INSTITUTO POLITÉCNICO DE BRAGANÇA
Escola Superior de Comunicação,
Administração e Turismo

Programação IV
2015/2016

Informática e Comunicações

Trabalho elaborado por:

Bruno Ferreira – a31977
Pedro Belchior – a33042

Índice

Introdução

Este trabalho foi desenvolvido no âmbito da unidade curricular Programação IV da licenciatura em Informática e Comunicações da Escola Superior de Administração, Comunicações e Turismo do instituto Politécnico de Bragança.

O trabalho consiste no desenvolvimento de uma aplicação para a plataforma Android, usando o ambiente de desenvolvimento Android Studio.

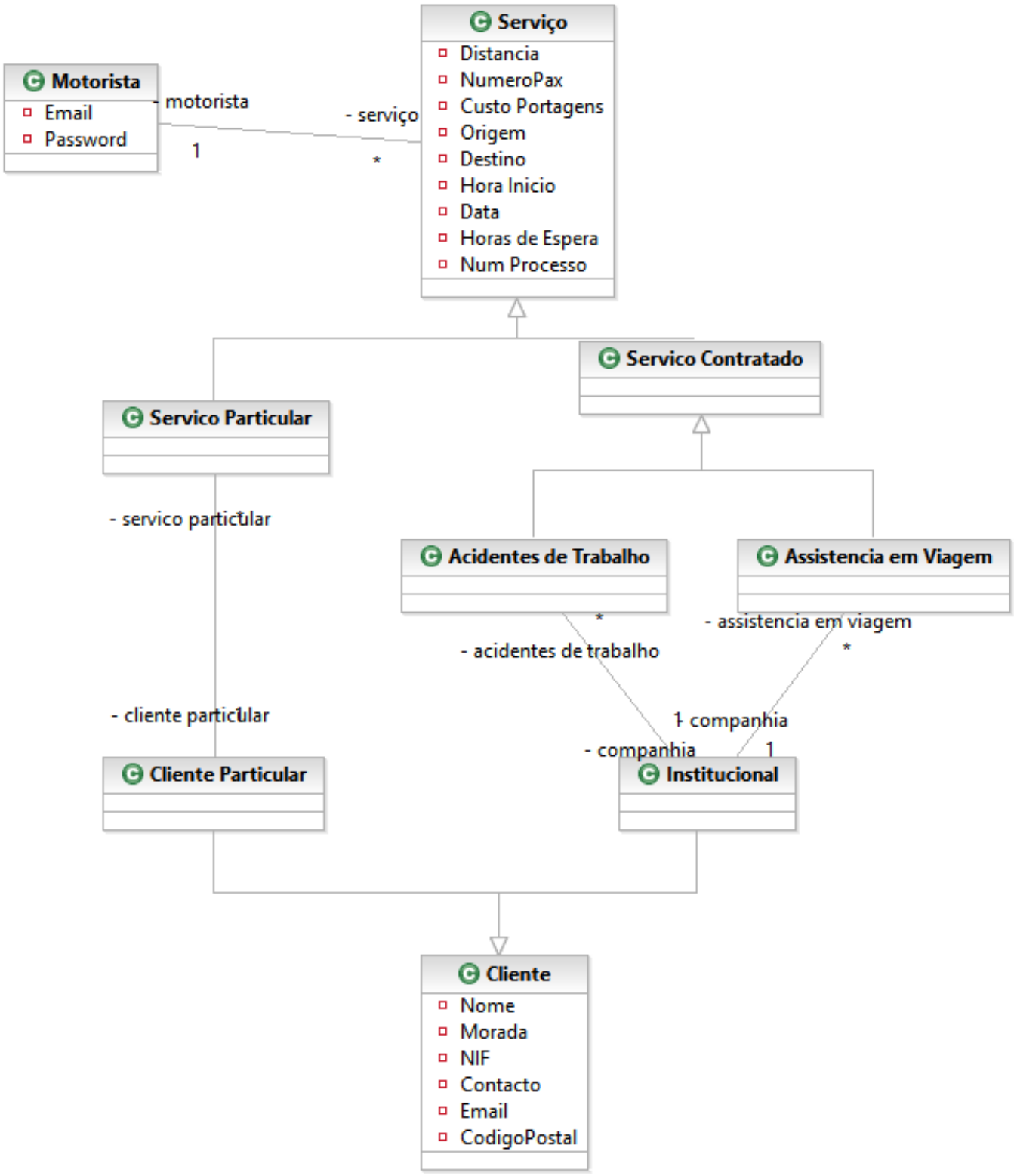
A aplicação tem como objetivo fazer o processamento informático dos serviços de táxi.

Descrição dos Requisitos

A aplicação deve:

- gravar pontos de coordenadas GPS que representem fidedignamente o trajeto efetuado pelo táxi desde a recolha dos passageiros até à sua descarga;
- armazenar a data e hora de carga dos passageiros;
- armazenar o número de passageiros transportados;
- armazenar o número do processo respetivo ao serviço caso seja um serviço pedido por uma companhia de seguros;
- armazenar o nome do cliente que solicitou o serviço;
- armazenar o custo das portagens caso existam;
- armazenar as horas de espera caso existam;
- calcular e armazenar a distancia percorrida;
- armazenar as coordenadas da praça de táxis à qual está associado o utilizador(neste caso o motorista);
- calcular e armazenar o trajeto desde a praça de táxis até ao local de carga dos passageiros;
- calcular e armazenar o trajeto desde o local de descarga dos passageiros até à praça de táxis;
- armazenar dados dos clientes;
- consultar e modificar dados dos clientes;
- consultar e modificar dados dos serviços efetuados;
- enviar as informações relativas ao serviço para o email do utilizador;
- proteção dos dados.

Diagramas de Classes



Funcionalidades

Todos os requisitos foram implementados. A nível de consultas apenas tivemos tempo para implementar consultas por nome de cliente e por número de processo do serviço.

Os dados são armazenados prioritariamente num *Web Server*, no entanto, caso não haja acesso à Internet ou o *Web Server* não esteja ativo, os dados são guardados localmente e na próxima utilização da aplicação em que haja acesso à Internet esses dados serão sincronizados.

É sempre mantida uma lista de Clientes com apenas alguns atributos (Nome do Cliente, Tipo de Cliente e Email) no dispositivo móvel.

Todos os dados armazenados no modo *offline* são armazenados em ficheiros XML.

A data e a hora do serviço é armazenada utilizando o tempo do sistema.

O número de passageiros e a hora de espera são introduzidos manualmente pelo utilizador.

O número de processo dos serviços particulares é calculado usando a hora e a data do sistema. Já o dos serviços para companhias é introduzido manualmente pelo utilizador.

O trajeto é gravado num ficheiro XML usando as funcionalidades GPS do sistema. É utilizada a *Roads API* da *Google* para corrigir os erros de captura do GPS.

O trajeto da praça de táxis para o local de carga e do local de descarga para a praça de táxis é calculado usando a *Directions API* da *Google.*

O endereço do local de carga e do local de descarga é calculado usando a *Geocoding API* da *Google.*

O email com as informações relativas ao serviço é enviado logo após a submissão do serviço pelo utilizador.

O acesso a todos os dados armazenados localmente ou no *Web Server* está sujeito a autenticação prévia. Sendo que a *password* é guardada depois de encriptada.

Estrutura da Aplicação

A aplicação está organizada usando seis *Packages* diferentes:

Package ClassesDados:

Este *Package* tem duas classes que instanciam os dois objetos de dados usados na aplicação.

Classe Cliente

Classe que define os objetos do tipo cliente com os seus atributos.

Classe Serviço

Classe que define os objetos do tipo serviço com os seus atributos.

Package Constantes

Este Package é constituído por uma única classe, classe Constants.
Nesta classe, tal como o nome indica, são armazenadas constantes globais usadas pelas diferentes classes da aplicação.

Package ClassesHelper

Este package é constituído por nove classes, que como o nome indica, armazenam métodos que são utilizados nas diferentes *Activities*.

Classe GPSHandler

Como o nome indica esta classe contém métodos que manipulam o sensor de GPS.

```
package p4.geretaxi;

import ...

public class GPSHandler {

    private LocationManager lManager;
    private LocationListener lListener = null;

    private static Context context;

    public GPSHandler(Context c) {...}

    public void initGPS(final String processo, final Activity activity) {...}

    public void listenerClose() {...}

    public void displayPromptForEnablingGPS(final Activity activity)
    {...}
}
```

Método initGPS

Recebe como parâmetros uma String e uma Activity.
Ativa um LocationManager, se o GPS não está ativo invoca um Dialog, definido nesta mesma classe que pergunta ao utilizador se pretende ativar o GPS. Instancia-se um LocationListener e faz-se o override do Método OnLocationChanged. Esse override consiste na invocação do método que grava as coordenadas GPS num ficheiro XML. Faz o update da localização a cada 4000 *ms* e/ou 300 *m.*

Método listenerClose

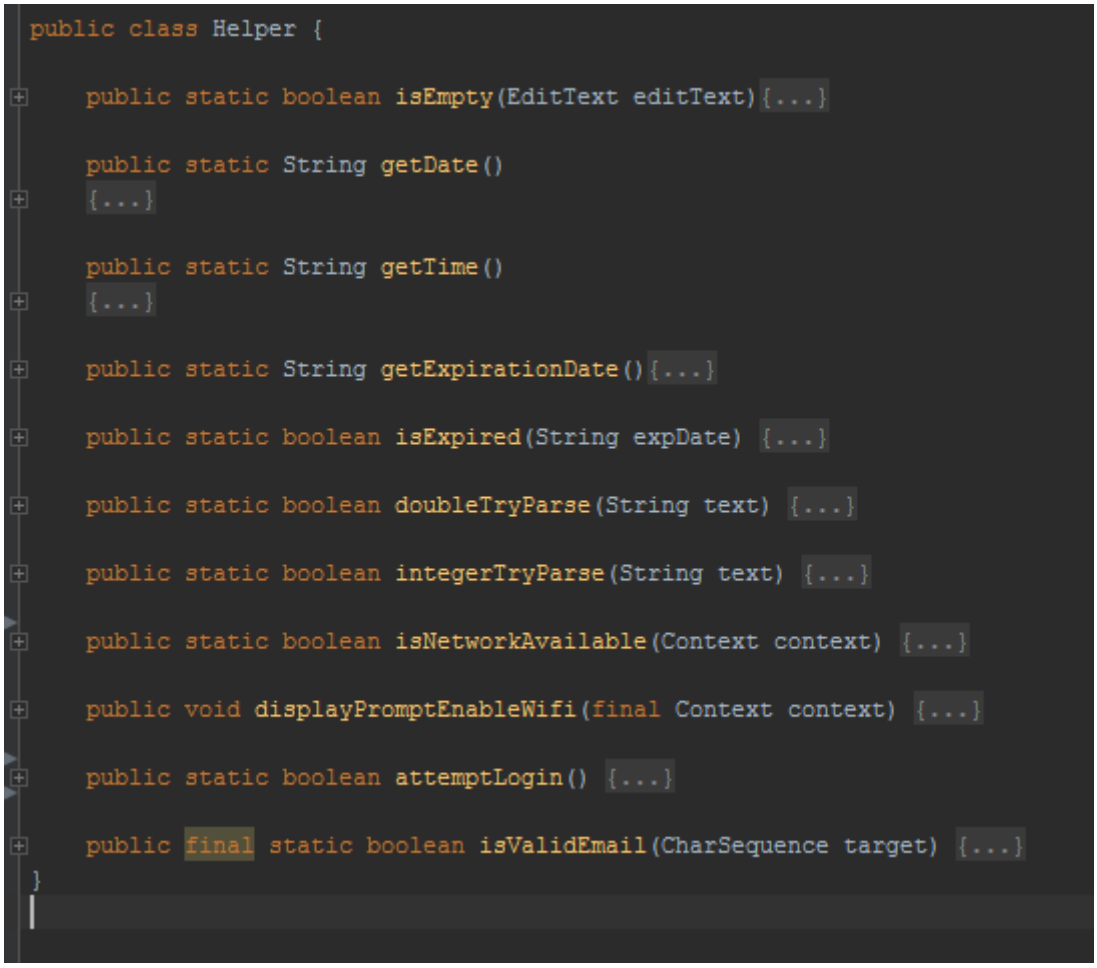
Verifica se o LocationListener está ativo, em caso afirmativo desativa-o.

Método displayPromptForEnablingGPS

Recebe como parâmetro uma Activity e ativa um AlertDialog que envia o utilizador para os Settings do dispositivo de modo que possa ativar o GPS.

Classe Helper

Classe que reúne uma série de métodos utilitários usados pelas outras classes da aplicação.



Método isEmpty

Método estático que recebe como parâmetro uma EditText e retorna true se a EditText está vazia e false caso contrário.

Método getDate

Método estático que retorna uma String contendo a data do sistema convertida para o formato dd-MM-yyyy.

Método getTime

Método estático que retorna uma String contendo o tempo do sistema convertido para o formato hh:mm.

Método getExpirationDate

Método estático que retorna uma data posterior em 30 dias à data em que foi invocado.

Método isExpired

Método estático que recebe uma String contendo uma data e retorna true se essa data for anterior à data atual.

Método doubleTryParse

Método estático que recebe uma String e que retorna true se essa String contem um Double.

Método integerTryParse

Método estático que recebe uma String e que retorna true se essa String contem um Integer.

Método isNetworkAvailable

Método estático que recebe como parâmetro um Context e retorna true se verificar que o dispositivo está registado numa rede.

Método displayPromptEnableWifi

Método que ativa um AlertDialog para enviar o utilizador para as definições do dispositivo que ativam a conexão à internet.

Método attemptLogin

Método estático que lê o email e a password que estão armazenadas nas SharedPreferences do dispositivo e que caso exista conexão tenta fazer autenticação no Web Server. Caso o Web Server esteja desativado, ele verifica a validade da autenticação guardada no dispositivo. Se for válida coloca a sessão que está guardada nas SharedPreferences com true e retorna true. Caso consiga fazer autenticação no Web Server coloca a sessão a true e atualiza a validade da autenticação para uma data posterior em 30 dias. Em todos os outros casos coloca a sessão a false. Se não houver conexão à Internet verifica a validade da autenticação. Se for válida coloca a sessão a true, caso contrário coloca a sessão a false.

Método isValidEmail

Método estático que recebe um CharSequence e retorna true caso essa CharSequence represente um email.

Classe Mail

```
public class Mail extends javax.mail.Authenticator {
    private String _user;
    private String _pass;

    private String[] _to;
    private String _from;

    private String _port;
    private String _sport;

    private String _host;

    private String _subject;
    private String _body;

    private boolean _auth;

    private boolean _debuggable;

    private Multipart _multipart;


    public Mail() {...}

    public boolean send() throws Exception {...}

    public void addAttachment(String filename) throws Exception {...}
    private String getUserMail() {...}

    public void set_to(String[] _to) {...}

    public String get_from() {...}

    public void set_subject(String _subject) {...}

    public void set_body(String _body) {...}
```

Classe que estende a classe javax.mail.Authenticator.

Contrutor por omissão Mail

Cria o objecto do tipo mail e faz o set dos atributos host, port, sport, user e pass com os atributos necessários para fazer uso da conta de email geretaxiapp@gmail.com. À propriedade from é atribuído o valor do email do utilizador.

Método send

Método que envia o mail retornando true caso tenha tido sucesso.

Método addAtachment

Método que recebe como parâmetro uma String contendo o nome de um ficheiro e que o adiciona ao *mail* a ser enviado.

Classe MyApplication

```
public class MyApplication extends Application {

    private static Context context;

    public void onCreate() {...}

    public static Context getAppContext() {...}
}
```

Classe que estende a classe Application e que tem como atributo estático um Context.

Método onCreate

Método que atribui o valor da aplicação ao atributo estático Context.

Método getAppContext

Método estático que retorna o atributo Context da classe.

Classe PasswordEncrypt

```
public class PasswordEncrypt {

    private Encryption encryption;

    public String getEncrypted(String pass){...}


}
```

Classe que tem como atributo um objecto do tipo Encryption.

Método getEncrypted

Método que recebe como argumento uma String que contem uma *password*. Devolve essa String depois de a encriptar fazendo uso do método estático getDefault da classe Encryption.

Classe PasswordValidator

```java
public class PasswordValidator{

    private Pattern pattern;
    private Matcher matcher;

    private static final String PASSWORD_PATTERN =
            "^(?=.*[0-9])(?=.*[a-z])(?=.*[A-Z])(?=.*[@#$%^&+=!_?«»',.-])(?=\\S+$).{8,}$";

    public PasswordValidator(){...}

    /**
     * Validate password with regular expression
     * @param password password for validation
     * @return true valid password, false invalid password
     */
    public boolean validate(final String password){...}
}
```

Tem como atributos um Pattern e um Matcher. Contem uma constante estática do tipo String que é uma RegularExpression.

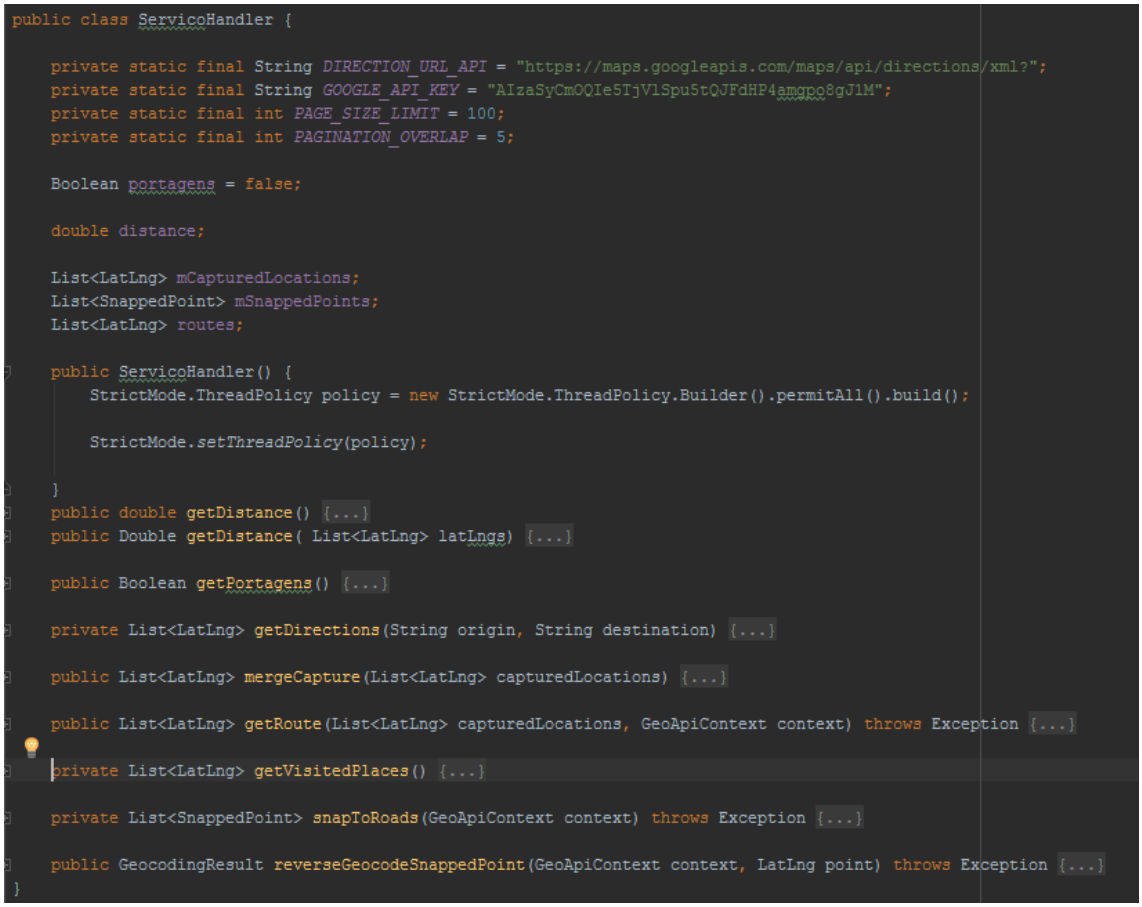
Método PasswordValidator

Construtor por omissão que compila para um padrão (Pattern) a RegularExpression.

Método validate

Recebe como parâmetro uma String contendo uma *password* e retorna true se essa password for validada pela RegularExpression.

Classe ServicoHandler



```
public class ServicoHandler {

    private static final String DIRECTION_URL_API = "https://maps.googleapis.com/maps/api/directions/xml?";
    private static final String GOOGLE_API_KEY = "AIzaSyCmOQIe5TjVlSpu5tQJFdHP4amgpo8gJlM";
    private static final int PAGE_SIZE_LIMIT = 100;
    private static final int PAGINATION_OVERLAP = 5;

    Boolean portagens = false;

    double distance;

    List<LatLng> mCapturedLocations;
    List<SnappedPoint> mSnappedPoints;
    List<LatLng> routes;

    public ServicoHandler() {
        StrictMode.ThreadPolicy policy = new StrictMode.ThreadPolicy.Builder().permitAll().build();

        StrictMode.setThreadPolicy(policy);

    }
    public double getDistance() {...}
    public Double getDistance( List<LatLng> latLngs) {...}

    public Boolean getPortagens() {...}

    private List<LatLng> getDirections(String origin, String destination) {...}

    public List<LatLng> mergeCapture(List<LatLng> capturedLocations) {...}

    public List<LatLng> getRoute(List<LatLng> capturedLocations, GeoApiContext context) throws Exception {...}

    private List<LatLng> getVisitedPlaces() {...}

    private List<SnappedPoint> snapToRoads(GeoApiContext context) throws Exception {...}

    public GeocodingResult reverseGeocodeSnappedPoint(GeoApiContext context, LatLng point) throws Exception {...}
}
```

Classe que reúne uma série de métodos que são usados no cálculo dos trajetos, distancias percorridas e na interpolação dos pontos capturados para as estradas mais próximas.

Método ServicoHandler

Construtor por omissão que faz o override das ThreadPolicy de modo a que possam existir *network operations* na *MainThread.*

Método getDistance

Método polimórfico.
A versão sem parâmetros arredonda o atributo da classe Distance para duas casas decimais.
A versão que recebe como parâmetro uma List<LatLng> calcula a distancia entre o primeiro e o ultimo ponto da lista usando o método estático computeDistanceBetween da classe SphericalUtil.

Método getDirections

Método privado que recebe como parâmetros uma String contendo as coordenadas de origem e outra String contendo as coordenadas de destino de um trajeto. Este método faz um pedido web à Direction API. Com a resposta atribui um valor ao atributo portagens e ao atributo distance. Chama o método parseDirections da classe XMLHandler para converter a resposta ao pedido para uma List<LatLng> devolvendo essa lista.

Método mergeCapture

Método que recebe uma List<LatLng> que contem os pontos capturados ao longo do trajeto. Invoca o método da mesma classe getDistance para obter a distância percorrida durante o serviço. Invoca o método getDirections, duas vezes, primeiro para obter o trajeto entre a praça de táxis e o início do serviço e a segunda vez para obter o trajeto entre o final do serviço e a praça de táxis. Faz a concatenação das listas de pontos de maneira a armazenar todo o trajeto desde a praça de táxis até à praça de táxis. Soma as distâncias obtidas nas várias chamadas ao getDirections. Retorna a List<LatLng> que concatenou.

Método getRoute

Recebe como parâmetros uma List<LatLng> que contem todo o trajeto do serviço e um GeoApiContext. Chama o método SnapToRoads para fazer a interpolação dos pontos para a estrada mais próxima. Elimina os pontos que tem a mesma *placeId* e devolve essa mesma List<LatLng>.

Método getVisitedPlaces

Método que percorre a lista que contém os pontos interpolados e povoa uma nova lista com apenas um ponto de cada *placeId*. Retorna essa nova lista.

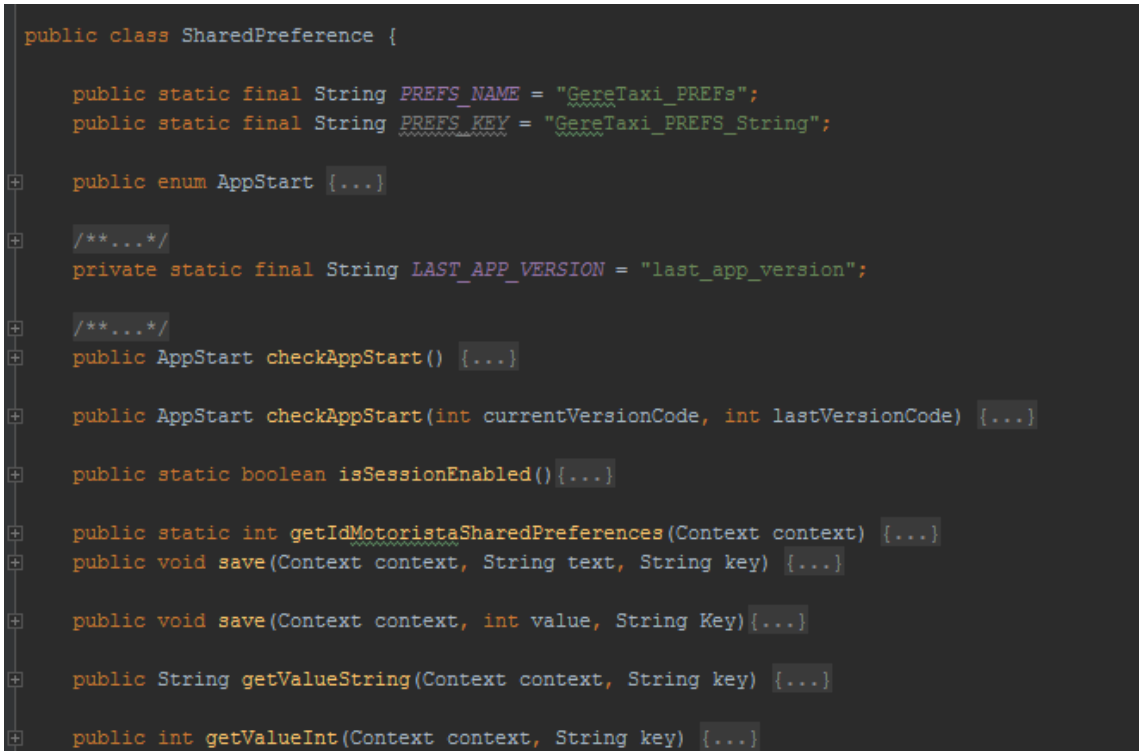
Método snapToRoads

Recebe como parâmetro um GeoApiContext faz a paginação da lista de pontos que contém o trajeto para puder fazer diversos pedidos à Roads Api. Ou seja, como esta Api tem um limite de 100 pontos por pedido ele divide a lista em sublistas, cada uma contendo no máximo 100 pontos. Depois de faz a junção dos diferentes pedidos numa única lista. Retorna essa lista.

Método reverseGeocodeSnappedPoint

Recebe como parâmetros um GeoApiContext e um ponto LatLng. Usa o método estático reverseGeocode da Geocoding Api para obter um GeocodingResult correspondente a esse ponto. Devolve esse GeocodingResult.

Classe SharedPreference



```
public class SharedPreference {

    public static final String PREFS_NAME = "GereTaxi_PREFs";
    public static final String PREFS_KEY = "GereTaxi_PREFS_String";

    public enum AppStart {...}

    /**...*/
    private static final String LAST_APP_VERSION = "last_app_version";

    /**...*/
    public AppStart checkAppStart() {...}

    public AppStart checkAppStart(int currentVersionCode, int lastVersionCode) {...}

    public static boolean isSessionEnabled(){...}

    public static int getIdMotoristaSharedPreferences(Context context) {...}
    public void save(Context context, String text, String key) {...}

    public void save(Context context, int value, String Key){...}

    public String getValueString(Context context, String key) {...}

    public int getValueInt(Context context, String key) {...}
```

Método checkAppStart

Método polimórfico, a versão sem argumentos verifica qual é a versão atual da aplicação. Lê a versão que foi guardada nas SharedPreferences, na ultima vez que a aplicação foi iniciada. Chama a sua outra versão para comparar as versões e guarda nas SharedPreferences o valor atual.
A versão que recebe como argumentos dois inteiros, compara esses mesmos devolvendo um objeto do tipo Enum AppStart first time se é a primeira instalação, first time version se é uma atualização e normal se é um inicio da aplicação normal.

Método isSessionEnabled

Método estático que retorna o valor associado à chave SESSION guardado nas SharedPreferences.

Método getIdMotoristaSharedPreferences

Método estático que retorna o valor associado à chave ID_MOTORISTA guardado nas SharedPreferences.

Método save

Método Polimórfico, uma versão recebe um Context e duas Strings. Guarda o valor da String text associado à chave KEY nas SharedPreferences. A outra versão recebe um Context um int e uma String e guarda o valor do int value associado à String Key.
.

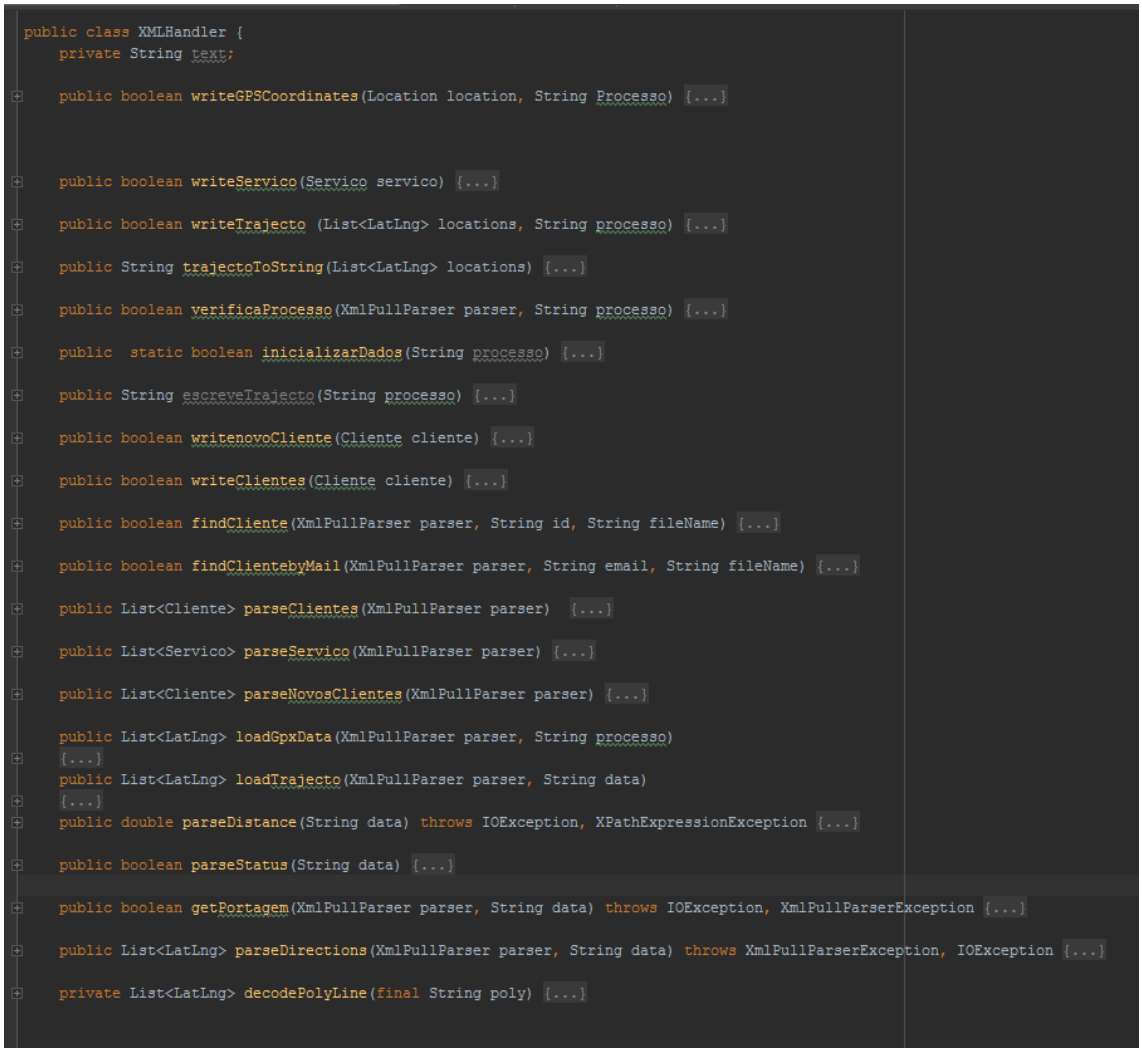
Método getValueString

Método que recebe um Context e uma String key e devolve a String associada à chave que recebeu.

Método getValueInt

Método que recebe um Context e ums String Key e devolve o int associado à chave que recebeu.

Classe XMLHandler



```
public class XMLHandler {
    private String text;

    public boolean writeGPSCoordinates(Location location, String Processo) {...}



    public boolean writeServico(Servico servico) {...}

    public boolean writeTrajecto (List<LatLng> locations, String processo) {...}

    public String trajectoToString(List<LatLng> locations) {...}

    public boolean verificaProcesso(XmlPullParser parser, String processo) {...}

    public  static boolean inicializarDados(String processo) {...}

    public String escreveTrajecto(String processo) {...}

    public boolean writenovoCliente(Cliente cliente) {...}

    public boolean writeClientes(Cliente cliente) {...}

    public boolean findCliente(XmlPullParser parser, String id, String fileName) {...}

    public boolean findClientebyMail(XmlPullParser parser, String email, String fileName) {...}

    public List<Cliente> parseClientes(XmlPullParser parser)  {...}

    public List<Servico> parseServico(XmlPullParser parser) {...}

    public List<Cliente> parseNovosClientes(XmlPullParser parser) {...}

    public List<LatLng> loadGpxData(XmlPullParser parser, String processo)
    {...}
    public List<LatLng> loadTrajecto(XmlPullParser parser, String data)
    {...}
    public double parseDistance(String data) throws IOException, XPathExpressionException {...}

    public boolean parseStatus(String data) {...}

    public boolean getPortagem(XmlPullParser parser, String data) throws IOException, XmlPullParserException {...}

    public List<LatLng> parseDirections(XmlPullParser parser, String data) throws XmlPullParserException, IOException {...}

    private List<LatLng> decodePolyLine(final String poly) {...}
```

Classe que contem uma série de métodos para ler e escrever ficheiros e Strings usando XML.

Método writeServico

Método que recebe um objeto do tipo Servico como parâmetro e escreve num ficheiro XML e retorna true se for bem-sucedido, false se não for bem-sucedido.

Método writeTrajeto

Método que recebe uma List<LatLng> e uma String processo como parâmetro, escreve o trajeto num ficheiro XML e devolve true se for bem sucedido, false se não for bem sucedido

Método trajetoToString

Método que recebe uma List<LatLng> com coordenadasGPS e converte para uma String, retorna a String final com todas as coordenadas.

Método verificaProcesso

Método que recebe como parâmetros um XmlPullParser e uma String processo. Este método percorre o ficheiro de serviços XML e verifica se o processo existe no ficheiro. Retorna true se existir o processo.

Método writeNovoCliente

Método que recebe um objeto do tipo Cliente como parâmetro e que escreve os dados desse mesmo objeto num ficheiro XML, retorna true se for bem sucedido ou false se ocorrer algum erro na escrita.

Método writeCliente

Método que recebe um objeto do tipo Cliente como parâmetro e que escreve os dados desse mesmo objeto num ficheiro XML, retorna true se for bem sucedido ou false se ocorrer algum erro na escrita

Método findCliente

Método que recebe como parâmetros um XmlPullParser, uma String id e uma String fileName. Este método procura por um determinado cliente num ficheiro XML através do seu atributo id. Retorna true se encontrar.

Método findClienteByMail

Método que recebe como parâmetros um XmlPullParser, uma String email e uma String fileName. Este método procura por uma determinado cliente num ficheiro XML através do seu atributo email. Retorna true se encontrar.

Método parseClientes

Método que recebe como parâmetro um XmlPullParser e que é responsável por listar todos os clientes existes no ficheiro clientes.xml. Retorna uma List<Cliente> com todos os clientes encontrados.

Método parseServico

Método que recebe como parâmetro um XmlPullParser e que é responsável por listar todos os serviços existentes no ficheiro serviços.xml. Retorna uma List<Servico> com todos os serviços encontrados.

Método parseNovosClientes

Método que recebe como parâmetro um XmlPullParser e que é responsável por listar todos os serviços existentes no ficheiro novosclientes.xml. Retorna uma List<Cliente> com todos os novos clientes encontrados.

Método loadGpxData

Método que recebe como parâmetros um XmlPullParser e uma String processo que lê um ficheiro XML e retorna uma List<LatLng> com todas as coordenadas GPS de um determinado serviço com um determinado numero de processo.

Método loadTrajecto

Método que recebe como parâmetros um XmlPullParser e uma String data que lê um ficheiro XML e retorna uma List<LatLng> com todas as coordenadas GPS de um determinado ficheiro.

Método parseDistance

Método que recebe como parâmetro uma String data que contem a resposta em XML do pedido efetuado à Directions Api. Transforma a String num InputStream usando o método toInputStream da classe IOUtils do Apache Commons. Usa uma instância da classe XPath para ir buscar ao InputStream o valor que está contido no Node "/DirectionsResponse/route/leg/distance/value", valor esse que corresponde à distância total do trajeto devolvido pela Directions Api.

Método parseStatus

Método que recebe como parâmetro uma String data que contem a resposta em XML do pedido efetuado à Directions Api. Transforma a String num InputStream usando o método toInputStream da classe IOUtils do Apache Commons. Usa uma instância da classe XPath para ir buscar ao InputStream o valor que está contido no Node "/DirectionsResponse/status", valor esse indica se o trajeto é válido ou não, ou seja, se for OK o trajeto é valido, se for ZERO RESULTS o trajeto é inválido.

Método getPortagem

Método que recebe como parâmetros um XmlPullParser e uma String data que contem a resposta do pedido à Directions Api. Transforma a String num InputStream usando o método toInputStream da classe IOUtils do Apache Commons. Faz o parse desse InputStream procurando a TAG html_instructions e se essa tag contiver o valor portagem ou o valor toll retorna true.

Método parseDirections

Método que recebe como parâmetros um XmlPullParser e uma String data que contem a resposta do pedido à Directions Api. Transforma a String num InputStream usando o método toInputStream da classe IOUtils do Apache Commons. Faz o parse desse InputStream procurando a TAG overview_polyline e de seguida procura a TAG points que é a TAG que contem todos os pontos do trajeto devolvido pela resposta da Directions Api em forma codificada. O método em seguida invoca o método decodePolyline e retorna a List<LatLng> devolvida por esse método.

Método decodePolyline

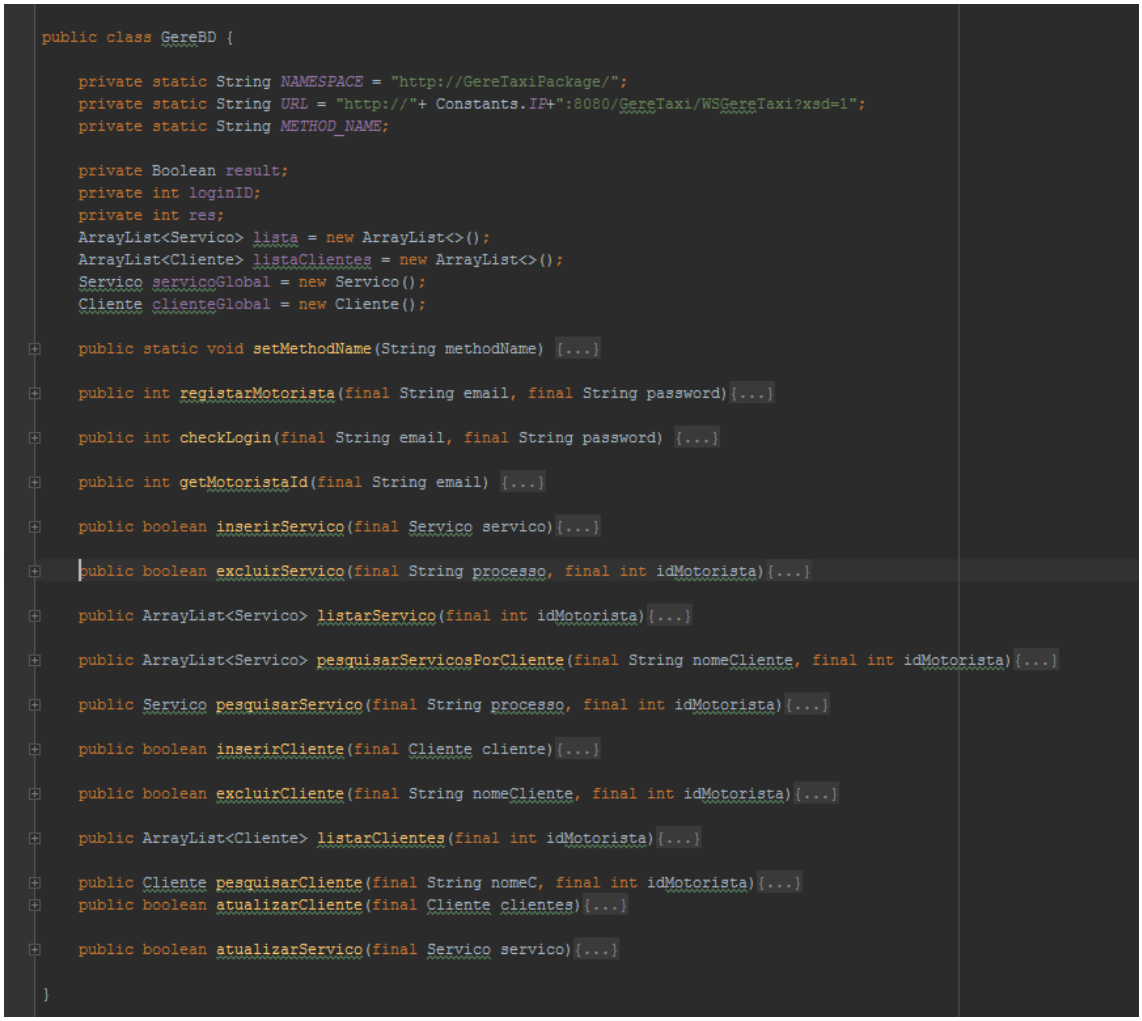
Método que recebe uma String poly que contem todos os pontos do trajeto devolvido pela resposta da Directions Api em forma codificada e que procede à sua descodificação usando um algoritmo fornecido pela Google.

Método inicializarDados

Método estático que recebe como parâmetro uma String processo e que remove o ficheiro XML com o nome do processo. Retorna true se for bem sucedido.

Package WebServiceClass

Classe GereBD



```
public class GereBD {

    private static String NAMESPACE = "http://GereTaxiPackage/";
    private static String URL = "http://"+ Constants.IP+":8080/GereTaxi/WSGereTaxi?xsd=1";
    private static String METHOD_NAME;

    private Boolean result;
    private int loginID;
    private int res;
    ArrayList<Servico> lista = new ArrayList<>();
    ArrayList<Cliente> listaClientes = new ArrayList<>();
    Servico servicoGlobal = new Servico();
    Cliente clienteGlobal = new Cliente();

    public static void setMethodName(String methodName) {...}

    public int registarMotorista(final String email, final String password){...}

    public int checkLogin(final String email, final String password) {...}

    public int getMotoristaId(final String email) {...}

    public boolean inserirServico(final Servico servico){...}

    public boolean excluirServico(final String processo, final int idMotorista){...}

    public ArrayList<Servico> listarServico(final int idMotorista){...}

    public ArrayList<Servico> pesquisarServicosPorCliente(final String nomeCliente, final int idMotorista){...}

    public Servico pesquisarServico(final String processo, final int idMotorista){...}

    public boolean inserirCliente(final Cliente cliente){...}

    public boolean excluirCliente(final String nomeCliente, final int idMotorista){...}

    public ArrayList<Cliente> listarClientes(final int idMotorista){...}

    public Cliente pesquisarCliente(final String nomeC, final int idMotorista){...}
    public boolean atualizarCliente(final Cliente clientes){...}

    public boolean atualizarServico(final Servico servico){...}

}
```

Faz os pedidos e o respetivo tratamento ao Web Service usando a biblioteca KSOAP2.

Método registarMotorista

Tem como parâmetros uma String email e uma String password. Faz um pedido Soap ao Web Service com o objetivo de registar um objeto do tipo Motorista na base de dados. Retorna -2 se o Web Service estiver offline, -1 se houve erro e caso haja sucesso na inserção.

Método checkLogin

Tem como parâmetros uma String email e uma String password. Faz um pedido Soap ao Web Service com o objetivo de verificar as credenciais do Motorista. Retorna -2 se o serviço estiver offline, -1 se o utilizador não está registado, 0 se o utilizador está registado mas a password está incorreta e 1 se houve sucesso na autenticação.

Método getMotoristaId

Tem como parâmetro uma String email. Faz um pedido Soap ao web service com o objetivo de receber o id o Motorista correspondente ao email. Retorna -2 se o web service estiver offline, -1 se o email não estiver registado e retorna o id do Motorista se o email existir.

Método inserirServico

Tem como parâmetro um objeto do tipo Servico. Faz um pedido Soap ao web service com o objetivo de inserir o serviço na base de dados. Retorna true se for bem sucedido.

Método excluirServico

Tem como parâmetros uma String processo e um int idMotorista. Faz um pedido Soap ao web service com o objetivo de iliminar o serviço na base de dados. Retorna true se for bem sucedido.

Método listarServico

Tem como parâmetro um int idMotorista. Faz um pedido Soap ao web service com o objetivo de listar todos os serviços na base de dados correspondentes ao idMotorista que recebe. Retorna uma List<Servicos> se for bem sucedido.

Método pesquisarServicosPorCliente

Tem como parâmetros uma String nomeCliente e um int idMotorista. Faz um pedido Soap ao web service com o objetivo de listar todos os serviços na base de dados correspondentes ao idMotorista e ao nomeCliente que recebe. Retorna uma List<Servicos> se for bem sucedido.

Método pesquisarServico

Tem como parâmetros uma String processo e um int idMotorista. Faz um pedido Soap ao web service com o objetivo de pesquisar um serviço na base de dados correspondente à String processo e ao idMotorista que recebe. Retorna um objeto do tipo Servico se for bem sucedido.

Método inserirCliente

Tem como parâmetro um objeto do tipo Cliente. Faz um pedido Soap ao web service com o objetivo de inserir o cliente na base de dados. Retorna true se for bem sucedido.

Método excluirCliente

Tem como parâmetros uma String nomeCliente e um int idMotorista. Faz um pedido Soap ao web service com o objetivo de excluir o registo na base de dados correspondentes ao nomeCliente e idMotorista que recebe. Retorna true se for bem sucedido.

Método listarClientes

Tem como parâmetro um int idMotorista. Faz um pedido Soap ao web service com o objetivo de listar todos os clientes que se encontram na base de dados correspondentes ao idMotorista que recebe. Retorna uma List<Cliente> se for bem sucedido.

Método pesquisarCliente

Tem como parâmetro uma String nomeC e um int idMotorista. Faz um pedido Soap ao web service com o objetivo de listar todos os clientes registados na base de dados correspondentes ao nomeC e ao idMotorista que recebe. Retorna um objeto Cliente se for bem sucedido.

Método atualizarCliente

Tem como parâmetro um objeto Cliente. Faz um pedido Soap ao web service com o objetivo de atualizar o registo correspondente ao cliente na base de dados. Retorna true se for bem sucedido.

Método atualizarServico

Tem como parâmetro um objeto Servico. Faz um pedido Soap ao web service com o objetivo de atualizar o registo correspondente ao servico na base de dados. Retorna true se for bem sucedido.

Package TarefasAssincronas

Este package é constituído por duas classes que contêm tarefas assíncronas.

Classe TarefaMail

```
public class TarefaMail {

    public AsyncTask<String, Void, Void> mailInfo = new AsyncTask<String, Void, Void>() {...};

    private boolean mailRegisto(String email, String pass) {...}
}
```

Classe contem uma tarefa assíncrona para enviar um email.

Método mailRegisto

Método que recebe como parâmetros a String email e a String pass. Instancia um objeto do tipo mail e envia o mail de confirmação de registo contendo os dados do registo. Retorna true se for bem sucedido.

AssyncTask<String,Void,Void> mainInfo

Executa o método mailRegisto de forma assíncrona.

Classe TarefaSincronizar

```
public class TarefaSincronizar {

    GeoApiContext mContext;
    File file;
    List<LatLng> mCapturedLocations;
    List<Servico> servicos;
    XMLHandler parser = new XMLHandler();

    public AsyncTask<Void, Void, Void> sincronizar = new AsyncTask<Void, Void, Void>() {...};

    private void sincronzaServicos() {...}

    private Servico defineServico(Servico servico) {...}

    private void sincronizaClientes() {...}
}
```

Classe que contem uma tarefa assíncrona e vários métodos privados com o objetivo de sincronizar os dados guardados no modo offline com os dados guardados na base de dados.

Método sincronizaServicos

Método que verifica se existe no sdcard um ficheiro chamado serviços.xml e caso exista faz uso do método parseServico da classe XMLHandler para criar uma lista de serviços. Se essa lista não estiver vazia é invocado para cada elemento da lista defineServico para fazer os diversos requests à Directions e Roads Api respetivamente

edepois insere o serviço na base de dados. Se essa inserção for bem sucedida remove o serviço da lista e no final se a lista estiver vazia o ficheiro é apagado.

Método defineServico

Recebe como parâmetro um objecto do tipo Servico. Cria uma lista de pontos com o trajeto capturado usando o método loadTrajeto da classe XMLHandler. Obtem o GeocodingResult da origem e do destino. Acrescenta ao trajeto a parte da praça de táxis até ao inicio e do final até à praça de táxis fazendo uso do método mergeCapture da class XMLHandler. Elimina da lista e faz a interpolação para a estrada mais próxima usando o método getRoute da class XMLHandler. Atribui um valor à propriedade trajeto fazendo uso do método rajetoToString da class XMLHandler. Atribui um valor à propriedade distance fazendo uso do método getDistance da classe XMLHandler. Também atribui valores à propriedade idMotorista gravado nas SharedPreferences. Retorna o serviço.

Método sincronizaClientes

Método que verifica se existe no sdcard um ficheiro chamado novosclientes.xml e caso exista faz uso do método parseNovosClientes da class XMLHandler para criar uma lista de clientes. Se essa lista não estiver vazia é definido o idMotorista de cada cliente que posteriormente são inseridos na base de dados. Se essa inserção for bem sucedida remove o cliente da lista e no final se a lista estiver vazia o ficheiro é apagado.

Package Fragments

Conjunto de classes que implementam DialogFragments.

Classe DatePickerFragment

```
public class DatePickerFragment extends DialogFragment{

    private DatePickerDialog.OnDateSetListener onDateSetListener;

    public DatePickerFragment() {
    }

    //LISTENER PARA A DATA
    public void setOnDateSetListener(DatePickerDialog.OnDateSetListener onDateSetListener) {...}

    @Override
    public Dialog onCreateDialog(Bundle savedInstanceState) {...}
```

Classe que estende a classe DialogFragment e que é responsável por ativar um dialog com um calendário.

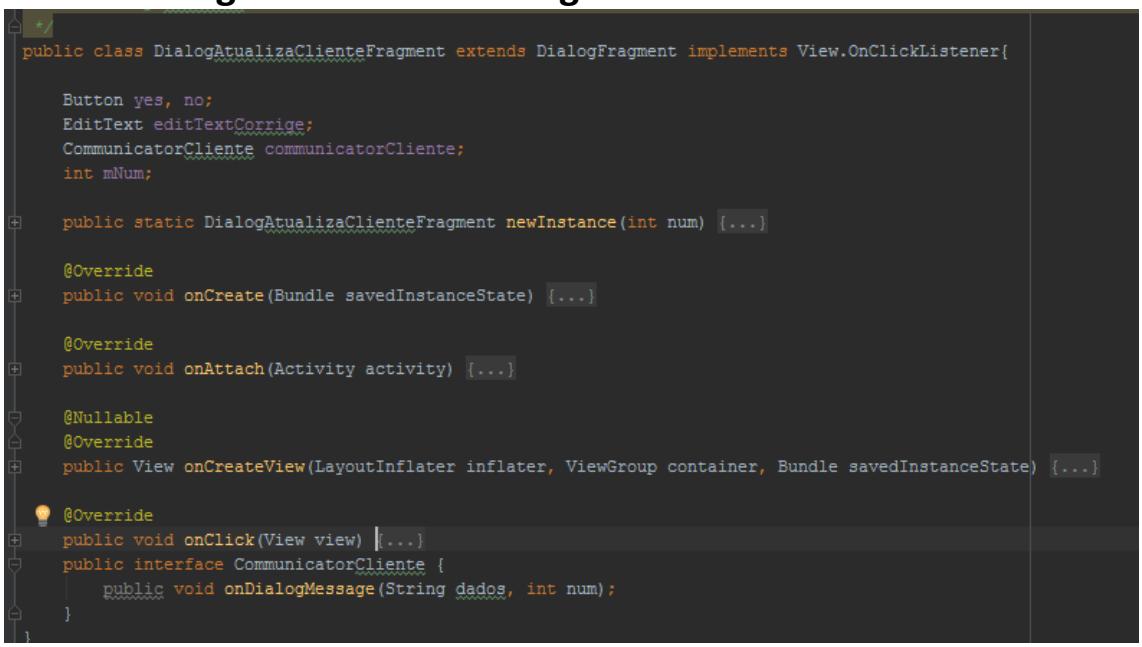
Método setOnDateSetListener

É um listener para a data.

Método onCreateDialog

Define um objeto do tipo Calendar com dia, mês e ano com o titulo Data do Serviço.

Classe DialogAtualizaClienteFragment



```
*/
public class DialogAtualizaClienteFragment extends DialogFragment implements View.OnClickListener{

    Button yes, no;
    EditText editTextCorrige;
    CommunicatorCliente communicatorCliente;
    int mNum;

    public static DialogAtualizaClienteFragment newInstance(int num) {...}

    @Override
    public void onCreate(Bundle savedInstanceState) {...}

    @Override
    public void onAttach(Activity activity) {...}

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {...}

    @Override
    public void onClick(View view) {...}
    public interface CommunicatorCliente {
        public void onDialogMessage(String dados, int num);
    }
}
```

Classe responsável pelo DialogFragment que serve para atualizar os dados de um cliente.

Método newInstance

Construtor estático da classe que recebe um int num como parâmetro e que o envia via Bundle para o método onCreate.

Método onCreate

Override do método onCreate que acrescenta a receção do inteiro enviado pelo método newInstance.

Método onAttach

Override que instancia a interface CommunicatorCliente.

Método onCreateView

Responsavel pelo inflate e pelas linkagens dos elementos de layout que se encontram no ficheiro corrige_dados_s_dialog.

Método onClick

Método que faz o listening dos botões do DialogFragment. Se for clicado o botão Confirmar ele verifica o atributo mNum da classe se esse valor for igual a três ele valida os dados usando método isValidEmail da classe Helper. Coloca os dados na interface communicatorCliente. Se os mNum for maior ou igual que cinco ele valida os dados usando o método integerTryParse da class Helper e coloca-os na interface communicatorCliente. Se for for clicado o botão cancelar apenas se faz o dismiss do dialog.

Interface CommunicatorCliente

Método onDialogMessage

Método que recebe uma String dados e um int mNum.

DialogCorrigeDadosFragment

```
public class DialogCorrigeDadosFragment extends DialogFragment implements View.OnClickListener {

    Button yes, no;
    EditText editTextCorrige;
    TextView txt_dia;
    CommunicatorCorrige communicatorCorrige;
    int mNum;


    public static DialogCorrigeDadosFragment newInstance(int num) {...}
    @Override
    public void onAttach(Activity activity) {...}

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {...}

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {...}

    @Override
    public void onClick(View view) {...}

    public interface CommunicatorCorrige {
        public void onDialogMessage(String dados, int num);
    }
}
```

Classe responsável pelo DialogFragment que serve para corrigir os dados de um serviço.

Método newInstance

Construtor estático da classe que recebe um int num como parâmetro e que o envia via Bundle para o método onCreate.

Método onCreate

Override do método onCreate que acrescenta a receção do inteiro enviado pelo método newInstance.

Método onAttach

Override que instancia a interface CommunicatorCorrige.

Método onCreateView

Responsavel pelo inflate e pelas linkagens dos elementos de layout que se encontram no ficheiro corrige_dados_s_dialog.

Método onClick

Método que faz o listening dos botões do DialogFragment. Se for clicado o botão Confirmar ele verifica o atributo mNum. Se esse valor for menor do que seis ele coloca-o diretamente na interface CommunicatorCorrige. Se o valor for igual a seis, correspondente ao atributo número de passageiros ele valida se é um inteiro utilizando o método integerTryParse da class Helper. Depois faz o parse propriamente dito e valida se é um numero compreendido entre 1 e 8 colocando-o depois na interface communicatorCorrige. Se tem o valor superior a seis, correspondente ao valor das portagens ou das horas de espera, verifica se o valor é um Double fazendo uso do método doubleTryParse da classe Helper. Se for validado faz o parse propriamente dito e verifica se é positivo, se for validado coloca-o na interface CommunicatorCorrige.

Interface CommunicatorCorrige

Método onDialogMessage

Método que recebe uma String dados e um int mNum.

DialogCustoPortagensFragment

```
public class DialogCustoPortagemFragment extends DialogFragment implements View.OnClickListener {

    Button yes, no;
    EditText editTextPortagem;
    Communicator communicator;

    @Override
    public void onAttach(Activity activity) {...}

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, ViewGroup container, Bundle savedInstanceState) {...}

    @Override
    public void onClick(View view) {...}
    public interface Communicator {...}


}
```

Classe responsável pelo DialogFragment que serve para corrigir os dados de um serviço.

Método onCreate

Override do método onCreate que acrescenta a receção do inteiro enviado pelo método newInstance.

Método onAttach

Override que instancia a interface Communicator.

Método onCreateView

Responsavel pelo inflate e pelas linkagens dos elementos de layout que se encontram no ficheiro custom_portagem_dialog_fragment.

Método onClick

Método que faz o listening dos botões do DialogFragment. Se for clicado o botão Confirmar coloca os dados introduzidos pelo utilizador no Communicator e atribui o valor 1 à propriedade confirm para indicar que os dados foram realmente introduzidos. No botão Cancelar inicializa o valor portagens a 0.0 e coloca o valor confirm a 0.

Interface Communicator

Método onDialogMessage

Método que recebe uma String portagem e um int confirm.

DialogSpinnerFragment

```
public class DialogSpinnerFragment extends DialogFragment implements View.OnClickListener {

    Button yes, no;
    Spinner spinner;
    CommunicatorCorrige communicatorCorrige;
    int mNum;
    XMLHandler handler;
    GereBD gereBD;
    ArrayList<String> clientesSpinner = new ArrayList<>();
    private String nomeCliente;

    public static DialogSpinnerFragment newInstance(int num) {...}
    @Override
    public void onAttach(Activity activity) {...}

    @Override
    public void onCreate(@Nullable Bundle savedInstanceState) {...}

    @Nullable
    @Override
    public View onCreateView(LayoutInflater inflater, @Nullable ViewGroup container, @Nullable Bundle savedInstanceState) {...}

    @Override
    public void onClick(View view) {...}

    public interface CommunicatorCorrige {...}
}
```

Classe responsável pelo DialogFragment que serve para corrigir os dados de um serviço.

Método newInstance

Construtor estático da classe que recebe um int num como parâmetro e que o envia via Bundle para o método onCreate.

Método onCreate

Override do método onCreate que acrescenta a receção do inteiro enviado pelo método newInstance.

Método onAttach

Override que instancia a interface CommunicatorCorrige.

Método onCreateView

Responsavel pelo inflate e pelas linkagens dos elementos de layout que se encontram no ficheiro corrige_dados_s_dialog. Verifica se o dispositivo está conectado a alguma rede usando o método isNetworkAvailable da class Helper. Se estiver cria uma lista de clientes usando o método listarClientes da classe GereBD e para cada um dos elementos da lista adiciona o valor da sua propriedade nome a um ArrayList<String>. Caso não esteja registado em nenhuma rede cria uma lista de clientes usando o método parseCliente da classe XMLHandler, ou seja, cria a lista de clientes guardados localmente no dispositivo e para cada um dos elementos desta lista adiciona o valor da propriedade nome a um ArrayList<String>. Finalmente, coloca o ArrayList<String> num ArrayAdapter<String> de modo a mostrar essa lista num Spinner.
O Spinner tem associado o método **setOnItemSelectedListener** que atribui o valor selecionado ao atributo nomeCliente.

Método onClick

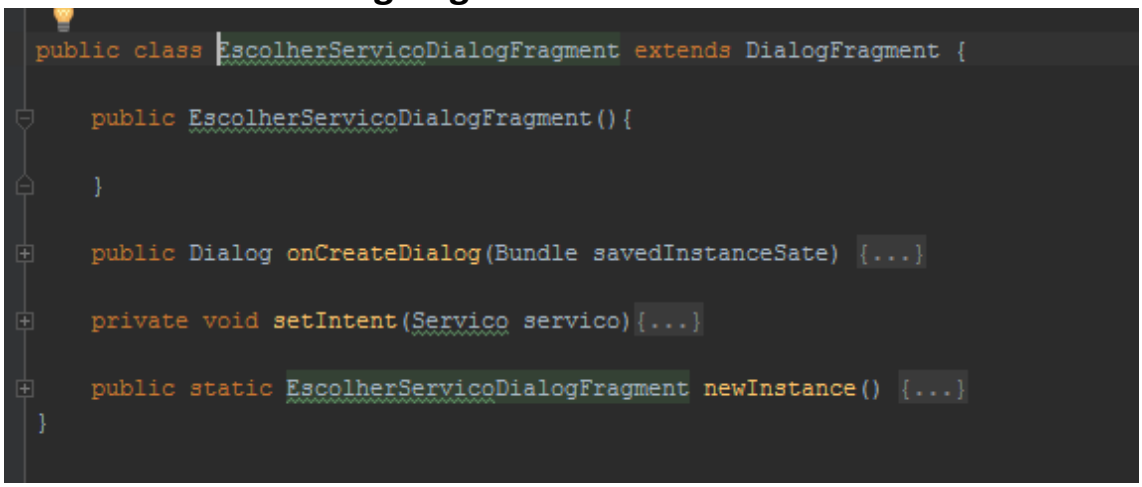
Se for clicado o botão confirmar coloca os dados introduzidos pelo utilizador na interface CommunicatorCorrige.
Se for clicado o botão cancelar faz o dismiss do Dialog.

Interface CommunicatorCorrige

Método onDialogMessage

Método que recebe uma String dados e um int num.

EscolherServicoDialogFragment



Classe responsável pelo DialogFragment que serve escolher o tipo de serviço que se vai iniciar.

Construtor por omissão

Vazio.

Método newInstance

Método estático que instancia a classe;

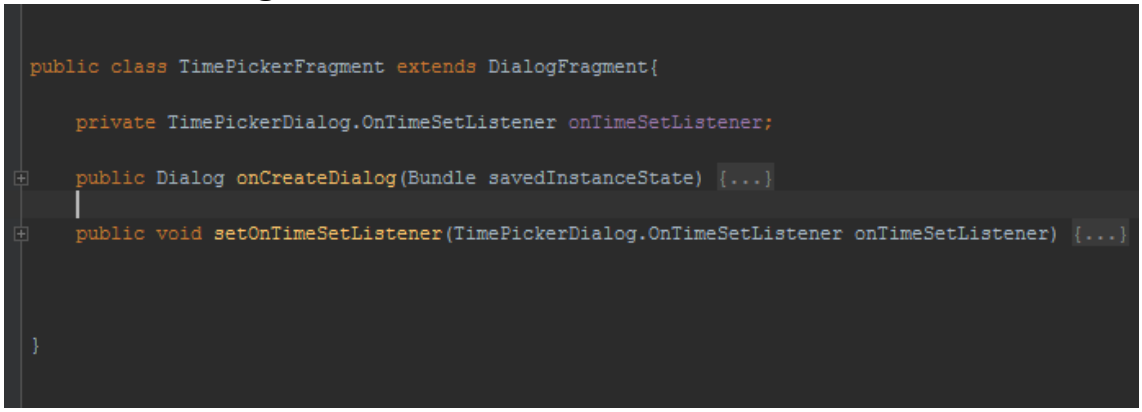
Método onCreateDialog

Método que cria um AlertDialog. Atribui-lhe o titulo definico pela @String/escolhe_tipo_servico e povoado pelo conteúdo do @array/serviço_tipo_array Instancia um onClickListener que define o valor do atributo tipoDeServico dependendo qual o valor escolhido e faz um Intent para a activity IniciaServicoActivity fazendo uso do método setIntent.

Método setIntent

Recebe como parâmetro um objeto do tipo Serviço, faz um Intent para IniciaServicoActivity colocando como extra o serviço.

TimePickerFragment



Classe responsável por apresentar um Dialog que permite ao utilizador registar a hora de início de um serviço.

Método onCreateDialog

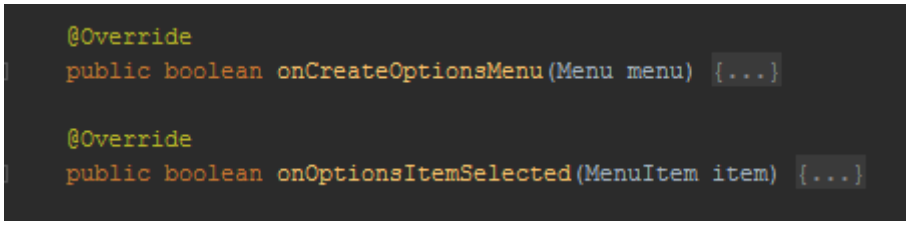
Instancia um objeto do tipo Calendar com horas e minutos.
Instancia também um objeto da Classe TimePickerDialog.
Faz um setTitle para Hora de Inicio.

Método setOnTimeSetListener

Instancia o valor selecionado pelo utilizador.

Package Activities

Menu



Todas as activities exceto a Coordenadas instanciam o menu que tem os seguintes métodos:

onCreateOptionsMenu

Faz o inflater do menu usando o layout main_menu .

Método onOptionsItemSelected

Faz as ações correspondestes às escolhas do utilizador que podem ser:
Logout, em que são apagados os valores que estão armazenados nas SharedPreferences relativamente ao email, password, idMotorista, SESSION e validade e direciona o utilizador para a LoginActivity. Caso as escolha seja definições o utilizador é direcionado para a CoordenadasActivity. Caso seja escolha Menu Principal o utilizador é direcionado para a MenuActivity.

MainActivity

```
public class MainActivity extends AppCompatActivity {

    ArrayList<Servico> lista;
    Boolean resultado;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    @Override
    public boolean onCreateOptionsMenu(Menu menu) {...}
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {...}

    public void onClickMain(View v) {...}

    public void onClickbuttonLoginMain(View v) {...}


}
```

É a launch activity.

Método onCreate

Faz a linkagem com os elementos do layout activity_main.
Verifica se é a primeira vez que está a correr a aplicação ou se é a primeira que está a correr a ultima versão da aplicação, ou se é uma utilização normal.
Nos dois primeiros casos envia o utilizador para a StorePreferencesActivity. Se for normal usa o método atemptLogin da class Helper . Se o método devolver true envia o utilizador para a MenuActivity e verifica se tem acesso à rede usando o método isNetworkAvailable da classe Helper, se tal acontecer inicia a tarefa assíncrona sincronizar da class TarefaSincronizar. Caso não o método atemptLogin devolva falso e tenha acesso à rede, envia o utilizador para a LoginActivity. Caso não tenha rede informa o utilizador que está em modo offline e envia-o para a MenuActivity.

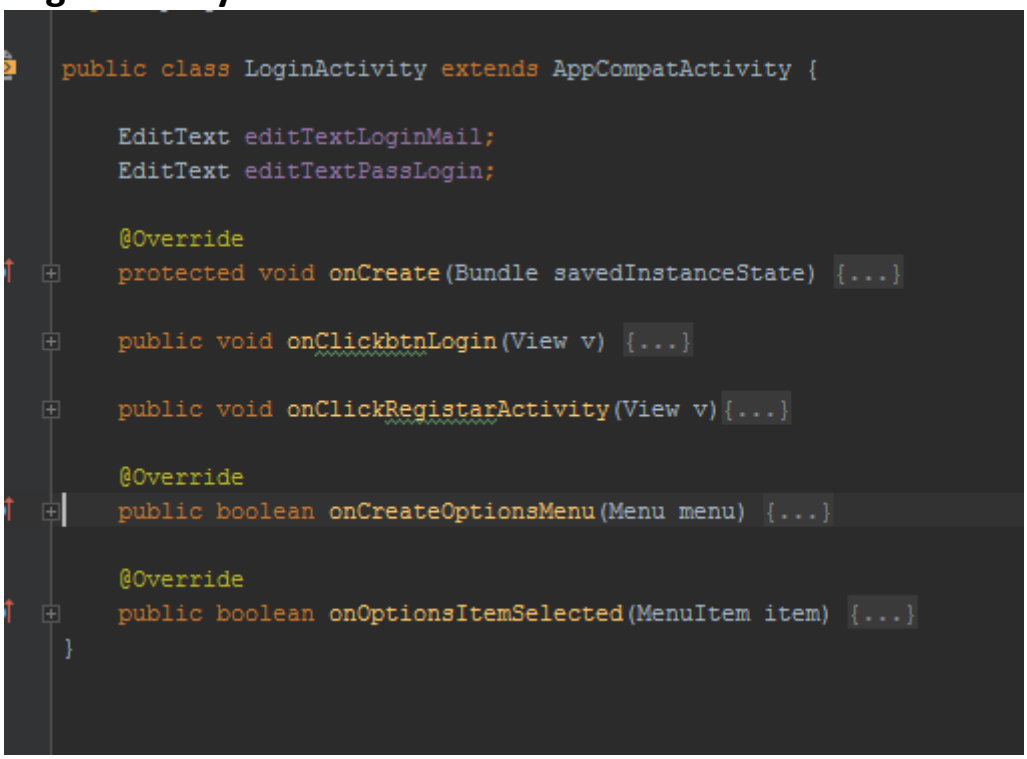
Método onClickMain

Envia o utilizador para a MenuActivity.

Método onClickButtonLoginMain

Envia o utilizador para a LoginActivity,

LoginActivity



Método onCreate

Faz a linkagem com os elementos do layout activity_login.

Método onClickBtnLogin

Verifica se as duas EditText estão vazias usando os métodos isEmpty da class Helper. Caso contrário informa o utilizador e faz return.
Verifica se tem acesso à rede.
Verifica se o email introduzido é valido, se não for informa o utilizador e faz return. Se sim encripta a password introduzia pelo utilizador fazendo uso do método getEncrypted da classe PasswordEncrypt. Usa o método checkLogin da class GereBD, se receber o valor 1 informa o utilizador que efetuou o login e tenta obter o idMotorista utilizando o método getMotoristaId da mesma classe. Se teve sucesso grava as preferências do utilizador em relação ao email, password, idMotorista, SESSION e validade. Depois inicia a tarefa sincronizar. Por ultimo envia o utilizador para a main activity. Caso não consiga obter o idMotorista informa o utilizador. Em todos os outros resultados da chamada do método checkLogin o utilizador é informado da razão pela qual não consegue fazer login.

Método onClickRegistarActivity

Envia o utilizador para a StoredPreferencesActivity.

MenuActivity

```
public class MenuActivity extends AppCompatActivity {

    Button btnGerirClientes;
    GereBD gereBD;
    XMLHandler handler;
    List<Cliente> listaClientes;
    boolean permission;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    public void onClickIniciarServico(View v) {...}

    public void onClickGerirClientes(View v){...}

    public void onClickGerirServicos(View v){...}
```

Método onCreate

Faz a linkagem com os elementos do layout acitivty_menu
Verifica se a SESSION está ativa usando o metido isSessionEnabled da classe SharedPreference. Caso esteja ativa o botão btnGerirClientes. Caso contrário se estiver ligado a uma rede envia o utlizador para a LoginActivity.

Método onClickIniciarServico

Verifica se está ligado a uma rede. Caso esteja cria uma lista de clientes usando o método listarClientes da classe GereBD e se essa lista não for vazia coloca o atributo permission a true. Caso não tenha aceso à rede povoa a lista de clientes com o método parseClientes. Se a lista não for vazia coloca a permission a true.
Verifica a permission, se true, ativa o dialog EscolherServicoDialogFragment.
Caso contrário informa o utilizador que deverá inserir clientes ou sincronizar a aplicação.

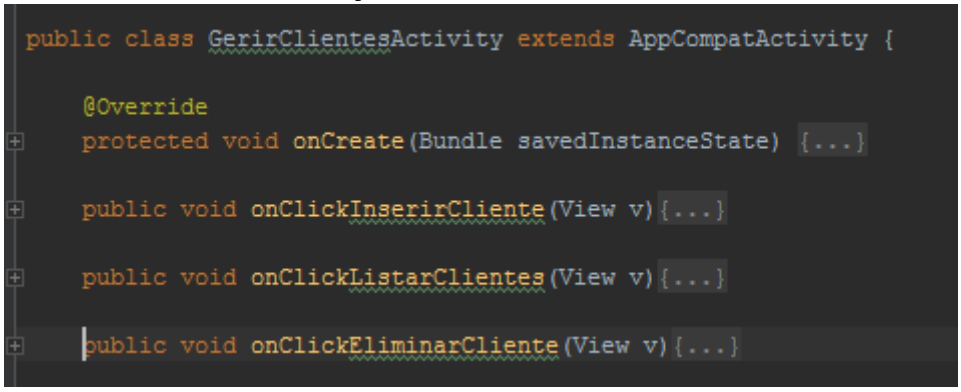
Método onClickGerirClientes

Envia o utilizador para a atividade GerirClientesActivity.

Método onClickGerirServico

Envia o utilizador para GerirServicosActivity.

GerirClientesActivity



Método onCreate

Faz a linkagem com os elementos do layout activity_gerir_clientes

Método onClickInserirCliente

Envia o utilizador para a atividade InserirNovoClienteActivity.

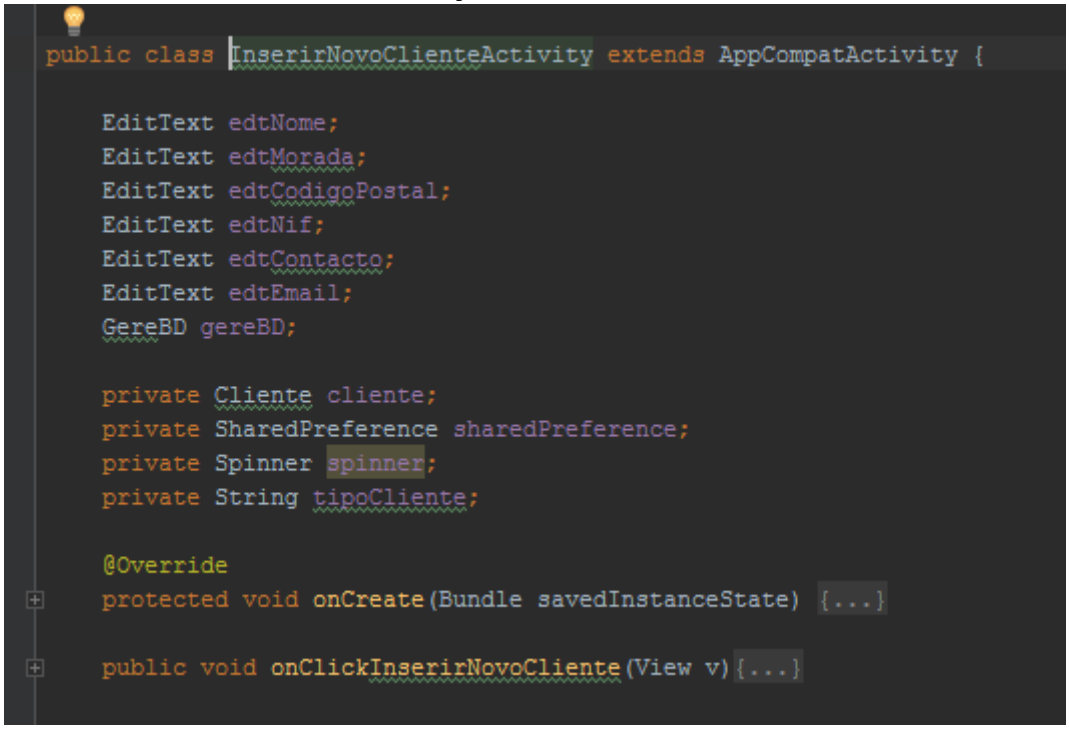
Método onClickListarClientes

Envia o utilizador para a atividade ConsultarClientesActivity.

Método onClickEliminarCliente

Envia o utilizador para a atividade EliminarClientesActivity.

InserirNovoClienteActivity



```
public class InserirNovoClienteActivity extends AppCompatActivity {

    EditText edtNome;
    EditText edtMorada;
    EditText edtCodigoPostal;
    EditText edtNif;
    EditText edtContacto;
    EditText edtEmail;
    GereBD gereBD;

    private Cliente cliente;
    private SharedPreference sharedPreference;
    private Spinner spinner;
    private String tipoCliente;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    public void onClickInserirNovoCliente(View v){...}
```

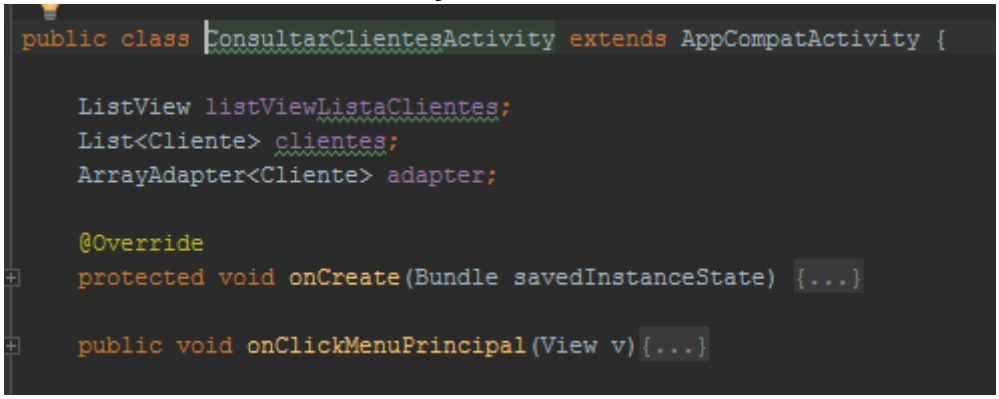
Método onCreate

Instancia um spinner usando o array tipo_cliente_array e o layout simple_spinner_item. E o dropdownResourse spinnerdropdown_item.
O listener do spinner atribui o valor escolhido pelo utilizador à propriedade tipoCliente.

Método onClickinserirNovoCliente

Verifica se as EditText que recebem o nome, a morada e o nif estão vazias. Se sim informa o utilizador que os campos são obrigatórios e faz return. Caso contrario faz o set das propriedades inseridas pelo utilizador e verifica se as propriedades opcionais estão preenchidas fazendo o respetivo set em caso positivo.
Posteriormente escreve o cliente localmente usando o método writeClientes da classe XMLHandler. Depois verifica se tem rede, em caso positive tenta inserir o cliente na case de dados. Em caso de insucesso informa o utilizador e escreve o cliente localmente usando o método writeNovoCliente da classe XMLHandler e ao mesmo tempo escreve usando o metodo wrteClientes. Por fim envia o utilizador para a ConsultarClientesActivity.

ConsultarClientesActivity

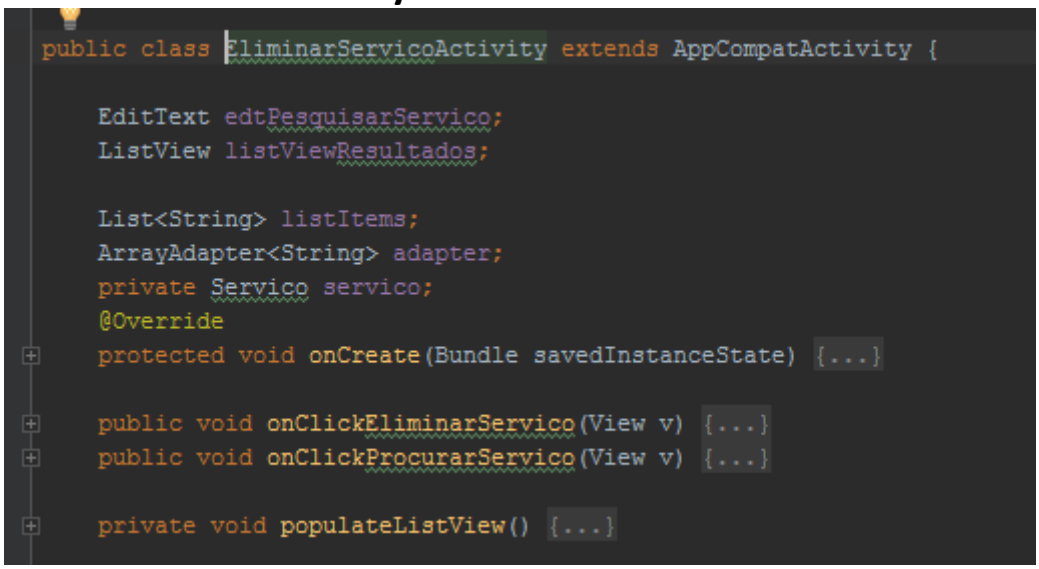


```
public class ConsultarClientesActivity extends AppCompatActivity {

    ListView listViewListaClientes;
    List<Cliente> clientes;
    ArrayAdapter<Cliente> adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    public void onClickMenuPrincipal(View v){...}
```

Método onCreate

Faz a linkagem com os elementos do layout activity_consultar_clientes
Verifica se tem ligação a uma rede, caso positivo povoa uma lista de clientes usando o método listarClientesda classe GereBD e usa o adapter para mostrar essa lista numa ListView. Caso não tenha ligação à rede informa o cliente e povoa a lista de clientes com os clientes armazenados localmente usando o método parseClientes. Mostrando-os depois numa ListView.
Instancia um listener setOnItemClickListener que envia o utilizador para MostraCienteActivity enviando como extra o cliente selecionado.

EliminarClienteActivity



```
public class EliminarServicoActivity extends AppCompatActivity {

    EditText edtPesquisarServico;
    ListView listViewResultados;

    List<String> listItems;
    ArrayAdapter<String> adapter;
    private Servico servico;
    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    public void onClickEliminarServico(View v) {...}
    public void onClickProcurarServico(View v) {...}

    private void populateListView() {...}
```

Método onCreate

Faz a linkagem com os elementos do layout activity_eliminar_cliente.

Método onClickButtonPesquisar

Verifica se a EditText não está vazia. Caso não esteja vazia instancia um ArrayList<Items>. Verifica se tem ligação a uma rede. Instancia um cliente através do método pesquisar cliente da classe GereBD. Verifica se o resultado da pesquisar é igual

ao cliente a eliminar. Se sim povoa uma lista com os atributos do cliente que será mostrada numa ListView. Se não tiver acesso à rede povoa uma lista de clientes com o método parserClientes da classe XMLHandler, depois percorre essa lista. Se o nome a procurar não for igual ao nome do cliente da lista ele adiciona esse cliente a uma nova lista. Por fim elimina o ficheiro clientes.xml e depois reescreve-o utilizando a nova lista.

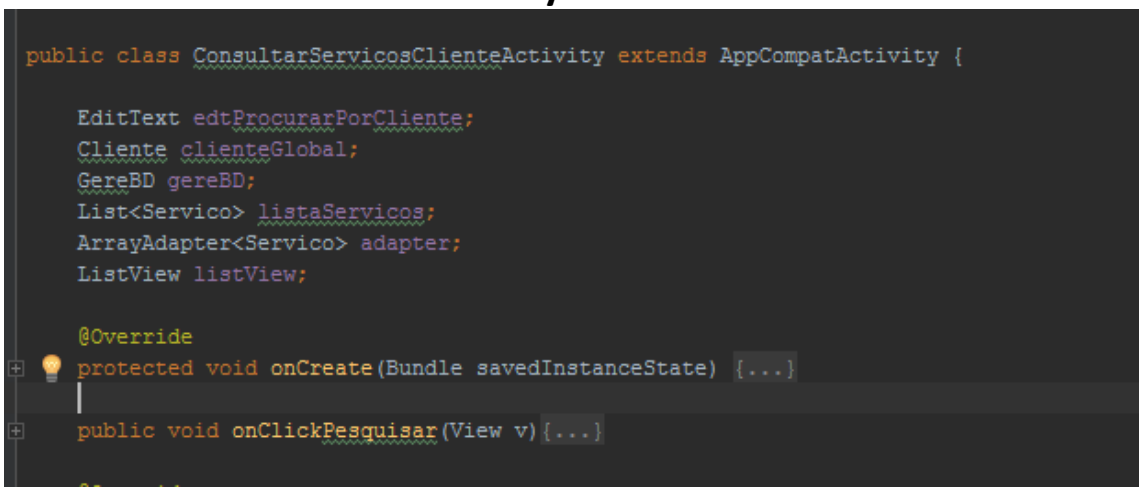
Método populateListView

Preenche uma lista de Strings com os atributos do cliente a eliminar.

Método onClickButtonEliminar

Verifica se tem ligação à rede. Se tem tenta eliminar o cliente da base de dados usando o método excluirCliente da classe GereBD. Se teve sucesso elimina o cliente do ficheiro clientes.xml.

ConsultarServicoClienteActivity



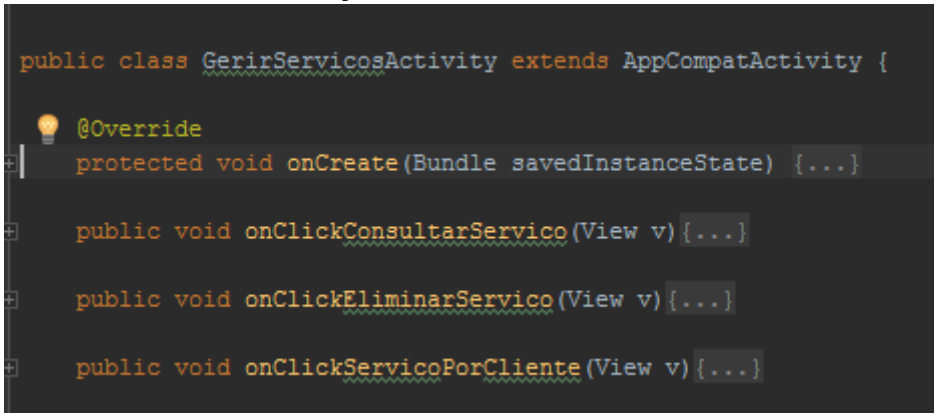
Método onCreate

Faz a linkagem dos elementos do layout activity_consultar_clientes.

Método onClickPesquisar

Verifica se a EditText está vazia, se sim, informa o utilizador e faz return. Caso contrário verifica se existe ligação à rede e procura o cliente na base de dados utilizado o método pesquisarCliente da classe GereBD. Se o resultado foi positivo, povoa uma lista de serviços usando o método pesquisarServicosPorCliente da classe GereBD e usa um adapter para mostrar essa lista usando uma ListView.
É instanciado um listener setOnItemClickListener para a ListView que envia o utilizador para MostraServicoActivity com o extra serviço selecionado e o boolean atualizar a true.

GerirServicoActivity



Método onCreate

Faz a linkagem com os elementos do layout activity_gerir_servicos.

Método onClickConsultarServico

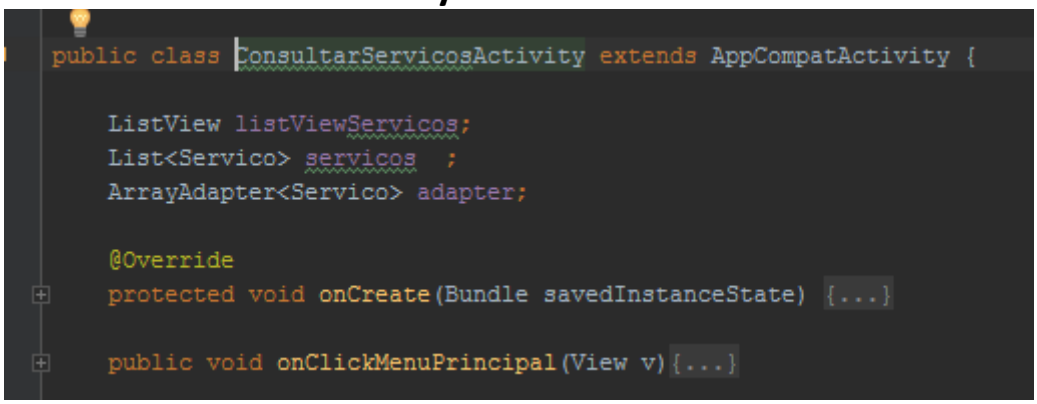
Envia o utilizador para a atividade ConsultarServicosActivity.

Método onClickEliminarServico

Envia o utilizador para a atividade EliminarServicoActivity.

Método onClickServicoPorCliente
Envia o utilizador para a atividade ConsultarServicosPorClienteActivity

ConsultarServicosActivity



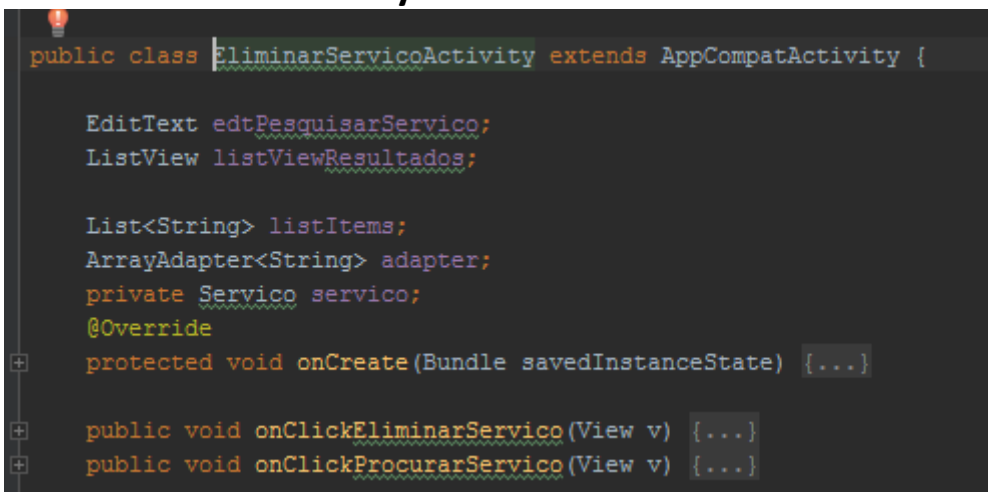
Método onCreate

Faz a linkagem com os elementos do layout activity_consultar_servicos.
Verifica se tem ligação à rede. Caso tenha povoa uma lista de serviços, usando o método listarServico da classe GereBD. Mostra essa lista usando um adapter numa listView. Caso não tenha net verifica se o ficheiro clientes.xml existe, se existir povoa a lista de serviços com o método parseServicos da classe XMLHandler e mostra essa lista usando um adapter numa listView. A listView implementa um listener setOnItemClickListener que envia o utilizador para a atividade MostraServicoActivity colocando o serviço selecionado como extra e a variável boleana atualizar a true.

Método onClickMenuPrincipal

Envia o utilizador para a atividade MenuActivity.

EliminarServicoActivity



Método onCreate

Faz a linkagem com os elementos do layout activity_eliminar_servico.

Método onClickEliminarServico

Se tiver internet tenta eliminar o serviço da base de dados usando o método excluirServico da classeGereBD. Se tiver sucesso elimina o ficheiro localmente usando o método parseServico conjuntamente com uma lsita de serviços e fazendo a comparação dos serviços presentes nessa lista com o serviço a eliminar.

Método onClickProcurarServico

Se a EditText não está vazia povoa um objeto serviço com o método pesquisarServico da classe GereBD, se o atributo processo do objeto retornado for igual ao valor introduzido na EditText, povoa-se uma lista com os atributos desse objeto e usando um adapter mostra-se numa ListView.

Método populateListView

Povoa uma lista com os atributos de um objeto do tipo serviço.

FormularioDeServicoActivity

```
public class FormularioDeServicoActivity extends AppCompatActivity{

    Servico servico = new Servico();
    NumberPicker numberPicker = null;
    TextView txtData;
    TextView txtHoraDeInicio;
    private String data;
    private String horaDeInicio;
    private String tipoServico;
    Button btnData;
    Button btnHora;

    //EDIT TEXTS FORMULARIO
    EditText edtProcesso;
    EditText edtCliente;
    EditText edtOrigem;
    EditText edtDestino;
    EditText edtCustoPortagens;
    EditText edtDistanciaPercorrida;
    EditText edtHorasDeEspera;
    EditText edtTrajeto;
    Spinner spinner;



    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    public void onClickTimePicker(View v){...}


    public void onClickDatePicker(View v){...}

    public void onClickInserirServico(View v){...}
}
```

Método onCreate

Faz a linkagens com os elementos do layout activity_formulario_de_servico
Instancia um spinner e povoa esse spinner com os elementos do serviço_tipo_array e implementa um listener setOnItemClickListener que atribui o valor escolhido pelo utilizador à propriedade tipoServico.

Método onClickTimePicker

Implementa um dialog do tipo TimePicker que permite ao utilizador escolher a hora e os minutos.

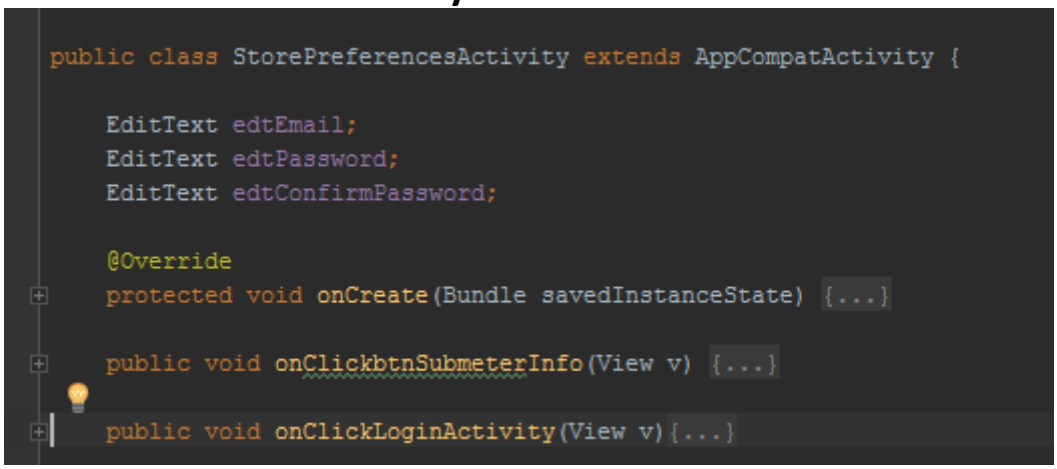
Método onClickDatePicker

Implementa um dialog do tipo DatePicker que permite ao utilizador escolher a data com dia, mês e ano.

Método onClickInserirServico

Verifica se as EditText não estão vazias. Se não estão vazias faz o set das diversas propriedades do objeto serviço. Por fim envia o utilizador para o MostraServicoActivity enviando o serviço como um extra.

StoredPreferencesActivity



Método onCreate

Faz a linkagem com os elementos do layout activity_storedpreferences

Método onClickBtnSubmeterInfo

Verifica se as EditText não estão vazias. Caso contrário avisa o utilizador e faz return; Verifica se o email introduzido é válido. Caso contrário informa o utilizador e faz return.Verifica se as duas passwords introduzidas coincidem. Caso contrário informa o utilizador e faz return. Verifica se a formação da password segue as regras definidas na class PasswordValidator usando o método validate. Encripta a password usando o método getEncrypted da classe PasswordEncrypt. Se tem net tenta fazer o checkLogin da classe GereBD, se o resultado for -1 tenta fazer o registo usando o método registarMotorista e grava as preferências (email, pass, idmotorista, validade e SESSION) nas SharedPreferences. Executa a tarefa mailInfo da classe TarefaMail, ou seja, envia um email com os dados do registo ao utilizador. Por último envia o utilizador para a CoordenadasActivity. Nos outros casos informa o utilizador das razões porque não conseguiu fazer o registo.

Método onClickLoginActivity

Envia o utilizador para o LoginActivity.

CoordenadasActivity

```
public class CoordenadasActivity extends FragmentActivity{

    Button btnSateliteMap;
    private GoogleMap mMap;
    private LocationListener lListener;
    private LocationManager lManager;
    private GPSHandler handler;
    private LatLng latLng;
    EditText location_tf;
    EditText edtLatitude;
    EditText edtLongitude;
    private Helper helper;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    @Override
    protected void onResume() {...}
    public void onSearch(View view)
    {...}

    //MUDA O TIPO DE MAPA ENTRE MAPA NORMAL E MAPA SATÉLITE
    public void changeType(View view)
    {...}

    private void setUpMapIfNeeded() {...}
    //MARCA POSIÇOES NO MAPA
    private void setUpMap(Location newLocation){...}
    //POLYMORFED
    private void setUpMap(LatLng latLng){...}

    public void getLocationForMap(final Activity activity) {...}

    public void onClickGuardarLocation(View v){...}
```

Método onCreate

Faz a linkagem com os elementos do layout activity_coordenadas.
Invoca o método setUpMapIfNeeded.

Método onResume

É um override que acrescenta a invocação do setUpMapIfNeeded.

Método onSearch

Verifica se a EditText location_tf não está vazia. Se não está vazia tenta fazer a geocoding do Morada introduzida pelo utilizador usando o método getFromLocationName da classe Geocoder. Se o resultado não é nulo extrai as coordenadas geográficas e mostra-as no mapa.

Método changeType

Modifica o tipo de mapa entre satélite e normal dependendo do click do utilizador.

Método setUpMapIfNeeded

SAe o mapa não estiver instanciado o método instacia-o e depois invoca o método getLocationForMap.

Método setUpMap

Tem duas versões, uma recebe uma Location e outra recebe um ponto LatLng. Ambas mostram essa localização no mapa.

Método getLocationForMap

Este método ativa um locationManager que por sua vez gere um locationListener que faz o override do método onLocationChanged acrescentando-lhe uma invocação do método setUpMap.

Método onClickGuardarLocation

Verifica se as EditText Latitude e Longitude não estão vazias. Se não estão vazias guarda os valores nas SharedPreferences associados as KEYS Lat e Lon. Se estiverem vazias guardam os valores detetados automaticamente pelo locationListener.

IniciaServicoActivity

```
public class IniciaServicoActivity extends AppCompatActivity {

    List<LatLng> mCapturedLocations;
    private GeoApiContext mContext;
    Button terminar;
    EditText editTextProcesso;
    EditText editTextPassageiros;
    Spinner spinner;
    private boolean portagens;
    private String nomeCliente;
    GereBD gereBD;
    ArrayList<String> clientesSpinnerParticular = new ArrayList<>();
    ArrayList<String> clientesSpinnerCompanhia = new ArrayList<>();
    XMLHandler handler;
    GPSHandler gpsHandler = new GPSHandler(this);
    private Servico servico = new Servico();
    ArrayAdapter<String> adapter;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    public void onClickIniciar(View v) {...}

    public void onClickTerminar(View v) throws Exception {...}
```

Método onCreate

Faz a linkagem com os elementos do layout activity_inicia_servico.
Verifica se o tipo do serviço é particular. Se sim faz a concatenação das strings SPD+Helper.getData+H+getTime. Depois remove os ":" e define este valor como o valor do atributo numProcesso do objeto do tipo Servico. Depois verifica se tem ligação a uma rede. Se sim povoa uma lista de clientes com o resultado da invocação do método listarClientes da classe GereBD. Depois para cada um dos clientes da lista verifica se é particular. Se sim adiciona-o a uma lista de clientes particulares. Se não adiciona a uma lista de clientes de companhia. Se não tiver ligação à rede faz um processo análogo mas lendo os clientes gravados localmente. Depois instancia um spinner mostrando apenas os clientes do tipo de serviço que foi escolhido. O Spinner implementa um setOnItemClickListener que atribui à propriedade nomeCliente o valor escolhido pelo utilizador.

Método onClickIniciar

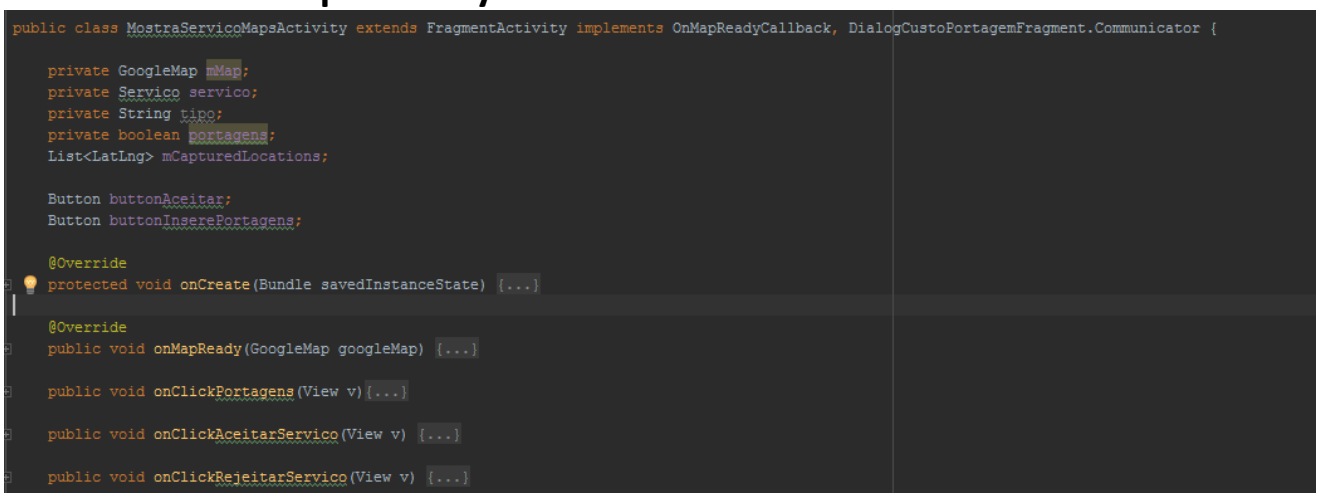
Verifica se as EditText estão vazias. Se sim informa o utlizador e faz return. Depois verifica se o numero de passageiros está entre 1 e 8 se não informa o utilizador e faz return. Depois começa a atribuir valores aos atributos do objeto do tipo Serviço usando os valores definidos pelo utilizador. A Data e a Hora de Inicio são obtidas usando as funções getTime e getDate da classe Helper.

Depois tenta inicializar o ficheiro onde serão guardados os pontos do trajeto. Se der erro informa o utilizador e faz return. Por ultimo inicio a captura dos pontos usando o método initGPS da classe GPSHandler.

Método onClickterminar

Desativa o locationListener usando o método listenerClose da classe GPSHandler. Verifica se existe ligação à rede. Se sim faz o parse dos pontos capturados para uma list de LatLng. Se o tamanho dessa lista é inferio a 1 informa o utilizador que ocorreu um erro e envia-o para o MenuActivity e faz return. Lê a primeira posição da lista de LatLng e obtem o GeocodingResult usando o método reverseGeocodeSnappedPoint da classe ServicoHandler. Faz o mesmo para a ultima posição da lista.
Atribui estes resultados aos atributos Origem e Destino do objeto serviço.
Invoca o método mergeCapture da classe ServicoHandler para adicionar ao trajeto o trajeto da +praça de táxis até ao inicio do serviço e do final do serviço até à praça de táxis. Depois chama o método getRoute da classe ServicoHandler para fazer a interpolação para a estrada mais próxima e a eliminação dos pontos com o mesmo placeid. Atribui o valor da propriedade distance usando o método getDistance da classe ServicoHandler. Atribui um valor à variável booleana portagens usando o método getPortagens da classe ServicoHandler. Depois deconstroi a lista de coordenadas em duas arraylist<Double> para puder passar para outra atividade visto que as List<LatLng> não são serializable. Depois envia o utilizador para MostraServicoMapsActivity conjuntamente com os dois ArrayList<Double> mais o objecto Servico mais a variável booleana portagens. Caso não tenha acesso à rede, chama o método displayPromptEnableWifi para perguntar ao user se não quer ativar ligação à internet. Depois faz o parser para uma List<LatLng> dos dados capturados usando o método loadGpxData da classe XmlHandler. Se essa lista tiver o tamanho maior do que 1 atribui um valor à propriedade trajeto usando o método trajetoToString da classe XmlHandler. Atribui um valor às propriedades origem e destino usando a primeira e a ultima posição da lista de pontos capturados. Atribui o valor zero à distancia. Tenta escrever o serviço localmente e por fim envia o Utilizador para MenuActivity.

MostraServicoMapsActivity



```
public class MostraServicoMapsActivity extends FragmentActivity implements OnMapReadyCallback, DialogCustoPortagemFragment.Communicator {

    private GoogleMap mMap;
    private Servico servico;
    private String tipo;
    private boolean portagens;
    List<LatLng> mCapturedLocations;

    Button buttonAceitar;
    Button buttonInserePortagens;

    @Override
    protected void onCreate(Bundle savedInstanceState) {...}

    @Override
    public void onMapReady(GoogleMap googleMap) {...}

    public void onClickPortagens(View v){...}

    public void onClickAceitarServico(View v) {...}

    public void onClickRejeitarServico(View v) {...}
```

Método onCreate

Faz a linkagem com os elementos do layout activity_mostra_servico_maps. Obtem um supportMapFragment e é notificado quando o mapa está pronto a ser usado. Recebe os extras enviados pela Activity IniciaServico. Reconstrói a List<LatLng> com o trajeto.

Método onMapReady

É um override que mostra uma polyline com os pontos do trajeto.

Método onClickPortagens

Ativa o Dialog CustoDePortagemFragment

Método onClickAceitarServico

Atribui um valor à propriedade trajeto do serviço usando o método trajetoTostring da classe XmlHandler. Depois escreve esse trajeto num ficheiro usando o método writeTrajeto. Por fim envia o utilizador para MostraServicoActivity juntamente com o serviço como extra.

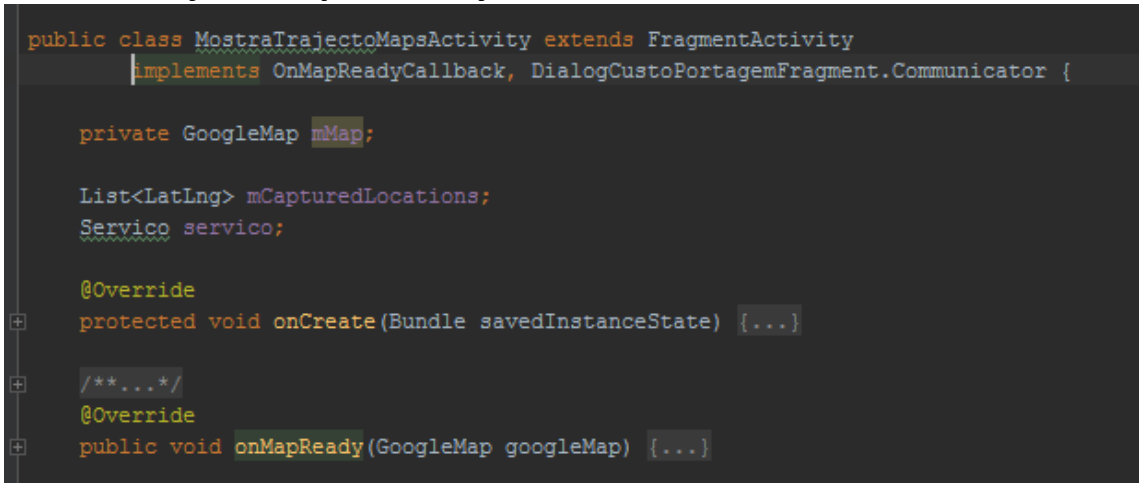
Método onClickRejeitarServico

Envia ao utilizador para FormulárioDeServiçoActivity.

Método onDialogMessage

É um override da interface DialogCustoPortagemFragment que recebe o valor da portagem caso exista e atribui a String recebida ao custo de portagens. E usa o int confirm para desbloquear o botão aceitar.

MostraTrajetoMapsActivity



Método onCreate

Faz a linkagem com os elementos do layout activity_mostra_trajeto_maps.
Obtem um supportMapFragment e é notificado quando o mapa está pronto a ser usado.Reconstrói a List<LatLng> com o trajeto. Recebe um serviço através de um intente enviado pela atividade MostraServicoActivity.

Método onMapReady

Desenha uma polyline no mapa usando os pontos recebidos pelo Intent.

MostraServicoActivity

Método onCreate

Faz a linkagem com os elementos do layout activity_mostra_servico.
Recebe um serviço através de um Intent e um Boolean através de outro Intent.
Este boolean determina a visibilidade de alguns botões. Povoa uma lista contendo os dados do serviço para ser usada numa ListView. Mostra-se a ListView, a ListView tem um listener associado que responde à interação com o user de duas maneiras diferentes. Se o user clicar no item correspondente à propriedade nome do cliente ativa um DialogSpinnerFragment e se clicar numa outra qualquer ativa um DialogCorrigeDadosFragment.

Método populateListView

Instancia um ArrayAdapter para mostrar os dados do serviço. Faz o set de uma TextView dependendo do Tipo de Serviço.

Método onClickAtualizar

Verifica se tem conexão. Se sim tenta atualizar os dados na base de dados usando o método atualizar serviço da classe GereBD. O utilizador é informado de todos os resultados possíveis. E é enviado para a GerirServicosActivity.

Método onClickSubmeterServico

Verifica se tem conexão à internet. Se sim verifica se a sessão está iniciada, caso não esteja envia o utilizador para a LoginActivity e faz return. Faz o set do idMotorista usando o valor guardado nas SharedPreferences. Tenta inserir o serviço na base de dados usando o método inserirServico da classe GereBD. Se não conseguiu avisa o utilizador e guarda o serviço localmente usando o método writeServico da classe XMLHandler. Se conseguiu informa o utilizador e inicia a tarefa assíncrona enviaMail. Senão tiver internet escreve o serviço locamente. Por fim envia o utilizador para a MenuActivity.

Método onClickVerTrajeto

Envia o utilizador para MostraTrajetoMapsActivity, juntamente com o objeto serviço.

Método onDialogMessage

Recebe os dados que foram alterados pelo utilizador no Dialog, atualizando a ListView posteriormente.

AssyncTask<Void,Void,Boolean> enviaMail

Método doInBackground

Envia um mail com o número do processo no assunto e o objeto do serviço no corpo da mensagem e ainda o ficheiro em formato XML com o trajeto em anexo.
Retorna true se conseguiu envia-lo.

Método onPostExecute

Recebe o resultado do método doInBackground . Este método apaga todos os ficheiros que foram criados em relação a este serviço.

Conclusão

Com este trabalho tivemos a oportunidade de desenvolver uma aplicação desde o projeto até à fase de produção. Não tivemos tempo para desenvolver todas as funcionalidades que pretendíamos, aumentar ao tipo de consultas tais como: consultar serviços por data, consultar serviços por tipo de cliente, consultar distancias percorridas por data, etc. No entanto achamos que a aplicação desenvolvida já consegue cumprir o seu papel.

Apêndice

Instruções para teste

Para puder consultar os serviços e os clientes que já estão armazenados na base de dados terá que se autenticar ou com:
Email: Pedrobelchior75@gmail.com
Password: Jolly_Jumper!75

Email: Bruno.ferreira.692@gmail.com
Password: 1Qwerty$

Se quiser verificar a funcionalidade de enviar email depois do serviço ser submetido sem ter criado um User e ter Submetido serviços basta mudar o atributo from da linha 56 da classe Mail para o seu email de teste.

Poderá verifica-lo quando clicar em serviços e clicar em submeter.

Poderá ainda realizar um novo registo para verificar a funcionalidade de captura de coordenadas de GPS caso teste no telemóvel.
Se optar por esta forma poderá ainda adicionar clientes quer esteja offline como offline.
A Sincronização entre os ficheiros locais e o Web Service só acontece quando reinicia a aplicação.
Para aceder ao Web Service deverá alterar o valor da Constante IP que se encontra na linha 89 da classe Constants para o IP da maquina onde tem o Web Service alojado.