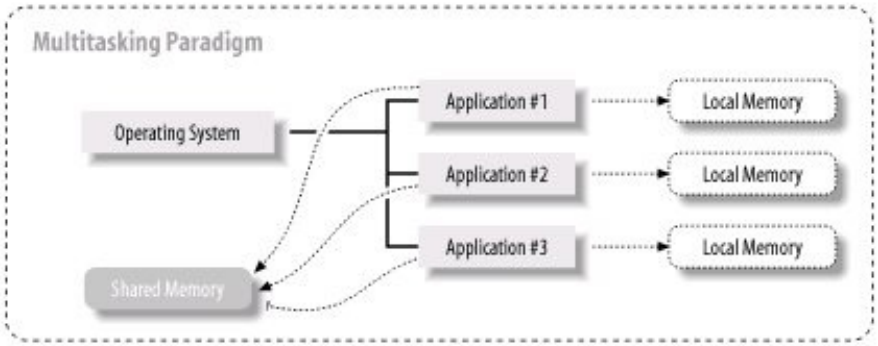
SENAI
SÃO PAULO

Android

Thread, Handler, AsyncTask, HttpURLConnection e ObjectMapper

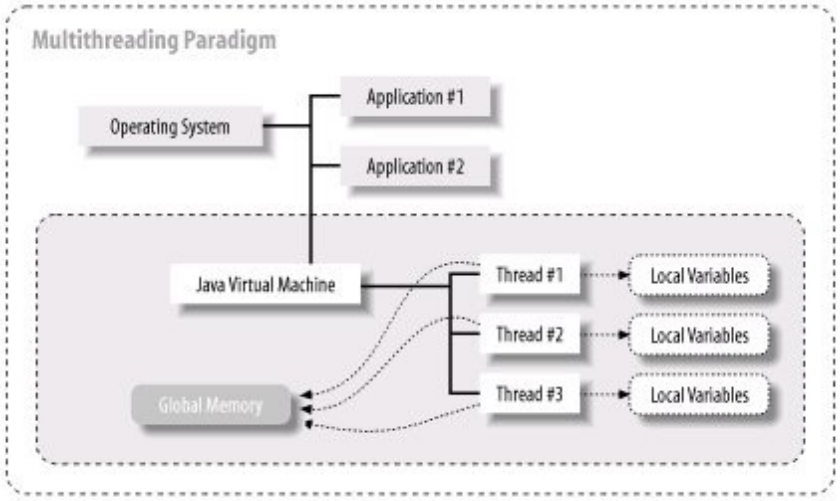
Multi-Thread

- **Multiprocessamento**
  - A capacidade de ter mais de um programa funcionando no que parece ser simultâneo
- Duas maneiras
  - O Sistema Operacional interromper o programa sem consultar primeiro
  - Os programas são interrompidos apenas quando estão querendo produzir controle



Multi-Thread

- **Linhas de Execução (Multi-Thread)**
  - Num mesmo processo cada tarefa é chamada de Thread (linha de execução).
- Qual é a diferença?
  - Cada processo tem um conjunto completo de variáveis próprias, as threads compartilham os mesmos dados.
  - É necessário muito menos sobrecarga para criar e destruir threads individuais.



Exemplos

- Um navegador deve tratar com vários hosts, abrir uma janela de correio eletrônico ou ver outra página, enquanto descarrega dados.
- A própria linguagem de programação Java usa uma thread para fazer coleta de lixo em segundo plano evitando assim o proglema de gerenciar memória!
- Os programas GUI têm uma thread separada para reunir eventos da interface com o usuário do ambiente operacional hospedeiro.

Multi-Thread

- **Classes MultiThread**
  - Estender Thread ou implementar a interface Runnable
  - Colocar o código que precisar ser executado no método run.

```
public class NovaTread extends Thread {
  // Outros atributos e métodos

  @Override
  public void run() {
    // Executa um conjunto de operações
  }
}

public class NovaClasse implements Runnable
  // Outros atributos e métodos

  @Override
  public void run() {
    // Executa um conjunto de operações
  }
}
```

Multi-Thread

- **Executanto a Thread**
  - Criar um objeto Thread no caso das classes que implementam Runnable
  - Ativar a thread através da chamada do método start()

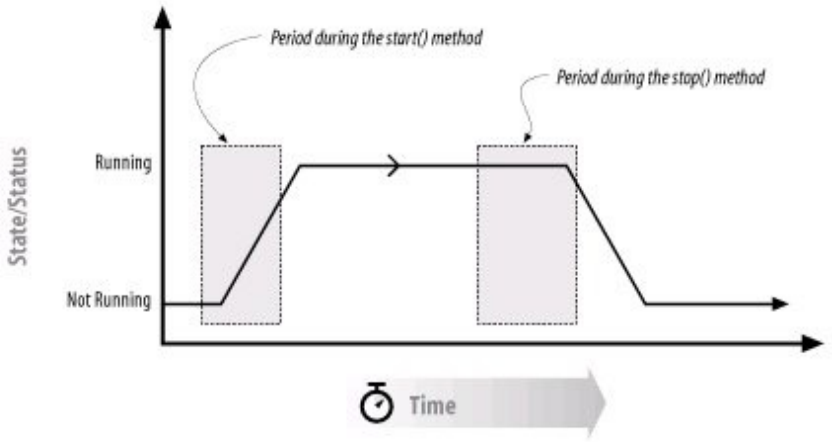
```
public class OutraClasse {
  public static void main(String[] args) {
    // Criando e iniciando uma Thread
    NovaThread umaThread = new NovaThread();
    umaThread.start();

    // Criando e iniciando um Objeto Runnable
    NovaClasse umRunnable = new NovaClasse();
    Thread aThread = new Thread(umRunnable);
    aThread.start();
  }
}
```

Executando uma Thread

```
public void onClick(View v) {
    new Thread(new Runnable() {
        public void run() {
            Bitmap b = loadImageFromNetwork("http://example.com/image.png");
            mImageView.setImageBitmap(b);
        }
    }).start();
}
```

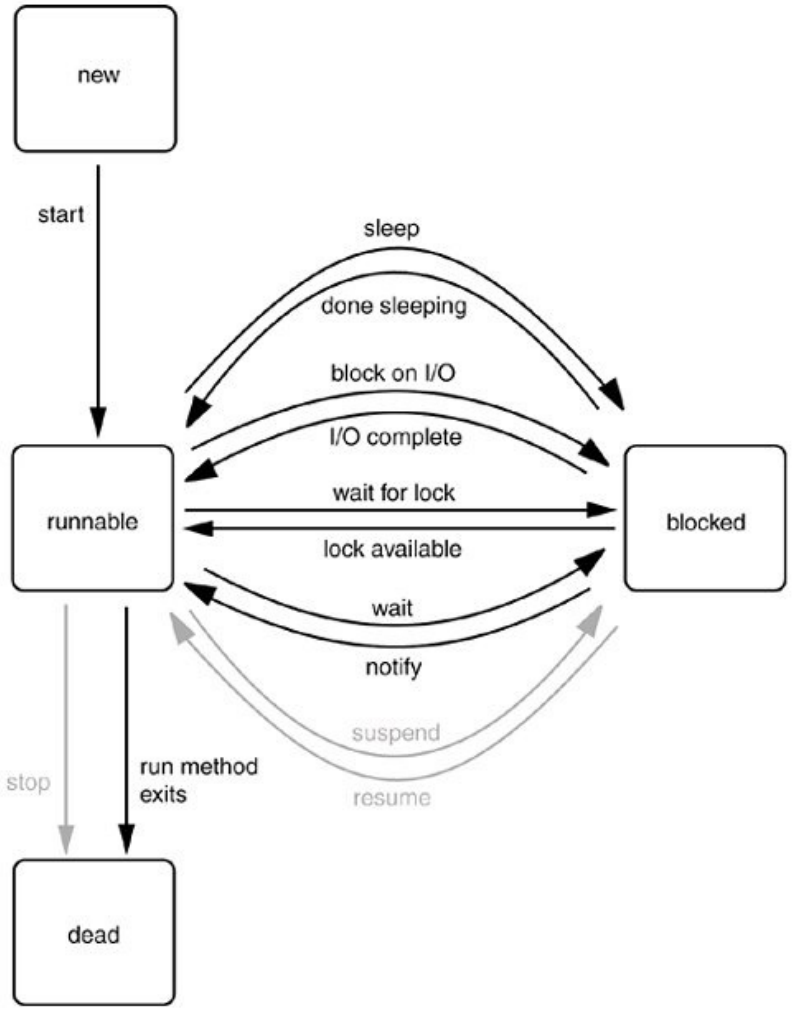
Multi-Thread

- **Estados de linha de execução**
  - Novo
  - Passível de execução
  - Bloqueada
  - Morta
- **Threads novas**
  - Quando você cria uma thread com o operador new, neste estado o método run() não começou a executar.



Multi-Thread

- **Threads passível de execução**
  - Quando você chama o método start, a thread é passível de execução (runnable).
  - *Uma* thread *passível de execução, não significa estar em execução (ela pode estar aguardando na fila de execução de threads).*



Multi-Thread

- **Threads bloqueadas**
  - Alguém chama o método sleep() da thread.
  - A thread chama uma operação que está bloqueando entrada/ Saída (I/O).
  - A thread chama o método wait()
  - A thread tenta bloquear um objeto que está bloqueado por outra execução.
  - Quando uma thread está bloqueada, outra thread é escalada para execução.
- **Threads Mortas**
  - Ela morre de morte natural, pois o método run encerra normalmente.
  - Ela morre abruptamente, pois uma exceção não capturada encerra o método run.
  - Para descobrir se uma thread está viva, use o método isAlive().

Multi-Thread

- **Interrompendo uma Thread**
  - O método run da thread deve testar pela interrupção de sua execução com o utilizando o método isInterrupted()
  - Chamar o metodo interrupted() da thread para solicitar a sua interrupção

```
public void run() {
  for (;;) {
    // Executa algumas operações

    // Testa se a Thread foi interrompida
    if (Thread.interrupted()) {
      // Thread Interrompida, retorne
      return;
    }
  }
}
```

```
public void run() {
  for(;;) {
    try {
      // Executa algumas operações

      // Espera 5 segundos
      Thread.sleep(5000);
    } catch (InterruptedException ex) {
      // Thread Interrompida, retorne
      return;
    }
  }
}
```

Multi-Thread

- **Prioridades da Thread**
  - MIN_PRIORITY (configurada como 1 na classe Thread)
  - NORM_PRIORITY (configurada como 5)
  - MAX_PRIORITY (configurada como 10)
  - A prioridade é atribuida através do método setPriority() da Thread

Sincronismo

- Na maioria dos aplicativos com múltiplas linhas de execução duas ou mais linhas precisam compartilhar o acesso aos mesmos objetos.
- Se duas linhas de execução têm acesso ao mesmo objeto pode acontecer a corrupção dos dados alterados, resultando em objetos danificados

Multi-Thread

- Para resolver este problema existem duas maneiras:
  - Sincronizar o método

```
public synchronized void calcula() {
  // executa operações
}
```

  - Sincronizar o acesso ao objeto

```
public void executa() {
  synchronized (objeto) {
    // executa operações
  }
}
```

Handler

A principal função de Handler é permitir atualizar os dados na tela quando estes são obtidos através da utilização de Threads.

```
public void onClick(final View view) {
    view.setEnabled(false);

    final Handler handler = new Handler();

    new Thread() {
        @Override
        public void run() {
            for (long id : listarIds()) {
                final Album album = leAlbumDaRede(id);

                if (album != null) {
                    handler.post(() → { carregaAlbumNaTela(album); });
                }

                // Aguarda 4 segundos
                SystemClock.sleep( ms: 5000);
            }
            handler.post(() → { view.setEnabled(true); });
        }
    }.start();
}
```

Exemplo

Quanto é necessário o acesso à recursos de rede, tal como a consulta a um WebService via REST, implementamos:

```
private String leJson(final String url) throws Exception {
    code = HttpURLConnection.HTTP_INTERNAL_ERROR;
    json = null;

    try {
        HttpURLConnection con = configConnection(url);

        if (con != null) {
            con.setRequestMethod("GET");
            con.setDoInput(true);
            con.connect();

            code = con.getResponseCode();
            if (code == HttpURLConnection.HTTP_OK) {
                BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
                json = readJson(in);
                in.close();
            }
            con.disconnect();
        }
    } catch (IOException ex) {
        Log.e( tag: "Consultar.in", ex.getMessage());
    }

    if (code != 200)
        Toast.makeText(Main.context,  text: "Falha em localizar o Album", Toast.LENGTH_LONG).show();

    return json;
}
```

Exemplo

Porém este tipo de implementação apresenta falhas por utilizar recursos de rede da Thread Principal o que compromete a responsividade das aplicações Android, assim ao ser executado o método implementado, nos é lançada a exceção **NetworkOnMainThreadException**.

```
android.os.NetworkOnMainThreadException
    at android.os.StrictMode$AndroidBlockGuardPolicy.onNetwork(StrictMode.java:1448)
    at java.net.AbstractPlainSocketImpl.doConnect(AbstractPlainSocketImpl.java:355)
    at java.net.AbstractPlainSocketImpl.connectToAddress(AbstractPlainSocketImpl.java:200)
    at java.net.AbstractPlainSocketImpl.connect(AbstractPlainSocketImpl.java:182)
    at java.net.SocksSocketImpl.connect(SocksSocketImpl.java:356)
    at java.net.Socket.connect(Socket.java:616)
```

Exemplo

```
private String leJson(final String url) throws Exception {
    Thread job = new Thread() {
        public void run() {

            code = HttpURLConnection.HTTP_INTERNAL_ERROR;
            json = null;

            try {
                HttpURLConnection con = configConnection(url);

                if (con != null) {
                    con.setRequestMethod("GET");
                    con.setDoInput(true);
                    con.connect();

                    code = con.getResponseCode();
                    if (code == HttpURLConnection.HTTP_OK) {
                        BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));
                        json = readJson(in);
                        in.close();
                    }
                    con.disconnect();
                }
            } catch (IOException ex) {
                Log.e( tag: "Consultar.in", ex.getMessage());
            }

            if (code != 200)
                Toast.makeText(Main.context,  text: "Falha em localizar o Album", Toast.LENGTH_LONG).show();

        }
    };

    job.start();
    job.join();

    return json;
}
```

A solução é utilizar uma **Thread** para que o *bloco de instruções* seja executada fora da *thread principal.*

O inconveniente desta implementação é que os dados retornarão após a finalização do método.

Para resolver este problema esperamos pela finalização da **Thread** utilizando o método **join()**.

Exemplo

Esta aplicação obtêm uma lista de chaves de acesso de todos os registros para percorrer um a um e apresentar seus dados numa Activity a cada 5 segundos.

Ocorre que nada é apresentado na tela até a finalização do laço **for**.

Isto ocorre porque os dados são obtidos através de Threads distintas da thread principal.

```
public void onClick(View view) {
    view.setEnabled(false);

    for (long id : listarIds()) {
        final Album album = leAlbumDaRede(id);

        if (album != null) {
            carregaAlbumNaTela(album);
        }

        // Aguarda 4 segundos
        SystemClock.sleep( ms: 5000);
    }
    view.setEnabled(true);
}
```

Exemplo

Ao utilizarmos outra Thread para a execução desta ação não temos sucesso e recebemos um **RuntimeException** por tentar acessar a thread principal a partir de outra Thread.

```
public void onClick(final View view) {
    view.setEnabled(false);

    new Thread() {
        @Override
        public void run() {
            for (long id : listarIds()) {
                final Album album = leAlbumDaRede(id);

                if (album != null) {
                    carregaAlbumNaTela(album);
                }

                // Aguarda 4 segundos
                SystemClock.sleep( ms: 5000);
            }
            view.setEnabled(true);
        }
    }.start();
}
```

```
java.lang.RuntimeException: Can't create handler inside thread that has not called Looper.prepare()
    at android.os.Handler.<init>(Handler.java:203)
    at android.os.Handler.<init>(Handler.java:117)
    at android.view.textservice.SpellCheckerSession$1.<init>(SpellCheckerSession.java:104)
    at android.view.textservice.SpellCheckerSession.<init>(SpellCheckerSession.java:104)
```

Exemplo

Todo acesso entre Thread deve ser efetuado através do **Handler**.

Obtemos um *Handler* da *Three* a qual queremos nos comunicar e utilizamos os métodos **handleMessage()** ou **post()** para enviar mensagens ou objetos do tipo **Runnable** para a outra *Thread*.

```
public void onClick(final View view) {
    view.setEnabled(false);

    final Handler handler = new Handler();

    new Thread() {
        @Override
        public void run() {
            for (long id : listarIds()) {
                final Album album = leAlbumDaRede(id);

                if (album != null) {
                    handler.post(new Runnable() {
                        @Override
                        public void run() {
                            carregaAlbumNaTela(album);
                        }
                    });
                }

                // Aguarda 4 segundos
                SystemClock.sleep( ms: 5000);
            }
            handler.post(new Runnable() {
                @Override
                public void run() {
                    view.setEnabled(true);
                }
            });
        }
    }.start();
}
```

AsyncTask

A **AsyncTask** é uma implementação que simplifica a utilização de *threads* e *handlers* de forma a isolar as fases de *pré inicialização*, *execução em background*, *atualização de views* e *pós execução.*

Ao lado temos a implementação da solução anterior utilizando *AsyncTask.*

```
@SuppressLint("StaticFieldLeak")
public void onClick(final View view) {
    new AsyncTask<Void, Album, Void>() {
        @Override
        protected Void doInBackground(Void... voids) {
            for (long id : listarIds()) {
                final Album album = leAlbumDaRede(id);

                publishProgress(album);

                // Aguarda 4 segundos
                SystemClock.sleep( ms: 5000);
            }

            return null;
        }

        @Override
        protected void onProgressUpdate(Album... albuns) {
            if (albuns[0] != null) {
                carregaAlbumNaTela(albuns[0]);
            }
        }

        @Override
        protected void onPostExecute(Void voids) {
            view.setEnabled(true);
        }
    }.execute();
}
```

Acesso HTTP

Para o acesso **HTTP** pelo Android a qualquer WebService é necessária a utilização da classe **HttpURLConnection**.

Com ela parametrizamos o tipo do conteúdo conjunto de caracteres, limite de tempo de resposta, etc

```
public static HttpURLConnection configConnection(String url)
        throws MalformedURLException, IOException {
    HttpURLConnection con = (HttpURLConnection) new URL(url).openConnection();
    con.setRequestProperty("Content-Type", "application/json");
    con.setRequestProperty("Accept", "application/json");
    con.setRequestProperty("Accept-Charset", "utf-8");
    con.setConnectTimeout(15000);
    con.setReadTimeout(10000);

    return con;
}
```

Leitura do JSON

Após uma consulta a um WebService normalmente é necessário efetuar a leitura dos dados retornados.

```
HttpURLConnection con = configConnection(url);

if (con != null) {
    con.setRequestMethod("GET");
    con.setDoInput(true);
    con.connect();

    code = con.getResponseCode();
    if (code == HttpURLConnection.HTTP_OK) {
        BufferedReader in = new BufferedReader(new InputStreamReader(con.getInputStream()));

        StringBuilder buffer = new StringBuilder( capacity: 100);
        String linha = in.readLine();
        while (linha != null) {
            buffer.append(linha);
            linha = in.readLine();
        }

        json = buffer.toString();

        in.close();
    }
    con.disconnect();
}
```

Os dados formatados como JSON podem ser lidos como uma **String** da mesma forma como lemos de um arquivo.

ObjectMapper

Podemos transformar os dados contidos num JSON em um Objeto Java utilizando o **ObjectMapper**.

```
dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:26.1.0'
    implementation 'com.android.support.constraint:constraint-layout:1.0.2'
    implementation 'com.android.support:design:26.1.0'

    implementation 'org.codehaus.jackson:jackson-mapper-asl:1.9.13'
}
```

Para utilizar o **ObjectMapper** em um projeto Android é necessário incluir a implementação *jackson-mapper* no arquivo *build.gradle do projeto.*

ObjectMapper

Depois basta passar o JSON para um *StringReader* e chamar o método **readValue()** do **ObjectMapper**.

Como resultado teremos o Objeto Java.

```
String json = leJson( url: url + "album/" + id);

if (!json.isEmpty()) {
    ObjectMapper mapper = new ObjectMapper();
    Album album = mapper.readValue(new StringReader(json), Album.class);
}
```

FIM