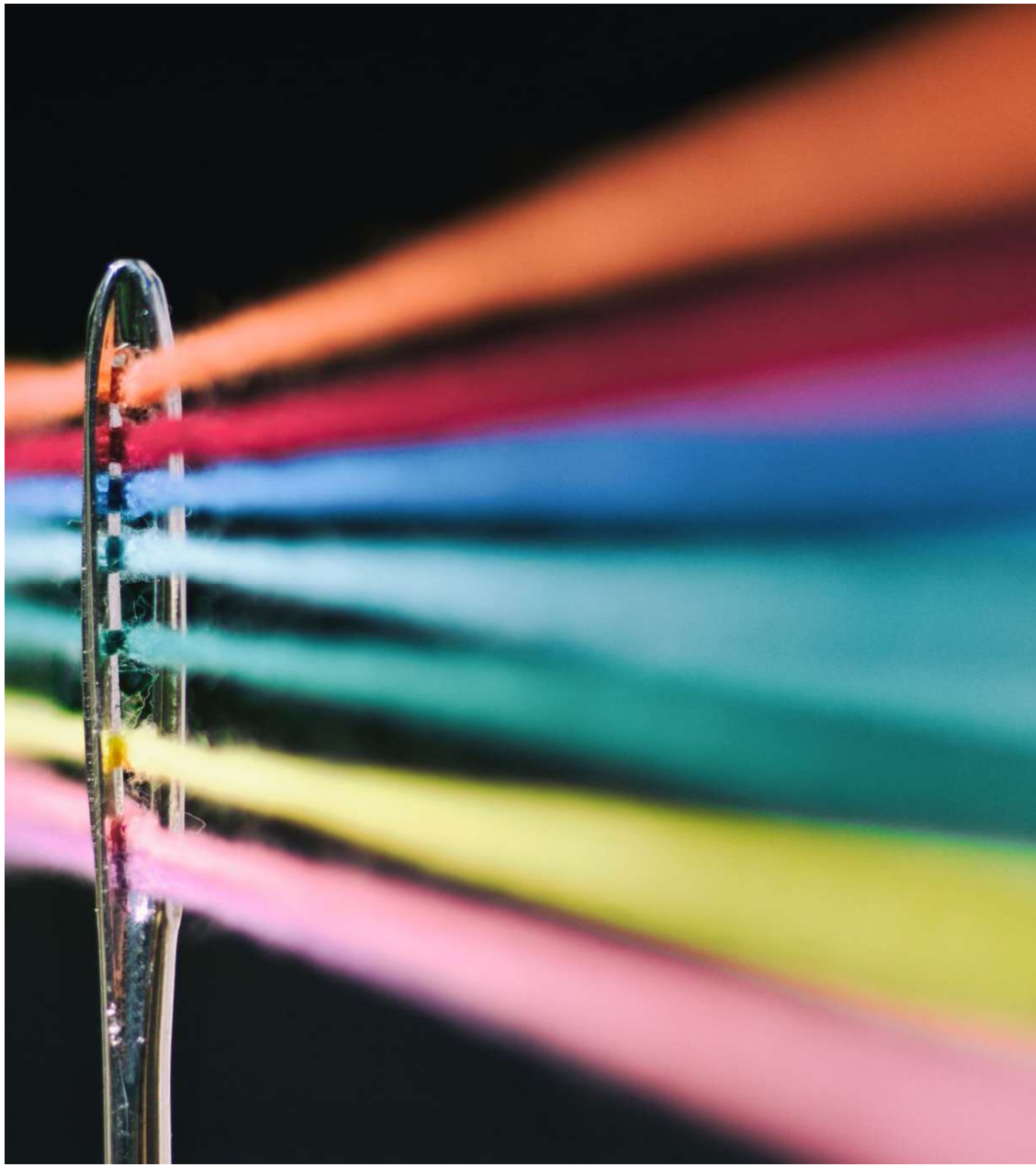
Desenvolvimento de Apps para Mobile

Fabrício Tonetto Londero

Aula 10 - Background Tasks

Threads

- Threads são miniprocessos criados em tempo de execução;
- Cada thread possui sua própria porção de código a ser executada;
- Fácil de trabalhar, porém, limitadas no momento em que há a necessidade de se atualizar componentes da interface;



Threads – exemplo 1

- Primeiramente, define-se uma classe que herde a classe `Thread` e implemente o método `run()` :

```
class minhaThread extends Thread{

    public void run(){
        Log.i("teste", "thread iniciada");
    }
}
```

Threads – exemplo 1

- Em seguida, instanciamos um objeto da classe definida anteriormente, e fazemos uma chamada ao método `start()`;

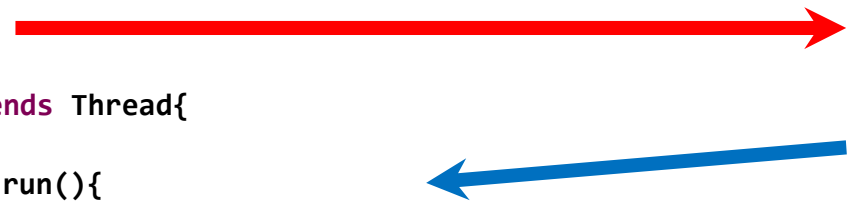
```
public class AulaBR_exThreads1 extends Activity {
        @Override
        protected void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.activity_main);

                minhaThread t = new minhaThread();
                t.start();
        }

        class minhaThread extends Thread{

                public void run(){
                        Log.i("teste", "thread iniciada");
                }
        }

}
```



A chamada ao `start()` faz executar o método `run()` da Thread.

Threads – exemplo 2

Problema: tentar atualizar alguma informação na interface na thread pode dar problemas, ou não...

```
public class AulaBR_exThreads1 extends Activity {
	@Override
	protected void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.activity_main);

		minhaThread t = new minhaThread();
		t.start();
	}

	class minhaThread extends Thread{
		public void run(){
			TextView t = (TextView) findViewById(R.id.textView1);
			t.setText("Thread Iniciada");
		}
	}

}
```

AsyncTask

- **AsyncTask** é uma classe usada para executar tarefas de longo tempo de execução (como acessar banco de dados, internet, etc.) fora da thread principal, evitando travar a UI.
- **AsyncTastk** encapsula todo um processo pesado de forma bastante organizada, além de fornecer métodos para modificar a Interface Gráfica **antes**, **durante** e **depois** da execução deste processo

AsyncTask - Estrutura

- Primeiramente, definimos uma classe que herde AsyncTask:

```
class Contador extends AsyncTask<Param1, Param2, Param3> {

}
```

- Sendo que:
  - **Param1**: tipo de dado/objeto que deseja passar por parâmetro no momento em que a tarefa é colocada em execução;
  - **Param2**: tipo de dado/objeto que deseja passar como resultado parcial da operação
    - geralmente **int**, indicando a porcentagem concluída;
  - **Param3**: tipo de dado/objeto que deseja receber como resultado da operação;

Caso não for passar informações, usa-se **Void**

AsyncTask - Estrutura

- Descrição dos métodos:

- **doPreExecute:**
  - Método executado antes da tarefa principal ser iniciada.

- **doInBackground:**
  - método que irá executar todo o processamento pesado da nossa tarefa.

- **onProgressUpdate:**
  - Executa sempre que for feita uma chamada ao método publishProgress. É geralmente utilizada para atualizar alguma informação na interface com o usuário;

- **onPostExecute:**
  - Executado quando o método doInBackground termina sua execução. **Recebe como parâmetro o retorno do método doInBackground**;

```java
class MinhaTask extends AsyncTask<Void, Integer, String> {

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        progresso.setVisibility(View.VISIBLE);
        texto.setText("Iniciando...");
    }

    @Override
    protected String doInBackground(Void... voids) {
        for (int i = 1; i <= 5; i++) {
            try {
                Thread.sleep(1000);
                publishProgress(i * 20);
            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        return "Concluído!";
    }
```

```
    @Override
    protected void onProgressUpdate(Integer... values) {
        super.onProgressUpdate(values);
        texto.setText("Progresso: " + values[0] + "%");
        progresso.setProgress(values[0]);
    }

    @Override
    protected void onPostExecute(String resultado) {
        super.onPostExecute(resultado);
        progresso.setVisibility(View.GONE);
        texto.setText(resultado);
    }
}
```

Services

- Um serviço é um componente que executa em segundo plano sem interação direta com o usuário;
- Como um serviço não possui interface, ele não está ligado ao ciclo de vida de uma activity:
  - Pode ser iniciado em uma activity, porém, pode continuar executando quando a mesma termina;
- São utilizados em a**tividades repetitivas** e de **longa duração**, tais como downloads, operações de sincronização, processamento de dados, etc.

Services

- Importante!
  - Um serviço não é um processo separado, a não ser que seja especificado para isso. Se não for, será rodado no mesmo processo da aplicação
  - Um serviço não é uma thread. Não é, portanto, uma maneira apropriada de se realizar tarefas fora do fluxo principal;
  - Deve ser utilizado quando se deseja avisar o SO sobre alguma tarefa que deve ser executada em segundo plano, ou ainda para expor algumas funcionalidades para outras aplicações.

Services - Exemplo

- Primeiramente, um **`Service`** precisa ser adicionado no **`AndroidManifest.xml,`** dentro de **`<application>`**:

```
<service
      android:name="MeuServico"
      android:icon="@drawable/ic_launcher"
      android:label="Meu Servico" >
</service>
```

Services - Exemplo

Em seguida, criamos uma classe java com o mesmo nome do serviço definido anteriormente:

```
public class MeuServico extends Service {

	@Override
	public void onCreate() {

	}

	@Override
	public int onStartCommand(Intent intent, int flags, int startId) {
		// TODO do something useful
		return Service.START_NOT_STICKY;
	}

	@Override
	public IBinder onBind(Intent intent) {
		// TODO for communication return IBinder implementation
		return null;
	}
}
```

Services - Exemplo

E então disparamos o serviço de algum outro lugar (de outro serviço, de uma activity, etc...)

```
public class MainActivity extends Activity {

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

                // cria-se uma intent para o serviço
                Intent i= new Intent(this, MeuServico.class);
                // e podemos adicionar parâmetros também
                i.putExtra("param", "valor_param");

                this.startService(i);
        }

}
```

Faz executar o método `onCreate()` do serviço, e em seguida o `onStartcommand(...)`

Services

- Para parar um serviço, podemos utilizar:
  - `Context.stopService(Intent it)`, de fora do serviço, para parar a intent que deu início ao serviço;
  - `stopSelf()`, de dentro do serviço

Services

O método `startCommand` retorna um inteiro:

- START_NOT_STICKY
  - Se o SO mata o serviço depois que o método `onStartCommand()` retornar um valor, não recria o serviço.
- START_STICKY
  - Se o SO mata o serviço depois que o método `onStartCommand()` retornar um valor, recria o serviço e chama o método `onStartCommand()`, mas não reentrega a última Intent.
- START_REDELIVER_INTENT
  - Se o SO mata o serviço depois que o método `onStartCommand()` retornar um valor, recria o serviço e chama o método `onStartCommand()`, e reentrega a última Intent.

Services

- É possível criar um processo separado para executar o serviço;

- Para isso, basta adicionar a seguinte linha na sua definição no AndroidManifest.xml:

```
<service
        android:name="MeuServico"
        android:process=":meu_processo"
        android:icon="@drawable/ic_launcher"
        android:label="Meu Servico" >
 </service>
```

```java
//deixar ela static ou tornar uma classe externa
public static class MeuServico extends Service {

    private Handler handler = new Handler();
    private int segundos = 0;

    @Override
    public void onCreate() {

    }

    @Override
    public int onStartCommand(Intent intent, int flags, int startId) {

        segundos = 0;

        handler.postDelayed(new Runnable() {
            @Override
            public void run() {
                Log.i("Service", "run...");
                segundos++;
                if (segundos < 10) {
                    handler.postDelayed(this, 1000);
                } else {
                    Log.i("Service", "10 segundos depois...");
                    Toast.makeText(getApplicationContext(), "10 segundos se passaram!", Toast.LENGTH_LONG).show();
                    stopSelf();
                }
            }
        }, 1000);
        return Service.START_NOT_STICKY;
    }

    @Nullable
    @Override
    public IBinder onBind(Intent intent) {
        return null;
```

IntentServices

- Uma alternativa a Services são as `IntentServices`;
- Podem ser utilizados em tarefas de duração mais longa, geralmente sem comunicação com a Activity principal da aplicação;
- Uma `IntentService` é disparada através de uma Intent, e então uma nova thread é criada e o método `onHandleIntent()` é executado;
- Uma `IntentService` só pode ser disparada pela thread principal da aplicação
  - Enquanto que uma `Service` poderia ser disparada por qualquer thread.

IntentServices - Exemplo

Primeiramente, um `Service` precisa ser adicionado no `AndroidManifest.xml`:

```
<service
     android:name="MinhaIntentService" >
</service>
```

IntentServices - Exemplo

Em seguida, criamos uma classe java com o mesmo nome da IntentService definida anteriormente:

```
public class MinhaIntentService extends IntentService {

	@Override
	public void onCreate() {
		super.onCreate();
		Log.i("teste", "Intent Service criada");

	}

	@Override
	protected void onHandleIntent(Intent arg0) {
		Log.i("teste", "Intent Service iniciada");

	}

	public MinhaIntentService() {
		super("MinhaIntentService");
	}

}
```

IntentServices - Exemplo

E então disparamos a IntentService da **thread principal**

```
public class AulaBR_exIntentService1 extends Activity {

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

                // Cria-se uma intent para a IntentService
                Intent i= new Intent(this, MinhaIntentService.class);

                this.startService(i);
        }

}
```

Faz executar o método `onCreate()` do serviço, e em seguida o `onHandleIntent(...)`

```java
public static class MinhaIntentService extends IntentService {

    @Override
    public void onCreate() {
        super.onCreate();
        Log.i("teste", "Intent Service criada");
    }

    @Override
    protected void onHandleIntent(Intent arg0) {
        Log.i("teste", "IntentService iniciado");
        try {
            Thread.sleep(5000);
        } catch (InterruptedException e) {
            e.printStackTrace();
        }

        Handler handler = new Handler(Looper.getMainLooper());
        handler.post(() -> Toast.makeText(getApplicationContext(), "Tarefa finalizada!", Toast.LENGTH_SHORT).show());
    }

    public MinhaIntentService() {
        super("MinhaIntentService");
    }
}
```

Service vs IntentService

`Service` usa a thread principal da aplicação, logo, pode acabar bloqueando toda a aplicação;

`IntentService` é disparada por Intent, que faz com que saia do fluxo principal da aplicação para uma thread secundária;

`Service` só para manualmente, por exemplo, através do método stopSelf();

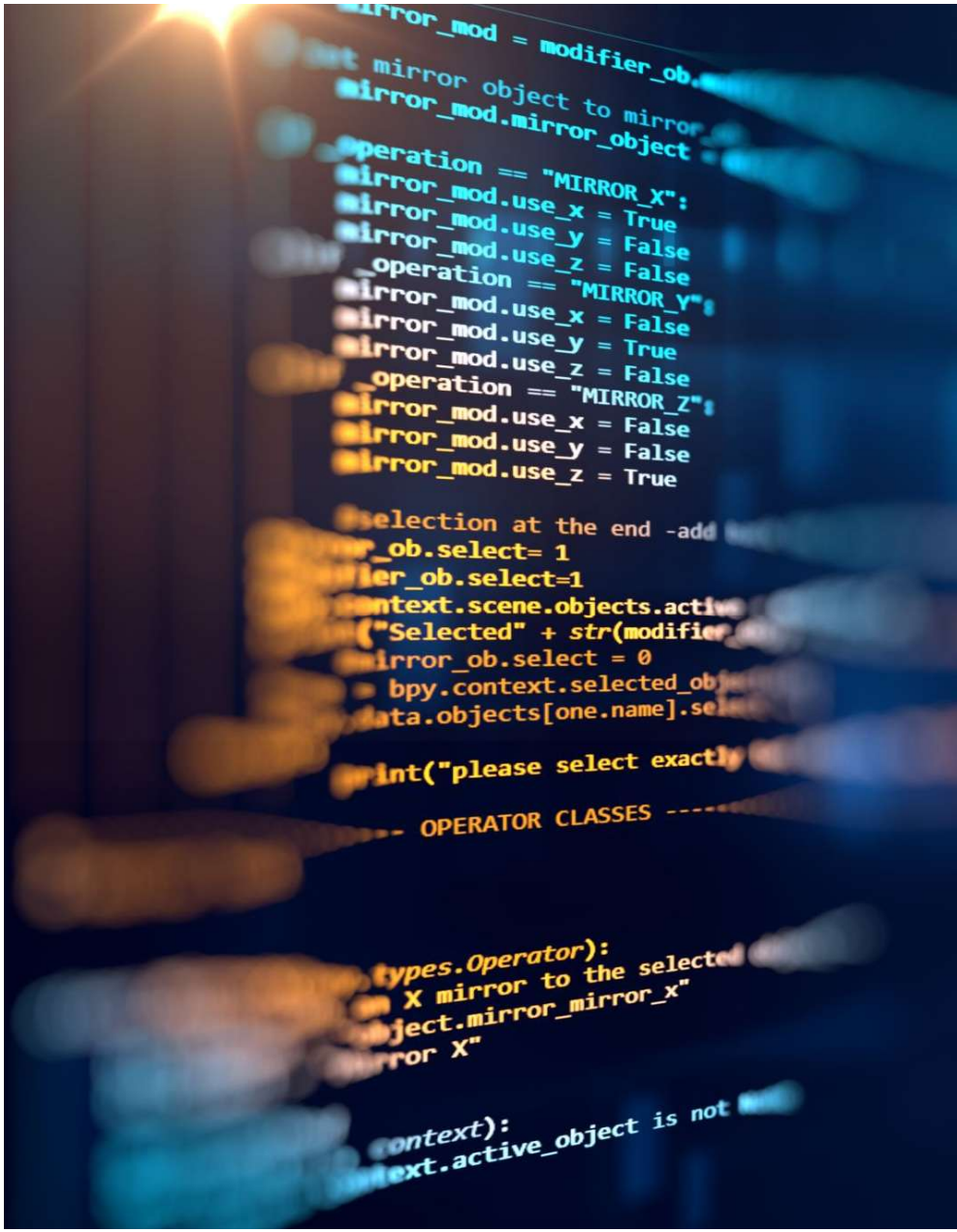
`IntentService` para automaticamente quando não há nada a ser executado;

Resumo

Thread

- **O que é**: Execução paralela, básica do Java.
- **Uso ideal**: Processos simples e curtos, como cálculo ou delay.
- **Cuidados**: Não pode atualizar a UI diretamente; precisa usar Handler.

Exemplo de uso:
Esperar 3 segundos e fazer algo (ex: delay de splash screen).



Resumo

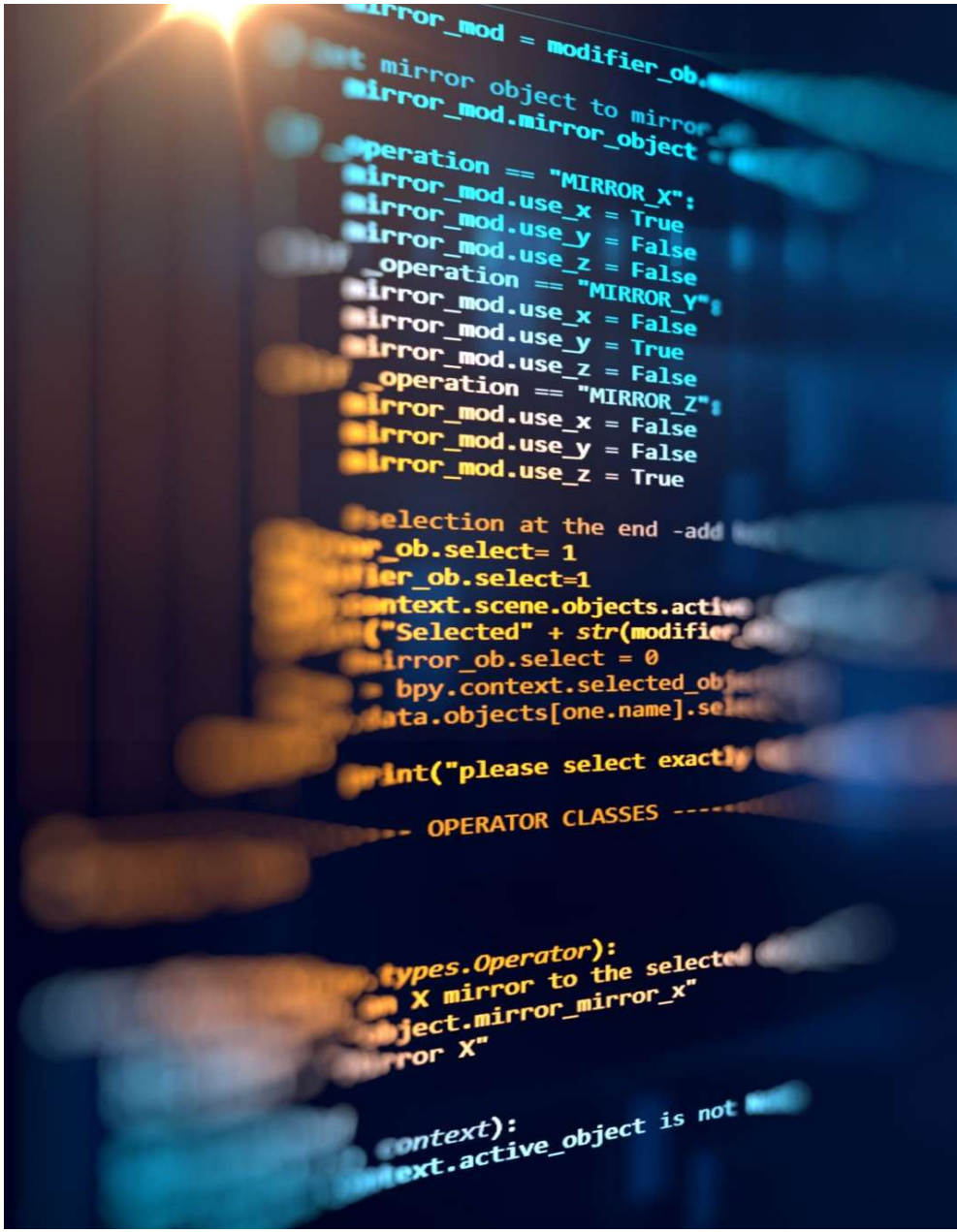
AsyncTask ⚠️ **(*obsoleto a partir do Android 11*)**

- **O que é**: Classe Android para facilitar o uso de Thread com comunicação com a interface.
- **Uso ideal**: Tarefas curtas e que interajam com a interface (ex: carregar dados).
- **Cuidados**: Foi **descontinuado**, evite em novos projetos.
- **Exemplo de uso**:
Buscar dados de uma API e atualizar a tela.

Resumo

Service

- **O que é**: Componente que roda **em segundo plano**, mesmo com o app fechado.
- **Uso ideal**: Tarefas longas ou contínuas (ex: player de música, monitor GPS).
- **Cuidados**: Não roda em uma Thread separada por padrão. Use Thread dentro dele se for tarefa longa.
- **Exemplo de uso**:
App de rastreamento que envia localização periodicamente.

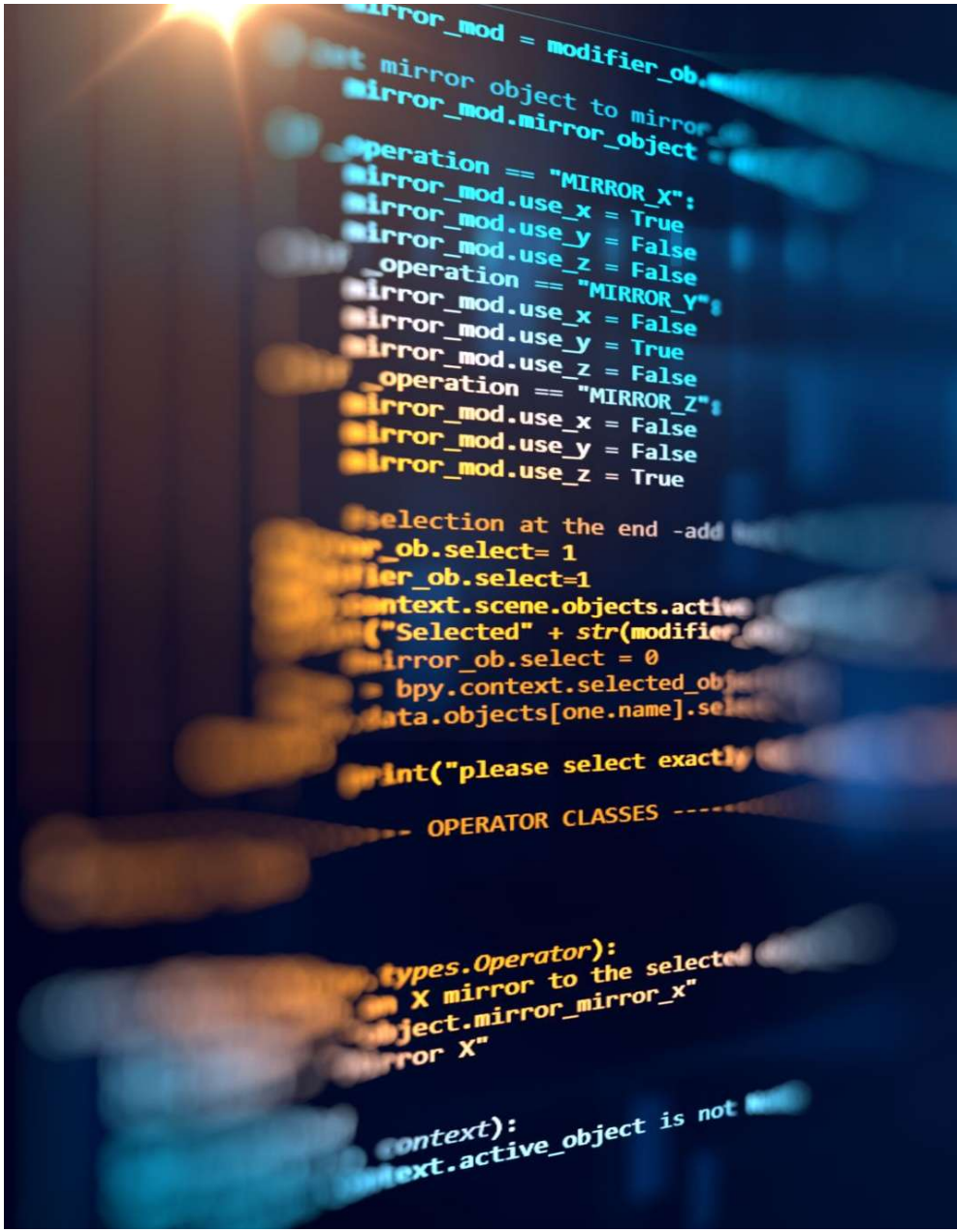


Resumo

IntentService ⚠️ (*também descontinuado, mas útil para ensino*)

- **O que é**: Subclasse de Service que já roda em uma **Thread separada** e **encerra sozinho**.
- **Uso ideal**: Tarefas **curtas** que precisam rodar em background.
- **Cuidados**: Também descontinuado
- **Exemplo de uso**:

Fazer upload de um arquivo ou salvar dados em background.



Resumo

- Tarefa rápida e isolada → Thread
- Precisa interagir com a UI→Thread + Handler (em vez de AsyncTask)
- Tarefa em segundo plano mesmo com app fechado → Service
- Tarefa rápida em segundo plano (com fim automático)→IntentService

Exercício

Desenvolva um simulador de download em que o usuário digita em um EditText o tamanho de um arquivo e que, ao clicar em um botão, é feita uma simulação de um download, além de desabilitar o botão no começo deste processo. O botão deve ser habilitado somente quando o download terminar. Além disso, quando o download terminar, é exibida uma mensagem na tela informando isso.