1 Introdução

Neste material, prosseguimos com os estudos sobre as principais características da linguagem Dart.

2 Desenvolvimento

2.1 Estruturas de repetição As estruturas de repetição de Dart são as seguintes.

- **for**
- **for each**
- **while**
- **do/while**

Veja um exemplo de for.

```
void main() {
  for (int i = 0; i < 10; i++) {
    print(i);
  }
}
```

No exemplo a seguir, iteramos sobre uma coleção (veremos mais sobre coleções em breve) utilizando um for each.

```
void main() {
  const nomes = ['Pedro', 'Maria', 'João'];
  //em vez de final, podemos usar var e o tipo
  //explícito da varíavel (String nesse caso)
  //mas não podemos usar const
  for (final nome in nomes) {
    print(nome);
  }
}
```

Como em outras linguagens de programação, as três regiões do for são opcionais. No exemplo a seguir, dado que não há condição de continuidade, temos um loop infinito.

Professor Rodrigo Bossini <professorbossini@gmail.com>
Produção: agosto de 2023

```
void main() {
  //loop infinito com for
  for ( ; ; ) {
    print('loop infinito');
  }
}
```

O while é semelhante àquele que conhecemos de outras linguagens.

```
void main() {
  //loop com while
  int contador = 0;
  while (contador <= 10) {
    print("Contador: $contador");
    contador++;
  }
}
```

Assim como o do/while.

```
void main() {
  //loop com do/while
  int i = 0;
  do {
    print("O valor de i é $i");
    i++;
  } while (i < 10);
}
```

2.2 enum Muitas linguagens de programação oferecem um recurso conhecido como enum Ele permite que especifiquemos uma lista de constantes, promovendo a legibilidade do programa. Por exemplo, ao invés de representarmos os dias da semana como números de 1 a 7, podemos criar um enum, que é uma lista com os nomes dos dias explícitos.

```dart
enum DiaSemana {SEGUNDA, TERCA, QUARTA, QUINTA, SEXTA, SABADO,
DOMINGO}
void main(){
  //exibindo a lista inteira
  print(DiaSemana.values);
  DiaSemana dia = DiaSemana.SEGUNDA;
  //exibindo o nome desse valor
  print(dia.name);
  switch (dia){
    case DiaSemana.SEGUNDA:
      print("Nããããããããooooooooooooo!!!!!!");
    case DiaSemana.TERCA:
      print(":(");
    case DiaSemana.QUARTA:
      print(":|");
    case DiaSemana.QUINTA:
      print(":)");
    case DiaSemana.SEXTA:
      print("Eeeeeeeeeeeeeeeeeeee!!!!!!");
    case DiaSemana.SABADO:
      print("=DDDDDD");
    case DiaSemana.DOMINGO:
      print(" :):");
  }
}
```

2.3 Instalação do SDK, VS Code, pasta e teste inicial Passaremos a utilizar uma instalação local de Dart e editaremos código utilizando o VS Code. O SDK de Dart pode ser obtido a partir do seguinte link.

https://dart.dev/get-dart

Se tiver planos de utilizar o framework Flutter, pode ser boa ideia fazer a sua instalação. Inclusive, ela já inclui o SDK de Dart, já que Flutter é escrito em Dart. Por isso, **se optar por instalar o Flutter, não precisará instalar o Dart à parte. Flutter pode ser obtido em**

https://docs.flutter.dev/get-started/install

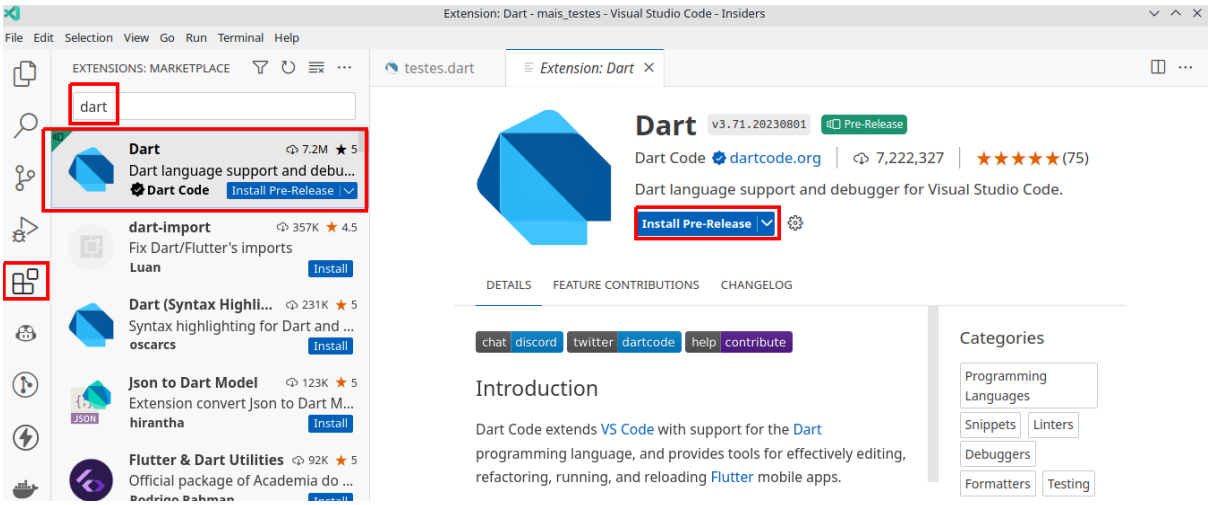
A seguir, crie uma pasta para abrir os seus projetos. Se estiver usando o Windows, pode ser interessante usar algo como

C:\Users\seuUsuario\Documents\projetos_dart\mais_testes

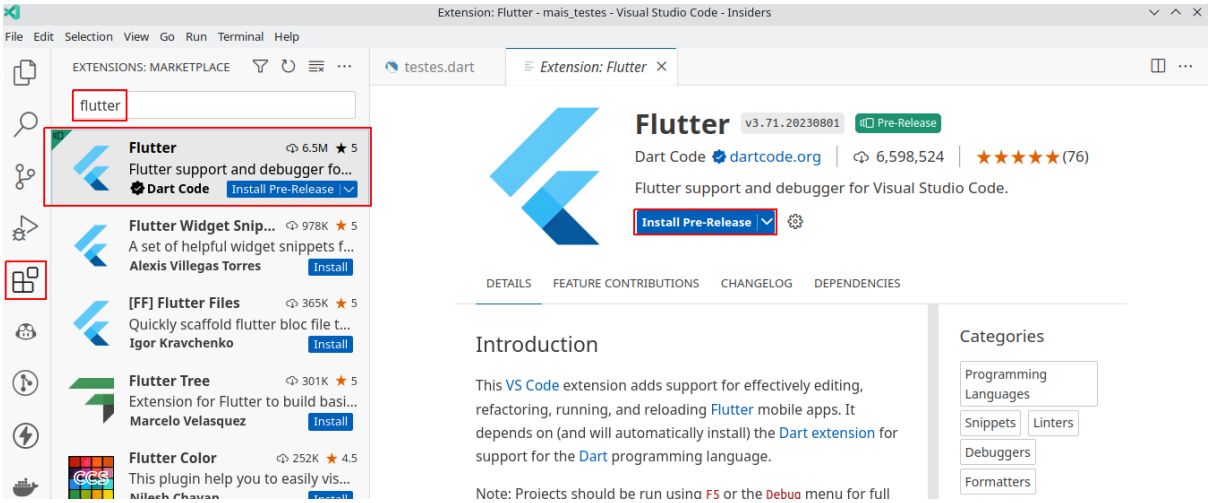
A seguir, abra o VS Code e clique File >> Open Folder. Navegue até a pasta e vincule o VS Code a ela. No VS Code, crie um arquivo chamado **testes.dart**. Claro, o nome pode ser qualquer um.

É recomendável fazer a instalação da extensão Dart para o VS Code. Para tal, abra a seção Extensions dele e busque por Dart.

> **Nota**. É natural que seu VS Code mostre o texto **Install** em vez de **Install Pre-Release**, a menos que também esteja utilizando a versão insiders do VS Code.



Caso vá desenvolver com Flutter, também é interessante fazer a instalação da extensão própria para esse arcabouço. Observe que se optar por fazê-lo, estará instalando também a extensão Dart, pois ela é uma dependência da extensão Flutter.



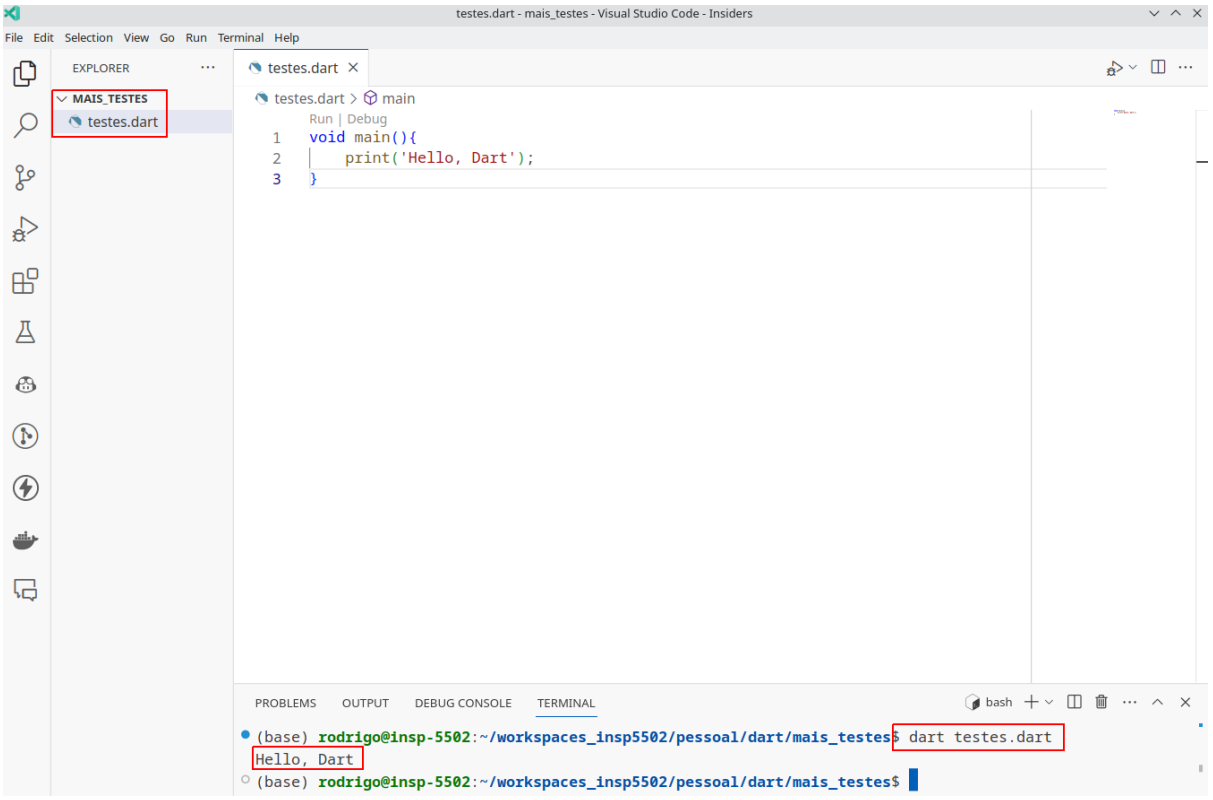
Abra o arquivo testes.dart no VS Code e use o seguinte código para testar o seu ambiente.

```
void main(){
print('Hello, Dart');
}
```

Clique Terminal >> New Terminal para abrir um terminal interno do VS Code. Use

dart testes.dart

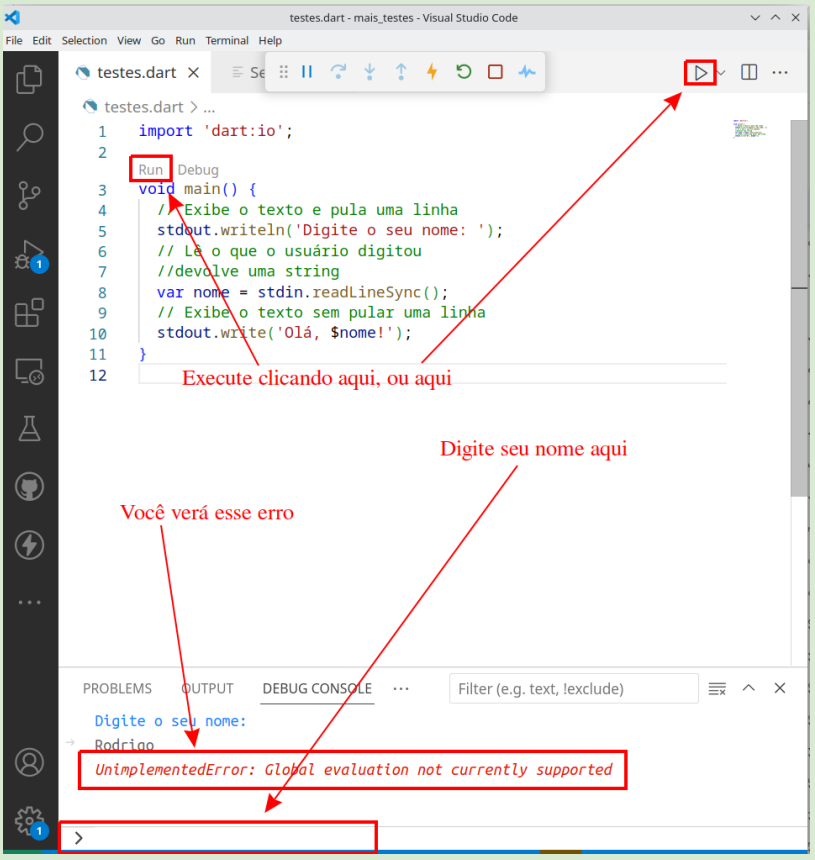
para fazer o teste. Veja o resultado esperado.



2.4 Entrada e saída de dados Os objetos stdin e stdout representam, respectivamente, a entrada e a saída padrão. Eles fazem parte do pacote dart:io. Podemos utilizá-los para interagir com o usuário.

```dart
import 'dart:io';
void main() {
// Exibe o texto e pula uma linha
stdout.writeln('Digite o seu nome: ');
// Lê o que o usuário digitou
//devolve uma string
var nome = stdin.readLineSync();
// Exibe o texto sem pular uma linha
stdout.write('Olá, $nome!');
}
```

Nota. Quando executamos um programa Dart utilizando os atalhos gráficos da extensão, como na figura a seguir, por padrão, a execução é feita utilizando-se o **Debug Console** provido pela extensão.



É importante sabermos que:

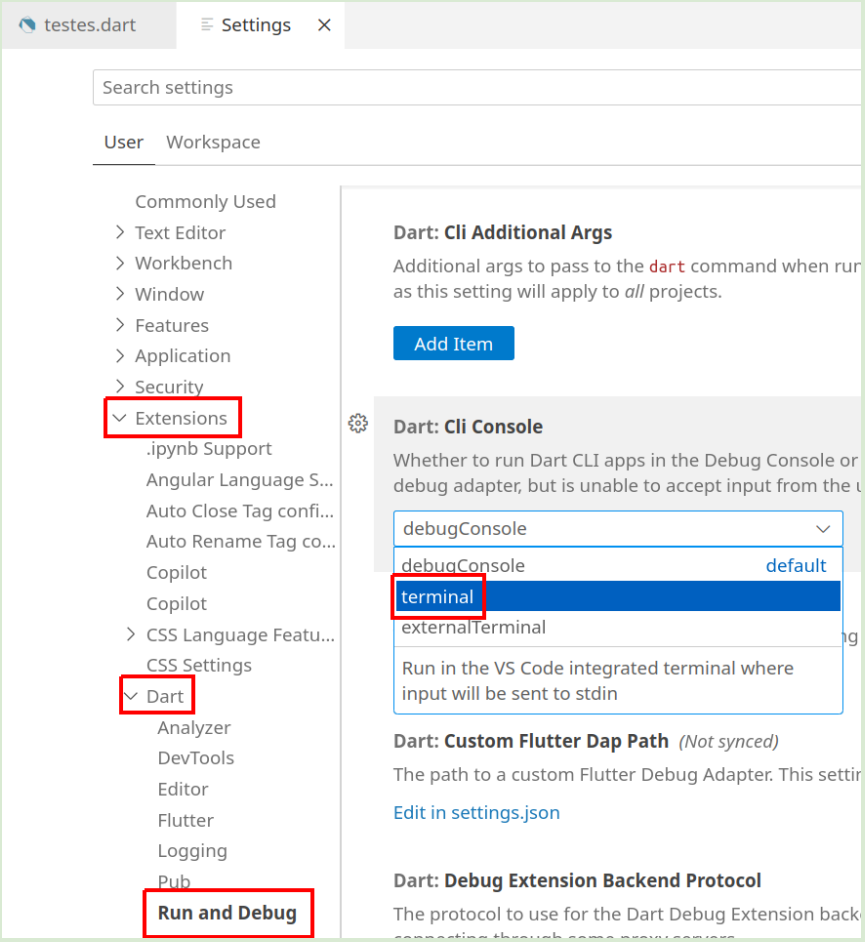
- Como o nome sugere, ele tem suporte a um mecanismo para depuração (execução pausada linha a linha) de código

- Não é possível fazer a entrada de dados usando o pacote **dart:io**.

Se desejar, é possível alterar o ambiente de execução utilizado pela extensão. Podemos deixar de utilizar o debug console e passar a utilizar o terminal padrão. Com isso

- perdemos o suporte a debug oferecido pelo debug console
- podemos capturar dados no terminal usando o pacote **dart:io** ainda que executemos a aplicação usando os atalhos visuais da extensão.

Essa configuração pode ser feita clicando-se em **File >> Preferences >> Settings.** Depois disso, clique em **Extensions >> Dart >> Run and Debug**. Encontre a opção **Cli Console** à direita e troque para **terminal**. Observe.



Se desejar, também é possível não fazer alteração alguma e, quando for necessário capturar dados usando o pacote **dart:io**, executar o programa com **dart arquivo.dart** direto no terminal, em vez de utilizar os atalhos gráficos da extensão.

Observe que o método de captura de dados devolve uma String. Se for necessário lidar com números, é preciso fazer a conversão usando os métodos parse de int, num e/ou double.

```
import 'dart:io';
void main() {
//erro, precisa converter antes
int idade = stdin.readLineSync();
}
```

Pode ser tentador fazer a seguinte conversão. Mas ela também causa erro.

```
import 'dart:io';
void main() {
//erro
//estamos tentando entregar String? para parse
//ele somente recebe String
int idade = int.parse(stdin.readLineSync());
}
```

Isso acontece pois

- O método readLineSync devolve **String?**. É um tipo que representa uma string ou null.

- O método parse recebe **String**. É um tipo que representa uma string necessariamente diferente de null.

Ou seja, é um erro tentar passar null ao método parse. Algo assim: int.parse(null);

E agora?

Você pode fazer validações com estruturas de seleção e/ou repetição, dependendo do que deseja.

```
import 'dart:io';

void main() {
print('Digite sua idade: ');
//String? admite uma string ou null
String? entrada = stdin.readLineSync();
//validação com if else
if (entrada != null) {
int idade = int.parse(entrada);
print ("Você tem $idade anos");
}

//validação com while
while (entrada == null) {
print('Digite sua idade: ');
entrada = stdin.readLineSync();
}
int idade = int.parse(entrada);
print ("Você tem $idade anos");
}
```

Outra opção é usar o operador **!**. Ele é uma promessa de que a expressão sobre a qual opera é diferente de null. Mas observe que estamos nos sujeitando a um erro envolvendo null em tempo de execução.

```
import 'dart:io';
void main() {
print('Digite sua idade: ');
//o operador ! opera sobre a expressão que o antecede
//neste caso, o retorno do método readLineSync()
//estamos prometendo que o valor não será nulo para o
compilador
//ele fala: blz, você se vira com a Dart VM
int idade = int.parse(stdin.readLineSync()!);
print(idade);
}
```

2.5 Controlando o número de casas decimais Podemos controlar o número de casas decimais usando o método **toStringAsFixed**.

```
void main() {
//com um número fixo
print('${154.3434234.toStringAsFixed(2)}');
//com uma variável
var variavel = 2.3134353543;
print('${variavel.toStringAsFixed(2)}');
}
```

2.6 Valores aleatórios A geração de valores aleatórios pode ser feita por métodos da classe Random.

```
import 'dart:math';

void main() {
var gerador = Random();
//inteiro aleatório entre 0 e 5
var n1 = gerador.nextInt(6);
print(n1);

//inteiro aleatório entre 1 e 10
var n2 = gerador.nextInt(10) + 1;
print(n2);

//inteiro aleatório entre 0 e 100
var n3 = gerador.nextInt(101);
print(n3);

//real aleatório entre 0 e 1 ([0, 1))
var n4 = gerador.nextDouble();
print(n4);

//real aleatório entre 0 e 100 ([0, 100))
var n5 = gerador.nextDouble() * 100;
print(n5);

//real aleatório entre 100 e 150
var n6 = gerador.nextDouble() * 50 + 100;
print(n6);

//booleano aleatório
var n7 = gerador.nextBool();
print(n7);
}
```

2.7 Papel, pedra e tesoura Utilizemos os recursos vistos até então, ou grande parte deles, para fazer a implementação de um clássico Papel, pedra e tesoura. Para isso, vamos criar uma nova aplicação. Desta vez, ela será um projeto Dart "completo". Para tal, abra um terminal vinculado à sua pasta de projetos Dart (fora de qualquer projeto específico) e use

dart create pedra_papel_tesoura -t console

para criar a aplicação.

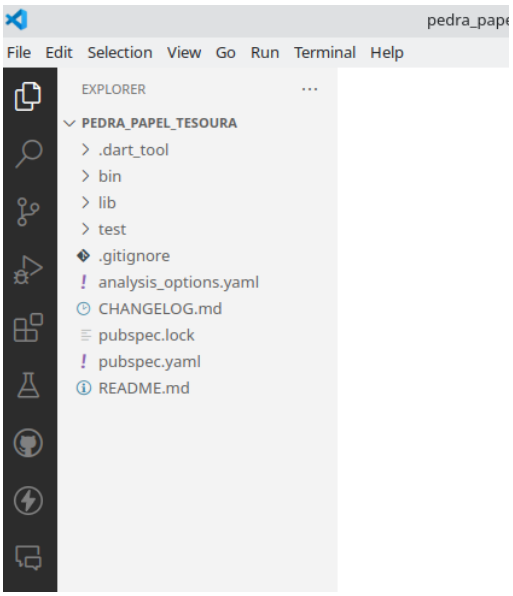
Nota. Use as regras descritas a seguir para escolher o nome de sua aplicação. Basicamente, use **snake_case**.

https://dart.dev/tools/pub/pubspec#name

Nota. O parâmetro **t** significa template ou modelo. Estamos usando o modelo console, ou seja, estamos criando uma aplicação para executar na linha de comando. Há outros modelos. Veja:

https://dart.dev/tools/dart-create

Depois de criar o projeto, abra o VS Code e clique em **File >> Open Folder.** Abra a pasta que acaba de ser criada. Veja o resultado esperado.



Veja uma descrição sucinta de cada item.

.dart_tool: pasta que contém arquivos utilizados por várias ferramentas Dart.
bin: pasta que contém o arquivo .dart principal. A ideia é que definamos a função main ali.
lib: pasta que contém arquivos .dart em geral. É comum fazermos a implementação da aplicação nesta pasta e em subpastas dela.
test: pasta que contém arquivos de teste. Eles podem ser executados com **dart test**.
.gitignore: o clássico arquivo que o Git utiliza para decidir quais arquivos não incluir no repositório
analysis_options.yaml: arquivo de configurações utilizado para, por exemplo, configurar o **linter** do Dart.

Nota. No dicionário, é possível encontrar a seguinte definição para a palavra linter.

"a machine for removing the short fibers from cotton seeds after ginning."

Traduzindo:
"uma máquina para remover as fibras curtas das sementes de algodão após o descaroçamento"

Há também a seguinte definição:

"línter é um subproduto obtido da planta do algodão que tem uma grande importância. Trata-se das fibras bem curtinhas que ficam grudadas no caroço do algodão após a retirada das mais longas. É o que "sobra" no beneficiamento."

https://soudealgodao.com.br/blog/o-que-e-linter-conheca-o-residuo-de-algodao-que-e-pau-pra-toda-obra/

Também é interessante falar da palavra **lint**, que significa algo como "fiapo". lint também é o nome de uma ferramenta criada por Stephen C. Johnson, conhecido por ter trabalhado, por exemplo, no desenvolvimento do yacc (yet another C Compiler).

Resumindo, quando falamos de linter/lint, estamos nos referindo a características indesejáveis (fiapos) no código que desejamos remover.

Veja os fiapos "core" e recomendados de Dart aqui:

https://pub.dev/packages/lints

CHANGELOG.md um arquivo escrito em Markdown para registrar as alterações feitas na aplicação ao longo do tempo.

Nota. A linguagem Markdown é muito utilizada e é muito útil conhecê-la. Veja mais em:

https://www.markdownguide.org/

pubspec.lock: utilizamos para garantir que todos os desenvolvedores de um projeto estão utilizando as mesmas versões de cada dependência. Veja mais aqui:

https://medium.com/codingmountain-blog/what-is-pubspec-lock-and-why-should-flutter-devs-care-86ab4d0b47fc

pubspec.yaml Especificamos as dependências do projeto aqui.

README.md Arquivo em markdown que descreve aspectos do projeto, como ele pode ser executado etc.

Sua vez! Tente fazer o jogo, com a ajuda do seu professor. Seja criativo! Faça validações, use enum, variações de switch/case etc.

Veja uma possível implementação. Observe que usamos alguns recursos a serem estudados mais a fundo em breve.

Abra o arquivo **pedra_papel_tesoura.dart** da pasta **lib**. Apague todo o seu conteúdo e substitua pela seguinte função. Ela representa o nosso jogo.

```
void jogo(){

}
```

A seguir, abra o arquivo **pedra_papel_tesoura.dart** da pasta **bin** e ajuste seu conteúdo como a seguir.

```
//show indica que queremos apenas aquele membro
import 'package:pedra_papel_tesoura/pedra_papel_tesoura.dart' show
jogo;

void main(List<String> arguments) {
  jogo();
}
```

A seguir, no arquivo **pedra_papel_tesoura.dart** da pasta lib, importamos os pacotes

- dart:io para capturar as opções do usuário
-dart:math para gerar valores aleatórios, representando a opção do usuário

```
import 'dart:io';
import 'dart:math';
```

Escrevemos um **enum** para representar as opções do jogo, incluindo a opção de sair.

```
import 'dart:io';
import 'dart:math';

enum OPCAO { pedra, papel, tesoura, sair }
```

Na função jogo, vamos

- escrever um loop que executa até que o usuário digite 4, indicando que deseja sair.

- a cada iteração, usar a função sleep alimentada com uma instância de Duration para que, a cada iteração, aguardemos 3 segundos.

Os comentários na função jogo indicam as funções que precisamos implementar para obter a implementação completa do jogo, aplicando o paradigma da **divisão e conquista**: cada pedacinho vai ser implementado por uma função separada. Observe que haverá um erro de compilação temporário já que a variável que representa a opção do usuário ainda não foi inicializada. Nada para se preocupar. Ela será inicializada na hora certa.

```
import 'dart:io';
import 'dart:math';

enum OPCAO { pedra, papel, tesoura, sair }

void jogo() {
  int opUsuario;
  do{

    //exibir menu

    //capturar opção do usuário, validando

    //se o usuário digitar 4, sair

    //senão

    //sortear escolha do computador

    //mapear opcao usuario, de int para enum


    //mapear opcao computador, de int para enum

    //exibir as opções de cada um

    //decidir quem venceu, ou se houve empate

    //exibir o resultado
    sleep(Duration(seconds: 3));
    }while(opUsuario != 4);
}
```

Embora não seja estritamente necessário, vamos escrever uma função que exibe algo recebido como parâmetro, essencialmente operando da mesma forma que a função print. De fato, ela usa a função print para cumprir a sua missão.

```
import 'dart:io';
import 'dart:math';

enum OPCAO { pedra, papel, tesoura, sair }

void exibe(String texto) {
  print(texto);
}

void jogo() {
  ...
```

A seguir, escrevemos uma função responsável por capturar a opção do usuário.

```
import 'dart:io';
import 'dart:math';

enum OPCAO { pedra, papel, tesoura, sair }

void exibe(String texto) {
  print(texto);
}

int pegaOpcaoUsuario() {
  return int.parse(stdin.readLineSync()!);
}


void jogo() {
  ...
```

A função a seguir recebe uma opção e verifica se ela está no intervalo válido.

```
import 'dart:io';
import 'dart:math';

enum OPCAO { pedra, papel, tesoura, sair }

void exibe(String texto) {
  print(texto);
}

int pegaOpcaoUsuario() {
  return int.parse(stdin.readLineSync()!);
}

bool opcaoEhValida(int opcao) {
  return opcao >= 1 && opcao <= 4;
}


void jogo() {
...
```

Uma vez que tenhamos uma opção em mãos, representada como um número inteiro, precisaremos fazer um mapeamento de int para enum. Lembre-se que o índice do enum começa a partir do zero (pedra = 0, papel = 1 etc). Para tornar intuitivo para o usuário, deixamos ele fazer a sua escolha no intervalo de 1 a 4. Por isso, subtraímos 1 na hora de indexar o enum.

```
import 'dart:io';
import 'dart:math';

enum OPCAO { pedra, papel, tesoura, sair }

void exibe(String texto) {
  print(texto);
}

int pegaOpcaoUsuario() {
  return int.parse(stdin.readLineSync()!);
}

bool opcaoEhValida(int opcao) {
  return opcao >= 1 && opcao <= 4;
}

OPCAO mapeiaOpcao(int opcao) {
  return OPCAO.values[opcao - 1];
}


void jogo() {
...
```

A próxima função verifica se houve empate. Se não houve, ela diz quem venceu.

```
...
OPCAO mapeiaOpcao(int opcao) {
  return OPCAO.values[opcao - 1];
}


String decideResultado(OPCAO opcaoUsuario, OPCAO opcaoComputador) {
if (opcaoUsuario == opcaoComputador) return "Empate";
if (opcaoUsuario == OPCAO.papel && opcaoComputador == OPCAO.pedra ||
opcaoUsuario == OPCAO.pedra && opcaoComputador == OPCAO.tesoura ||
opcaoUsuario == OPCAO.tesoura && opcaoComputador == OPCAO.papel) {
  return "Você venceu";
}
  return "Computador venceu";
}


void jogo() {
...
```

Agora podemos começar a implementar o jogo. Começamos exibindo o menu ao usuário, capturando a sua opção e verificando se ele já deseja sair, por ter digitado o valor 4.

```
...
void jogo() {
  int opUsuario;
  do{
    //exibir menu
    //capturar opção do usuário, validando
    do{
      exibe('1-Pedra\n2-Papel\n3-Tesoura\n4-Sair');
      opUsuario = pegaOpcaoUsuario();
    }while(!opcaoEhValida(opUsuario));
    //se o usuário digitar 4, sair
    //senão
    if(opUsuario != 4){
      //sortear escolha do computador

      //mapear opcao usuario, de int para enum

      //mapear opcao computador, de int para enum

      //exibir as opções de cada um

      //decidir quem venceu, ou se houve empate

      //exibir o resultado

      sleep(Duration(seconds: 3));
    }
  }while(opUsuario != 4);
  exibe('Até logo');
}
```

Verifique se está tudo ok executando o programa com

dart run

A escolha do computador pode ser simulada pelo método nextInt da classe Random.

```
void jogo() {
  int opUsuario;
  do{
    //exibir menu
    //capturar opção do usuário, validando
    do{
      exibe('1-Pedra\n2-Papel\n3-Tesoura\n4-Sair');
      opUsuario = pegaOpcaoUsuario();
    }while(!opcaoEhValida(opUsuario));
    //se o usuário digitar 4, sair
    //senão
    if(opUsuario != 4){
      //sortear escolha do computador
      int opComputador = Random().nextInt(3) + 1;
      //mapear opcao usuario, de int para enum

      //mapear opcao computador, de int para enum

      //exibir as opções de cada um

      //decidir quem venceu, ou se houve empate

      //exibir o resultado

      sleep(Duration(seconds: 3));
    }
  }while(opUsuario != 4);
  exibe('Até logo');
}
```

O próximo passo é traduzir as opções do usuário e do computador, fazendo uso do enum. Podemos também exibí-las.

```
void jogo() {
  int opUsuario;
  do{
    //exibir menu
    //capturar opção do usuário, validando
    do{
      exibe('1-Pedra\n2-Papel\n3-Tesoura\n4-Sair');
      opUsuario = pegaOpcaoUsuario();
    }while(!opcaoEhValida(opUsuario));
    //se o usuário digitar 4, sair
    //senão
    if(opUsuario != 4){
      //sortear escolha do computador
      int opComputador = Random().nextInt(3) + 1;
      //mapear opcao usuario, de int para enum
      OPCAO opcaoUsuario = mapeiaOpcao(opUsuario);
      //mapear opcao computador, de int para enum
      OPCAO opcaoComputador = mapeiaOpcao(opComputador);
      //exibir as opções de cada um
      exibe(
        'Você (${opcaoUsuario.name}) vs (${opcaoComputador.name})
Computador'
      );
      //decidir quem venceu, ou se houve empate
      //exibir o resultado
      sleep(Duration(seconds: 3));
    }
  }while(opUsuario != 4);
  exibe('Até logo');
}
```

Execute novamente com

dart run

A seguir, resta

- decidir o vencedor (ou se houve empate)

- exibir o resultado

- exibir ********** para separar uma rodada da outra

```
void jogo() {
  int opUsuario;
  do{
    //exibir menu
    //capturar opção do usuário, validando
      do{
        exibe('1-Pedra\n2-Papel\n3-Tesoura\n4-Sair');
        opUsuario = pegaOpcaoUsuario();
      }while(!opcaoEhValida(opUsuario));
      //se o usuário digitar 4, sair
      //senão
      if(opUsuario != 4){
        //sortear escolha do computador
        int opComputador = Random().nextInt(3) + 1;
        //mapear opcao usuario, de int para enum
        OPCAO opcaoUsuario = mapeiaOpcao(opUsuario);
        //mapear opcao computador, de int para enum
        OPCAO opcaoComputador = mapeiaOpcao(opComputador);
        //exibir as opções de cada um
        exibe(
        'Você (${opcaoUsuario.name}) vs (${opcaoComputador.name})
Computador'
        );
        //decidir quem venceu, ou se houve empate
        String vencedor =
            decideResultado(opcaoUsuario, opcaoComputador);
        //exibir o resultado
        exibe(vencedor);
        exibe('**********************************');
        sleep(Duration(seconds: 3));
      }
  }while(opUsuario != 4);
  exibe('Até logo');
}
```

Faça novos testes com

dart run

Referências

Dart programming language | Dart. Google, 2023. Disponível em <https://dart.dev/>. Acesso em agosto de 2023.