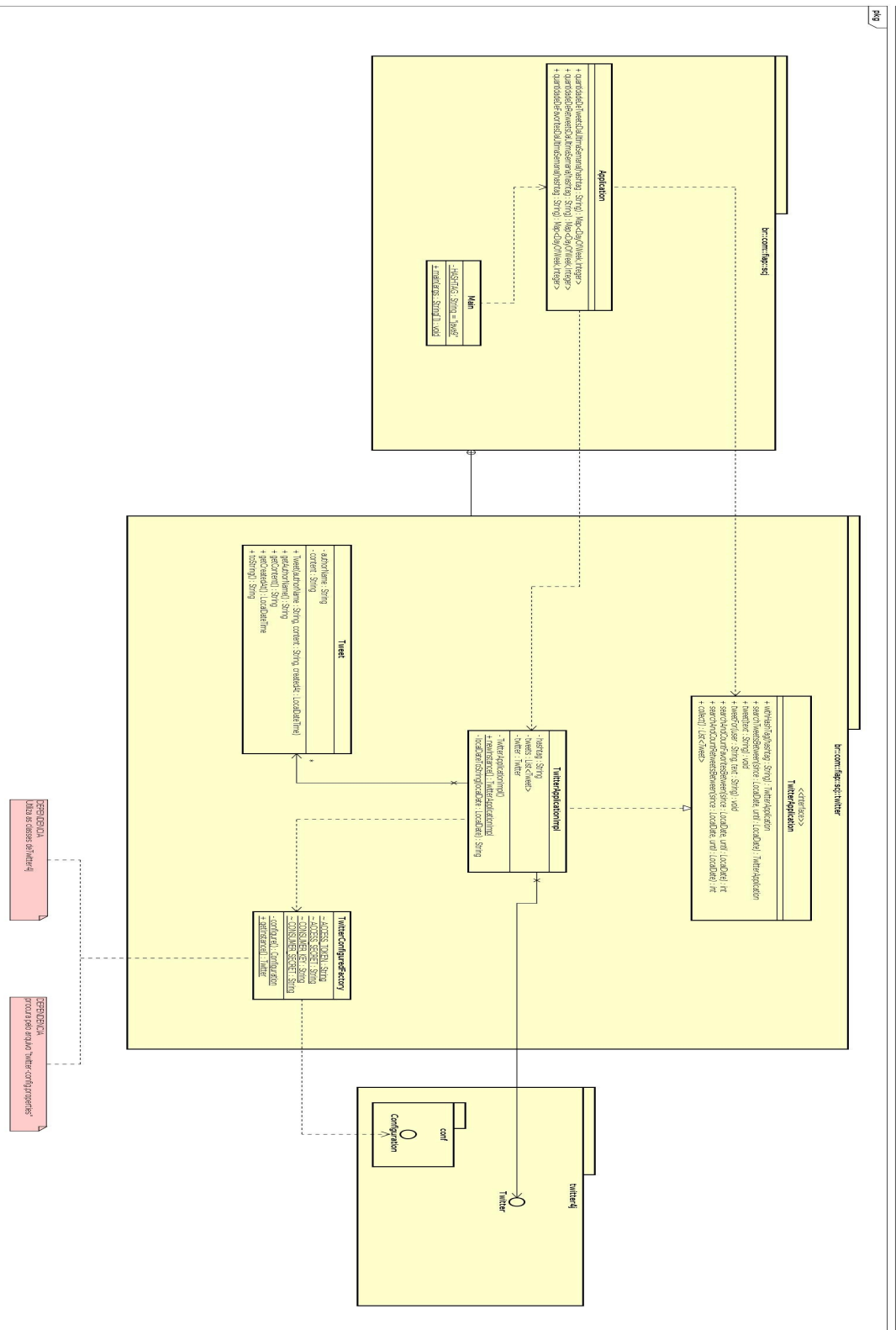


Por convenção, criamos nossos pacotes inicialmente com o domínio da aplicação ao contrario. No caso `br.com.fiap`

Após esta estrutura é definido o pacote “scj” onde ficarão as classes relacionadas a um contexto geral da aplicação, como por exemplo a classe do método main. E dentro deste pacote existe outro pacote chamado “twitter” no qual possui classes e interfaces que trabalham diretamente com a API do Twitter4J encapsulando as funcionalidades solicitadas neste projeto.



3 Diagramas de classe



OBS: Imagens e o projeto do diagrama acompanha o projeto no Github

4 Diagramas de sequencia

Diagrama de sequencia relacionado ao método quantidade de tweetsTest

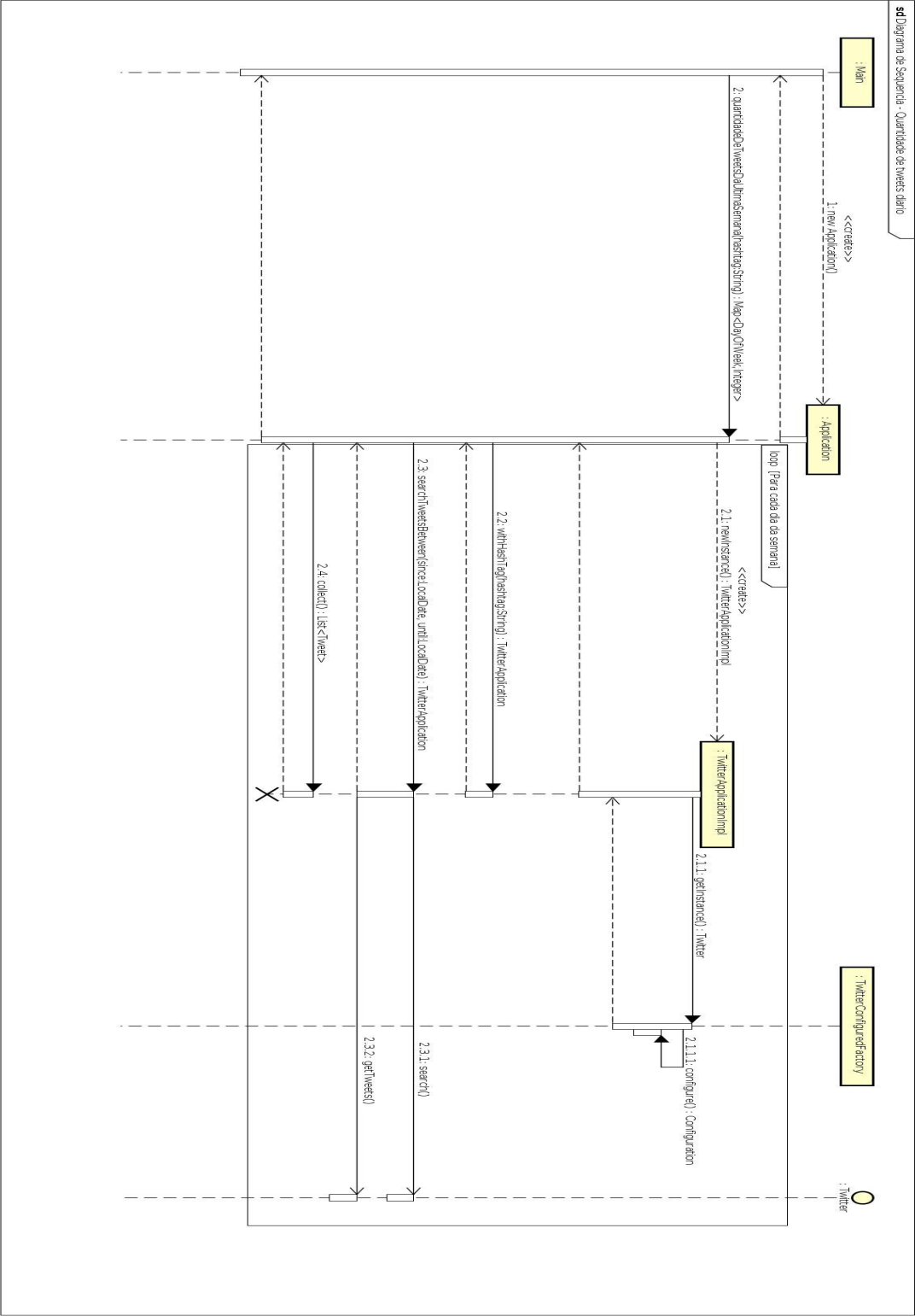
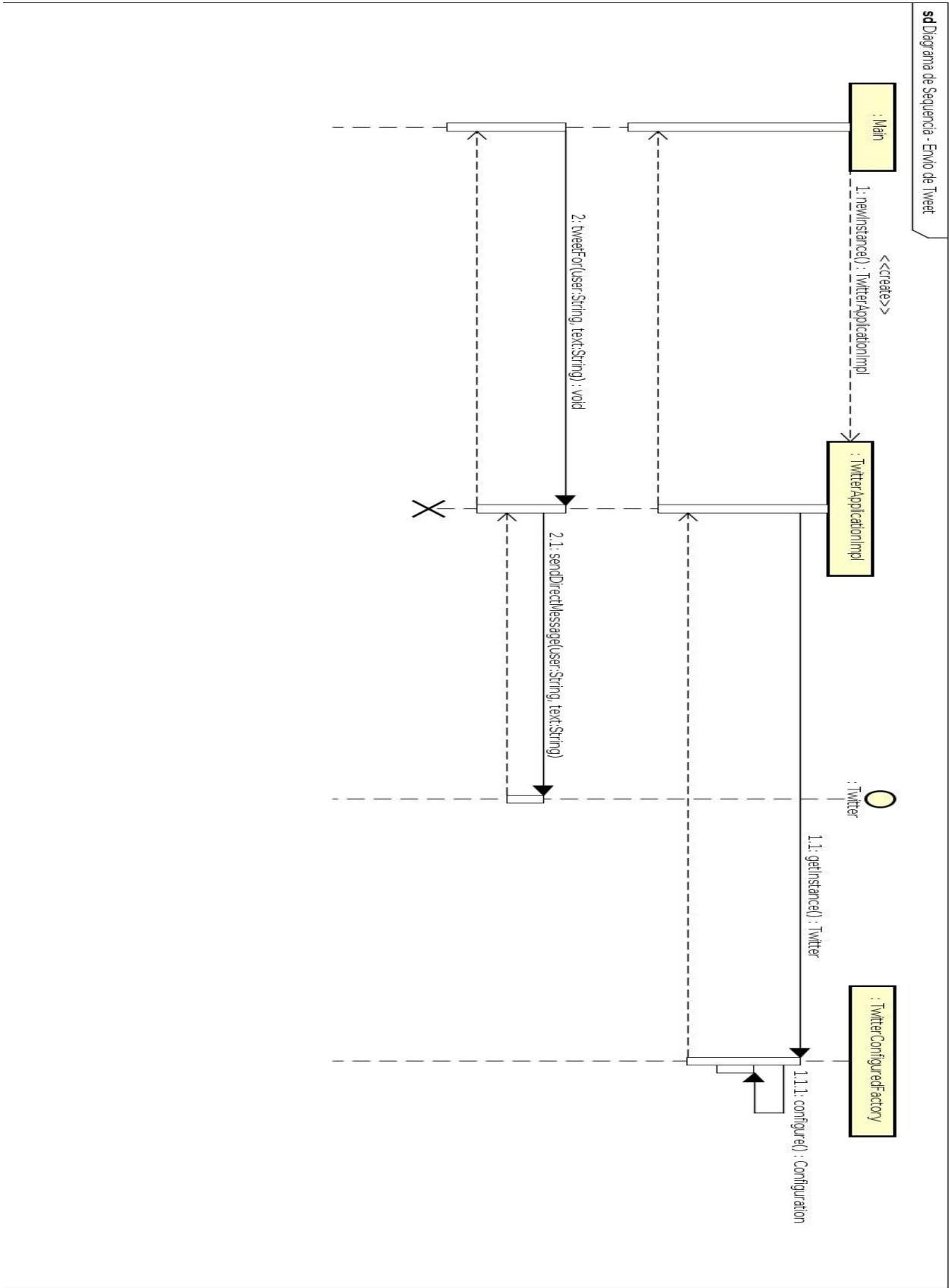
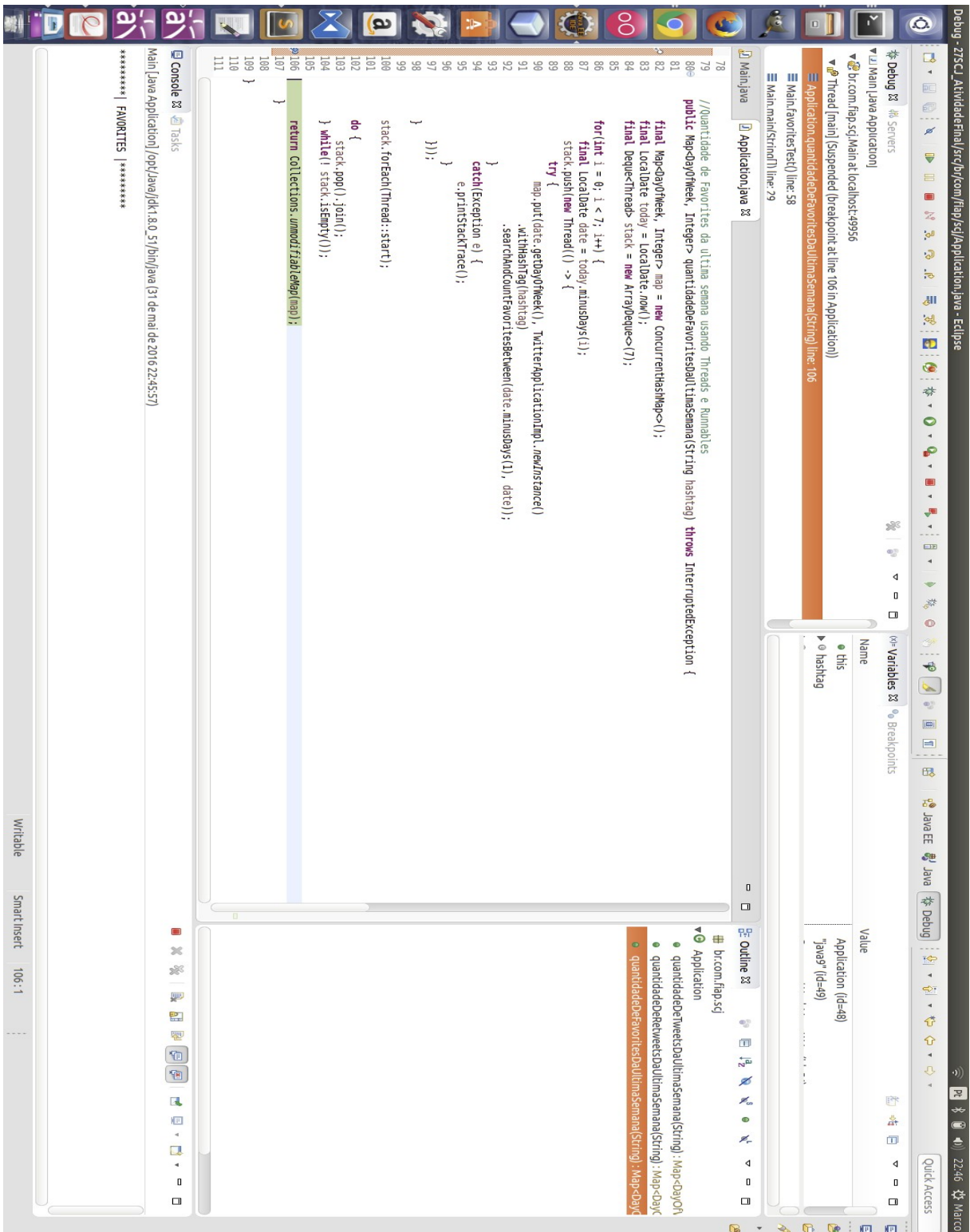


Diagrama de sequencia relacionado ao método tweetForTest



OBS: Imagens e o projeto do diagrama acompanha o projeto no Github

5 Screenshots



SÃO PAULO
2016

Este método recupera a quantidade de favoritos da ultima semana criando uma Thread para cada dia da semana para otimizar o tempo da busca.

Primeiramente se é definido um pilha do tipo Thread com sua capacidade igual a quantidade de dias da semana. Depois, em um loop com a quantidade de dias da semana, é armazenada na pilha instancias de Threads com classes anonimas de Runnable contendo nossa lógica de busca para um determinado dia em questão.

Após armazenar todos, finalmente se é executado o método start() percorrendo os elementos da pilha e após isso se é desempilhada os elementos da pilha de Thread invocando o método join() para aguardar o término da thread e a mesma se juntar a Thread principal. Quando todas as threads executarem se é retornado uma coleção chave-valor do tipo Map somente leitura contendo as quantidades e o dia da semana em questão.

Debug - z75CJ_AtividadeFinal/src/br/com/fiap/scj/twitter/TwitterApplicationImpl.java - Eclipse

Debug 33 Servers

Main [Java Application]

br.com.fiap.scj.Main at localhost:49956

Thread [main] (Suspended (breakpoint at line 106 in Application))

Application.quantidadeDeFavoritosDaUltimaSemana(String) line: 106

Main.favoritesTest() line: 58

Main.main(String[]) line: 29

/opt/java/dk1.8.0_57/bin/java (31 de mai de 2016 22:45:57)

Variables 33 Breakpoints

Name	Value
this	Application (id=48)
hashtag	"java9" (id=49)
map	ConcurrentHashMap<K,V> (id=54)
today	LocalDate (id=56)
stack	ArrayDeque<E> (id=58)

Main.java Application.java TwitterApplicationImpl.java 33

```

96
97
98     return totalCount;
99 }
100
101 @Override
102 public int searchAndCountFavoritesBetween(LocalDate since, LocalDate until) throws TwitterException {
103     requireNonNull(hashtag, "Hashtag de pesquisa não pode ser nulo");
104
105     Query query = new Query(hashtag);
106     query.setSince(LocalDate.toString(since));
107     query.setUntil(LocalDate.toString(until));
108
109     int totalCount = 0;
110     QueryResult result = twitter.search(query);
111     do {
112         query = result.nextQuery();
113         totalCount += result.getTweets()
114             .stream()
115             .map(status -> status.getFavoriteCount())
116             .reduce((value1, value2) -> value1 + value2)
117             .get();
118     } ifNonNull(query) { result = twitter.search(query); }
119     while(result.hasNext());
120
121     return totalCount;
122 }
123
124 @Override
125 public void tweet(String text) throws TwitterException {
126     twitter.updateStatus(text);
127 }

```

Outline 33

- br.com.fiap.scj.twitter
 - TwitterApplicationImpl
 - twitter: Twitter
 - hashtag: String
 - tweets: List<Tweet>
 - TwitterApplicationImpl()
 - newInstance(): TwitterApplicationImpl
 - withHashtag(String): TwitterApplicationImpl
 - searchTweetsBetween(LocalDate, LocalDate): TwitterApplicationImpl
 - searchAndCountRetweetsBetween(LocalDate, LocalDate): TwitterApplicationImpl
 - searchAndCountFavoritesBetween(LocalDate, LocalDate): TwitterApplicationImpl
 - tweet(String): void
 - tweetFor(String, String): void
 - collect(): List<Tweet>
 - localDateToString(LocalDate): String

Writable Smart Insert 105:50

Este método utiliza as classes da API Twitter4J para efetuar buscas através da classe Query. Com esta classe definimos as datas de início e fim e através do método search efetuamos a busca.

Este método retorna um QueryResult que deve ser iterado de forma página quando os resultados ultrapassam o valor padrão de resultados por página.

Enquanto iteramos os resultados estes são transformados, reduzidos e armazenados através da API Stream do Java 8