VOLTAR

Orientação a Objetos com DART

Nessa aula você aprenderá os conceitos básicos de orientação à objetos e como aplicá-los na linguagem Dart. Veremos como criar classes na linguagem, como criar construtores, fazer o encapsulamento e herança no Dart.

NESTE TÓPICO

Classes e Objetos em Dart

Nesta aula vamos começar a aprender sobre orientação a objetos com Dart e fique tranquilo(a), pois além de muito intuitivo este conceito, é fácil de entender. Este é um conceito muito importante e muito usado, inclusive, pelo mercado profissional de desenvolvimento de aplicações (FELIX, 2017).

Bom, repare à sua volta e você verá que há vários objetos ao seu redor, dos mais distintos. Nós estamos cercados por objetos, o tempo todo. Dentro da programação podemos utilizar o mesmo conceito de objetos do mundo real, para deixá-la mais simples e muito mais organizada.

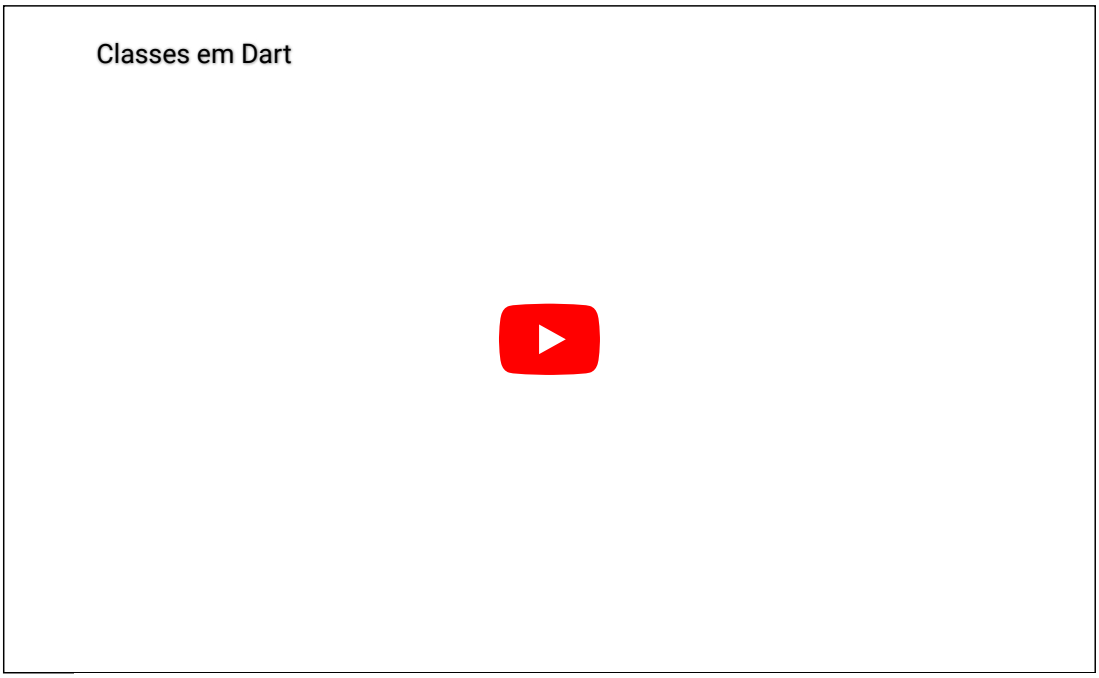
Os objetos do mundo real (que vivemos) possuem essencialmente três coisas: Nome, características e comportamentos. O nome é o que classifica o objeto dentro do mundo como, por exemplo, uma tesoura, um celular, um computador etc. Já as características são o que cada objeto possui, como cor, modelo, código de barras, peso, largura, altura, idade etc. Finalmente os comportamentos são o que os objetos fazem, ou seja, as ações que podemos tomar com cada um dos objetos.

Na programação este conceito é muito parecido. O nome é, de fato, o nome dos objetos. As características são suas variáveis (ou atributos) e os comportamentos são os métodos, que usamos para invocar alguma ação do objeto.

Na programação ainda temos que considerar que os objetos só existem quando o programa está em execução, como no mundo real. Se pensarmos que estamos inseridos em um programa de computador, então os objetos que estão a nossa volta só existem pois foram programados. No código-fonte nós não temos o objeto em si, mas sim a classe que gera o objeto (SCHWARZMÜLLER, 2020).

A classe, então, é nada mais do que um modelo de como o objeto deve ser construído com o programa em execução. Cada objeto, na programação, chamamos de instância de uma determinada classe. Podemos pensar que a classe é como uma ?planta? (da arquitetura) para a construção do objeto.

Para entendermos como este conceito funciona na programação em Dart, vamos desenvolver o conceito de uma classe ?Pessoa? que possui uma série de atributos e comportamentos. O vídeo abaixo mostra um exemplo de criação da classe Pessoa em Dart. Essa classe e este conceito serão utilizados no desenvolvimento desta aula. Todos os códigos foram implementados utilizando-se a ferramenta online disponível em: http://dartpad.dev



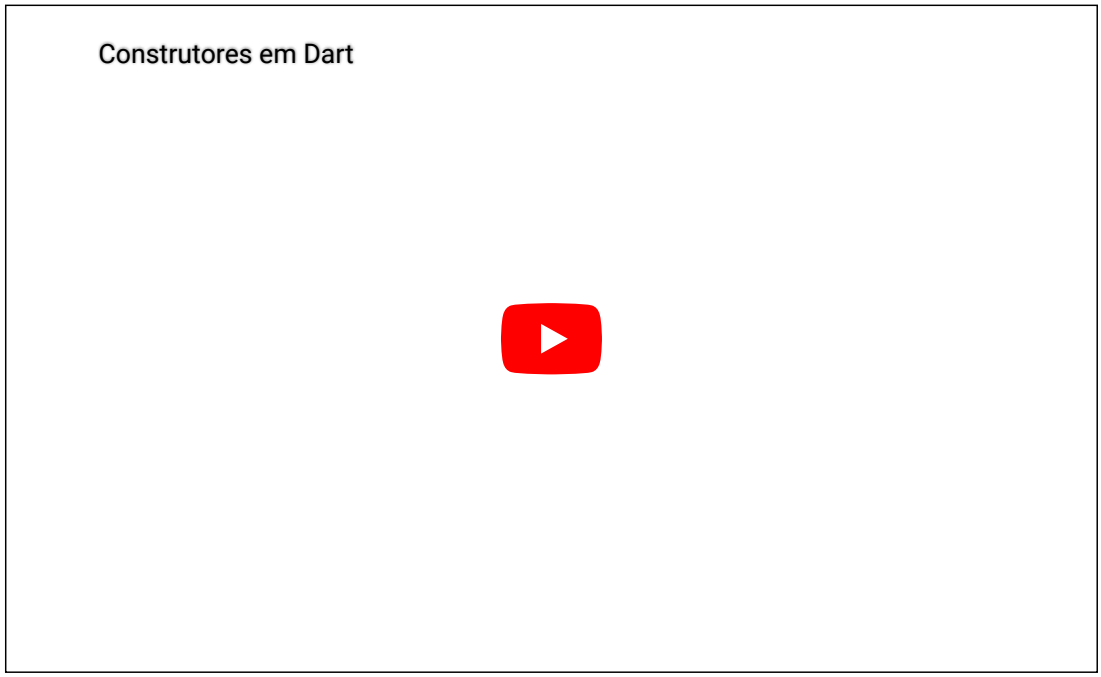
O código abaixo apresenta o código do vídeo anterior:

```
class Pessoa{

  //atributos
  String nome;
  String email;
  int idade;

  //métodos:
  void fazerAniversario(){
    idade++; //idade = idade + 1;
    print('Ôba, festinha!');
  }

  String falarEmail(){
    return 'Meu e-mail é $email';
  }

  void comer(String comida){
    print('Hmmm, adoro comer $comida');
  }

}

void main() {
  Pessoa p1 = Pessoa();
  Pessoa p2 = Pessoa();

  p1.nome = 'Josefina';
  p1.email = 'josefina@josefina.com';
  p1.idade = 18;

  print(p1.nome);
  print(p1.idade);
  print(p1.falarEmail());

  p1.fazerAniverssario();

  print(p1.idade);

  p1.comer('Batatas fritas');

  p2.nome = 'Chapolimn';
  print(p2.idade);
}
```

Construtores

Um recurso muito útil e utilizado nas classes são seus construtores, que permitem que um objeto seja construído através de outro. Sem o construtor devemos o objeto é construído na memória sem que seus parâmetros sejam definidos, ou seja, nulos.

O construtor permite que os valores dos atributos da classe construída sejam passados na hora de se declarar um objeto dessa classe. Muito complicado, não? Para entender melhor este conceito, não deixe de ver o vídeo abaixo.



O código visto abaixo reproduz o código utilizado no vídeo anterior.

```
class Pessoa {
  String nome;
  String email;
  int idade;

  /*
  Pessoa(String nome, String email, int idade){
    this.nome = nome;
    this.email = email;
    this.idade = idade;
  }*/

  Pessoa(this.nome, this.email, this.idade);

  void fazerAniverssario() {
    print('Ôba, festa!');
    idade++; //é o mesmo que idade = idade + 1
  }

  String falarEmail() {
    return "Meu e-mail é $email";
  }

  void comer(String comida) {
    print("Hmmm, adoro comer $comida!");
  }
}

void main() {
  Pessoa p1 = new Pessoa("Josefina", "josefina@uninove.br", 50);
  Pessoa p2 = Pessoa("Chapolim", "chapolim@uninove.br", 80); //o "new" pode ser supr
imido


//   p1.nome = "Josefina";
//   p1.email = "josefina@uninove.br";
//   p1.idade = 50;

//   p2.nome = "Chapolim";
//   p2.email = "chapolim@uninove.br";
//   p2.idade = 80;

  print(p1.idade);
  p1.fazerAniverssario();
  print(p1.idade);

  print("----");

  p2.falarEmail(); //não faz nada pois retornar-se uma String
  print(p2.falarEmail());
  p2.comer("Batatas fritas");
}
```

Encapsulamento

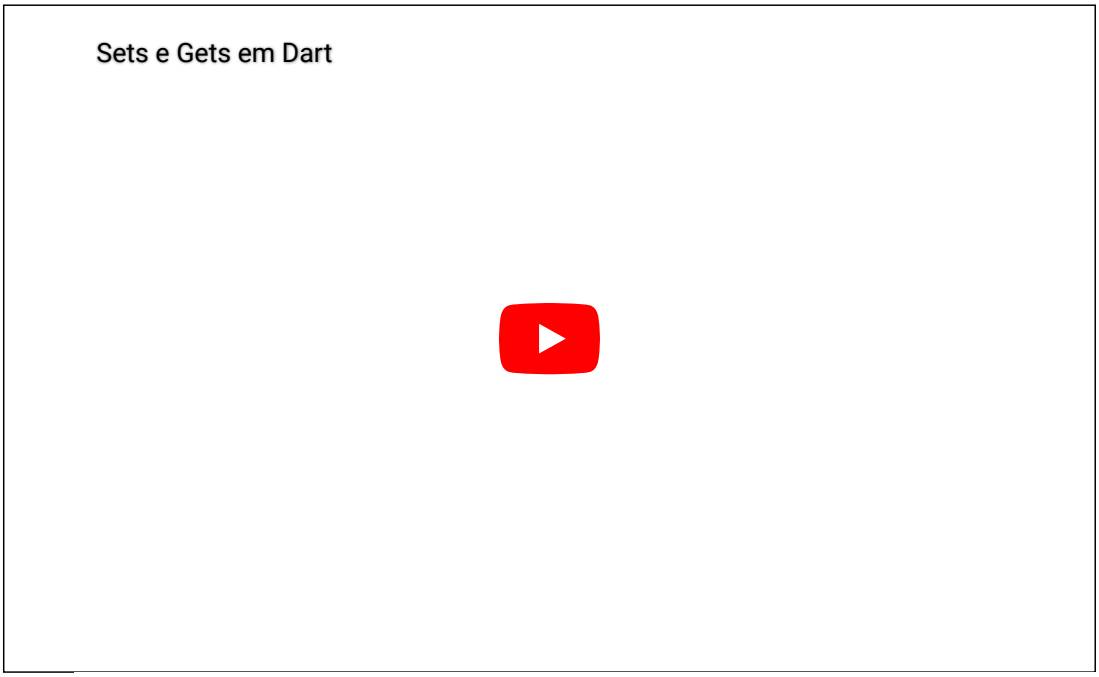
O encapsulamento é uma característica da orientação a objetos que permite com que os mecanismos internos dos métodos de uma classe sejam ocultados para as demais. Neste caso não é preciso se preocupar como cada método implementa suas funções, pois isso passa a ser responsabilidade deles mesmos, apenas usamos o precisamos deles.

Em outras palavras podemos dizer que o encapsulamento é usado para evitarmos que cada atributo das classes que criamos sejam acessados e modificados de forma indevida, ou seja, inclui uma camada de abstração que

são justamente os métodos responsáveis por fazer estes acessos, chamados métodos modificadores de acesso, que são os famosos "sets" e "gets".

Vamos pensar no exemplo da classe "Pessoa", como no código anterior, mais especificamente no atributo da idade. Não faria sentido qualquer outro objeto que utiliza a classe "Pessoa" reduzir a idade de alguém, ou zerar. A idade, neste caso, só pode ser acessada pelo própria ?Pessoa?.

Para melhor entendimento de como aplicar o encapsulamento no Dart, veja o vídeo abaixo.



O código abaixo expressa o exemplo criado no vídeo:

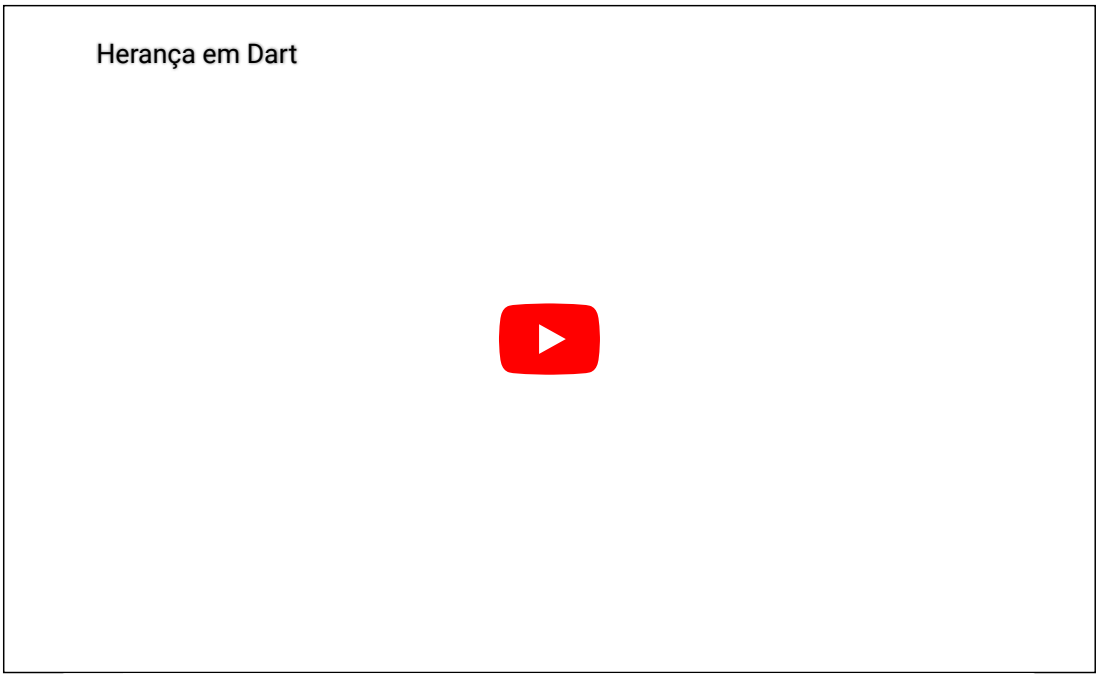
```
class Pessoa {
  String nome;
  String _email;
  int _idade;

  Pessoa(this.nome, this._email, this._idade);

  int get idade => _idade;

  String get email => _email;

  set idade(int idade){
    if(idade > 0 && idade < 150){
      _idade = idade;
    }
  }

  void fazerAniverssario() {
    print('Ôba, festa!');
    _idade++; //é o mesmo que idade = idade + 1
  }

  String falarEmail() {
    return "Meu e-mail é $_email";
  }

  void comer(String comida) {
    print("Hmmm, adoro comer $comida!");
  }
}

void main() {
  Pessoa p1 = new Pessoa("Josefina", "josefina@uninove.br", 50);
  Pessoa p2 = Pessoa("Chapolim", "chapolim@uninove.br", 80); //o "new" pode ser suprimido

//   p1.nome = "Josefina";
//   p1.email = "josefina@uninove.br";
//   p1.idade = 50;

//   p2.nome = "Chapolim";
//   p2.email = "chapolim@uninove.br";
//   p2.idade = 80;

  p1.idade = 200;
  print(p1.idade);
  p1.fazerAniverssario();
  print(p1.idade);
  print(p1.email);

  print("----");

  p2.falarEmail(); //não faz nada pois retornar-se uma String
  print(p2.falarEmail());
  p2.comer("Batatas fritas");
}
```

Herança

No mundo orientado a objetos o conceito de herança é muito útil pois evita a reescrita e duplicidade de códigos. No mundo real o que é uma herança? É recebe todos os bens de outra pessoa. No mundo orientado a objetos o conceito é parecido: Um objeto que herda de outro, recebe todos os atributos e métodos dele e possui suas próprias características.

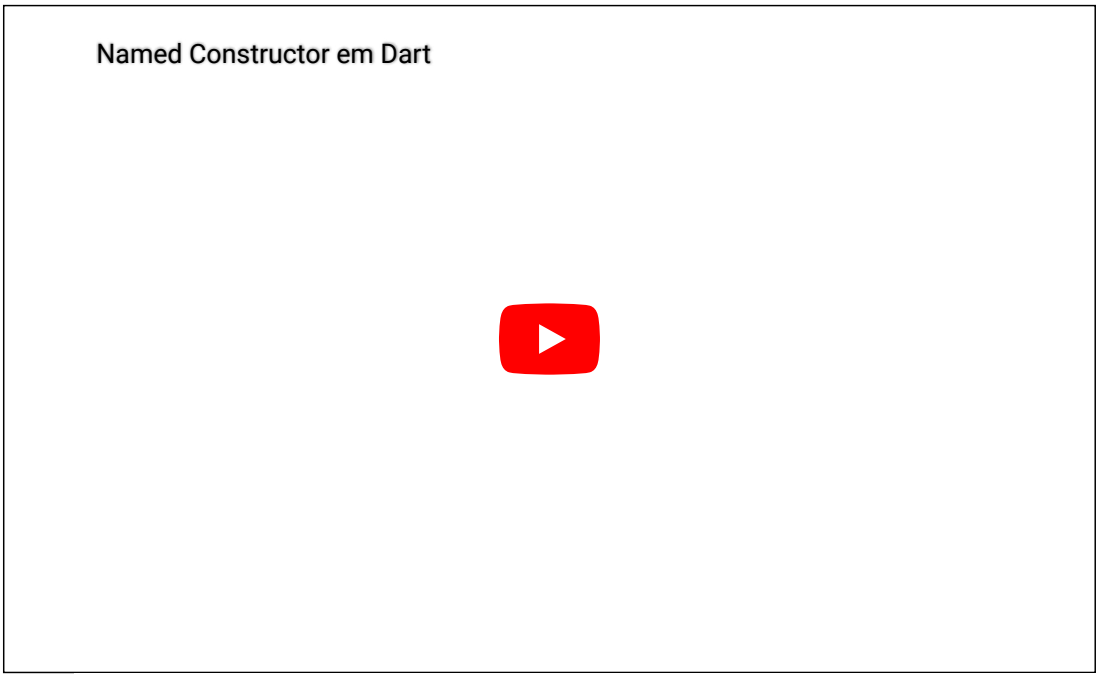
A ideia é que você construa uma classe e para aproveitar tudo que ela tem em outra classe e, para isso, você pode aplicar a herança.

O vídeo abaixo mostra como funcionam os construtores em Dart:




```
class Aluno extends Pessoa {
  int ra;
  Aluno(int this.ra, String nome, String email) : super(nome, email);
}

class Professor extends Pessoa {
  int matricula;
  Professor(this.matricula, String nome, String email) : super(nome, email);
}

class Pessoa{
  String nome;
  String email;

  Pessoa(this.nome, this.email);
}

void main(){
  Aluno a1 = Aluno(123, "Fulaninho", "fulainho@uni9.edu.br");
  a1.nome = "Fulaninho";
  a1.email = "fulano@uni9.edu.br";
  a1.ra = 1234;

  Professor p = Professor(0101, "Thiago", "thiago.traue@uni9.pro.br");
 // p.nome = "Thiago";
  //p.email = "thiago.traue@uni9.pro.br";
 // p.matricula = 011011;
}
```

Outro recurso muito útil e utilizado no Flutter é o 'named constructor' ou construtor nomeado que é uma forma diferente de acionar o construtor através de uma chamada nomeada, ou seja, um método interno é executado na classe construída sempre que este construtor especial é chamado. Para entender como isso funciona no Dart, veja o vídeo abaixo:



E, claro, o código do vídeo abaixo:

```
class Pessoa {
  String nome;
  String email;

  Pessoa(this.nome, this.email);
}

class Aluno extends Pessoa {
  int ra;

  Aluno(int this.ra, String nome, String email) : super(nome, email);

  Aluno.matricular(String nome, String email) : super(nome, email){
    print('Seja muito bem-vindo(a) ao curso de Dart, $nome');
  }
}

class Professor extends Pessoa {
  int matricula;

  Professor(int matricula, String nome, String email) : super(nome, email);

  Professor.contratar(String nome, String email) : super(nome, email){
    print('Bem-vindo(a) Professor(a) $nome');
  }
}

void main() {
  Aluno a1 = Aluno(123, "Josefina", "josefina@uni9.edu.br");
  Professor p1 = Professor(321, "Thiago", "thiago@uni9.pro.br");

  Aluno a2 = Aluno.matricular("João", "jhony@uninove.br");
  a2.ra = 456;

  Professor p2 = Professor.contratar("Thiago", "traue@uninove.br");
}
```

Bastante código até agora não?! Sim, mas esperamos sinceramente que você esteja bastante animado(a) para continuar com o curso e começar a desenvolver aplicações mobile híbridas com códigos muito bonitos, graças ao Flutter.

Quiz

Exercício Final

Orientação a Objetos com DART

INICIAR

Referências

FELIX, Rafael. **Programação orientada a objetos.** São Paulo: Pearson, Ed. Pearson, 2017. *E-book.* Disponível em: https://plataforma.bvirtual.com.br/Leitor/Loader/128217/pdf. Acesso em 19 nov. 2020.

SCHWARZMÜLLER, Maximilian. **Learn Flutter and Dart to Build iOS and Android Apps 2020.** Oreilly, Packt Publishing, 2020. *Vídeo*. Disponível em: https://learning.oreilly.com/videos/learn-flutter-and/9781789951998/. Acesso em 19 nov. 2020.

BRACH, Gilard, LARS, Bak. Dart: a new programming language for structured web programming. **GOTO Conference**, 10 out. 2011. Disponível em: https://gotocon.com/dl/goto-aarhus-2011/slides/GiladBracha_and_LarsBak_OpeningKeynoteDartANewProgrammingLanguageForStructuredWebProgramming.pdf. Acesso em: 12 nov. 2020.

DART. **Dart documentation**. *Site*. Disponível em: https://dart.dev/. Acesso em: 12 nov. 2020.

OKEDIRAN, O. O. *et al*. Mobile operating systems and application development platforms: a survey. **Int. J. Advanced Networking and Applications**. v.6, n.1, p. 2195-2201, july-aug. 2014. Disponível em: https://www.ijana.in/download%206-1-9.php?file=V6I1-9.pdf. Acesso em: 12 nov. 2020.

WINDMILL, Eric. **Flutter in action.** Nova Iorque: Manning publications, 2020. *E-book*. Disponível em: https://learning.oreilly.com/library/view/flutter-in-action/9781617296147/, Acesso em: 12 nov. 2020.

SINHA, Sanjib . **Quick start guide to Dart programming:** create high performance applications for the web and mobile. Lompoc, CA, EUA: Apress, 2019. *E-book*. Disponível em: https://learning.oreilly.com/library/view/quick-start-guide/9781484255629/. Acesso em: 12 nov. 2020.

Avalie este tópico

ANTERIOR
A linguagem DART, declaração de variáveis, desvios condicionais e funções

Índice

PRÓXIMO
Coleções com Mapas e Listas em DART

Biblioteca (https://www.uninove.br/conheca-a-uninove/biblioteca/sobre-a-biblioteca/apresentacao/)
Portal Uninove (http://www.uninove.br)
Mapa do Site

Ajuda? (https://ava.uninove.br/seu/AVA/topico/topico.php?idCurso=)