Universidade Federal do Piauí (UFPI)

Centro de Ciências da Natureza (CCN)

Departamento de Computação (DC)

Pedro Ivo Soares Barbosa

Engenharia de Software II *Professor: Armando Soares Sousa*

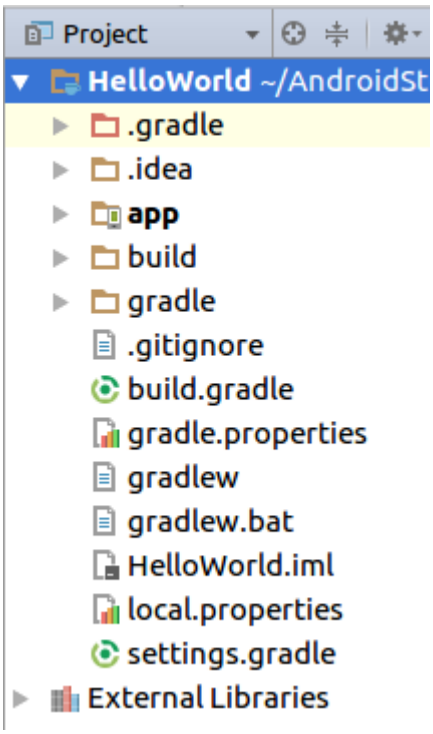
2016.2

Tutorial de Regras de Negócio (Controladores) no Android

Índice

1. Estrutura do Projeto Android

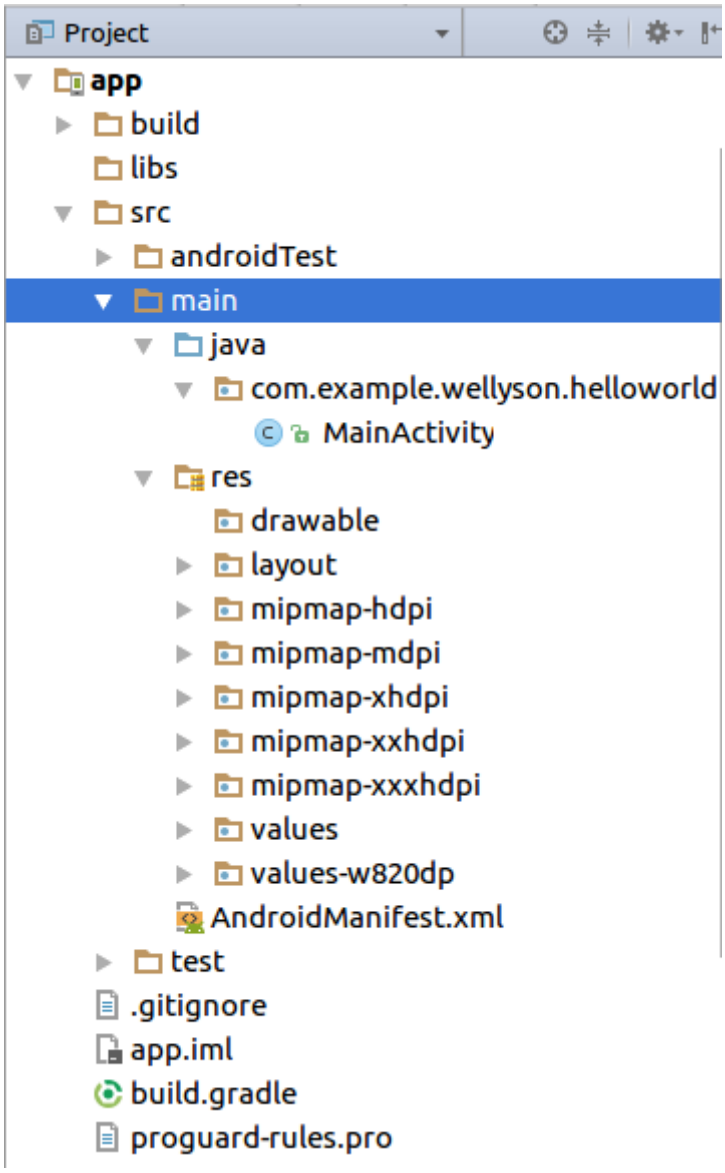
O Android Studio pode abrir um projeto de cada vez, e cada projeto pode conter um ou mais módulos. Podemos alterar a forma de visualização do projeto no menu superior da área de diretórios do projeto. Exemplo de visualização no modo 'Project':



- Arquivos da raiz do projeto:

app: módulo padrão do projeto.
build.gradle: arquivo de configuração do Gradle que vale para todo o projeto. Você provavelmente não vai alterar nada nesse arquivo.
gradle.properties: arquivo de propriedades para customizar o build do Gradle.
gradlew.bat: script que executa o build do Gradle para compilar o projeto.
local.properties: arquivo com as configurações locais do projeto.
settings.gradle: arquivo de configuração do Gradle que indica quais módulos devem ser compilados.

A pasta *app* é onde ficam os arquivos da aplicação. Na pasta */app/src/main* estão os arquivos .java, .xml e imagens do aplicativo



- Arquivos do módulo app:

build: pasta em que ficam os arquivos compilados do módulo. O arquivo apk que é o aplicativo compilado fica na pasta build/outputs/apk.

R.java: classe gerada automaticamente ao compilar o projeto e permite que aplicação acesse qualquer recurso como arquivos e imagens utilizando as constantes desta classe. Nunca deve ser alterada manualmente.
libs: Pasta para inserir os arquivos .jars que devem ser compilados com o projeto.
src/main/java: pasta com as classes Java, por exemplo, a MainActivity.
src/main/res: pasta que contém os recursos da aplicação, como imagens, layouts de telas e arquivos de internacionalização.
res/drawable: pasta com as imagens da aplicação. Como existem diferentes resoluções de telas, é possível customizar as imagens para adequar a cada uma. Para isso, há diversas pastas para as imagens, com resoluções específicas.
res/mipmap: pasta com o ícone da aplicação, o qual por padrão chama-se ic_launcher.png. Também apresenta variações conforme a densidade da tela do dispositivo.
res/layout: contém os arquivos XML de layouts para construir as telas da aplicação.
res/menu: contém os arquivos XML que criam os menus da aplicação que são os botões da action bar.
res/values: contém os arquivso XML utilizados para internacionalização, configuração de temas e outras configurações.

Cada arquivo seja imagenm ou XML dentro da pasta /app/res contém uma referência na classe R, que é automaticamente gerada ao compilar o projeto. Cada vez que se altera, adiciona ou remove um arquivo dessas pastas, a classe R é alterada também.

Ex.: /res/drawable/smile.png (será gerada a constante R.drawable.smile)
/res/layout/teste.xml (será gerada a constante R.layout.teste)

Essas constantes são utilizadas no código-fonte da aplicação para referenciar esses objetos.

1.1 Arquivo AndroidManifest.xml:

É a base de uma aplicação Android e contém todas as configurações necessárias para executar a aplicação.

```xml
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.wellyson.helloworld">

    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="HelloWorld"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>

</manifest>
```

O pacote principal do projeto é declarado na tag *<package>*

Cada Actitvity do projeto deve estar declarada no arquivo e para isso utiliza-se a tag *<activity>*, a qual recebe o nome da classe. O Android Studio faz isso automaticamente ao criarmos uma Activity pela IDE.

A tag *<intent-filter>* é necessária para customizar a forma como a activity será iniciada. A ação *MAIN* significa que a activity pode ser iniciada isoladamente, como o ponto inicial da aplicação. A categoria *LAUNCHER* indica que activity estará disponível para o usuário na tela inicial junto com as outras aplicações instaladas.

1.2 Arquivo build.gradle:

No projeto existe o arquivo *build.gradle* padrão de todos os módulos e o arquivo *app/build.gradle* com as configurações de compilação do módulo *app*.

No arquivo *app/build.gradle* você configura a versão do aplicativo e também a versão mínima do Android (API Level) que ele suporta, além de declarar as bibliotecas que são necessárias para a compilação.

2. Tratamento de Eventos e Exceções

2.1. Evento de botão:

Exemplo de uma tela de login simples e o tratamento do evento quando o botão 'Login' é clicado:

- MainActivity.xml:

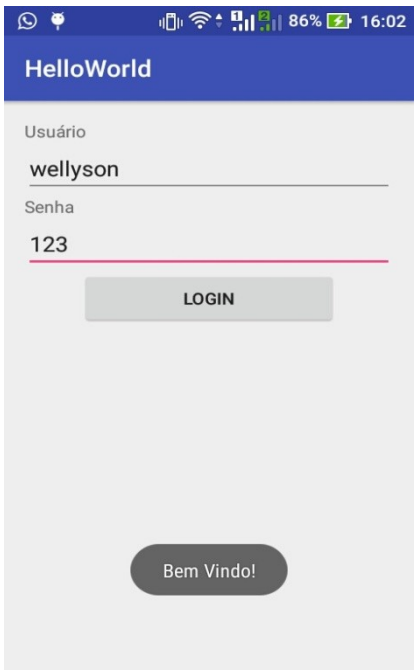
```xml
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/activity_main"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:gravity="center_vertical"
    android:orientation="vertical"
    android:padding="16dp"
    tools:context="com.example.wellyson.helloworld.MainActivity">

    <TextView
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:text="@string/usuario" />

    <EditText
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:id="@+id/tUsuario" />

    <TextView
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:text="@string/senha" />

    <EditText
        android:layout_width="match_parent" android:layout_height="wrap_content"
        android:id="@+id/tSenha" />

    <Button
        android:layout_width="200dp" android:layout_height="wrap_content"
        android:text="@string/login"
        android:id="@+id/button"
        android:layout_gravity="center"/>
</LinearLayout>
```

Foi adicionado um id para os dois campos de texto e também para o botão. Isso é feito com a tag *android:id*. No código vamos utilizar o método *findViewById(id)* para acessar esses objetos pelo id.

- MainActivity.java:

```java
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        Button btLogin = (Button) findViewById(R.id.button);
        btLogin.setOnClickListener(onClickLogin());
    }

    private View.OnClickListener onClickLogin() {
        return new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                TextView tUsuario = (TextView)
                        findViewById(R.id.tUsuario);
                TextView tSenha = (TextView)
                        findViewById(R.id.tSenha);
                String usuario = tUsuario.getText().toString();
                String senha = tSenha.getText().toString();

                if("wellyson".equals(usuario) && "123".equals(senha)) {
                    alert("Bem Vindo!");
                } else {
                    alert("Login ou senha incorreta!.");
                }
            }
        };
    }

    private void alert(String s) {
        Toast.makeText(this, s, Toast.LENGTH_SHORT).show();
    }
}
```

Para tratar os eventos de um botão é utilizado o método *setOnClickListener(listener)*. Esse método recebe como argumento uma instância da interface *android.view.View.OnClickListener*. Essa interface define o método *onClick(view)*, o qual é chamado quando o evento ocorrer, passando como argumento o objeto da view que gerou o evento, nesse caso, o botão.



Outro método bastante utilizado para tratar os eventos é definir o nome do método que deve ser chamado no arquivo XML de layout.

```
<Button
    android:layout_width="200dp" android:layout_height="wrap_content"
    android:text="Login"
    android:id="@+id/button"
    android:layout_gravity="center"
    android:onClick="onClickBtLogin"/>
```

O método 'onClickBtLogin(view)' deve existir na classe da activity:

```
public void onClickBtLogin(View view){
    //trata o evento de clique...
}
```

Existem várias formas de escrever este mesmo código para tratar o evento de um botão, porém o primeiro método utilizado separa de forma simples os eventos de cada botão, se tivermos vários, e evita erros por parte do programador.

2.2. Exceções:

```
try{

    //bloco em que as exceções podem ocorrer.
}

catch (Exception e){

    Toast.makeText(this, "mensagem de erro", Toast.LENGTH_SHORT).show();

}
```

Ao capturar a exceção uma mensagem de erro pode ser mostrada ao usuário utilizando *Toast*, *AlertDialog* ou enviadas como log ao *LogCat*

para ajudar na depuração.

```
private static final String TAG = "Testes";

Log.e(TAG, "mensagem");
```

A constante *TAG* serve para filtrar as mensagens da aplicação no *LogCat*.

Existem vários métods da classe *Log*, uma descrição mais completa pode ser encontrada no site: https://developer.android.com/reference/android/util/Log.html

3. Intents

A classe *android.content.Intent* é o coração do Android. Uma intent é uma mensagem da aplicação para o sistema operacional, solicitando que algo seja realizado. Pode ser utilizada para:

1. Enviar uma mensagem para o Sistema Operacional
2. Abrir uma nova tela da aplicação
3. Ligar para um número de celular
4. Abrir o browser em uma página da internet
5. Enviar uma mensagem por SMS ou Email
6. Exibir algum endereço, localização ou rota no Google Maps
7. Abrir a agenda de contatos para selecionar um contato
8. Abrir a galeria de fotos ou vídeos para selecionar um arquivo de multimídia
9. Tocar uma música ou vídeo
10. Abrir a câmera e solicitar que uma foto ou vídeo sejam gravados

Algumas intents precisam de permissões para funcionar, como nos casos de fazer ligação, acessar a lista de contatos e utilizar a internet. Para isso, é necessário adicionar a permissão no arquivo *AndroidManifest.xml*, como indicado a seguir

```
<uses-permission android:name="android.permission.CALL_PHONE"/>
<uses-permission android:name="android.permission.READ_CONTACTS"/>
<uses-permission android:name="android.permission.INTERNET"/>
```

Exemplo de intents nativas :

Realizar Ligação:

```
try{
    Uri uri = Uri.parse("tel:123456789");
    Intent intent = new Intent(Intent.ACTION_CALL, uri);
    startActivity(intent);
}catch (SecurityException e){
    //Tratar exceção
}
```

Enviar SMS:

```
try{
    Uri uri = Uri.parse("sms:123456789");
    Intent smsIntent = new Intent(Intent.ACTION_SENDTO, uri);
    smsIntent.putExtra("body", "olá");
    startActivity(smsIntent);
}catch (SecurityException e){
    //Tratar exceção
}
```

Navegação entre telas e inicialização de uma nova activity

Para navegar para a próxima tela, é criado um objeto do tipo *android.content.Intent* informando a classe da activity que deve ser chamada. Ao criar a intent, é preciso passar a referência do contexto, que é a classe *android.content.Context.*

Geralmente vemos o contexto ser referenciado por **this** no código fonte

Intent it = new Intent(this, TelaDoisActivity.class);

startActivity(it);

Se quisermos, através dessa intent, passar algum tipo de parâmetro, podemos usar o método *putExtra* da intent, o qual recebe como parâmetro um identificador e um valor.

Intent it = new Intent(this, TelaDoisActivity.class);

it.putExtra(“nome”, “Pedro”);

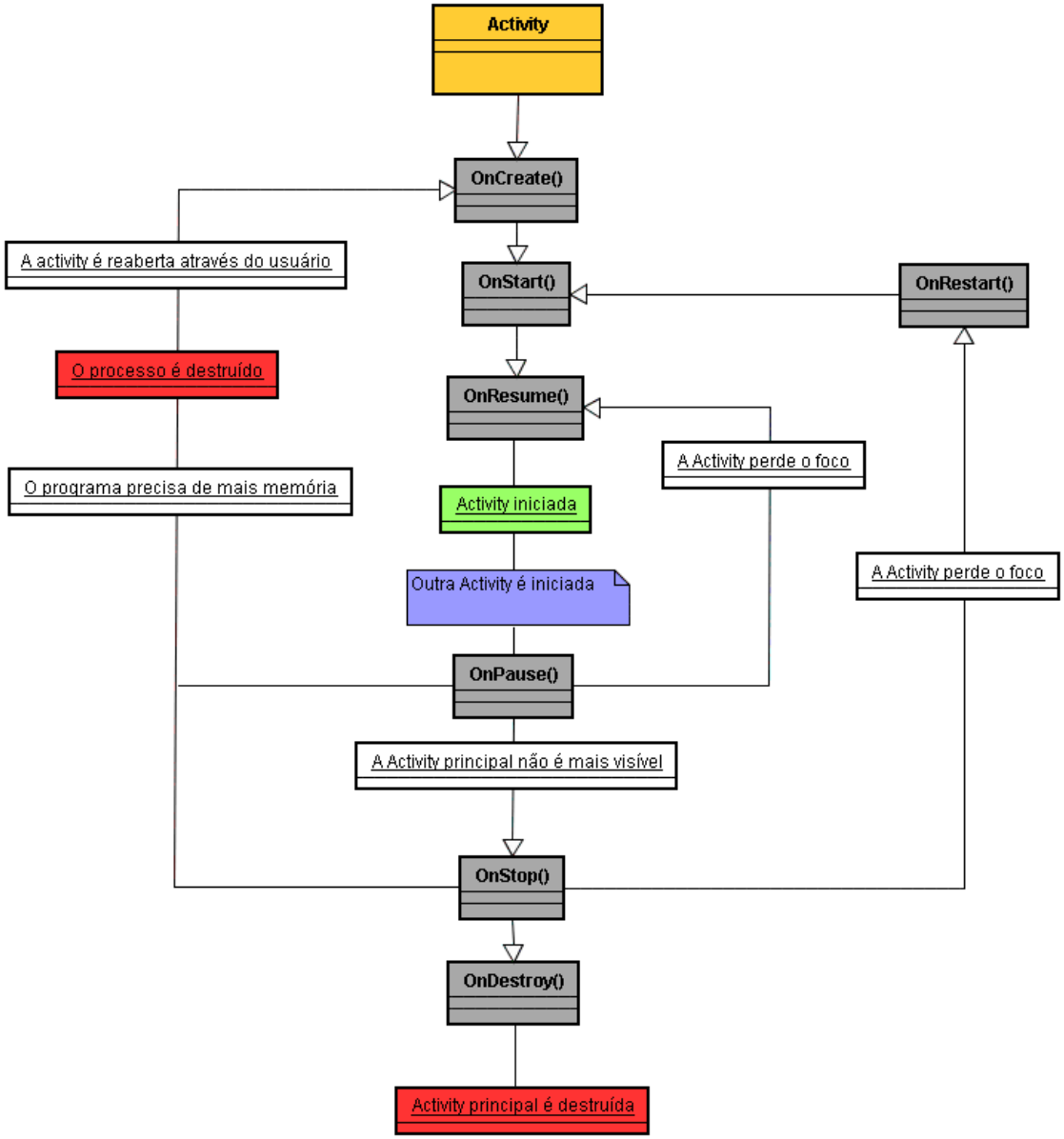
startActivity(it);

Para pegar de voltar o parâmetro na activity destino, obtemos a intent com o método *getIntent(),* e obtemos o parâmetro usando o método *getExtra* e suas variantes, que recebe como parâmetro o identificador.

Intent it = getIntent();

String nome = it.getStringExtra(“nome”);

4. Ciclo de Vida da Activity



onCreate() - É a primeira função a ser executada em uma Activity. Geralmente é a responsável por carregar os layouts XML e outras operações de inicialização. É executada apenas uma vez.

onStart() - É chamada imediatamente após a onCreate() – e também quando uma Activity que estava em background volta a ter foco.

o*nResume()* - Assim como a onStart(), é chamada na inicialização da Activity e também quando uma Activity volta a ter foco. Qual a diferença entre as duas? A onStart() só é chamada quando a Activity não estava mais visível e volta a ter o foco, a onResume() é chamada

nas “retomadas de foco”.

onPause() - É a primeira função a ser invocada quando a Activity perde o foco (isso ocorre quando uma nova Activity é iniciada).

onStop() - *S*ó é chamada quando a Activity fica completamente encoberta por outra Activity.

onDestroy() - A última função a ser executada. Depois dela, a Activity é considerada “morta” – ou seja, nao pode mais ser relançada. Se o usuário voltar a requisitar essa Activity, um novo objeto será contruído.

onRestart() - Chamada imediatamente antes da onStart(), quando uma Activity volta a ter o foco depois de estar em background.

5. Threads

Linha ou Encadeamento de execução (em inglês: *Thread*), é uma forma de um processo dividir a si mesmo em duas ou mais tarefas que podem ser executadas concorrencialmente. O suporte à *thread* é fornecido pelo próprio sistema operacional no caso da linha de execução ao nível do núcleo (em inglês: *Kernel-Level Thread (KLT)*), ou implementada através de uma biblioteca de uma determinada linguagem, no caso de uma *User-Level Thread (ULT*). == Uma thread permite, por exemplo, que o usuário de um programa utilize uma funcionalidade do ambiente enquanto outras linhas de execução realizam outros cálculos e operações.

Falando de Android...

6.1. Introdução

Quando um aplicativo é aberto no Android, um processo dedicado no sistema operacional é criado para executá-lo. Cada processo tem uma única thread, conhecida como Main Thread ou UI Thread, responsável por gerenciar todos os eventos da tela, assim como atualizar a interface gráfica.

Código que executa uma Thread:

```
new Thread(){
    public void run(){
        //Código que deve executar em segundo plano aqui
    };
}.start();
```

Uma thread deve ser filha da Classe Thread e deve implementar o método *run()*. Ao chamar o método *start()*, thread é iniciada, ou seja, o método *run()* vai executar em segundo plano.

No caso do Android, sempre que uma thread é iniciada, o Android não permite que uma thread diferente da principal atualize a interface gráfica da tela. Por isso, a classe thread *android.os.Handler* foi criada com o objetivo de enviar uma mensagem para a thread principal, para que,

algum momento apropriado, essa mensagem possa ser processada e atualizar a interface gráfica da tela(view).

```
final Handler handler = new Handler();
new Thread(){
    public void run(){
        //Código que deve executar em segundo plano aqui
        handler.post(new Runnable() {
            @Override
            public void run() {
                //Código que atualiza a interface aqui
            }
        });
    };
}.start();
```

Ou pode-se usar o método *runOnUIThread(runnable)*, que é um atalho para usar um handler que está dentro da activity.

Usamos a classe *android.os.Handler* por dois bons motivos:

1.Atualizar a interface(view) sempre que uma thread foi utilizada para fazer algum processamento em segundo plano.

2. Agendar uma mensagem *android.os.Message* ou um *java.lang.Runnable* para executar em determinado momento.

6.2. Método *sendMessage(msg)*

Para enviar uma mensagem com a classe *Handler*, podemos utilizar o método *sendMessage(msg)* e suas variantes.

sendMessage(msg), Envia a mensagem informada para a fila de mensagens para ser processada assim que possível

sendEmptyMessage(int), Envia a mensagem contendo apenas o atributo *what* informando como parâmetro.

sendMessageDelayed(msg,long), Envia a mensagem para a fila de mensagens, mas ela é processada somente depois de determinado tempo informado.

sendMessageAtTime(msg, long), Envia a mensagem para a fila de mensagens, mas ela é processada apenas na data informada.

Para usar o método *sendMessage(msg)* ou uma de suas variações, é preciso criar uma subclasse da classe *android.os.Handler*.

6.3. Método *post(runnable)*

Outra forma de enviar uma mensagem é com o método *postMessage(runnable),* que funciona de forma similar ao método *sendMessage(msg),* mas recebe uma implementação da interface *java.lang.Runnable.*

Indica-se usar um *Java.lang.Runnable* em vez de enviar uma mensagem com a classe *android.os.Message* , pois não é necessário criar uma subclasse de *android.os.Handler*.

Para executar ou agendar um *java.lang.Runnable*, são usados os mesmos métodos para enviar uma mensagem, com os mesmos argumentos, mas agora os nomes dos métodos começam com a palavra

post(...) .

post(Runnable), Adiciona o Runnable na fila de mensagens

postDelayed(Runnable, long), Adiciona o Runnable na fila de mensagens, mas somente executa o processo depois do tempo especificado em milissegundos

postAtTime(Runnable, long), Adiciona o Runnable na fila de mensagens, mas somente executa o processo na data especificado em milissegundos

6.4. Atualizando a view dentro de uma thread

Como foi visto anteriormente, não é possível atualizar a interface gráfica a partir de threads que não seja a Thread Principal. É preciso que as outras threads enviem mensagens à Thread Principal solicitando que esta modifique a interface gráfica.

A solução é criar um método, dentro do qual declaramos um ***runOnUIThread***, o qual recebe no construtor um ***Runnable***, então, dentro do método ***run*** do Runnable realizamos todas as alterações desejadas na interface gráfica.

6.5. Agendando tarefas contínuas na activity

A classe *Handler* é muito utilizada para executar tarefas de modo contínuo na activity, como por exemplo, executar alguma ação em intervalos de tempo regulares. Para isso, o que devemos fazer é declarar um objeto do tipo ***Handler*** , então usamos o método ***postDelayed*** que recebe um objeto ***Runnable*** e um número (long) que é exatamente o intervalo de tempo no qual acontecerá a ação declarada no método ***run*** do Runnable.

Para finalizar, dentro do método ***onDestroy*** da activity, usamos o método do handler, chamado ***removeCallbacksAndMessages(null),*** o qual cancela todas as mensagens enviadas ao handler. Isso é necessário para garantir que nenhuma mensagem seja entregue com a activity fechada.