Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática – CEEI
Unidade Acadêmica de Sistemas e Computação – UASC
Disciplina: Laboratório de Programação 2

Laboratório 02

Como usar esse guia:

- Leia atentamente cada etapa
- Referências bibliográficas incluem:
  - material de referência desenvolvido por professores de p2/lp2 em semestres anteriores (ONLINE)
  - o livro Use a cabeça, Java (LIVRO-UseCabecaJava)
  - o livro Java para Iniciantes (Livro-JavaIniciantes)
- Quadros com dicas tem leitura opcional, use-os conforme achar necessário
- Preste atenção nos trechos marcados como importante (ou com uma exclamação)!

Sumário

Acompanhe o seu aprendizado

Conteúdo sendo exercitado

- Classes básicas
- Uso de objetos
- Uso do toString
- Introdução a documentação com javadoc

Objetivos de aprendizagem

- Iniciar o entendimento sobre classes e objetos. Observar que uma classe encapsula atributos e métodos e os objetos são instâncias dessa classe. Um programa OO é constituído de objetos que realizam tarefas a partir das “mensagens” enviadas a eles (chamadas de métodos públicos) e podem se comunicar entre si.

Perguntas que você deveria saber responder após este lab

- O que é uma classe?
- O que é um objeto?
- Qual a diferença entre encapsulamento e ocultação da informação?
- O que significa dizer que cada objeto possui sua identidade?
- Quais os componentes essenciais de uma classe?
- Objetos são acessados por meio de variáveis de referência. O que isso significa?
- Dê exemplos de métodos de acesso (get) e modificadores (set).
- O que é o método toString() e em quais situações devemos sobrescrevê-lo?

Material para consulta

- Referências bibliográficas incluem:
    - material de referência desenvolvido por professores de p2/lp2 em semestres anteriores (ONLINE)
    - o livro Use a cabeça, Java
    - o livro Java para Iniciantes
    - os livros:
        - Core Java
            - https://plataforma.bvirtual.com.br/Acervo/Publicacao/1238
        - Java, Como programar (Deitel)
            - https://plataforma.bvirtual.com.br/Acervo/Publicacao/39590
- Tutorial oficial Java
- Javadoc (tutorial oficial)
- Entendendo a diferença entre classes e objetos
- Tutorial de uso do GitHub Classroom

Introdução

A disciplina de Programação 2 tem como foco o design e a construção de programas e funcionalidades mais completas que necessitam de mais design do código. Nesse sentido, este e os próximos laboratórios trabalharão de forma diferente: cada laboratório terá uma especificação das funcionalidades a serem implementadas. Os critérios de avaliação dos laboratórios terão como base essa especificação.

É importante também que, a partir de agora, você comece a trabalhar com a IDE Eclipse. Uma IDE é um ambiente integrado de desenvolvimento que oferece muitas facilidades durante a codificação. Recomendamos **fortemente** que neste momento você utilize o **Eclipse,** pois ele auxiliará no seu processo de aprendizagem. Ao longo dos labs daremos algumas dicas de como realizar certas ações no Eclipse, e não temos como dar todas as dicas para outras IDEs.

Toda IDE moderna trabalha com o conceito de PROJETO. Um projeto é uma coleção de diretórios, arquivos de imagem, código-fonte, além de configurações próprias (versão de compilação, versão de execução, dependências) e dados de versão de código. Tipicamente, um projeto é estruturado com o diretório **src**, onde ficarão os pacotes com os códigos-fonte do projeto. É comum a presença de outros diretórios, como resources, para o armazenamento de imagens e sons, javadoc, para a documentação, entre outros.

Como exemplo de classe e uso de objetos, considere o exemplo do controle acadêmico abaixo. Observe como o Aluno foi definido na classe Aluno (Aluno.java) e como os objetos dessa classe foram utilizados no ControleAcademico.

```
public class Aluno {

	private String nome;
	private int anoNascimento;
	private double cra;

	public Aluno(String nome, int anoNascimento) {
		this.nome = nome;
		this.cra = 0.0;
		this.anoNascimento = anoNascimento;
	}

	public void setCra(double cra) {
		this.cra = cra;
	}

	public int getIdade() {
		return 2021 - anoNascimento;
	}

	public String toString() {
		return "Aluno - "  + this.nome;
	}
```

```
}
```

```
public class ControleAcademico {
      public static void main(String[] args) {
            Aluno a1 = new Aluno("JOAO", 1990);
            Aluno a2 = new Aluno("MARIA", 2000);
            System.out.println(a1);
            System.out.println(a2);
      }
}
```

Importante!

A partir de agora você deve documentar seu código! Para isso utilizaremos o modelo Javadoc de documentação. O javadoc é um modelo de comentário de código que pode gerar, automaticamente, documentação como essa:

https://docs.oracle.com/en/java/javase/21/docs/api/java.base/java/lang/String.html

O Javadoc costuma ser o primeiro contato do desenvolvedor com um programa, e é importante que ele seja bem escrito. Este bloco de comentários deve ser inserido anteriormente à entidade que irá documentar. Veja os exemplos de documentação para diversas entidades de Java:

classe	`/**` `* Representação de um estudante, especificamente de computação, matriculado da * UFCG. Todo aluno precisa ter uma matrícula e é identificado unicamente` `* por esta matrícula.` `*` `* @author Matheus Gaudencio` `*/` `public class Aluno {` `...`
atributo	`/**` `* Matrícula do aluno. No formato 2XXXYYZZZ onde XX é o ano e` `* semestre de entrada, YY é o código do curso e ZZ é um identificador * do aluno no curso.` `*/` `private String matricula;`
construtor	`/**` `* Constrói um aluno a partir de sua matrícula e nome.` `* Todo aluno começa com o campo CRA como nulo.` `*` `* @param matricula a matrícula do aluno, no formato "0000000000"` `* @param nome o nome do aluno` `*/` `public Aluno(String matricula, String nome) {` `...`
método	`/**`

	`` * Retorna a String que representa o aluno. A representação segue o * formato "MATRICULA - Nome do Aluno". * * @return a representação em String de um aluno. */ public String toString() { ... ```
Métodos com parâmetros usam a notação @param para indicar o que significa cada parâmetro.	

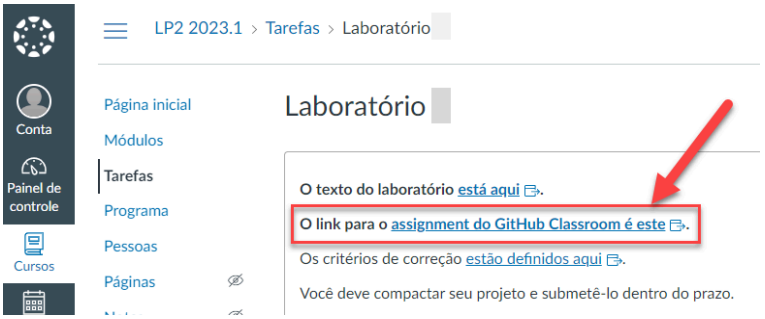
No Eclipse, no menu Project, opção "Generate Javadoc..." o Eclipse vai organizar o javadoc escrito por você em páginas HTML semelhantes às da API de java.

Tutorial de uso do GitHub Classroom

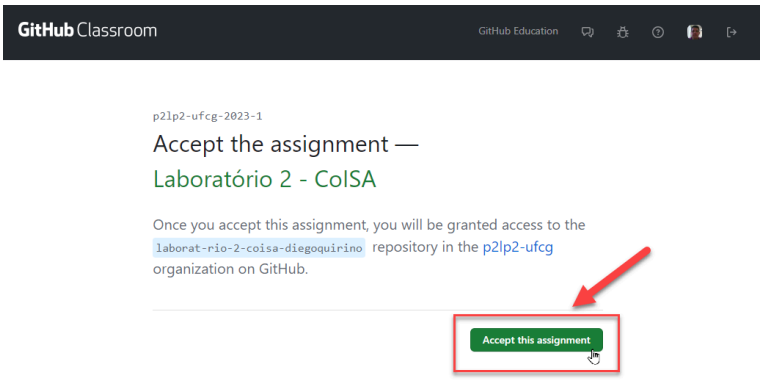
Vamos utilizar o **GitHub Classroom** para o envio e correção de atividades da Disciplina LP2. Todas as vezes que um novo laboratório é disponibilizado, o procedimento contido neste tutorial tem que ser realizado novamente. O **GitHub** é uma plataforma online de hospedagem de código-fonte, amplamente utilizada por desenvolvedores e equipes de programação em todo o mundo. Essa ferramenta é vital para o desenvolvimento individual ou colaborativo de software, permitindo que programadores trabalhem em projetos em conjunto, compartilhando, revisando e gerenciando suas alterações de forma eficiente. Já o **GitHub Classroom** é uma extensão do *GitHub* que visa facilitar o processo de ensino e aprendizagem de programação, fornecendo um ambiente virtual onde projetos e exercícios são disponibilizados aos alunos, enquanto estes utilizam o *GitHub* como plataforma para o envio do código-fonte com suas resposta.

Como treinamento, ***vamos enviar agora a classe Aluno*** (veja o código-fonte na seção introdutória) como resposta ao *assignment* do ***Laboratório 2 - CoISA***. Não se preocupe, **até a data final de entrega** você pode **reenviar** o seu código quantas vezes forem necessárias.

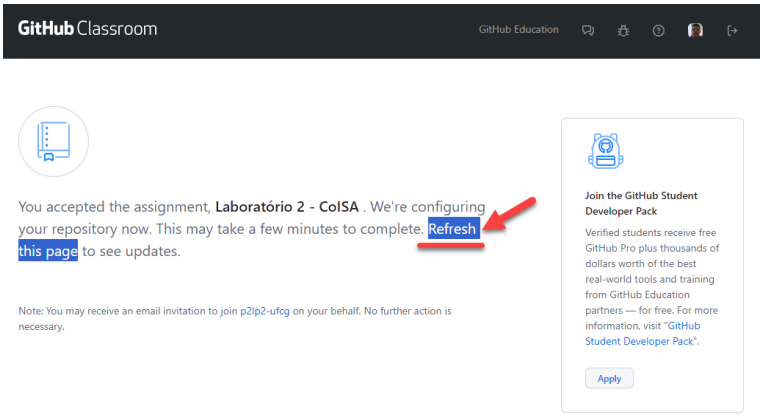
1. Na atividade do Laboratório 2 no Canvas, ***clique link para o assignment do GitHub Classroom***.



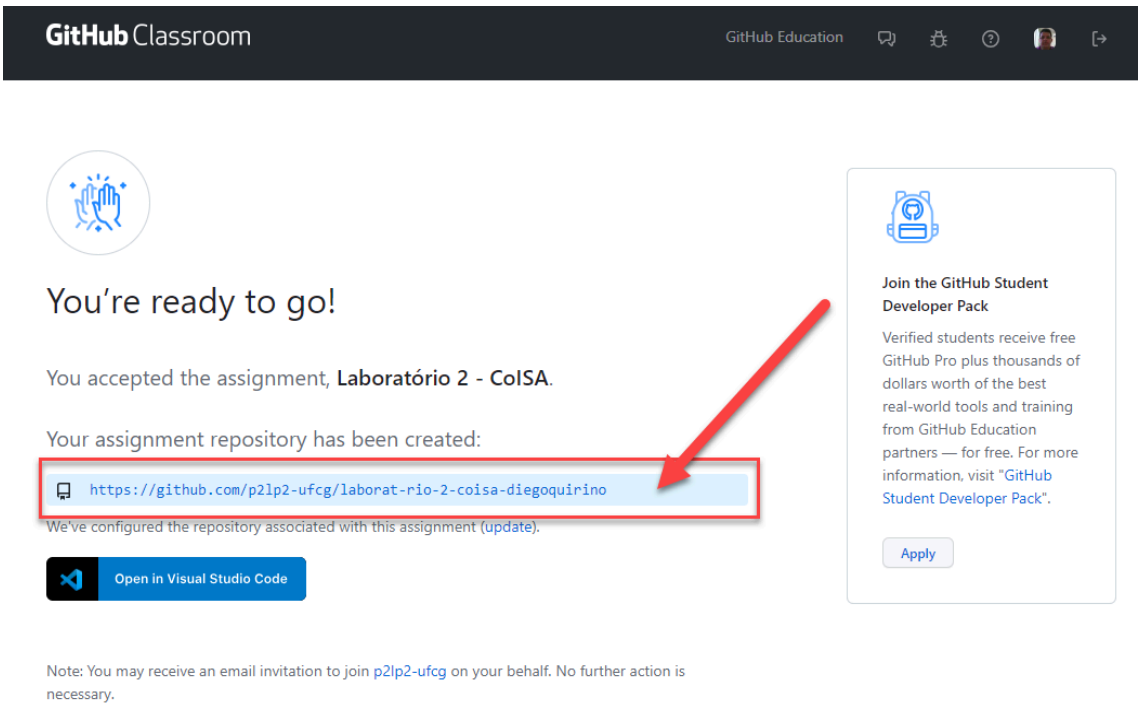
2. Uma vez *logado* no *GitHub*, **clique no botão (verde) "Accept this assignment"**.



3. Aguarde um minuto e, após este tempo, **atualize a página do navegador** (F5).

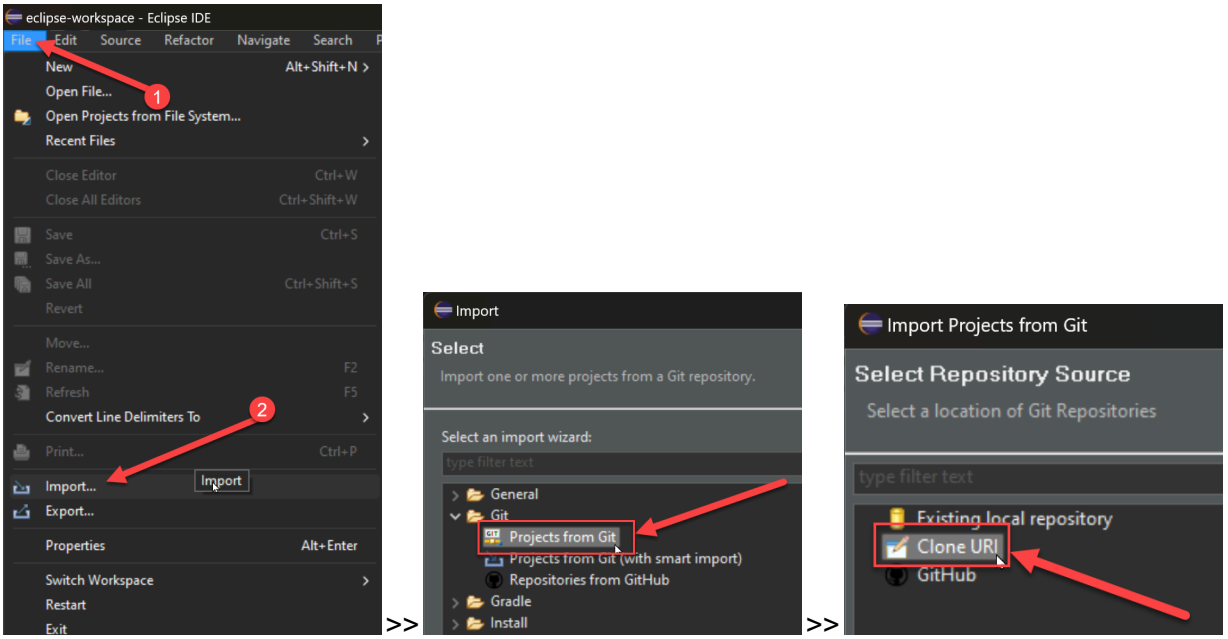


4. **Pronto!** Neste momento o seu repositório da atividade está criado no GitHub Classroom.
    a. **Copie o endereço exibido** para o assignment repository, pois é ele que iremos utilizar no Eclipse para baixar o código-fonte inicial do laboratório.
    b. *Lembre-se: este repositório é privado e não pode ser compartilhado com outros colegas de turma*.

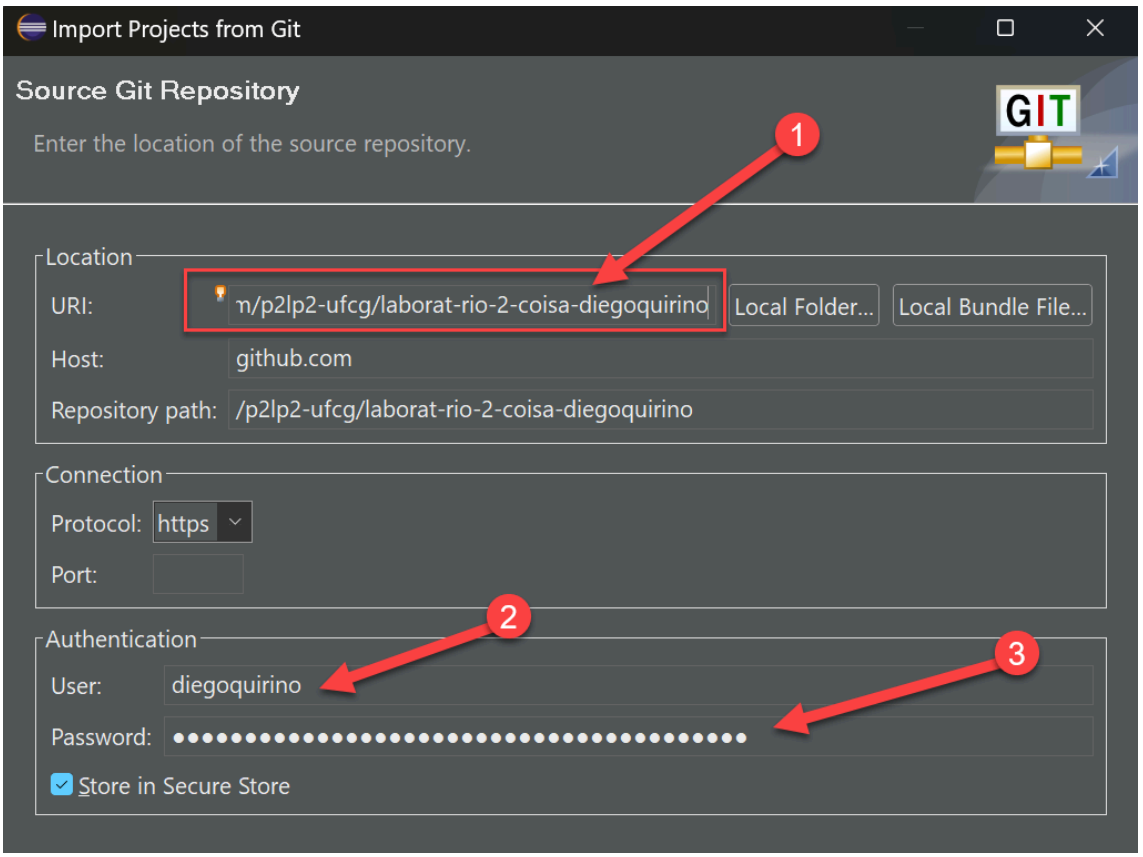


5. **Abra o Eclipse** - demonstraremos no Eclipse
    a. Selecione *File* > *Import* (para importação de projetos)

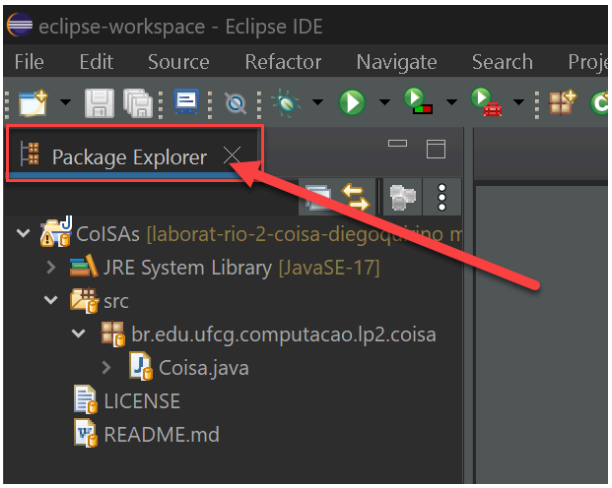
b. Em seguida, *Git > Projects from Git*
c. Por fim, *Clone URI*



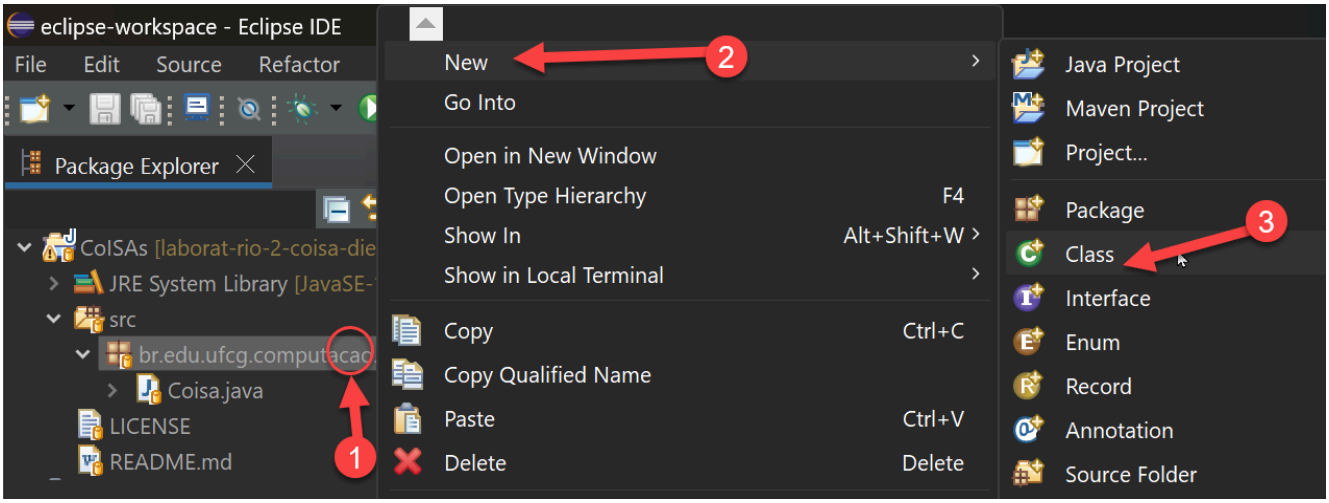
6. Agora, **insira [1] no campo "URI" a URL do seu repositório** no GitHub Classroom.
   a. Em seguida, **digite [2] no campo "User" o nome do seu usuário do GitHub**; e,
   b. **[3] No campo "Password" o seu "Personal access tokens (classic)"** (este é o tutorial para criá-lo → documentação oficial do GitHub).



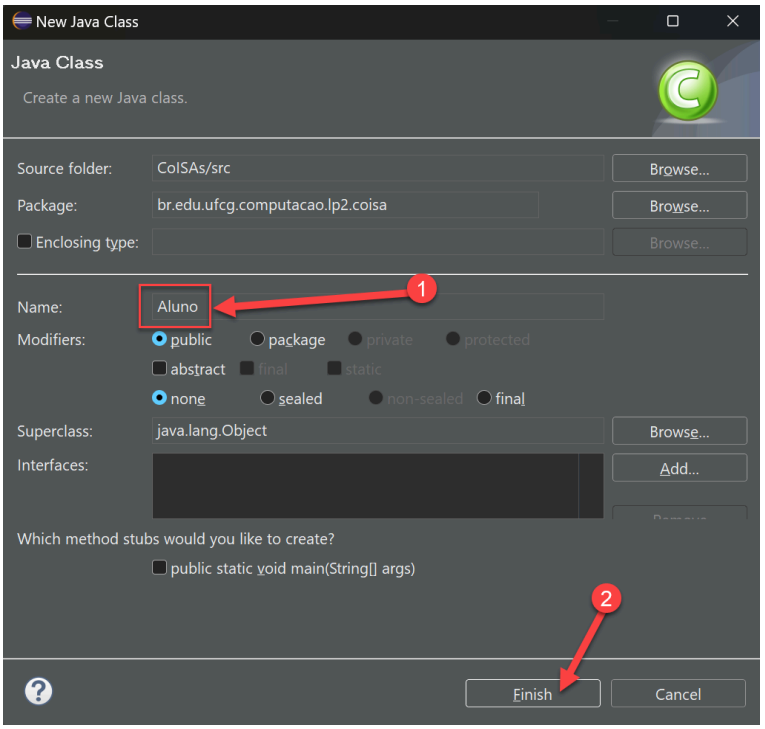
7. ***Avance pelas próximas etapas na IDE até que o projeto seja aberto no editor*** (a visão que melhor exibe este tipo de projeto no Eclipse é a *Package Explorer*)



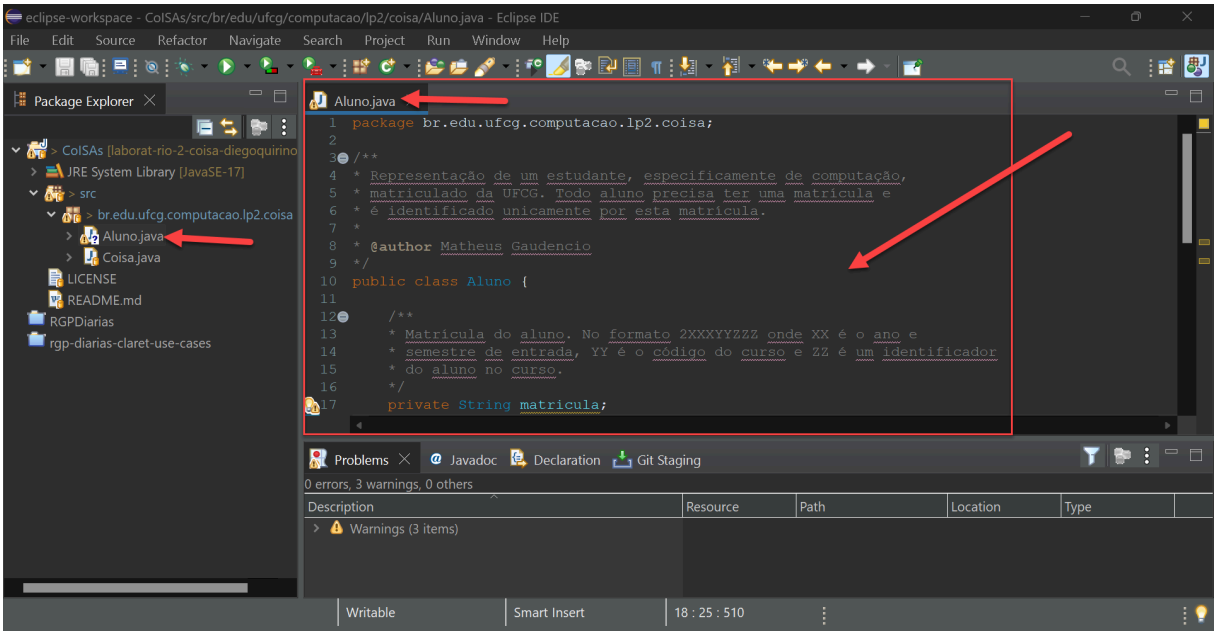
8. No pacote `br.edu.ufcg.computacao.p2lp2.coisa`, **crie a classe `Aluno`** (veja o código na seção introdutória), através da operação: [1] clique com o botão direito do mouse sobre o pacote; [2] New; [3] Class.



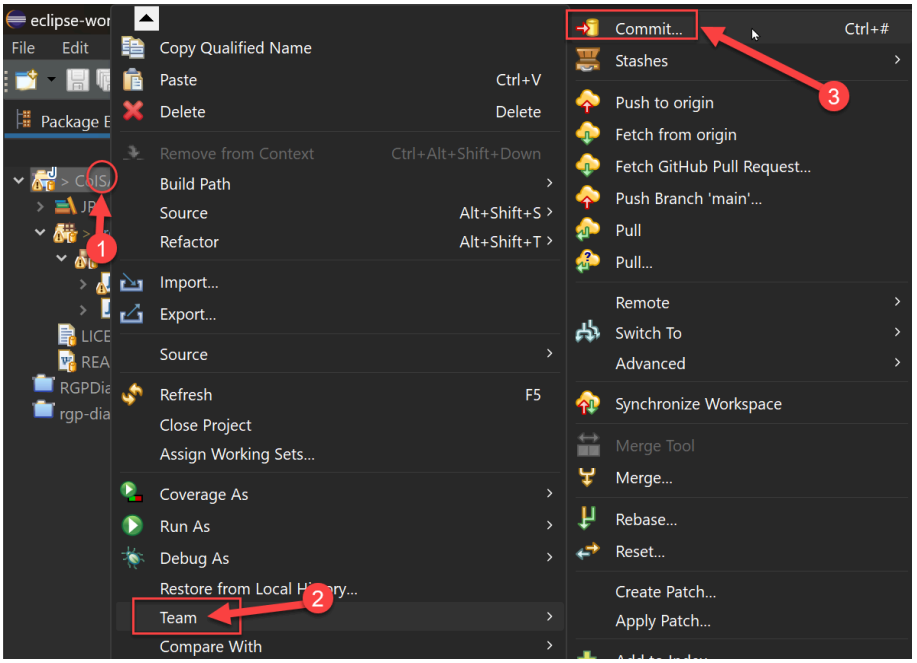
9. **[1] Digite o nome da classe** (no caso, `Aluno`) e **[2] clique para finalizar**.



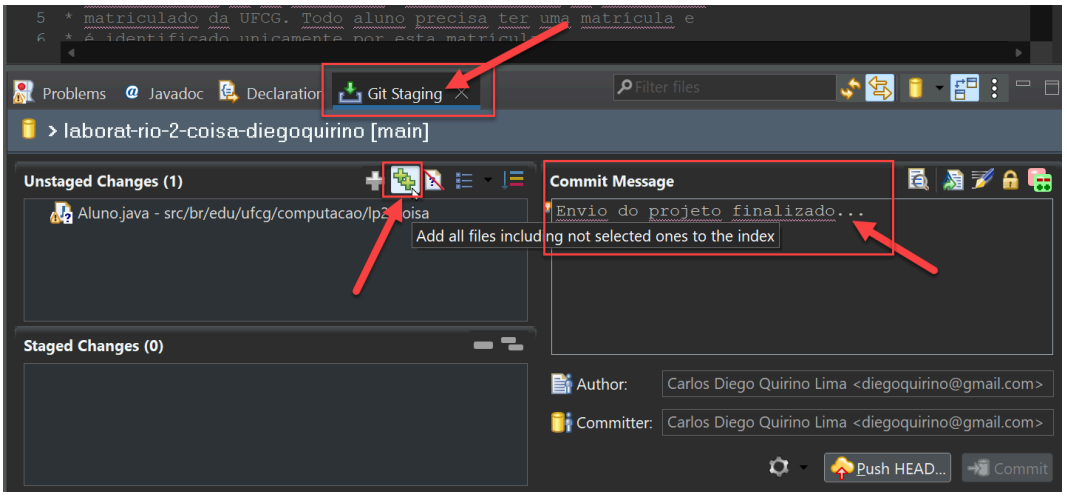
10. **Insira o conteúdo** da classe (sugerimos utilizar parte [do código disponibilizado na seção introdutória]).



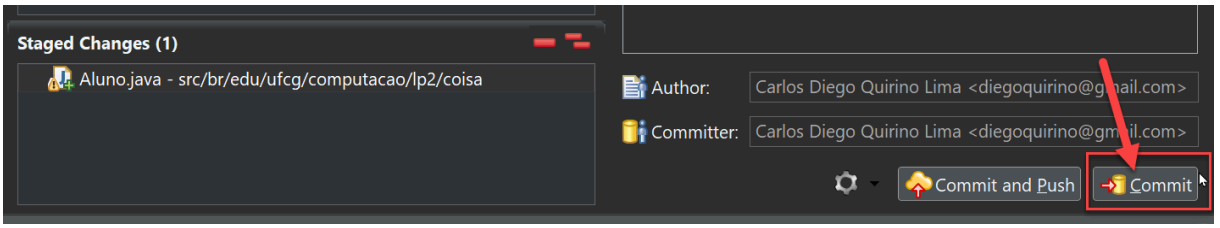
11. Chegou a hora de *enviar o projeto*.
    a. [1] Clique com o **botão direito do mouse sobre o Projeto**
    b. Em seguida, [2] selecione a **opção *Team***;
    c. E, por último, [3] selecione ***Commit***…



12. Neste momento, o Eclipse irá apresentar a **[1] aba de Git Staging**. Nela, você deverá **[2] adicionar as modificações** que foram feitas e **[3] informar um texto (mensagem) que ficará registrado no commit**.

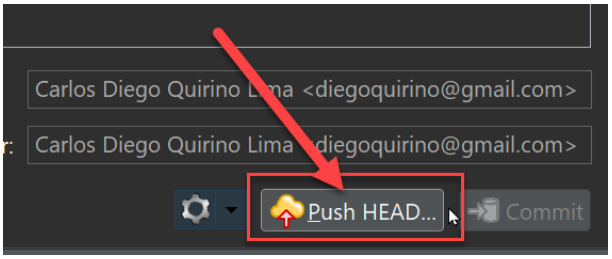


13. Agora, vamos **salvar localmente** o trabalho que está sendo realizado, **efetivando o commit.**



No contexto do controle de versão, "commit" se refere a salvar as alterações feitas em um arquivo ou conjunto de arquivos em um repositório local. Cada commit representa um ponto na história do projeto, onde você registra as modificações realizadas. É como tirar uma fotografia do estado atual dos seus arquivos. Ao fazer um commit, você adiciona um comentário descrevendo as mudanças realizadas, o que ajuda a entender o que foi feito no futuro e facilita a colaboração em equipe. Você pode fazer quantos commits quiser, salvando o que está sendo feito localmente, inserindo um comentário para cada nova adição.

14. Por fim, vamos **salvar remotamente ("na nuvem", enviando para o GitHub Classroom)**, a entrega efetiva do trabalho, **realizando o push**.



O "push" está relacionado ao ato de enviar os commits que você fez no repositório local para um repositório remoto, como o GitHub. Após realizar alterações e commits no seu computador, é possível compartilhá-los com outras pessoas ou manter um backup seguro enviando-os para um servidor remoto através do comando "push". Dessa forma, suas **alterações se tornam acessíveis a outros colaboradores (o professor e os monitores, para fins de correção do projeto)**. No contexto profissional, possibilita a colaboração e o compartilhamento eficiente do código em um ambiente colaborativo.

Gostou de usar o Git? Que tal aprender um pouco mais?

Abaixo, segue um material complementar (como bônus, não-obrigatório):

- GitHub | Guia Completo do Iniciante
- https://nvie.com/posts/a-successful-git-branching-model/

Controle Institucional da Situação Acadêmica (CoISA)

Neste projeto, você deve desenvolver um sistema capaz de gerenciar o uso dos laboratórios de Ciência da Computação (LCC's) e sua vida acadêmica. O CoISA é um sistema complexo, logo a separação de responsabilidades através da criação de classes é de **extrema** importância, pois tem como objetivo estruturar, coerentemente, o seu programa.

A vida do aluno pode ser organizada em quatro atividades básicas: **(1) organizar seu tempo de uso de internet para as disciplinas**, o que é bem importante considerando as distrações das redes sociais, **(2) estudar para as disciplinas**, **(3) organizar resumos de estudo** e **(4) acompanhar sua rotina de descanso**. Para permitir o controle dessas 4 atividades, você irá desenvolver um sistema que permite avaliar a quantidade de tempo de internet (online) que você tem usado nas disciplinas, a quantidade de horas que você tem estudado, o cadastro de resumos de estudos e, por fim, como está sua rotina de descanso.

Assim, para cada uma das atividades, é descrito um conjunto de valores referentes a cada atividade e de ações que podem ser feitas para aquela atividade.

Atividade	**Estado (dados)**	**Ações**
Rotina de descanso	Horas de descanso Números de semana	Definir horas de descanso Definir números de semana Verificar estado de descanso
Registro de tempo online	Nome da disciplina Tempo investido online (horas com a disciplina) Tempo esperado (horas para a disciplina)	Adicionar tempo online Verificar tempo online Imprimir estado
Disciplina	Nome da disciplina Horas de estudo Notas 1, 2, 3 e 4	Cadastrar horas de estudo Cadastrar uma nota Calcular média Verificar se foi aprovado Imprimir estado
Resumos de estudo	Conteúdo Resumos	Adicionar resumo Conta resumos Visualizar resumo Exibir resumos

Para facilitar a sua vida, já existe uma classe pronta que irá ser executada para testar o funcionamento do seu programa. Por este programa é possível ver as classes que devem ser implementadas (dica: pelo menos uma classe para cada atividade) bem como os métodos públicos. Você deve copiar essa classe no seu programa e fazê-la funcionar sem alterações de acordo com as especificações que virão a seguir. Execute sempre esta classe para garantir que você está desenvolvendo corretamente cada atividade.

```java
package br.edu.ufcg.computacao.p2lp2.coisa;
public class Coisa {
  public static void main(String[] args) {
      registrarDescanso();
      System.out.println("-----");
      registrarTempoOnline();
      System.out.println("-----");
      controlarDisciplina();
      System.out.println("-----");
      registrarResumos();
  }
  public static void registrarDescanso() {
      Descanso descanso = new Descanso();
      System.out.println(descanso.getStatusGeral());
      descanso.defineHorasDescanso(30);
      descanso.defineNumeroSemanas(1);
      System.out.println(descanso.getStatusGeral());
      descanso.defineHorasDescanso(26);
      descanso.defineNumeroSemanas(2);
      System.out.println(descanso.getStatusGeral());
      descanso.defineHorasDescanso(26);
      descanso.defineNumeroSemanas(1);
      System.out.println(descanso.getStatusGeral());
  }
  private static void registrarTempoOnline() {
      RegistroTempoOnline tempoLP2 = new RegistroTempoOnline("LP2", 30);
      tempoLP2.adicionaTempoOnline(10);
      System.out.println(tempoLP2.atingiuMetaTempoOnline());
      tempoLP2.adicionaTempoOnline(10);
      tempoLP2.adicionaTempoOnline(10);
      System.out.println(tempoLP2.atingiuMetaTempoOnline());
      tempoLP2.adicionaTempoOnline(2);
      System.out.println(tempoLP2.atingiuMetaTempoOnline());
      System.out.println(tempoLP2.toString());
      RegistroTempoOnline tempoP2 = new RegistroTempoOnline("P2");
      System.out.println(tempoP2.toString());
  }
  private static void controlarDisciplina() {
      Disciplina prog2 = new Disciplina("PROGRAMACAO 2");
      prog2.cadastraHoras(4);
      prog2.cadastraNota(1, 5.0);
      prog2.cadastraNota(2, 6.0);
      prog2.cadastraNota(3, 7.0);
```

```
        System.out.println(prog2.aprovado());
        prog2.cadastraNota(4, 10.0);
        System.out.println(prog2.aprovado());
        System.out.println(prog2.toString());
    }
    private static void registrarResumos() {
        RegistroResumos meusResumos = new RegistroResumos(100);  // 100 resumos

        meusResumos.adiciona("Classes", "Classes definem um tipo e a base de
código para criação de objetos.");
        meusResumos.adiciona("Tipo", "Identifica a semântica (operações e
significados) de um conjunto de dados.");

        String[] resumos = meusResumos.pegaResumos();

        for (int i = 0; i < meusResumos.conta(); i++) {
            System.out.println(resumos[i]);
        }

        System.out.println();
        System.out.println("Resumos: ");
        System.out.println(meusResumos.imprimeResumos());
        System.out.println(meusResumos.temResumo("Classes"));
        System.out.println(meusResumos.temResumo("Objetos"));
    }
}
```

Essa classe ao ser executada deve produzir a seguinte saída:

```
cansado
descansado
cansado
descansado
-----
false
true
true
LP2 32/30
P2 0/120
-----
false
true
PROGRAMACAO 2 4 7.0 [5.0, 6.0, 7.0, 10.0]
-----
Classes: Classes definem um tipo e a base de código para criação de objetos.
```

```
Tipo: Identifica a semântica (operações e significados) de um conjunto de dados.

Resumos:
- 2 resumo(s) cadastrado(s)
- Classes | Tipo
true
false
```

A seguir apresentaremos em detalhe cada funcionalidade, buscando facilitar o seu entendimento do que deve ser feito.

1. Sua rotina de descanso

Primeiramente, é importante acompanhar a rotina de descanso do aluno. Para nosso aluno, ele deve descansar 26 horas por semana, ou mais, para se considerar descansado. Isto, claro, não inclui as horas de sono, mas de atividades de lazer em geral. Considere que o aluno começa cansado, caso não tenha registrado horas de descanso ou número de semanas. A classe de descanso precisa ter 3 métodos:

- **void defineHorasDescanso(int valor)**
- **void defineNumeroSemanas(int valor)**
- **String getStatusGeral()**

2. Registro de tempo online

O registro de tempo online deve ser responsável por manter a informação sobre quantidade de horas de internet que o aluno tem dedicado a uma disciplina remota. Para cada disciplina, seria criado um objeto para controle desse estado (tempo online usado).

Para uma disciplina de X horas, espera-se que o aluno dedique o dobro de tempo online para realizar tal disciplina. Por padrão, consideramos disciplinas de 60 horas, onde o tempo online esperado seria de 120 horas. É possível passar a quantidade de horas online esperada para a disciplina como parâmetro na construção do objeto que representa o registro de tempo online. O *toString* deste objeto deve gerar uma String que o representa contendo: o nome da disciplina, o tempo online já usado e o tempo online esperado.

Ao atingir o tempo esperado, o método de verificação de tempo online deve indicar que o usuário atingiu o tempo esperado para aquela disciplina (através de um booleano). Mesmo atingindo o tempo online esperado, o usuário ainda pode adicionar dados, ou seja, ele pode usar mais tempo online para aquela disciplina.

Para implementar esta funcionalidade, você deve criar uma classe **RegistroTempoOnline** com dois construtores: **RegistroTempoOnline (String nomeDisciplina)** e **RegistroTempoOnline (String nomeDisciplina, int tempoOnlineEsperado)**. O tempo será representado em horas.

É importante também implementar os métodos abaixo:

- **void adicionaTempoOnline(int tempo)**
- **boolean atingiuMetaTempoOnline()**

- **String toString()**

3. Disciplina

Por padrão toda disciplina tem 4 notas. Calcula-se a média da disciplina, fazendo a média aritmética dessas 4 notas (sem arredondamento). Todo aluno é considerado aprovado quando atinge a média igual ou acima de 7.0. Caso alguma das notas não seja cadastrada, ela é considerada como zero. Se a alguma nota (nota 1, nota 2, nota 3 ou nota 4) for cadastrada mais de uma vez, consideramos a última nota cadastrada (ou seja, é possível repor notas nesse sistema). É possível cadastrar horas de estudo para determinada disciplina. As horas de estudo são cumulativas, ou seja, a nova quantidade cadastrada soma-se às previamente cadastradas.

O *toString()* deve gerar uma representação da disciplina que contenha o nome da disciplina, o número de horas de estudo, a média do aluno e as notas de cada prova.

Cada objeto que representa uma disciplina começa sem horas de estudo cadastradas. Para desenvolver esta funcionalidade, você deve implementar a classe **Disciplina** que tem o construtor **Disciplina(String nomeDisciplina)**. Implemente, também, os métodos:

- **void cadastraHoras(int horas)**
- **void cadastraNota(int nota, double valorNota)** // notas possíveis: 1, 2, 3 e 4
- **boolean aprovado( )**
- **String toString( )**

4. Resumo de Estudos

Para acompanhar os estudos, é preciso ter um pequeno registro de resumos dos estudos realizados ao longo do período. Para isso, é possível inicializar um registro de resumos que armazenará até uma quantidade limitada de resumos. Ao adicionar um resumo após atingir o limite de resumos, o primeiro resumo cadastrado é substituído. Em seguida, caso esse mesmo sistema receba uma nova adição, o segundo resumo cadastrado é substituído e assim sequencialmente.

Resumos são pequenos textos, descrições ou anotações referente a um tema. O resumo não está necessariamente a uma disciplina na medida que um estudo de um tema pode ser útil para diferentes conteúdos.

Para controlar seus resumos, você deve criar a classe **RegistroResumos** com o construtor **RegistroResumos(int numeroDeResumos)**. Esta classe deve ter os métodos:

- **void adiciona(String tema, String resumo)**
- **String[] pegaResumos()**
- **String imprimeResumos()**
- **int conta()**
- **boolean temResumo(String tema)**

5. Bônus!

Ao terminar as atividades propostas anteriormente, você pode tentar completar as tarefas descritas a seguir. **A classe Coisa deve continuar funcionando e não deve ser alterada**! Crie uma classe, CoisaBonus com um main que exercita as funcionalidades implementadas.

5.1 - Mais Notas na Disciplina

Na classe Disciplina, crie um construtor adicional que recebe também o número de notas da disciplina. Além desse construtor, crie outro construtor que recebe o nome da disciplina, número de notas e um array de inteiros com os pesos de cada uma das notas, para o cálculo de uma média ponderada.

Por exemplo, uma disciplina de 2 notas e pesos [6, 4] tem sua média calculada como (6 * notas[0] + 4 * notas[1]) / 10. Caso o array de pesos não seja passado, considere cada nota com mesmo peso (ou seja, a média é uma média aritmética: (notas[0] + notas[1]) / 2).

5.2 - Busca de Resumos de Estudo

Por vezes o aluno não lembra qual o tema cadastrado mas lembra de parte do texto de um resumo cadastrado. Assim, para facilitar a vida do usuário, deve ser possível fazer uma busca sobre as anotações cadastradas.

A busca retorna uma lista de strings com os temas onde a palavra buscada faz parte da anotação. A busca deve ignorar se a chave de pesquisa está em maiúscula ou minúscula. Por exemplo, no nosso sistema, ao buscar por "UM", ambos os temas Classes e Tipos devem ser retornados na lista de strings de resposta. O array retornado deve ser apresentado em ordem alfabética ou em ordem lexicográfica.

Ou seja, você deve implementar, além dos métodos básicos de **RegistroResumos**, o método:

- **String[] busca(String chaveDeBusca)**

5.3 - Comentário sobre a rotina de descanso

Além da rotina de descanso, o aluno pode cadastrar um emoji que descreva sua última sensação em geral. Por exemplo, há certos dias que ele pode estar "**:(**", "*_*", ":o)","<(^_^<)", "¯_(ツ)_/¯" ...

O emoji, não tem relação com o cálculo da rotina de descanso do aluno. Ele expressa o sentimento da aluna no momento, estando ela cansada ou não.

Caso seja registrado um emoji, ele deve ser retornado como adicional ao estado de saúde geral do aluno (ex.: descansado - *_*). No entanto, caso o estado geral do cansaço do aluno mude isto é, haja uma alteração do número de horas de descanso ou de semanas que leve a uma mudança de estado do cansaço do aluno, o último emoji registrado deve ser removido.

Assim, além da alteração no método **getStatusGeral()**, é preciso adicionar um novo método:

- **void definirEmoji(String valor)**

5.4 - Linha de Comando

Faça com que seu sistema receba comandos da entrada e que traduzam tais comandos em ações nos 4 objetos desse sistema (exemplo: "TEMPOONLINE HORAS 2" para adicionar 2 horas ao objeto de registro online… ou "RESUMO Objetos Entidades em memória para carregar dados e com tipo

associado" para adicionar um resumo). Não é necessário criar um "Menu" para o seu sistema, mas você pode fazê-lo. Crie uma classe CoisaCLI que terá essa interação com o usuário, mantendo Coisa e CoisaBonus, intactas.

Entrega

No Eclipse, faça um projeto que contenha as classes necessárias para fazer funcionar o programa passado inicialmente. É opcional fazer as funcionalidades bônus.

É importante que todo o código esteja devidamente documentado com javadoc (entretanto, não é necessário documentar métodos privados nesse sistema, nem mesmo métodos acessadores ou modificadores). Isto significa que classes, atributos e métodos devem ser descritos usando o formato de Javadoc apresentado.

Para a entrega, siga o tutorial de uso do GitHub Classroom.

Seu programa será avaliado pela corretude e, principalmente, pelo DESIGN do sistema. É importante:

- Usar nomes adequados de variáveis, classes, métodos e parâmetros.
- Fazer um design simples, legível e que funciona. É importante saber, apenas olhando o nome das classes e o nome dos métodos existentes, identificar quem faz o que no código.