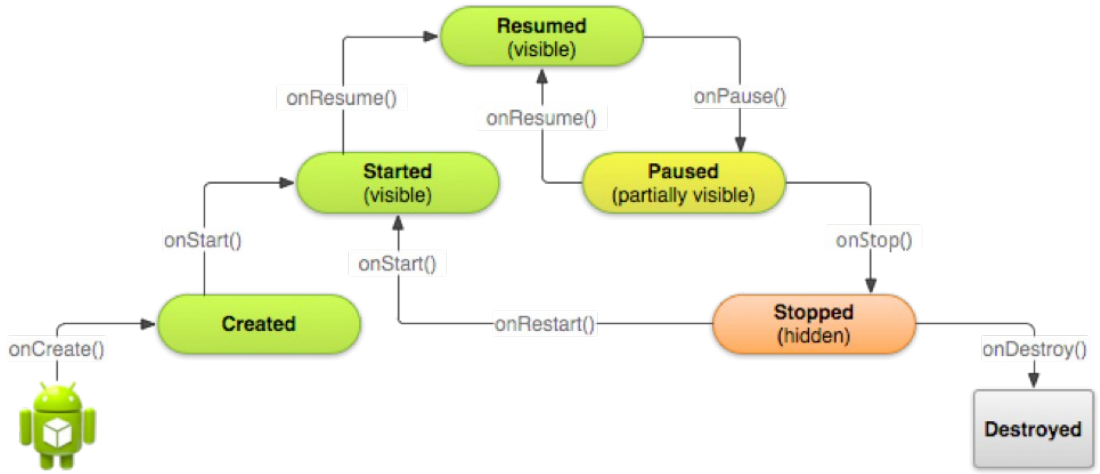
O conteúdo desta aula baseia-se na estrutura conhecida como ciclo de vida de uma Activity. Ela é apresentada na Figura 1.

Figura 1.




**Exemplo prático**

Neste exemplo iremos ilustrar o uso dos métodos do ciclo de vida de uma Activity para ativar e desativar o hardware de GPS conforme a aplicação ganha e perde o foco do usuário.

**Passo 1 (Criando o projeto)** No Android Studio, crie um novo projeto com os seguintes dados.

Template: Basic Activity
Name: Ciclo de Vida GPS e Mapas
Package Name: br.com.bossini
Save Location: Mantenha o valor padrão
Language: Java
Minimum API Level: API 23: Android Marshmallow

Mantenha qualquer outro campo com seu valor padrão e clique em Finish.

**Passo 2 (Inspecionando os arquivos criados)** Note que agora temos dois arquivos .xml para descrever a tela: activity_main.xml e content_main.xml. O primeiro inclui o segundo. No primeiro, definimos um layout principal e o botão flutuante. Ele inclui o segundo, no qual iremos adicionar o conteúdo da tela.

**Passo 3 (Declarando os componentes para uso do GPS)** O Android possui diferentes APIs para lidar com GPS. Iremos utilizar as classes LocationManager e LocationListener para isso. LocationManager é uma classe que permite lidar com o hardware de GPS. Iremos criar uma instância de LocationListener e sobrescrever um método de interesse, que será chamado quando houver alguma atualização na localização do

usuário. Trata-se de uma aplicação do conhecido padrão de projeto Observer. Na classe MainActivity, declare as variáveis de instância mostradas na Listagem 3.1.

Listagem 3.1

```
private LocationManager locationManager;
private LocationListener locationListener;
```

**Passo 4 (Inicializando a variável locationManager)** LocationManager é um serviço do sistema operacional que podemos obter por meio do método getSystemService. No método onCreate, chame esse método como mostra a Listagem 4.1.

Listagem 4.1

```
locationManager = (LocationManager)
getSystemService(Context.LOCATION_SERVICE);
```

**Passo 5 (Inicializando a variável locationListener)** Agora iremos criar uma instância de LocationListener. Como LocationListener é uma interface, precisaremos criar uma interface que a implementa. Faremos isso por meio de uma classe interna anônima. No método onCreate, escreva a atribuição da Listagem 5.1. Não se esqueça de deixar o Android Studio te ajudar completando o código para você.

Listagem 5.1

```
locationListener = new LocationListener() {
    @Override
    public void onLocationChanged(Location location) {

    }
    @Override
    public void onStatusChanged(String provider, int status, Bundle
extras) {
    }
    @Override
    public void onProviderEnabled(String provider) {
    }
    @Override
    public void onProviderDisabled(String provider) {
    }
};
```

**Passo 6 (Ativando o hardware de GPS)** Note que até então não ativamos o hardware de GPS. Isso será feito quando registrarmos o locationListener como observador do locationManager. Nesta aplicação iremos ligar o GPS quando ela ganhar o foco do usuário e desligá-lo quando a aplicação perder o foco. Por isso, iremos escolher um par de métodos do ciclo de vida, por exemplo, onStart para ativar e onStop para desativar. Note que ativar o hardware de GPS é uma atividade que requer que o usuário forneça permissão em tempo de execução. Daí o código da Listagem 6.1.

Listagem 6.1

```
@Override
protected void onStart() {
    super.onStart();
    //a permissão já foi dada?
    if (ActivityCompat.checkSelfPermission(
            this, Manifest.permission.ACCESS_FINE_LOCATION) ==
            PackageManager.PERMISSION_GRANTED){
        //somente ativa
        //a localização é obtida via hardware, intervalo de 0
segundos e 0 metros entre atualizações

locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
0, 0, locationListener);
    }
    else{
        //permissão ainda não foi nada, solicita ao usuário
        //quando o usuário responder, o método
onRequestPermissionsResult vai ser chamado
        ActivityCompat.requestPermissions(this,
                new String[]
{Manifest.permission.ACCESS_FINE_LOCATION}, 1001);
    }
}
```

**Passo 7 (Adicionando a permissão no AndroidManifest.xml)** Além de solicitar a permissão em tempo de execução, é necessário informá-la no arquivo AndroidManifest.xml. Veja a Listagem 7.1. A tag uses-permission fica fora da tag application.

Listagem 7.1

```
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

**Passo 8 (Tratando a resposta do usuário)** Quando pedimos permissão em tempo de execução ao usuário, um diálogo será exibido e ele bloqueará a execução do programa até que o usuário decida o que deseja. Uma vez que ele interaja com o diálogo, a execução será desbloqueada desencadeando uma chamada ao método onRequestPermissionsResult, o qual possui parâmetros que nos permite identificar qual foi a resposta do usuário. Dependendo da resposta do usuário iremos ativar ou não o hardware de GPS. Veja o método na Listagem 8.1.

Listagem 8.1

```
@Override
public void onRequestPermissionsResult(int requestCode, @NonNull
String[] permissions, @NonNull int[] grantResults) {
    if (requestCode == 1001){
        if (grantResults.length > 0 && grantResults[0] ==
PackageManager.PERMISSION_GRANTED){
            //permissão concedida, ativamos o GPS
            if (ActivityCompat.checkSelfPermission(this,
                    Manifest.permission.ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED){

locationManager.requestLocationUpdates(LocationManager.GPS_PROVIDER,
0, 0, locationListener);
            }
        }
        else{
            //usuário negou, não ativamos
            Toast.makeText(this, getString(R.string.no_gps_no_app),
Toast.LENGTH_SHORT).show();
        }
    }
}
```

**Passo 9 (Criando a string e trocando o valor hard coded)** Note que usamos um recurso de string chamado no_gps_no_app sem que ele exista. Precisamos criá-lo. Para isso, abra o arquivo strings.xml (na pasta values) e adicione o código da Listagem 9.1.

Listagem 9.1

```
<string name="no_gps_no_app">Sem o GPS o aplicativo não
funciona</string>
```

Note também que utilizamos o código 1001 para representar a requisição por permissão de uso do hardware de GPS. Isso ocorre pois em um aplicativo podemos pedir diferentes permissões e o Android utiliza um número inteiro para identificá-las e permitir que elas sejam diferenciadas quando o método onRequestPermissionsResult for chamado. É uma péssima prática manter o valor fixo como está no código, o que caracteriza o uso do anti-pattern “números mágicos”. Quando usamos esse anti-pattern, nosso código possui números espalhados que fazem com que a aplicação supostamente funcione mas cujo significado é, em geral, obscuro para quem tenta inspecionar o código. Assim, crie a constante da Listagem 9.2 e substitua as ocorrências do número 1001 por ela.

Listagem 9.2

```
private static final int REQUEST_CODE_GPS = 1001;
```

Nota: O número 1001 não tem nada de especial. Qualquer número inteiro positivo serve, basta não repetir o mesmo número para requisições diferentes.

**Passo 10 (Desabilitando o hardware de GPS)** A fim de economizar bateria, iremos desabilitar o hardware de GPS quando a aplicação perder o foco do usuário. Para isso, iremos utilizar o método onStop, que é naturalmente o complemento do método onStart. Veja sua implementação na Listagem 10.1.

Listagem 10.1

```
@Override
protected void onStop() {
    super.onStop();
    locationManager.removeUpdates(locationListener);
}
```

**Passo 11 (Ajustando o TextView para exibir as coordenadas obtidas)** A fim de exibir as coordenadas obtidas pelo GPS, iremos utilizar o TextView já existente no aplicativo. Para isso, faça o seguinte:

11.1 Abra o arquivo content_main.xml e adicione o id coordenadasTextView ao TextView ali existente (**android:id="@+id/locationTextView"**).

11.2 Apague a propriedade text do TextView que exibe atualmente o texto Hello World.

11.3 No arquivo MainActivity.java, adicione a variável de instância da Listagem 11.1.

Listagem 11.1

```
private TextView locationTextView;
```

11.4 No método onCreate, inicialize a variável locationTextView com o código da Listagem 11.2. Lembre-se de que isso só pode ocorrer depois de o método setContentView ser executado. Assim, coloque essa instrução logo depois da chamada ao método setContentView.

Listagem 11.2

```
locationTextView = findViewById(R.id.locationTextView);
```

**Passo 12 (Exibindo as coordenadas)** Conforme a localização do usuário muda, o método onLocationChanged (do LocationManager, que já criamos) é chamado. Agora iremos implementar esse método de modo que ele pegue as coordenadas latitude e longitude e simplesmente faça com que o locationTextView as exiba. Veja a Listagem 12.1.

Listagem 12.1

```
@Override
public void onLocationChanged(Location location) {
    double lat = location.getLatitude();
    double lon = location.getLongitude();
    locationTextView.setText(String.format("Lat: %f, Long: %f", lat,
lon));
}
```

**Passo 13 (Buscando restaurantes por perto)** Quando o usuário clicar no botão flutuante, queremos que uma busca por restaurantes seja realizada e um mapa do Google Maps os exiba. Para isso, precisamos:

13.1 Criar duas variáveis de instância para guardar latitude e lontitude. Veja a Listagem 13.1.

Listagem 13.1

```
private double latitudeAtual;
private double longitudeAtual;
```

13.2 Atualizar os valores das novas variáveis no método onLocationChanged, como mostra a Listagem 13.2.

Listagem 13.2

```
@Override
public void onLocationChanged(Location location) {
    double lat = location.getLatitude();
    double lon = location.getLongitude();
    latitudeAtual = lat;
    longitudeAtual = lon;
    locationTextView.setText(String.format("Lat: %f, Long: %f", lat,
lon));
}
```

13.3 Utilizar os valores no método onClick do observer já registrado no botão flutuante. Veja a Listagem 13.3.

Listagem 13.3

```
@Override
public void onClick(View view) {
    Uri gmmIntentUri =
            Uri.parse(String.format("geo:%f,%f?q=restaurantes",
                    latitudeAtual, longitudeAtual));
    Intent mapIntent = new Intent(Intent.ACTION_VIEW, gmmIntentUri);
    mapIntent.setPackage("com.google.android.apps.maps");
    startActivity(mapIntent);
}
```