|  |
| --- |
|  |
| Autoatendimento Bancário |
| Atividade final da matéria:  ***Introdução ao Java para Android*** |

**Flávio Mitsuyoshi Tamanaha Ota – RM: 47097**

**Leandro de Freitas – RM: 47052**

**Rodrigo Tetsuya Yamashiro Ota – RM: 47100**

**Wellington Sérgio Martiniano Santos – RM47086**

**10 de Agosto de 2014**

Este documento tem o objetivo descrever o projeto desenvolvido como trabalho final da matéria de Introdução ao Java para Android, ministrado por Michel Fernandes para o curso de MBA em desenvolvimento movél e games da FIAP (5MOB).

Sumário

[Introdução 3](#_Toc394069960)

[Calculadora 4](#_Toc394069961)

[Logging 5](#_Toc394069962)

[ Java.util.logger 6](#_Toc394069963)

[ FileHandler 6](#_Toc394069964)

[ java.util.logging.Formatter 6](#_Toc394069965)

[Diagrama de Classes do Logging 6](#_Toc394069966)

[Mensagens 6](#_Toc394069967)

[ Messages.properties 6](#_Toc394069968)

[ Messages.java 7](#_Toc394069969)

[ MessageUtils.java 7](#_Toc394069970)

[ StaticImport 7](#_Toc394069971)

[Diagrama de Classes de Mensagens 7](#_Toc394069972)

[Configuração 7](#_Toc394069973)

[Diagrama de classes de configuração 8](#_Toc394069974)

[User Interface (UI) 8](#_Toc394069975)

[Lanterna 9](#_Toc394069976)

[Telas 9](#_Toc394069977)

[Orientação a Objetos 10](#_Toc394069978)

[ TextElement.java 10](#_Toc394069979)

[ Display.java 10](#_Toc394069980)

[ Key.java 11](#_Toc394069981)

[Diagrama de Classes 11](#_Toc394069982)

[Threads 11](#_Toc394069983)

[Design 12](#_Toc394069984)

[Documentação do Código 12](#_Toc394069985)

[Dependências 13](#_Toc394069986)

[Diagrama completo de Classes 15](#_Toc394069987)

[Empacotamento 15](#_Toc394069988)

[Conclusão 18](#_Toc394069989)

## Introdução

Através deste documento que será apresentado junto com o código fonte, buscamos mostrar a nossa solução para o projeto apresentado, justificando cada escolha baseando em conceitos técnicos e experiência que serão avaliados.

Em relação às tecnologias aplicadas, procuramos utilizar a maior parte de conhecimentos ministradas em aula e que seriam viáveis para a solução do projeto proposto.

Ponto a ser considerar foi pensado de forma a ter uma usabilidade para que não seja tão complexa a sua utilização.

Mesmo que o Autoatendimento Bancário pareça ser um dos mais difíceis vimos que podemos aprender muito quando aceitamos alguns desafios.

## Autoatendimento Bancário

Dentre os projetos apresentados, escolhemos o autoatendimento bancário.

Dado que não foram colocados muitos requisitos para seu desenvolvimento, tentamos buscar um modo de fácil entendimento e que utilize as formas mais simples para que usuários não tenham problemas ao utilizar e nem se confundam.

Tivemos como inspiração as operações de banco utilizadas em celulares apenas para algumas modificações.

Dadas limitações técnicas colocadas no projeto, decidimos pela utilização do próprio console para a iteração com o usuário.

Embora nosso autoatendimento não tenha acesso ao banco de dados assim ficamos limitados ao programa com um saldo de R$ 10.000,00.

Não poderíamos nem tivemos tempo de inserir todas as funcionalidades bancárias colocamos apenas as mais simples.

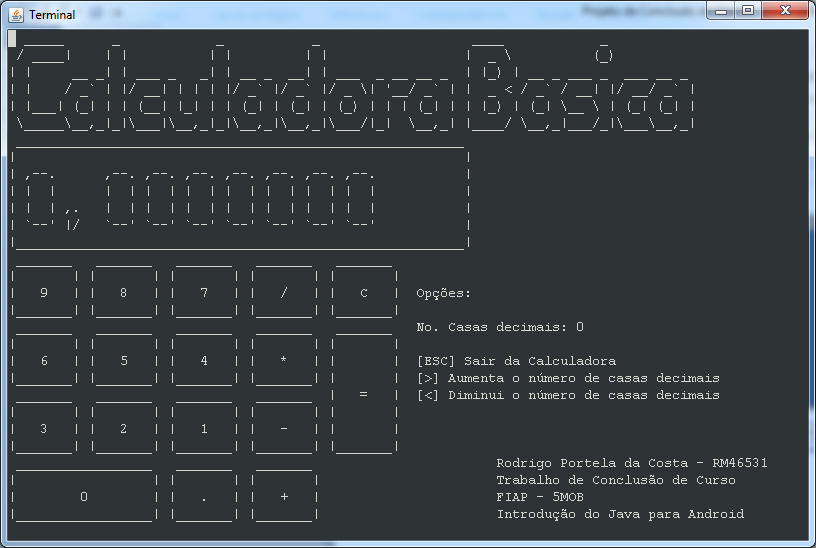


Figura 2 - Tela principal da calculadora

## Logging

O log é uma informação preciosa, tanto para o usuário, quanto para o responsável técnico pela sustentação do aplicativo.

Na figura abaixo, temos o relacionamento da aplicação até o destino do log armazenado

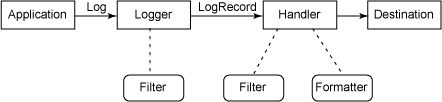


Figura 3 - Relacionamento do Logger

Ainda por falar em log, temos diversos níveis de mensagens que podemos apresentar. Esses níveis estão definidos de modo hierárquico, e cada nível é definido pelo tipo de mensagem que precisamos apresentar

|  |  |
| --- | --- |
| Nivel | Aplicação |
| FINEST | Mensagens da aplicação q |
| FINE | Mensagens de debug, que servem para passar informações técnicas para ajudar na manutenção do programa.  Ex. mensagem de conexão com o banco. |
| INFO | Informações referentes ao funcionamento do programa.  Ex. Arquivo salvo com sucesso. |
| WARNING | Mensagem que precisem de uma atenção mas que não comprometam o funcionamento da aplicação.  Ex. Já existe o usuário informado |
| SEVERE | Mensagens de erros que comprometam a aplicação e que não deveriam ocorrer.  Ex. Dropped connection. |

Dentro do escopo de log, utilizamos as seguintes tecnologias:

### Java.util.logger

Utilização ampla de logger para registrar mensagens das operações realizadas e os erros que possam acontecer.

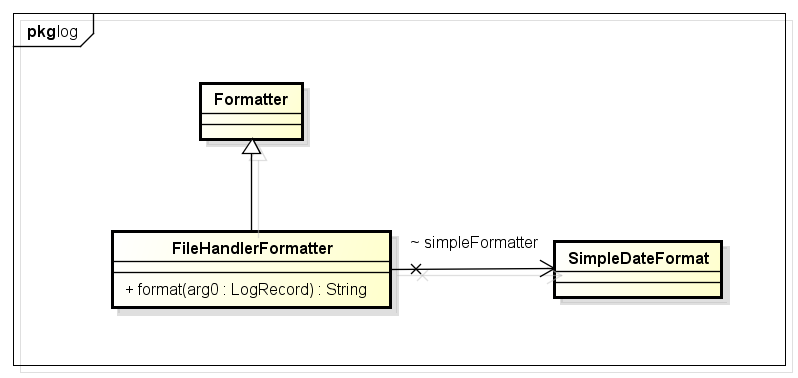
### FileHandler

Utilização de fileHandler para atender ao requisito que exige a persistência dos logs da aplicação em arquivo.

### java.util.logging.Formatter

Implementação da interface java.util.logging.Formatter para gerar o log conforme proposto no requisito do sistema.

### Diagrama de Classes do Logging



## Mensagens

Buscamos utilizar os conceitos de coesão e desacoplamento no tratamento e exibição de mensagens, visando facilitar a legibilidade e consequentemente a manutenção do código.

Com isso, criamos uma série de classes e artefatos que tornaram o desenvolvimento muito mais simples, são eles:

### Messages.properties

Arquivo contendo todas as mensagens do sistema

### Messages.java

Enum com todos os tipos das mensagens. O enum define o domínio das mensagens presentes no arquivo messages.properties.

### MessageUtils.java

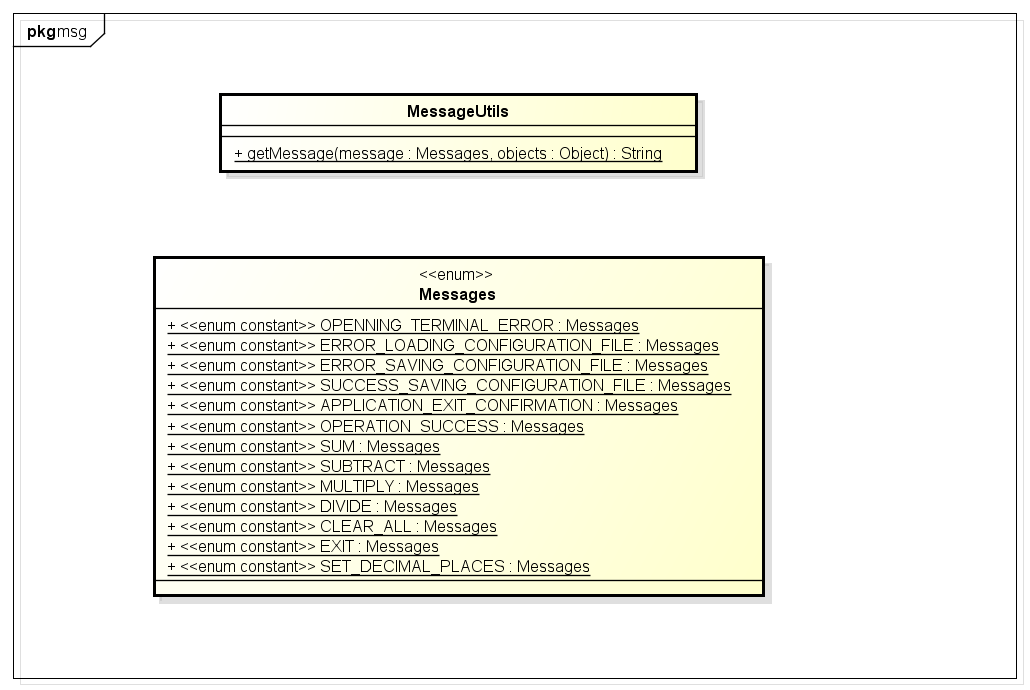
Classe responsável por recuperar as mensagens do arquivo de mensagens e complementá-la com as informações passadas pelo usuário.

Quando o sistema deseja uma mensagem, ele informa qual é o enum de mensagens que a classe MessageUtils retorna a mensagem pronta para exibição.

### StaticImport

Os valores do enum utilizados, quanto o método do MessageUtils responsável por retornar a mensagem, utilizam o staticimport, facilitando a leitura e manutenção do código.

### Diagrama de Classes de Mensagens



## Configuração

A aplicação utiliza arquivos properties para armazenar as configurações do aplicativo.

Para facilitar esse processo de leitura e gravação, criamos uma classe denominada Configuration.java que provê métodos que asbtraem a complexidade de manipular as configurações do sistema.

A aplicação é capaz de se autoconfigurar quando no momento da primeira execução, o aplicativo não identificar a presença do arquivo calculadora.properties no diretório da aplicação.

Em contrapartida, caso já exista o arquivo, o aplicativo realiza a leitura das configurações existentes do aplicativo.

Durante o processo de saída do aplicativo, a rotina de gravação das configurações da calculadora é acionado, com isso, quando o usuário acionar a calculadora novamente, a formatação da calculadora é preservada.

### Diagrama de classes de configuração



## User Interface (UI)

A UI é a interface que o usuário utilizará para acessar as funcionalidades da aplicação.

Uma aplicação que não é construída pensando no seu usuário, certamente pode não atender a seu objetivo.

Apesar do limitante imposto pelos requisitos do projeto, que deveríamos utilizar apenas o console ou Dialogs do Java, entendemos que se conseguíssemos tornar a calculadora algo visível e tangível para o usuário, conseguiríamos um resultado com muito mais qualidade.

Infelizmente o console não funciona de modo posicional, o que faz com que as iterações que o usuário realiza com a calculadora estejam sujeitos a uma barra de rolagem.

Buscamos então, alguma implementação pronta para a elaboração de interface gráfica em texto, que possibilitasse trabalhar em uma tela de console posicional.

### Lanterna

A biblioteca Lanterna foi criada para suprir uma lacuna do Java, que sempre trouxe muitos recursos de interface gráfica, mas que abandonou as interfaces textuais.

Mais informações desta biblioteca encontram-se disponíveis no caminho a seguir

<https://code.google.com/p/lanterna/>

### 

### Telas

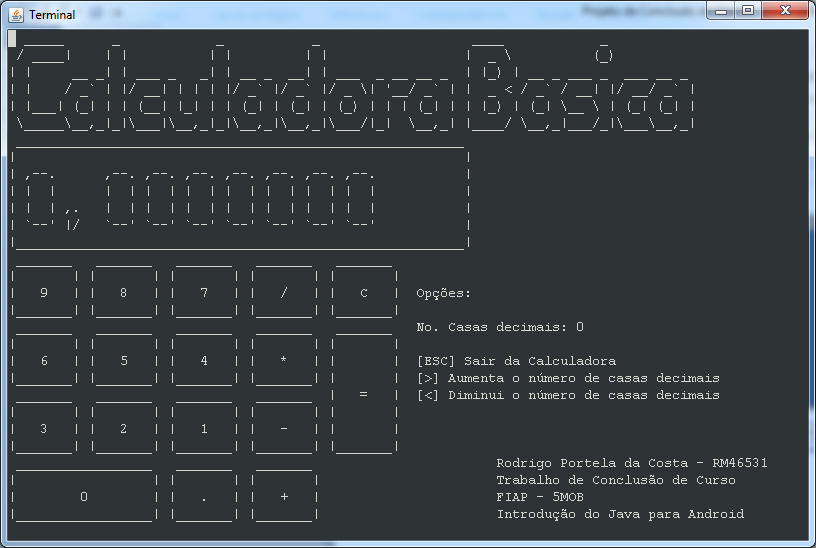


Figura 4 - Tela Principal

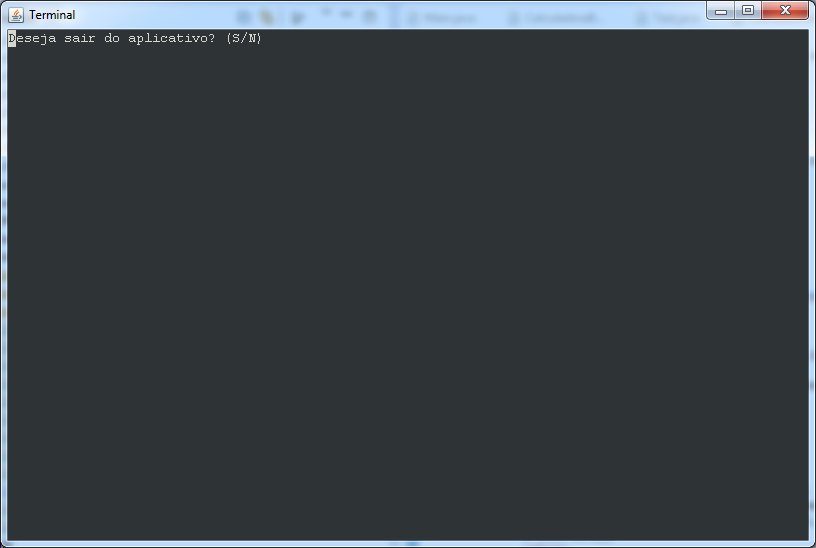


Figura 5 - Confirmação de saída

### Orientação a Objetos

Na proposta de tornar a calculadora algo visível para o usuário, entendemos que devemos desenhar os elementos que compõe a calculadora.

Assim, analisamos quais seriam os elementos ativos da calculadora e entendemos que precisaríamos de teclas que o usuário utiliza como entrada de dados e um display que funcionaria como uma saída de dados para o usuário.

### TextElement.java

Todos os elementos que são desenhados no console, possuem características comuns como a localização cartesiana do objeto e o método que as classes filhas precisam possuir e implementar para desenhar-se no console.

Assim, entendemos que precisamos de um molde para esses elementos e que utilizamos o conceito de classe abstrata.

### Display.java

Classe que apresenta os números digitados, resultado e informações de não conformidade da operação. Essa classe possui um método que o desenha no console e utiliza caracteres gráficos em ASCII com o intuito de melhorar a visibilidade do resultado das digitações e das operações.

### Key.java

Classe que representa uma tecla que o usuário pode pressionar no teclado. Essa tecla pode ter 3 dimensões pré-definidas, um caractere que aciona essa tecla e possui um método que o desenha no console.

### Diagrama de Classes

### C:\Users\magna2\Desktop\Classes da Interface com o Usuário.png

## Threads

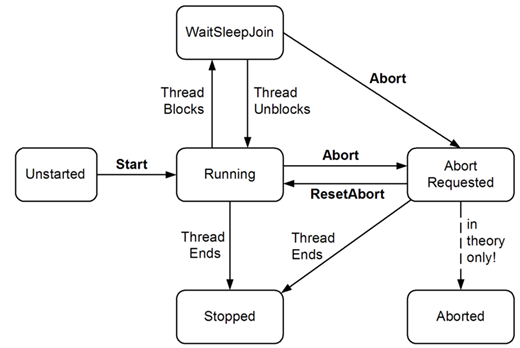
Para melhorar a experiência do usuário, permitindo ele ver o que ele está digitando, geramos um efeito de realce nas teclas que estão sendo pressionadas.

Quando o usuário pressiona uma tela, o aplicativo realça a tecla e inicia um thread com um tempo definido de 200ms, e ao final desse tempo, o aplicativo remove o realce da tecla.

Esse comportamento utiliza threads para evitar o congelamento da aplicação durante o efeito do realce da tecla.

Assim o usuário tem a percepção de que a aplicação está respondendo a suas digitações.

### Ciclo de vida de Thread



## Design

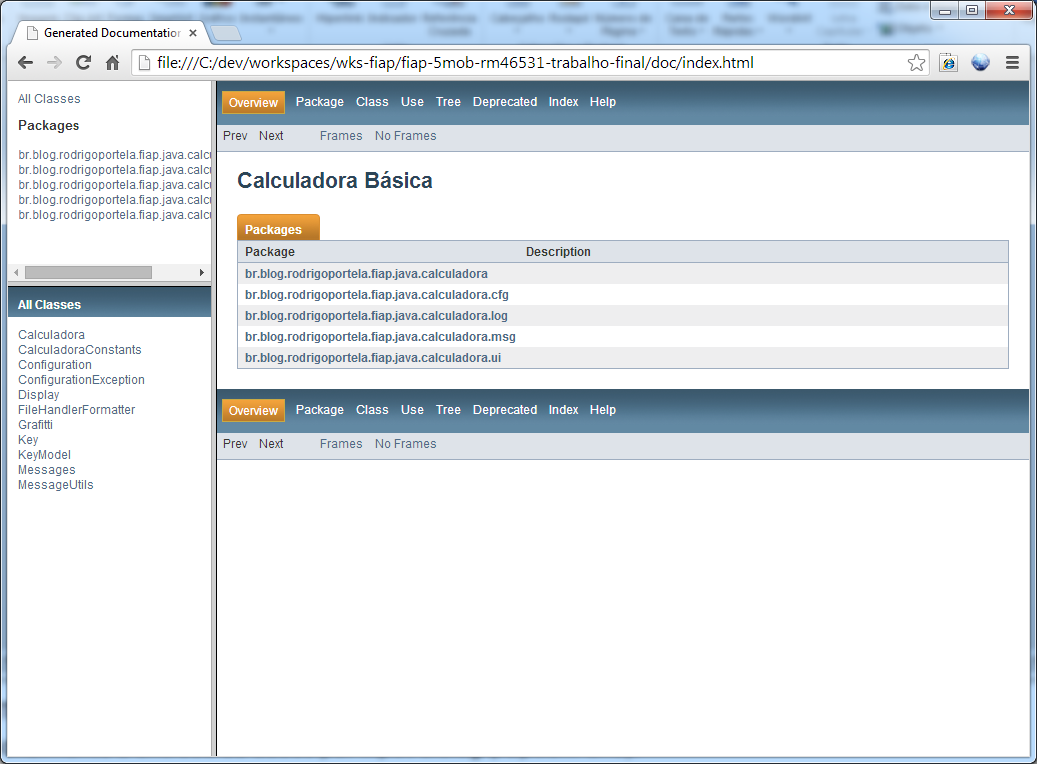
Dado a limitação do console, visando construir uma aplicação que seja agradável ao usuário, utilizamos o conceito de Grafitti com caracteres ASCII, que é uma técnica de criar pseudo-elementos gráficos utilizando caracteres texto.

Assim, conseguimos como resultado uma calculadora com uma aparência muito mais intuitiva para um usuário leigo.

## Documentação

### Documentação do Código

A documentação do código foi elaborada baseada em Javadoc e pode ser acessada diretamente da pasta doc/javadoc contida dentro do projeto da calculadora.



### Documentação técnica

Foi elaborada uma documentação básica para utilizando o Astah. Dentro dele podemos encontrar todas as classes e diagramas que foram utilizados para a elaboração deste documento.

## Dependências

Para construir o aplicativo utilizamos o jr7 para desenvolve-lo então para compila-lo e testa-lo aconselhamos que utilizem o jr7.

## Diagrama completo de Classes

## Empacotamento

Utilizamos os pacotes com uma nomenclatura de fácil entendimento:

*br.com.fiap.sistema:* possui a classe main da execução.

*br.com.fiap.sistema.conta:* possui as classes que criam a conta e fazem sua movimentação.

*br.com.fiap.sistema.exceptions:*  possui as classes que tratam os erros possiveis no Autoatendimento.

*br.com.fiap.sistema.intefaces:* possui a Interface do calculo do Imposto.

## Casos de Uso

|  |
| --- |
| Criando a conta |
| Pré-requisitos: iniciar a aplicação |
| Usuário informa seu nome;  * Informa o número da agência; * Informa o número da conta; * Informa sua senha; * Sistema Informa que usuário foi premiado e possui um saldo de 10.000 reais; * E pede para que se insira a operação; |

|  |
| --- |
| Fazendo um Saque |
| Pré-requisitos: Ter executado pelo menos o Caso de uso 1 |
| Usuário informa a operação 1;  * Informa o valor do saque; * Sistema informa que o saque foi feito com sucesso; * Sistema pergunta se o usuário quer fazer mais alguma operação; * Usuário pode inserir 1 para fazer mais operações ou inserir 2 para sair; |

|  |
| --- |
| Fazendo um Depósito |
| Pré-requisitos: Ter executado pelo menos o Caso de uso 1 |
| Usuário informa a operação 2;  * Informa o valor do deposito; * Sistema informa que o deposito foi feito com sucesso; * Sistema pergunta se o usuário quer fazer mais alguma operação; * Usuário pode inserir 1 para fazer mais operações ou inserir 2 para sair; |

|  |
| --- |
| Consulta saldo |
| Pré-requisitos: Ter executado pelo menos o Caso de uso 1 |
| Usuário informa a operação 3;  * Sistema informa que o saldo da conta; * Sistema pergunta se o usuário quer fazer mais alguma operação; * Usuário pode inserir 1 para fazer mais operações ou inserir 2 para sair; |

|  |
| --- |
| Consulta extrato |
| Pré-requisitos: Ter executado pelo menos o Caso de uso 1 |
| Usuário informa a operação 4;  * Informa o tipo de impressão 1 na tela 2 arquivo e 3 cancela;   + Caso o usuário digite 1 o extrato aparecerá na tela;   + Caso o usuário digite 2 será gerado um extrato em arquivo;   + Caso o usuário digite 3 sera cancelada a consulta; * Sistema pergunta se o usuário quer fazer mais alguma operação; * Usuário pode inserir 1 para fazer mais operações ou inserir 2 para sair; |

|  |
| --- |
| Cálculo de Imposto |
| Pré-requisitos: Ter executado pelo menos o Caso de uso 1 |
| Usuário informa a operação 5;  * Sistema informa que o imposto da conta; * Sistema pergunta se o usuário quer fazer mais alguma operação; * Usuário pode inserir 1 para fazer mais operações ou inserir 2 para sair; |

|  |
| --- |
| Sair |
| Pré-requisitos: Ter executado pelo menos o Caso de uso 1 |
| Usuário informa a operação 0;  * Sistema encerra a aplicação; |

## Conclusão

Concluímos que se aprende realmente quando começamos a exercitar, esse trabalho nos fez utilizar conceitos que em grande maioria já utilizaríamos em java trabalhando, mas que tirando como uma base para o android será muito útil para nossas futuras aplicações.

O autoatendimento bancário foi um desafio a ser superado que em equipe e com o conhecimento adquirido e relembrado pelas aulas se tornou menos complicado.