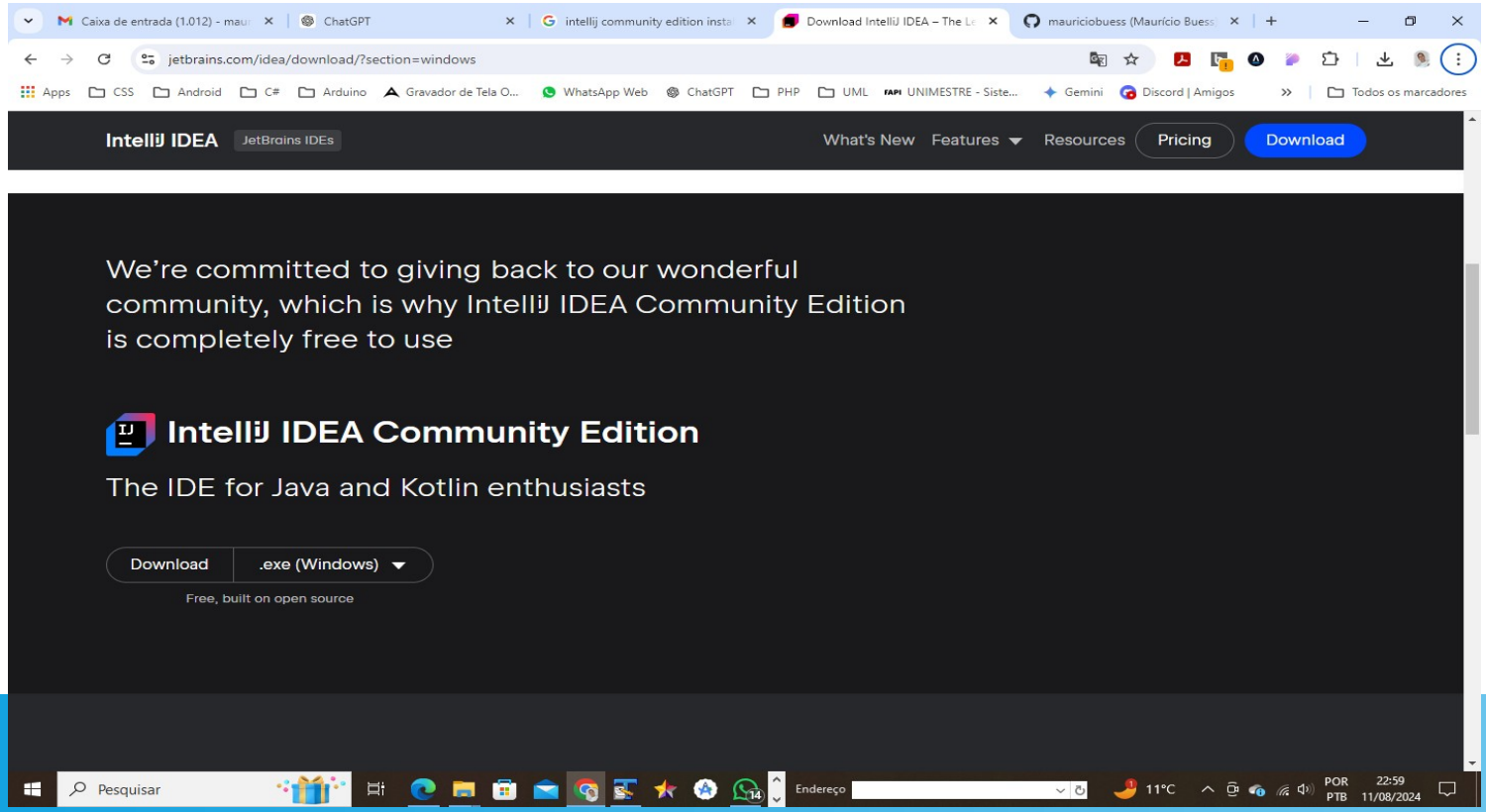
Desenvolvimento de Aplicativos Móveis

Professor Maurício Buess
mbuess@up.edu.br
https://github.com/mauriciobuess

Intellij Idea Community Edition – Jet Brains

- Acesse: https://www.jetbrains.com/idea/download



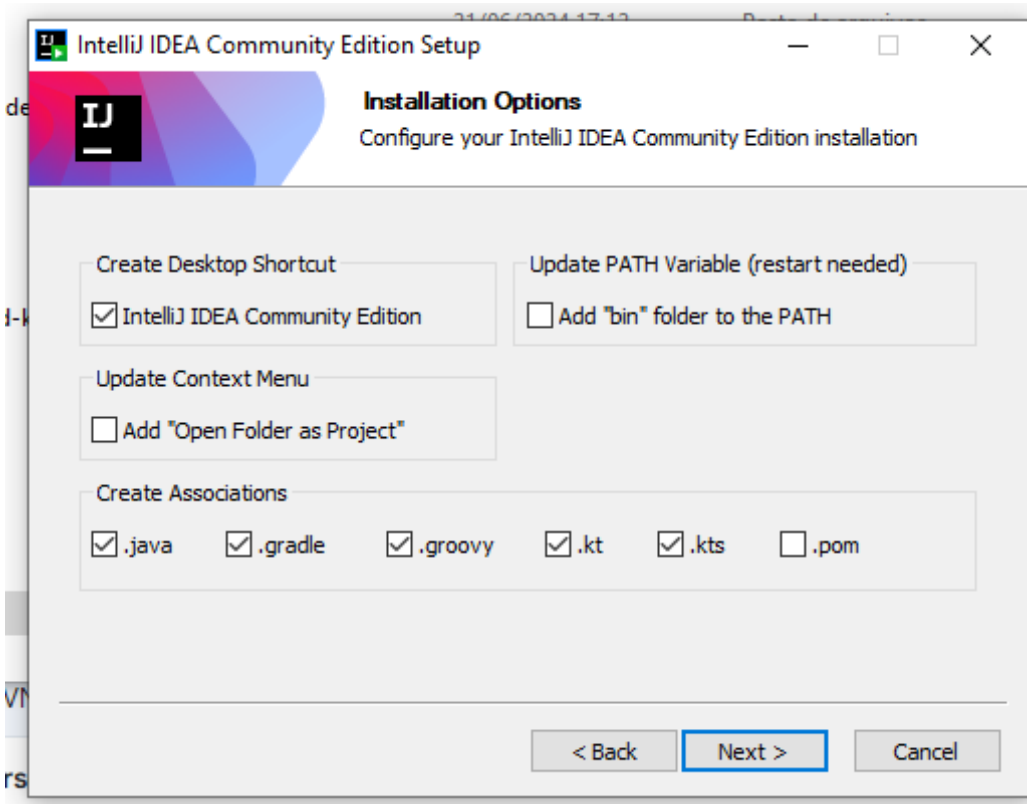
Intellij Idea Ultimate – Jet Brains

- Versão Community
  - Permite o uso de quase todas as funcionalidades da IDE
  - Não há custos
  - Destinada à aprendizagem

- Execute o arquivo de instalação (ideaIC-2024.2)

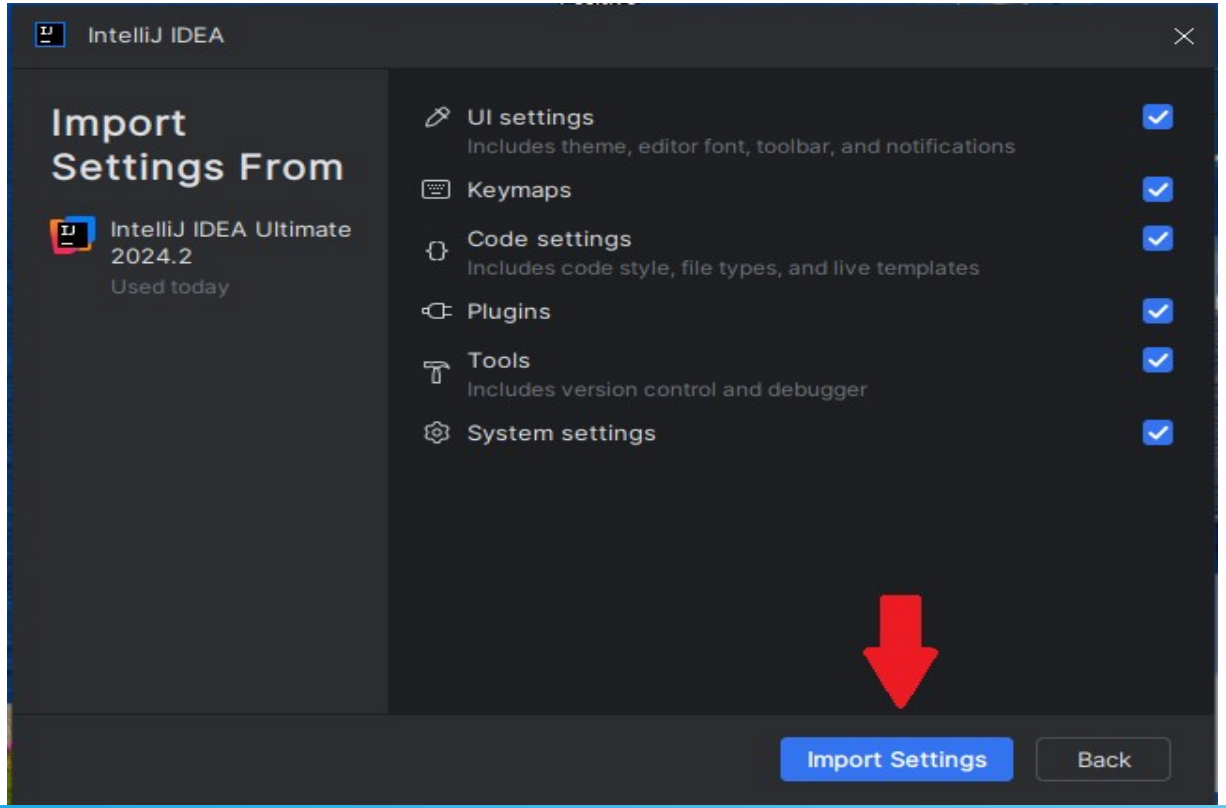
Intellij Idea Ultimate – Jet Brains

- Use o botão [Next] até a tela “Instalation Options”;
- Selecione os checkbox conforme imagem ao lado;
- Use o botão [Next], [Next], [Install] e [Finish]



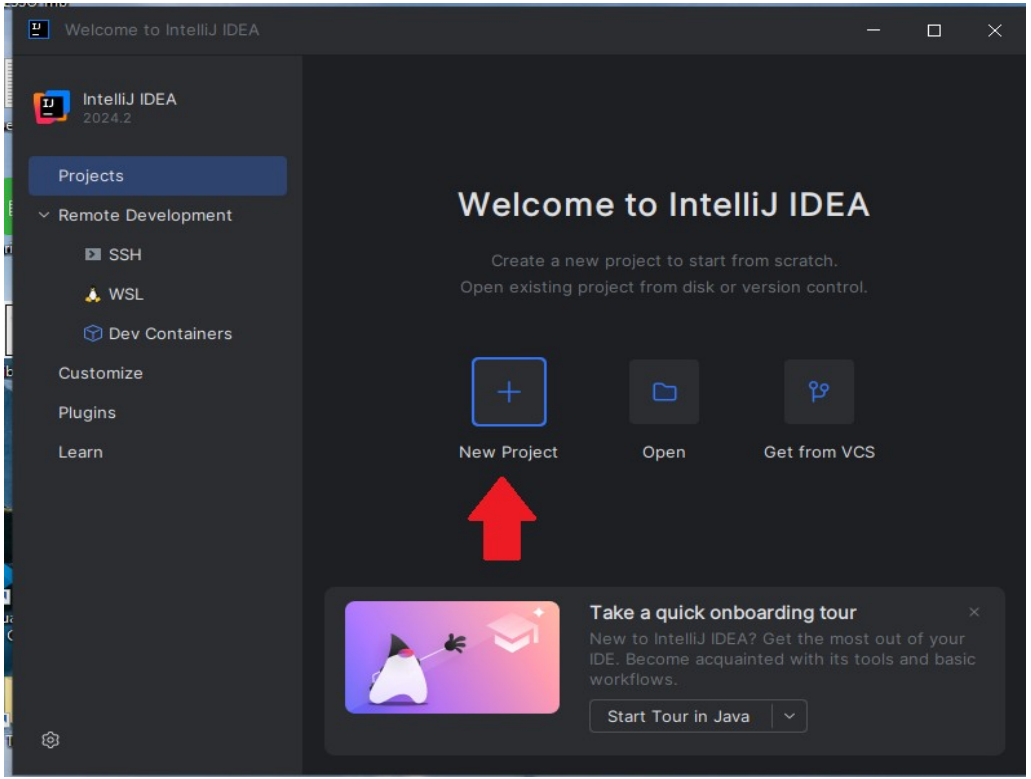
Intellij Idea Ultimate – Jet Brains

- Acesse:



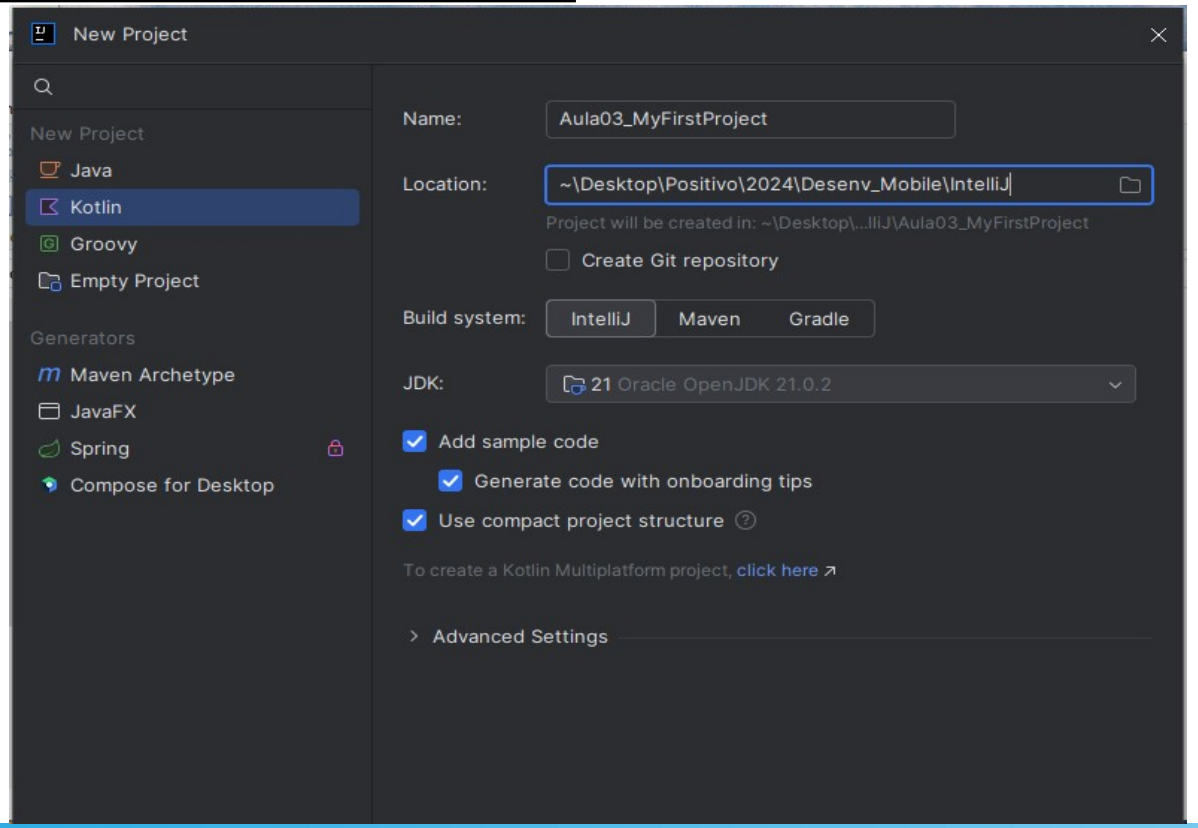
Intellij Idea Ultimate – Jet Brains

- Tela inicial da IDE
- Opção “New Project” para novos projetos / exercícios
- Opção “Open” para continuar ou alterar projetos já criados
- “Get from VCS” buscar projetos em versionadores (github...)

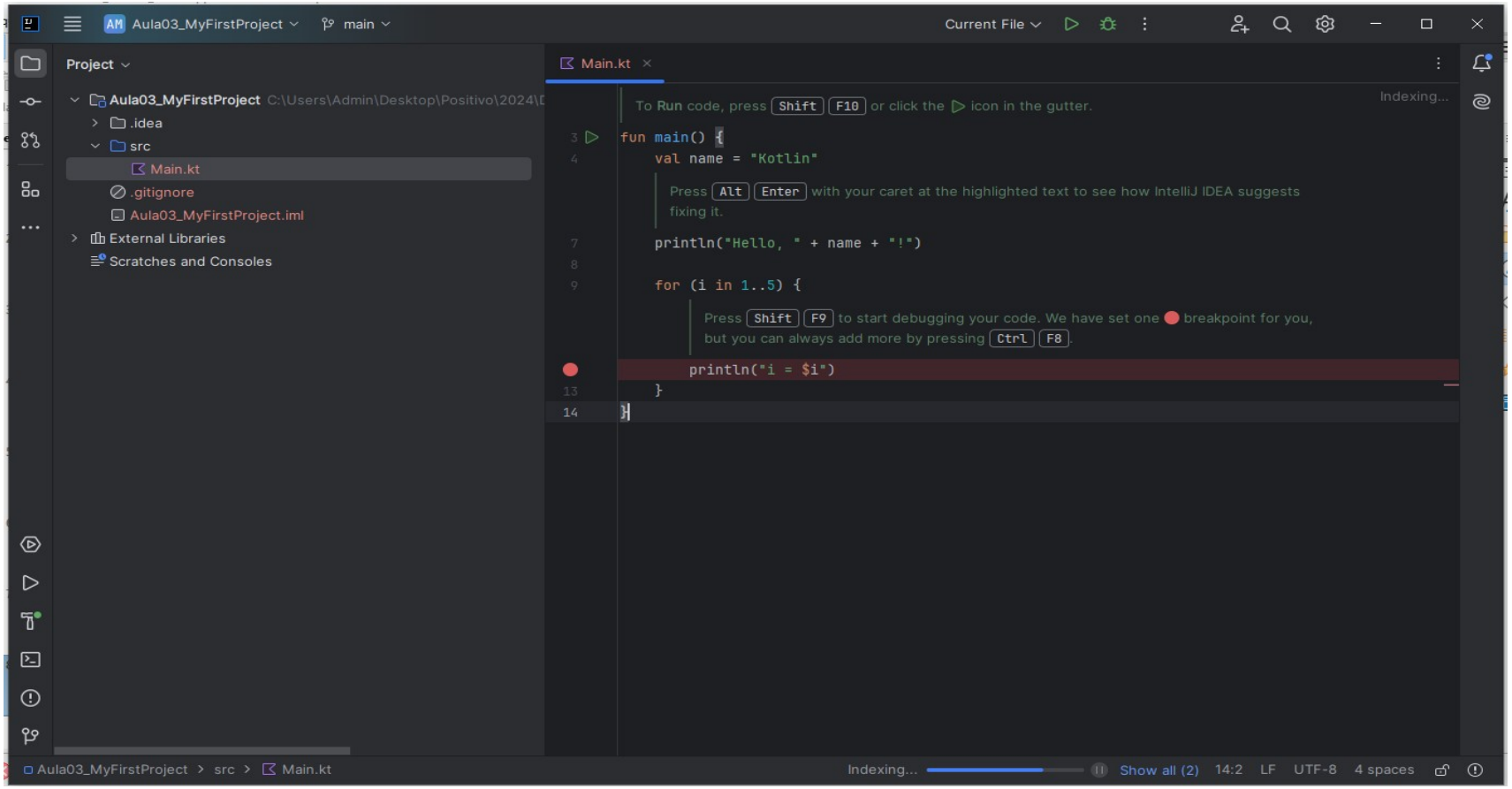


Intellij Idea Ultimate – Primeiro Projeto

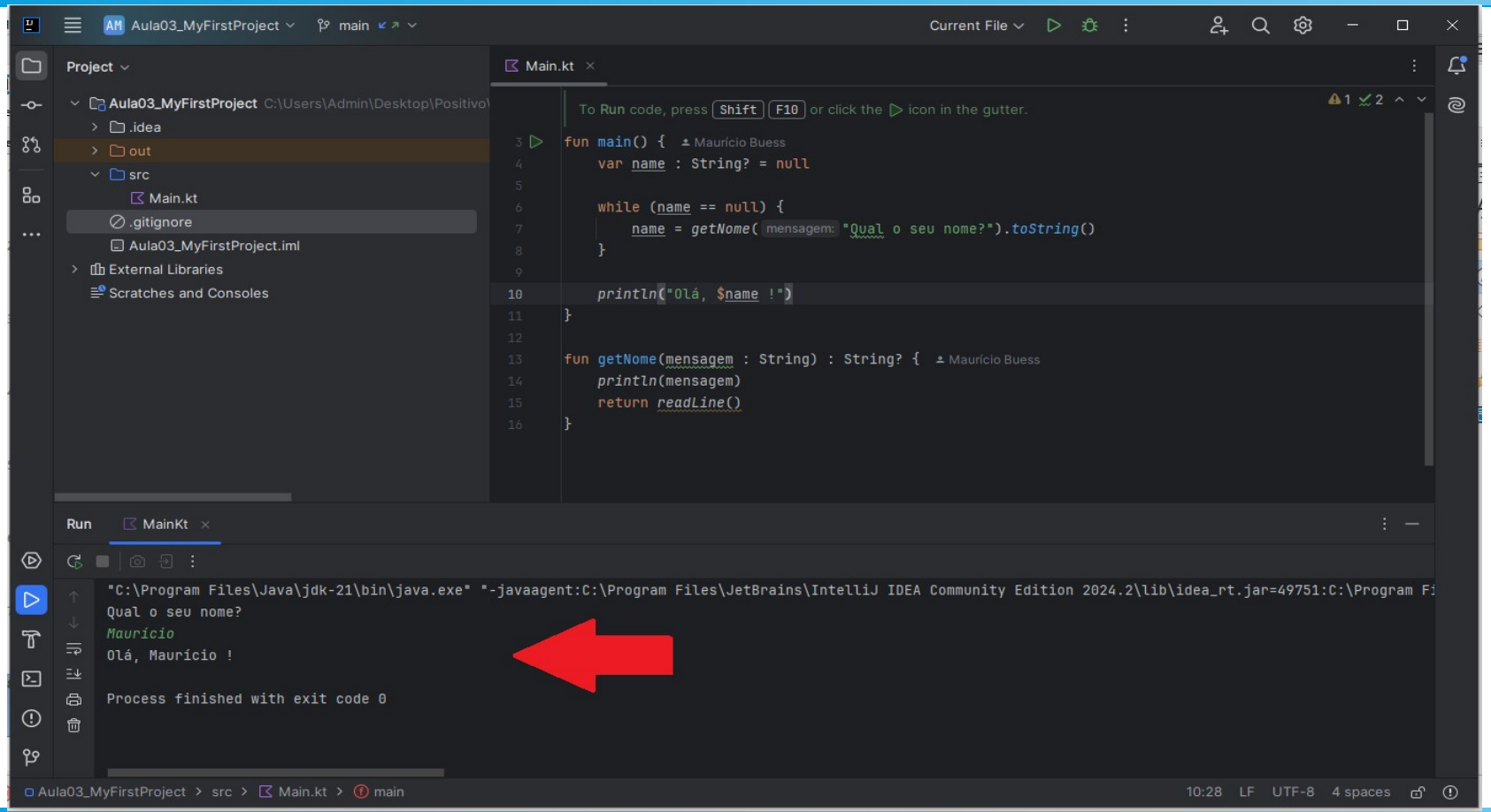
- Criar novo projeto selecionando o Kotlin como linguagem escolhida;
- Marque opção para iniciar um projeto com exemplo de código
- Nome do projeto: Aula03_MyFirstProject



Desenvolvimento Mobile



Desenvolvimento Mobile



Exercício 1: Calculadora de Desconto

Você foi contratado para criar um programa simples que calcula o valor final de um produto com desconto. O programa deve receber o preço original do produto e a porcentagem de desconto, e deve calcular e exibir o preço final após o desconto. O programa deve ser finalizado quando o operador informar o preço original menor ou igua a zero.

Dicas:

- Use a função readLine() para obter a entrada do usuário.
- Converta a entrada de string para Double usando toDouble().
- Calcule o valor do desconto multiplicando o preço original pela porcentagem de desconto dividida por 100.
- Subtraia o valor do desconto do preço original para obter o preço final.
- Exiba o preço final utilizando println().

```
fun main() {
    var precoOriginal : Double = 0.00
    var porcentagemDesconto : Double = 0.00
    var desconto : Double
    var precoFinal : Double

    do {
        print("Digite o preço original do produto (0 para finalizar): ")
        precoOriginal = readLine()!!.toDouble()

        if (precoOriginal > 0) {
            print("Digite a porcentagem de desconto: ")
            porcentagemDesconto = readLine()!!.toDouble()

            desconto = precoOriginal * (porcentagemDesconto / 100)
            precoFinal = precoOriginal - desconto

            println("O preço final com desconto é: R$ $precoFinal")
        }
    } while (precoOriginal > 0)
}
```

Comentários:

- **Entrada do Usuário:** readLine() solicita ao usuário uma entrada de texto.
- **Verificação Not-null**: !! verifica que a entrada não é null. Se for null, uma exceção será lançada.
- **Conversão:** A String lida é convertida em um Double usando .toDouble().
- **Atribuição:** O valor Double é atribuído à variável precoOriginal.

```
fun main() {
    var precoOriginal : Double
    var porcentagemDesconto : Double

    do {
        precoOriginal = getValor("Digite o preço original do produto (0 para finalizar):")

        if (precoOriginal > 0) {
            porcentagemDesconto = getValor("Digite a porcentagem de desconto: ")
            informaDesconto(precoOriginal, porcentagemDesconto, "O preço final com desconto é: R$ ")
        }
    } while (precoOriginal > 0)
}
fun getValor(msg : String) : Double {
    print(msg)
    return readLine()!!.toDouble()
}
fun informaDesconto(precoVlr : Double, porcentualVlr : Double, msg : String) {
    println(msg + " ${ precoVlr - (precoVlr * (porcentualVlr / 100))}")
}
```

Exercício 2: Contador de Palavras

Uma empresa precisa "subir" um website em que há uma funcionalidade que permite ao usuário digitar um elogio ou crítica, mas esse espaço deverá ser limitado a uma certa quantidade de palavras a ser definida. O gerente do projeto pediu para você
criar um programa que conte o número de palavras em uma frase fornecida pelo usuário. O programa deve retornar e exibir o número total de palavras.

Dicas:

- Use a função readLine() para obter a frase do usuário.
- Utilize o método split() da classe String para dividir a frase em palavras, usando o espaço como delimitador.
- A função split() retorna uma lista; obtenha o tamanho dessa lista para contar o número de palavras.
- Exiba o número de palavras utilizando println().

```
fun main() {
    print("Digite uma frase: ")
    val frase = readLine()!!

    val palavras = frase.split(" ")
    val numeroPalavras = palavras.size

    println("Número de palavras na frase: $numeroPalavras")
}
```

Comentários:

- **val frase = readLine()!! -** readLine() lê uma linha de entrada do usuário a partir do console, retornando uma String? (? sinaliza que pode ser null). Associado ao operador !! que força o Kotlin a tratar o valor como não-null (gerando uma exceção caso o null ocorra). O resultado dessas operações é atribuído à constante frase.

- **val palavras = frase.split(" ") -** A variável frase (que contém a frase digitada pelo usuário) é dividida em palavras usando o método split(" ").
  split(" ") divide a string em partes, usando o espaço " " como delimitador.
  O resultado é uma lista de palavras (List<String>), que é armazenada na variável palavras.

- **val numeroPalavras = palavras.size -** palavras.size retorna o número de elementos na lista palavras, ou seja, o número de palavras na frase original.
  Esse valor (número de palavras) é armazenado na variável numeroPalavras.

```
fun main() {
    print("Digite uma frase: ")
    val frase = readLine()!!
    val contaPalavras = contaPalavras(frase, true)
    println("Conteúdo de [contaPalavras]-> $contaPalavras")

}
fun contaPalavras(frase : String, isMostraMsg : Boolean = false) :
Int {
    val totalPalavras = frase.split(" ").size
    if (isMostraMsg) {
        println("Número de palavras na frase: $totalPalavras")
    }
    return totalPalavras
}
```

Exercício 3: Calculadora de Média de Consumo de Combustível

Você precisa desenvolver um programa para calcular a média de consumo de combustível de um automóvel. O usuário deve informar a distância percorrida em quilômetros e a quantidade de combustível consumida em litros. O programa deve calcular e exibir a média de consumo de combustível em quilômetros por litro (km/l).

Dicas:

- Use a função readLine() para obter a distância percorrida e a quantidade de combustível do usuário.
- Converta as entradas de string para Double.
- Calcule a média de consumo dividindo a distância percorrida pela quantidade de combustível.
- Exiba a média de consumo utilizando println(), formatando o resultado para duas casas decimais para maior clareza.

```
fun main() {
    var distancia    : Double
    var combustivel  : Double
    var mediaConsumo : Double

    do {
        do {
            distancia = getValor("Digite a distância percorrida em km:")
        } while (!isNumeroInteiro(distancia))

        if (distancia > 0) {
            do {
                combustivel = getValor("Digite a quantidade de combustível consumida em litros: ")
            } while (!isNumeroInteiro(combustivel) || combustivel<=0)

            mediaConsumo = distancia / combustivel
            println("A média de consumo de combustível é: %.2f km/l".format(mediaConsumo))
        }
    } while (distancia>0)
}
```

```
fun getValor(msg : String) : Double {
    print(msg)
    val valor = readln()!!.toDouble()
    return valor
}
fun isNumeroInteiro(numero : Double) :
Boolean {
    return (numero%1)==0.0
}
```

Paradigma programação orientada a objetos

- O que são (representam) objetos?
- Quais são as características de um objeto?
- Quando é necessário “criar” um objeto?
- Como se transfere um objeto do mundo real para a memória da máquina?
- Quais são os princípios da POO?

Paradigma programação orientada a objetos

- O que são (representam) objetos?
  - Objetos são instâncias de classes. Eles representam entidades do mundo real ou conceitos abstratos dentro do contexto de um programa de computador.
  - Particularmente, considero objetos como resultado de uma situação onde o desenvolvedor deve criar um novo tipo de dados (dado abstrato complexo);
  - Dado abstrato pois tem sua origem de uma situação e não necessariamente de um objeto físico.
  - Complexo pois se trata de um tipo de dados diferente dos tipos de dados primitivos.

Paradigma programação orientada a objetos

- Quais são as características de um objeto?
  - Atributo(s), propriedade(s) ou campo(s) → o sinônimo de atributo é característica, qualidade. Logo, pode-se entender que atributo de um objeto são as características do mesmo.
  - As características de um objeto pertencem ao próprio objeto e são definidas por tipos de dados, podendo ser dos tipos primitivos ou complexos (outros objetos).

Exemplo:

```
class Carro(var cor: String
            , var marca: String
            , var modelo: String
            , var velocidade: Int)
```

Paradigma programação orientada a objetos

- Quais são as características de um objeto?
  - Além do atributo(s) os objetos possuem comportamentos que buscam descrever as possíveis ações pertinentes ou interessantes (na ótica da resolução do problema proposto) do mesmo.
  - O comportamento de um objeto é definido pelos métodos (também chamados de funções ou operações). Esses métodos operam nos atributos do objeto e podem realizar ações, como alterar o estado do objeto ou interagir com outros objetos.

```
class Carro(var cor: String, var marca: String, var modelo: String, var velocidade: Int) {
    fun acelerar() { velocidade += 10  }
    fun frear() { velocidade -= 10 }
}
```

Paradigma programação orientada a objetos

- Quais são as características de um objeto?

  - Cada objeto tem uma identidade única que o distingue de outros objetos, mesmo que eles tenham o mesmo estado (atributos com valores iguais) e comportamento. Em muitas linguagens, a identidade do objeto é representada por seu endereço de memória.
  - Esse comportamento ocorre, justamente, pela natureza permitir a existência de mais de um objeto idêntico ao outro.

Paradigma programação orientada a objetos

- Quando é necessário “criar” um objeto?
  - Quando for preciso representar uma entidade específica em um programa que deve ter um estado e comportamento próprios.
  - O uso de objetos permite a modelagem do problema de forma intuitiva, refletindo como essas entidades interagem no mundo real ou no contexto do problema que está sendo resolvido.
  - Objetos também são usados para agrupar dados e os comportamentos que operam sobre esses dados. Se há uma estrutura de dados que precisa de operações específicas, encapsular esses dados em um objeto é apropriado.
  - Criar objetos permite reutilizar código de forma eficiente. Se há uma lógica comum que se aplica a várias partes do sistema, encapsulá-la em uma classe permite que se crie vários objetos dessa classe sem duplicar o código.

Paradigma programação orientada a objetos

- Como se transfere um objeto do mundo real para a memória da máquina?
  - Para transferir um objeto do mundo real para a memória de uma máquina, deve-se seguir um processo que envolve modelar e representar o objeto no contexto de programação e computação;

  - **Modelagem**: Defina uma classe ou estrutura que represente o objeto real.
  - **Instanciação**: Crie uma instância do objeto, alocando memória e inicializando os atributos.
  - **Armazenamento**: O objeto é armazenado na memória heap ou stack.
  - **Manipulação**: Interaja com o objeto através de métodos e propriedades.
  - **Persistência**: Se necessário, serialize e armazene o objeto de forma permanente.
  - **Descarte**: A memória é liberada quando o objeto não é mais necessário.

Paradigma programação orientada a objetos

- Quais são os princípios da POO?
  - **Encapsulamento** protege o estado interno dos objetos e agrupa dados e métodos relacionados.
  - **Herança** permite a criação de novas classes a partir de classes existentes, promovendo o reuso de código.
  - **Polimorfismo** oferece flexibilidade ao permitir que objetos de diferentes classes sejam tratados de forma uniforme.
  - **Abstração** simplifica o uso e o entendimento de objetos, expondo apenas os detalhes essenciais e ocultando os complexos.

Exemplo objeto carro

```
class Carro(var cor: String
            , var marca: String
            , var modelo: String) {
    var velocidade: Int = 0
    fun acelerar() {
        velocidade += 10
    }
    fun frear() {
        velocidade -= 10
    }
    fun status() {
        println("O carro $marca $modelo de cor
                $cor está a $velocidade km/h.")
    }
}
```

```
fun main() {
    val meuCarro = Carro("Vermelho", "Toyota", "Corolla")
        // Criação de um objeto (instância da classe Carro)
    meuCarro.acelerar()  // Alteração do estado do objeto
    meuCarro.status()    // Comportamento do objeto
}
```

- **Encapsulamento** protege o estado interno dos objetos e agrupa dados e métodos relacionados.

```
class ContaBancaria(private var saldo: Double) {
    fun depositar(valor: Double) {
        if (valor > 0) {
            saldo += valor
        }
    }
    fun sacar(valor: Double) {
        if (valor <= saldo) {
            saldo -= valor
        }
    }
    fun verSaldo(): Double {
        return saldo
    }
}
```

- O atributo “saldo” só poderá ser alterado “dentro” do objeto

Desenvolvimento Mobile

- **Herança** permite criar novas classes a partir de classes existentes, promovendo o reuso de código. A classe nova (classe derivada ou filha) herda os atributos e métodos da classe existente (classe base ou pai).

```
open class Veiculo(val marca: String
                   , val modelo: String) {
    fun ligar() {
        println("O veículo está ligado")
    }
}
class Carro(marca: String, modelo: String
, val numPortas: Int) : Veiculo(marca, modelo) {
    fun abrirPortas() {
        println("Abrindo $numPortas portas")
    }
}
```

```
class Moto(marca: String
           , modelo: String)
                : Veiculo(marca, modelo) {
    fun empinar() {
        println("A moto está empinando")
    }
}
```

- As classes Carro e Moto herdam a classe Veiculo;
- Herança é, na verdade, a extensão de uma classe

Desenvolvimento Mobile

- **Polimorfismo** permite que objetos de diferentes classes sejam tratados como objetos de uma classe comum. Isso é possível porque classes derivadas podem substituir ou estender o comportamento das classes base.

```
fun mostrarDetalhes(veiculo: Veiculo) {
    println("Marca: ${veiculo.marca}, Modelo: $
{veiculo.modelo}")

    veiculo.ligar()
}
```

```
fun main() {
    val carro = Carro("Toyota", "Corolla", 4)
    val moto = Moto("Honda", "CB500")

    mostrarDetalhes(carro)
    mostrarDetalhes(moto)
}
```

- **Abstração** é o princípio de expor apenas os detalhes essenciais de um objeto e ocultar os detalhes de implementação. Em outras palavras, é a simplificação de um conceito ou entidade, mostrando apenas o que é relevante para o contexto em que é usado.

```
abstract class Forma {
    abstract fun calcularArea(): Double
    abstract fun calcularPerimetro(): Double
}
class Circulo(val raio: Double) : Forma() {
    override fun calcularArea(): Double {
        return Math.PI * raio * raio
    }
    override fun calcularPerimetro(): Double {
        return 2 * Math.PI * raio
    }
}
```

```
class Retangulo(val largura: Double
                , val altura: Double) : Forma() {
    override fun calcularArea(): Double {
        return largura * altura
    }
    override fun calcularPerimetro(): Double {
        return 2 * (largura + altura)
    }
}
```