

Diego Antonio Fernandes de Aguiar – RM 32098
Marcelo Souza da Silva – RM 32005
Ricardo Leonardo Pansonato – RM 31838
Venus Rodrigues – RM 35376

**DESENVOLVIMENTO DE APLICAÇÕES JAVA - SOA
E INTERNET DAS COISAS
(Turma 29SCJ)
FUNDAMENTOS DA TECNOLOGIA JAVA
E MODELAGEM VISUAL UML 2.0
Prof. Msc. Michel Pereira Fernandes**



Telegram Bot Digital Bank

Projeto Final de Avaliação da disciplina de Fundamentos da Tecnologia JAVA e Modelagem UML 2.0 da MBA em Desenvolvimento de Aplicações JAVA - SOA e Internet das Coisas da FIAP.

São Paulo
2017

INDICE

1	<i>Introdução</i>	<i>4</i>
2	<i>Componentes, Bibliotecas e frameworks utilizados.....</i>	<i>5</i>
3	<i>Organização do Sistema.....</i>	<i>6</i>
3.1	Diagrama de Classes	9
3.2	Diagrama de Sequência	11
4	<i>Funcionalidades do Sistema</i>	<i>12</i>
4.1	Exibição de “boas vindas”	13
4.2	Criação de conta	14
4.3	Modificação de conta	15
4.4	Inclusão de dependentes	16
4.5	Exibição de dados da conta	18
4.6	Realização de Depósito	18
4.7	Realização de Saque	19
4.8	Solicitação de Extrato	20
4.9	Solicitação de empréstimo	20
4.10	Exibição de Saldo devedor e prazo de pagamento de empréstimo	21
4.11	Exibição detalhada dos lançamentos	22
4.12	Exibição de retiradas	22
4.13	Exibição de tarifas de serviço	23
4.14	Tela de ajuda	23
5	<i>Código fonte do projeto</i>	<i>25</i>

RESUMO DA ATIVIDADE

Este projeto consistiu em desenvolver um *Bot* (contração da palavra *robot*, robô em inglês), no aplicativo Telegram, para funcionar como um SAC robotizado, fornecendo interações programadas para realização de algumas transações bancárias de um suposto Banco Virtual, como:

1. Tela de boas-vindas do banco;
2. Criação de conta;
3. Modificação de conta;
4. Inclusão de dependentes (conta-conjunta);
5. Exibição dos dados do titular e dependentes;
6. Depósito;
7. Saque (custo do serviço R\$ 2,50);
8. Solicitação de extrato (custo do serviço R\$ 1,00);
9. Solicitação de empréstimo, cujo prazo máximo é de 3 anos e valor máximo é de 40 vezes o saldo da conta (custo do serviço R\$ 15,00 além de juros de 5% a.m.);
10. Exibição de saldo devedor do empréstimo e prazo de pagamento;
11. Exibição dos lançamentos detalhada, com somatória ao final;
12. Exibição das retiradas, com somatória ao final;
13. Exibição das tarifas de serviço, com somatória ao final dos serviços que já foram utilizados na conta; e
14. Tela de ajuda.

Tendo todas as operações devem ser armazenadas para futuras consultas em arquivo texto.

Assim através do *Bot*, clientes e não clientes que quiserem se comunicar com o banco para solicitar serviços, enviando mensagens pelo aplicativo de mensagem instantânea da rede social Telegram e receber respostas específicas sobre o assunto abordado.

A proposta do projeto foi de aplicar os conhecimentos adquiridos e revisados em todo seu conteúdo, abrangendo conceitos de modelagem de classes, diagramas, controle de exceção, coleções, *streams*, IO, threads, dentre outros abordados na disciplina de FUNDAMENTOS DA TECNOLOGIA JAVA E MODELAGEM VISUAL UML 2.0, ministrada pelo Prof. Msc. Michel Pereira Fernandes.

Este Relatório Técnico do projeto desenvolvido abordará os seguintes aspectos do projeto: Componentes, bibliotecas e frameworks utilizados; Explicação do uso de cada pacote (organização do sistema), classes e métodos; Diagrama de classes; Diagrama de sequência; Capturas de telas da aplicação comentando cada funcionalidade relevante e seu respectivo passo a passo; e por fim, o *Link* do GitHub com o código-fonte enviado.

Palavras-chave: 1. API Telegram Bot; 2. Digital Bank; 3. Fundamentos Java e UML

1 Introdução

Com o avanço da tecnologia móvel no mundo inteiro, explorando, dentre outras formas pelos comunicadores instantâneos aliados aos assistentes virtuais que tem se mostrado cada vez mais inteligentes e ajudando as pessoas a cumprirem suas tarefas do cotidiano estabelecendo uma plataforma de serviços de autoatendimento, com possibilidades também abertas as empresas.

Neste contexto, as redes sociais como o Telegram e o Facebook viram canal de atendimento em tempo real para as instituições, utilizando dos recursos dos *bots* proporcionando facilidade e flexibilidade de integração. Conforme o mercado de aplicativos móveis cresce, as empresas tentam buscar cada vez mais atrair a atenção dos clientes e consumidores. Os *bots* tem o potencial de transformar o modo como as pessoas se comunicam com as empresas. Eles prometem agilizar as interações entre empresas e clientes, além de automatizar processos de vendas que antes precisariam de mediação de usuários.

Os *bots* também são mais fáceis de instalar se comparados aos aplicativos, já que é só abrir o aplicativo de mensagens, localizar o *bot* e começar a conversar. Além do mais, aplicativos de mensagens possuem níveis de engajamento muito maiores que a maioria dos demais aplicativos. Eles ainda são mais rápidos e permitem a criação de mais interação entre as diversas funcionalidades.

Os bancos financeiros, por sua vez, estão nascendo ou migrando para a realidade digital, onde não é obrigatório ter agências físicas para obter seus principais serviços. Com o acesso massificado da internet, é possível realizar desde a abertura da conta e transações de qualquer natureza remotamente.

As mais variadas instituições já utilizam os *bots* para interação com os clientes, como por exemplo: ShopFácil.com, Visa, Alcoólicos Anônimos, UOL, Motoboy.com, Banco Original, entre outros que hoje estão presentes no Facebook Messenger.

Em suma, os *bots* estão cada vez mais presentes em nosso dia a dia, e estão assumindo um papel fundamental para os usuários que procuram cada vez mais praticidade, respostas rápidas e soluções objetivas. Esta nova tendência de serviços está crescendo cada vez mais não apenas para saciar as necessidades dos clientes, como também por parte das empresas visando redução de custos e maior eficiência em processos que hoje são executados manualmente, pois os *bots* podem ser uma poderosa ferramenta para aumentar o engajamento dos clientes, haja vista que conseguem atender várias pessoas de forma simultânea e estão disponíveis 24 horas por dia, sete dias por semana.

2 Componentes, Bibliotecas e frameworks utilizados

Na construção do sistema foram adicionadas algumas bibliotecas ao projeto para facilitar e oferecer recursos à implementação do *bot*, quais sejam:

a. API Java API for Telegram Bots:

. Esta API permite conectar *bots* ao sistema do Telegram. Os *Telegram Bots* são contas especiais que não exigem um número de telefone adicional para configurar. Essas contas servem como uma interface para codificar em algum lugar do seu servidor. Possui o código 100% aberto para todos os desenvolvedores que desejam criar aplicativos Telegram.

b. API Java *java.io*:

. Assim como todo o resto das bibliotecas em Java, a parte de controle de entrada e saída de dados (conhecido como *io*) é orientada a objetos e usa os principais conceitos mostrados até agora: interfaces, classes abstratas e polimorfismo. A ideia atrás do polimorfismo no pacote *java.io* é de utilizar fluxos de entrada (*InputStream*) e de saída (*OutputStream*) para toda e qualquer operação, seja ela relativa a um arquivo, a um campo *blob* do banco de dados, a uma conexão remota via *sockets*, ou até mesmo às entrada e saída padrão de um programa (normalmente o teclado e o console). No projeto a persistência em arquivo texto foi suportada por esta API e em identificadores *Serializable*.

c. API Java *java.util*:

. Esta biblioteca do Java é comumente utilizada em vários projetos, pois contém as classes e as interfaces usadas para tratar coleções como tabelas de *hash*, as listas vinculadas, pilhas e dicionários. Este pacote também fornece classes para lidar com fusos horários, manipulação de data e hora, internacionalização e permite a adição para outras classes utilitárias, como: um tokenizador de string, um gerador de números aleatórios e uma matriz de *bits*. No projeto esta API foi utilizada para manipulação de *List*, *Properties* (no armazenamento do TOKEN fornecido pelo Telegram), *Date*, *Locale*, etc.

d. API Java *java.text*:

. Esta biblioteca do Java fornece classes e interfaces para manusear texto, datas, números e mensagens de forma independente das linguagens naturais. No projeto foi adicionada para formatações utilizando principalmente: *DateFormat*, *NumberFormat* e *SimpleDateFormat*.

e. Apache Commons Lang 3:

. Esta biblioteca fornecida pela Apache é um pacote de classes utilitárias Java para as classes que estão na hierarquia *java.lang*, como métodos de manipulação de String, métodos numéricos básicos, reflexão de objeto, concorrência, criação e serialização e *Properties*. Além disso, contém aprimoramentos básicos para *java.util.Date* e uma série de utilitários dedicados a ajudar com métodos de construção, como *hashCode*, *toString* e *equals*. No projeto foi utilizado o *StringUtils* que fornece vários métodos de comparação com string, e também o *tuple.Pair* que fornece uma representação de tupla (par de dois elementos) possibilitando a manipulação livre de elementos com chave e valor.

Na construção do sistema não foram utilizado componentes adicionadas, tão pouco frameworks.

3 Organização do Sistema

Como o projeto foi desenvolvido na linguagem de programação Java, que é uma linguagem orientada a objetos (OO), utilizou-se o paradigma OO, no qual criamos objetos e fazemos com que eles interajam, sendo o sistema construído em classes (representação dos objetos) e estas são agrupadas em Pacotes, proporcionando uma organização das Classes por escopo e, ainda, o controle de acesso às classes, seus métodos e atributos, de acordo com o objeto das classes na aplicação.

Visto ao exposto o projeto foi dividido nos seguintes Pacotes:

a. Pacote *bot*:

Este pacote contém apenas a Classe *FiapTelegramBot*, a qual dispõe a implementação do método *main* da aplicação, ou seja, por onde executamos o sistema. Nesta Classe é inicializado o funcionamento do *bot* em si, utilizando os recursos da API Java API for Telegram Bots. Portanto, por convenção, esta classe é chamada "Principal" ou "Programa", pois contém o método *main*. Por sua função específica no programa, foi decidido isolá-la em um pacote exclusivo.

b. Pacote *command*:

Este pacote foi criado com o intuito de armazenar as classes referentes aos comandos que o *bot* irá suportar e executar, bem como a sistemática para suas execuções. Possui as Classes: *Command* e *CommandFactory*, e ainda o subpacote *impl*, que contém um Classe para cada comando suportado pelo *bot*.

A Classe *Command* é a abstração dos comandos de funcionalidades que o *bot* é capaz de realizar. Ou seja, é a classe de representação abstrata das funcionalidades do *bot*, trabalhando como uma classe “semi-pronta”, a partir da qual todas as implementações de funcionalidades devem partir (estender). Contém, também, os métodos comuns para realização de todos os comandos (ler mensagem passada pelo usuário, verificar necessidade de correção na informação passada pelo usuário e checar a conta do usuário que está interagindo).

A Classe *CommandFactory* possui lógica referente a execução das funcionalidades por parte do *bot*. Ela recebe as mensagens passadas pelos usuários e direciona a Classe de implementação referente ao comando solicitado, ou seja, cria o objeto da funcionalidade referente a solicitação passada pelo usuário. Vulgarmente falando, é a fábrica dos comandos de funcionalidades que o *bot* é capaz de realizar. Foi solicitado pelo usuário realizar saque a fábrica encontra a classe que representa essa funcionalidade e instancia o objeto para realização da funcionalidade.

c. Pacote *impl*:

Este pacote foi criado com o intuito de armazenar as classes de implementação dos comandos de funcionalidades que o *bot* é capaz de realizar. Assim, este pacote

contém uma classe para cada funcionalidade do *Bot Digital Bank*. Estas classes de implementação estendem a Classe Abstrata *Command*, ou seja, são subclasses de *Command*, sendo em cada uma, programada a lógica e sistemática específica de cada funcionalidade do *bot* (criar conta bancária, realizar saque, realizar depósito, etc.).

d. Pacote *dao*:

Este pacote foi criado para armazenar as Classes de Objetos de Acesso a dados (ou simplesmente DAO, acrônimo de *Data Access Object*), este conceito é um padrão para persistência de dados que permite separar regras de negócio das regras de acesso a banco de dados, sendo assim foi criado este pacote exclusivo às Classes dessa natureza. Como o projeto do *Bot Digital Bank* prevê apenas a persistência das operações realizadas pelos usuários em arquivo texto, o referido pacote possui somente a Classe *TelegramFileDAO*, responsável pela esta persistência, contendo a sistemática de criação, gravação e leitura de informações em arquivo utilizando recursos da API *java.io*.

e. Pacote *model*:

Este pacote foi criado para armazenar as classes referentes aos modelos de negócio do sistema, ou seja, as representações dos objetos alusivos ao cenário bancário. Estes são os objetos que serão manipulados pelas implementações das funcionalidades do *bot* e que terão suas informações persistidas pelo *dao*. Em nosso cenário, mapeamos a regra de negócio da seguinte maneira:

- Uma conta bancária de usuário (*AccountUser*) é uma extensão do próprio usuário, ou pessoa (*Person*), como melhor entender; e possui um balanço (*accountBalance*) que é modificado a medida que o usuário realiza as operações na conta, sendo assim o saldo. Da mesma forma, possui um balanço referente a possíveis empréstimos (*loanBalance*).
- Esta conta pode possuir nenhuma ou várias pessoas dependentes, assim a classe possui uma lista de objetos dependentes (*Dependent*), que por sua vez, também é uma extensão da Classe Pessoa, haja vista que usuário dono da conta e seus dependentes possuem os mesmos dados básicos.
- As Operações da conta anteriormente citadas foram mapeadas como uma classe específica (*AccountStatement*) que contém o atributo tipo da operação, que determinará a mudança no *accountBalance*.
- Este tipo foi definido como um *Enumeration* (*AccountStatementType*) de modo que facilite na caracterização de cada operação, pois especifica se a operação vai adicionar ou subtrair no balanço da conta do usuário, qual a descrição textual da operação e qual a taxa que o *Bot Digital Bank* cobrará na operação.

- A operação de empréstimo, por sua vez, foi definida como uma especialização da classe de operação, implementando suas especificidades (*LoanStatement*).

Visto ao exposto o pacote é constituído pelas Classes: *Person*; *AccountUser*; *Dependent*; *AccountStatement*; *LoanStatement*; e pelo *Enumeration*: *AccountStatementType*.

f. Pacote *exception*:

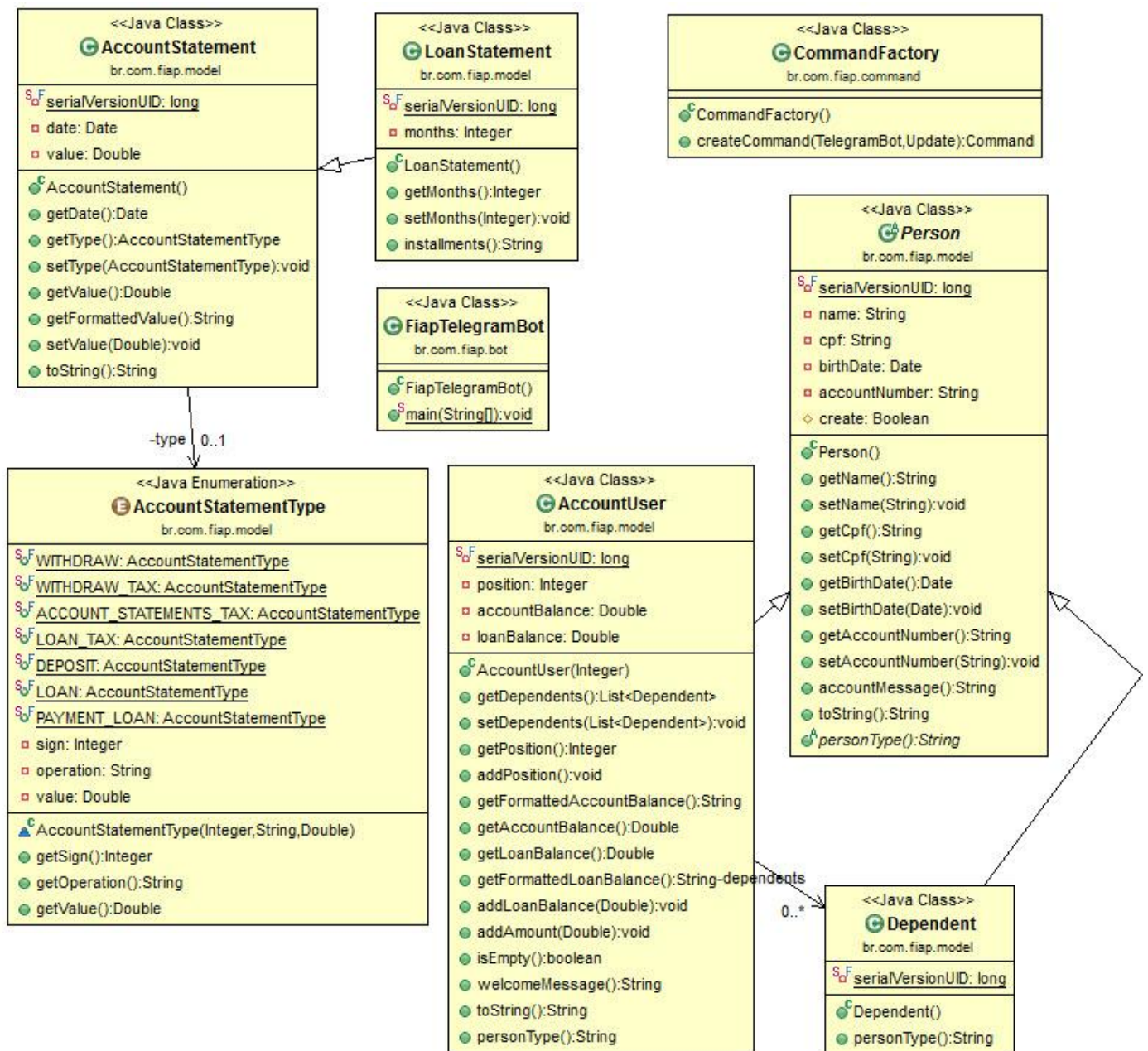
Este pacote foi criado para armazenar as classes referentes às exceções verificados do projeto e centralizar todas as exceções verificadas do projeto. Para tanto foi criada a Classe *OperationNotAllowed* que estende a Classe *Exception*. As exceções verificadas, ou checadas, representam condições inválidas em áreas fora do controle imediato do sistema, tais como: problemas de entradas inválidas pelos usuários, banco de dados, falhas de rede, arquivos ausentes.

3.1 Diagrama de Classes

As Classes do sistema foram dispostas em dois Diagramas de Classe, nos quais é possível identificar a representação da estrutura e relações das classes que servem de modelo para os objetos.

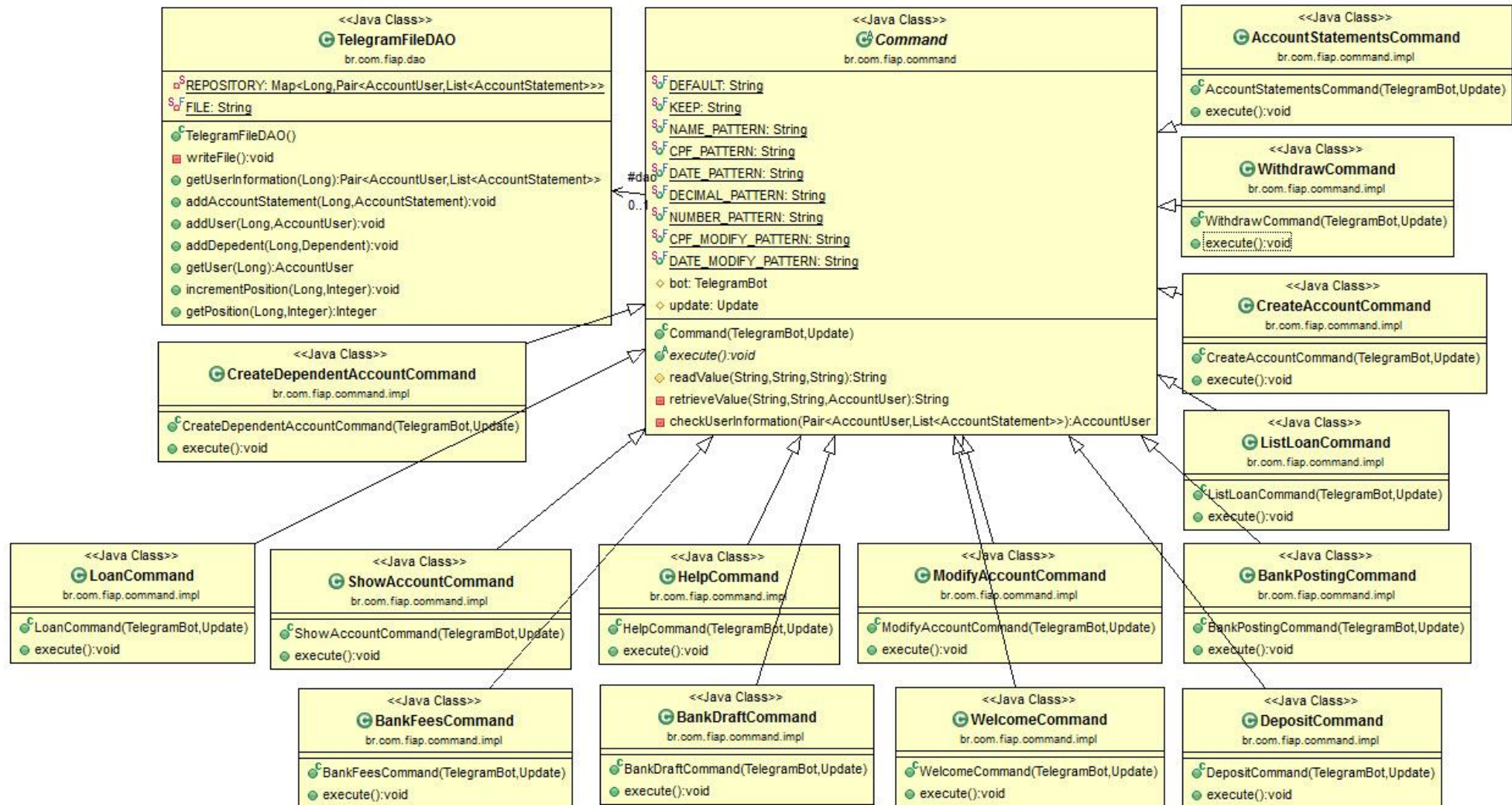
a. Diagrama 1:

. Classes do Pacote **model**, do Pacote **bot** e a Classe **CommandFactory**, que juntamente com a Classe **Command**, compõe o Pacote **command**.



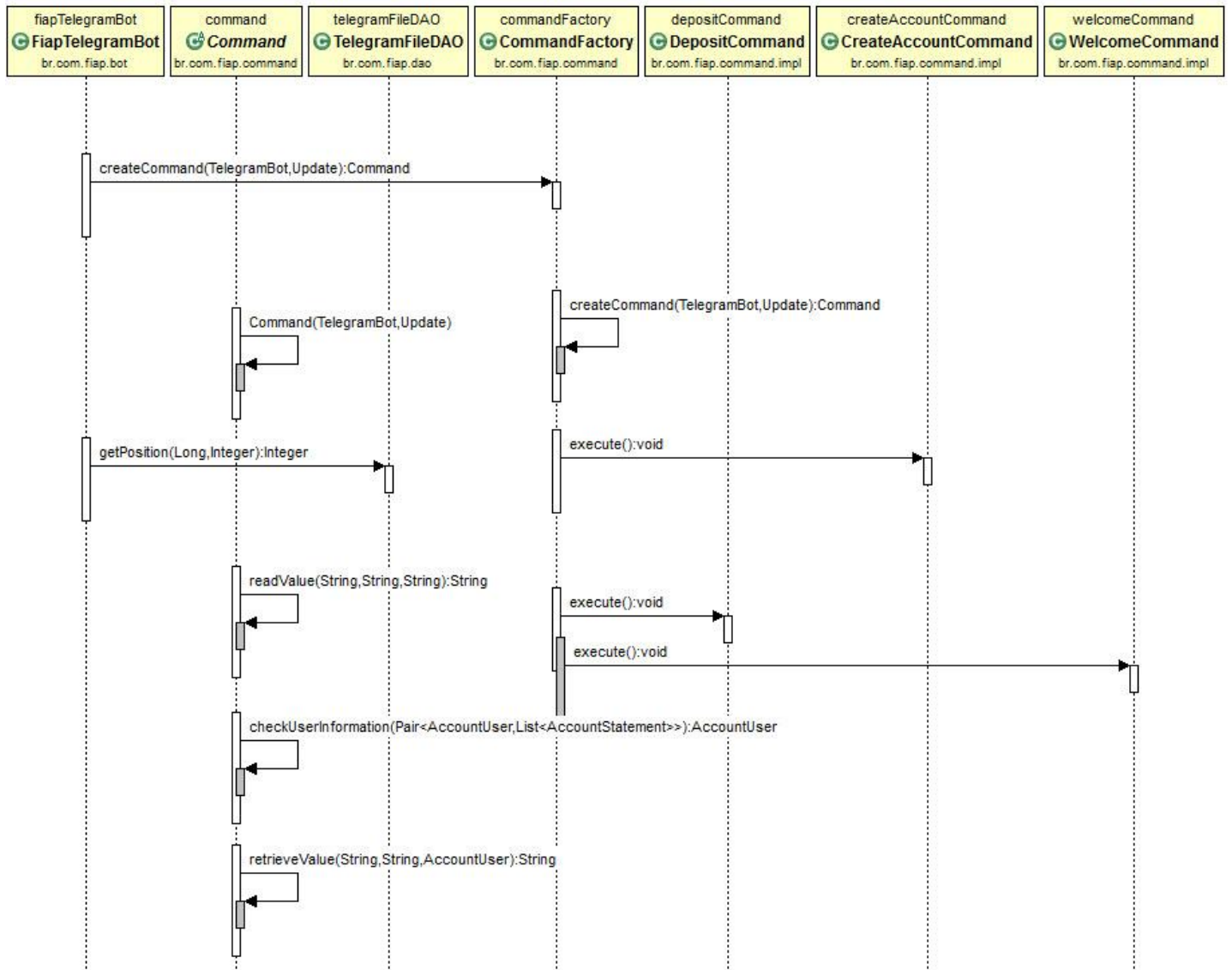
b. Diagrama 2:

Classes do Pacote **impl**, do Pacote **dao** e a Classe **Command**, que juntamente com a Classe **CommandFactory** (diagrama anterior), compõe o Pacote **command**.



3.2 Diagrama de Sequência

A fim de representar a sequência de mensagens trocadas entre as instâncias das classes do sistema, foi elaborado o Diagrama de Sequência abaixo, demonstrando a utilização das funcionalidades *CreateAccountCommand*, *WelcomeCommand* e *DepositCommand*.

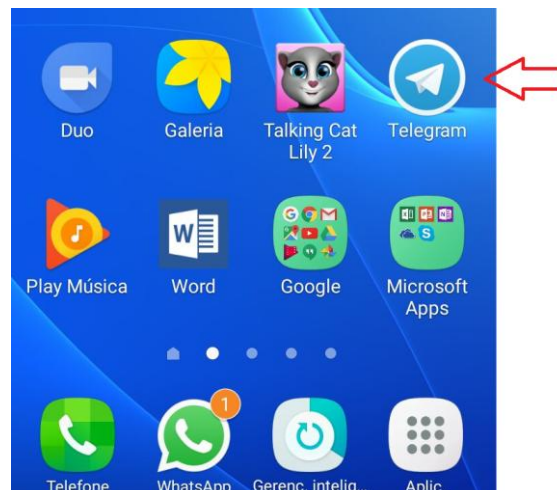


4 Funcionalidades do Sistema

Neste capítulo são descritas as funcionalidades do sistema, através de capturas de tela do mensageiro, sendo realizada uma interação simulada com o *bot*.

Para entrar no projeto, siga os seguintes passos:

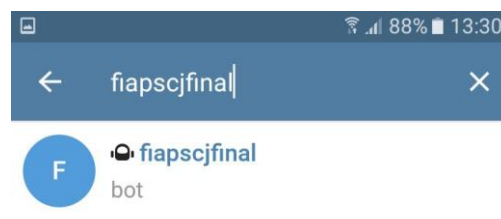
- a. Entrar no aplicativo Telegram (caso ainda não tenha instalado, baixe na *Play Store*/*Apple Store*):



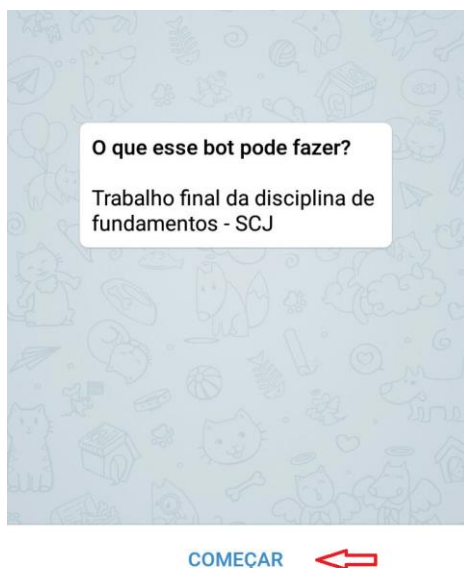
- b. Clique na lupa para pesquisar o *bot* do projeto:



- c. Na linha de pesquisa, informe: **fiapscjfinal**



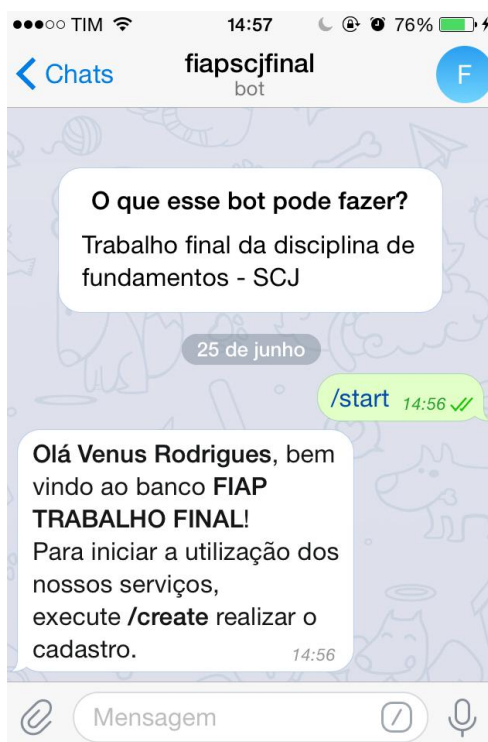
- d. Assim que selecionado o *bot* **fiapscjfinal**, será apresentada a tela a seguir. Clique em "Começar":



4.1 Exibição de “boas vindas”

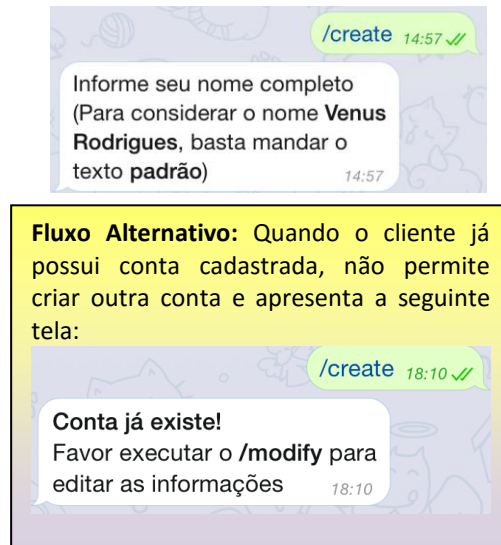
Após clicar em “Começar” pela primeira vez (tela anterior), automaticamente será apresentada a [Tela de Boas Vindas](#), sendo que poderá ser novamente acionada digitando-se [/start](#).

Essa tela exibe o nome completo que está cadastrado no Telegram e informa que para utilizar nossos cadastros, deve-se iniciar o processo criando uma conta através do comando [/create](#).



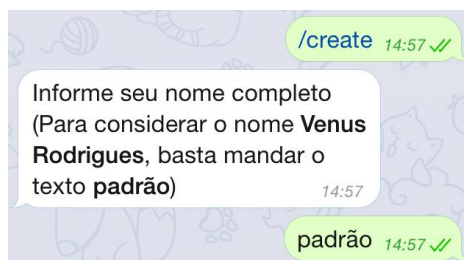
4.2 Criação de conta

- a. Para **CRIAR** uma conta, digite `/create` e aparecerá a seguinte tela:



- b. Informe o Nome Completo para criar sua conta.

Nesse passo, você terá a opção de seguir a criação de sua conta com o nome que já está registrado no seu Telegram. Para manter esse nome, siga a instrução da tela e digite `padrão`, caso queira informar outro nome, basta digitar o nome completo (não há nenhuma validação nessa informação).

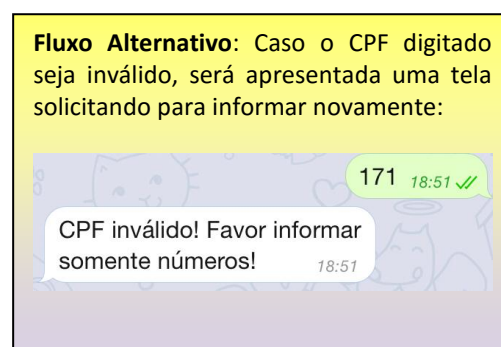
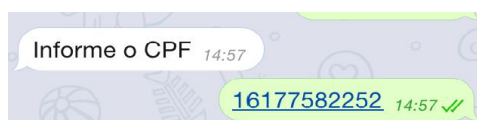


- c. Informe o CPF para criar a conta.

Aparecerá na tela uma mensagem `"Informe o CPF"` para a criação da conta.

. O CPF deverá ser informado com 11 dígitos, sem pontos ou traços (somente números).

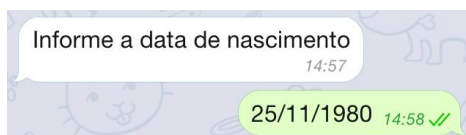
. A validação é apenas de 11 dígitos numéricos. Não é validado se o CPF informado bate com o cálculo do dígito verificador.



d. Informe a Data de Nascimento para criar a conta.

Aparecerá na tela uma mensagem “[Informe a data de nascimento](#)” para a criação da conta.

. A Data de Nascimento deverá ser informada no formato DD/MM/AAAA.

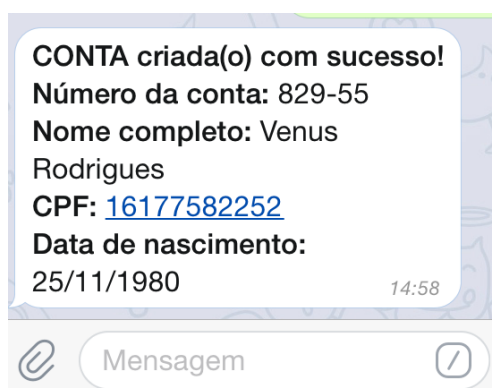


Fluxo Alternativo: Caso a Data de Nascimento seja inválida, será apresentada uma tela para informar novamente:



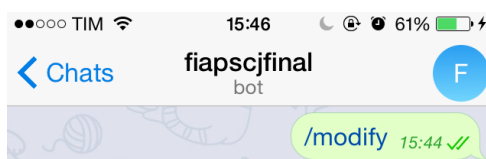
e. Conclusão da Criação de Conta.

Aparecerá na tela uma mensagem informando que sua conta foi criada:



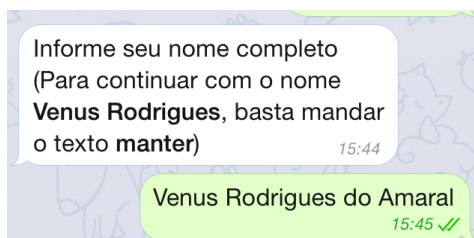
4.3 Modificação de conta

a. Para MODIFICAR qualquer informação de sua conta, digite [/modify](#).



b. Alterar o Nome Completo de sua conta.

Será apresentada uma mensagem solicitando para informar o nome correto que substituirá o nome atual:



Fluxo Alternativo: Caso seu nome já esteja correto, não é necessário informá-lo novamente, basta digitar **manter**, conforme mensagem em tela.

c. Alterar o CPF de sua conta.

Será apresentada uma mensagem solicitando para informar o CPF correto que substituirá o CPF atual. No exemplo, foi informado **manter** para que não fosse alterado.

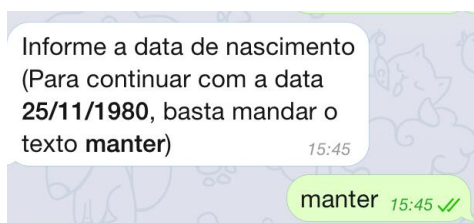
. O CPF nesse processo de alteração realiza as mesmas validações do CPF informado no comando /create.



d. Alterar a Data de Nascimento de sua conta.

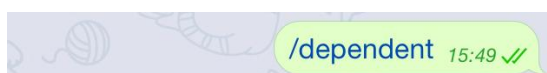
Será apresentada uma mensagem solicitando para informar a Data de Nascimento que substituirá a data atual. No exemplo, foi informado **manter** para que não fosse alterado.

. A data nesse processo de alteração realiza as mesmas validações da Data de Nascimento no comando /create.



4.4 Inclusão de dependentes

a. Para inserir um Dependente na sua conta, digite /dependent.



b. Informe o Nome Completo do Dependente.



c. Informe o CPF do Dependente.

. O CPF nesse processo realiza as mesmas validações do CPF informado no comando /create.



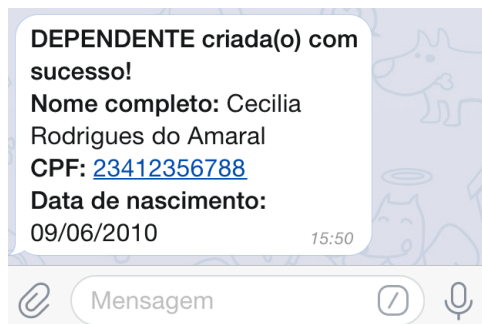
d. Informe a Data de Nascimento do Dependente.

. A data nesse processo realiza as mesmas validações da Data de Nascimento no comando /create.



e. Conclusão da Inclusão de Dependente.

Aparecerá na tela uma mensagem informando que o Dependente foi incluído na conta:



Poderão ser criados mais dependentes na mesma conta, basta chamar o comando /dependent novamente e inserir as informações solicitadas.

DEPENDENTES

Nome: Cecilia Rodrigues do Amaral
CPF: [23412356788](#)
Data de nascimento:
 09/06/2010

Nome: Marcio do Amaral
CPF: [12345578900](#)
Data de nascimento:
 11/06/1989

18:44

4.5 Exibição de dados da conta

- a. Para **CONSULTAR** os dados de sua conta e de seus dependentes, digite [/show](#) e serão apresentados os dados cadastrados.

[/show](#) 16:24 ✓

FI [fiapscjfinal](#)

Informações do titular da CONTA
Nome: Venus Rodrigues do Amaral
CPF: 16177582252
Data de nascimento: 25/11/1980
Número da conta: 829-55
Saldo atual: R\$ 0,00
Saldo de empréstimo: R\$ 0,00

DEPENDENTES

Nome: Cecilia Rodrigues do Amaral
CPF: 23412356788
Data de nascimento: 09/06/2010

4.6 Realização de Depósito

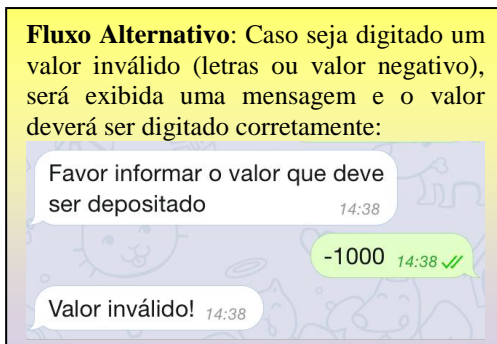
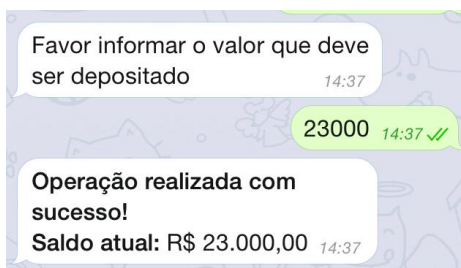
- a. Para realizar um **DEPÓSITO** em sua conta, digite [/deposit](#).

[/deposit](#) 16:35 ✓

- b. Informe o Valor do Depósito.

Aparecerá uma mensagem solicitando o valor do depósito. Digite o valor desejado para depósito e em seguida aparecerá uma mensagem confirmando que o depósito foi efetuado com sucesso.

. Utilizar a vírgula para informar valores com decimais.



4.7 Realização de Saque

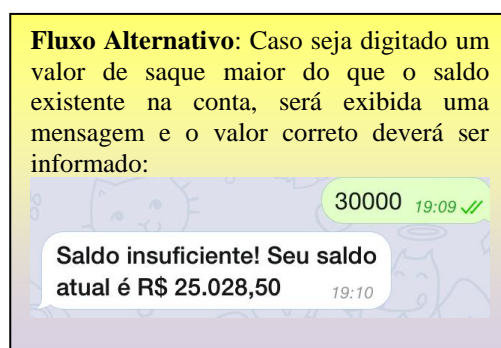
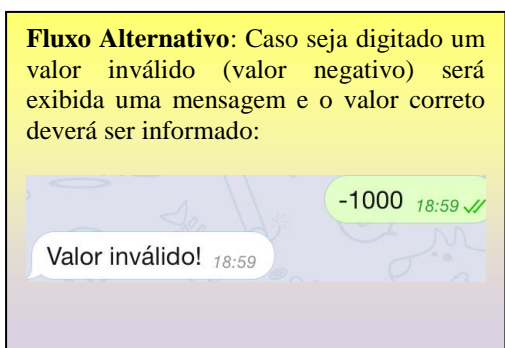
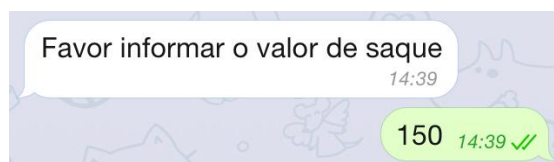
- a. Para realizar **SAQUE** em sua conta, digite [/withdraw](#).



- b. Informe o Valor do Saque.

Aparecerá uma mensagem solicitando o valor do saque.

. Será descontado o valor de R\$ 2,50 (referente à taxa de saque).



c. Saque realizado com sucesso.

Será apresentada a mensagem de sucesso, junto ao saldo atual, já descontado o valor da taxa de saque (R\$ 2,50):



4.8 Solicitação de Extrato

a. Para solicitar um **EXTRATO**, digite [/statements](#):



b. Exibição do Extrato.

Será exibido um extrato contendo cada lançamento realizado e o saldo atual, descontado o valor da taxa de Solicitação de Extrato (R\$ 1,00):



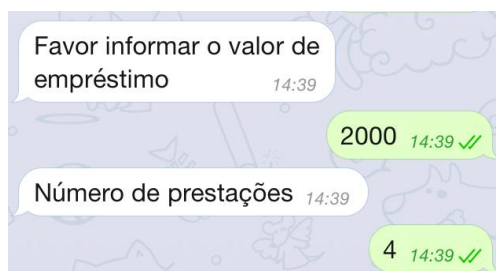
4.9 Solicitação de empréstimo

a. Para solicitar um **EMPRÉSTIMO**, digite [/loan](#):



b. Informe Valor do Empréstimo e Número de Prestações.

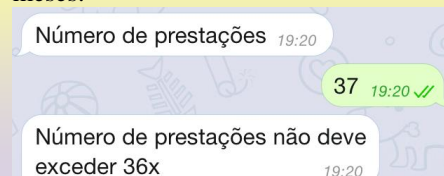
- . Está sendo calculada uma taxa de R\$ 15,00 por empréstimo.
- . O juro de 5% ao mês foi calculado por juros simples e não composto.
- . Para cada empréstimo é verificado se o valor desejado é até 40x do valor do saldo da conta.



Fluxo Alternativo: Caso seja informado um valor de empréstimo maior que 40x o saldo atual da conta, será exibida uma mensagem e deverá ser informado um valor abaixo de 40x:



Fluxo Alternativo: Caso seja informado um número de prestações maior que 36 (3 anos), será exibida uma mensagem e solicitado um número de prestações até 36 meses:



Fluxo Alternativo: Caso seja informado um valor de empréstimo ou número de prestações negativo, será exibida uma mensagem e deverá ser informado um valor positivo.



c. Conclusão do Empréstimo.

- . Será exibida uma mensagem informando se a operação foi realizada e o saldo, já acrescentando o valor do empréstimo e descontando o valor da taxa de R\$ 15,00:



4.10 Exibição de Saldo devedor e prazo de pagamento de empréstimo

- a. Para Exibir os Dados do Empréstimo, digite `/list_loan`.

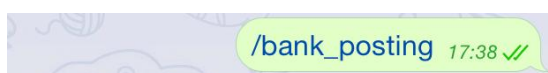


- b. Exibir Dados do Empréstimo e prazo de pagamento.

```
EMPRÉSTIMOS
-----
Data: 25/06/2017 05:11:53
Descrição: Empréstimo
Valor: R$ 2.000,00
Número de parcelas: 4
-----
1. R$ 525,00 vencimento: 25/07/2017
2. R$ 525,00 vencimento: 25/08/2017
3. R$ 525,00 vencimento: 25/09/2017
4. R$ 525,00 vencimento: 25/10/2017
-----
Saldo atual: R$ 2.100,00
-----
```

4.11 Exibição detalhada dos lançamentos

- a. Para exibir os Lançamentos Detalhados, digite `/bank_posting`.



`/bank_posting` 17:38 ✓✓

- b. Exibir os lançamentos de crédito da conta e a somatória desses créditos:

```
LANÇAMENTOS
-----
Data: 25/06/2017 04:36:30
Descrição: Depósito
Valor: R$ 23.000,00
-----
Data: 25/06/2017 07:00:19
Descrição: Depósito
Valor: R$ 100,50
-----
Data: 25/06/2017 07:07:55
Descrição: Depósito
Valor: R$ 100,00
-----
Total: R$ 23.200,50
```

4.12 Exibição de retiradas

- a. Para exibir as Retiradas, digite `/bank_draft`:



`/bank_draft` 17:45 ✓✓

- b. Exibir detalhes de todos os saques realizados:

```
SAQUE
-----
Data: 25/06/2017
04:53:29
Descrição: Saque
Valor: -R$ 150,00
-----
Total: -R$ 150,00 17:45
```

4.13 Exibição de tarifas de serviço

- a. Para Exibir as Tarifas de Serviços que já foram descontadas na conta, digite `/bank_fees`.

```
/bank_fees 17:54 ✓✓
```

- b. Exibir os valores fixos de cada taxa, e em seguida, quais taxas já foram descontadas da conta do cliente, com o total de taxas descontadas:


```
TAXAS E SERVIÇOS
-----
Taxa do saque: R$ 2,50
Taxa do extrato: R$ 1,00
Taxa do empréstimo: R$ 15,00
-----
Data: 25/06/2017 04:53:29
Descrição: Taxa do saque
Valor: -R$ 2,50
-----
Data: 25/06/2017 05:03:22
Descrição: Taxa do extrato
Valor: -R$ 1,00
-----
Data: 25/06/2017 05:11:53
Descrição: Taxa do empréstimo
Valor: -R$ 15,00
-----
Total: -R$ 18,50
```

4.14 Tela de ajuda

- a. Para verificar quais os comandos estão disponíveis, basta consultar a Tela de Ajuda, digitando `/help`:

```
/help 17:54 ✓✓
```

b. Exibir mensagem informando os comandos disponíveis na aplicação:

 **fiapscjfinal**

- /start** - Tela de boas-vindas do banco
- /create** - Criação de conta
- /modify** - Modificação de conta
- /dependent** - Inclusão de dependentes (conta-conjunta)
- /show** - Criação de conta
- /deposit** - Depósito
- /withdraw** - Saque
- /statements** - Solicitação de extrato
- /loan** - Solicitação de empréstimo
- /list_loan** - Exibição de saldo devedor do empréstimo e prazo de pagamento
- /bank_posting** - Exibição dos lançamentos detalhada
- /bank_draft** - Exibição das retiradas
- /bank_fees** - Exibição das tarifas de serviço

5 Código fonte do projeto

O código fonte do projeto foi armazenado na plataforma de hospedagem de código para controle de versão e colaboração, *GitHub*. Esta plataforma permitiu que os membros do grupo trabalhassem colaborativamente no projeto e de qualquer lugar, precisando apenas dispor de acesso à internet.

Para tanto, foi criado um repositório online no *GitHub*, denominado **SCJ-FUND-Final**, a partir do usuário do *GitHub* do membro do grupo Ricardo Leonardo Pansonato, onde estão disponíveis todos os arquivos gerados no projeto, sendo acessíveis através de qualquer navegador de internet pelo seguinte *link*:

<https://github.com/ricardopansonato/SCJ-FUND-Final>.

Ainda, o projeto pode ser clonado na íntegra através do *link*:

<https://github.com/ricardopansonato/SCJ-FUND-Final.git>.

BIBLIOGRAFIA

- Apache Software Foundation. (Junho de 2017). *Apache Commons Lang*. Fonte: commons.apache.org: <https://commons.apache.org/proper/commons-lang/index.html>
- Fernandes, P. M. (2017). Tutorial Básico de Integração da API Telegram Bot. São Paulo, Brasil: FIAP.
- GitHub. (Junho de 2017). *GitHub*. Fonte: Features For collaborative coding - developers work better, together | GitHub: <https://github.com/>
- Grupo Caelum. (Junho de 2017). *Pacote java.io - Java e Orientação a Objetos*. Fonte: Caelum | Cursos de Java, .NET, Android, PHP, Scrum, HTML, CSS e JavaScript: <https://www.caelum.com.br/apostila-java-orientacao-objetos/pacote-java-io/#15-2-orientacao-a-objetos-no-java-io>
- HE:labs. (junho de 2017). *HE:blog*. Fonte: Blog da HE:labs: <https://helabs.com/blog/empresas-incluem-bot-no-messenger-do-facebook-para-suas-estrategias-de-negocio/>
- Oracle and/or its affiliates. (Junho de 2017). *java.util (Java Platform SE 8)*. Fonte: Java Platform Standard Edition 8 Documentation: <https://docs.oracle.com/javase/8/docs/api/java/util/package-summary.html>
- Rock Content. (junho de 2017). *Marketing de Conteúdo* . Fonte: Blog Marketing de Conteúdo: <http://marketingdeconteudo.com/facebook-messenger-chatbot/>
- Telegram Messenger. (junho de 2017). *Telegram APIs*. Fonte: telegram.org: <https://core.telegram.org/bots>