SI700 – Programação para Dispositivos Móveis

Aula 2 – Orientação a Objetos na Linguagem Dart

Prof. Ulisses Martins Dias

2023

Faculdade de Tecnologia – Unicamp

Roteiro

Tratamento de Execeções

Orientação a Objetos

Construtores

Herança

Getters e Setters

Classes Abstratas e Interfaces

Contexto Estático

Tratamento de Execeções

Tratamento de Execeções

- Durante a execução do programa, erros podem fazer o nosso aplicativo abortar. É função do programador tratar os erros para não causarem prejuízos.
- As palavras reservadas try, on, catch e finally são usadas para esta finalidade.
- O bloco try deve ser usado para delimitar onde o erro poderá ocorrer.
- O bloco on especifica um tipo de erro.

```
int x = 12;
int y = 0;
try {
   int res = x ~/ y;
   print("Resultado: $res");
} on IntegerDivisionByZeroException{
   print('Divisão por zero');
}
// Divisão por zero
```

- O bloco catch é usado quando uma instância de uma classe de exceção é necessária.

```
int x = 12;
int y = 0;
try {
   int res = x ~/ y;
   print("Resultado: $res");
} catch(e) {
   print(e);
}
// IntegerDivisionByZeroException
```

- A união de on e catch também é possível.

```
int x = 12;
int y = 0;
try {
   int res = x ~/ y;
   print("Resultado: $res");
} on IntegerDivisionByZeroException catch(e) {
   print(e);
}
// IntegerDivisionByZeroException
```

- O bloco finally contém código que deve ser usado independente da ocorrência de uma exceção, após try/on/catch.

```
int x = 12;  int y = 0;
try {
   int res = x ~/ y;
   print("Resultado: $res");
} on IntegerDivisionByZeroException {
   print('Divisão por zero');
} finally {
   print('Bloco finally invocado');
}
/*  Divisão por zero
    Bloco finally invocado  */
```

Tratamento de Execeções

- Em várias situações, um ponto do código encontra uma situação anormal, mas não pode tratar a situação, então lançará uma exceção para que seja possível tratar o problema em outro local.
- O comando throw serve para isso, para iniciar uma exceção que se não for tratada ocasionará a saída abrubta do programa.

Tratamento de Execeções

```
main() {
  try {
    setIdade(-2);
  } catch(e) {
    print('Error: ${e.getMessage()}');
} }
void setIdade(int age) {
   if(age<0) {
      throw new AgeException();
   }
}
class AgeException implements Exception{
  String errorMessage() => 'Idade negativa';
}
```

Orientação a Objetos

- Em Dart, as classes são criadas com a palavra reservada class.

```
class Professor { }
```

- As classes podem ter atributos. Se você não instanciar os atributos, eles terão o valor null.

```
class Professor {
  String nome;
  int idade;
}
```

- Atributos podem ser estáticos, o que significa que você pode utilizá-los sem instanciar objetos.

```
class Professor {
  static String vinculo = "Unicamp";
}
```

- Classes podem definir seu próprio comportamento por meio de métodos. Esses métodos também podem ser estáticos.

```
class Professor {
  String nome;
  String sobrenome;

  String nomeProfessor(){
    return "$nome $sobrenome";
  }

  static String getVinculo(){
    return "Unicamp";
  }
}
```

Construtores

Construtores

- Construtores geram instâncias. Caso você não defina um, Dart assume que existe um construtor que não recebe parâmetros.
- Existem vários tipos de construtores, você pode definir um construtor com o mesmo nome da classe e sem nenhum parâmetro, isso será chamado de construtor default.

Construtores

```
main() {
  Professor p = new Professor();
  print(p.nome); // Ulisses
}
class Professor {
  String nome;
  Professor(){
    this.nome = "Ulisses";
  }
}
```

Construtores

- Também é possível definir um construtor com parâmetros, para que seja possível instanciar um objeto passando alguns valores de inicialização.
- Em ambos os casos, o construtor sempre tem o mesmo nome da classe e não possui identificador de tipo de retorno.

Construtores

```
class Professor {
  String nome;
  String sobrenome;

  Professor(String n, String sn) {
    nome = n;
    sobrenome = sn;
  }
  String displayName() {
    return "$nome $sobrenome";
  }
}
```

- Um construtor que apenas atribui valores a atributos é um padrão muito comum. Por isso, foi criado um atalho. A palavra reservada this referencia a instância atual.

```
class Professor {
 String nome;
 String sobrenome;
 Professor(this.nome,this.sobrenome);
}
```

Construtores

- Em Dart, pode existir apenas um construtor default ou um construtor com parâmetros.
- No caso de serem necessárias outras formas de instanciar objetos, é preciso criar construtores nomeados.

```
Professor p = new Professor.meuProprioConstrutor();
print(p.nome); // Ulisses

class Professor {
    String nome;
    Professor(this.nome);
    Professor.meuProprioConstrutor(){
    nome = "Ulisses";
   }
}
```

Herança

Herança

- Herança é feita com a palavra reservada extends.
- A palavra reservada super referencia a classe mãe. No início do construtor, deve haver uma chamada para um construtor da classe mãe. A mesma será feita de forma implícita para o construtor default caso o usuário não a declare.
- No caso de a classe mãe não ter um construtor default, então a classe filha deverá invocar super com os parâmetros necessários.

Herança

```
class Pessoa {
  String nome;
  String sobrenome;
  Pessoa(){
    print("Nova pessoa");
  }
  String displayName() {
    return "Dr. $nome $sobrenome";
  }
}
```

Herança

```
class Professor extends Pessoa {
  Professor(String n, String sn){
    print("Novo professor");
    this.nome = n;
    this.sobrenome = sn;
  }
}

class Subst extends Professor {
  Subst() : super("Alan","Tal"){
    print("Novo Substituto");
  }
  Subst.myConst(String n, String sn): super(n, sn);
}
```

Herança

```
main(){
  Professor p = new Professor(
    "Ulisses", "Dias"
  );
/* Nova pessoa
    Novo professor  */
  Subst s = new Subst();
/*  Nova pessoa
    Novo professor
    Novo Substituto    */

  print(p.displayName());  // Dr. Ulisses Dias
  print(s.displayName());  // Dr. Alan Tal
}
```

- A mesma regra de chamada de um construtor da classe mãe vale para construtores com nome. Nesse caso, uma chamada super foi feita em Subst.myConst.
- Uma classe filha não pode acessar o construtor da classe mãe com uma chamada a super() no corpo das funções, por isso colocamos após o ":" na sintaxe vista nos códigos.
- Você pode usar o construtor com nome da classe mãe ao invés do default ou do parametrizado. Neste caso, apenas o construtor com nome da classe mãe será invocado na instanciação da classe filha.

Herança

```
class Pessoa {
  String nome;
  String sobrenome;
  Pessoa.build(this.nome,
               this.sobrenome);
  String displayName() {
    return "$nome $sobrenome";
  }
}
class Professor extends Pessoa {
  Professor(n, sn):super.build(n,sn);
  String displayName() => "Dr. $nome $sobrenome";
}
```

Getters e Setters

Getters e Setters

- Em Dart, todos os membros de uma classe são públicos a não ser que comecem com um *underscore*, o que os tornam privados dentro do arquivo `.dart`.
- Métodos **getters** são criados automaticamente para todos os atributos públicos e métodos **setters** são criados para todos os atributos públicos não marcados como final.
- No caso dos membros privados, você deverá criar um getter e um setter para serem acessíveis de fora do arquivo.
- As palavras reservadas set e get servem para criarmos **getters** e **setters**.

Getters e Setters

```
class Professor {
 String _nome;
 String _sobre;
 Professor(this._nome,this._sobre);

 String get nome => "$_nome $_sobre";
 set nome(n) {
  List strings = n.split(" ");
  _nome =strings[0];
  _sobre=strings.sublist(1).join(" ");
 }
}

main() {
  Professor p = new Professor("Ulisses",
```

Getters e Setters

```
main() {
  Professor p = new Professor("Ulisses",
                              "Dias");
  print(p.nome); // Ulisses Dias
  p.nome = "Danielle Dias";
  print(p.nome); // Danielle Dias
}
```

Classes Abstratas e Interfaces

Classes Abstratas

- Classes abstratas não serão instanciadas.
- Métodos sem corpo são abstratos.

Classes Abstratas

```
//  Classe abstrata, não pode ser instanciada.
abstract class Professor {
  // Método abstrato, precisa ser sobrescrito na classe filha
  verificaID();

  // Método concreto, não precisa ser sobrescritno na classe filha
  verificaNome(){
    print("Ulisses Dias");
  }
}
```

Interfaces

- Em Dart, não temos uma palavra reservada para declarar interfaces como acontece com Java. Nesse caso, qualquer classe possui uma interface própria que pode ser implementada por outras classes.
- A palavra reservada implements serve para dizer que uma classe irá implementar a interface própria de outra classe.
- Nesse caso, nenhum dos métodos serão herdados, mas todos deverão ser reescritos. Uma consequência disso é que você não poderá invocar os métodos da superclasse por meio de super.
- Uma classe pode implementar mais de uma superclasse.

```
class Pessoa {
  void nasce(){
  }
  void cresce(){
  }
  void morre(){
  }
}

class Funcionario{
  void trabalha(){
  }
}
```

```
class Professor implements Pessoa, Funcionario {

  // A implementação dos métodos  a seguir é mandatória.
  void nasce(){
  }
  void cresce(){
  }
  void morre(){
  }
  void trabalha(){
  }
}
```

```
int main(){
  Pessoa p = new Professor();
  Funcionario f = new Professor();
}
```

Contexto Estático

Contexto Estático

- A palavra reservada static serve para gerar contexto estático e pode ser aplicada tanto a métodos quanto a atributos.
- Membros estáticos são armazenados em memória apenas uma vez, independente do número de instâncias da classe.
- Membros estáticos só podem ser acessados usando o nome da classe.

Contexto Estático

- Instâncias de uma classe não podem acessar os métodos estáticos diretamente, o que difere do que ocorre com Java. Isso criar uma separação entre o contexto da instância e o contexto da classe.
- Atributos estáticos podem ser mudados inadvertidamente em vários lugares do código, por isso são normalmente declarados como const. Isso evita que sejam usados como variáveis globais, mas como constantes globais.
- De dentro de um método estático, você não poderá acessar os membros de instância da mesma classe, apenas os membros estáticos.

Contexto Estático

```
class Professor  {
  // Atributo estático
  static double pi = 3.14;

  // Atributo de instância
  double altura = 1.74;

  // Método estático
  static void nasce(){
    print(pi);
    // Não podemos acessar altura
    // print(altura);
  }
```

```
  // Método de instância
  void cresce(){
    /* Podemos acessar as variáveis
    estáticas e de instâncias.  */
    print(pi);
    print(Professor.pi);
    print(altura);
  }
}
```

Contexto Estático

```
int main(){
  // Podemos acessar membros estáticos sem precisar de instâncias
  print(Professor.pi);

  // Instanciando professor
  Professor p = new Professor();

  // Não podemos acessar membros estáticos com as instâncias
  //print(p.pi);

  //As chamadas a seguir são possíveis
  Professor.nasce();
  p.cresce();
}
```