Sumário

[**Aulas sobre a classe CONJUNTO - DS201 – Programação II** 4](#_Toc87515400)

[**1.** **Pasta: 2021-10-04.S** 4](#_Toc87515401)

[**AULA 1 – Dia 04 de Outubro 2021: Dúvidas sobre métodos obrigatórios e introdução a classe Conjunto** 5](#_Toc87515402)

[**1.2.Lista de Arquivos usados na aula 1** 9](#_Toc87515403)

[**1.2.1. Arquivo:Main.java** 9](#_Toc87515404)

[**1.2.2.Arquivo: Conjunto.java** 12](#_Toc87515405)

[**2.** **Pasta: Aula 06-10-2021\_DS201** 12](#_Toc87515406)

[**AULA 2 – Aula 06-10-2021\_Classes genérica, conjunto e data** 12](#_Toc87515407)

[**2.1.Lista de arquivos usados na aula (Aula 2)** 13](#_Toc87515408)

[**2.1.1Arquivo: Conjunto.java** 13](#_Toc87515409)

[**2.1.2.Arquivo: Main.Java** 13](#_Toc87515410)

[**2.1.3.Arquivo: Data.Java** 15](#_Toc87515411)

[**3.** **Pasta: Aula 07-10-2021\_DS201** 37](#_Toc87515412)

[**AULA 3 : Dia 07 de Outubro de 2021 – Continuação da classe Conjunto (método remova)** 37](#_Toc87515413)

[**3.1. Lista de Arquivos da pasta Aula 07-10-2021** 37](#_Toc87515414)

[**3.1.1.Arquivo: Conjunto.java** 37](#_Toc87515415)

[**3.1.2.Arquivo: Data.java** 43](#_Toc87515416)

[**3.1.3.Arquivo: Main.java** 61](#_Toc87515417)

[**4.** **Pasta: 2021-10-13.S** 66](#_Toc87515418)

[**4.1.** **Lista de Arquivos da pasta (sem transcrição de aulas)** 66](#_Toc87515419)

[**4.1.1.Arquivo: Aluno.java** 66](#_Toc87515420)

[**4.1.2.Arquivo: Conjunto.java** 67](#_Toc87515421)

[**4.1.3.Arquivo: Data.java** 74](#_Toc87515422)

[**4.1.4.Arquivo: Main.java** 92](#_Toc87515423)

[**5.** **Pasta: Aula 14-10-2021\_DS201** 94](#_Toc87515424)

[**AULA 5 – Reflect para chamar clone (classe Conjunto) – Aula 14 Outubro 2021** 94](#_Toc87515425)

[**5.1. Lista de Arquivos usados na aula (aula 5)** 95](#_Toc87515426)

[**5.1.1.Arquivo: Programa2.java** 95](#_Toc87515427)

[**5.1.2.Arquivo: Programa 3.java** 98](#_Toc87515428)

[**5.1.4. Versão final dos métodos que usam o clone atualizados conforme abaixo, para a classe Conjunto <X>** 101](#_Toc87515429)

[**5.1.5.Arquivo: Programa4.java** 104](#_Toc87515430)

[**5.1.6.Arquivo: Conjunto.java (VERSÃO FINAL)** 105](#_Toc87515431)

[**5.1.7.Arquivo: Programa1.java** 112](#_Toc87515432)

[\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ **6. Pasta: 18-10-2021\_DS201** 113](#_Toc87515433)

[**AULA 6 – Sockets (dia 18 Outubro 2021)** 113](#_Toc87515434)

[**6.1.Lista de Arquivos usados na aula 6.** 116](#_Toc87515435)

[**6.1.1.Arquivo: 1ª versão do Reativo.java** 116](#_Toc87515436)

[**6.1.2.ARQUIVO: 2ª versão do Reativo.Java** 120](#_Toc87515437)

[**6.1.3.ARQUIVO: 2ª versão do proativo.Java** 122](#_Toc87515438)

[**6.1.4.ARQUIVO: REATIVO DA DATA.JAVA** 125](#_Toc87515439)

[**6.1.5.Arquivo: PROATIVO DO DATA.JAVA** 126](#_Toc87515440)

[**6.1.6.Arquivo: DATA COM SERIALIZABLE.JAVA** 129](#_Toc87515441)

[**6.1.7.Arquivo: 1ª versão de Proativo.java** 135](#_Toc87515442)

[***6.1.8.Pasta 1)Só texto*** 136](#_Toc87515443)

[**6.1.8.1.Arquivo: Proativo.java** 136](#_Toc87515444)

[**6.1.8.2.Arquivo: Reativo.java** 138](#_Toc87515445)

[***6.1.9.Pasta 2) Objetos*** 139](#_Toc87515446)

[**6.1.9.1.Arquivo: Proativo.java** 139](#_Toc87515447)

[**6.1.9.2.Arquivo: Reativo.java** 141](#_Toc87515448)

[**6.2.*Pasta 3) Datas*** 143](#_Toc87515449)

[**6.2.1.Arquivo: Data.java** 143](#_Toc87515450)

[**6.2.2.Arquivo: Proativo.java** 149](#_Toc87515451)

[**6.2.3.Arquivo: Reativo.java** 151](#_Toc87515452)

[**6.2.4.Arquivo: Anotações.txt** 153](#_Toc87515453)

[**7. Pasta: Aula 20-10-2021** 155](#_Toc87515454)

[**7.1.Subpasta: 2021-10-20.A** 155](#_Toc87515455)

[**7.1.1.Pasta: 1)ThreadsSEMCompartilhamentoDeED** 155](#_Toc87515456)

[**AULA 7- Thread SEM Compartilhamento De ED - Dia 20 Out 2021** 155](#_Toc87515457)

[**7.2. Lista de Arquivos usados na aula 7** 162](#_Toc87515458)

[**7.2.1.Arquivo: Programa.Java** 162](#_Toc87515459)

[**7.2.2.Arquivo: TarefaDoTipo1.java** 163](#_Toc87515460)

[**7.2.3.Arquivo: TarefaDoTipo2.java** 164](#_Toc87515461)

[**7.2.4.Arquivo: TarefaDoTipo3.java** 165](#_Toc87515462)

[**7.2.5.Arquivo: Teclado.java** 166](#_Toc87515463)

[**7.3.Pasta: 2)ThreadsCompartilhandoED** 172](#_Toc87515464)

[**AULA 7 – Thread Compartilhando ED (aula 20 Out 2021, parte2)** 172](#_Toc87515465)

[**7.4. Lista de Arquivos usados na aula 7, parte 2.** 178](#_Toc87515466)

[**7.4.1.Arquivo: Programa.Java** 178](#_Toc87515467)

[**7.4.2.Arquivo: TarefaDoTipo1.java** 180](#_Toc87515468)

[**7.4.3.Arquivo: TarefaDoTipo2.java** 181](#_Toc87515469)

[**7.4.4.Arquivo: TarefaDoTipo3.java** 182](#_Toc87515470)

[**7.4.5.Arquivo: TarefaDoTipo4.java** 184](#_Toc87515471)

[**7.4.6.Arquivo: Teclado.java** 186](#_Toc87515472)

[**8. Pasta: Aula 21-10-2021** 192](#_Toc87515473)

[**AULA 8- Thread Compartilhando ED com Limites (dia 21 Out 2021) : produtor e consumo** 192](#_Toc87515474)

[**8.1. Lista dos Arquivos usados na aula 8** 196](#_Toc87515475)

[**8.1.1.Arquivo: Programa.java** 196](#_Toc87515476)

[**8.1.2. Arquivo:TarefaDoTipo2.java** 197](#_Toc87515477)

[**8.1.3.Arquivo: TarefaDoTipo4.java** 200](#_Toc87515478)

[**8.1.4.Arquivo: Programa.java** 204](#_Toc87515479)

[**8.1.5.Arquivo: TarefaDoTipo1.java** 206](#_Toc87515480)

[**8.1.6.Arquivo: TarefaDoTIpo2.java** 208](#_Toc87515481)

[**8.1.7.Arquivo: TarefaDoTipo3.java** 209](#_Toc87515482)

[**8.1.8.Arquivo: TarefaDoTipo4.java** 211](#_Toc87515483)

[**8.1.9.Arquivo: Teclado.java** 214](#_Toc87515484)

[**8.2.Atividade: Leia-me.txt** 220](#_Toc87515485)

[**9. Pasta: Aula 27-10-2021** 221](#_Toc87515486)

[**AULA 9 – Vector (dia 27 Out 2021)** 221](#_Toc87515487)

[**9.1.Lista de Arquivos usados na aula 9** 227](#_Toc87515488)

[**9.1.1.Arquivo: Vector.java** 227](#_Toc87515489)

[**9.1.2.Arquivo: Programa.java** 231](#_Toc87515490)

[**10. Pasta: Aula 28-10-2021** 231](#_Toc87515491)

[**AULA 10 – Sincronismo ( Dia 28 Out 2021)** 231](#_Toc87515492)

[**9.1.Lista de Arquivos usados na aula 10** 232](#_Toc87515493)

[**9.1.1.Arquivo: Vector,java** 232](#_Toc87515494)

[**10.1.Lista dos arquivos usados na aula 10** 233](#_Toc87515495)

[**10.1.1.Arquivo: Programa.java** 233](#_Toc87515496)

[**10.2) Pasta: Meu Vector Sincronizado** 250](#_Toc87515497)

[**10.2.1.Arquivo: Programa.java** 250](#_Toc87515498)

[**10.2.2.Arquivo: Vector.java** 251](#_Toc87515499)

[**10.3) Pasta: Threads com compartilhamento de ED (sincronismo externo)** 255](#_Toc87515500)

[**10.3.1.Arquivo: Programa.java** 255](#_Toc87515501)

[**10.3.2.Arquivo: TarefaDoTipo1.java** 257](#_Toc87515502)

[**10.3.3.Arquivo: TarefaDoTipo2.java** 258](#_Toc87515503)

[**10.3.4.Arquivo: TarefaDoTipo3.java** 260](#_Toc87515504)

[**10.3.5.Arquivo: TarefaDoTipo4.java** 263](#_Toc87515505)

[**10.3.6.Arquivo: Teclado.java** 265](#_Toc87515506)

**Aulas sobre a classe CONJUNTO - DS201 – Programação II**

1. **Pasta: 2021-10-04.S**

**Arquivo: Aula 04-10-2021\_DS201\_.mp4**

**AULA 1 – Dia 04 de Outubro 2021: Dúvidas sobre métodos obrigatórios e introdução a classe Conjunto**

O **método hashcode ele calcula um número natural de um jeito esquisito, não busca nem encontra nada**. A utilidade de calcular o hashcode é para buscas eficientes; se tem o objeto guardado em estruturas já prontas: hashmap e hashtable. Exemplo: guardar objetos da classe Data; pode encontrar os objetos mais rápido por conta da estrutura de armazenamento. Essa estrutura ultra-rápida só vai funcionar se tiver o hashcode. Hashcode é usado em situações mais críticas: aplicação médica se demorar o paciente morre. As estruturas ultra-rapidas são desperdiçadoras de memória. O hashcode faz e não usa nunca, ele viabiliza estruturas ultra-rápidas.

O **equals vê se dois objetos são iguais fazendo a comparação dentro do seu programa**.

**Para o hashcode , se o valor for um array, cada posição do array é uma variável separada e vai calcular para cada posição como se fosse separado no for**. Para calcular 1 posição do vetor, tem que pegar a variável, multiplicar pelo número primo e somar o cálculo praquela posição. Depois na segunda, vai pegar o cálculo da primeira; multiplicar pelo número primo novamente, e somar o cálculo da segunda. Quando pegar a terceira, pega a variável que já está incluindo a primeira e a segunda; multiplicar pelo número primo, calcular a terceira e somar ali. Quando acabar o vetor inteiro, nessa variável já terá sido calculado o hashcode de cada posição. Existem muitos jeitos de calcular hashcode (é empírico), mas foi apresentado o modo mais simples. Nenhum método obrigatório é para ser muito comprido. O que incomoda é fazer um método estranho que faz, larga lá; será que alguém está usando? O que você não está entendendo as vezes não é para entender. Se você sabe como faz, e acredita que é importante; faz e deixa lá. Com o passar do tempo pode ser que se acostume com os métodos. Esses métodos, cada um segue um padrão. O jeito de fazer é bem padrão; diferente de fazer um programa do zero e pensar numa lógica.

**Esses métodos obrigatórios com exceção do hashcode; eles têm a ver com trazer para as classes que você cria, coisas que todas as outras classes e tipos primitivos possuem.** Mas a sua classe senão fizer aqueles métodos não vai ter. Por exemplo: você tem dois números. Você pode comparar com maior, menor, menor igual, maior igual. O fato de comparar dois números faz parte das linguagens, normal; não precisa fazer nada para comparar.

**E se eu precisar comparar duas datas, dois objetos?** Como eu faço senão tiver o método compareTo? Vamos começar uma classe nova, deixando a classe Data para trás. Nessa classe nova não tem getters nem setters e nem é para ter; pois se colocar estraga o funcionamento. Nem sempre é para ter getters e setters. Se eu precisar comparar dois objetos, imagina a classe Data sem getters. Se eu precisasse comparar dois objetos não tem como: porque não tem getters, os atributos são private e não consigo comparar. Não tenho como saber se são iguais, se uma vem antes da outra; e isso é tão básico com tipos primitivos. Porque não vou conseguir comparar Data? Por isso tem os métodos: equals, compareTo. Para comparar se duas datas são iguais, uso equals; é a salvação para você conseguir fazer coisas que faz com os tipos primitivos. Comparar é coisa básica em um algoritmo. E senão existisse maior, menor para os números não seria complicado? Senão fizer compareTo não tem como saber se uma data é menor que a outra; sem o clone não tem como fazer uma cópia estável (shallow copy e deep copy). O uso da depy copy exige clone e construtor de cópia; a shallow copy pode criar uma cópia instável (mexo no original e a cópia muda junto).

**Imagina que você tem um número que quer mudar, mas no fim do algoritmo ele quer printar do jeito que ele era.** Muda o número do looping, vai mudando e no fim quer printar igual no início. Eu crio uma variável e guarda na variável; o número começa a mudar muitas vezes. Lá no fim do programa que eu preciso pegar o valor original vou na variável e quero que ela esteja igual. E se na hora que precisar pegar o número de cópia ele tiver mudado? Vai dar problema. É o que acontece quando faz uma shallow copy instável. É tão normal fazer cópia e precisar usá-la.

**Os métodos obrigatórios existem para que você consiga fazer coisas básicas/simples que faz com tipos, que existem na vida cotidiana.** Os métodos obrigatórios trazem a simplicidade para a sua classe; se tiver toString pode printar string; o equals e compareTo para comparar; o clone e construtor de cópia para fazerem uma cópia estável que não mude sozinha quando mudar o objeto original. O hashcode viabiliza estruturas ultra-rápidas; pega o valor de cada atributo para fazer as contas no ret.

**No hashcode , se tiver um atributo de qualquer tipo; se você quiser calcular o hashcode usa ele.hashcode, sendo ele um objeto com métodos**. A grande diferença é quando NÃO é objeto, usa a classe wrapper. Se for o tipo booleano usa o new Boolean, se for caractere usa new Character , se for inteiro usa new Integer. Eu crio uma instância da classe Wrapper e essa instância terá hashcode; faz Integer.hashcode. No cálculo usa a classe Wrapper. Mas quando tiver os atributos você guarde como um tipo primitivo, pois como wrappers gasta mais memória; só guardaria direto como Wrapper se precisar usar métodos como: no Character para ver se é maiúsculo, minúsculo. Se somente usa o wrapper na hora de calcular o hashcode, deixa guardado como wrapper acaba gastando mais memória e no resto do programa usa como um tipo primitivo. Então era melhor guardar como um tipo primitivo normal em prol da memória. Você cria, usa e descarta; porque usa só no hashcode.

**Qual a diferença entre os métodos clone e construtor de cópia?** Tanto faz se faz a cópia com o clone ou construtor de cópia, pois os dois fazem cópias que funcionam do mesmo jeito. Faz os dois por questão de compatibilidade nas classes prontas, pois não sabemos qual dos dois a classe vai usar. (45min56s). Na hora da gente usar, usa o que quiser.

**Vamos começar a falar sobre a classe nova; talvez não dê tempo de fazer nada hoje.** Vai explicar a motivação sobre o que será feito. Os parâmetros vimos em métodos. (59min45s). O fato de ter parâmetros em métodos faz com que a gente consiga influenciar alguma coisa no método. Por exemplo: Você tem na classe Tracinhos, tem o método revela, cujo papel é fazer aparecer uma letra. O fato desse método ter parâmetro, vai passar uma letra. Eu escolho a letra que vai aparecer no tracinho; o método não mostra o tempo todo a mesma letra, mostra a letra que eu quero por ter parâmetro.Percebe que eu consigo influenciar no método porque ele tem parâmetro.

**Se você tem o método como o revele, ele faz aparecer uma letra nos tracinhos. Se eu passar o parâmetro de letra ele mostra a letra que eu quero: se o parâmetro for A aparece A.** A letra que eu passar pro revele aparece. O revele tem outro parâmetro que é a posição; se eu quiser que a letra apareça na posição 0, aparece a posição 0. Se quiser que coloque a posição 3 aparece o 3. O papel do revele é fazer aparecer uma letra. Mas eu posso exercer a minha influência nesse método, fazer aparecer a letra que eu quero no lugar que eu quero. Ele não vai aparecer sempre a mesma letra, mostra a letra que eu quero; não mostra sempre a mesma posição, mostra na posição que eu quero. Isso porque o método tem parâmetro para passar a posição e letra que eu quero.

**O fato de um método ter parâmetro faz com que eu possa ao usar o método, influenciar como ele vai fazer o trabalho dele.** O trabalho dele é mostrar uma letra, mas eu influencio onde ele vai mostrar a letra e qual a letra que ele vai mostrar. Só consigo influenciar o jeito que ele vai funcionar quando tem parâmetro no método. Um método que não tem parâmetro, por exemplo getDia: sempre vai retornar o dia que tiver ali, se chamar 50x sempre vem o mesmo dia por não ter parâmetro. Ao contrário do método revele, que se eu chamar 50x, cada vez pode aparecer uma letra e posição diferentes; pode ter tantas variedades.

**Na aula de hoje vai trazer a novidade de poder passar parâmetro em classe. Eu quero poder ter classe que tem parâmetro; não estamos acostumados a ter parâmetros em classe.** Como vai funcionar isso? Basicamente a ideia vai ser a seguinte. Você vai criar uma classe, e você vai dizer que a sua classe tem um parâmetro x. Então, na classe conjunto temos:

Public class Conjunto <X>

{}

**O jeito de colocar parâmetro em método é colocando parâmetro entre parênteses; em classe coloca entre os sinais de menor e maior.** A classe Conjunto tem o parâmetro x. Esse parâmetro poderia se chamar y, tipo, o nome que você quisesse. Quando você coloca parâmetro em método, você sempre diz o tipo do parâmetro: int, float. Em classe você só coloca o nome e não fala o tipo pois não funciona. Só existe um tipo de parâmetro que posso passar para a classe, e esse tipo sempre é outra classe. Por exemplo:

Public class Main

{

public static void main (String [] args)

{

Try

{

Conjunto <String> c1 = new Conjunto <String> ();

Conjunto <Integer> c2 = new Conjunto <Integer> (6);

Conjunto <Data> c3 = new Conjunto <Data> ();

Conjunto <Conjunto <String>> c4 = new Conjunto <Conjunto <String>> (15);

**Eu vou poder passar como parâmetro a String (ao lado do c1) .** Eu posso passar Integer porque também é classe, assim como Data. Eu posso fazer como parâmetro um conjunto de string porque conjunto é uma classe. Não tem problema eu passar Conjunto <String> para a classe Conjunto. Mas o fato que o parâmetro de uma classe que tem parâmetro sempre será outra classe. Logo, o parâmetro de uma classe sempre será outra classe que eu já criei ou já estava pronta em java. (01h10min14s). Eu posso usar coisa de biblioteca, tal como as classes Wrapper: Integer; classes que eu fiz: Conjunto e Data. Classe pronta: String.

**Se o parâmetro de classe é outra classe não precisa dizer o tipo,** pois o parâmetro de classe sempre será classe. Quando é método, o parâmetro tem tantas possibilidades, se for inteiro tem que falar que é int. Se é real tem que falar que é float. Nos métodos eu preciso dizer o que é que eu quero.

**E para quê eu vou passar o X como parâmetro?** Porque depois eu vou poder criar nessa classe Conjunto métodos. Eu posso criar o método inclua, recebe como parâmetro x da classe X. O fato é que se eu tenho na classe Conjunto um método inclua que aceita receber xizinho da classe xzão. Lá na main para o c1 o X é String. Sendo assim eu vou poder incluir uma String no c1. Então:

Public class Main

{

public static void main (String [] args)

{

Try

{

Conjunto <String> c1 = new Conjunto <String> ();

c1.inclua (“C”);

c1.inclua (“C++”);

c1.inclua (“JAVA”);

c1.inclua (“PHYTON”);

Conjunto <Integer> c2 = new Conjunto <Integer> (6);

Conjunto <Data> c3 = new Conjunto <Data> ();

Conjunto <Conjunto <String>> c4 = new Conjunto <Conjunto <String>> (15);

Se c1 tem String, o X é uma String e o parâmetro x é uma string.

No método inclua da classe Conjunto:

Public class Conjunto <X>

{

public void inclua (X x) throws Exception

{

//...,

}

**Mas na linha c1.inclua (“C”) não posso trocar C por 5 porque 5 é número.** O método inclua do objeto c1 não aceita número e sim string. Porque? Porque está escrito que c1 é da classe Conjunto e o parâmetro é a classe String. Portanto o c1 é String e o método inclua recebe String. Se colocar 5 dará erro de compilação.

**Em compensação no c2 quando foi declarado a classe Conjunto foi parametrizado com Integer então somente aceita declarar inteiros.** Pro c2, X é Integer; o parâmetro do inclua é Integer; quando for usar na main eu tenho que passar um Integer. No inclua do c2 não pode passar String, seria inaceitável. No inclua do c2 posso passar números inteiros.

**O c1 é um conjunto de strings e só aceita strings, de igual modo, o c2 é um conjunto de Integer e só pode aceitar inteiros.** E o c3? O c3 é conjunto de datas, eu posso incluir no c3 várias datas. No c3 posso incluir a data 19/1/1966. Como o c3 é conjunto de Data, pro c3 o X é Data e o parâmetro do inclua tem que ser Data. Portanto, pode colocar no segundo c3.inclua a data de 29/6/1992 e posso até declarar e instanciar o objeto data Natal valendo 25/12/2021 e na linha debaixo fazer c3.inclua (natal). Isso tudo porque c3 é conjunto de data, se colocar string ou número vai dar erro, fala tipo errado. O c3 só aceita Data. O inclua é feito no c3, o c3 foi declarado como data, X é Data e inclua aceita data.

**E para o c4, se eu quiser ver como ficaria o conjunto de conjuntos (o c4) como que ficaria?** Você declara um conjunto de String a, coloca nele amarelo e verde. Declara um conjunto de String b e coloca nele preto, azul e roxo. Declara um conjunto de String c e coloca nele rosa e branco. Declara o conjunto de string d e coloca nele lilás, laranja e cinza. E agora c4 é o conjunto de conjuntos de string! Eu posso colocar no c4 o a, b, c , d. No lugar do a não posso colocar Natal por ser uma data., nem o número 3 por ser número, nem colocar a palavra JAVA por ser string. Só posso colocar conjuntos de string como parâmetro no c4.inclua. O c4 é um conjunto onde X é um conjunto de Strings; o parâmetro do inclua tem que ser conjunto de Strings. Ele só pode receber um objeto da classe conjunto de String contendo nenhum, 3. Tem que ser um conjunto de Strings.

**Por conta da nossa classe Conjunto ter parâmetro, na hora de usá-la eu posso decidir o que ela vai guardar.** A classe Conjunto é conjunto de alguma coisa (X). O parâmetro influenciou no que a minha classe vai guardar, eu resolvo o que vai guardar. Eu posso exercer a minha influência sobre a maneira da minha classe funcionar.

Então:

**1.2.Lista de Arquivos usados na aula 1**

**1.2.1. Arquivo:Main.java**

Public class Main

{

public static void main (String [] args)

{

Try

{

Conjunto <String> c1 = new Conjunto <String> ();

c1.inclua (“C”);

c1.inclua (“C++”);

c1.inclua (“JAVA”);

c1.inclua (“PHYTON”);

Conjunto <Integer> c2 = new Conjunto <Integer> (6);

c2.inclua (2);

c2.inclua (3);

c2.inclua (5);

c2.inclua (7);

Conjunto <Data> c3 = new Conjunto <Data> (8);

c3.inclua (new Data ((byte) 19, (byte) 1, (short) 1966));

c3.inclua (new Data ((byte) 29, (byte) 6, (short) 1992));

Data natal = new Data ((byte) 25, (byte) 12, (short) 2021);

c3.inclua (natal);

Conjunto <String> a = new Conjunto <String> ();

a.inclua (“amarelo”);

a.inclua (“verde”);

Conjunto <String> b = new Conjunto <String> ();

b.inclua (“preto”);

b.inclua (“azul”);

b.inclua (“roxo”);

Conjunto <String> c = new Conjunto <String> ();

c.inclua (“rosa”);

c.inclua (“branco”);

Conjunto <String> d = new Conjunto <String> ();

d.inclua (“lilás”);

d.inclua (“laranja”);

d.inclua (“cinza”);

Conjunto <Conjunto <String>> c4 = new Conjunto <Conjunto <String>> (15);

c4.inclua (a);

c4.inclua (b);

c4.inclua (c);

c4.inclua (d);

}

Catch (Exception erro)

{

System.err.println (erro.getMessage()); //

}

}

}

**O número ou a falta dele que vai dentro de cada declaração de objetos da classe conjunto na main (c1, c2, c3 e c4) ao colocar o parênteses, é porque na classe conjunto eu vou fazer dois construtores:** um construtor sem parâmetro - se eu quiser criar um conjunto e ao criar, pro construtor não passar parâmetro nenhum eu posso- porque o new Conjunto <String> () não é parâmetro para o construtor, o String é parâmetro para a classe e o () é para o construtor. Nós vamos ter um construtor que se eu passar 8 vai ter um conjunto para guardar até 8 coisas. Se eu passar 6, vou poder guardar até 6 coisas; se eu passar 15 ele pode guardar até 15 coisas. Se eu passar nada, ele assume o valor de 10 e posso guardar até 10 coisas. Mas é construtor que ainda vamos fazer.

**É importante entender que a sua classe Conjunto tem o parâmetro X e dentro dessa classe, tem o método inclua que tem como parâmetro o X da classe X e o tipo x da classe X.** Tem que ver o que o inclua aceita a partir da classe X passada como parâmetro. Se mais tarde eu resolver na classe Conjunto fazer o método tem que vai ver se tem alguma coisa no conjunto, o parâmetro também será X; vou procurar se tem alguma coisa no conjunto, mas que tipo de coisa? Algo do tipo x. Se o meu conjunto é String eu incluo string e depois vejo se tem String. Se eu quiser fazer um método getElemento para recuperar o elemento que tem no conjunto, o elemento que eu recuperar o retorno vai ser String. Eu pego um elemento que pode ser 1,2,3 do meu conjunto e seja o elemento que for , o elemento será string. Se o elemento é X a sua resposta será x. Se o elemento for data, retorno será data; se for inteiro o retorno será um inteiro. Isso quer dizer que, se o meu conjunto é um conjunto de string na main, então X é string; o inclua aceita string, o tem procura string, o getElemento procura algo lá na posição i do conjunto e retorna string, o que ele vai achar lá é string porque o conjunto é de string. Vai retornar, seguramente tem que ser uma string. Se eu tiver que fazer um método para remover alguma coisa do conjunto , aquilo que eu vou remover certamente é do tipo x e será uma string.

**Olhando na main, os métodos para c1 será sempre string porque X é String; para c2 onde X é Integer , os métodos serão do tipo integer.** Para o c3 onde X é Data, os métodos serão do tipo data. Para o c4 onde X é conjunto <String>, os métodos são do tipo conjunto String. Portanto, uma vez que eu decido o que eu vou passar como parâmetro para a classe Conjunto (qual é o X) tem um monte de implicações que determinam os parâmetros dos métodos para aquele X que escolhi na main e é aplicado na classe Conjunto.

Logo:

**1.2.2.Arquivo: Conjunto.java**

Public class Conjunto <X>

{

public void inclua (X x) throws Exception

{

//...,

}

public boolean tem (X x) throws Exception

{

//...

}

public X getElemento (int i ) throws Exception

{

//....

}

public void remova (X x) throws Exception

{

//...,

}

A classe que tem parâmetro é chamada de classe genérica; porque eu resolvo o que a classe vai guardar a partir do parâmetro escolhido.

1. **Pasta: Aula 06-10-2021\_DS201**

**Arquivo: Aula 06-10-2021\_DS201\_.mp4**

**AULA 2 – Aula 06-10-2021\_Classes genérica, conjunto e data**

**Na aula passada falamos sobre classes genéricas.**

**2.1.Lista de arquivos usados na aula (Aula 2)**

**2.1.1Arquivo: Conjunto.java**

Public class Conjunto <X>

{

public void inclua (X x) throws Exception

{

//...,

}

public boolean tem (X x) throws Exception

{

//...

}

public X getElemento (int i ) throws Exception

{

//....

}

public void remova (X x) throws Exception

{

//...,

}

-------------------------------

**2.1.2.Arquivo: Main.Java**

Public class Main

{

public static void main (String [] args)

{

Try

{

Conjunto <String> c1 = new Conjunto <String> ();

c1.inclua (“C”);

c1.inclua (“C++”);

c1.inclua (“JAVA”);

c1.inclua (“PHYTON”);

Conjunto <Integer> c2 = new Conjunto <Integer> (6);

c2.inclua (2);

c2.inclua (3);

c2.inclua (5);

c2.inclua (7);

Conjunto <Data> c3 = new Conjunto <Data> (8);

c3.inclua (new Data ((byte) 19, (byte) 1, (short) 1966));

c3.inclua (new Data ((byte) 29, (byte) 6, (short) 1992));

Data natal = new Data ((byte) 25, (byte) 12, (short) 2021);

c3.inclua (natal);

Conjunto <String> a = new Conjunto <String> ();

a.inclua (“amarelo”);

a.inclua (“verde”);

Conjunto <String> b = new Conjunto <String> ();

b.inclua (“preto”);

b.inclua (“azul”);

b.inclua (“roxo”);

Conjunto <String> c = new Conjunto <String> ();

c.inclua (“rosa”);

c.inclua (“branco”);

Conjunto <String> d = new Conjunto <String> ();

d.inclua (“lilás”);

d.inclua (“laranja”);

d.inclua (“cinza”);

Conjunto <Conjunto <String>> c4 = new Conjunto <Conjunto <String>> (15);

c4.inclua (a);

c4.inclua (b);

c4.inclua (c);

c4.inclua (d);

}

Catch (Exception erro)

{

System.err.println (erro.getMessage()); //

}

}

}

----------------

**E tínhamos a classe Data porque a gente estava usando na Main.**

**2.1.3.Arquivo: Data.Java**

public class Data implements Comparable<Data>, Cloneable

{

private byte dia, mes;

private short ano;

private static boolean isBissexto (short ano)

{

if (ano<1583) // vigencia do Calendario Juliano

if (ano%4 == 0)

return true;

else

return false;

// vigencia do Calendario Gregoriano

if (ano%400 == 0)

return true;

if (ano%4==0 && ano%100!=0)

return true;

return false;

}

private static boolean isValida (byte dia, byte mes, short ano)

{

if (dia<1 || dia>31)

return false;

if (mes<1 || mes>12)

return false;

if (ano==0)

return false;

if ((mes==4 || mes==6 || mes==9 || mes==11) && dia>30)

return false;

if (mes==2 && dia>29)

return false;

if (ano==1582 && mes==10 && dia>4 && dia<=14)

return false; // Bula Papal Inter Gravissimas, do Papa Gregório XIII

if (dia>28 && mes==2 && !Data.isBissexto(ano))

return false;

return true;

}

public Data (byte dia, byte mes, short ano) throws Exception

{

if (!Data.isValida(dia,mes,ano))

throw new Exception ("Data invalida");

this.dia = dia;

this.mes = mes;

this.ano = ano;

}

public void setDia (byte dia) throws Exception

{

if (!Data.isValida(dia,this.mes,this.ano))

throw new Exception ("Dia invalido");

this.dia = dia;

}

public void setMes (byte mes) throws Exception

{

if (!Data.isValida(this.dia,mes,this.ano))

throw new Exception ("Mes invalido");

this.mes = mes;

}

public void setAno (short ano) throws Exception

{

if (!Data.isValida(this.dia,this.mes,ano))

throw new Exception ("Ano invalido");

this.ano = ano;

}

public byte getDia ()

{

return this.dia;

}

public byte getMes ()

{

return this.mes;

}

public short getAno ()

{

return this.ano;

}

/\*

// há validações duplicadas

public Data getDiaAnterior ()

{

Data ret=null;

if (this.ano==1582 && this.mes==10 && this.dia==15)

{

try

{

ret = new Data ((byte)4,this.mes,this.ano);

}

catch (Exception erro1)

{} // ignora erro por ser impossível

}

else if (this.ano==1 && this.mes==1 && this.dia==1)

{

try

{

ret = new Data ((byte)31,(byte)12,(short)-1);

}

catch (Exception erro2)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)(this.dia-1),this.mes,this.ano))

{

try

{

ret = new Data ((byte)(this.dia-1),this.mes,this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)31,(byte)(this.mes-1),this.ano))

{

try

{

ret = new Data ((byte)31,(byte)(this.mes-1),this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)30,(byte)(this.mes-1),this.ano))

{

try

{

ret = new Data ((byte)30,(byte)(this.mes-1),this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)29,(byte)(this.mes-1),this.ano))

{

try

{

ret = new Data ((byte)29,(byte)(this.mes-1),this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)28,(byte)(this.mes-1),this.ano))

{

try

{

ret = new Data ((byte)28,(byte)(this.mes-1),this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else

{

try

{

ret = new Data ((byte)31,(byte)12,(short)(this.ano-1));

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

return ret;

}

\*/

public Data getDiaAnterior ()

{

Data ret=null;

if (this.ano==1582 && this.mes==10 && this.dia==15)

{

try

{

ret = new Data ((byte)4,this.mes,this.ano);

}

catch (Exception erro1)

{} // ignora erro por ser impossível

}

else if (this.ano==1 && this.mes==1 && this.dia==1)

{

try

{

ret = new Data ((byte)31,(byte)12,(byte)-1);

}

catch (Exception erro2)

{} // ignora erro por ser impossível

}

else

{

try

{

ret = new Data ((byte)(this.dia-1),this.mes,this.ano);

}

catch (Exception erro3)

{

try

{

ret = new Data ((byte)31,(byte)(this.mes-1),this.ano);

}

catch (Exception erro4)

{

try

{

ret = new Data ((byte)30,(byte)(this.mes-1),this.ano);

}

catch (Exception erro5)

{

try

{

ret = new Data ((byte)29,(byte)(this.mes-1),this.ano);

}

catch (Exception erro6)

{

try

{

ret = new Data ((byte)28,(byte)(this.mes-1),this.ano);

}

catch (Exception erro7)

{

try

{

ret = new Data ((byte)31,(byte)12,(byte)(this.ano-1));

}

catch (Exception erro8)

{} // ignora erro por ser impossível

}

}

}

}

}

}

return ret;

}

//

public Data ha (int qtdDias) throws Exception

{

Data referencia = (Data)this.clone();

referencia.retroceda(qtdDias);

return referencia;

}

/\*

// há validações duplicadas

public Data getDiaSeguinte ()

{

Data ret=null;

if (this.ano==1582 && this.mes==10 && this.dia==4)

{

try

{

ret = new Data ((byte)15,this.mes,this.ano);

}

catch (Exception erro1)

{} // ignora erro por ser impossível

}

else if (this.ano==-1 && this.mes==12 && this.dia==31)

{

try

{

ret = new Data ((byte)1,(byte)1,(short)1);

}

catch (Exception erro2)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)(this.dia+1),this.mes,this.ano))

{

try

{

ret = new Data ((byte)(this.dia+1),this.mes,this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)1,(byte)(this.mes+1),this.ano))

{

try

{

ret = new Data ((byte)1,(byte)(this.mes+1),this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else

{

try

{

ret = new Data ((byte)1,(byte)1,(short)(this.ano+1));

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

return ret;

}

\*/

public Data getDiaSeguinte ()

{

Data ret=null;

if (this.ano==1582 && this.mes==10 && this.dia==4)

{

try

{

ret = new Data ((byte)15,this.mes,this.ano);

}

catch (Exception erro1)

{} // ignora erro por ser impossível

}

else if (this.ano==-1 && this.mes==12 && this.dia==31)

{

try

{

ret = new Data ((byte)1,(byte)1,(byte)1);

}

catch (Exception erro2)

{} // ignora erro por ser impossível

}

else

{

try

{

ret = new Data ((byte)(this.dia+1),this.mes,this.ano);

}

catch (Exception erro3)

{

try

{

ret = new Data ((byte)1,(byte)(this.mes+1),this.ano);

}

catch (Exception erro4)

{

try

{

ret = new Data ((byte)1,(byte)1,(byte)(this.ano+1));

}

catch (Exception erro5)

{} // ignora erro por ser impossível

}

}

}

return ret;

}

//

public Data daquiA (int qtdDias) throws Exception

{

Data referencia = new Data (this); // usando construtor de cópia

referencia.avance(qtdDias);

return referencia;

}

/\*

public void tranformeSeNoDiaAnterior ()

{

if (this.ano==1582 && this.mes==10 && this.dia==15)

this.dia=4;

else if (this.ano==1 && this.mes==1 && this.dia==1)

{

this.dia=31;

this.mes=12;

this.ano=-1;

}

else if (Data.isValida((byte)(this.dia-1),this.mes,this.ano))

this.dia--;

else if (Data.isValida((byte)31,(byte)(this.mes-1),this.ano))

{

this.dia=31;

this.mes--;

}

else if (Data.isValida((byte)30,(byte)(this.mes-1),this.ano))

{

this.dia=30;

this.mes--;

}

else if (Data.isValida((byte)29,(byte)(this.mes-1),this.ano))

{

this.dia=29;

this.mes--;

}

else if (Data.isValida((byte)28,(byte)(this.mes-1),this.ano))

{

this.dia=28;

this.mes--;

}

else

{

this.dia=31;

this.mes=12;

this.ano--;

}

}

\*/

public void tranformeSeNoDiaAnterior ()

{

Data anterior = this.getDiaAnterior();

this.dia=anterior.dia;

this.mes=anterior.mes;

this.ano=anterior.ano;

}

//

public void retroceda (int qtdDias) throws Exception

{

if (qtdDias<=0)

throw new Exception ("Quantidade inválida");

for (int i=0; i<qtdDias; i++)

this.tranformeSeNoDiaAnterior();

}

/\*

public void tranformeSeNoDiaSeguinte ()

{

if (this.ano==1582 && this.mes==10 && this.dia==4)

this.dia=15;

else if (this.ano==-1 && this.mes==12 && this.dia==31)

{

this.dia=1;

this.mes=1;

this.ano=1;

}

else if (Data.isValida((byte)(this.dia+1),this.mes,this.ano))

this.dia++;

else if (Data.isValida((byte)1,(byte)(this.mes+1),this.ano))

{

this.dia=1;

this.mes++;

}

else

{

this.dia=1;

this.mes=1;

this.ano++;

}

}

\*/

public void tranformeSeNoDiaSeguinte ()

{

Data seguinte = this.getDiaSeguinte();

this.dia=seguinte.dia;

this.mes=seguinte.mes;

this.ano=seguinte.ano;

}

//

public void avance (int qtdDias) throws Exception

{

if (qtdDias<=0)

throw new Exception ("Quantidade inválida");

for (int i=0; i<qtdDias; i++)

this.tranformeSeNoDiaSeguinte();

}

@Override

public String toString ()

{

return this.dia+"/"+this.mes+"/"+this.ano;

}

@Override

public boolean equals (Object obj)

{

if (this==obj) return true;

if (obj==null) return false;

//if (!(obj instanceof Data)) return false;/

//if (obj.getClass()!=this.getClass()) return false;

if (obj.getClass()!=Data.class) return false;

Data data = (Data)obj;

if (this.dia!=data.dia) return false;

if (this.mes!=data.mes) return false;

if (this.ano!=data.ano) return false;

return true;

}

@Override

public int hashCode ()

{

//int ret=super.hashCode(); // super.hashCode() por NÃO herdar DIRETAMENTE de Object (e SIM EXPLICITAMENTE de outra classe)

int ret=666; // um valor qualquer positivo qualquer por herdar DIRETAMENTE de Object (e NÃO EXPLICITAMENTE de outra classe)

ret = 13\*ret + new Byte (this.dia).hashCode(); // this.dia

ret = 3\*ret + new Byte (this.mes).hashCode(); // this.mes

ret = 7\*ret + new Short (this.ano).hashCode(); // this.ano

if (ret<0) ret = -ret;

return ret;

}

/\*

Estamos neste método desejando comparar this e data;

deveremos retornar um valor negativo, caso this seja

menor que data;

deveremos retornar um valor positivo, caso this seja

maior que data;

deveremos retornar ZERO, caso this seja

IGUAL à data.

\*/

@Override

public int compareTo (Data data)

{

if (this.ano<data.ano) return -1;

if (this.ano>data.ano) return 1;

if (this.mes<data.mes) return -1966;

if (this.mes>data.mes) return 1966;

if (this.dia<data.dia) return -666;

if (this.dia<data.dia) return -666;

if (this.dia>data.dia) return 666;

return 0;

}

public Data (Data modelo) throws Exception

{

if (modelo==null)

throw new Exception ("Modelo ausente");

this.dia = modelo.dia;

this.mes = modelo.mes;

this.ano = modelo.ano;

}

@Override

public Object clone ()

{

Data ret=null;

try

{

ret = new Data (this);

}

catch (Exception erro)

{} // ignorando Exception, pois sei que não ocorrera

return ret;

}

}

**Explicou que classe genérica é uma classe que tem parâmetro e posso dar o nome que eu quiser.** Quando tiver uma classe com parâmetro o valor da classe sempre será uma outra classe, conforme a main: conjunto de string, conjunto de integer, conjunto de data, conjunto de conjunto string. A classe String passada como parâmetro é bem importante porque o inclua pro c1 vai incluir string; pro c2 o integer, pro c3 a data, pro c4 o conjunto de string. A versatilidade está na classe Conjunto <X> pois se X for String, os métodos inclua recebe string, tem retorna string, o getElemento retorna string o remova recebe string. Muitos métodos vão ser afetados pelo fator de ter fornecido X como string, integer,data ou conjunto string ou o que eu for fornecer. (11min51s).

**A gente tem várias lacunas a serem completadas na classe Conjunto <X> para fazer essas classes funcionarem.** A gente precisa declarar atributos para a classe Conjunto. Precisa pensar quais serão os atributos, declarar local para guardar os elementos que fazem parte do meu conjunto.

**Quando chamo o método inclua, ele vai incluir o que eu mandei incluir em algum lugar. Vou ter que declarar algum lugar para que o método inclua guarde as coisas que eu inclui para procurar alguma coisa quando o método tem tiver que procurar.** Eu vou ter que definir um lugar para guardar as coisas que fazem parte do meu conjunto para depois recuperar elementos, remover elementos. É muito importante ter um lugar para guardar os dados que vão formar o nosso vetor.

**Nós vamos definir o seguinte: os atributos elem e qtd.** Esses atributos vão servir para gente poder saber a quantidade de elementos que efetivamente tem no meu conjunto, inicialmente a quantidade é zero. Quando declaro private Object[] elem começam os problemas **,** porque quando vai programar sempre tem problemas a serem contornados. A gente gostaria de declarar um atributo chamado elem, e poder dizer que esse atributo elem será um vetor de coisas do tipo X. Mas na linguagem java, ela não lida muito bem com a declaração de vetores que tem tipo indeterminado como é o caso do tipo x, que está em aberto, não sabemos o que é e posteriormente será fornecido. Ele é um tipo por enquanto indeterminado. (15min42s).

**O Java não lida bem com vetores de tipo indeterminado; reparem que a linguagem Java não me dá problema quando declaro parâmetros nos métodos como (X x).** O X x dos métodos é porque na classe Conjunto tem o parâmetro <X> que é uma classe (X é uma classe) . Para o método inclua o parâmetro vai ser xzinho (ou x) e o tipo dele será a classe xzão (ou X). (17min36s). A linguagem java não vê problema em declarar parâmetro cujo tipo seja uma classe X; a linguagem java não tem problema nenhum em permitir me declarar retornos nos métodos cujo tipo seja uma classe indeterminada X. A linguagem java também não teria nenhum problema em me deixar declarar atributos senão fosse vetor (por exemplo, não teve problema em declarar private X elem). O problema reside mesmo em declarar vetor de tipo indeterminado (como private X[] elem). Isso não é um erro querer um vetor de coisas do tipo X, seria normal, mas a linguagem java não deixa.

**O grande problema da linguagem java está na declaração de vetores de tipos indeterminados. Como vamos contornar esse problema?** Declararemos esse vetor como sendo um vetor de Object : private Object [] elem; o Object vimos em clone, equals. O Object é uma classe da qual todas as classes herdam; todas as classes são consideradas Object cada uma com uma pequena variação. É como se tivéssemos a classe cachorro e da classe cachorro herda a classe pitbull. Pitbull é considerado cachorro porque herda da classe cachorro. Herdando da classe cachorro, poderíamos ter uma classe chamada Poodle e a classe Poodle também seria considerada cachorro porque poodle herda de cachorro, poodle é cachorro. Logo, sempre que acontece uma herança, a classe que herda é considerada como sendo a classe que herdou com alguma adaptação; e assim sendo, como todas as classes herdam de Object, todas as classes são consideradas Object.

**Por isso se eu declaro um vetor de Object (como private Object [] elem) , nesse vetor vou poder guardar qualquer coisa. Pode guardar objetos da classe Data?** Sim, porque Data herda de Object então Data é Object. Posso guardar objetos das classes String, Integer . Qualquer classe é considerada Object **,** portanto nesse vetor posso guardar o que eu quiser, incluindo objetos da classe X (classe indeterminada, misteriosa). Esse vetor até permitiria objetos de qualquer classe, mas não vai acontecer porque ele é private sem acesso de fora da classe; o método inclua é público e ele vai guardar coisas do vetor mas só recebe coisas do tipo, coisas que o inclua guardar. O atributo elem é private e não pode acessar diretamente. No fim das coisas eu vou conseguir o que queria, guardar coisas do tipo X no vetor elem através do método inclua. Eu vou ter que programar o inclua nos três pontinhos eu vou pegar o x e guardar no vetor elem: vou usar o vetor elem dentro do método citado. (30min06s).

**Nós temos que começar a dimensionar o vetor elem; nós vamos fazer um construtor como (após os atributos e antes do método inclua):**

Public Conjunto (int capInicial) throws Exception // 1º construtor

{

If (capInicial <=0)

throw new Exception (“Capacidade inválida”);

// this.elem – new x [capInicial];

this.elem = new Object [capInicial];

}

**Esse construtor vai receber um número inteiro que vai ser capacidadeInicial (capInicial) , o tamanho do vetor e como pode ter problemas;** eu verifico se o que foi passado como parâmetro não for número negativo, pois alguém já viu vetor com tamanho de 0, -5, -15? Não. Então se por acaso o usuário chamar o método e passar valor negativo, eu vou perceber no if e lançar exceção e não vão conseguir. Senão tiver exceção, eu adoraria instanciar como new X, mas java não aceita vetores do tipo X. Então, vou instanciar como new Object [capInicial].

**Vou fazer um segundo construtor, posso ter quantos eu quiser. A gente costumava ter 2: 1 construtor e um de cópia**. Mas posso ter 3, na classe Conjunto terá 3 construtores contando com o de cópia. Nesse segundo construtor, ele não tem parâmetros sendo considerado como construtor padrão. O segundo construtor fica abaixo do 1º construtor. Logo:

public Conjunto () // 2º construtor é o construtor padrão (construtor sem parâmetros)

{

//this.elem = new X [10];

this.elem = new Object [10];

}

**O construtor que tem parâmetro (como o 1º construtor) a gente chama de nada.**

**Tem o construtor de cópia, sendo como os outros. O que faz um construtor de cópia ser um construtor de cópia é o fato dele ter como parâmetro :** um parâmetro só e é um parâmetro da própria classe. Exemplo: estou na classe Conjunto meu parâmetro é Conjunto. Logo (após o método remova):

public Conjunto (Conjunto <X> modelo) throws Exception // 3º construtor é o construtor de cópia (construtor com 1 parâmetro)

{

If (modelo == null)

throws new Exception (“Modelo ausente”);

//....

}

**Portanto o construtor de cópia é aquele que tem um único parâmetro e o tipo é a própria classe.** (35min45s). Outros construtores não tem nome (1º construtor). O construtor padrão (2º construtor) vai instanciar o vetor , só que ao invés de instanciar com a capacidade que a anta fornecer no capInicial, ela coloca o parâmetro e com base nele vai ser esse o tamanho do vetor. No primeiro construtor, a anta fornece o tamanho. No construtor a anta não escolhe nada porque o construtor padrão já determinada dentro do seu método que o tamanho do vetor será dez. (Podia ser qualquer tamanho, mas eu que defini que o tamanho padrão é dez, quando alguém criou o conjunto e não falou o tamanho eu digo que é dez). Exemplo: na Main instanciei o conjunto c1 e não falei o tamanho, então ele será 10. No outro exemplo, eu instanciei c2 e falei o tamanho 6, então o vetor de c2 terá o tamanho 6; assim como em c3 será 8 e em c4 será 10.

**Vamos pensar nos outros métodos a serem completados. Vamos fazer o método chamado ondeEsta , que será muito útil mas será esquisito.** Ele é um método privado, antas não usam esse método. Esse método eu fiz para me ajudar a fazer outros métodos dentro da classe. Sempre que você faz um método que é só para você usar você põe private. Reparem que se o método é público, parâmetros que a anta fornece eu valido e lanço exceção. Se o método é private, como as antas não conseguem passar; eu nem valido nada porque sei que não vai dar erro, eu confio no meu código.

**Esse método ondeEsta vai retornar um vetor de object. Ele é um método esquisito; método público é para anta usar e ela não consegue usar nem os que são simples, quanto mais os esquisitos.** Se o método for private, não tem problema fazer métodos esquisitos e você não tem problema com isso. Esse método serve para descobrir se o x está armazenado em this.elem, que é onde eu armazeno os elementos do meu conjunto. Estando, devo informar em que posição está. Não estando, devo informar em que posição posso incluir caso queira incluir. (45min24s).

**Eu tenho se x está armazenado no vetor, uso o for , começo na posição i no vetor em 0, vou aumentando a posição até chegar na quantidade de coisas guardadas no vetor.** Eu vejo se x é igual ao que tem armazenado no vetor elem na posição i (linha do equals).O i começou com 0 e depois foi para 1,2,3,4 até chegar na quantidade. Eu olho cada posição desse vetor e vendo se é igual ao x. Se eu entrar no if equals é porque achei o x. Quando eu acho, declaro um vetor de Object chamado ret, e eu inicializo esse vetor com um vetor de 2 elementos: true (é verdade que eu achei), i (é a posição onde eu achei o x) e retornei esse vetor ret. Tenho o atributo elem(onde guardo todos os elementos do meu atributo) e no método ondeEsta tenho o vetor ret (crio o vetor ret só para retornar dentro do método que eu achei o x e a posição dele). Toda vez que faço o método e escrevo boolean ele só pode retornar uma coisa: verdadeiro ou falso, como no método tem. Mas no método ondeEsta quero retornar duas coisas: true quando achar e a posição que achei o x. Para isso, criei um vetor, coloquei as coisas lá dentro e retornei o vetor. O true vai ser uma caixinha do vetor e a posição como outra caixinha do vetor. O vetor vai ter duas caixinhas: o 0 terá o true (eu achei) e a segunda caixinha 1 (guarda i que é a posição onde eu achei o x).

Fazer Object[] ret = {true,i}; é o mesmo que (de modo mais longo):

Object [] ret = new Object [2]; // object vetor ret é igual a new object vetor de tamanho2.

ret (0) = true; // na posição 0 é que eu achei o que eu queria;

ret (1) = i; // na posição 1 é onde eu achei o x;

return ret; // retorno o vetor que acabei de criar.

{true, i} cria o vetor mais rápido igual a new.

**Só que Object [] ret = new Object [2]; faz com que eu crie um vetor todo vazio**. Diferente de Object[] ret = {true,i}; que cria vetor com2 posições (0 e 1) e já põe true na posição 0 e i na posição1 . Ou seja, ele criou o vetor e já preencheu as posições.

**Se esse for executar de cabo a rabo e não achar o x no vetor (de 0 a qtd), ele não entrou no if nenhuma vez.** Se isso acontecer ele não retornou nada por não ter entrado no if. Então ele vai criar um vetor ret com falso e posição qtd. e após retornar o ret com esses dados porque é falso que eu achei, eu não achei. Não estando, eu tenho que informar a posição que devo incluir que é qtd; porque se qtd for 10, é porque o meu vetor tem coisa guardada de 0 a 9, logo, onde eu posso incluir? Na 10 porque até a 9 está cheio de coisa. Onde tem o true e i tem uma chavinha, que serve para declarar um vetor e inicializar.

**Quando declaro vetor como int [] vet = new int [100]; eu tenho um vetor de tamanho 100 e está tudo vazio porque não preenchi o vetor.** Agora, se eu declarar como: int [] vet2 = {2,3,5,7,11,13,17}; o meu vetor2 terá tamanho 7, vai ter posições de 0 a 6. O 2 está na posição 0, o 3 na posição 1, o 5 na posição 2 e afins. As chaves não apenas criam o vetor como preenche as caixinhas do vetor, ou seja, cria vetor com conteúdo. Se o seu vetor tem 7 itens dá para preencher com a chavinha, mas valores altos como 100 ou 1000 não tem como usar a chavinha. Quando o vetor for muito grande, é melhor criar com new dizendo o tamanho e depois usar um for para preencher os valores nas posições. A quantidade é uma posição vazia: se a quantidade for 100, está preenchido até 99 e a posição 100 estará livre. Se qtd for 5, vai estar preenchido até 4 e 5 estará livre. A posição indicada por qtd sempre está livre, qtd-1 está ocupada por ser a última posição que tenho coisa guardada. É meio parecido com a posicaodaIezimaOcorrencia. O método ondeEsta é private que eu fiz para me ajudar nos outros métodos da classe. Esse método vai ser útil no método inclua porque eu vou procurar o que eu quero incluir, se já existir lanço exceção para evitar elementos repetidos. Senão existe vou incluir. Na hora de remover vai usar o método ondeEsta para procurar o que quer remover, e na hora de achar vai lá e remove. O método ondeEsta é de apoio aos outros métodos.

Logo:

//depois do construtor padrão e antes do método inclua:

private Object[] ondeEsta (X x)

{

for (int i=0; i<this.qtd; i++)

if (x.equals(this.elem[i]))

{

Object[] ret = {true,i};

return ret;

}

Object[] ret = {false,this.qtd};

return ret;

}

**Vamos fazer o método inclua, que fica após o método ondeEsta (na verdade, o método inclua fica depois do redimensionese, mas o método redimensione-se ainda não foi explicado).** Como o método inclua é publico eu tenho que validar, vendo se o parâmetro x é nulo. A anta manda incluir só faltava ela não me fornecer o que tem que incluir; eu dou exceção de elemento ausente. Agora é hora de usar o método auxiliar:

**Declaro um vetor de object chamado onde, chamo o método auxiliar ondeEsta para procurar x (passo x como parâmetro).** Você sabe que na posição 0 está um boolean que guardo na variável achou (passei 0 como parâmetro do vetor onde). Você sabe que o que está na posição 1 é um integer , guardo o integer na variável posição.

**Na variável achou vai estar verdadeiro ou falso que peguei lá da posição zero. E na variável posição vai ter a posição 1 de achou caso achou seja true.** Na variável posição se o achou der falso eu vou incluir na posição de onde. Logo, estou com todas as informações nas 3 linhas do vetor onde: já procurei o x no onde, vejo se encontrei no achou e já sei qual a posição de x na variável posição. Se eu precisar incluir porque achou der falso, vou incluir na variável posição, senão mostrar onde está a posição a partir da caixinha 1.

Logo:

public void inclua (X x) throws Exception

{

if (x==null)

throw new Exception ("Elemento ausente");

Object[] onde = this.ondeEsta(x);

boolean achou = (Boolean)onde[0];

int posicao = (Integer)onde[1];

if (achou)

throw new Exception ("Elemento repetido");

if (this.qtd==this.elem.length)

this.redimensioneSe (2\*this.elem.length);

this.elem[posicao] = x;

this.qtd++;

}

**Sempre que a gente usar o método, o método que usa outro método vai precisar ser coerente. Se o método sempre retorna na posição 0 um boolean eu tenho que usar no método atual o 0 como boolean.** Eu não posso dizer que na posição 0 é um boolean e usar um número na mesma posição que dá ruim.O uso tem que combinar com a forma que foi feita. A posição 0 é um boolean e a posição 1 um inteiro i. Se eu achei e quero incluir vai gerar a exceção elemento repetido.Se eu chegar no ponto do programa depois do if é porque o que estou querendo incluir não é repetido (não foi para a exceção) e pode incluir.

**Depois do if (achou) tem os ... para fazer a inclusão que será feita depois. Depois da inclusão feita, eu vou terminar esse método escrevendo this.qtd ++;**

**No começo o meu vetor de elementos tinha a capacidade que disseram que queria que ele tivesse ou tinha 10 senão dissesse nada. Mas estava tudo vazio**. A primeira vez que eu chamei o inclua alguma coisa for incluída (na área dos ...) e o qtd aumenta. Cada vez que inclui o qtd aumenta. Uma hora o qtd já aumentou tanto que o vetor elem vai estar cheio e não cabe mais nada. Eu poderia se eu quisesse fazer algo como:

If (achou)

throw new Exception (“Elemento ausente”);

if (this.qtd == this.elem.length) // serve para detectar se o vetor está cheio, fazendo qtd ser igual ao comprimento do vetor.

throw new exception (“Não cabe mais nada”);

**Se a quantidade de coisas incluídas for igual ao tamanho do vetor, essa situação é onde não cabe mais nada; poderia lançar a exceção não cabe mais nada.** O que eu vou fazer nessa situação? Lançar exceção. Guardar não dá porque está cheio. A segunda possibilidade seria aumentar o tamanho do vetor. E como que faz para aumentar tamanho de vetor? Tem outro método private, para que eu acho que tem que aumentar o vetor por falta de espaço. Chamo o método redimensionese.

private void redimensioneSe (int novaCap)

{

//X [] novo = new X [novaCap];

Object[] novo = new Object [novaCap];

for (int i=0; i<qtd; i++)

novo[i] = this.elem[i];

this.elem = novo;

}

**O método para mim (private) é o redimensionese; e esse método eu passo como paramentro a novacap do vetor que retorna int (novaCap é o objeto para nova capacidade do vetor).** Eu adoraria declarar um vetor do tipo novo como new x [novaCap] mas como java não aceita, faço o seguinte. Declarei o vetor chamado novo da classe Object que é instanciado como Object com essa nova capacidade novaCap. Depois do comando, comando [] novo eu tenho um vetor novo com capacidade nova. Agora faço um for de 0 até quantidade; vou pegando do velho e ponho no novo (novo [i] = this.elem[i]) até que tenha passado tudo do velho para o novo. Desta forma, o velho vira o novo. (como this.elem= novo). Não fica com 2 vetores porque estão sendo armazenados no mesmo lugar. Como o velho virou o novo, no lugar do velho ficou o novo. O efeito final é como se fosse um redimensionamento, mas não é bem assim, visualmente parece a mesma coisa. Não é o mesmo vetor que aumentou de tamanho é um vetor novo. Não tem como mudar de tamanho o mesmo vetor por isso que se cria um novo; o efeito final é o mesmo. O vetor será passado no int novaCap do meu vetor? Então, no método inclua eu altero o que farei quando chegar que qtd = length:

// método inclua

If (this.qtd == this.elem.length)

this.redimensionese (2\*this.elem.length); // mando redimensionar pro tamanho atual \* 2, então o vetor dobrou o tamanho. Ele vai dobrar toda vez que encher.

this.elem [posição] = x; // agora que cabe eu guardo

this.qtd++;

} // fecha método inclua

**Nunca falta espaço pois está para faltar ele dobra de tamanho.Depois de redimensionar vai caber.** Agora eu guardo. Para ter certeza de que tudo que fez aqui está funcionando, eu vou incluir o toString.

// falar disso aula que vem

@Override

public String toString ()

{

String ret="{";

if (this.qtd>0)

ret += this.elem[0];

for (int i=1; i<this.qtd; i++)

ret += ","+this.elem[i];

ret += "}";

return ret;

}

**Tira ou comenta os métodos em branco para evitar erros de compilação. Para testar se está tudo ok, vamos na Main e coloca um println em todos os cs**. Logo:

public class Main

{

public static void main (String[] args)

{

try

{

Conjunto<String> c1 = new Conjunto<String> ();

c1.inclua("C");

c1.inclua("C++");

c1.inclua("JAVA");

c1.inclua("PYTHON");

System.out.println(c1);

Conjunto<Integer> c2 = new Conjunto<Integer> (6);

c2.inclua(2);

c2.inclua(3);

c2.inclua(5);

c2.inclua(7);

System.out.println(c2);

Conjunto<Data> c3 = new Conjunto<Data> (8);

c3.inclua(new Data ((byte)19,(byte)1,(short)1966));

c3.inclua(new Data ((byte)29,(byte)6,(short)1992));

Data natal = new Data ((byte)25,(byte)12,(short)2021);

c3.inclua(natal);

System.out.println(c3);

Conjunto<String> a = new Conjunto<String> ();

a.inclua("amarelo");

a.inclua("verde");

Conjunto<String> b = new Conjunto<String> ();

b.inclua("preto");

b.inclua("azul");

b.inclua("roxo");

Conjunto<String> c = new Conjunto<String> ();

c.inclua("rosa");

c.inclua("branco");

Conjunto<String> d = new Conjunto<String> ();

d.inclua("lilas");

d.inclua("laranja");

d.inclua("cinza");

Conjunto<Conjunto<String>> c4 = new Conjunto<Conjunto<String>> (15);

c4.inclua(a);

c4.inclua(b);

c4.inclua(c);

c4.inclua(d);

System.out.println(c4);

}

catch (Exception erro)

{

System.err.println (erro.getMessage()); // {{amarelo,verde},{preto,azul,roxo},{rosa,branco},{lilas,laranja,cinza}}

}

}

**Toda vez que eu printo é porque eu tenho/uso a classe String.** O resultado da compilação foi:

(C, C++, JAVA, PHYTON)

(2,3,5,7)

(19/1/1966, 29/6/1992, 25/12/2021)

({amarelo, verde}, {preto,azul,roxo}, {rosa,branco} ,{lilás, laranja, cinza}]

...Program finished whit exit code 0

Please ENTER to exit console.

**Eu criei um conjunto c1 com as linguagens da linha 1 e printou; criei um conjunto com números na linha 2 e printou. Cada casinha do vetor é uma posição no vetor elem.:** uma casinha tem c, c++, 2, 3, {amarelo,verde} é uma posição pois é um conjunto de conjunto -cada casinha tem um conjunto-. As linhas 3 de datas e 4 de conjuntos das cores todos printaram. No c4 os quatro conjuntinhos a,b,c,d foram colocados no conjuntão c4 e eu printei o conjuntão c4.

1. **Pasta: Aula 07-10-2021\_DS201**

**Arquivo: Aula 07-10-2021\_DS201.mp4**

**AULA 3 : Dia 07 de Outubro de 2021 – Continuação da classe Conjunto (método remova)**

**3.1. Lista de Arquivos da pasta Aula 07-10-2021**

**3.1.1.Arquivo: Conjunto.java**

public class Conjunto <X> implements Comparable <Conjunto>, Cloneable

{

private Object[] elem; // private X[] elem;

private int qtd=0;

public Conjunto (int capInicial) throws Exception

{

if (capInicial<=0)

throw new Exception ("Capacidade inválida");

//this.elem = new X [capInicial];

this.elem = new Object [capInicial];

}

public Conjunto () // construtor padrão (construtor sem parâmetros)

{

//this.elem = new X [10];

this.elem = new Object [10];

}

// este método deve descobrir se o x está armazenado em this.elem;

// estando, devo informar em que posição está;

// não estando, devo informar em que posição posso incluir, caso queira incluir

private Object[] ondeEsta (X x)

{

for (int i=0; i<this.qtd; i++)

if (x.equals(this.elem[i]))

{

Object[] ret = {true,i};

return ret;

}

Object[] ret = {false,this.qtd};

return ret;

}

private void redimensioneSe (int novaCap)

{

//X [] novo = new X [novaCap];

Object[] novo = new Object [novaCap];

for (int i=0; i<this.qtd; i++)

novo[i] = this.elem[i];

this.elem = novo;

}

public void inclua (X x) throws Exception

{

if (x==null)

throw new Exception ("Elemento ausente");

Object[] onde = this.ondeEsta(x);

boolean achou = (Boolean)onde[0];

int posicao = (Integer)onde[1];

if (achou)

throw new Exception ("Elemento repetido");

if (this.qtd==this.elem.length)

this.redimensioneSe (2\*this.elem.length);

this.elem[posicao] = x;

this.qtd++;

}

public boolean tem (X x) throws Exception

{

if (x==null)

throw new Exception ("Elemento ausente");

Object[] onde = this.ondeEsta(x);

boolean achou = (Boolean)onde[0];

return achou;

}

public X getElemento (int i) throws Exception

{

if (i<0 || i>=this.qtd)

throw new Exception ("Elemento invalido");

return (X)this.elem[i];

}

public void remova (X x) throws Exception

{

if (x==null)

throw new Exception ("Elemento ausente");

Object[] onde = this.ondeEsta(x);

boolean achou = (Boolean)onde[0];

int posicao = (Integer)onde[1];

if (!achou)

throw new Exception ("Elemento inexistente");

for (int i=posicao; i<this.qtd-1; i++)

this.elem[i] = this.elem[i+1];

this.qtd--;

this.elem[this.qtd]=null;

}

// faça todos os métodos obrigatorios

public Conjunto (Conjunto<X> modelo) throws Exception // construtor de cópia (construtor com um parâmetro, cujo tipo é a própria classe)

{

if (modelo==null)

throw new Exception ("Modelo ausente");

this.elem = elem;

this.qtd = qtd;

}

@Override

public Object clone ()

{

Conjunto ret = null;

try {

ret = new Conjunto(this);

} catch (Exception e) { }

return ret;

}

@Override

public String toString ()

{

String ret="{";

if (this.qtd>0)

ret += this.elem[0];

for (int i=1; i<this.qtd; i++)

ret += ","+this.elem[i];

ret += "}";

return ret;

}

@Override

public boolean equals (Object obj){

if(this == obj)

return true;

if(obj == null)

return false;

if(obj.getClass() != Conjunto.class)

return false;

Conjunto conjunto = (Conjunto)obj;

if(this.elem != conjunto.texto)

return false;

return true;

}

@Override

public int compareTo (Conjunto conjunto)

{

return this.elem.compareTo(conjunto.texto);

}

public int hashCode ()

{

int ret = 31;

ret = ret \* 31 + new Integer(this.elem).hashCode();

ret = ret \* 31 + new Integer (this.qtd).hashCode ();

if(ret < 0)

ret = -ret;

return ret;

}

}

**+++++++++++++++++++++++++++**

**3.1.2.Arquivo: Data.java**

public class Data implements Comparable<Data>, Cloneable

{

private byte dia, mes;

private short ano;

private static boolean isBissexto (short ano)

{

if (ano<1583) // vigencia do Calendario Juliano

if (ano%4 == 0)

return true;

else

return false;

// vigencia do Calendario Gregoriano

if (ano%400 == 0)

return true;

if (ano%4==0 && ano%100!=0)

return true;

return false;

}

private static boolean isValida (byte dia, byte mes, short ano)

{

if (dia<1 || dia>31)

return false;

if (mes<1 || mes>12)

return false;

if (ano==0)

return false;

if ((mes==4 || mes==6 || mes==9 || mes==11) && dia>30)

return false;

if (mes==2 && dia>29)

return false;

if (ano==1582 && mes==10 && dia>4 && dia<=14)

return false; // Bula Papal Inter Gravissimas, do Papa Gregório XIII

if (dia>28 && mes==2 && !Data.isBissexto(ano))

return false;

return true;

}

public Data (byte dia, byte mes, short ano) throws Exception

{

if (!Data.isValida(dia,mes,ano))

throw new Exception ("Data invalida");

this.dia = dia;

this.mes = mes;

this.ano = ano;

}

public void setDia (byte dia) throws Exception

{

if (!Data.isValida(dia,this.mes,this.ano))

throw new Exception ("Dia invalido");

this.dia = dia;

}

public void setMes (byte mes) throws Exception

{

if (!Data.isValida(this.dia,mes,this.ano))

throw new Exception ("Mes invalido");

this.mes = mes;

}

public void setAno (short ano) throws Exception

{

if (!Data.isValida(this.dia,this.mes,ano))

throw new Exception ("Ano invalido");

this.ano = ano;

}

public byte getDia ()

{

return this.dia;

}

public byte getMes ()

{

return this.mes;

}

public short getAno ()

{

return this.ano;

}

/\*

// há validações duplicadas

public Data getDiaAnterior ()

{

Data ret=null;

if (this.ano==1582 && this.mes==10 && this.dia==15)

{

try

{

ret = new Data ((byte)4,this.mes,this.ano);

}

catch (Exception erro1)

{} // ignora erro por ser impossível

}

else if (this.ano==1 && this.mes==1 && this.dia==1)

{

try

{

ret = new Data ((byte)31,(byte)12,(short)-1);

}

catch (Exception erro2)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)(this.dia-1),this.mes,this.ano))

{

try

{

ret = new Data ((byte)(this.dia-1),this.mes,this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)31,(byte)(this.mes-1),this.ano))

{

try

{

ret = new Data ((byte)31,(byte)(this.mes-1),this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)30,(byte)(this.mes-1),this.ano))

{

try

{

ret = new Data ((byte)30,(byte)(this.mes-1),this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)29,(byte)(this.mes-1),this.ano))

{

try

{

ret = new Data ((byte)29,(byte)(this.mes-1),this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)28,(byte)(this.mes-1),this.ano))

{

try

{

ret = new Data ((byte)28,(byte)(this.mes-1),this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else

{

try

{

ret = new Data ((byte)31,(byte)12,(short)(this.ano-1));

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

return ret;

}

\*/

public Data getDiaAnterior ()

{

Data ret=null;

if (this.ano==1582 && this.mes==10 && this.dia==15)

{

try

{

ret = new Data ((byte)4,this.mes,this.ano);

}

catch (Exception erro1)

{} // ignora erro por ser impossível

}

else if (this.ano==1 && this.mes==1 && this.dia==1)

{

try

{

ret = new Data ((byte)31,(byte)12,(byte)-1);

}

catch (Exception erro2)

{} // ignora erro por ser impossível

}

else

{

try

{

ret = new Data ((byte)(this.dia-1),this.mes,this.ano);

}

catch (Exception erro3)

{

try

{

ret = new Data ((byte)31,(byte)(this.mes-1),this.ano);

}

catch (Exception erro4)

{

try

{

ret = new Data ((byte)30,(byte)(this.mes-1),this.ano);

}

catch (Exception erro5)

{

try

{

ret = new Data ((byte)29,(byte)(this.mes-1),this.ano);

}

catch (Exception erro6)

{

try

{

ret = new Data ((byte)28,(byte)(this.mes-1),this.ano);

}

catch (Exception erro7)

{

try

{

ret = new Data ((byte)31,(byte)12,(byte)(this.ano-1));

}

catch (Exception erro8)

{} // ignora erro por ser impossível

}

}

}

}

}

}

return ret;

}

//

public Data ha (int qtdDias) throws Exception

{

Data referencia = (Data)this.clone();

referencia.retroceda(qtdDias);

return referencia;

}

/\*

// há validações duplicadas

public Data getDiaSeguinte ()

{

Data ret=null;

if (this.ano==1582 && this.mes==10 && this.dia==4)

{

try

{

ret = new Data ((byte)15,this.mes,this.ano);

}

catch (Exception erro1)

{} // ignora erro por ser impossível

}

else if (this.ano==-1 && this.mes==12 && this.dia==31)

{

try

{

ret = new Data ((byte)1,(byte)1,(short)1);

}

catch (Exception erro2)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)(this.dia+1),this.mes,this.ano))

{

try

{

ret = new Data ((byte)(this.dia+1),this.mes,this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)1,(byte)(this.mes+1),this.ano))

{

try

{

ret = new Data ((byte)1,(byte)(this.mes+1),this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else

{

try

{

ret = new Data ((byte)1,(byte)1,(short)(this.ano+1));

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

return ret;

}

\*/

public Data getDiaSeguinte ()

{

Data ret=null;

if (this.ano==1582 && this.mes==10 && this.dia==4)

{

try

{

ret = new Data ((byte)15,this.mes,this.ano);

}

catch (Exception erro1)

{} // ignora erro por ser impossível

}

else if (this.ano==-1 && this.mes==12 && this.dia==31)

{

try

{

ret = new Data ((byte)1,(byte)1,(byte)1);

}

catch (Exception erro2)

{} // ignora erro por ser impossível

}

else

{

try

{

ret = new Data ((byte)(this.dia+1),this.mes,this.ano);

}

catch (Exception erro3)

{

try

{

ret = new Data ((byte)1,(byte)(this.mes+1),this.ano);

}

catch (Exception erro4)

{

try

{

ret = new Data ((byte)1,(byte)1,(byte)(this.ano+1));

}

catch (Exception erro5)

{} // ignora erro por ser impossível

}

}

}

return ret;

}

//

public Data daquiA (int qtdDias) throws Exception

{

Data referencia = new Data (this); // usando construtor de cópia

referencia.avance(qtdDias);

return referencia;

}

/\*

public void tranformeSeNoDiaAnterior ()

{

if (this.ano==1582 && this.mes==10 && this.dia==15)

this.dia=4;

else if (this.ano==1 && this.mes==1 && this.dia==1)

{

this.dia=31;

this.mes=12;

this.ano=-1;

}

else if (Data.isValida((byte)(this.dia-1),this.mes,this.ano))

this.dia--;

else if (Data.isValida((byte)31,(byte)(this.mes-1),this.ano))

{

this.dia=31;

this.mes--;

}

else if (Data.isValida((byte)30,(byte)(this.mes-1),this.ano))

{

this.dia=30;

this.mes--;

}

else if (Data.isValida((byte)29,(byte)(this.mes-1),this.ano))

{

this.dia=29;

this.mes--;

}

else if (Data.isValida((byte)28,(byte)(this.mes-1),this.ano))

{

this.dia=28;

this.mes--;

}

else

{

this.dia=31;

this.mes=12;

this.ano--;

}

}

\*/

public void tranformeSeNoDiaAnterior ()

{

Data anterior = this.getDiaAnterior();

this.dia=anterior.dia;

this.mes=anterior.mes;

this.ano=anterior.ano;

}

//

public void retroceda (int qtdDias) throws Exception

{

if (qtdDias<=0)

throw new Exception ("Quantidade inválida");

for (int i=0; i<qtdDias; i++)

this.tranformeSeNoDiaAnterior();

}

/\*

public void tranformeSeNoDiaSeguinte ()

{

if (this.ano==1582 && this.mes==10 && this.dia==4)

this.dia=15;

else if (this.ano==-1 && this.mes==12 && this.dia==31)

{

this.dia=1;

this.mes=1;

this.ano=1;

}

else if (Data.isValida((byte)(this.dia+1),this.mes,this.ano))

this.dia++;

else if (Data.isValida((byte)1,(byte)(this.mes+1),this.ano))

{

this.dia=1;

this.mes++;

}

else

{

this.dia=1;

this.mes=1;

this.ano++;

}

}

\*/

public void tranformeSeNoDiaSeguinte ()

{

Data seguinte = this.getDiaSeguinte();

this.dia=seguinte.dia;

this.mes=seguinte.mes;

this.ano=seguinte.ano;

}

//

public void avance (int qtdDias) throws Exception

{

if (qtdDias<=0)

throw new Exception ("Quantidade inválida");

for (int i=0; i<qtdDias; i++)

this.tranformeSeNoDiaSeguinte();

}

@Override

public String toString ()

{

return this.dia+"/"+this.mes+"/"+this.ano;

}

@Override

public boolean equals (Object obj)

{

if (this==obj) return true;

if (obj==null) return false;

//if (!(obj instanceof Data)) return false;/

//if (obj.getClass()!=this.getClass()) return false;

if (obj.getClass()!=Data.class) return false;

Data data = (Data)obj;

if (this.dia!=data.dia) return false;

if (this.mes!=data.mes) return false;

if (this.ano!=data.ano) return false;

return true;

}

@Override

public int hashCode ()

{

//int ret=super.hashCode(); // super.hashCode() por NÃO herdar DIRETAMENTE de Object (e SIM EXPLICITAMENTE de outra classe)

int ret=666; // um valor qualquer positivo qualquer por herdar DIRETAMENTE de Object (e NÃO EXPLICITAMENTE de outra classe)

ret = 13\*ret + new Byte (this.dia).hashCode(); // this.dia

ret = 3\*ret + new Byte (this.mes).hashCode(); // this.mes

ret = 7\*ret + new Short (this.ano).hashCode(); // this.ano

if (ret<0) ret = -ret;

return ret;

}

/\*

Estamos neste método desejando comparar this e data;

deveremos retornar um valor negativo, caso this seja

menor que data;

deveremos retornar um valor positivo, caso this seja

maior que data;

deveremos retornar ZERO, caso this seja

IGUAL à data.

\*/

@Override

public int compareTo (Data data)

{

if (this.ano<data.ano) return -1;

if (this.ano>data.ano) return 1;

if (this.mes<data.mes) return -1966;

if (this.mes>data.mes) return 1966;

if (this.dia<data.dia) return -666;

if (this.dia<data.dia) return -666;

if (this.dia>data.dia) return 666;

return 0;

}

public Data (Data modelo) throws Exception

{

if (modelo==null)

throw new Exception ("Modelo ausente");

this.dia = modelo.dia;

this.mes = modelo.mes;

this.ano = modelo.ano;

}

@Override

public Object clone ()

{

Data ret=null;

try

{

ret = new Data (this);

}

catch (Exception erro)

{} // ignorando Exception, pois sei que não ocorrera

return ret;

}

}

+++++++++++++++++++++++++++

**3.1.3.Arquivo: Main.java**

public class Main

{

public static void main (String[] args)

{

try

{

Conjunto<String> c1 = new Conjunto<String> ();

c1.inclua("C");

c1.inclua("C++");

c1.inclua("JAVA");

c1.inclua("PYTHON");

System.out.println(c1);

Conjunto<Integer> c2 = new Conjunto<Integer> (6);

c2.inclua(2);

c2.inclua(3);

c2.inclua(5);

c2.inclua(7);

System.out.println(c2);

Conjunto<Data> c3 = new Conjunto<Data> (8);

c3.inclua(new Data ((byte)19,(byte)1,(short)1966));

c3.inclua(new Data ((byte)29,(byte)6,(short)1992));

Data natal = new Data ((byte)25,(byte)12,(short)2021);

c3.inclua(natal);

System.out.println(c3);

Conjunto<String> a = new Conjunto<String> ();

a.inclua("amarelo");

a.inclua("verde");

Conjunto<String> b = new Conjunto<String> ();

b.inclua("preto");

b.inclua("azul");

b.inclua("roxo");

Conjunto<String> c = new Conjunto<String> ();

c.inclua("rosa");

c.inclua("branco");

Conjunto<String> d = new Conjunto<String> ();

d.inclua("lilas");

d.inclua("laranja");

d.inclua("cinza");

Conjunto<Conjunto<String>> c4 = new Conjunto<Conjunto<String>> (15);

c4.inclua(a);

c4.inclua(b);

c4.inclua(c);

c4.inclua(d);

System.out.println(c4);

}

catch (Exception erro)

{

System.err.println (erro.getMessage()); // {{amarelo,verde},{preto,azul,roxo},{rosa,branco},{lilas,laranja,cinza}}

}

}

}

**Na aula passada falou sobre conjunto, classe genérica tem parâmetros; vetor com elemento genérico não pode e por isso que foi feito vetor de object**. O vetor deve ser dimensionado; fizemos 2 construtores e 1 cópia. Fez os métodos ondeEsta, redimensionese. (18min07s)

**Para o método toString temos que: ele serve para ver se o inclua estava funcionado**. O toString vai ter que retornar uma string por isso que declara um objeto ret que vale {, preenche essa string e no fim ele retorna ret. O preenchimento começa com ret abrindo chaves, e no fim vai ser acrescentado no ret um fecha chaves. No meio acontece que se a quantidade de elementos do meu conjunto for maior de zero, isto é, tem elementos (pelo menos 1), eu acrescento no ret o elemento 0 (primeiro elemento). Depois faço um for começando 1, indo até a quantidade e acrescento elemento. Dentro do for, o ret recebe uma vírgula a direita do direito. O zero faz separado porque zero não tem vírgula antes dele. Se i fosse 0, a qtd teria que ser menor que 0, como não tem vetor com posição menor que 0 o for ficaria inútil. Do elemento 1 até o último, todos tem vírgula a esquerda. Eu concateno vírgula e números. O teste de qtd>0 é para ver se tem algum elemento no conjunto pois pode ser conjunto vazio. Se tiver mais do que zero elementos, eu posso pegar o elemento 0 (posição 0) e colocar na string.

**Agora vamos fazer o método tem , é bem tranquilo de fazer tendo em vista que já tenho pronto o ondeEsta.**O método tem será feito como adiante:

public boolean tem (X x) throws Exception

{

if (x==null)

throw new Exception ("Elemento ausente");

Object[] onde = this.ondeEsta(x);

boolean achou = (Boolean)onde[0];

return achou;

}

**O método tem é público, parâmetro x; verifica se x é nulo, se sim lança exceção.** Senão for nulo, chamo o método ondeEstá que retorna um vetor de Object chamado onde(boolean, integer). Eu vou na posição 0, acho um boolean para ver se achou. Se achou, retorno que achou. Só quero saber se tem ou não o x no meu vetor: basta saber se achou ou não.

**O método getElemento é feito da seguinte maneira:**

public X getElemento (int i) throws Exception

{

if (i<0 || i>=this.qtd)

throw new Exception ("Elemento invalido");

return (X)this.elem[i];

}

**O getElemento é bem simples, ele vai começar testando se o que a anta passou é valido.** O getElemento é para pegar um elemento no vetor. No vetor vai ter elemento do 0 até o fim; se passar elemento negativo traz exceção; se ela quer um elemento maior que a quantidade (só tem 10 elementos e quer o 50) vai dar exceção. Senão deu exceção, retorno para anta o elemento i.

**O método remova é feito da seguinte maneira:**

public void remova (X x) throws Exception

{

if (x==null)

throw new Exception ("Elemento ausente");

Object[] onde = this.ondeEsta(x);

boolean achou = (Boolean)onde[0];

int posicao = (Integer)onde[1];

if (!achou)

throw new Exception ("Elemento inexistente");

for (int i=posicao; i<this.qtd-1; i++)

this.elem[i] = this.elem[i+1];

this.qtd--;

this.elem[this.qtd]=null;

}

**O método remova é para tirar um certo elemento x do vetor.** Primeiro tem que checar se o x é nulo, se for dá exceção.Se a anta forneceu o que é pra remover, mas nem sei se existe. Eu tenho que procurar o x que a anta quer remover; procuro pelo ondeEsta o x. O ondeEsta retorna um vetor de objeto; na posição 0 pego o boolean e guardo na variável achou, vou na posição 1 e guardo a posição do x. Agora eu já sei se tem lá no meu conjunto o que a anta quer remover porque eu mandei procurar, já sei se achou e qual a posição.

**Se por algum acaso não achou, é porque x não existe, mando exceção de elemento inexistente.** E agora vamos desenhar a situação. Imagina um vetor vet tem os seguintes elementos: 2,3, 5,7 ,11,13 . Vou desenhar. Estou desenhando um vetor.

elem: 2 3 5 7 11 13

Posição: 0 1 2 3 4 5

qtd: 6 x:5 achou: true posição: 2

**Vamos fazer de conta que o x é o 5. Senão desenhar a lógica não sai. Quando eu mandei procurar o x que é 5, ele vai achar, portanto achou vai ser true e x está na posição 2.** E agora precisa fazer a lógica da remoção. Você tem que remover o que está na posição 2. Você vai ter que pegar o 7 e jogar na posição do 5. Agora tem 2 elementos 7. Vai pegar o 11 e jogar no 7, e resolve o problema de ter dois 7 mas agora tem dois 11. Vai pegar o 13 e jogar no lugar do 11. Agora tem dois 13 e acabou a repetição. Agora eu tenho que jogar nulo na posição 5. Que lugar é esse? qtd-1. qtd-1 é o lugar do último; se qtd é 6, qtd-1 é 5. Jogando nulo na posição 5, eu diminuo qtd para 5. Agora pronto, está removido, tem 5 elementos que não se repetem; e qtd vale 5. Agora vamos programar.:

**Eu vou fazer o for com uma certa variável i começando na posição que eu quero remover (posição 2), avariável i está aumentando até chegar em qtd-1 (vai até o penúltimo).** Por que na hora de indexar eu pego i+1 (pego o último). Se o i chegasse o último, quando pegasse i+1 a repetição não faria sentido. O i começa valendo 2, eu pego i+1 que é 3 e jogo na posição i (2). Então eu pego o 7 e jogo no 2. Aumenta o i, (3), pego a posição 4 (11) e jogo na posição 3. O i agora é 4, pego a posição 5 (13) e jogo na posição 4 (11). Aumenta o i que é 5 e o for parou, porque estou no último: 5 é menor que 5 ? não, então parou.

for (int i=posicao; i<this.qtd-1; i++)

this.elem[i] = this.elem[i+1];

**O for acabou e deixou o 13 repetido e isso não pode. O qtd vale quanto? 6 porque tinha coisas guardadas; depois que exclui o 5 era para ter 5 coisas guardadas.** Faço this.qtd-- ; e agora, vou lá no vetor na posição 5 jogo nulo (this.elem[this.qtd] = null;

this.qtd--;

this.elem[this.qtd]=null;

**O this.qtd é a posição no vetor. Sempre que quero mexer numa casinha que é enumerada eu coloco o número. Quando coloco null, a casinha ainda existe, mas não tem mais o 13**. O vetor nasceu com uma certa quantidade de casinhas todas vazias, coloquei uma coisa, só naquela posição ficou preenchida. Sempre teve casinhas vazias. O vetor não diminui de tamanho nem aumenta; o tamanho do vetor continua o mesmo, mas agora terá uma casinha vazia se quiser incluir elementos. Qualquer coisa que o usuário for fazer só vai até o qtd, se printar vai até o qtd. Nada acontece depois do qtd.

**Quando criei o vetor c1, não coloquei tamanho e ele tinha 10 casinhas vazias.** O c2 nasceu com tamanho 6 e todas com casinhas vazias. O conjunto nasce com todas as casinhas vazias, conforme for preenchendo eu preencho até onde quero de 1 a 1 e na hora de printar vai printar o que está preenchido (até qtd). Por exemplo: em c1 tem 4 casinhas preenchidas.

**Quando você manda incluir i não cabe e dobra o tamanho, fica com um monte de casinhas vazias.** Usa 1 e aumenta o qtd em 1+. Uma coisa é quantidade de casinhas que tem no vetor, outra coisa é a quantidade ocupada guardada no qtd. Tudo o que eu faço no algoritmo vai até qtd.

Fazer os métodos obrigatórios.

1. **Pasta: 2021-10-13.S**

**Arquivos:**

**NÃO TEM TRANSCRIÇÃO DE VÍDEOAULA, APENAS OS ARQUIVOS JAVA**

* 1. **Lista de Arquivos da pasta (sem transcrição de aulas)**

**4.1.1.Arquivo: Aluno.java**

public class Aluno

{

//...

private int idade;

private String nome;

private Data nascimento;

//...

//...

public void setIdade (int i) throws Exception

{

if (i<=5)

throw new Exception ("Idade invalida");

this.idade = i;

}

public int getIdade ()

{

return this.idade;

}

public void setNome (String n) throws Exception

{

if (n==null)

throw Exception ("Nome ausente");

this.nome = n;

}

public String getNome ()

{

return this.nome;

}

public void setNascimento (Data n) throws Exception

{

if (n==null)

throw Exception ("Data ausente");

this.nascimento = new Data (n);

//this.nascimento = (Data)n.clone();

}

public Data getNacimento ()

{

return new Data (this.nascimento);

//return (Data)this.nascimento.clone();

}

//...

}

------------------

**4.1.2.Arquivo: Conjunto.java**

public class Conjunto <X>

{

private Object[] elem; // private X[] elem;

private int qtd=0;

private int capacidadeInicial;

public Conjunto (int capInicial) throws Exception

{

if (capInicial<=0)

throw new Exception ("Capacidade inválida");

//this.elem = new X [capInicial];

this.elem = new Object [capInicial];

this.capacidadeInicial = capInicial;

}

public Conjunto () // construtor padrão (construtor sem parâmetros)

{

//this.elem = new X [10];

this.elem = new Object [10];

this.capacidadeInicial = 10;

}

// este método deve descobrir se o x está armazenado em this.elem;

// estando, devo informar em que posição está;

// não estando, devo informar em que posição posso incluir, caso queira incluir

private Object[] ondeEsta (X x)

{

for (int i=0; i<this.qtd; i++)

if (x.equals(this.elem[i]))

{

Object[] ret = {true,i};

return ret;

}

Object[] ret = {false,this.qtd};

return ret;

}

private void redimensioneSe (int novaCap)

{

//X [] novo = new X [novaCap];

Object[] novo = new Object [novaCap];

for (int i=0; i<this.qtd; i++)

novo[i] = this.elem[i];

this.elem = novo;

}

public void inclua (X x) throws Exception

{

if (x==null)

throw new Exception ("Elemento ausente");

Object[] onde = this.ondeEsta(x);

boolean achou = (Boolean)onde[0];

int posicao = (Integer)onde[1];

if (achou)

throw new Exception ("Elemento repetido");

if (this.qtd==this.elem.length)

this.redimensioneSe (2\*this.elem.length);

for (int i=this.qtd-1; i>=posicao; i--)

this.elem[i+1] = this.elem[i];

if (x instanceof Cloneable)

this.elem[posicao] = x.clone();

else

this.elem[posicao] = x;

this.qtd++;

}

public boolean tem (X x) throws Exception

{

if (x==null)

throw new Exception ("Elemento ausente");

Object[] onde = this.ondeEsta(x);

boolean achou = (Boolean)onde[0];

return achou;

}

public X getElemento (int i) throws Exception

{

if (i<0 || i>=this.qtd)

throw new Exception ("Elemento invalido");

if (this.elem[i] instanceof Cloneable)

return (X)this.elem[i].clone();

else

return (X)this.elem[i];

}

public void remova (X x) throws Exception

{

if (x==null)

throw new Exception ("Elemento ausente");

Object[] onde = this.ondeEsta(x);

boolean achou = (Boolean)onde[0];

int posicao = (Integer)onde[1];

if (!achou)

throw new Exception ("Elemento inexistente");

for (int i=posicao; i<this.qtd-1; i++)

this.elem[i] = this.elem[i+1];

this.qtd--;

this.elem[this.qtd]=null;

if (this.elem.length>this.capacidadeInicial && this.qtd<=(int)(0.25\*this.elem.length))

this.redimensioneSe ((int)(0.5\*this.elem.length));

}

@Override

public String toString ()

{

String ret="{";

if (this.qtd>0)

ret += this.elem[0];

for (int i=1; i<this.qtd; i++)

ret += ","+this.elem[i];

ret += "}";

return ret;

}

@Override

public boolean equals (Object obj)

{

if (this==obj) return true;

if (obj==null) return false;

if (this.getClass()!=obj.getClass()) return false;

Conjunto<X> conj = (Conjunto<X>)obj;

if (this.capacidadeInicial!=conj.capacidadeInicial) return false;

if (this.qtd !=conj.qtd) return false;

//leva em conta a ordem

//for (int i=0; i<this.qtd; i++)

// if (!this.elem[i].equals(conj.elem[i]))

// return false;

// sem levar em conta a ordem

for (int i=0; i<this.qtd; i++)

{

boolean achou = (Boolean)this.ondeEsta((X)conj.elem[i])[0];

if (!achou)

return false;

}

return true;

}

@Override

public int hashCode ()

{

int ret=1;

ret = 19\*ret + new Integer (this.capacidadeInicial).hashCode();

ret = 13\*ret + new Integer (this.qtd ).hashCode();

for (int i=0; i<this.qtd; i++)

//if (this.elem[i]!=null)

ret = 7\*ret + this.elem[i].hashCode();

if (ret<0) ret=-ret;

return ret;

}

public Conjunto (Conjunto<X> modelo) throws Exception // construtor de cópia (construtor com um parâmetro, cujo tipo é a própria classe)

{

if (modelo==null)

throw new Exception ("Modelo ausente");

this.capacidadeInicial=modelo.capacidadeInicial;

this.qtd =modelo.qtd;

//this.elem = new X [this.modelo.length];

this.elem = new Object [modelo.elem.length];

for (int i=0; i<this.qtd; i++)

this.elem[i] = modelo.elem[i];

}

public Object clone ()

{

Conjunto<X> ret=null;

try

{

ret = new Conjunto<X> (this);

}

catch (Exception erro)

{}

return ret;

}

}

--------------

**4.1.3.Arquivo: Data.java**

public class Data implements Comparable<Data>, Cloneable

{

private byte dia, mes;

private short ano;

private static boolean isBissexto (short ano)

{

if (ano<1583) // vigencia do Calendario Juliano

if (ano%4 == 0)

return true;

else

return false;

// vigencia do Calendario Gregoriano

if (ano%400 == 0)

return true;

if (ano%4==0 && ano%100!=0)

return true;

return false;

}

private static boolean isValida (byte dia, byte mes, short ano)

{

if (dia<1 || dia>31)

return false;

if (mes<1 || mes>12)

return false;

if (ano==0)

return false;

if ((mes==4 || mes==6 || mes==9 || mes==11) && dia>30)

return false;

if (mes==2 && dia>29)

return false;

if (ano==1582 && mes==10 && dia>4 && dia<=14)

return false; // Bula Papal Inter Gravissimas, do Papa Gregório XIII

if (dia>28 && mes==2 && !Data.isBissexto(ano))

return false;

return true;

}

public Data (byte dia, byte mes, short ano) throws Exception

{

if (!Data.isValida(dia,mes,ano))

throw new Exception ("Data invalida");

this.dia = dia;

this.mes = mes;

this.ano = ano;

}

public void setDia (byte dia) throws Exception

{

if (!Data.isValida(dia,this.mes,this.ano))

throw new Exception ("Dia invalido");

this.dia = dia;

}

public void setMes (byte mes) throws Exception

{

if (!Data.isValida(this.dia,mes,this.ano))

throw new Exception ("Mes invalido");

this.mes = mes;

}

public void setAno (short ano) throws Exception

{

if (!Data.isValida(this.dia,this.mes,ano))

throw new Exception ("Ano invalido");

this.ano = ano;

}

public byte getDia ()

{

return this.dia;

}

public byte getMes ()

{

return this.mes;

}

public short getAno ()

{

return this.ano;

}

/\*

// há validações duplicadas

public Data getDiaAnterior ()

{

Data ret=null;

if (this.ano==1582 && this.mes==10 && this.dia==15)

{

try

{

ret = new Data ((byte)4,this.mes,this.ano);

}

catch (Exception erro1)

{} // ignora erro por ser impossível

}

else if (this.ano==1 && this.mes==1 && this.dia==1)

{

try

{

ret = new Data ((byte)31,(byte)12,(short)-1);

}

catch (Exception erro2)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)(this.dia-1),this.mes,this.ano))

{

try

{

ret = new Data ((byte)(this.dia-1),this.mes,this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)31,(byte)(this.mes-1),this.ano))

{

try

{

ret = new Data ((byte)31,(byte)(this.mes-1),this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)30,(byte)(this.mes-1),this.ano))

{

try

{

ret = new Data ((byte)30,(byte)(this.mes-1),this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)29,(byte)(this.mes-1),this.ano))

{

try

{

ret = new Data ((byte)29,(byte)(this.mes-1),this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)28,(byte)(this.mes-1),this.ano))

{

try

{

ret = new Data ((byte)28,(byte)(this.mes-1),this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else

{

try

{

ret = new Data ((byte)31,(byte)12,(short)(this.ano-1));

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

return ret;

}

\*/

public Data getDiaAnterior ()

{

Data ret=null;

if (this.ano==1582 && this.mes==10 && this.dia==15)

{

try

{

ret = new Data ((byte)4,this.mes,this.ano);

}

catch (Exception erro1)

{} // ignora erro por ser impossível

}

else if (this.ano==1 && this.mes==1 && this.dia==1)

{

try

{

ret = new Data ((byte)31,(byte)12,(byte)-1);

}

catch (Exception erro2)

{} // ignora erro por ser impossível

}

else

{

try

{

ret = new Data ((byte)(this.dia-1),this.mes,this.ano);

}

catch (Exception erro3)

{

try

{

ret = new Data ((byte)31,(byte)(this.mes-1),this.ano);

}

catch (Exception erro4)

{

try

{

ret = new Data ((byte)30,(byte)(this.mes-1),this.ano);

}

catch (Exception erro5)

{

try

{

ret = new Data ((byte)29,(byte)(this.mes-1),this.ano);

}

catch (Exception erro6)

{

try

{

ret = new Data ((byte)28,(byte)(this.mes-1),this.ano);

}

catch (Exception erro7)

{

try

{

ret = new Data ((byte)31,(byte)12,(byte)(this.ano-1));

}

catch (Exception erro8)

{} // ignora erro por ser impossível

}

}

}

}

}

}

return ret;

}

//

public Data ha (int qtdDias) throws Exception

{

Data referencia = (Data)this.clone();

referencia.retroceda(qtdDias);

return referencia;

}

/\*

// há validações duplicadas

public Data getDiaSeguinte ()

{

Data ret=null;

if (this.ano==1582 && this.mes==10 && this.dia==4)

{

try

{

ret = new Data ((byte)15,this.mes,this.ano);

}

catch (Exception erro1)

{} // ignora erro por ser impossível

}

else if (this.ano==-1 && this.mes==12 && this.dia==31)

{

try

{

ret = new Data ((byte)1,(byte)1,(short)1);

}

catch (Exception erro2)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)(this.dia+1),this.mes,this.ano))

{

try

{

ret = new Data ((byte)(this.dia+1),this.mes,this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else if (Data.isValida((byte)1,(byte)(this.mes+1),this.ano))

{

try

{

ret = new Data ((byte)1,(byte)(this.mes+1),this.ano);

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

else

{

try

{

ret = new Data ((byte)1,(byte)1,(short)(this.ano+1));

}

catch (Exception erro3)

{} // ignora erro por ser impossível

}

return ret;

}

\*/

public Data getDiaSeguinte ()

{

Data ret=null;

if (this.ano==1582 && this.mes==10 && this.dia==4)

{

try

{

ret = new Data ((byte)15,this.mes,this.ano);

}

catch (Exception erro1)

{} // ignora erro por ser impossível

}

else if (this.ano==-1 && this.mes==12 && this.dia==31)

{

try

{

ret = new Data ((byte)1,(byte)1,(byte)1);

}

catch (Exception erro2)

{} // ignora erro por ser impossível

}

else

{

try

{

ret = new Data ((byte)(this.dia+1),this.mes,this.ano);

}

catch (Exception erro3)

{

try

{

ret = new Data ((byte)1,(byte)(this.mes+1),this.ano);

}

catch (Exception erro4)

{

try

{

ret = new Data ((byte)1,(byte)1,(byte)(this.ano+1));

}

catch (Exception erro5)

{} // ignora erro por ser impossível

}

}

}

return ret;

}

//

public Data daquiA (int qtdDias) throws Exception

{

Data referencia = new Data (this); // usando construtor de cópia

referencia.avance(qtdDias);

return referencia;

}

/\*

public void tranformeSeNoDiaAnterior ()

{

if (this.ano==1582 && this.mes==10 && this.dia==15)

this.dia=4;

else if (this.ano==1 && this.mes==1 && this.dia==1)

{

this.dia=31;

this.mes=12;

this.ano=-1;

}

else if (Data.isValida((byte)(this.dia-1),this.mes,this.ano))

this.dia--;

else if (Data.isValida((byte)31,(byte)(this.mes-1),this.ano))

{

this.dia=31;

this.mes--;

}

else if (Data.isValida((byte)30,(byte)(this.mes-1),this.ano))

{

this.dia=30;

this.mes--;

}

else if (Data.isValida((byte)29,(byte)(this.mes-1),this.ano))

{

this.dia=29;

this.mes--;

}

else if (Data.isValida((byte)28,(byte)(this.mes-1),this.ano))

{

this.dia=28;

this.mes--;

}

else

{

this.dia=31;

this.mes=12;

this.ano--;

}

}

\*/

public void tranformeSeNoDiaAnterior ()

{

Data anterior = this.getDiaAnterior();

this.dia=anterior.dia;

this.mes=anterior.mes;

this.ano=anterior.ano;

}

//

public void retroceda (int qtdDias) throws Exception

{

if (qtdDias<=0)

throw new Exception ("Quantidade inválida");

for (int i=0; i<qtdDias; i++)

this.tranformeSeNoDiaAnterior();

}

/\*

public void tranformeSeNoDiaSeguinte ()

{

if (this.ano==1582 && this.mes==10 && this.dia==4)

this.dia=15;

else if (this.ano==-1 && this.mes==12 && this.dia==31)

{

this.dia=1;

this.mes=1;

this.ano=1;

}

else if (Data.isValida((byte)(this.dia+1),this.mes,this.ano))

this.dia++;

else if (Data.isValida((byte)1,(byte)(this.mes+1),this.ano))

{

this.dia=1;

this.mes++;

}

else

{

this.dia=1;

this.mes=1;

this.ano++;

}

}

\*/

public void tranformeSeNoDiaSeguinte ()

{

Data seguinte = this.getDiaSeguinte();

this.dia=seguinte.dia;

this.mes=seguinte.mes;

this.ano=seguinte.ano;

}

//

public void avance (int qtdDias) throws Exception

{

if (qtdDias<=0)

throw new Exception ("Quantidade inválida");

for (int i=0; i<qtdDias; i++)

this.tranformeSeNoDiaSeguinte();

}

@Override

public String toString ()

{

return this.dia+"/"+this.mes+"/"+this.ano;

}

@Override

public boolean equals (Object obj)

{

if (this==obj) return true;

if (obj==null) return false;

//if (!(obj instanceof Data)) return false;/

//if (obj.getClass()!=this.getClass()) return false;

if (obj.getClass()!=Data.class) return false;

Data data = (Data)obj;

if (this.dia!=data.dia) return false;

if (this.mes!=data.mes) return false;

if (this.ano!=data.ano) return false;

return true;

}

@Override

public int hashCode ()

{

//int ret=super.hashCode(); // super.hashCode() por NÃO herdar DIRETAMENTE de Object (e SIM EXPLICITAMENTE de outra classe)

int ret=666; // um valor qualquer positivo qualquer por herdar DIRETAMENTE de Object (e NÃO EXPLICITAMENTE de outra classe)

ret = 13\*ret + new Byte (this.dia).hashCode(); // this.dia

ret = 3\*ret + new Byte (this.mes).hashCode(); // this.mes

ret = 7\*ret + new Short (this.ano).hashCode(); // this.ano

if (ret<0) ret = -ret;

return ret;

}

/\*

Estamos neste método desejando comparar this e data;

deveremos retornar um valor negativo, caso this seja

menor que data;

deveremos retornar um valor positivo, caso this seja

maior que data;

deveremos retornar ZERO, caso this seja

IGUAL à data.

\*/

@Override

public int compareTo (Data data)

{

if (this.ano<data.ano) return -1;

if (this.ano>data.ano) return 1;

if (this.mes<data.mes) return -1966;

if (this.mes>data.mes) return 1966;

if (this.dia<data.dia) return -666;

if (this.dia<data.dia) return -666;

if (this.dia>data.dia) return 666;

return 0;

}

public Data (Data modelo) throws Exception

{

if (modelo==null)

throw new Exception ("Modelo ausente");

this.dia = modelo.dia;

this.mes = modelo.mes;

this.ano = modelo.ano;

}

@Override

public Object clone ()

{

Data ret=null;

try

{

ret = new Data (this);

}

catch (Exception erro)

{} // ignorando Exception, pois sei que não ocorrera

return ret;

}

}

----------------

**4.1.4.Arquivo: Main.java**

public class Main

{

public static void main (String[] args)

{

try

{

Conjunto<String> c1 = new Conjunto<String> ();

c1.inclua("C");

c1.inclua("C++");

c1.inclua("JAVA");

c1.inclua("PYTHON");

System.out.println(c1);

Conjunto<Integer> c2 = new Conjunto<Integer> (6);

c2.inclua(2);

c2.inclua(3);

c2.inclua(5);

c2.inclua(7);

System.out.println(c2);

Conjunto<Data> c3 = new Conjunto<Data> (8);

c3.inclua(new Data ((byte)19,(byte)1,(short)1966));

c3.inclua(new Data ((byte)29,(byte)6,(short)1992));

Data natal = new Data ((byte)25,(byte)12,(short)2021);

c3.inclua(natal);

natal.setAno((short)2025);

System.out.println(c3);

Conjunto<String> a = new Conjunto<String> ();

a.inclua("amarelo");

a.inclua("verde");

Conjunto<String> b = new Conjunto<String> ();

b.inclua("preto");

b.inclua("azul");

b.inclua("roxo");

Conjunto<String> c = new Conjunto<String> ();

c.inclua("rosa");

c.inclua("branco");

Conjunto<String> d = new Conjunto<String> ();

d.inclua("lilas");

d.inclua("laranja");

d.inclua("cinza");

Conjunto<Conjunto<String>> c4 = new Conjunto<Conjunto<String>> (15);

c4.inclua(a);

c4.inclua(b);

c4.inclua(c);

c4.inclua(d);

System.out.println(c4);

}

catch (Exception erro)

{

System.err.println (erro.getMessage()); // {{amarelo,verde},{preto,azul,roxo},{rosa,branco},{lilas,laranja,cinza}}

}

}

}

1. **Pasta: Aula 14-10-2021\_DS201**

**Arquivo: Reflect para chamar clone (classe Conjunto)\_Aula 14102021.mp4**

**AULA 5 – Reflect para chamar clone (classe Conjunto) – Aula 14 Outubro 2021**

A nossa **classe Conjunto por ser uma classe genérica pode ser criada para fazer conjuntos do que a gente quiser.** Trabalhamos com conjutnos de Strings; mas não é a única possibilidade. A classe genérica pode ser usada para criar conjuntos de qualquer coisa. No programa1.java, eu crio um conjunto de datas chamado c, foi instanciado com novo conjunto de datas com tamanho inicial 3. Como a gente usa um conjunto de datas? Declaro a data, instancio a data (com valores de 19/1/1966) e estou incluindo a data niverMaligno no conjunto c. Posso declarar outra data, instanciá-la com valores de 29/8/1992 e incluir a data niverMalignoJr no conjunto c. Mando printar o conjunto e as duas datas apareciam printadas entre chaves; é a maneira que o toString printa conjuntos, separados por vírgula.

**Uma coisa que a gente precisa saber sobre a classe conjunto, se eu declarar um conjunto c e declarar uma data como niverMaligno e depois declarar e incluir uma segunda data**. Quando eu printar o nosso conjunto, deverá aparecer na tela entre chaves as duas datas separadas por vírgula.

**Mas se eu, usando o objeto que foi declarado como niverMalignoJr, usar o método setMes e alterar o valor do mês e printar o conjunto c; o conjunto vai estar alterado e vai ter na segunda data o mês 6 ao invés de ter o mês 8.** Isso é um fato bem preocupante, porque eu inclui as datas que são guardadas pelo método inclua no vetor chamado elem. O vetor elem é private para não ser mexida fora da classe conjunto. Todavia, foi exatamente isso que aconteceu. Eu não mexi com o conjunto c apenas com o objeto niverMalignoJr.

**O que exatamente está acontecendo? A gente precisa saber que todos os objetos em java são variáveis que guardam endereços de memória, como ponteiros.** Quando declarou o objeto c é uma variável capaz de guardar um endereço de memória. Quando dei um new Conjunto um conjunto foi criado em algum lugar da memória e o endereço dele foi guardado em c.

**Declarei um objeto niverMalignoJr como uma variável capaz de guardar endereço de memória, instanciei em new para ter lugar na memória para guardar os dados de uma data.** O construtor guardou 29,8,1992. O new retornou o endereço de memória que foi o nosso objeto niverMalignoJr. Quando usei o método inclua do objeto c, passei o endereço de memória do objeto niverMalignoJr passado ao método inclua. O inclua recebe o endereço de memória que foi passado e depois de muitas validações vai pegar o endereço de memória recebido (x) e armazenar em uma posição do vetor elem.

**Por isso quando eu chego mais abaixo e chamo setMes para alterar o mês da data cujo endereço de memória está guardado no objeto niverMalignoJr,** eu altero de 8 para 6; o endereço de memória está salvo no objeto niverMalignoJr e no meu vetor elem. O vetor elem mudou.

**Se eu resolvesse na classe conjunto, logo após o inclua implementar um novo método:**

**Imagina que eu implementasse o método public x getElemento (int i) throws Exception (que vem logo após o inclua).** Então:

public x getElemento (int i) throws Exception

{

if (i <0 | | i>=this.qtd)

throw new Exception (“Elemento inválido”);

return this.elem [i];

}

**Esse método getElemento permitiria recuperar um elemento que está no meu conjunto, por exemplo no programa2.java.**

**5.1. Lista de Arquivos usados na aula (aula 5)**

**5.1.1.Arquivo: Programa2.java**

public class Programa2

{

public static void main (String [] args)

{

try

{

Conjunto <Data> c = new Conjunto <Data> (3);

//..

//guardo várias datas no Conjunto <Data> c

// ...

Data data = c.getElemento (2);

Data.setMes ((byte) 6);

System.out.println (c);

}

catch (Exception erro)

{}

**Eu poderia no programa2.java ter um conjunto c instanciado e nele ter várias datas (esse é o tamanho inicial, mas o conjunto cresce se necessário).** Se eu guardei várias datas e eu quiser usar o método getElemento que acabei de fazer a data que está guardada na posição 2 (seria a 3ª data). Eu desejo recuperar a 3ª data e guardar no objeto data da classe Data. Eu passo 2 como a posição da data a recuperar; i =2 , validei se é menor ou maior que as posições do vetor. Como não deu exceção, pego do vetor de elementos e retorno na posição i a data guardada.

**O mesmo problema que apontou no programa1 de ter o mês alterado na main, não é para pegar uma variável e fazer uma coisa e alterar o mês e depois mudar o valor da variável private na classe Conjunto, isso não é para acontecer.** Esse mesmo problema pode acontecer no programa2 etapa de getElemento. Porque? Porque eu estou solicitando a data guardada na posição 2, se eu for no meu conjunto, na posição 2 do vetor elem tem o endereço de memória onde foi conseguido espaço para guardar valores da data (dia,mês e ano). Nesse meu vetor elem tenho vários endereços de memória diferentes de várias datas e eu estou retornando com o comando o endereço de memória que está armazenado na posição2. Nesse endereço de memória que está armazenado na posição 2, o que eu retornei vai ser armazenado no objeto data da main. Eu tenho no vetor na posição 2 um endereço e o mesmo endereço é guardado no objeto data porque o que foi retornado de getElemento foi guardado no objeto data. Se eu pegar a data e usar o método setMes para alterar o mês para 6, o mês será alterado na data do meu conjunto porque lá no vetor tem o mesmo endereço.O que é que eu deveria fazer?

**Tem uma outra implementação da classe conjunto onde ele já fez o que precisaria ser feito. (Arquivo: Conjunto.java).** Lá no nosso método inclua eu não deveria o endereço de uma data para incluir no meu conjunto e simplesmente pegar o endereço que recebi e guardar no meu vetor. Eu deveria verificar se o objeto cujo endereço está em x e foi recebido pelo meu método inclua como parâmetro, tenho que verificar se o x é clonável (instanceof). Se for clonável eu deveria guardar no meu vetor não necessariamente o objeto que eu recebi, mas sim uma cópia recebida, usaria o método clone para guardar uma cópia do x,. Se eu fizesse a linha this.elem [this.qtd] = (X) x.clone (); //erro de compilação o que iria acontecer? Eu iria pegar a data cujo endereço está guardado nesse x, o método clone iria fazer uma cópia dessa data em outro lugar da memória e iria retornar o endereço desse outro lugar da memória onde tem a cópia. E eu armazenaria no meu vetor esse outro endereço onde está a cópia.

**Se o x não for clonável , eu guardaria no meu vetor o próprio x. A nossa Data é clonável, quando eu guardasse o x que fosse data eu entraria no if e faria o clone para ter a cópia de data.** Mas se eu trabalhasse com o conjunto de strings não são clonáveis e entraria no else e armazenaria o próprio x recebido. No else eu guardo no vetor elem uma shallow copy de x e, no if o vetor elem guarda uma deep copy no formato (X) x.clone () de x.

**A mesma coisa que fazemos para o inclua precisaremos fazer para o getElemento. Ao invés de simplesmente retornar o endereço que está na posição i (que é o else que retorna this.elem[i]) , eu deveria verificar se o que tem na posição i é um endereço de posição de um objeto que é clonável.** Se for clonável eu devo pegar o endereço de memória ali guardado daquele objeto (this.elem[i]) e devo chamar o método clone para clonar e retornar o objeto clonado, retornar o endereço de uma cópia após a cópia ser criada. Seria o endereço dessa cópia que o método getElemento retornaria e seria recebido na main e guardado em data. Se eu fizer um setMes estarei alterando a cópia. Com isso eu termino os problemas de não ter cópia quando tem conjunto alterado tendo variável private mexida.

**Na classe conjunto quando eu uso o método clone ele vai clonar o que tiver no vetor. E a gente que o que tem no vetor são objetos da classe X (this.elem = (X[]) new Object [capInicial]) que nesse exemplo, no programa2 é Data.** A gente sabe que em private X[] = elem; nós guardamos Data porque X é Data. E quando eu chego no método getElemento e dentro dele eu uso o método clone para clonar o objeto cujo endereço tenho lá no vetor, eu sei que a cópia dele vai resultar numa data. O método clone sempre retorna Object. Por isso que tenho o typecast (X) para forçar para o tipo Data, pois o retorno do clone geralmente é Object e com o typecast o retorno do clone será uma Data. Como eu tenho X pode ser qualquer classe. Eu digo o conjunto do que eu quero. Nesses programas de hoje o X é Data e o getElemento precisa retornar Data. E o clone vai retornar Data, está clonando data mas retorna como clone; eu forço a ser visto como Data para retornar como Data já que se espera que eu retorne Data (usando o typecast).

**Mesma coisa acontece no método inclua , eu pego o x que eu sei que é do tipo X e nesses programas que usam a classe Conjunto o X é Data.** Se eu fizer uma copia clonada de x vou conseguir uma Data. O clone vai retornar a Data como Object, por isso eu forço o (X) para ser visto como Data e retornado como Data; por isso que usa typecast.

**O que eu precisava fazer de fato é o que eu preciso fazer. Mas os métodos que usam clone vão causar erros de compilação tanto no inclua como no getElemento.** Ao compilar realmente deram os erros de compilação esperados. Porque? Clone tem acesso protegido em object. Não posso chamar clone a partir de qualquer objeto. Eu chamo clone a partir da classe X que é uma classe genérica, misteriosa e qualquer que teria sido passada como parâmetro para um conjunto. (31min19s).

**Preste atenção no fato de que estou usando o clone com um chamante do tipo X (que é o return (X) this.elem[i].clone (); no método getElemento).** E qual é a única coisa que sei sobre o tipo X ? Ele vai ser passado como parâmetro na classe Conjunto que irá recebe-lo. Quando eu fizer um programa que usa a classe Conjunto vou ter que fornecer alguma coisa para ser X, no Programa1 ofereci Data. Mas a única coisa que eu sei que o X que está sendo oferecido como parâmetro pro Conjunto é que ele é uma Classe. Pois as classes genéricas tem parâmetros que sempre devem ser classes. O parâmetro X é Data na aula passada era String, e ambos são classes. Todas as classes herdam de Object, no X tem os métodos de Object. Por conta disso, eu consigo chamar os métodos de Object, como por exemplo o método Equals.

**Vamos ver o método ondeEsta pois ele é usado no equals. O método ondeEsta chama o método equals para um objeto x da classe X.** O método equals é aceito sem dar erro; a partir do chamante da classe X. Também não dá erro quando no hashcode (34min32s) quando eu pego o que que tem guardado no vetor (this.elem [i]) que é um objeto da classe X e partir dele, usando ele como chamante o uso como chamante.

**Se eu fizer a mesma coisa para chamar clone vai dar erro dentro do método getElemento**

if (this.elem [i] instanceof Cloneable)

return (X) this.elem[i].clome (); // o chamante era um objeto da classe X assim como no método inclua,

**Porque não dá erro em equals e hashCode mas dá erro com clone?** O equals e hashcode são métodos que existem na classe Object e a classe X como qualquer classe herda da classe Object e tem os métodos equals e hashcode mas NÃO herda da classe Object o método Clone. Por isso que não posso chamar clone direto. Os métodos que eu sei que são herdados de Object ele aceita que eu chame. Mesmo que o método descubra que x instanceof Cloneable seja verdadeiro e tenha o método clone; mesmo que dentro do getElemento o if (this.elem [i] instanceof Cloneable) tenha o método clone pois em ambos os casos a classe X é Data e Data tem clone. O if se baseia no tipo de X. O x é do tipo X, X herda de Object, Object não tem clone para ser usado, então não pode usar clone. É um problema que pode ser contornado; igual o erro de vetor de X com this.elem = new X [capInicial].

**Para contornar o problema de Clone , faremos a classe Programa3.java**

**5.1.2.Arquivo: Programa 3.java**

public class Programa3

{

Public static void main (String [] args )

{

String mensagem = “ JAVA: A linguagem mais Top que existe”;

String parte;

parte = mensagem.substring (23,26);

System.out.println (parte);

}

}

**A princípio, o programa3 não tem nada a vê com o programa que estava sendo mostrado a pouco (conjunto, classe genérica, clone).** O programa3 vai mostrar um ponto a ser compreendido; para contornar o problema de chamar o clone da classe genérica X.

**Esse programa3 declara uma string chamada mensagem e coloca nessa string a seguinte mensagem: “Java: a linguagem mais top que existe”.** Em seguida declarei outra string chamada parte e chamei o método que existe na classe String que é o subtring e eu usei como chamante mensagem. Passei dois valores: 23 e 26. O 23 é a posição do char T, portanto o O é a posição 24, o p a posição 25 e o char vazio é a posição 26.

T- o - p (null)

23 24 25 26

**Quando a gente pega um subtring tem que em posição a subtring começa que é na 23; e qual a posição seguinte ao qual a posição que termina (é o char vazio 26).** Ele gera um subtring que começa na posição 23 incluindo o 23 e termina na posição 26 sem incluir a posição 26. Com a chamada do subtring eu isolei o Top e guardei no string parte e printei o objeto parte, foi printado Top.

**Sobre o método subtring que foi usado no programa4, se for no google e digitar String API java 14 (documentação java 14) eu encontro os links, se clicar no Oracle eu caio na documentação da classe String.** A coisa interessante sobre a classe String é que ela tem 2 métodos substring:

**substring (in beginIndex) 🡪** tem 1 parâmetro e retorna uma string que é a substring dessa string. Esse submétodo pega uma substring que começa na posição beginIndex e por não ter segundo parâmetro para determinar onde para ele vai até o fim da substring.

**substring (in beginIndex, int endIndex) 🡪** tem 2 parâmetros e retorna uma string que é a substring dessa string. O método que a gente chamou, recebe 2 parâmetros: beginIndex e endIndex.

**A classe String tem 2 métodos substring, com 1 e 2 parâmetros; usamos o que tem 2 parâmetros.** Agora pense de um modo muito estranho. O programa3 funciona e é simples. Para fazer o programa4 mais complicado, mas que funcione do mesmo jeito. Veja o programa4.java

**No programa4 eu importo a biblioteca java.lang.reflect.\* e esse programa tem na classe main o objeto mensagem que é uma string e contem a mesma mensagem sobre Java.** Eu tenho a string chamado parte que recebia a substring, mas de modo bem mais complicado.

**A primeira coisa que eu faço é declarar um objeto classe de uma classe pronta do java chamada Class .** Essa classe class representa uma classe. Quando estudamos sobre o método equals , usávamos o método getClass. O método getClass retorna um objeto da classe Class e esse objeto classe será capaz de armazenar uma classe; eu estou pegando o objeto mensagem como chamante do método getClass () vou descobrir a classe do objeto mensagem. O que tem guardado no objeto classe? A classe do objeto mensagem que nós sabemos ser string. Dá para guardar uma classe dentro de uma variável? Sim, pois o objeto classe tem guardado a classe String do objeto mensagem.

**Agora eu declarei um vetor chamado tpsParmsForms , é uma abreviação para tipos dos parâmetros formais.** Um parâmetro formal é quando você declara um método qualquer, por exemplo: contem, ele pode receber parâmetros; nesse caso vai receber o parâmetro conj. O parâmetro que eu coloco quando eu declaro um método qualquer; o parâmetro colocado na implementação do método é chamado de parâmetro formal.

**Mas mais tarde vou usar a classe conjunto na main, pode ser que eu use o método contém e ao ser chamado eu passe um parâmetro.** O parâmetro que eu passo na hora de chamar um método é chamado de parâmetro real.

**Os parâmetros reais quando são chamados entram no lugar dos parâmetros formais e o método executa em cima dos parâmetros reais na hora de chamar um método.** Observe que eu declarei um vetor de objetos tpsParmsForms da classe Class <?>; a classe Class é uma classe genérica. Estou instanciando esse vetor com novo vetor de objetos da classe Class de tamanho 2 porque eu estou aqui para simular a chamada do substring e tem 2 parâmetros.

**Depois de declarados o tpsParmsForms, eu coloco int na posição 0 e int na posição 1. Esse é um vetor que guarda classes; em ambas as posições uso int.class.** Porque quero chamar o método substring.

**Agora vamos declarar (54min36s) um objeto chamado método da classe Method. Objetos da classe Method são capazes de guardar métodos.** Esse nosso objeto método vai guardar o que? Eu pego a classe que está no objeto classe (que é a String pegando a mensagem no método getClass e guardando no objeto class) ; através do getMethod da classe Class pegar o objeto chamante classe extrair um método da classe String. Qual método eu pego? Substring. Mas tem 2 métodos de Substring. Eu quero o que aceita esse tipo de tpsParmsForms , ou seja, eu quero o que aceita 2 ints.Estou passando o vetor tpsParmsForms no getMethod para pegar o método substring da classe String que está guardada em classe. Tenho que pegar o substring que aceita 2 parâmetros (dois ints). Peguei o método e guardei no objeto método. O objeto método tem guardado o método subtring que tem 2 parametros que aceite dois ints.

**Agora vou declarar outro vetor chamado parmsReais é de Object e eu instancio como new Object que recebe tamanho 2.** O método subtring vai precisar de 2 parametros reais quando vou chamar. Na posição 0 coloquei o 23; na posição 1 do vetor coloquei 26 que é o outro numero inteiro que quero passar quando chama subtring. (igual em parte = mensagem.substring (23,26)).

**Agora eu vou pegar o método que está guardado no objeto método que é o substring da classe String, aquele que aceita 2 parâmetros int.** Eu vou invocar esse método. Quando um método é chamado ele precisa de um this que eu coloquei como mensagem (mensagem como objeto chamante) e os parmsReais como os parâmetros que eu quero fornecer. Chamei o método substring pela método invoke e ele vai retornar uma string menor que está dentro do objeto mensagem. Porém vai retornar como object; eu forço pelo typecast (String) o retorno a ser encarado como uma string e guardo no objeto chamado parte que é uma string. Eu printo a parte. (01h01min46s).

**Os meus catchs estão fazendo nada; porque eu sei o que estou fazendo e não é para dar erro; eu chamo coisas que existem.** Se eu pedisse pro usuário digitar o erro pode ser que desse um erro muito grande. Ao contrário, eu estou dizendo os métodos. Por isso que eu ignoro os (possíveis) erros.

**Tudo aquilo que aprendemos (01h06min36s) até o momento foi para contornar o erro do clone na classe conjunto.** O jeito diferente seria usar o modo estranho para chamar o subtring para chamar o clone.Como chamar o clone pode dar erro, em qualquer classe genérica sempre que eu quiser clonar um objeto cuja classe seja parâmetro de uma classe genérica; como no método getElemento que eu quero clonar X que é parâmetro da minha classe Conjunto. Sempre que eu tiver uma classe genérica qualquer, que tem lá os seus parâmetros e eu precisar pegar um objeto cuja classe é parâmetro para a minha classe genérica. Sempre que eu precisar clonar um objeto cujo parâmetro seja X da classe genérica, eu vejo que dá erro ao ter a chamada direta dentro da classe X, eu contorno usando um outro jeito. Crio a classe chamada clonador.

**A classe Clonador terá o método clone; será uma classe genérica pois recebe como parâmetro X, clona objetos de qualquer tipo que eu passar como parâmetro**. No método clone, ela vai receber o objeto x dessa classe X; sendo o X passado como parâmetro para a classe Clonador e vai retornar o objeto da classe X que seria cópia do objeto x. Vou ter que chamar o clone desse x; vou usar a biblioteca reflect que encontra a classe method, exceptions**.**  Eu pego a classe x, chamo o getClass e guardo no objeto classe vinda da classe Class que é genérica. **(01h11min31s).** Logo, o que eu guardo no objeto classe é a classe do objeto x (que é o objeto que eu quero clonar).

**Em seguida eu declaro o vetor tpsParmsForms , é de classes e coloco nulo.** Coloco nulo porque pretendo chamar clone que não tem parâmetro.

**O método é o objeto da classe Method que recebe nulo. Coloquei nesse objeto método a classe (que é x que está sendo clonado) .getMehod (o objeto classe é o objeto chamante para o método getMethod).** Estou pegando o método clone. Qual? Só tem um mas tenho que dizer: estou pegando o método clone que está pegando o tpsParmsForms, ou seja, que aceita não receber parâmetro formal nenhum pois o vetor é nulo.No objeto método eu guardo o clone da classe X que é a classe do objeto x que eu quero clonar.

**O que eu faço? Declaro parâmetros reais um vetor de Object também nulo por não ter parâmetros. Eu faço o seguinte**: declaro o objeto ret como se fosse o objeto parte; o ret vai servir para guardar o retorno da minha clonagem. Pego o objeto método (que é o método clone) da classe X que é o objeto da classe x. Então eu pego o objeto método e invoco mandando para ser o chamante o x (chamante do método invoke) e passando o vetor parmsReais que é nulo (não tem parâmetro no clone). Se eu clonei x que é da classe X o retorno deve ser o clone do objeto x; mas ele vem para mim como object. Faço o typecast com X para retornar como a classe X no retorno do método clone chamado pelo invoke. Assim consigo guardar o clone no objeto ret que é da classe X. Eu retorno ret como a cópia clonada do meu x.

**Agora vamos corrigir o problema do clone dentro da classe Conjunto <X> usando a classe Clonador <X> .** No método inclua, dentro do if (x instanceof Cloneable) eu declaro um objeto chamado clonador da classe Clonador <X> que vai clonar coisas do tipo X; e instancio como new Clonador <X> (); na linha debaixo, o vetor elem na posição this.qtd recebe clonador.clone (x) daí o clonador clona; retorna a cópia clonada que é guardarda em this.elem [this.qtd].

**Mesma coisa no método getElemento , dava erro na chamada direta de clone. Para me livrar desse erro, vou na versão final da classe Conjunto <X>** , se o vetor elem na posição i for clonável eu declaro o objeto clonador da classe clonador. Vou usar o objeto clonador como chamante da classe clone (lá da classe clonador) para clonar o que tem o vetor elem na posição i.

**5.1.4. Versão final dos métodos que usam o clone atualizados conforme abaixo, para a classe Conjunto <X>**

public void inclua (X x) throws Exception

{

if (x==null)

throw new Exception (“Elemento ausente”);

if (this.tem (x))

throw new Exception (“Elemento repetido”);

if (this.qtd == this.elem.length)

this.redimensioneSe (2\*this.elem.length);

if (x instanceof Cloneable)

{

Clonador <X> clonador = new Clonador <X> ();

this.elem [this.qtd] = clonador.clone;

}

else

this.elem [this.qtd] = x;

this.qtd ++;

}

public X getElemento (int i) throws Exception

{

if (i<0 | | i>=this.qtd)

throw new Exception (“Elemento inválido”);

if (this.elem[i] instanceof Cloneable)

{

Clonador <X> clonador = new Clonador <X> ();

return clonador.clone (this.elem [i]);

}

else

return this.elem[i];

}

Vamos ver o arquivo: **Clonador.java**

import java.lang.reflect.\*;

public class Clonador <X>

{

public x clone (X x)

{

//assim obtemos a classe de instancia no objeto x,que, conforme sabemos , é a classe X, // e a armazenamos no objeto chamado classe

Class <?> class = x.getClass ();

//null porque chamaremos um método sem parâmetros

Class <?> [] tpsParmsForms = null;

// assim obtemos o método chamado clone, sem parâmetros, da Class <?> armazenada no //objeto classe (sabemos que a classe é a classe X)

Method método = null

try

{

Método = classe.getMethod (“clone”, tpsParmsForms);

}

catch (NoSuchMethodException erro)

{}

// null porque chamaremos um método sem parâmetros

Object [] parmsReais = null;.

// assim chamamos o método armazenado no objeto chamado método, fazendo com que //o objeto chamado x seja para ele o objeto chamante (o this) e fazendo com que não //receba parâmetros reais (por isso o vetor nulo chamado parmsReais é fornecido; o //resultado da chamada, que certamente é da classe X, mas que está sendo retornado como Object , tem seu tipo mudado para X e então, é atribuído ao objeto chamado ret

X ret = null;

try

{

ret = (X) método.invoke (x.parmsReais);

}

catch (InvocationTargetException erro)

{} //

catch (IllegalAccessException erro)

{}

// ret = (X) x.clone ();

return ret;

}

}

**5.1.5.Arquivo: Programa4.java**

import java.lang.reflect.\*;

public class Programa3

{

public static void main (String [] args)

{

String mensagem = “ JAVA: A linguagem mais Top que existe”;

String parte;

// assim obtemos a classe da instância no objeto mensagem, que conforme sabemos, é a classe String, e a armazenamos no objeto chamado classe.

Class <?> classe = mensagem.getClass ();

// 2 porque chamaremos um método com 2 parametros

Class <?> [] tps ParmsForms = new Class <?> [2];

// int.class 2 vezes porque ambos os parâmetros do método que chamaremos são do tipo int

tpsParmsForms [0] = int.class;

tpsParmsForms [1] = int.class;

// assim obtemos o método chamado substring com 2 parametros do tipo in da Class <?> armazenada no objeto classe (sabemos que a classe é a classe String)

Method método = null;

try

{

Método = classe.getMethod (“substring”, tpsParmsForms);

}

catch (NoSuchMethodException erro)

{}

// 2 porque chamaremos um método com 2 parametros

Object [] parmsReais = new Object [2];

// 23 e 25 são, respectivamente, os paraemtros reais que desejamos passar ao método que chamaremos

parmsReais [0] = 23;

parmsReais [1] = 26;

// assim chamamos o método armazenado no objeto chamado método, fazendo com que

// o objeto chamado mensagem seja para ele o objeto chamante (o this) e fazendo com //que receba os parâmetros presentes no vetor chamado parmsReais; o resultado da //chamada, que certamente é uma String; mas que está sendo retornado como Object, tem seu tipo mudado para String e, então, é atribuído ao objeto chamado parte.

parte = null

try

{

Parte = (String) método.invoke (mensagem.parmsReais);

}

catch (InvocationTargetException erro)

{} // 59min07s

catch (IllegalAccessException erro)

{}

// parte = mensagem.substring (23,26);

System.out.println (parte);

}

**5.1.6.Arquivo: Conjunto.java (VERSÃO FINAL)**

public class Conjunto<X>

{

private X [] elem;

private int qtd= 0;

public Conjunto (int capInicial) throws Exception

{

if (capInicial <=0)

throw new Exception (“Capacidade invalida”);

//this.elem = new X [capInicial];

this.elem = (X []) new Object [capInicial];

}

public Conjunto ()

{

//this.elem = new X [1];

this.elem (X[]) new Object [10];

}

private void redimensioneSe (int novaCap)

{

//X [] novo = new X [novaCap];

Object[] novo = new Object [novaCap];

for (int i=0; i<this.qtd; i++)

novo[i] = this.elem[i];

this.elem = novo;

}

// retorna a posição em this.elem onde está x,ou retorna -1 quando não encontrar x em this.elem.

private int ondeEsta (X x)

{

for (int i=0; i<this.qtd; i++)

if (x.equals (this.elem [i]))

return -1;

}

public boolean tem (X x) throws Exception

{

if (x==null)

throw new Exception (“Elemento ausente”);

return this.ondeEsta (x) != -1;

}

public void inclua (X x) throws Exception

{

if (x==null)

throw new Exception (“Elemento ausente”);

if (this.tem (x))

throw new Exception (“Elemento repetido”);

if (this.qtd == this.elem.length)

this.redimensioneSe (2\*this.elem.length);

if (x instanceof Cloneable)

this.elem [this.qtd] = (X) x.clone (); //erro de compilação

else

this.elem [this.qtd] = x;

this.qtd ++;

}

public X getElemento (int i) throws Exception

{

if (i<0 | | i>=this.qtd)

throw new Exception (“Elemento inválido”);

if (this.elem[i] instanceof Cloneable)

return (X) this.elem[i].clone (); //erro de compilação

else

return this.elem[i];

}

Public Conjunto <X> união (Conjunto <X> conj) throws Exception

{

If (conj == null)

throw new Exception (“Conjunto ausente”);

Conjunto <X> ret = new Conjunto <X> (this.qtd + conj.qtd);

Conjunto <X> menor, maior;

if (this.qtd < conj.qtd)

{

menor = this;

maior = conj;

}

else

{

menor = conj;

maior = this;

}

ret.qtd = maior.qtd;

for (int i = 0; i<maior.qtd; i++)

ret.elem[i] = maior.elem [i];

for (int i=0, i<menor.qtd; i++)

{

try

{

ret.inclua (menor.elem [i]);

}

Catch (Exception erro)

{} // ignoro repetidos e toco em frente

}

return ret;

}

public Conjunto <X> intersecção (Conjunto <X> conj) throws Exception

{

if (conj ==null)

throw new Exception (“Conjunto ausente”);

Conjunto <X> menor, maior;

Conjunto <X> menor, maior;

if (this.qtd<conj.qtd)

{

menor = this;

maior = conj;

} // aqui é 33min41s

else

{

menor = conj;

maior = this;

}

Conjunto <X> ret = new Conjunto <X> (menor.qtd);

{

boolean tem = false;

try

{

tem = maior.tem (menor.elem[i]);

}

catch (Exception erro)

{} // sei que no conjunto menor não há null

if (tem)

{

ret.elem [ret.qtd] = menor.elem[i];

ret.elem [ret.qtd] = menor.elem[i];

ret.qtd++;

}

}

return ret;

}

public Conjunto <X> menos (Conjunto <X> conj) throws Exception

{

Conjunto <X> ret = new Conjunto <X> (this.qtd);

for (int i = 0; i<this.qtd; i++)

if (!conj.tem (this.elem[i]))

{

ret.elem[ret.qtd~= this.elem[i];

ret.qtd++;

} // aqui são 33min44s.

return ret;

}

public boolean contem (Conjunto <X> conj) throws Exception

{

if (conj==null)

throw new Exception (“Conjunto ausente”);

for (int i=0; i<this.qtd; i++)

if (this.ondeEsta(conj.elem[i] == -1)

return false;

return true;

}

public boolean estaContido (Conjunto <X> conj) throws Exception

{

if (conj==null)

throw new Exception (“Conjunto ausente”);

return conj.contem (this).

}

@Override

public String toString ()

{

String ret = “{“ ;

If (this.qtd>0)

ret += this.elem [0];

for (int i =1; i<this.qtd; i++)

for (int i =1; i<this.qtd; i++)

ret += “,” +this.elem[i];

ret += “}”;

return ret;

}

@Override

@Override

public boolean equals (Object obj)

{

if (this==obj) return true;

if (obj==null) return false;

if (this.getClass () != obj.getClass () ) return false;

Conjunto <X> conj = (Conjunto <X>) obj;

if (this.qtd != conj.qtd) return false;

// leva em conta a ordem;

// for (int i=0; i<this.qtd; i++)

// if (!this.elem[i].equals (conj.elem[i]))

// return false;

//sem levar em conta a ordem

for (int i =0; i<this.qtd; i++)

if (this.ondeEsta (conj.elem[i] == -1)

return false;

return true;

}

@Override

public int hashCode ()

{

int ret =1;

ret = 13\*ret + new Integer (this.qtd).hashCode ();

for (int i=0; i<this.qtd; i++)

//if (this.elem[i] !=null)

ret = 7\*ret +this.elem[i].hashCode ();

if (ret<0) ret=-ret;

return ret;

}

}

**5.1.7.Arquivo: Programa1.java**

public class Programa

{

public static void main (String [] args)

{

try

{

Conjunto <Data> c = new Conjunto ,Data> 3/

Data niverMaligno = new Data ((byte) 19, (byte)1, (short) 1966));

c.inclua (niverMaligno);

Data niverMalignoJr = new Data ((byte) 29, (byte) 8 , (short) 1992));

c.inclua (niverMalignoJr);

System.out.println (c);

// {19/01/1966, 29/08/1992}

niverMalignoJr.setMes (byte) 6 );

System.out.println (c);

// // {19/01/1966, 29/06/1992}

}

catch (Exception erro)

{}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  
**6. Pasta: 18-10-2021\_DS201**

**Arquivos: Sockets\_Aula\_18-10-2021.mp4**

**AULA 6 – Sockets (dia 18 Outubro 2021)**

Ele vai **explicar como a gente pode implementar programas que potencialmente podem rodar em máquinas diferentes e vão se comunicar com a rede uns com os outros;** vão trocar dados e vão dessa maneira colaborar uns com os outros para realização de um objetivo maior.

**Vamos lembrar sobre o significado da palavra Sistema . Ouvimos muito no contexto computacional, falar a respeito sobre sistema:** o sistema da minha empresa não é muito bom, caiu o sistema. O que é um sistema? É um conjunto de programas que formam algo maior; conjunto de programas que se complementam e formam algo maior que é o sistema, eles colaboram entre si.

**O que vai ser ensinado hoje, não deixa de ser um sistema; pois não há nada que impeça o sistema de se comunicar pela rede**. Alguns sistemas não se comunicam pela rede. Estamos focados no sistema cujo tipo se comunicam uns com os outros através da rede.

**Para compreender como são feitos esses programas, vamos fazer uma analogia com telefones fixos.** Vamos pensar em 2 pessoas querendo se comunicar /falar por um telefone fixo. As pessoas falando pelo telefone fixo seriam equivalentes a programas se comunicando pela rede. As pessoas seriam como os programas que querem se comunicar: as pessoas usam o telefone fixo e os programas a rede.

**Como acontece a comunicação pelo telefone fixo? Na casa de todas as pessoas que tem o telefone fixo, chega um cabo/fio que fica pregado na parede da pessoa.** O fio sai da central telefônica e chega até a casa dos assinantes. Uma ponta do fio está na casa da pessoa e a outra ponta está na central telefônica. Para que serve uma central telefônica? Para comutar. Comutar significa ligar duas pontas que estão chegando na central.

**Por exemplo, eu quero falar com você (05min01s). Na minha casa tem uma ponta de fio que está chegando na central.** A central conecta essa ponta do meu fio com a ponta do fio que sai da central e chega na sua casa. Como a central fez essa conexão entre os fios? É como se o fio que estivesse na minha casa estivesse conectado com o fio que está na sua casa. E vamos nos falar.

**Para poder se falar, o assinante precisa ter o aparelho telefônico conectado a esse fio. Esse aparelho deve ser um em cada casa. Os aparelhos telefônicos tem duas partes fundamentais:**

1) Um microfone que você coloca na direção da boca para falar;

2) Um alto-falante que você coloca na direção do ouvido para escutar;

**É assim que acontece a comunicação entre pessoas usando o telefone fixo. E se fossem programas?** Dá para fazer a analogia e compreender como seria na programação. Esse fio que conecta o meu aparelho telefônico com o seu aparelho telefônico. Na programação seria o fio imaginário, a própria rede faz essa conexão.Lembre-se que tem uma ponta de fio na minha casa e uma ponta de fio na sua casa. Tem 2 pontas de fio. Na programação, vou precisar ter 2 objetos da classe Socket: um deles num programa e outro deles em outro programa. Para que esses dois programas possam se comunicar.

**O microfone ligado a cada ponta do fio na programação seria o objeto de uma dessas classes: PrintWriter ou PrintStream, DataOutputStream, ObjectOutputStream, etc.** Então um objeto instanciado por uma dessas classes acima associado ao Socket; é o necessário para transmitir os dados entre os programas. O telefone serve para transmitir a voz, ele capta a voz e mandar. E objetos instanciados dessas classes acima associado ao Socket são para a mesma finalidade: para mandar dados de um programa para outro programa. Para o telefone conseguir falar ele precisa usar o microfone. Pro programa conseguir mandar dados ele precisa de um objeto instanciado de uma das classes associados ao software.

**O alto-falante ligado a cada ponta do fio, o papel do autofalante serve para escutar. No programa, o objeto instanciado de uma das classes instanciadas adiante associado ao Socket: BufferedReader, DataInputStream, ObjectInputStream, etc** é que serve como se fosse o alto-falante. O alto falante serve para receber a voz de outra pessoa; então esses objetos servem para receber os dados. Na hora de receber os dados, eu mando através do objeto de uma dessas classes como a BufferedReader.

**Como eu vou escolher a classe apropriada para cada situação? Depende do que eu quero transmitir.** Se eu quiser transmitir texto, eu uso PrintWriter ou PrintStream; para receber eu uso BufferedReader.

**Se o que eu quero mandar são dados binários (fotos, músicas): eu uso DataOutputStream para mandar e DataInputStream para receber.** Se eu quiser objetos para o outro programa receber e usar : eu posso usar ObjectOutputStream para mandar o objeto e oObjectInputStream para receber esse objeto. O etc é muito importante pois tanto para mandar como receber são algumas classes de exemplo, pois sempre tem alternativas.

**Precisamos conhecer o mundo que tem a arquitetura cliente/servidor. O que é um sistema cliente/servidor?** É um sistema que tem pelo menos 2 programas; onde 1 faz o papel de servidor e outro o papel de cliente. O servidor fica esperando (é passivo) que o cliente se conecte a ele para pedir alguma coisa, nós conhecemos vários programas que tem arquitetura cliente/servidor. Por exemplo, quando você quer ler email, você entra no programa leitor de email que é o cliente. Posso entrar no celular, se for para web entro por navegador ou pode ser que ele seja um programa que não seja para web e sim para usar dentro da máquina e entra as telas desse programa.**(14min44s).**

**O programa leitor de e-mail é um programa cliente. Na hora em que você manda ler o e-mail, o programa cliente se comunica com o programa servidor que é onde os seus e-mails estão guardados.** E pede pro servidor os e-mails guardados (pode pedir todos, os de hoje, os não lidos) e mostra pro usuário ( o cliente pede pro servidor e o servidor manda). O cliente mostra na tela os assuntos dos e-mails e o usuário escolhe ler um deles e clica qual deles quer ler. Daí o cliente manda pro servidor um pedido pra mandar o conteúdo texto do e-mail. O servidor manda pro programa cliente e o programas cliente mostra na tela para a pessoa.

**Você sabe que quando você está lendo e-mails eles não estão na minha casa. Estão remotamente em algum lugar por aí; e quem lê esses e-mails é o servidor. Na sua casa, você usa um programa, no seu celular você usa um programa.** Esse programa é o cliente que se comunica com o servidor, pede um email. O cliente se comunica com um servidor mandando apagar um email. O cliente se comunica com o servidor mandando apagar um email. Logo, todas as coisas que você quer fazer, você faz no seu programa cliente que está no seu computador, celular. Mas o programa não tem acesso direto aos e-mails pis não estão no celular e sim remotamente em algum lugar. O que o programa cliente faz? Se comunica com aquele outro programa que seria o cliente servidor e ele sim vai acessar os e-mails na máquina onde estão guardados, vai mandar pro cliente que vai mostrar na tela e você vai conseguir ver. (17min13s). A arquitetura cliente (proativo)/ servidor (reativo). O servidor reage as iniciativas do cliente.

**Existe um programa proativo (cliente) e reativo (reage as iniciativas do cliente, o servidor). O reativo roda numa certa máquina e as máquinas precisam ser encontradas no mar de máquinas que é a rede.** Para identificar a máquina onde roda o reativo, eu uso o ip da máquina. Toda máquina tem um endereço ip que eu uso para identificar uma máquina na rede. Alguém sabe qual é o endereço ip da máquina do google que responde as suas consultas/buscas? Muito difícil, Ao invés disso, todo mundo sabe o nome da máquina do google onde eu posso fazer as minhas buscas:www.google.com . Uma máquina não precisa ser necessariamente identificada através de um endereço ip, posso identificar através de um nome simbólico.

**Para a máquina ser localizada na rede preciso ter o ip, mas a gente não sabe de cor os ips e sim os nomes simbólicos**. Existe o DNS (domain name server) que tem o papel de traduzir o nome simbólico de uma máquina para o endereço ip de uma máquina. Se eu souber o endereço ip de uma máquina posso escrever; se sei apenas o nome simbólico posso escrever, mas tudo funciona com o ip. Sem problemas porque o DNS vai traduzir o nome simbólico para o endereço ip da máquina. Por exemplo, numa prompt de comando para saber o ip da google poderia dar ping -4 [www.google.com](http://www.google.com) (escrevi o nome simbólico de uma máquina) e me respondeu com o ip: 172.217.167.100 e quem traduziu foi o dns.

**Os nomes simbólicos as vezes são nomes completos como** [**www.google.com**](http://www.google.com) **ou mais simples como gaia.** Um nome curto só é acessível na rede local onde essa máquina se encontra. Na rede da casa dele tem essa máquina. Da minha casa, não vou conseguir saber o ip e como um erro da máquina não estar acessível. Por exemplo: ping -4 Venus vai dar erro.(host vênus é a máquina vênus). Rede local é rede lan.

**Já o Google que tem que ser acessível pra todo mundo, tem o nome de uma rede global (WAN).** Se eu der o comando ipconfig ele mostra tudo que é ip que a sua máquina tem e pode ter vários ips (endereço ipv4) como placa de rede, adaptador de rede (wifi)

**Pode acontecer de vários programas reativos (servidores) estarem rodando numa mesma máquina.** Por isso, na hora do proativo se conectar, não basta saber em qual máquina se conectar cada reativo adota uma porta e o proativo deve saber a máquina + porta. Eu tenho que saber o ip da máquina servidor mas qual dos 3 programas que vai ser o servidor para aquele cliente que chega? Cada reativo adota uma porta (número) diferente e natural, o proativo deve saber o ip da máquina (ip ou nome simbólico) +porta.

**Para o programa Reativo.java é um servidor; e o programa Proativo.java é um cliente. Esses dois programas vão trocar textos.**

**No programa PROATIVO eu vou digitar um texto,** ele manda o texto para o programa Reativo. O programa Reativo recebe o texto e printa na tela.

**No programa REATIVO, tem que importar as bibliotecas io e net ;** o java.io importo para escrever e ler dados; o java.net é para transmitir, usar os recursos da rede para receber e enviar dados. Dentro do programa reativo, declaro o objeto da classe ServerSocket chamado pedido e instancio com novo ServerSocket e passei como parâmetro o 12345 ; esse número é a porta que o meu reativo está adotando. Para adotar a porta passa o número pro construtor de ServerSocket**.**

**O objeto pedido é o que vai fazer o meu programa reativo ficar esperando conexões do proativo. Como eu faço pro reativo ficar esperando conexões?** Declaro o objeto conexão da classe Socket que recebe o objeto pedido (objeto chamante) para chamar o método accept (). Sempre que chama o aceept o programa para e fica esperando um pedido de conexão, enquanto não tiver feito pelo proativo, o servidor fica parado esperando. E na hora que aconteceu um pedido de conexão? O accept retorna um socket e eu guardo o socket no objeto conexão. O socket é como se fosse a ponta do fio de telefone do programa reativo.

**Agora preciso do aparelho telefônico: autofalante e microfone. Para o autofalante:**

**6.1.Lista de Arquivos usados na aula 6.**

**6.1.1.Arquivo: 1ª versão do Reativo.java**

import java.io.\*;

import java.net.\*;

public class Reativo

{

public static void main (String[] args)

{

try

{

ServerSocket pedido = new ServerSocket (12345);

// fio do telefone fixo preso na parede

Socket conexao = pedido.accept();

BufferedReader receptor =

New BufferedReader( // passo pro construtor de BufferedReader o new InputStreamReader

New InputStreamReader ( // passo pro construtor de InputStreamReader o conexão.getInputStream ()

conexão.getInputStream () ) ); // autofalante ; 🡪 conexão é o meu socket (associa ao fio) para criar o autofalante (receptor)

//Toda vez que eu usar o BufferedReader para receber qualquer coisa ele vai receber através do socket (receptor).

// Eu declaro o objeto transmissor que serve como um microfone, vem da classe //PrintWriter ; instancio new PrintWriter e passo para o seu construtor //conexão.getOutputStream (). Conexão é o nosso socket , o fio.

/\*PrintWriter transmissor =

New PrintWriter (

conexão.getOutputStream ()); // Microfone \*/

String texto;

do

{

texto = (String)receptor.readLine ();

System.out.println (texto);

}

while (!texto.toUpperCase().equals ("FIM"));

//transmissor.close();

receptor.close();

conexao.close();

pedido.close();

}

catch (Exception erro)

{

System.err.println (erro.getMessage());

}

}

}

**Lá no programa PROATIVO acontece algo parecido mas não é bem igual.** Também implanta as mesmas bibliotecas, mas não tem server socket; apenas o socket de conexão. O proativo é o cliente. Fiz socket conexão = new Socket (“localhost”, 12345); em new Socket passo como parâmetros o localhost (máquina local) e a porta que eu quero me conectar. Usou localhost pois os dois programas estão na mesma máquina. Se tivesse em máquinas distintas teria que colocar o nome ou ip no lugar de localhost lá onde roda o servidor. No new Socket coloco qual o número da porta que o servidor adotou. Os dois estão conectados por terem socket em ambos (ponta do fio). Tenho que associar em cada ponta do fio o microfone e autofalante.

**No programa REATIVO já associei o receptor como autofalante** e o transmissor como microfone.

**No programa PROATIVO eu crio um receptor e um transmissor.** Depois crio o objeto teclado. O jeito de montar o receptor e transmissor são iguais, o que difere é o jeito de conseguir obter o socket. Consegue o socket fazendo new Socket, passa pro construtor como parâmetro a máquina que está rodando o servidor (nome ou ip) e a porta do servidor. Os comandos de receptor e transmissor são iguais. Feito isso já tenho um transmissor e receptor associados a uma ponta que é o socket e já tenho um transmissor e receptor de outro programa associado a ponta do fio de outro socket.

**Se no PROATIVO eu printar alguma coisa no transmissor, o reativo vai conseguir ler através do receptor.** Se o REATIVO printar alguma coisa no transmissor, o proativo vai conseguir ler no receptor. Ou seja, o que um printa no transmissor o outro lê no receptor. É como se tivesse o transmissor de um ligado no receptor do outro. (45min36s).

**No programa PROATIVO eu declarei mais um BufferedReader e chamei de teclado, instanciei como new Buffered Reader recebendo no construtor o parâmetro new InputStreamReader.** O InputStreamReader tem como parâmetro o System.in. Eu usei System.in ao invés de conexão.getInputStream Quando escrevo conexão.getInputStream, eu estou associando o meu BufferedReader ao socket; sempre que eu for ler esse receptor eu estarei lendo alguma coisa que veio da ponta do fio que é o socket, coloquei a indicação que a origem dos dados é a conexão. Já o objeto teclado tem declarado o System.in eu associei ao teclado; quer dizer que toda vez que eu ler o meu teclado que está associado ao BufferedReader estou lendo algo do teclado, pois a origem dos dados é o teclado.

**Eu declaro uma String texto. Faço o loop “do” e dentro dele, eu li alguma coisa que foi digitada no teclado; não aprendemos sobre digitar dados no teclado** . Eu li uma string do teclado e guardo na variável texto. Como estou no proativo, leu uma linha de texto. Ele printa a linha de texto no transmissor (mandou pro reativo). O comando flush é para envio imediato para o outro programa.

**No REATIVO ele vai declarar uma string texto; vai ter o looping “do” . Dentro do looping ele faz a string texto ser igual a receptor.readLine , ele vai ler o receptor.** Ele lè o que o proativo printou no transmissor. O receptor do reativo está ligado no transmissor do proativo. No reativo, ele leu o que o proativo mandou , usando o receptor.readLine no reativo e depois guarda na variável texto. Depois disso, o reativo printou na tela.

**Eu digito no proativo e printo no reativo. Digito no reativo, ele manda pro receptor do proativo.** No REATIVO o do recebe os dados do proativo e printa. E no outro programa? O PROATIVO lê do teclado, printa e manda pro reativo. Tudo o que for digitado no proativo vai ser mandado pro reativo, lido no reativo, printado na tela do reativo. A repetição no REATIVO vai até quando? Quando for mandado um texto escrito FIM o loop para. Porque? O loop fica em repetição enquanto não for verdade que o texto recebido do proativo, transformado em maiúscula seja igual a FIM.

**No loop do REATIVO fico recebendo e printando.** Quando receber FIM, paro loop; fecho o transmissor, fecho o receptor, fecho a conexão, fecho o canal de pedido e o programa termina.

**E O PROATIVO? Na hora que o usuário digitar no teclado FIM eu mando o Fim para o reativo e de lá o reativo vai parar.** Mas no PROATIVO ele também vai parar porque vai receber o texto digitado do usuário, vai ver se não é verdade que o texto digitado, transformado em maiúscula é igual a fim: ele le e manda pro reativo. Se for igual a fim, o proativo para o loop; fecha o transmissor, fecha o receptor, fecha a conexão e termina o programa.

**Ao rodar os programas temos que:**

**Se rodar primeiro o proativo ele vai tentar se conectar com o reativo que ainda está desligado e vai dar erro de conexão. (Conexão recusada).** Por isso que tem que rodar primeiro o reativo e depois o proativo. Os programas funcionam com envio imediato do proativo. Os dois programas terminam juntos com o reativo fechando primeiro. Como tem envio de texto, foi usado BufferedReader e PrintWritter. (55min38s).

**O nosso programa reativo NÃO precisa de um transmissor, posso deixar como comentário.** O reativo somente recebe as linhas e printa, nunca transmite nada por conta do modo que foi construído. (56min16s). O programa que só manda é o proativo.

**Então no REATIVO temos que:**

/\*PrintWriter transmissor =

New PrintWriter (

conexão.getOutputStream ()); // Microfone \*/

...

//transmissor.close();

**Se quisesse fazer um programa que manda e recebe, teria que colocar transmissor e receptor nos dois programas.** Do jeito que foi feito, o reativo não precisa de transmissor e o proativo não precisa de receptor. No REATIVO, comentar a linha de transmissor.close dentro do while.

**No PROATIVO não tem mais receptor no objeto, nem as chamadas de receptor.close Logo:**

/\*BufferedReader receptor =

New BufferedReader( // passo pro construtor de BufferedReader o new InputStreamReader

New InputStreamReader ( // passo pro construtor de InputStreamReader o conexão.getInputStream ()

conexão.getInputStream () ) ); // autofalante ; 🡪 conexão é o meu socket (associa ao fio) para criar o autofalante (receptor) \*/

//receptor.close ();

**Agora quer mostrar uma nova versão (2ª versão) dos programas Reativo e Proativo onde não usa transmissor do tipo PrintWritter nem PrintStream;** e no receptor não usa BufferedReader. Pois agora o Proativo e Reativo transmitem objetos.

**6.1.2.ARQUIVO: 2ª versão do Reativo.Java**

import java.io.\*;

import java.net.\*;

public class Reativo

{

public static void main (String[] args)

{

try

{

ServerSocket pedido = new ServerSocket (12345);

// fio do telefone fixo preso na parede

Socket conexao = pedido.accept();

/\*

// tranmissor sempre antes do receptor

ObjectOutputStream transmissor =

new ObjectOutputStream (

conexao.getOutputStream ()); // microfone

\*/

//receptor sempre depois do transmissor

ObjectInputStream receptor =

new ObjectInputStream (

conexao.getInputStream ()); // autofalante

String texto;

do

{

texto = (String)receptor.readObject ();

System.out.println (texto);

}

while (!texto.toUpperCase().equals ("FIM"));

/\*

transmissor.close();

\* \*/

receptor.close();

conexao.close();

pedido.close();

}

catch (Exception erro)

{

System.err.println (erro.getMessage());

}

}

}

**6.1.3.ARQUIVO: 2ª versão do proativo.Java**

import java.io.\*;

import java.net.\*;

public class Proativo

{

public static void main (String[] args)

{

try

{

// fio do telefone fixo preso na parede

Socket conexao = new Socket ("localhost",12345);

// tranmissor sempre antes do receptor

ObjectOutputStream transmissor =

new ObjectOutputStream (

conexao.getOutputStream ()); // microfone

/\*

//receptor sempre depois do transmissor

ObjectInputStream receptor =

new ObjectInputStream (

conexao.getInputStream ()); // autofalante

\*/

BufferedReader teclado =

new BufferedReader (

new InputStreamReader (

System.in));

String texto;

do

{

texto = teclado.readLine ();

transmissor.writeObject (texto);

transmissor.flush (); // para envio imediato

}

while (!texto.toUpperCase().equals ("FIM"));

transmissor.close();

/\*

receptor.close();

\*/

conexao.close();

}

catch (Exception erro)

{

System.err.println (erro.getMessage());

}

}

}

**Na segunda versão do REATIVO, ele tem um receptor que não é do tipo BufferedReader e sim ObjectInputStream;** significa que ele é capaz de receber objetos de qualquer classe. No reativo não precisa de transmissor: pois ele sempre vai fazer no do:

**Receber o texto em texto =(String) receptor.readObject (); 🡪** o receptor como objeto chamante vai chamar o método readObject, isto é, vai receber o objeto do Proativo e vai guardar na variável texto sendo enxergado como uma String. Após guardar o objeto no texto, vai ser printado esse texto. Na hora de receber os dados , vai receber strings como objetos porque a string é um texto no objeto. Lendo o meu objeto receptor através do método readObject. Esse método readObject retorna um Object mas como fazendo o meu programa para String, eu sei que serão passados apenas Strings, eu faço typecast para enxergar como String e guardo na variável texto que é String. O texto foi recebido como ObjectInputStream (e no transmissor será a classe ObjectOutputStream) faz com que receba e envie objetos de qualquer tipo de classe, incluindo as Strings.

No reativo, o resto do programa é tudo igual.

**Na segunda versão de PROATIVO temos que o transmissor usa a classe ObjectOutputStream como tipo de transmissor** , não tem receptor pelo modo do par dos programas foram feitos. Na hora de transmitir dentro do loop do: eu uso transmissor.writeObject (texto); porque o transmissor é do tipo de objeto. Eu continuo mandando textos como objetos do tipo String. Se eu quisesse que no proativo ter um receptor para receber dados seria do tipo ObjectInputStream eu vou ter que receber e transmitir objetos. Tem uma única restrição: só podemos receber e transmitir objetos que implementam que herdam da interface Serializable e Strings podem ser recebidos e transmitidos porque as Strings herdam da interface Serializable.

**A documentação da classe String implementa as interfaces Serializable, CharSequence, Comparable <String>** . Só classes que implementam a classe Serializable pode ser transmitida como ObjectOutputStream e recebidas através do ObjectInputStream.

**Para concluir, quer mostrar que é possível transmitir e receber objetos que pertencem a uma classe que foi nós mesmos que criamos.** Digamos que eu queira fazer uma classe que transmite e recebe data. Eu posso pegar a classe DATA que fizemos e implemente a interface Serializable.

**Para usar as classes ObjectInputReader e ObjectOutputReader precisamos importar a biblioteca import java.io.Serializable.** Faço no cabeçalho public class Data implements Serializable.

O restante da classe permanece o mesmo.

**Para a CLASSE REATIVO DE DATA temos que ele continua usando ObjectOutputStream e ObjetInputStream ; usa o ObjectInputStream no receptor e continua sem transmissor** . Eu declaro um objeto obj da classe Object recebendo nulo. Ele fica num looping fazendo o obj receber receptor.readline (recebe o objeto que pode ser a data ou a string fim ou qualquer outra coisa) e depois printa o obj. Eu sei que vai receber 4 datas e depois o fim porque eu fiz o outro programa. Ele fica no loop do senão for verdade que o obj (o que foi recebido pelo readObject do receptor) é uma string. O instanceof serve para testar tipo; enquanto não for string fica no do. Se for string, enxergando o obj como string, transformando em letras maiúsculas é igual a palavra fim o programa para. Então o loop continua senão for string ou se a string recebida não for fim.

Nesse programa, quando receber a string Fim do Proativo ele vai parar de executar.

**Para a CLASSE PROATIVO DE DATA temos que não tem receptor, pois apenas transmissor.** E no transmissor usa o ObjectOutputStream . Declarou 4 objetos da classe Data: pai, mãe, eu e filho e instanciei esses objetos com new Data. No primeiro colocou a data de aniversário do pai (8/5/1939), no segundo o da mãe (16/3/1943), no terceiro o dele (19/1/1966) e no quarto o do filho (29/6/1992). Criei 4 datas: pai, mãe, eu e filho.

**Em seguida, tem um print com a mensagem de apertar o enter para enviar a primeira data; programa espera eu digitar algo e dar enter por conta do teclado.readLine.** Quando der enter, vai executar o comando (transmissor.writeObject (pai)) enviando a data de nascimento do pai pro reativo; envia imediatamente,

**Printa pressione enter para enviar a segunda data, espera digitar a data da mãe e dar enter, envia pro reativo com envio imediato. Printa pressione enter para enviar a terceira data, espera digitar a data dele e dar enter, envia pro reativo com envio imediato.** Depois, printa pressione enter para enviar a quarta e última data, espera digitar a data filho e dar enter, envia pro reativo com envio imediato. Depois escreve fim e envia. Assim que o fim for enviado, fecha o transmissor, fecha a conexão e programa proativo termina. Os primeiros writeObject mandaram data (4 datas) o último mandou fim.

**No proativo printou todas as datas mas terminou tão rápido que nem deu tempo de ler a 4ª data.** Se eu quiser, posso acrescentar antes de enviar fim para terminar o programa, assim antes de terminar o programa ele espera o usuário apertar enter e antes dele a última data já foi printada.

System.out.print (“ Pressione ENTER para terminar o programa”\_);

teclado.readLine ();

transmissor.writeObject ("FIM");

transmissor.flush (); // para envio imediato

**6.1.4.ARQUIVO: REATIVO DA DATA.JAVA**

import java.io.\*;

import java.net.\*;

public class Reativo

{

public static void main (String[] args)

{

try

{

ServerSocket pedido = new ServerSocket (12345);

// fio do telefone fixo preso na parede

Socket conexao = pedido.accept();

/\*

// tranmissor sempre antes do receptor

ObjectOutputStream transmissor =

new ObjectOutputStream (

conexao.getOutputStream ()); // microfone

\*/

//receptor sempre depois do transmissor

ObjectInputStream receptor =

new ObjectInputStream (

conexao.getInputStream ()); // autofalante

Object obj=null;

do

{

obj = receptor.readObject();

System.out.println (obj);

}

while (!(obj instanceof String) || !((String)obj).toUpperCase().equals ("FIM"));

/\*

transmissor.close();

\*/

receptor.close();

conexao.close();

pedido.close();

}

catch (Exception erro)

{

System.err.println (erro.getMessage());

}

}

}

**6.1.5.Arquivo: PROATIVO DO DATA.JAVA**

import java.io.\*;

import java.net.\*;

public class Proativo

{

public static void main (String[] args)

{

try

{

// fio do telefone fixo preso na parede

Socket conexao = new Socket ("localhost",12345);

// tranmissor sempre antes do receptor

ObjectOutputStream transmissor =

new ObjectOutputStream (

conexao.getOutputStream ()); // microfone

/\*

//receptor sempre depois do transmissor

ObjectInputStream receptor =

new ObjectInputStream (

conexao.getInputStream ()); // autofalante

\*/

BufferedReader teclado =

new BufferedReader (

new InputStreamReader (

System.in));

Data pai = new Data ( 8, 5, 1939);

Data mae = new Data (16, 3, 1943);

Data eu = new Data (19, 1, 1966);

Data filho = new Data (29, 6, 1992);

System.out.print ("Pressione ENTER para enviar a primeira data");

teclado.readLine ();

transmissor.writeObject (pai);

transmissor.flush (); // para envio imediato

System.out.print ("Pressione ENTER para enviar a segunda data");

teclado.readLine ();

transmissor.writeObject (mae);

transmissor.flush (); // para envio imediato

System.out.print ("Pressione ENTER para enviar a terceira data");

teclado.readLine ();

transmissor.writeObject (eu);

transmissor.flush (); // para envio imediato

System.out.print ("Pressione ENTER para enviar a quarta e ultima data");

teclado.readLine ();

transmissor.writeObject (filho);

transmissor.flush (); // para envio imediato

System.out.print ("Pressione ENTER para terminar o programa");

teclado.readLine ();

transmissor.writeObject ("FIM");

transmissor.flush (); // para envio imediato

transmissor.close();

/\*

receptor.close();

\*/

conexao.close();

}

catch (Exception erro)

{

System.err.println (erro.getMessage());

}

}

}

**6.1.6.Arquivo: DATA COM SERIALIZABLE.JAVA**

import java.io.Serializable;

public class Data implements Serializable

{

private int dia, mes, ano;

private boolean bissexto (int ano)

{

if (ano%400==0)

return true;

if (ano%4==0 && ano%100!=0)

return true;

return false;

}

private boolean valida (int dia, int mes, int ano)

{

if (dia<1 || dia>31)

return false;

if (mes<1 || mes>12)

return false;

if ((mes==4 || mes==6 || mes==9 || mes==11) && dia>30)

return false;

if (dia>29 && mes==2)

return false;

if (dia>28 && mes==2 && !bissexto(ano))

return false;

return true;

}

public Data (int dia, int mes, int ano) throws Exception

{

if (!valida(dia,mes,ano))

throw new Exception ("Data invalida");

this.dia = dia;

this.mes = mes;

this.ano = ano;

}

public void setDia (int dia) throws Exception

{

if (!valida(dia,this.mes,this.ano))

throw new Exception ("Data invalida");

this.dia=dia;

}

public void setMes (int mes) throws Exception

{

if (!valida(this.dia,mes,this.ano))

throw new Exception ("Data invalida");

this.mes=mes;

}

public void setAno (int ano) throws Exception

{

if (!valida(this.dia,this.mes,ano))

throw new Exception ("Data invalida");

this.ano=ano;

}

public int getDia ()

{

return this.dia;

}

public int getMes ()

{

return this.mes;

}

public int getAno ()

{

return this.ano;

}

// solucao conservadora, pois nao crio data sem saber que é valida

// solucao "gastona", pois valida 2 vezes, para fazer new e no contrutor quando ja fez new

// é importante ressaltar que poderíamos ter adotado a abordagem ousada adotada no proximo

// método para fazer esse metodo; fizemos cada um com uma abordagem diferente apenas para

// ilustrar essa abordagens.

public Data diaSeguinte ()

{

Data ret=null;

try

{

if (valida(this.dia+1,this.mes,this.ano))

ret = new Data (this.dia+1,this.mes,this.ano);

else

if (valida(1,this.mes+1,this.ano))

ret = new Data (1,this.mes+1,this.ano);

else

ret = new Data (1,1,this.ano+1);

}

catch (Exception erro)

{} // sei que nao vai dar erro

return ret;

}

// solucao ousada, pois vai tentando criar datas que talvez estejam erradas e

// no tratamento das excecoes, faz novas tentativas que tambem podem estar erradas

// solucao economica, pois nao valido antes de criar new Data; confio na excecao

// que o new Data me der

// é importante ressaltar que poderíamos ter adotado a abordagem conservadora adotada no

// método anterior para fazer esse metodo; fizemos cada um com uma abordagem diferente

// apenas para ilustrar essa abordagens.

public Data diaAnterior ()

{

Data ret = null;

try {

ret = new Data (this.dia-1,this.mes,this.ano);

}

catch (Exception a) {

try {

ret = new Data (31,this.mes-1,this.ano);

}

catch (Exception b) {

try {

ret = new Data (30,this.mes-1,this.ano);

}

catch (Exception c) {

try {

ret = new Data (29,this.mes-1,this.ano);

}

catch (Exception d) {

try {

ret = new Data (28,this.mes-1,this.ano);

}

catch (Exception e) {

try {

ret = new Data (31,12,this.ano-1);

}

catch (Exception f)

{} // sei que nao vai dar erro

}

}

}

}

}

return ret;

}

// 1 dos 3 metodos obrigatorios SEMPRE

public String toString ()

{

return (this.dia<10?"0":"")+

this.dia+

"/"+

(this.mes<10?"0":"")+

this.mes+

"/"+

(this.ano<0?-this.ano:this.ano)+

(this.ano<0?"AC":"");

}

// 1 dos 3 metodos obrigatorios SEMPRE

// serve para comparar this com obj

public boolean equals (Object obj)

{

if (this==obj)

return true;

if (obj==null)

return false;

//if (!(obj instanceof Data))

//if (obj .getClass() != class)

if (this.getClass() != obj.getClass())

return false;

//if (this.dia != ((Data)obj).dia ||

// this.mes != ((Data)obj).mes ||

// this.ano != ((Data)obj).ano)

// return false;

Data data = (Data)obj;

if (this.dia != dia ||

this.mes != mes ||

this.ano != ano)

return false;

return true;

}

}

**6.1.7.Arquivo: 1ª versão de Proativo.java**

import java.io.\*;

import java.net.\*;

public class Proativo

{

public static void main (String[] args)

{

try

{

// fio do telefone fixo preso na parede

Socket conexao = new Socket ("localhost",12345);

/\*BufferedReader receptor =

New BufferedReader( // passo pro construtor de BufferedReader o new InputStreamReader

New InputStreamReader ( // passo pro construtor de InputStreamReader o conexão.getInputStream ()

conexão.getInputStream () ) ); // autofalante ; 🡪 conexão é o meu socket (associa ao fio) para criar o autofalante (receptor) \*/

PrintWriter transmissor =

New PrintWriter (

conexão.getOutputStream ()); // Microfone

BufferedReader teclado = new BufferedReader (new InputStreamReader (System.in)) ;

String texto;

do

{

texto = teclado.readLine ();

transmissor.println (texto);

transmissor.flush ()/ // para envio imediato

}

While (!texto.toUpperCase ().equals (“FIM”));

transmissor.close ();

//receptor.close ();

conexão.close ();

***6.1.8.Pasta 1)Só texto***

**6.1.8.1.Arquivo: Proativo.java**

import java.io.\*;

import java.net.\*;

public class Proativo

{

public static void main (String[] args)

{

try

{

// fio do telefone fixo preso na parede

Socket conexao = new Socket ("localhost",12345);

/\*

BufferedReader receptor =

new BufferedReader (

new InputStreamReader (

conexao.getInputStream ())); // autofalante

\*/

PrintWriter transmissor =

new PrintWriter (

conexao.getOutputStream ()); // microfone

BufferedReader teclado =

new BufferedReader (

new InputStreamReader (

System.in));

String texto;

do

{

texto = teclado.readLine ();

transmissor.println (texto);

transmissor.flush (); // para envio imediato

}

while (!texto.toUpperCase().equals ("FIM"));

transmissor.close();

/\*

receptor.close();

\*/

conexao.close();

}

catch (Exception erro)

{

System.err.println (erro.getMessage());

}

}

}

++++++++++++++++++++++++++++++++++++++

**6.1.8.2.Arquivo: Reativo.java**

import java.io.\*;

import java.net.\*;

public class Reativo

{

public static void main (String[] args)

{

try

{

ServerSocket pedido = new ServerSocket (12345);

// fio do telefone fixo preso na parede

Socket conexao = pedido.accept(); // 1 ponta do fio

BufferedReader receptor =

new BufferedReader (

new InputStreamReader (

conexao.getInputStream ())); // autofalante

/\*

PrintWriter transmissor =

new PrintWriter (

conexao.getOutputStream ()); // microfone

\*/

String texto;

do

{

texto = receptor.readLine ();

System.out.println (texto);

}

while (!texto.toUpperCase().equals ("FIM"));

/\*

transmissor.close();

\*/

receptor.close();

conexao.close();

pedido.close();

}

catch (Exception erro)

{

System.err.println (erro.getMessage());

}

}

}

-----------------------------------------------

***6.1.9.Pasta 2) Objetos***

**6.1.9.1.Arquivo: Proativo.java**

import java.io.\*;

import java.net.\*;

public class Proativo

{

public static void main (String[] args)

{

try

{

// fio do telefone fixo preso na parede

Socket conexao = new Socket ("localhost",12345);

// tranmissor sempre antes do receptor

ObjectOutputStream transmissor =

new ObjectOutputStream (

conexao.getOutputStream ()); // microfone

/\*

//receptor sempre depois do transmissor

ObjectInputStream receptor =

new ObjectInputStream (

conexao.getInputStream ()); // autofalante

\*/

BufferedReader teclado =

new BufferedReader (

new InputStreamReader (

System.in));

String texto;

do

{

texto = teclado.readLine ();

transmissor.writeObject (texto);

transmissor.flush (); // para envio imediato

}

while (!texto.toUpperCase().equals ("FIM"));

transmissor.close();

/\*

receptor.close();

\*/

conexao.close();

}

catch (Exception erro)

{

System.err.println (erro.getMessage());

}

}

}

+++++++++++++++++++++++++

**6.1.9.2.Arquivo: Reativo.java**

import java.io.\*;

import java.net.\*;

public class Reativo

{

public static void main (String[] args)

{

try

{

ServerSocket pedido = new ServerSocket (12345);

// fio do telefone fixo preso na parede

Socket conexao = pedido.accept();

/\*

// tranmissor sempre antes do receptor

ObjectOutputStream transmissor =

new ObjectOutputStream (

conexao.getOutputStream ()); // microfone

\*/

//receptor sempre depois do transmissor

ObjectInputStream receptor =

new ObjectInputStream (

conexao.getInputStream ()); // autofalante

String texto;

do

{

texto = (String)receptor.readObject ();

System.out.println (texto);

}

while (!texto.toUpperCase().equals ("FIM"));

/\*

transmissor.close();

\* \*/

receptor.close();

conexao.close();

pedido.close();

}

catch (Exception erro)

{

System.err.println (erro.getMessage());

}

}

}

-----------------------------------------

**6.2.*Pasta 3) Datas***

**6.2.1.Arquivo: Data.java**

import java.io.Serializable;

public class Data implements Serializable

{

private int dia, mes, ano;

private boolean bissexto (int ano)

{

if (ano%400==0)

return true;

if (ano%4==0 && ano%100!=0)

return true;

return false;

}

private boolean valida (int dia, int mes, int ano)

{

if (dia<1 || dia>31)

return false;

if (mes<1 || mes>12)

return false;

if ((mes==4 || mes==6 || mes==9 || mes==11) && dia>30)

return false;

if (dia>29 && mes==2)

return false;

if (dia>28 && mes==2 && !bissexto(ano))

return false;

return true;

}

public Data (int dia, int mes, int ano) throws Exception

{

if (!valida(dia,mes,ano))

throw new Exception ("Data invalida");

this.dia = dia;

this.mes = mes;

this.ano = ano;

}

public void setDia (int dia) throws Exception

{

if (!valida(dia,this.mes,this.ano))

throw new Exception ("Data invalida");

this.dia=dia;

}

public void setMes (int mes) throws Exception

{

if (!valida(this.dia,mes,this.ano))

throw new Exception ("Data invalida");

this.mes=mes;

}

public void setAno (int ano) throws Exception

{

if (!valida(this.dia,this.mes,ano))

throw new Exception ("Data invalida");

this.ano=ano;

}

public int getDia ()

{

return this.dia;

}

public int getMes ()

{

return this.mes;

}

public int getAno ()

{

return this.ano;

}

// solucao conservadora, pois nao crio data sem saber que é valida

// solucao "gastona", pois valida 2 vezes, para fazer new e no contrutor quando ja fez new

// é importante ressaltar que poderíamos ter adotado a abordagem ousada adotada no proximo

// método para fazer esse metodo; fizemos cada um com uma abordagem diferente apenas para

// ilustrar essa abordagens.

public Data diaSeguinte ()

{

Data ret=null;

try

{

if (valida(this.dia+1,this.mes,this.ano))

ret = new Data (this.dia+1,this.mes,this.ano);

else

if (valida(1,this.mes+1,this.ano))

ret = new Data (1,this.mes+1,this.ano);

else

ret = new Data (1,1,this.ano+1);

}

catch (Exception erro)

{} // sei que nao vai dar erro

return ret;

}

// solucao ousada, pois vai tentando criar datas que talvez estejam erradas e

// no tratamento das excecoes, faz novas tentativas que tambem podem estar erradas

// solucao economica, pois nao valido antes de criar new Data; confio na excecao

// que o new Data me der

// é importante ressaltar que poderíamos ter adotado a abordagem conservadora adotada no

// método anterior para fazer esse metodo; fizemos cada um com uma abordagem diferente

// apenas para ilustrar essa abordagens.

public Data diaAnterior ()

{

Data ret = null;

try {

ret = new Data (this.dia-1,this.mes,this.ano);

}

catch (Exception a) {

try {

ret = new Data (31,this.mes-1,this.ano);

}

catch (Exception b) {

try {

ret = new Data (30,this.mes-1,this.ano);

}

catch (Exception c) {

try {

ret = new Data (29,this.mes-1,this.ano);

}

catch (Exception d) {

try {

ret = new Data (28,this.mes-1,this.ano);

}

catch (Exception e) {

try {

ret = new Data (31,12,this.ano-1);

}

catch (Exception f)

{} // sei que nao vai dar erro

}

}

}

}

}

return ret;

}

// 1 dos 3 metodos obrigatorios SEMPRE

public String toString ()

{

return (this.dia<10?"0":"")+

this.dia+

"/"+

(this.mes<10?"0":"")+

this.mes+

"/"+

(this.ano<0?-this.ano:this.ano)+

(this.ano<0?"AC":"");

}

// 1 dos 3 metodos obrigatorios SEMPRE

// serve para comparar this com obj

public boolean equals (Object obj)

{

if (this==obj)

return true;

if (obj==null)

return false;

//if (!(obj instanceof Data))

//if (obj .getClass() != class)

if (this.getClass() != obj.getClass())

return false;

//if (this.dia != ((Data)obj).dia ||

// this.mes != ((Data)obj).mes ||

// this.ano != ((Data)obj).ano)

// return false;

Data data = (Data)obj;

if (this.dia != dia ||

this.mes != mes ||

this.ano != ano)

return false;

return true;

}

}

+++++++++++++++++++++++++++++++++++++++

**6.2.2.Arquivo: Proativo.java**

import java.io.\*;

import java.net.\*;

public class Proativo

{

public static void main (String[] args)

{

try

{

// fio do telefone fixo preso na parede

Socket conexao = new Socket ("localhost",12345);

// tranmissor sempre antes do receptor

ObjectOutputStream transmissor =

new ObjectOutputStream (

conexao.getOutputStream ()); // microfone

/\*

//receptor sempre depois do transmissor

ObjectInputStream receptor =

new ObjectInputStream (

conexao.getInputStream ()); // autofalante

\*/

BufferedReader teclado =

new BufferedReader (

new InputStreamReader (

System.in));

Data pai = new Data ( 8, 5, 1939);

Data mae = new Data (16, 3, 1943);

Data eu = new Data (19, 1, 1966);

Data filho = new Data (29, 6, 1992);

System.out.print ("Pressione ENTER para enviar a primeira data");

teclado.readLine ();

transmissor.writeObject (pai);

transmissor.flush (); // para envio imediato

System.out.print ("Pressione ENTER para enviar a segunda data");

teclado.readLine ();

transmissor.writeObject (mae);

transmissor.flush (); // para envio imediato

System.out.print ("Pressione ENTER para enviar a terceira data");

teclado.readLine ();

transmissor.writeObject (eu);

transmissor.flush (); // para envio imediato

System.out.print ("Pressione ENTER para enviar a quarta e ultima data");

teclado.readLine ();

transmissor.writeObject (filho);

transmissor.flush (); // para envio imediato

System.out.print ("Pressione ENTER para terminar o programa");

teclado.readLine ();

transmissor.writeObject ("FIM");

transmissor.flush (); // para envio imediato

transmissor.close();

/\*

receptor.close();

\*/

conexao.close();

}

catch (Exception erro)

{

System.err.println (erro.getMessage());

}

}

}

+++++++++++++++++++++++++++++++++++++++

**6.2.3.Arquivo: Reativo.java**

import java.io.\*;

import java.net.\*;

public class Reativo

{

public static void main (String[] args)

{

try

{

ServerSocket pedido = new ServerSocket (12345);

// fio do telefone fixo preso na parede

Socket conexao = pedido.accept();

/\*

// tranmissor sempre antes do receptor

ObjectOutputStream transmissor =

new ObjectOutputStream (

conexao.getOutputStream ()); // microfone

\*/

//receptor sempre depois do transmissor

ObjectInputStream receptor =

new ObjectInputStream (

conexao.getInputStream ()); // autofalante

Object obj=null;

do

{

obj = receptor.readObject();

System.out.println (obj);

}

while (!(obj instanceof String) || !((String)obj).toUpperCase().equals ("FIM"));

/\*

transmissor.close();

\*/

receptor.close();

conexao.close();

pedido.close();

}

catch (Exception erro)

{

System.err.println (erro.getMessage());

}

}

}

-----------------------------------------

**6.2.4.Arquivo: Anotações.txt**

telefones fixos

existe um cabo (fio) que sai de uma central telefônica e chega até a casa dos assinantes

central telefonica comuta

todo assinante dever ter um aparelho telefônico conectado ao fio que chega na sua casa

um aparelho telefônico tem 2 partes:

1) um microfone que você coloca na direção da boca para falar

2) um autofalante que você coloca na direção do ouvido para escutar

--------

programas que se comunicam pela rede

"fio" -> fio imaginário, a própria rede

"2 pontas do fio" -> 2 objetos da classe Socket

"microfone ligado a cada ponta do fio" -> objeto instanciado de uma das classes abaixo

associado ao socket:

PrintWriter ou PrintStream,

DataOutputStream,

ObjectOutputStream,

etc

"autofalante ligado a cada ponta do fio" -> objeto instanciado de uma das classes abaixo

associado ao socket:

BufferedReader,

DataInputStream,

ObjectInputStream,

etc

--------

arquitetura cliente/servidor

programa proativo / programa reativo

o reativo roda numa certa máquina

ip identifica uma máquina na rede

nome simbólico é traduzido para ip pelo DNS (domain name server)

pode acontecer de vários programas reativos estarem rodando numa máquina

por isso, na hora do proativo se conectar, nao basta saber em qual máquina se conectar

cada reativo adota uma porta e o proativo deve saber a máquina + porta

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**7. Pasta: Aula 20-10-2021**

**7.1.Subpasta: 2021-10-20.A**

**7.1.1.Pasta: 1)ThreadsSEMCompartilhamentoDeED**

**Arquivos: ThreadsSemCompartilhamentoDeEd.mp4**

**AULA 7- Thread SEM Compartilhamento De ED - Dia 20 Out 2021**

Hoje vai **explicar como fazer programas que executam várias tarefas ao mesmo tempo ou que pelo menos deem a impressão de estar executando várias tarefas ao mesmo tempo.** Para vocês se situarem nesse contexto que a gente vai estudar agora, gostaria de chamar a atenção na forma pela qual os programas que a gente já costuma fazer não só na nossa disciplina, mas já feitas nas outras disciplinas. Preste atenção na forma pela qual elas executam.

**Vamos pensar num programa feito em Java; a gente chama o programa, entra em execução o método main da classe executável do programa.** O método main chama algum outro método e ele para de executar; e o outro método começa a executar. O método chamado vai executando; pode ser que demore um pouco, enquanto que o método main está parado, esperando o método que ele chamou parar a execução.

**O método chamado vai executando e numa certa hora ele termina. Após o método terminar, a execução volta pro método main e a execução continua comando a comando, a partir de onde ela tinha parado.** Até que outro método seja chamado pelo método main e nessa hora vai acontecer a mesma coisa descrita anteriormente (main fica em segundo plano esperando o método chamado ser executado totalmente; para depois o main ser executado como principal).

**Normalmente, programas convencionais executam da forma descrita nos parágrafos anteriormente.** Mas, já devem ter usados programas não convencionais, embora nunca tenham feito. Por exemplo, pode ter usado ou visto alguém usar, um programa que é um jogo. Vamos imaginar um jogo que seja rpg online, na tela do seu jogo vários personagens eventualmente se movem ao mesmo tempo.

**Um dos personagens pode estar associado a você que é um dos jogadores; de forma que você mexeu o seu mouse, ou as suas teclas com setas; e aquele personagem se move porque responde aos seus comandos.** Mas, pode ter um monte de outros personagens na tela que se movem e não são comandados por você e sim outras pessoas que estão jogando.

**E todos esses personagens estão se movendo ao mesmo tempo, ou pelo menos dão a impressão de estarem se movendo ao mesmo tempo.** Um programa que funcione do mesmo jeito que o rpg é um programa que difere um bocado em termos de estrutura; daquele programa tradicional supracitado. E eu quero que vocês percebam que é comum a gente ter programas que executam (ou dão a impressão) de estarem executando várias coisas ao mesmo tempo.

**Imagine o programa do Google que guarda os e-mails do Gmail. Imagine que você e eu decidamos, no mesmo momento, que nós queremos ler e-mails.** O programa do google vai ter que realizar 2 coisas ao mesmo tempo (ou dar a impressão); porque tem mandar pra mim e pra você os nossos e-mail. E como a gente decidiu ler os e-mails no mesmo momento, o servidor do google deveria nos mandar ao mesmo tempo os nossos e-mails respectivos. É um outro exemplo de programa que faz várias coisas ao mesmo tempo. No caso do servidor do google, ele possivelmente faz (ou dá a impressão de fazer), de milhares a milhões de coisas a fazer ao mesmo tempo. Porque, muita gente nesse mundo pode resolver que vai ler e-mail no mesmo instante. E, se conectar no gmail, no servidor da google. O servidor tem que mandar os e-mails de cada um ao mesmo tempo, nem que seja para passar essa impressão.

**É o objetivo nosso nessa vídeo-aula , entender como é que acontecer para a gente fazer programas com esse comportamento de executar várias coisas ao mesmo tempo.** De uma maneira bem rudimentar, é como se o servidor do google chamasse um método para chamar os meus e-mails; como se a main do programa chamasse um método para mandar os meus e-mails. E na hora que você pedisse os seus e-mails, ela também chamasse um método para mandar pra vc os seus e-mails. E ela não espera o método que chamou pra mandar para mim os e-mails acabar a execução. Ela chamou o método pra chamar pra mim os meus e-mails e sem terminar esse método, ela chamou um segundo para mandar pra você os seus e-mails.

**Como esses dois métodos foram chamados, um deles manda e-mail para mim e outro manda para você.** O grande mistério é como que ela conseguiu chamar um método que manda e-mails para você se o método que ela chamou antes, que manda e-mails para mim, ainda não terminou de executar? Não estamos acostumados a ver esse tipo de coisa.

**Ele está sugerindo não esperar o método chamado primeiro terminar, e sim já chamar o segundo método em seguida.** Antes a começar a se preocupar com implementações, precisa ser esclarecido duas coisas que foram faladas que precisam ser detalhadas.

**Nós vamos fazer programas que fazem ou dão a impressão de fazer várias coisas ao mesmo tempo.** O que acontece é que imagine que eu tenha um computador e o meu computador tem 4 processadores ou núcleos de processamento; são processadores quad-core. Todo mundo já ouviu falar em octa-core, são processadores que tem oito núcleos de processamento.

**Existem processadores com vários núcleos de processamento: dual-core, quad-core, octa-core.** Se um processador tem 2 núcleos de processamento, significa que ele é capaz de realizar de verdade, duas coisas ao mesmo tempo. Porque um dos núcleos de processamento fica processando uma das coisas; e o segundo núcleo fica processando a outra. As duas estão em execução simultânea; isso é possível em processadores duais-core.

**Se o processador tiver 4 núcleos de processamento, eu consigo colocar em execução simultânea verdadeiramente 4 coisas.** E no caso do octa-core, terão 8 coisas executando de verdade simultaneamente. (14min15s).

**A gente sabe que os processadores eventualmente executam bem mais coisas ao mesmo tempo.** Por exemplo, você já deve ter jogado um joguinho que tem dezenas de personagens se movimentando ao mesmo tempo; porque tem dezenas de jogadores jogando juntos em rede e os personagens estão todos agindo ao mesmo tempo. Tem muito mais do que 8 jogadores.

**Você já deve ter colocado pra executar vários programas ao mesmo tempo. (15min14s). Você já deve ter colocado um reprodutor de música para ouvir um som e ter aberto também um navegador pra baixar alguma coisa e também ter aberto um ambiente de programação para fazer um exercício de programação.** E todas essas coisas estão supostamente acontecendo ao mesmo tempo: enquanto programa, a música está sendo tocada e o download está acontecendo no navegador.

**E a gente já deve ter feito isso com muitos programas; e não apenas 3. Pode ser que um dia você já deve ter ficado ansioso para baixar uma série de coisas que precisava baixar; e de uma vez só você baixou 10,15,20 coisas ao mesmo tempo.** E como se isso não bastasse você ainda colocou uma música pra tocar; abriu o ambiente de programação e foi programar. Ou seja, tem uma dezena de coisas acontecendo ao mesmo tempo ou pelo menos parece acontecer. Porém, agora você sabe que, se o seu processador tem dois núcleos, apenas 2 coisas acontecem de verdade ao mesmo tempo. Se ele tem 4 núcleos, são 4 coisas acontecendo ao mesmo tempo; se tiver 8 são oito coisas acontecendo ao mesmo tempo.

**Mas como assim, se eu coloquei dezena de coisas para acontecer ao mesmo tempo e parece estar acontecendo?** O que acontece é que quando você coloca para executar vários programas ou, um programa só coloca várias coisas pra acontecer ao mesmo tempo. Note que isso não faz muita diferença. Se você roda vários programas, os vários programas estão rodando ao mesmo tempo. Se você roda um programa, esse programa tem um bando de personagens se mexendo ao mesmo tempo, só esse programa já está fazendo várias coisas ao mesmo tempo.

**Então, não faz muita diferença se as coisas que estão acontecendo ao mesmo tempo estão no mesmo programa ou em programas separados.** O fato é que tem várias coisas acontecendo ao mesmo tempo. E a quantidade de coisas acontecendo ao mesmo tempo é limitada pela quantidade de processadores. De repente, você tem mais coisas acontecendo do que processadores na sua máquina. Como que isso acontece? Isso acontece da seguinte maneira: quando você coloca muita coisa pra rodar ao mesmo tempo. Vamos supor que estamos falando de um processador que tenha 4 núcleos de processamento e digamos que eu coloquei pra rodar oito coisas, mas o meu processador só tem 4. Como que acontece? Quatro das oito coisas que eu pus pra rodar, rodam um pouquinho; uns milissegundos. Eles param e as outras quatro coisas começam a rodar um pouco, alguns milissegundos. Depois, roda o primeiro conjunto e assim por diante.

**Suponha que a gente tivesse nessa situação do quad-core; se eu fizesse o seguinte: colocasse oito coisas para rodar: a,b,c,d,e,f,g,h.** Vai acontecer que vão ser feito dois grupos: o primeiro com a,b,c,d; o segundo grupo com e,f,g,h. O primeiro grupo entra em execução ocupando os 4 processadores; executa um tempo pequeno, para. O segundo grupo executa um pouco, para e volta para executar o primeiro. Eles ficam revezando. Porque o processador não tem muitos núcleos de processamento e você colocou mais coisa do que ele aguenta rodar.

**Imagine que eu imagine mais coisas: a,b,c,d,e,f,g,h,i,j,k,l,m,n,op,q,r,s,t,u,v,w,x. Iria acontecer que iria formar um terceiro grupo (i,j,k,l). Então teriam três grupos, o primeiro (a,b,c,d), segundo (e,f,g,h), terceiro (i,j,k,l).** E todos iriam rodar por um certo tempo e parar para que o próximo pudesse ser executado, do primeiro ao terceiro. Sempre se revezando. Se eu for exagerando e for colocando cada vez mais grupos: 4º grupo (m,n,o,p), 5º grupo (q,r,s,t), 6º grupo (u,v,w,x). Digamos que eu tenho esses 6 grupos postas para executar: são 24 coisas para ser usadas em 4 processadores. O que vai acontecer é cada um reveza nos grupos que vão de 1 a 6, executa um pouco e para. Depois o grupo seguinte entra em execução por determinado tempo.

**Agora tenho muitos grupos, e o primeiro grupo parou faz tempo; porque os demais já foram executados. Estou na execução do grupo 6.** Quando a gente exagera muito e a máquina não é tão boa, a gente começa a perceber que as coisas não estão rodando de fato ao mesmo tempo. Tem alguns downloads que andam e outros ficam parados; depois os parados andam. A gente percebe porque quem parou primeiro (grupo 1) já está parado há muito tempo. Se a máquina não for muito boa e eu exagero na quantidade de coisas que eu ponho pra rodar, quando estiver no grupo 6, pode ter grupos que estão parados há muito tempo e nós percebemos essa demora.

**Não importa a quão rápida seja a máquina, se eu aumentar no exagero de coisas a serem executadas, mesmo máquinas melhores vão começar a deixar a gente perceber o truque de revezar atividades**, já que a máquina não tem tanto processamento assim. Senão exagerar tanto, como por exemplo deixar apenas 2 grupos; não desse para perceber esse truque; pois nenhum dos dois grupos esperam muito tempo.

**Máquina boa tem a ver com processador, memória, hd, ssd e afins; principalmente o processador para ver eficiência da máquina. A** gente diz que quando coisas estão sendo executadas de verdade, a gente diz que está acontecendo paralelismo; as coisas A,B.C,D não concorrem entre si, elas rodam simultaneamente de verdade porque no quad-core tem 4 núcleos e aqui temos 4 atividades/programas/coisas.

**E quando coisas não acontecem ao mesmo de verdade, mas dão a impressão de que está acontecendo ao mesmo tempo, chamamos de concorrência.** Porque o primeiro grupo concorre com o segundo para rodar no processador. É estabelecido um tempo para cada grupo rodar e fica revezando. Logo, quando um grupo está dentro do seu tempo e executando, o outro grupo está parado esperando a vez, os dois grupos concorrem para rodar.

**Agora vamos ver como programar isso que foi explicado anteriormente dentro de um mesmo programa.** Quando a coisa é entre dois programas rodando ao mesmo tempo, não é problema nosso é problema do sistema operacional: Linux, Windows fazer acontecer o que precisa acontecer.

**Quando a história de rodar várias coisas ao mesmo tempo: 2,3,4, acontece dentro de um programa só e é a gente que faz aquele programa, daí é problema nosso e temos que pensar em como fazer isso acontecer**. Vocês vão perceber que não é uma coisa muito complicada.

**A primeira coisa que deve saber é que um programa faz 1 coisa., que é o que a main manda fazer.** Quando você quer um programa que faça várias coisas ao mesmo tempo, já que tem a main fazendo 1 coisa.; você vai ter que se preocupar em fazer mais 1,2,3. Porque uma já tem que é a main fazendo o que ela está preparada pra fazer. Como vou programar cada uma dessas coisas a mais?

**Essas várias coisas que rodam ao mesmo tempo são chamadas de tarefas. Então, como eu programo uma tarefa, que é uma coisa que vai executar ao mesmo tempo que a main? Como eu programo uma tarefa?** Basicamente a gente faz uma classe que herda da classe Thread. (verifique a TarefaDoTipo1.java). Thread é uma classe que já existe pronta em java. Depois, implementa o método run que é um método que serve pra dizer o que essa segunda coisa que vai rodar junto com a main vai fazer. A gente pode entender que o método run fosse uma segunda main. Porque a main vai rodar ao mesmo tempo que o run; é como se o run fosse uma segunda main que vai ficar rodando junto com a primeira main que já tinha; cada um fazendo uma coisa diferente ao mesmo tempo.

**Então tem que implementar o método void run, e dentro dele você implementa o que quer que aconteça ao mesmo tempo que** a main está rodando, e talvez ao mesmo tempo que outras tarefas possam estar rodando. Logo, você programa o que deseja que rode dentro de run.

**Dentro de public void run (), temos que: declarou um caractere e pôs nele o valor inicial de ‘a’.** Antes do método run (fora dele) tem a variável booleana fim que começa com falso. A variável fim expressa o fim da minha tarefa, e é falso que ela já acabou. Dentro do método run, tem o caractere a e uma repetição de enquanto não for o fim desta tarefa que nós estamos programando (while (!this.fim) ). Vai printar na tela o caractere, faz um try catch pedindo para esperar 500 milissegundos, só para não printar muito rápido e não conseguir nem ler. No try ele faz a tarefa dormir .Depois do sleep, ela continua executando.

**Se o caractere for z, o caractere vira a. Senão, eu vou pegar o caractere que é a, vou fazer o typecast pra converter o caractere num int e eu vou conseguir o código do caractere (ascii) , depois somo 1 no código do caractere A, ao qual se tornou o código do caractere b. Então: ((int) caractere) + 1 é o código do caractere b.** Depois, converto pra char fazendo typecast. Agora eu tenho o caractere b sendo guardado na variável caractere. Como estou em looping em uma repetição: ele printa o caractere b, espera 500 milissegundos, verifica se já chegou no z e entra no else para virar d ; depois o ciclo se repete ao printar d. Quando o else virar z, ele printa o caractere z, dá o tempo de 500 millisegundos, verifica se o caractere é z, e como é ele vira a. (38min14s).

**Esse método run ele printa na tela letras minúsculas de a até z; e depois começa de novo o mesmo ciclo do alfabeto.** É claro que ele programou no arquivo TarefaDoTipo1.java esse run simples para servir de exemplo, mas pode escrever qualquer coisa que queira acontecer ao mesmo tempo. Se tivesse programando um jogo, poderia estar programado aqui no run o que um dos personagens precisaria fazer. E ele faz um looping, ao mesmo tempo enquanto outras tarefas fazem outras coisas.

**O que você programar na run vai de cada programa, você vai ver qual é o caso da situação. Eu fiz antes do run, o método morra, que é quando o programa termina, isto é, o fim é verdadeiro.** Se lá na main fizer a chamada do método morra, vai fazer o fim se tornar true. O loop para, o método run vai acabar e a tarefa vai parar/morrer/acabar. Então, se eu chamar o método morra, a tarefa morre; porque o fim se torna true o que interrompe o while de dentro de run. E quando o while para o run acaba; o que estava em execução simultânea com outras coisas, acaba.

**Na classe TarefaDoTipo2 (ver arquivo: TarefaDoTipo2.java) ele é muito parecida com a TarefaDoTipo1; o que muda é que o caractere usa letras maiúsculas. Então, atualmente eu tenho uma tarefa que printa letras minúsculas (tarefa do tipo 1) e outra que printa maiúsculas (tarefa do tipo 2).** O jeito de fazer as tarefas do tipo 1 e 2 são mais simples; diferente do modo que foi feito a tarefa do tipo 3.

**Se o jeito que foi mostrado nas tarefas do tipo 1 e 2 são preferíveis, porque mostrar do outro jeito que não é o mais preferível?** Como já estudamos herança, uma classe só pode herdar de uma outra classe. Ela não pode herdar de 2 classes. Então, digamos que eu precisasse fazer a classe que herde de outra classe que não a natural. (Preciso herdar de outra classe que não seja a thread), porque a lógica do meu programa exige a demanda de outras classes que não a thread prevista. Complicou tudo, porque se ela herda de uma outra, ela não vai poder herdar do thread. Já que em Java, classe herda de uma única classe e nunca de duas classes.

**Assim sendo, nessa situação eu vou escolher outro jeito de fazer tarefas que rodam simultaneamente ou aparentam rodar. Esse jeito da tarefa 1 e 2 é bem simples**. Mas eventualmente vou ter que escolher um jeito mais complicado. Eu estou numa situação onde a minha classe herda de uma outra e não pode herdar da classe thread ao mesmo tempo, isso não funciona. Então, realmente vou ter que escolher o outro jeito.

**A tarefa do tipo 3 (verifique o arquivo: TarefaDoTipo3.java) ela implementa a classe Runnable.** Ela não herda da classe Thread; mas ela também herda com a palavra implements da interface Runnable. Então, se a classe precisasse herdar de alguma outra classe, seria escrito no seu cabeçalho :

Public class TarefaDoTipo3 implements Runnable extends SeiLaQual

**Dessa forma, a classe herdava de quem precisa; e a implementação (implements) me garante a execução simultânea com demais programas**; tal como na tarefa 2 em seu cabeçalho o extends Threads garante a execução simultânea.

ELE NÃO ACEITA:

Public class TarefaDoTipo3 extends SeiLaQual, Thread;

**A minha saída já que realmente precisa que a classe TarefaDoTipo3 herde da classe SeiLaQual, foi fazer outra alternativa: herda de quem precisa herdar (com extends) e antes disso, coloque o implements Runnable.**

**Depois dentro da classe TarefaDoTIpo3, declaro um atributo do tipo Thread. A minha tarefa não herda da classe Thread; porém dentro da minha classe eu tenho um atributo do tipo thread. Esse atributo é instanciado como uma nova thread e passa como parâmetro o this.** Esse construtor de thread recebe o parâmetro this; esse parâmetro esperado é um parâmetro do tipo runnable. O this é o objeto tarefa da classe TarefaDoTipo3. Ele vai aceitar esse this? Sim, porque TarefaDoTipo3 herda de runnable; então a TarefaDoTIpo3 é runnable e por isso pode ser passado no construtor de thread. Após, declara o atributo booleano fim recebendo falso.

**O primeiro método é o public void start , no qual chama a tarefa do objeto this para ser começada pelo método start; mas será explicada depois.** Após isso, o método morra para deixar o fim verdadeiro e o terceiro método public void run que lê caracteres de 0 a 9 .

**O que a classe TarefaDoTipo3 possui de diferente é o implements Runnable (NÃO TEM EXTENDS, foi para justificar o modo mais complicado), o atributo tarefa da classe Thread;** o método start e no método run o caractere ser números de 0 a 9.

**Agora tem que mostrar como que tudo isso é posto para executar. Vamos voltar para o Programa.java, que é onde tem a Main.** Na main ,tem uma sequência de instruções a serem printadas para quem for usar o programa. Então temos que:

* Somente a main está rodando;
* Tecle enter para ativar a tarefa 1;
* Tecle novamente enter para ativar a tarefa 2;
* Tecle novamente enter para ativar a tarefa 3;
* Tecle novamente enter para desativar a tarefa 3;
* Tecle novamente enter para desativar a tarefa 2;
* Tecle novamente enter para desativar a tarefa 1;

**Depois, pede que o usuário digite uma string que seja captada pelo objeto teclado. Normalmente, seria feito: String x = Teclado.getUmString (); porque getUmString é uma função que retorna uma string e precisa ser armazenada em um objeto.** Porém, no caso em questão, não foi construído o objeto x da classe String. Porque eu chamo a função e não guardo retorno. Eu chamo a função como se fosse procedimento, como um void sem retorno. Quando eu chamo uma função desse jeito, basicamente acontece que a função vai retornar uma string e não vai servir pra nada. Eu faço isso para fazer o programa parar, pois quando manda escrever alguma coisa o programa para esperando a resposta do usuário para digitar.

**A main printa todas as instruções e depois fica parada esperando você digitar. Na hora que der o primeiro enter ela continua porque está parada no teclado.getUmaString (); declara a TarefaDoTIpo1 t1 = new TarefaDoTipo1 () e começa o objeto t1 pelo método start. (chama o método t1.start).** Na classe TarefaDoTipo1 não tem o método start; então como eu pude chamar o método start sem dar erro? Isso vai dar certo porque como a classe TarefaDoTipo1 herdou de Thread, ela herdou um monte de métodos de thread, como por exemplo o método start. E por causa dessa herança que na main pôde chamar o método start.

**Na hora que eu chamo t1.start; quando chamava um método de um objeto, normalmente o que acontecia?** Acontecia que a main parava e ficava parada esperando até terminar o método. Só que o método start é muito especial porque ele coloca o método run para executar em execução simultânea e ele termina. Na hora em que ele termina, a main continua. Porém, ele colocou o método run para executar em execução simultânea. A main está rodando junto com o método run da TarefaDoTipo1 t1, porque o start faz isso (põe o método run para rodar ao mesmo tempo dos outros métodos que já estejam rodando).

**Quando o start terminar, a main continua junto com o run. Quando chegar no teclado.getUmaStirng() a main vai ficar rodando, porém vai ficar parada.** Porque o comando de getUmaString () pede a digitação de uma string.: pede enter . Quando der o enter, a main continua a executar, junto com as tarefas do Tipo1t1, e instancia a TarefaDoTIpo2 t2. A TarefaDoTipo1 t1 imprime letras minúsculas enquanto a main espera digitar; quando dou enter, a main vai para a linha de criar o objeto t2 da classe TarefaDoTipo2 .

**Logo, vai estar rodando a TarefaDoTipo1 t1, enquanto que a main declara e instancia a TarefaDoTIpo2 t2 e chama o t2. Start**. Ao chamar o start, a main pôe pra rodar o run da TarefaDoTipo1 e também a run da TarefaDoTipo2 t2. Nesse momento, tem os runs das TarefaDoTipo1 (printa letras minúsculas) e TarefaDoTIpo t2 (printa letras maiúsculas). Como eu percebo o meu programa? Ele escreve intercalado letras maiúsculas com minúsculas pois rodam simultaneamente.

**E a main? Na hora em que ela chegar no terceiro Teclado.getUmString () , ela espera você digitar alguma coisa (além das tarefasDoTipo1 e 2 sendo executadas pelo run correspondentes).** Ao dar mais um enter, a main vai criar uma TarefaDoTipo3 t3, declara como new TarefaDoTIpo3 (e instancia). E chama o método t3.start. ().

**As tarefas do tipo 1 e 2 herdam de thread. Já a tarefa do tipo 3 ela não herda de thread e por isso não herdou o método start.** Por isso tem que fazer o método start para a TarefaDoTipo3 no qual chama o método start do atributo tarefa ao qual é o atributo que eu tenho do tipo thread. Na main, no objeto t3 parece ser a mesma coisa das demais tarefas mas não é. Pois nas tarefas do tipo 1 e 2 eu chamo métodos herdados da classe Thread, enquanto que na tarefa do tipo 3 é um método construído para start. (O método start que eu fiz chama o método start do objeto tarefa que é construído pela classe Thread; a tarefa é um atributo da TarefaDoTipo3). A tarefa do tipo 3 escreve números de 0 a 9.

**Depois de t3.start; o meu programa tem 4 coisas rodando ao mesmo tempo: as três tarefas e a main.** Assim sendo, a main chega no Teclado.getumSting () e espera a digitação; ao demorar pra digitar, você percebe sendo escrito as tarefas em simultâneo. Ao digitar o enter, a main continua e chama o método t3.morra para que a tarefa 3 termine (fim passa a ser verdadeiro, o run termina, a tarefa termina). Logo, nesse momento para de escrever dígitos.

**Depois disso, a main espera eu digitar um enter, enquanto eu observo na tela as letras maiúsculas e minúsculas sendo escritas.** Ao digitar enter, a tarefa 2 termina (de igual modo a tarefa1); agora a tela só terá letras minúsculas sendo escritas. Depois disso, a main espera eu digitar um enter, enquanto eu observo na tela as letras minúsculas sendo escritas. Ao teclar enter, a tarefa1 termina . Já não escreve mais nada.

Após isso, são printados:

* Somente a main está rodando; // igual no começo.
* Tecle enter para terminar o programa.

**Depois do último print, ele espera o usuário teclar enter com teclado.getUmaString()**

Por fim, a execução de programa.java ocorreu como esperado.

**7.2. Lista de Arquivos usados na aula 7**

**7.2.1.Arquivo: Programa.Java**

public class Programa

{

public static void main (String[] args)

{

System.out.println ("Somente a main esta rodando;");

System.out.println ("Tecle ENTER para ativar tarefa 1;");

System.out.println ("Tecle novamente ENTER para ativar tarefa 2;");

System.out.println ("Tecle novamente ENTER para ativar tarefa 3;");

System.out.println ("Tecle novamente ENTER para desativar tarefa 3;");

System.out.println ("Tecle novamente ENTER para desativar tarefa 2;");

System.out.println ("Tecle novamente ENTER para desativar tarefa 1.");

Teclado.getUmString();

TarefaDoTipo1 t1 = new TarefaDoTipo1 ();

t1.start ();

Teclado.getUmString();

TarefaDoTipo2 t2 = new TarefaDoTipo2 ();

t2.start ();

Teclado.getUmString();

TarefaDoTipo3 t3 = new TarefaDoTipo3 ();

t3.start ();

Teclado.getUmString();

t3.morra ();

Teclado.getUmString();

t2.morra ();

Teclado.getUmString();

t1.morra ();

System.out.println ("De novo, somente a main esta rodando;");

System.out.println ("Tecle ENTER para terminar o programa.");

Teclado.getUmString();

}

}

**+++++++++++++++++++++++++**

**7.2.2.Arquivo: TarefaDoTipo1.java**

public class TarefaDoTipo1 extends Thread

{

private boolean fim=false;

public void morra ()

{

this.fim=true;

}

public void run ()

{

char caractere='a';

while (!this.fim)

{

System.out.println (caractere);

try { this.sleep (500); } catch (Exception erro) {}

if (caractere=='z')

caractere = 'a';

else

caractere = (char)(((int)caractere)+1);

}

}

}

++++++++++++++++++++++++++++++++++++

**7.2.3.Arquivo: TarefaDoTipo2.java**

public class TarefaDoTipo2 extends Thread

{

private boolean fim=false;

public void morra ()

{

this.fim=true;

}

public void run ()

{

char caractere='A';

while (!this.fim)

{

System.out.println (caractere);

try { this.sleep (500); } catch (Exception erro) {}

if (caractere=='Z')

caractere = 'A';

else

caractere = (char)(((int)caractere)+1);

}

}

}

+++++++++++++++++++++++++++++++++++++++++

**7.2.4.Arquivo: TarefaDoTipo3.java**

public class TarefaDoTipo3 implements Runnable

{

private Thread tarefa = new Thread (this);

private boolean fim = false;

public void start ()

{

this.tarefa.start();

}

public void morra ()

{

this.fim=true;

}

public void run ()

{

char caractere='0';

while (!this.fim)

{

System.out.println (caractere);

try { this.tarefa.sleep (500); } catch (Exception erro) {}

if (caractere=='9')

caractere = '0';

else

caractere = (char)(((int)caractere)+1);

}

}

}

+++++++++++++++++++++++++++++++++++++++++++

**7.2.5.Arquivo: Teclado.java**

//import java.io.BufferedReader;

//import java.io.InputStreamReader;

//import java.io.IOException;

import java.io.\*;

public class Teclado

{

private static BufferedReader teclado =

new BufferedReader (

new InputStreamReader (

System.in));

public static String getUmString ()

{

String ret=null;

try

{

ret = teclado.readLine ();

}

catch (IOException erro)

{} // sei que nao vai dar erro

return ret;

}

public static byte getUmByte () throws Exception

{

byte ret=(byte)0;

try

{

ret = Byte.parseByte (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Byte invalido!");

}

return ret;

}

public static short getUmShort () throws Exception

{

short ret=(short)0;

try

{

ret = Short.parseShort (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Short invalido!");

}

return ret;

}

public static int getUmInt () throws Exception

{

int ret=0;

try

{

ret = Integer.parseInt (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Int invalido!");

}

return ret;

}

public static long getUmLong () throws Exception

{

//long ret=(long)0;

//long ret=0;

long ret=0L;

try

{

ret = Long.parseLong (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Long invalido!");

}

return ret;

}

public static float getUmFloat () throws Exception

{

//float ret=0;

//float ret=(float)0.0;

float ret=0.0F;

try

{

ret = Float.parseFloat (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Float invalido!");

}

return ret;

}

public static double getUmDouble () throws Exception

{

//double ret=0;

//double ret=(long)0;

//double ret=0L;

double ret=0.0;

try

{

ret = Double.parseDouble (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Double invalido!");

}

return ret;

}

public static char getUmChar () throws Exception

{

char ret=' ';

try

{

String str = teclado.readLine ();

if (str==null)

throw new Exception ("Char invalido!");

if (str.length() != 1)

throw new Exception ("Char invalido!");

ret = str.charAt(0);

}

catch (IOException erro)

{} // sei que nao vai dar erro

return ret;

}

public static boolean getUmBoolean () throws Exception

{

boolean ret=false;

try

{

String str = teclado.readLine ();

if (str==null)

throw new Exception ("Boolean invalido!");

if (!str.equals("true") && !str.equals("false"))

throw new Exception ("Boolean invalido!");

ret = Boolean.parseBoolean (str);

}

catch (IOException erro)

{} // sei que nao vai dar erro

return ret;

}

}

------------------------------------------

**7.3.Pasta: 2)ThreadsCompartilhandoED**

**Arquivos: ThreadCompartilhandoED.mp4**

**AULA 7 – Thread Compartilhando ED (aula 20 Out 2021, parte2)**

Hoje **vamos sofisticar o que já conhece sobre programação multitarefas**. Vai introduzir a ideia de ter uma estrutura de dados compartilhadas entre as diversas tarefas que a gente coloca pra executar através do método start. Para eu colocar para executar uma tarefa eu chamo o método start conforme visto anteriormente. Eu posso usá-lo em várias tarefas ( t1,t2,t3 e t4) e todas as tarefas as quais eu aplicar o método start entram em execução simultaneamente com a main. O que eu quero mostrar para vocês nessa videoaula é como ter uma estrutura de dados compartilhada por todas essas tarefas que estão em execução simultânea.

**Muitas aplicações da vida real necessitam que a gente tenha entre as tarefas uma estrutura, ou as vezes, diversas estruturas de dados compartilhadas.** E a gente precisa saber como é que a gente faz isso, é o objetivo dessa vídeo-aula. Então observe que na minha classe Programa (ver arquivo Programa.java) eu importo da biblioteca java.util.Vector ; eu importo a classe Vector que é uma estrutura de dados onde eu posso guardar várias informações.

**Vector é uma classe armazenadora de informações, sempre que eu declaro e instancio um objeto da Classe Vector, eu consigo armazenar informações, recuperar informações que eu armazenei.** Dentro do método main da cminha classe executável programa, eu declaro um Vector que contém /guarda Character e se chama armazenamento. Eu instancio o meu objeto armazenamento com um novo vector para guardar caracteres. (03min08s). E com isso eu tenho o objeto declarado e instanciado na main.

**Em seguida, são fornecidas algumas instruções dando o comando system.out.println:**

* Tecle ENTER para ativar as tarefas; // terão 4 tarefas.
* Tecle novamente ENTER para terminar o programa;

**Pede o método getUmString do teclado para capturar o enter. As tarefas são parecidas com algum incremento que a diferencie.** A tarefa 4 é completamente diferente. Depois têm os comandos que executam as 4 tarefas. Na main, as tarefas são criadas e instanciadas como se verifica abaixo. O parâmetro passado ao construtor é armazenamento; e depois cada objeto (t1,t,t3 e t4) tem o método start chamado para iniciar a run de cada tarefa.

TarefaDoTipo1 t1 = new TarefaDoTipo1 (armazenamento);

t1.start();

**Então, com 1 enter ativo as tarefas; depois espero outro enter para matar as tarefas de t4 a t1 pelo método morra.** Com isso, termina o programa pois a main vai terminar. Observe que a gente não tinha um construtor para tarefadotipo1 no vídeo anterior. Mas nesse vídeo atual nós temos um construtor, porque o vetor armazenamento está sendo passado como parâmetro. Isso evidencia a existência do construtor em todas as tarefas. O vetor foi instanciado na main, mas o objeto foi passado para todas as tarefas (quando criado objetos t1,t2,t3,t4) no formato de parâmetro para o construtor.

**Na primeira tarefa, temos o arquivo: TarefaDoTipo1.java . A classe TarefaDoTipo1 tem o extends thread (para herdar métodos). A novidade está em declarar o atributo private Vector <Character> armazenamento.** Observe que o atributo foi declarado, é um vector para guardar caractere chamado armazenamento, MAS não foi instanciado, uma vez que não estou fazendo uso daquele comando new (que é requerido para instanciar o objeto).

**Observe o construtor da tarefa 1, que recebe como parâmetro o objeto chamado armz, da classe Vector para guardar caracteres; ele recebe o objeto instanciado da main.** Porém, senão estiver instanciado na main vai ocorrer o lançamento da exceção. Ele lança a exceção de armazenamento ausente quando o parâmetro armz for nulo; isto é, quando o parâmetro armz não vier instanciado. Se vier instanciado , pego o parâmetro que eu recebi (armz) e guardo no atributo armazenamento (privativo).

**Mas eu não criei a instância no atributo privativo armazenamento devido a falta de new, mas ele está sendo instanciado no construtor quando recebe armz, mas sem o new.** Todo mundo deve lembrar que, em java, que objetos são ponteiros: variáveis capazes de guardar um endereço de memória.

**Voltando para a main, quando declarou o objeto armazenamento, ele declarou um ponteiro. Quando faz o objeto receber new Vector, o new Vector cria para mim uma instância de vector em algum local da memória que esteja livre no momento dessa criação.** Podemos entender que new Vector tem o efeito de solicitar ao sistema operacional uma área de memória livre. O sistema operacional vai procurar/descobrir uma área de memória livre e fornecer a região de memória para o new. O new retorna o endereço dessa região de memória. Então, quando uso o new retornou o endereço de memória onde foi criada essa nova instância de Vector. E eu guardei na variável armazenamento (que é capaz de guardar endereço de memória) o endereço da região de memória onde foi criada a instância de vector. É isso que significa declarar e instanciar um objeto.

**Quando declarei o objeto, eu declarei uma variável capaz de guardar endereço de memória, mas ainda não tinha dito esse endereço.** E por isso essa região era 0, por não ter sido inicializada. O zero guardado no armazenamento é entendido como null. Então significa que o objeto está nulo, não está instanciado. O comando new Vector <Character> , ele solicita a região de memória para o sistema operacional, onde pode guardar elementos dessa estrutura de dados . O sistema operacional procura o local livre para armazenar e fornece o local pro comando new. O comando new faz a execução do construtor para que essa região de memória seja inicializada. E o comando new retorna o endereço de memória dessa região. O endereço de memória retornado pelo new é guardado no objeto armazenamento que antes era zero; mas agora que eu guardei nele (objeto armazenamento) o retorno do new. O objeto é diferente de nulo, é o endereço de memória onde a instância de vector para guardar caracteres foi criada.

**Eu passei o endereço de memória contido no objeto armazenamento para os construtores das tarefas dos tipos 1,2,3 e 4.** O construtor de tarefa do tipo 1 recebeu o endereço de armz (15min21s), é o endereço do new que está na main. No construtor verifica se armz é nulo e lança exceção, senão guarda o endereço no atributo armazenamento. O atributo armazenamento começou com nulo, e depois de guardar o endereço de armz em armazenamento o atributo fica diferente de zero, ou seja, ele é instanciado. O endereço guardado no construtor da main, é passado/recebido no construtor da tarefa 1 como armz e guardado dentro do atributo armazenamento privativo. Todos os endereços de armz, armazenamento de main são os mesmos; pois o objeto armazenamento foi passado pros construtores de cada tarefa na main e capturados pelos construtores específicos de cada tarefa.

**Isto é, o que a main guardar em armazenamento estará disponível para usar na tarefa do tipo 1. Porque estará acessível através do endereço do atributo armazenamento.** Se a tarefa do tipo1 guardar algo nesse atributo, a main terá acesso; porque tanto o atributo como o objeto têm acesso ao mesmo endereço de memória.

**Então o vector que está no new da main está compartilhado entre a main e a TarefaDoTipo1; as duas têm acesso a esse vector. Logo, é assim que a gente faz compartilhamento da estrutura de dados**. O interessante é que esse compartilhamento que foi feito entre a main e a tarefa do tipo 1, ele também foi feito com as demais tarefas (tipo 2,3,4). Porque o programa passou para as tarefas quando criou os objetos t1,t2,t3,t4 o mesmo armazenamento como parâmetro aos construtores das mesmas tarefas.(Passou o endereço daquele mesmo vector que foi criado no new Vector <Character> e passado para os construtores dos new – todas as tarefas tem new- TarefaDoTipo2, TarefaDoTIpo3, TarefaDoTipo4). Ou seja, todas as tarefas estão compartilhando o mesmo vector que inclusive é compartilhado também pela main.

**Como que acontece para as outras tarefas compartilharem? É o mesmo esquema. A main criou e tem o endereço da instância criada dentro do objeto armazenamento. Passou a instância pro construtor de TarefaDoTipo1, TarefaDoTipo2, TarefaDoTipo3, TarefaDoTipo4.** Todos têm o mesmo construtor que é armazenamento, recebe o endereço da main; verifica se o endereço é nulo e manda exceção, do contrário ele guarda o armazenamento em armz. Logo, todas começam com a mesma biblioteca, mesmo construtor.

Depois do construtor, tem as declarações de private boolean fim =false e o método public void morra (ao qual declarada this.fim = true); esses comandos são comuns a todas as tarefas.

**Agora observem vocês que o método run da TarefaDoTipo1 faz algo muito parecido com a mesma tarefa da aula passada. Na TarefaDoTipo1 dessa aula, dentro do método void run há a repetição while (!this.fim) no qual não printa o caractere;** ele pede para adicionar o caractere ao vetor armazenamento; depois dá o tempo de 150milissegundos; se o caractere que começa no a for z, ele volta a ser a senão o caractere atual passa a ser acrescentado mais um para o caractere seguinte (se nessa rodada o caractere é a, no else ele vira b; vide transformações de char pra int para capturar os códigos de cada char em int e depois transformar novamente em char). Então, inicialmente eu guardei o ‘a’ no armazenamento e no fim do método pelo else, o ‘a’ passa a ser ‘b’. Depois guardo b e passa a ser c; e assim por diante. Quando else for ‘z’, eu guardo o ‘z’ e depois volta a ser ‘a’. Estou guardando o alfabeto em caracteres minúsculos sem printar.

**Na TarefaDoTipo2 ocorre a mesma lógica do método run da TarefaDoTipo1 (atual) mas ao invés de ser caractere minúsculo, será maiúsculo.** Eu estou armazenando sempre no mesmo Vector (aquele criado e instanciado na main), é a mesma estrutura de dados. Todas as run adicionam caracteres no mesmo vector de armazenamento, estando tudo misturado os elementos de cada tarefa. Na TarefaDoTipo3, faz o mesmo que a TarefaDoTIpo2 no método run porém os caracteres são dígitos; nos quais começa em 0 e vai até 9, quando chegar ao nove o caractere seguinte volta como zero. Todas as tarefas guardam caracteres no mesmo vector, e nunca printa no método run.

**A TarefaDoTipo4 é bem diferente das outras tarefas. A missão da tarefa do tipo 4 não é ficar guardando caracteres no vector de armazenamento.** A missão dessa TarefaDoTipo4 é pegar dessa estrutura (método public void run), caracteres guardados e printar. Então, as tarefas do tipo 1,2 e 3 guardam, mas a tarefaDoTipo4 pega o que está guardado e printa; podendo printar letras minúsculas, maiúsculas e dígitos porque são os dados armazenados das tarefas anteriores.

**Observe como foi feita o método run da TarefaDoTipo4. Nós temos o loop (while (!this.fim) que é válido enquanto o fim não é decretado. O fim é inicialmente falso e torna-se true quando chamo o método morra.** Na repetição de while, eu vejo quantas coisas estão guardadas no armazenamento (this.armazenamento.size ()) == 0), é o armazenamento compartilhado. Se no armazenamento a quantidade de coisas guardadas for zero, eu uso o método yield; esse método foi herdado da implementação de Runnable. Dentro da classe TarefaDoTipo4 eu tenho que declarar e instanciar o objeto tarefa que seja thread. E o meu objeto tarefa é que tem os métodos start, yield; eles são da classe Thread e o objeto tarefa é do tipo thread.

**Vocês devem se lembrar que na main, quando faço t1.start(); eu estou chamando na main o método start que t1 tem porque herdou de thread; a TarefaDoTipo2 também é a mesma coisa,** é na main que chamo o método start (mesmo que não apareça explícito na classe, pela tarefa x -1,2- herdar de thread que posso chamar o start).

**O objeto t3 que vem da TarefaDoTipo3, o t3 foi implementado SEM herdar de thread; pois no cabeçalho da TarefaDoTipo3 ele implementa Runnable.** Então, na main para chamar t3.start eu vou ter que fazer o método start na TarefaDoTipo3, eu implementei o método start como public void start () { this.tarefa.start ();} . O método start da tarefa3 chama o método start ligado ao objeto tarefa , ao qual o objeto tarefa é o objeto do tipo thread. Então eu fiz um start que chama o start do objeto que é atributo da minha classe que é da classe Thread.

**A TarefaDoTipo4 no método start é a mesma coisa da TarefaDoTIpo3. (Parei em 43min48s). A tarefaDoTIpo4 implementa runnable, tem o método start que eu fiz.** E esse último método chama o método start ligado ao atributo tarefa.; esse atributo foi declarando sendo do tipo thread.

E **o tipo thread tem o método start. Do mesmo jeito que thread tem o método start, seja porque a minha classe herdou , ou então porque a minha classe não herdou e eu tive que fazer como public void start () {this.tarefa.start();}**. Seja como for, a mesma coisa que acontece com esse start , acontece com o yield.

**Então no public void run que contém a repetição while (!this.fim) da tarefa do tipo 4, tem o método yield se as coisas do armazenamento forem nulas. O método yield vem de thread; quem tem thread é o atributo tarefa.** O método yield abre mão da sua vez no processador. Então, a minha tarefa do tipo 4 está rodando no processador, é a vez dela de rodar e ela ainda poderia rodar por mais algum tempo. Mas ela abre mão. A TarefaDoTipo4 sai do processador e dá a vez para outra thread rodar. (que pode ser tarefa 1,2,3). O método yield abre mão porque a tarefa do tipo 4 printa o que está no armazenamento; mas se o tamanho for zero, como eu irei printar? Então, não tem como exercer o papel de printar, a tarefa do tipo 4 abre mão da execução.

**Se o tamanho do armazenamento for maior que zero irei para o else do método run na tarefa do tipo 4. Nesse else, ele pega do armazenamento pelo método get o primeiro caractere (0) e coloca na variável caractere**. Depois disso, ele remove o caractere do armazenamento e printa na tela. E depois ele espera 200 milissegundos e pega o segundo caractere e assim por diante no loop. Quando ele perceber que o armazenamento não tem nada (ele entra no if) e abre mão de executar no processador.

**E quando o método morra for chamado? O loop de !this.fim vai parar. Porém, pode ser que ainda haja alguma coisa no armazenamento quando for chamado o método morra.** Ela morre mas não de uma vez, pois primeiro vai esvaziar o armazenamento. Então, no segundo while de dentro do método run:

**Enquanto o armazenamento não tiver tamanho zero, ela pega o caractere do armazenamento, coloca na variável caractere; depois remove do armazenamento, printa na tela e espera 600 milissegundos.** Ela já saiu do loop onde fim é false, porque já chamou o método morra, mas ainda pode ter algum resquício de coisa guardada que ainda não foi printada. Por isso, ela esvazia o armazenamento, pegando removendo e printando até o tamanho do armazenamento ficar zero. Quando o armazenamento for zero, o segundo loop para e o método run termina; ou seja, a thread morre.

**Na main, depois de criar e instanciar as tarefas com t1,t2,t3,t4 ; esses objetos começam pelo método start. Depois, é esperado que o usuário aperte o enter captado pelo teclado para matar as tarefas com as chamadas do método morra** (t1.morra (); t2.morra(); e assim por diante). Depois printa “ execução do programa finalizada”. E por fim, um catch onde sei que não vai dar erro.

**A execução do programa.java ocorre como esperado. O método main acabou, mas o programa não acabou, porque a tarefa do tipo 4 embora já tivesse recebido a ordem de morrer, ela ainda estava esvaziando o que estava armazenado pra morrer**. A main morreu antes da tarefa 4, e com isso o programa continuou rodando porque ainda tinha tarefa rodando. Nem sempre quando a main acaba o programa acaba. O programa acaba de verdade quando todas as tarefas terminarem em programas de multitarefas, e a main é encarada como uma tarefa de todas as outras.

**Isso não pode acontecer da main terminar e as coisas continuarem printando. Pra resolver essa esquisitice: após as ordens de morrer de t1,t2,t3 e t4 na main; eu chamo o método join para t1, t2, t3 e t4 como:**  
t4.morra ():

t3.morra ():

t2.morra ():

t1.morra ():

t1.join ():

t2.join ():

t3.join ():

t4.join ():

System.out.println (“Execução do programa finalizada”):

}

Catch (Exception erro) {} // sei que não passei null para o construtor de nenhuma das tarefas

**O método join faz que a main esperar a tarefa terminar. Então ela chama assim: t1.join (); fica esperando a tarefa t1 terminar. Terminou? Ela chama o t2.join e fica esperando a tarefa** t2 terminar. Terminou? Ela chama t3.join e fica esperando a tarefa t3 terminar. Terminou? ? Ela chama t4.join e fica esperando a tarefa t4 terminar. Só aí ela vai dar a mensagem de execução do programa finalizada; e depois termina os comandos do método main, ou seja, somente após todos os join terem sido finalizados a main de fato termina a sua execução e encerra o programa.

**Observe que no método join, a tarefa do tipo 1 foi herdado de thread e não tem nada implementado que se chama join; na tarefa do tipo 2 ocorre a mesma coisa como na tarefa do tipo 1 (o método join não aparece implementado).** Mas, nas tarefas do tipo 3 e 4, nós implementamos runnable , ou seja, herdamos de runable. E com isso, nas tarefas 3 e 4 nós temos um atributo do tipo thread chamado tarefa, e esse atributo tem os métodos start, join, yield: são métodos próprios de thread.

**Logo, como eu fiz na main para chamar t3.join(); e t4.join (); já que t3 e t4 não herdaram de thread o método join? Do mesmo jeito que eu fiz com o método start. Eu** fiz o próprio método start que chamava o método start do atributo tarefa que era do tipo thread. Assim sendo, eu fiz o método :

Public void join() throws InterruptedException

{

this.tarefa.join ();

}

**Logo, tive que fazer o método join nas tarefas do tipo 3 e 4 . Então, quando na main quando chama t1.join () e t2.join (); eu chamo o método join que tem em t1 e t2 porque foi herdado de thread . Já quando eu chamo t3.join () e t4.join () estou chamando o método join que tem no t3 e t4 mas que não foram herdados de thread,** eu fiz esses métodos e eles chamaram o método join do atributo tarefa do tipo thread. De qualquer forma, todos os join de t1,t2,t3,t4 vai fazer com que a main espere todas as tarefas terminarem para finalmente ela dar a mensagem de fim de programa e ela terminar.

**Na execução de programa.java com os usos do join, é para o programa acabar de verdade. O programa executou como esperado agora. Observe que, quando tem o pedido de morte, o ideal é que não tenha outra repetição , pois a segunda repetição na tarefa do tipo 4 (que verifica se tem algo no armazenamento para esvaziá-lo) atrasa a morte** – atrasa a morte da tarefa do tipo 4- e o fim da main; para evitar da main terminar antes da tarefa do tipo 4 principalmente, que foi colocado o método join em todas as tarefas (objetos t1,t2,t3 e t4) após a ordem de morte e antes do print de execução finalizada (que sinaliza que a main terminou).

**7.4. Lista de Arquivos usados na aula 7, parte 2.**

**7.4.1.Arquivo: Programa.Java**

import java.util.Vector;

public class Programa

{

public static void main (String[] args)

{

try

{

Vector<Character> armazenamento = new Vector<Character> ();

System.out.println ("Tecle ENTER para ativar as tarefas e");

System.out.println ("Tecle novamente ENTER para terminar o programa.");

Teclado.getUmString();

TarefaDoTipo1 t1 = new TarefaDoTipo1 (armazenamento);

t1.start ();

TarefaDoTipo2 t2 = new TarefaDoTipo2 (armazenamento);

t2.start ();

TarefaDoTipo3 t3 = new TarefaDoTipo3 (armazenamento);

t3.start ();

TarefaDoTipo4 t4 = new TarefaDoTipo4 (armazenamento);

t4.start ();

Teclado.getUmString();

t4.morra ();

t3.morra ();

t2.morra ();

t1.morra ();

t1.join();

t2.join();

t3.join();

t4.join();

System.out.println ("Execucao do programa finalizada.");

}

catch (Exception erro)

{} // sei que não passei null para o construtor de nenhuma das tarefas

}

}

+++++++++++++++++++++++++++

**7.4.2.Arquivo: TarefaDoTipo1.java**

import java.util.Vector;

public class TarefaDoTipo1 extends Thread

{

private Vector<Character> armazenamento;

public TarefaDoTipo1 (Vector<Character> armz) throws Exception

{

if (armz==null)

throw new Exception ("Armazenamento ausente");

this.armazenamento = armz;

}

private boolean fim = false;

public void morra ()

{

this.fim=true;

}

public void run ()

{

char caractere='a';

while (!this.fim)

{

this.armazenamento.add (caractere);

try { this.sleep (150); } catch (Exception erro) {}

if (caractere=='z')

caractere = 'a';

else

caractere = (char)(((int)caractere)+1);

}

}

}

+++++++++++++++++++++++++++++++

**7.4.3.Arquivo: TarefaDoTipo2.java**

import java.util.Vector;

public class TarefaDoTipo2 extends Thread

{

private Vector<Character> armazenamento;

public TarefaDoTipo2 (Vector<Character> armz) throws Exception

{

if (armz==null)

throw new Exception ("Armazenamento ausente");

this.armazenamento = armz;

}

private boolean fim = false;

public void morra ()

{

this.fim=true;

}

public void run ()

{

char caractere='A';

while (!this.fim)

{

this.armazenamento.add (caractere);

try { this.sleep (300); } catch (Exception erro) {}

if (caractere=='Z')

caractere = 'A';

else

caractere = (char)(((int)caractere)+1);

}

}

}

+++++++++++++++++++++++++++++++

**7.4.4.Arquivo: TarefaDoTipo3.java**

import java.util.Vector;

public class TarefaDoTipo3 implements Runnable

{

private Vector<Character> armazenamento;

public TarefaDoTipo3 (Vector<Character> armz) throws Exception

{

if (armz==null)

throw new Exception ("Armazenamento ausente");

this.armazenamento = armz;

}

private Thread tarefa = new Thread (this);

public void start ()

{

this.tarefa.start();

}

public void join () throws InterruptedException

{

this.tarefa.join();

}

private boolean fim = false;

public void morra ()

{

this.fim=true;

}

public void run ()

{

char caractere='0';

while (!this.fim)

{

this.armazenamento.add (caractere);

try { this.tarefa.sleep (450); } catch (Exception erro) {}

if (caractere=='9')

caractere = '0';

else

caractere = (char)(((int)caractere)+1);

}

}

}

+++++++++++++++++++++++++++++++++++++++++

**7.4.5.Arquivo: TarefaDoTipo4.java**

import java.util.Vector;

public class TarefaDoTipo4 implements Runnable

{

private Vector<Character> armazenamento;

public TarefaDoTipo4 (Vector<Character> armz) throws Exception

{

if (armz==null)

throw new Exception ("Armazenamento ausente");

this.armazenamento = armz;

}

private Thread tarefa = new Thread (this);

public void start ()

{

this.tarefa.start();

}

public void join () throws InterruptedException

{

this.tarefa.join();

}

private boolean fim = false;

public void morra ()

{

this.fim=true;

}

public void run ()

{

while (!this.fim)

{

if (this.armazenamento.size()==0)

this.tarefa.yield (); // abre mão de sua vez no processador

else

{

char caractere = this.armazenamento.get(0);

this.armazenamento.remove(0);

System.out.println (caractere);

try { this.tarefa.sleep (200); } catch (Exception erro) {}

//try { this.tarefa.sleep (150); } catch (Exception erro) {}

//try { this.tarefa.sleep (100); } catch (Exception erro) {}

//try { this.tarefa.sleep (50); } catch (Exception erro) {}

}

}

while (this.armazenamento.size()!=0)

{

char caractere = this.armazenamento.get(0);

this.armazenamento.remove(0);

System.out.println (caractere);

try { this.tarefa.sleep (600); } catch (Exception erro) {}

}

}

}

+++++++++++++++++++++++++++++++

**7.4.6.Arquivo: Teclado.java**

//import java.io.BufferedReader;

//import java.io.InputStreamReader;

//import java.io.IOException;

import java.io.\*;

public class Teclado

{

private static BufferedReader teclado =

new BufferedReader (

new InputStreamReader (

System.in));

public static String getUmString ()

{

String ret=null;

try

{

ret = teclado.readLine ();

}

catch (IOException erro)

{} // sei que nao vai dar erro

return ret;

}

public static byte getUmByte () throws Exception

{

byte ret=(byte)0;

try

{

ret = Byte.parseByte (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Byte invalido!");

}

return ret;

}

public static short getUmShort () throws Exception

{

short ret=(short)0;

try

{

ret = Short.parseShort (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Short invalido!");

}

return ret;

}

public static int getUmInt () throws Exception

{

int ret=0;

try

{

ret = Integer.parseInt (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Int invalido!");

}

return ret;

}

public static long getUmLong () throws Exception

{

//long ret=(long)0;

//long ret=0;

long ret=0L;

try

{

ret = Long.parseLong (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Long invalido!");

}

return ret;

}

public static float getUmFloat () throws Exception

{

//float ret=0;

//float ret=(float)0.0;

float ret=0.0F;

try

{

ret = Float.parseFloat (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Float invalido!");

}

return ret;

}

public static double getUmDouble () throws Exception

{

//double ret=0;

//double ret=(long)0;

//double ret=0L;

double ret=0.0;

try

{

ret = Double.parseDouble (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Double invalido!");

}

return ret;

}

public static char getUmChar () throws Exception

{

char ret=' ';

try

{

String str = teclado.readLine ();

if (str==null)

throw new Exception ("Char invalido!");

if (str.length() != 1)

throw new Exception ("Char invalido!");

ret = str.charAt(0);

}

catch (IOException erro)

{} // sei que nao vai dar erro

return ret;

}

public static boolean getUmBoolean () throws Exception

{

boolean ret=false;

try

{

String str = teclado.readLine ();

if (str==null)

throw new Exception ("Boolean invalido!");

if (!str.equals("true") && !str.equals("false"))

throw new Exception ("Boolean invalido!");

ret = Boolean.parseBoolean (str);

}

catch (IOException erro)

{} // sei que nao vai dar erro

return ret;

}

}

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**8. Pasta: Aula 21-10-2021**

**Pasta: 2021- 10-21.T&A**

**Arquivos: ThreadCompartilhandoEDcomLimites.mp4**

**AULA 8- Thread Compartilhando ED com Limites (dia 21 Out 2021) : produtor e consumo**

**Vamos falar a respeito de tempos. Em todas as tarefas do tipo 1,2,3 e 4 ele colocou dentro do método run no nosso loop colocou o try-catch.** Na primeira aula que falou sobre tarefas, o método sleep causa uma certa demora na nossa tarefa. A demora é importante para ter tempo de ler as coisas que estavam sendo printadas na tela. O arquivo da TarefaDoTIpo1 que tem dentro do run um try com this.sleep (150) com 150 milissegundos. Já na tarefa do tipo2 o tempo é de 300 milissegundos . Já na tarefa do tipo3 o tempo é de 450 milissegundos. Isso significa que a produção de caracteres minúsculos vai ser mais rápido porque a cada 150 milissegundos ele já produz outro catactere (na TarefaDoTIpo1). Já na tarefaDoTIpo2 que tem caracteres maiúisculos, elas levam o dobro do tempo (produz o A e depois de 300 milissegundos produz o B). A tarefa doTipo3 produz um número a cada 450 milissegundos que demora 3x mais que o tipo1.

**Eu poderia não ter essas demoras mas as coisas que as tarefas fazem podem demorar naturalmente; eu forço as demoras nas tarefas.** Quando você faz tarefas reais em aplicações reais tem demoras diferenciadas em diferentes tarefas que rodam ao mesmo tempo. É pela natureza da tarefa, da coisa que ela faz, algoritmo de funcionamento que determina as demoras.

**As tarefas do tipo 1, 2 e 3 produzem caracteres, respectivamente: letras maiúsculas, minúsculas e números**. Colocam o que produziram no armazenamento que é um vector para armazenar caracteres. Apenas a tarefa do tipo 4 que consome os dados de guardar armazenamento. Porque? A tarefa do tipo 4 pega um caractere do armazenamento, remove do armazenamento, printa o caractere e espera 200 milissegundos para o próximo caractere. **(PAREI EM 06MIN22S).**

**Nós temos 4 tipos de tarefa, e existem tempos diferentes; a tarefa do tipo 1 demora 150 milissegundos; a tarefa do tipo 2 demora 300 milissegundos; do tipo 3 450 milissegundos** e do tipo 4 tem vários tempos e vamos falar sobre eles: 200, 150, 100, 50. Do 150 ao 50 milissegundos estão como comentários.

**Pode acontecer das tarefas dos tipos 1,2 e 3 ficarem produzindo caracteres e pondo no vetor. E a tarefa do tipo 4, por ser lenta, ser apenas 1 que está consumindo; ela pode ser demorada pelo tempo que eu forcei ou pela vida real:** ela pega e demora pra fazer o trabalho que precisa ser feito. Nesse vídeo, foi forçado os tempos; na vida real ela demora o tempo que precisar, a depender do tipo de tarefa.

**Eu tenho 3 tipos de tarefa produzindo e somente 1 consumindo. Esse consumo pode demorar; pode acontecer da tarefa do tipo 4 nunca dar conta de consumir as tarefas produzidas pelos tipos 1, 2 e 3? As tarefas do tipo 1,2e 3 produzem muito mais enquanto a tarefa 4 demora para consumir.** Pode acontecer da tarefa do tipo 4 não dar conta de consumir tudo o que é produzido e a produção se acumular naquele vector. O vector cresce muito, devido as tarefas 1, 2 e 3 produzirem coisas. Em um certo tempo, o vector diminui porque a tarefa do tipo 4 remove as coisas. Fica nesse ciclo. Sempre vai no fim das contas; o tamanho da memória ocupada pelo vector ir crescendo bastante; pois mais cresce do que diminui. Mas cresce porque o consumidor do tipo 4, além de ser um só pode ter uma certa lentidão. Em média, o vector pode só crescer, e pode acabar ocupando toda a memória do computador; ele pode travar pela falta de memória no computador, causando travamentos. Isso pode ser um problema.

**Eu vou deixar o tempo de 200 milisegundos na tarefa do tipo 4. A tarefa é 1 só e tem a lentidão de 200 milissegundos.** Vai rodar o programa. Não pode rodar a tarefa do tipo 4 por não ter main. Vamos rodar a main no Programa.java.

**Começou a produção, chegou no fim. Perceba quantos ele tem que consumir depois de mandou parar; a tarefa do tipo 4 demora tanto para consumir porque não estava dando conta, ele demora 200 milissegundos.** Essa diferença pode ir se acumulando, isso foi no tempo de encher uma janela; a longo prazo pode acabar com a memória do computador. Se mudar no while (!this.fim):

While (!this.fim)

{

if (this.armazenamento.size () ==0)

this.tarefa.yield (); // abre mão de sua vez no processador

else

{

char caractere = this.armazenamento.get (0);

this.armazenamento.remove (0);

System.out.println (caractere);

Try (this.tarefa.sleep (200);} catch (Exception erro ) {}

//Try (this.tarefa.sleep (150);} catch (Exception erro ) {}

//Try (this.tarefa.sleep (100);} catch (Exception erro ) {}

//Try (this.tarefa.sleep (50);} catch (Exception erro ) {}

}

}

While (this.armazenamento.size () !=0)

{

Char caractere = this.armazenamento.get (0);

This.armazenamento.remove (0);

System.out.println (caractere);

Try {this.tarefa.sleep (600);} catch (Exception erro( {}

}

}

**Se no While (!this.fim) a linha ativa de try for a de 150 milissegundos. Ao executar, dispara o enter para ter produção e consumo.** Ao terminar de encher a janela, mando parar. Vai ficar menos tempo a tarefa do tipo 4 ficar consumindo; porque o tempo ficou mais rápido, saiu de 200 e foi para 150 milissegundos.

**Agora vamos deixar ativo 100 milissegundos para a tarefa do tipo 4. Como a tarefa do tipo 4 está com tempo diminuído, faz com que ela dê conta de consumir as coisas produzidas pelas tarefas 1,2 e 3.** O último teste deixou o tempo para 50 milissegundos para a tarefa do tipo 4; pois as tarefas dos tipos 1,2,3 estão respectivamente: 150, 300 e 450 milissegundos. É esperado que a tarefa do tipo 4 dê conta muito melhor do que é produzido pelas demais tarefas com 50 milissegundos; a hora que pedir pra parar ele para praticamente junto.

**No programa ele é didático, experimental, decidindo o tempo de demora. Normalmente, não tem esse poder de decisão; as tarefas demoram o que precisar, uma demora natural do algoritmo implementado na tarefa**. Ele quer despertar a preocupação com a ocupação descontrolada de memória; pois poderia acontecer esse descontrole de excesso de produções causando a demora na tarefa 4 (que pega os dados das outras tarefas pra consumir); pois tem pouca coisa sendo retirada. Assim sendo, o vector cresce até acabar com a memória do computador. Precisa estar atento a essa possibilidade.

**Já que não posso determinar os tempos da tarefa, o que deve ser feito? As tarefas fazem o que o algoritmo determinou a fazer.** O programa não pode capotar o computador; se os tempos forem muito diferentes e a tarefa 4 demorar muito para consumir, acumulando as demais tarefas. Elas podem se acumular demais. Temos que impor um limite; para parar de acumular, não pode ser uma coisa descontrolada.

**Até agora a pouco, as tarefas rodavam todas ao mesmo tempo. Ou senão tivesse processador pra rodar todas ao mesmo tempo, elas se revezavam no uso do processador.** O que fazia as tarefas eventualmente pararem seriam unicamente o fato de que haveria a necessidade delas se revezarem; já que não teria processador para processar todas e a cada tarefa seria atribuída uma fatia de tempo. Então, uma tarefa executa uma fatia de tempo e para; uma segunda usa a mesma fatia de tempo e para e assim por diante.

**Se eu tivesse processador para processar todas ao mesmo tempo , por exemplo se estivesse usando um processador octa core num computador que tivesse 8 núcleos de processamento**. O meu programa só tem 4 tarefas, na verdade 5 por ter a main. Logo, a princípio tem processador para processar tudo ao mesmo tempo, pois tem 5 tarefas e oito núcleos; nessa situação: 3 tarefas rodando ao mesmo tempo e produzindo; 1 tarefa rodando ao mesmo tempo e consumindo. Depende se a tarefa que está consumindo demora, é pesada e se pode estar acumulando as demais tarefas dentro do vector. Ia acabar estourando a memória. Se eu tenho menos núcleo de processamento, preciso revezar as tarefas. (23min31s).

**Digamos que estivesse usando um dual core (dois núcleos de processamento ) e 5 tarefas: 2 tarefas entrariam em execução (exemplo: tarefas 1 e 2), as duas ao mesmo tempo começam a produzir e colocar no vetor e vai enchendo.** Depois, acabou a fatia de tempo delas; vem a tarefa 3 e 4. A tarefa 3 continuando pondo coisas no vetor e a tarefa 4 tirando; só que a tarefa 4 talvez demore um pouco demais e não tenha dado conta de tirar.

**Tanto faz se a gente consegue rodar essas tarefas ao mesmo tempo ou se vão se revezar se existe esse desbalanceamento de tempo entre essas várias tarefas e é normal que exista; pois são tarefas diferentes.** Se fosse a mesma tarefa, não precisaria fazer tarefa 1,2,3; faria apenas a tarefa do tipo 1. São tarefas diferentes com algoritmos diferentes: é natural que ocorra tudo com tempos diferentes, ocupados para executar o vetor com tempos diferentes, é o esperado.

**O que eu irei fazer a respeito para que a coisa não fuga do controle?** (25min37mins). O que fazer para não deixar a tarefa do tipo 4 ficar ocupando o processador, querendo consumir sendo que não tem nada para consumir. Pois de repente a tarefa do tipo 4 é muito rápida e ela consome tudo; mesmo que não tenha nada para consumir, ela ocupa o processador. Ou o contrário, a tarefa do tipo 4 não dá conta e ela fica ocupando espaço na memória, não tem memória. Como manter isso sobre controle?

**Basicamente o que nós vamos usar é a classe Semaphore** (26min45s). Uma classe que implementa um semáforo. O que é um semáforo? Um semáforo é um controlador de recursos; ele impõe limites as minhas tarefas. Se as minhas tarefas tem uma certa quantidade de recursos e assim, quando eu falo recursos, pode ser qualquer coisa. Mas aqui nesse programa, recursos vão ser posições no vector de caractere armazenamento.

**Imaginem que eu declare (no arquivo: Programa.Java) dois objetos da classe semaphore. Essa classe está na biblioteca java.util.concurrent, Semaphore.** E imagine que eu declare dois objetos da classe Semaphore e que eu instancie os dois objetos o new semaphore. Que eu passe pro construtor do primeiro: 1024. E pro construtor do segundo, o zero. O que significam esses números? Significa a quantidade de recursos disponíveis.

**Então, quando chega no primeiro objeto e coloca 1024, eu estou falando que tenho 1024 recursos disponíveis no semáforo livre. O semáforo livre deixa, sempre que eu pedir recursos, ele me dá até chegar no limite de 1024.** Ao chegar no limite de 1024, ele não dá mais recursos e a minha tarefa quando pede um recurso e não tem, a minha tarefa para de ser executada; mesmo que a minha fatia de tempo dela ainda esteja com bom tamanho, mesmo que ainda tenha tempo de executar, ela é interrompida; é parada de executar.

**Os semáforos funcionam assim: eu estabeleço a quantidade de recursos que inicialmente eu tenho no meu semáforo, e eu peço recursos através de um método e removo recursos através de outro método.** Tem dois métodos em semáforos, que são métodos muito importantes; um pra pedir recursos e outro pra devolver recursos.

**No semáforo que chamei de livre tenho 1024 recursos; qual a minha intenção de ter posto 1024? Estabelecer que existem 1024 posições livres no armazenamento.** Sempre que eu for armazenar, eu peço pra esse semáforo, ele deixa; mas a posição fica sendo 1024-1. Na segunda vez eu peço, mas já não tem mais 1023. Até que chega uma hora que ele não deixa armazenar porque não tem mais recursos. Chegou no limite que eu estabeleci, de 1024.

**O semáforo ocupado, ele começou com zero; porque a princípio não tem nenhuma posição ocupada no vector chamado armazenamento**. Então, ele começa com zero; se por um acaso a tarefa do tipo 4 , precisa pegar alguma coisa guardada; ela precisa que tenha alguma coisa ocupando o vetor de armazenamento. Se ela pedir pro semáforo ocupado, o semáforo ocupado bloqueia e não deixa mais executar. Porque ela está querendo consumir e não tem o que consumir. Porque esse recurso está zerado; as posições ocupadas começam zeradas.

**Reparem que agora eu vou ter 3 objetos sendo compartilhados pelas tarefas. A main declarou esses 3 objetos: Vector <Character> armazenamento, semaphore livre e semaphore ocupado.** Tem os 3 objetos declarados e instanciados; e são passados como parâmetros aos construtores das minhas 4 tarefas:

**8.1. Lista dos Arquivos usados na aula 8**

**8.1.1.Arquivo: Programa.java**

Public class Programa

{

Public static void main (String [] args)

{

Try

{

Vector <Character> armazenamento = new Vector <Character> ();

Semaphore livre = new Semaphore (1024,true);

Semaphore ocupado = new Semaphore (0,true);

System.out.println (“ Tecle ENTER para ativar as tarefas e”);

System.out.println (“ Tecle novamente ENTER para terminar o programa.”);

Teclado.getUmString ();

TarefaDoTipo1 t1 = new TarefaDoTipo1 (armazenamento, livre, ocupado);

t1.start ();

TarefaDoTipo2 t2 = new TarefaDoTipo2 (armazenamento, livre, ocupado);

t2.start ();

TarefaDoTipo3 t3 = new TarefaDoTipo3 (armazenamento, livre, ocupado);

t3.start ();

TarefaDoTipo4 t4 = new TarefaDoTipo4 (armazenamento, livre, ocupado);

t4.start ();

Agora, os construtores tem 3 parâmetros; tanto nas tarefas do tipo 1,2,3 e 4. (verifique os Javas das tarefas do tipo 1,2,3 e 4**). Assim sendo, por exemplo, para a tarefa do tipo 2:**

**8.1.2. Arquivo:TarefaDoTipo2.java**

public class TarefaDoTipo2 extends Thread

{

Vector <Character> armazenamento;

Semaphore livre;

Semaphore ocupado;

public TarefaDoTipo2 (Vector <Character> armz, Semaphore lvr, Semaphore ocp) throws Exception

{

if (armz == null)

throw new Exception (“Armazenamento ausente”);

if (lvr == null)

throw new Exception (“Livre ausente”);

if (ocp == null)

throw new Exception (“Ocupado ausente”)

this.armazenamento = armz;

this.livre = lvr;

this.ocupado = ocp;

}

Private boolean fim = false;

Public void morra ()

{

This.fim = true;

}

Public void run ()

{

char caractere = ‘a’;

while (! this.fim)

{

this.livre.acquireUnityInterruptibly;

this.armazenamento.add (caractere);

this.ocupado.release ();

try { this.sleep (150); } catch (Expcetion erro) {}

if (caractere == ‘z’)

caractere = ‘a’;

else

caractere = (char) (( ( int) caractere) + 1 );

} // fim while

} // fim run

} // fim programa

**Todos os tipos de tarefa declaram objetos, mas não instanciam. Porque?** Se fizessem new nos objetos antes de fazer o construtor; eles não estariam compartilhando aqueles objetos instanciados na main.

**Todos os construtores de todos os tipos de tarefas recebem os três parâmetros: Vector <Character>, Semaphore lvr, Semaphore ocp; checam se são nulos e lançam exceções se forem nulos.** Depois, eles pegam o endereço recebido dos objetos que estão no construtor; os objetos que estão na main que instanciaram endereços de memória (usando new) a serem usados pelos programas das 4 tarefas. Os new retornaram os endereços e foram guardados nos objetos correspondentes que estão na main.

**Na main, os endereços guardados foram passados como parâmetros para todas as tarefas, assim que os objetos das tarefas (t1,t2,t3,t4) foram criados e instanciados.** Assim sendo, dentro do programa de cada tarefa, no construtor ele recebe os endereços da main e depois guardam com this nos nomes dos atributos (dentro do construtor de cada programa de tarefa). Os 4 tipos de tarefas usam o mesmo cabeçalho de bibliotecas, classes e construtor.

**Logo, os 4 tipos de tarefas estão compartilhando não só o Vector <Character> , mas também os dois semáforos. A main criou tudo, passou o endereço de tudo. Os endereços foram recebidos nos construtores de cada tipo de tarefa (no cabeçalho do método construtor**) e foram guardados nos atributos de cada tipo de tarefa (no programa de cada tipo de tarefa).

**Então, os 4 tipos de tarefa têm acesso aos mesmos Vector <Character>, semáforo livre e semáforo ocupado.** As tarefas do tipo 1,2 e 3, são tarefas que produzem caracteres e guardam naquele vector de caractere. (36min45s) A produção acontece no char caractere = ‘a’ dentro do void run. Ele entra no while (!this.fim) e 150 milissegundos depois o caractere vira b,c,d, e ,f e assim por diante (ele entra no else). Quando chegar no z ele vira a de novo.

**Ao invés de eu simplesmente pegar o caractere produzido e guardar no vector; na verdade no vector sempre cabe porque ele não esgota o espaço.** Então ele vai aceitando guardar , consumindo memória. Mas uma hora a memória do computador acaba.

**Antes, eu simplesmente guardava como na linha this.armazenamento.add (caractere); chamava o método add pra guardar.** Só que agora eu faço algo diferente: antes de guardar, eu pego o semáforo livre e faço que eu quero adquirir um recurso (método acquire) , eu peço um recurso. O semáforo livre, conforme vocês viram lá na main, foi instanciado como 1024. Então, na hora que eu peço um recurso, falo que eu quero adquirir um recurso, aquele 1024 vai pra 1023. Se eu falo que eu quero adquirir de novo, vai pra 1022; se pedir novamente vai para 1021. E todas as threads o tipo 1,2,e 3 na hora de guardar, antes de guardar, fala que quer adquirir um recurso que é uma posição livre. Fala que quer adquirir um recurso do semáforo livre.

**Então, se as tarefas do tipo 1,2 e 3 estão rodando todas ao mesmo tempo , e o semáforo livre contava a princípio com 1024 recursos , essas 3 tarefas vão baixando esses recursos a partir de 1024 (a cada pedido baixa -1), indo de 1024 pra 1023, 1023 pra 1022 e assim por diante.** E se por acaso zerar, porque o consumidor, a tarefa do tipo 4 não está dando conta de consumir, pra caso zerar. Na hora que chegar e falar quero adquirir um recurso livre, essa tarefa para de executar porque não tem recurso livre; e não vai guardar o caractere porque ela parou de executar quando ela manifestou o desejo de querer um recurso livre. Esse recurso era 1024 mas foi baixando até chegar em zero. A tarefa é interrompida por falta de recurso livre, mesmo que tenha tempo de execução. (40min08s).

**E mais tarde, quando o recurso livre estiver disponível porque o consumidor consumiu; o consumidor já tirou alguma coisa do vetor.** Quando o recurso livre estiver disponível, a tarefa 3 que pediu vai conseguir adquirir e guardar o caractere. Mas na hora que ela quer adquirir, senão tiver ela é interrompida. Isso acontece nas tarefas dos tipos 1,2,3 .

**O semáforo que chamei de livre é porque eu queria um bom nome, mas poderia ter qualquer nome, como x. Chamei de livre porque vou usar esse semáforo pra pedir recursos para ele, antes de guardar**. Ele vai começar com 1024, o que ele representa pra mim? Ele representa posições livres, dizendo que a princípio pode usar até 1024 posições que estão livres. Na hora que eu pedir o recurso, se bloqueou não me deixa guardar. Porque não deixa guardar? Porque eu estava querendo adquirir um recurso que era uma posição livre, então como era o contexto, eu resolvi chamar o semáforo de livre.

**Os tipos 1,2 e 3 fazem a mesma coisa no começo do while (!this.fim) : eles pedem para adquirir uma posição no semáforo livre ; não tendo, bloqueia a execução e não chega a guardar. Somente quando tiver recurso que desbloqueia e guarda recurso.** Agora, tendo recurso, ele guarda o que quer guardar e chama o método release depois de ter guardado. Então, ele chama o método release do semáforo ocupado. O semáforo ocupado é aquele semáforo que começou zerado. O release faz com que aumente 1 número no ocupado e baixe 1 número no livre.

**Eu quero adquirir uma posição do semáforo livre, eu guardo alguma coisa no armazenamento e eu libero (release significa liberar) um recurso do semáforo ocupado**. Assim sendo, o semáforo livre cai 1 posição, enquanto que o semáforo ocupado sobe 1 posição.

**Já a tarefa do tipo 4 que é uma tarefa consumidora:**

**8.1.3.Arquivo: TarefaDoTipo4.java**

Import java.util.Vector;

Import hava.util.concurrent.Semaphore;

Public class TarefaDoTipo4 implements Runnable

{

Vector <Character> armazenamento;

Semaphore livre;

Semaphore ocupado;

public TarefaDoTipo4 (Vector <Character> armz, Semaphore lvr, Semaphore ocp) throws Exception

{

if (armz == null)

throw new Exception (“Armazenamento ausente”);

if (lvr == null)

throw new Exception (“Livre ausente”);

if (ocp == null)

throw new Exception (“Ocupado ausente”)

this.armazenamento = armz;

this.livre = lvr;

this.ocupado = ocp;

}

Private Thread tarefa = new Thread (this);

{

This.tarefa.join ();

}

Private boolean fim = false;

Public void morra ()

{

This.fim= true;

}

Public void run ()

{

While (!this.fim)

{

This.ocupado.acquireUninterruptibly ();

Char caractere = this.armazenamento.get (0);

This.armazenamento.remove (0);

this.livre.release();

System.out.println (caractere);

try { this.tarefa.sleep (200); } catch (Exception erro) {}

//try { this.tarefa.sleep (150); } catch (Exception erro) {}

//try { this.tarefa.sleep (100); } catch (Exception erro) {}

//try { this.tarefa.sleep (50); } catch (Exception erro) {}

}

while (this.armazenamento.size()!=0)

{

this.ocupado.acquireUninterruptibly();

char caractere = this.armazenamento.get(0);

this.armazenamento.remove(0);

this.livre.release();

System.out.println (caractere);

try { this.tarefa.sleep (600); } catch (Exception erro) {}

}

}

}

**A tarefa do tipo 4 faz o seguinte: antes dela ir lá recuperar e remover alguma coisa do meu vector de caracteres chamado armazenamento (dentro do while (!this.fim)) , ela diz que quer adquirir um recurso. Mas quer adquirir um recurso ocupado porque ela é consumidora, ela precisa de posições ocupadas para poder consumir.** Depois disso, ela pega o que está no vetor, remove e libera uma posição livre. Ela precisa de uma ocupada e libera uma livre. Isso significa que, imaginem o seguinte cenário: eu coloquei pra rodar as 4 tarefas. Imagine que eu só tenho 1 ou 2 processadores; a tarefa do tipo 4 começou a rodar primeiro, antes de todas as outras. Se ela começou a rodar primeiro, ela vai querer consumir alguma coisa do vetor de caracteres, mas não tem nada, porque quem põe alguma coisa lá são as tarefas do tipo 1,2,3 e a tarefa do tipo 4 eu suponho que roda primeiro. O semáforo ocupado começa com zero recursos. O que vai acontecer é que a tarefa do tipo 4, antes de pegar e remover do armazenamento, ela quer adquirir um recurso controlado pelo semáforo ocupado. E o semáforo ocupado está com os recursos zerados e a minha tarefa do tipo 4 querendo adquirir um recurso.

**Não tem recurso no semáforo ocupado, o que faz com que a tarefa do tipo 4 seja interrompida ao querer adquirir um recurso no semáforo ocupado.** Na hora que tive rum recurso no semáforo ocupado, a tarefa do tipo 4 pode voltar a executar. Digamos que ela foi interrompida; então outra tarefa vai executar; digamos a tarefa do tipo 1.

**A tarefa do tipo 1 quer adquirir um recurso controlado pelo semáforo livre , começa com 1024. Então, tem 1024 recursos e ela quer 1; baixa para 1023 e ela consegue o recurso.** A tarefa do tipo 1 adiciona um caractere no vetor armazenamento e ela libera um recurso controlado pelo semáforo ocupado. Mas o semáforo ocupado estava zerado, na hora que ela libera, o ocupado vai para 1; de forma que a tarefa do tipo 4 vai conseguir executar porque o semáforo ocupado é diferente de zero.

**Logo, a tarefa do tipo 4 vai conseguir adquirir uma posição do semáforo ocupado, vai conseguir pegar o recurso que está no vetor, remover e liberar uma posição no semáforo livre.** Quando eu libero um recurso no semáforo livre, ele estava em 1023 agora volta para 1024.

**O que está acontecendo? As tarefas dos tipos 1,2 e 3 vão baixando o livre e aumentando o ocupado; elas baixam o livre por produzir e guardar nas posições que eu posso ocupar, se estiver dentro do limite do semáforo livre.** E a tarefa do tipo 4, ela baixa ocupado e aumenta o livre; ela libera o livre por estar consumindo das outras tarefas quando retirar do semáforo ocupado. Na tarefa do tipo 4, enquanto não zerar o livre, não teremos problema; as 4 tarefas vão poder executar. Mas se acontecer da tarefa do tipo 4 não der conta de consumir, o vetor de ocupado vai enchendo até chegar em 1024 posições cheias. Quando chegar no 1024, é porque alguma tarefa (tipo 1, 2 ou 3) pediu um recurso livre, era o último, zerou, guardou. A próxima vez que uma tarefa do tipo 1,2 ou 3 pedir um recurso livre, vai estar zerado e não vai rodar o armazenamento, a tarefa vai ser bloqueada nessa aquisição.

**Então, a tarefa do tipo 4 que não está dando conta; ela vai querer adquirir uma posição ocupada que agora está em 1024. Porque o semáforo livre estava em zero e o ocupado estava em 1024.** Baixou um do 1024, aumentou 1 no zero; baixou 2 no 1024, aumentou 2 no zero. A hora que o 1024 zerar, o ocupado que era zerado no começo chegou em 1024. A tarefa do tipo 4 vai pedir pra adquirir recurso do semáforo ocupado e vai conseguir porque tem 1024 posições. Consegue pegar do armazenamento, remover do mesmo vetor e liberar uma posição no semáforo livre. A posição livre, que era zero, volta a ser 1. E a tarefa do tipo 1,2 ou 3, a que ficou esperando primeiro o semáforo livre ter posição, vai poder voltar a executar pra produzir novos produtos; logo a tarefa bloqueada pelo semáforo livre, vai ser desbloqueada.

**O contrário também acontece. Digamos que o nosso consumidor fosse muito rápido, a tarefa 4 pede uma posição ocupada, consome.** Uma hora ele vai querer adquirir um recurso do semáforo ocupado e não vai ter porque ele é muito rápido. Então, ele vai ser bloqueado pelo semáforo ocupado. Porque estava sendo tão rapidinho na hora de consumir, que zerou o semáforo ocupado. Se semáforo for zero, a tarefa do tipo 4 é bloqueada pelo semáforo ocupado. E agora?

**E agora, as tarefas ou dos tipos 1,2,3; a que estive rodando no momento. Quando guardar alguma coisa no vector e liberar um recurso no semáforo ocupado; a tarefa dos tipos 1,2 ou 3 ao liberar um recurso do semáforo ocupado, permitirá o desbloqueio da tarefa do tipo 4.** Logo, a tarefa do tipo 4 consegue executar, conseguindo pegar alguma coisa do armazenamento, remover e libera posição no semáforo livre.

Elas ficam nesse esquema.

**As tarefas 1,2,3 pedem livre e liberam ocupado. A tarefa do tipo 4 é ao contrário, pede ocupado e libera livre. Eventualmente, a tarefa do tipo 4 pode ser presa no semáforo ocupado;** e vai ser solta pelas tarefas 1,2 ou 3 quando liberar uma posição no semáforo ocupado. Por outro lado, as tarefas 1,2 ou 3 podem estar presas no semáforo livre e serão soltas pela tarefa do tipo 4 quando liberar um recurso no semáforo livre.

**As tarefas ficam rodando à vontade, soltas; até que esgotem os recursos livres ou os recursos ocupados. Se esgotar os recursos livres, as tarefas 1 ,2 e 3 são bloqueadas por quererem recursos livres que não tem**. Se tiver rodando todas as tarefas e esgotar os recursos ocupados, a tarefa 4 será bloqueada por querer o recurso ocupado que não tem.

**Os semáforos são contadores que começam com certo valor. No caso do livre começa com 1024. Quando faço o método acquire para adquirir recursos, o contador livre baixa (de 1023 para 1022, 1021). E sempre quando faço release esse contador aumenta (no livre, de 1021 vai pra 1022 e assim por diante).** O mesmo ocorre para ocupado, que começa com zero, quando faz acquire diminui e o release aumenta. E quando estar zerado, não importa se é no libre ou ocupado, quem fizer acquire vai ser bloqueado e só vai ser liberado quando for feito release pois o recurso aumenta.

**O vector em si não tem limite de tamanho; mas pode ocorrer um desbalanceamento de tempo entre as tarefas; pode ocorrer da tarefa do tipo 4 não dar conta de pegar o conteúdo das posições do vector pra consumir e liberar posições livre.** Pois as tarefas 1,2,3 pode começar se acumular no vector e encher a memória; por isso que o semáforo livre tem o limite de 1024 recursos. O vector em si tem tamanho livre. As tarefas 1,2 e 3 antes de guardar pedem aquisição de posições livres para depois ser guardado em armazenamento.

**Cada vez que eu peço um recurso no semáforo livre ele baixa 1 recurso. Na hora que zerar, quem pedir acquire antes de guardar é bloqueado; como senão coubesse no vector.** O semáforo vai brecar as tarefas do tipo 1,2 ou 3 (quem fizer acquire quando semáforo livre estiver zerado) pois controla os recursos livres e está zerado. Não deixa guardar mais nada. Desse jeito, nunca vai ter mais de 1024 itens guardados no vetor armazenamento; quando pedir o 1025, eu vou pedir e ser bloqueado no semáforo livre.

**O limite para o semáforo ocupado é de 0 a princípio. Coloquei limites nos semáforos livre e ocupado; antes de fazer qualquer coisa no vector,** sempre pede recursos (acquire) seja para o semáforo livre (tarefas do tipo 1,2 ou 3) e ocupado (tarefa do tipo 4).

**Para a tarefa do tipo 4 ser executada com semáforo ocupado diferente de zero; as demais tarefas tem que ser executadas primeiro porque aumentam o semáforo ocupado.** Senão, a tarefa do tipo 4 é bloqueada pelo semáforo ocupado, devido ao fato desse semáforo ter zero recursos. (01h06min15s).

**Com esses semáforos eu consigo impor limites: ninguém tira nada se o semáforo ocupado estiver zerado.** Ninguém guarda nada se o semáforo livre estiver zerado. E o semáforo livre começa com 1024, ninguém guarda nada além de 1024. Para perceber o funcionamento disso, baixou o semáforo livre para 4; ou seja, só posso pedir 4x o recurso livre pro semáforo livre.

**Antigamente, ele deixava rodar até o fim e parava porque acumulava coisa demais no vector e quando parava, demorava demais pra tirar tudo o que tinha guardado, atrapalhava rendimento da aula.** Agora, deixa rodar porque não é pra acumular muita coisa no vector , porque não guarda além do limite. Os semáforos livre e ocupado controlam isso; pois o semáforo vem antes das atividades da tarefa; podendo bloquear no acquire quando der o limite. Quando parei, rapidamente acabou; poderia ser até 7, dependendo da quantidade de processadores no computador. A tarefa do tipo 4 pode printar até 4 coisas; se tiver 1 processador, pode ser que entre 1 coisa na tarefa 1 e receba a ordem de morrer; o mesmo ocorreu nas tarefas do tipo 2 e 3. Isso soma: 4 coisas da tarefa do tipo 4 mais 1 coisa de cada tarefa (tipo 1, 2 e 3) totalizando 7 coisas pra printar pela tarefa do tipo 4 (em caso extremo). Isto é, tinha 4 coisas guardadas no vector, a tarefa do tipo 4 pega essas coisas e mostra; não tem nada mais guardado, ela para. Entra a tarefa do tipo 1, guarda 1 coisa e recebe a ordem de morrer. Depois entra a tarefa do tipo 2, guarda 1 coisa e depois a ordem de morrer. Depois entra a tarefa do tipo 3, guarda 1 coisa e depois a ordem de morrer. Então, na pior das hipóteses eu veria (até) 7 coisas depois do enter pra finalizar a main, colocando o limite pra 4. Se fosse 1024, seria 1024 + 3 = 1027. Nunca teria guardado mais do que 4.

**A tarefa do tipo 4 está bem lenta em 200 milissegundos. Agora, mudou de 200 para 1000; sendo muito lerda:** na versão anterior, sem limite iria acumular muita coisa e travar o computador ao deixar rodando por dias, horas. Mas, agora por conta do limite do semáforo livre não é pra ter esse problema.

**Se na tarefa do tipo 4 colocar 1 milissegundos; ele não printa nada a mais quando para o programa; por estar muito rápida, ela dá conta das coisas guardadas no vetor.** Pode ser que tenha até 7 coisas no vetor, mas de 4 não passa. Ao parar, não sobrou nada no vector. Quando não dá conta se acumula indo até o limite semáforo livre + 3 coisas (tarefas 1,2 e 3).

**8.1.4.Arquivo: Programa.java**

import java.util.Vector;

import java.util.concurrent.Semaphore;

public class Programa

{

public static void main (String[] args)

{

try

{

Vector<Character> armazenamento = new Vector<Character> ();

Semaphore livre = new Semaphore (1024,true);

Semaphore ocupado = new Semaphore (0,true);

System.out.println ("Tecle ENTER para ativar as tarefas e");

System.out.println ("Tecle novamente ENTER para terminar o programa.");

Teclado.getUmString();

TarefaDoTipo1 t1 = new TarefaDoTipo1 (armazenamento, livre, ocupado);

t1.start ();

TarefaDoTipo2 t2 = new TarefaDoTipo2 (armazenamento, livre, ocupado);

t2.start ();

TarefaDoTipo3 t3 = new TarefaDoTipo3 (armazenamento, livre, ocupado);

t3.start ();

TarefaDoTipo4 t4 = new TarefaDoTipo4 (armazenamento, livre, ocupado);

t4.start ();

Teclado.getUmString();

t4.morra ();

t3.morra ();

t2.morra ();

t1.morra ();

/\*

t1.join();

t2.join();

t3.join();

t4.join();

\*/

System.out.println ("Execucao do programa finalizada.");

}

catch (Exception erro)

{} // sei que não passei null para o construtor de nenhuma das tarefas

}

}

+++++++++++++++++++++++++++++++++++++++++

**8.1.5.Arquivo: TarefaDoTipo1.java**

import java.util.Vector;

import java.util.concurrent.Semaphore;

public class TarefaDoTipo1 extends Thread

{

Vector<Character> armazenamento;

Semaphore livre;

Semaphore ocupado;

public TarefaDoTipo1 (Vector<Character> armz, Semaphore lvr, Semaphore ocp) throws Exception

{

if (armz==null)

throw new Exception ("Armazenamento ausente");

if (lvr==null)

throw new Exception ("Livre ausente");

if (ocp==null)

throw new Exception ("Ocupado ausente");

this.armazenamento = armz;

this.livre = lvr;

this.ocupado = ocp;

}

private boolean fim = false;

public void morra ()

{

this.fim=true;

}

public void run ()

{

char caractere='a';

while (!this.fim)

{

this.livre.acquireUninterruptibly();

this.armazenamento.add (caractere);

this.ocupado.release();

try { this.sleep (150); } catch (Exception erro) {}

if (caractere=='z')

caractere = 'a';

else

caractere = (char)(((int)caractere)+1);

}

}

}

++++++++++++++++++++++++++++++

**8.1.6.Arquivo: TarefaDoTIpo2.java**

import java.util.Vector;

import java.util.concurrent.Semaphore;

public class TarefaDoTipo2 extends Thread

{

Vector<Character> armazenamento;

Semaphore livre;

Semaphore ocupado;

public TarefaDoTipo2 (Vector<Character> armz, Semaphore lvr, Semaphore ocp) throws Exception

{

if (armz==null)

throw new Exception ("Armazenamento ausente");

if (lvr==null)

throw new Exception ("Livre ausente");

if (ocp==null)

throw new Exception ("Ocupado ausente");

this.armazenamento = armz;

this.livre = lvr;

this.ocupado = ocp;

}

private boolean fim = false;

public void morra ()

{

this.fim=true;

}

public void run ()

{

char caractere='A';

while (!this.fim)

{

this.livre.acquireUninterruptibly();

this.armazenamento.add (caractere);

this.ocupado.release();

try { this.sleep (300); } catch (Exception erro) {}

if (caractere=='Z')

caractere = 'A';

else

caractere = (char)(((int)caractere)+1);

}

}

}

++++++++++++++++++++++++++++++++

**8.1.7.Arquivo: TarefaDoTipo3.java**

import java.util.Vector;

import java.util.concurrent.Semaphore;

public class TarefaDoTipo3 implements Runnable

{

Vector<Character> armazenamento;

Semaphore livre;

Semaphore ocupado;

public TarefaDoTipo3 (Vector<Character> armz, Semaphore lvr, Semaphore ocp) throws Exception

{

if (armz==null)

throw new Exception ("Armazenamento ausente");

if (lvr==null)

throw new Exception ("Livre ausente");

if (ocp==null)

throw new Exception ("Ocupado ausente");

this.armazenamento = armz;

this.livre = lvr;

this.ocupado = ocp;

}

private Thread tarefa = new Thread (this);

public void start ()

{

this.tarefa.start();

}

public void join () throws InterruptedException

{

this.tarefa.join();

}

private boolean fim = false;

public void morra ()

{

this.fim=true;

}

public void run ()

{

char caractere='0';

while (!this.fim)

{

this.livre.acquireUninterruptibly();

this.armazenamento.add (caractere);

this.ocupado.release();

try { this.tarefa.sleep (450); } catch (Exception erro) {}

if (caractere=='9')

caractere = '0';

else

caractere = (char)(((int)caractere)+1);

}

}

}

++++++++++++++++++++++++++++++++++++

**8.1.8.Arquivo: TarefaDoTipo4.java**

import java.util.Vector;

import java.util.concurrent.Semaphore;

public class TarefaDoTipo4 implements Runnable

{

Vector<Character> armazenamento;

Semaphore livre;

Semaphore ocupado;

public TarefaDoTipo4 (Vector<Character> armz, Semaphore lvr, Semaphore ocp) throws Exception

{

if (armz==null)

throw new Exception ("Armazenamento ausente");

if (lvr==null)

throw new Exception ("Livre ausente");

if (ocp==null)

throw new Exception ("Ocupado ausente");

this.armazenamento = armz;

this.livre = lvr;

this.ocupado = ocp;

}

private Thread tarefa = new Thread (this);

public void start ()

{

this.tarefa.start();

}

public void join () throws InterruptedException

{

this.tarefa.join();

}

private boolean fim = false;

public void morra ()

{

this.fim=true;

}

public void run ()

{

while (!this.fim)

{

this.ocupado.acquireUninterruptibly();

char caractere = this.armazenamento.get(0);

this.armazenamento.remove(0);

this.livre.release();

System.out.println (caractere);

try { this.tarefa.sleep (200); } catch (Exception erro) {}

//try { this.tarefa.sleep (150); } catch (Exception erro) {}

//try { this.tarefa.sleep (100); } catch (Exception erro) {}

//try { this.tarefa.sleep (50); } catch (Exception erro) {}

}

while (this.armazenamento.size()!=0)

{

this.ocupado.acquireUninterruptibly();

char caractere = this.armazenamento.get(0);

this.armazenamento.remove(0);

this.livre.release();

System.out.println (caractere);

try { this.tarefa.sleep (600); } catch (Exception erro) {}

}

}

}

+++++++++++++++++++++++++++++++++++++

**8.1.9.Arquivo: Teclado.java**

//import java.io.BufferedReader;

//import java.io.InputStreamReader;

//import java.io.IOException;

import java.io.\*;

public class Teclado

{

private static BufferedReader teclado =

new BufferedReader (

new InputStreamReader (

System.in));

public static String getUmString ()

{

String ret=null;

try

{

ret = teclado.readLine ();

}

catch (IOException erro)

{} // sei que nao vai dar erro

return ret;

}

public static byte getUmByte () throws Exception

{

byte ret=(byte)0;

try

{

ret = Byte.parseByte (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Byte invalido!");

}

return ret;

}

public static short getUmShort () throws Exception

{

short ret=(short)0;

try

{

ret = Short.parseShort (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Short invalido!");

}

return ret;

}

public static int getUmInt () throws Exception

{

int ret=0;

try

{

ret = Integer.parseInt (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Int invalido!");

}

return ret;

}

public static long getUmLong () throws Exception

{

//long ret=(long)0;

//long ret=0;

long ret=0L;

try

{

ret = Long.parseLong (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Long invalido!");

}

return ret;

}

public static float getUmFloat () throws Exception

{

//float ret=0;

//float ret=(float)0.0;

float ret=0.0F;

try

{

ret = Float.parseFloat (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Float invalido!");

}

return ret;

}

public static double getUmDouble () throws Exception

{

//double ret=0;

//double ret=(long)0;

//double ret=0L;

double ret=0.0;

try

{

ret = Double.parseDouble (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Double invalido!");

}

return ret;

}

public static char getUmChar () throws Exception

{

char ret=' ';

try

{

String str = teclado.readLine ();

if (str==null)

throw new Exception ("Char invalido!");

if (str.length() != 1)

throw new Exception ("Char invalido!");

ret = str.charAt(0);

}

catch (IOException erro)

{} // sei que nao vai dar erro

return ret;

}

public static boolean getUmBoolean () throws Exception

{

boolean ret=false;

try

{

String str = teclado.readLine ();

if (str==null)

throw new Exception ("Boolean invalido!");

if (!str.equals("true") && !str.equals("false"))

throw new Exception ("Boolean invalido!");

ret = Boolean.parseBoolean (str);

}

catch (IOException erro)

{} // sei que nao vai dar erro

return ret;

}

}

**+++++++++++++++++++++++++++++++++++++++++++++++++++**

**8.2.Atividade: Leia-me.txt**

Implemente para a classe Conjunto os seguintes métodos:

// retornar um conjunto com todos os elementos do this e mais

// todos os elementos do conj, sem repetição de elementos

public Conjunto<X> uniao (Conjunto<X> conj) throws Exception

{

//...

}

// retornar um conjunto com os elementos comuns ao this e ao conj

// retornar um conjunto com os elementos comuns ao this e ao conj

public Conjunto<X> inteseccao (Conjunto<X> conj) throws Exception

{

//...

}

// retornar um conjunto com todos os elementos do this que

// não são também elementos do conj

public Conjunto<X> menos (Conjunto<X> conj) throws Exception

{

//...

}

// retornar true se o this contem o conj, ou seja se todos

// os elementos do conj forem também elementos do this;

// retornar false, caso contrário

public boolean contem (Conjunto<X> conj) throws Exception

{

//...

}

// retornar true se o conj contém o this, ou seja se todos

// os elementos do this forem também elementos do conj;

// retornar false, caso contrário

public boolean estaContido (Conjunto<X> conj) throws Exception

{

//...

}

**9. Pasta: Aula 27-10-2021**

**Pasta: 2021-10-27.A**

**Arquivo: Aula 27102021\_Vector\_DS201\_.mp4**

**AULA 9 – Vector (dia 27 Out 2021)**

**Na videoaula de hoje vai explicar um pouco mais sobre a classe Vector; já conhecemos superficialmente essa classe na aula de thread que compartilhava a estrutura de dados que se chamava armazenamento; ela era um vector de caractere.**  Hoje, nesse exemplo (arquivo: Programa.java) é uma classe Vector que armazena Strings.

**Em Java, a gente pode ter classes que tem parâmetros , a gente está acostumado a ver métodos com parâmetros, mas não estamos acostumados a ver classes com parâmetros.** E em java, é possível ter classes com parâmetros e a classe Vector é um exemplo de uma classe que tem parâmetros; o String posto entre os sinais de menor e maior que significa o parâmetro passado para a classe Vector. Pode ter um parâmetro ou mais.

**Quando as classes têm parâmetros, os parâmetros dessas classes sempre serão outra classe.** Nessa aula, o parâmetro da classe Vector é a classe String. Na aula anterior o vector armazenamento tinha como parâmetro a classe wrapper Charactere.

**O significado do parâmetro depende da classe que a recebe. Na classe vector, o Stirng significa que eu quero um vector para armazenar strings.** Quando eu declaro v1 é um vector para armazenar strings. Então, quando declaramos um objeto de uma classe e a classe tem parâmetros , esse objeto vai ser de um tipo parametrizado; no caso Vector de String. O v1 é um vector específico de string. Então, v1 é do tipo Vector de String .

**Pelo fato de Vector ter parâmetros, Vector é uma classe genérica. Como será que é feita uma classe genérica, como ela funciona?**

**Vocês devem se lembrar que no exemplo do vídeo passado, antes de começar o programa colocava o comando import java.util.vector , significa que eu estou usando a classe Vector da biblioteca java.util através da importação.** Quando vai usar várias classes de biblioteca, para evitar de dar muitas linhas de import no começo do programa; você pode colocar:

Import java.util.\*; // significa que você quer usar várias classes da biblioteca java.util. Então, a gente tinha isso escrito no começo do nosso programa do vídeo que compartilhava threads.

**No exemplo de hoje, não tem o comando import para importar as bibliotecas de java. Porque? Porque eu tenho uma alternativa.**

**Se eu tivesse posto o comando de importar o vector da classe java.util; dentro da classe executável não precisaria colocar a biblioteca ao declarar o objeto v1 e nem na sua instanciação.** Assim, eu poderia fazer:

Vector <String> v1 = new Vector <String> ();

**Quando tem o comando import no começo do programa antes de chamar a main, você escreve uma vez a biblioteca com a classe desejada** e evita de escrever a biblioteca toda vez que precisar chamar a classe Vector, independente do uso.

**Então, posso fazer de um dos dois jeitos. Geralmente, usa o import java.util.vector antes do programa pois se livra de escrever java.util** no programa inteiro quando for usar vector.

**Nesse programa, não usou o import para a classe Vector pois quer demonstrar como a classe Vector foi feita.** Logo, eu criei a classe Vector (arquivo: Vector.java) . Para criar a classe Vector, eu fiz public class Vector <X>. E no meu programa eu queria usar os dois vector: o vector que eu fiz (Vector <String> v2) e o que está pronto (Vector <String> v1).

**Se eu escrevesse o import antes da classe executável (dentro do programa.java), toda vez que eu escrevesse Vector ele ia achar que é o Vector da biblioteca.** Como na primeira declaração não escrevi o import (é o objeto v1), eu coloquei java.util.Vector e ele entende que é o vector que está pronto. E no vector v2, por não ter escrito java.util, ele entende que é o vector que eu fiz a partir da classe que eu criei (vector.java).

**Então, não usei import para ter 2 classes vector a serem usadas no mesmo programa (programa.java). :** a que está pronta pela biblioteca e a que eu fiz. Perceba que eu estou usando v1 e v2 para fazer as mesmas coisas.

**No v1, declaro um vector de string da biblioteca com a classe pronta vector, instancio o vector da biblioteca (como new java.util.vector <String>).** No v1 eu chamo 3x o método add para preencher o vector com as strings: C, C++ e JAVA, uma em cada posição respectivamente. Depois printo v1, no qual quando eu printo um objeto automaticamente o método toString é chamado e o resultado dele será printado. Eu mando remover o que estiver na posição 1 e depois printo v1 novamente.

**No v2, declaro Vector string a partir da minha classe vector. Instancio o vector da minha classe vector. No v2 eu chamo 3x o método add para preencher o vector com as strings: C, C++ e JAVA, uma em cada posição respectivamente.** Mando printar o v2; depois eu mando remover o que estiver na posição 1 do v2. Depois eu printo v2.

**Ele fez a classe Vector bem semelhante, praticamente tentando imitar a classe Vector original de Java para conseguir usar as duas do mesmo jeito.** Percebam que na hora que rodar o programa, o resultado tanto para a classe pronta como para a classe que eu fiz, são os mesmos.

Na execução, apareceu:

[C, C++, Java] // primeiro print de v1

[C, Java] //segundo print de v1

**//** linha branca gerada pelo println sem nenhum parâmetro, foi apenas pra separar

[C, C++, Java] // primeiro print de v2

[C, Java] //segundo print de v2

**---------------------------**

Program exited with code: 0

Pressione qualquer tecla para continuar...

(Parei em 15min48s).

**No primeiro bloco eu lido com v1, o v1 vem da classe vector já pronta da biblioteca; no segundo bloco eu lido com v2, cujo vector v2 é da minha classe que eu fiz.** Percebam que as saídas foram idênticas para ambos os vetores. Aparentemente a classe que eu fiz funciona e está idêntica a classe já pronta da biblioteca. Vamos querer entender o que é que foi implementado e como foi feito.

**Na classe Vector que ele implementou, não tem muitos métodos. Tem um construtor, o método size, o método private redimensioneSe (é um método para me ajudar na implementação, para ser usado apenas dentro da classe), método add, método get, método remove, método toString, método equals. Esses métodos estão longe de ser todos os métodos que existem na classe Vector da biblioteca.**

**A classe Vector da biblioteca tem muito mais métodos além desses, como o contains (que serve para verificar se o vector contains ou não uma certa informação), capacity (quantas coisas cabem naquele vector).** Então tem um bando de métodos que estão prontos na biblioteca java.util mas não sabiam da existência e que não foi feito na nossa classe Vector. Na nossa classe Vector, temos os poucos métodos que nós andamos usando nos exemplos da aula passada (get, remove, add) e dessa aula. Não fez mais métodos porque não era criar uma réplica da classe vector e sim passar como essa classe funciona.

**Depois, pode procurar os demais métodos na documentação e implementar na classe vector que está sendo montada.** Sempre que a gente tem um parâmetro, como no método remove tem o parâmetro posição; o parâmetro chamado posição é um parâmetro formal. Porque é o parâmetro que a gente tem quando está implementando o método. Então estou implementando o método remove, que tem o parâmetro inteiro que chamo de posição. Eu programo o meu método usando o parâmetro posição.

**Então no método, eu verifico se a posição for menor que zero ou a posição for maior que this.qtd-1 , traz a exceção específica com o parâmetro posição. Depois faz um for que começa com i valendo posição +1 , enfim.** Então usamos a palavra posição. Mas a gente sabe que quando o método remove for chamado, alguma coisa será passada para entrar no lugar de posição; como de fato acontece no programa (main) quando chamo o método remove eu passo 1 como posição. Esse 1 vai entrar no lugar da posição e eu vou verificar se 1 é menor do que zero (não é) ou se 1 é maior que this.qtd-1 ; vou fazer o for com a minha variável inteira valendo 1+1 = 2 . Ou seja, o que eu passar como parâmetro no remove dentro da main vai entrar no lugar da palavra posição e vai ser usado em todos os lugares que usa posição; na main posição é 1.

**O que é passado como 1 do lado de remove é um parâmetro; assim como a posição (que foi implementado).** O parâmetro que a gente usa na main quando o método já está implementado é o que chamamos de parâmetro real, é o que a gente aciona quando usa. O parâmetro real sempre entra no lugar do parâmetro formal na hora do uso pela main.

(Parei em 27min14s).

**Quando a gente pensa que a classe tem parâmetro, acontece a mesma coisa. No arquivo Vector.java, estamos definindo uma classe chamada Vector que recebe como parâmetro X.** O X poderia ser qualquer coisa entre <> ,pois é um parâmetro formal. O X é um parâmetro formal, quando eu for usar essa classe, algo vai entrar no lugar do X que será o parâmetro real. Então, na main quando escrevo Vector <String> v2 , String é o que estou passando como parâmetro real que vai substituir o X da minha classe Vector; isso vai acontecer em todos os lugares que aparecer X na minha classe. Isso quer dizer que o meu v2 terá um atributo chamado elem que tem um vetor de String.

**Nós temos o método add que recebe parâmetro algo que chamo de x do tipo X. Como no lugar do X entrou String, significa que pro vector v2, o método add vai receber o parâmetro x do tipo String.** O método get retorna algo do tipo X. Significa que nesse programa, onde eu declarei v2 como vector de String, coloquei String como parâmetro real. Então, String entrou no lugar de <X> e tudo que for X vai ser String, inclusive o retorno do método get.

**Agora vai explicar a lógica da implementação da classe Vector de X. Primeiramente, declarou uma constante (porque tem a palavra final) inteira privativa, chamada tamanho inicial que recebe 10**; porque inicialmente, todo vector que eu declarar vai ter lá dentro um atributo chamado elem que é um vetor de coisas do tipo X; e esse vetor terá o tamanho inicial de 10. No caso, elem será um vetor de String. E também terá um atributo qtd que é int.

**O construtor de vector faz primeiramente a colocação de zero no atributo qtd, e ele dimensiona o vetor elem com tamanho vector.tamanho\_inicial ; ou seja, com tamanho dez. Em vector.tamanho\_inicial tem um truque:**  em java, não aceita que eu faça new x dá erro de compilação, sendo X o parâmetro de uma classe genérica. O que eu gostaria de fazer é: this.elem = new X[Vector.TAMANHO\_INICIAL]; mas ele não aceita, não compila. Qual é o truque? Eu crio um novo vetor de object com tamanho inicial que é 10; e depois que eu criei o vetor de object, faço o typecast, eu converto o vetor de object que eu criei de new Object com tamanho inicial de 10, eu converto isso para um vetor de coisas do tipo X e guardo em elem e então ele aceita. Ele não aceita se for vetor de X direto, primeiro tenho que criar o vetor de object para depois converter para vetor de X.

**Então, o construtor de vector instanciou o atributo elem e também inicializou o atributo qtd com zero.** Fiz o método chamado size (usado na aula passada) e ele retorna this.qtd.

**O this.qtd inicialmente é zero, mas por exemplo, no método add, this.qtd aumenta em 1 unidade.** Por outro lado, se precisar remover (pelo método remove) qtd fica sendo o valor dele-1. Então, o atributo qtd sempre reflete o que tem guardado no meu vetor.

**O método add , ele recebe como parâmetro o x que será do tipo X. E X é o parâmetro da minha classe genérica Vector.** Nesse programa X é String, portanto x será do tipo String. O método add verifica se a quantidade de coisas que tenho guardadas é igual ao comprimento do meu vetor elem; se for verdadeiro, significa que x não vai caber porque o vetor vai estar cheio. Eu poderia lançar uma exceção dizendo que não cabe; mas a classe Vector faz outra coisa. Ela faz o vetor aumentar de tamanho pelo método redimensioneSe e passo como parâmetro 2.0F , significa 2 do tipo float. ; quando coloca 2.0 seria double, se colocasse 2 seria int. O método redimensioneSe vai dobrar o tamanho do meu vetor, porque recebeu 2.0F como parâmetro, é o que vai redimensionar o meu vetor. Assim sendo, o vetor elem vai ter o dobro do tamanho inicial; depois eu vou na posição qtd e guardo o x. O this.qtd aumenta em 1 unidade (a cada vez que chamar add qtd aumenta em 1).

**O método get serve para recuperar um valor que esteja guardado na posição que eu indicar. Esse método foi usado na aula anterior (do vector pela biblioteca java.io). Eu passo a posição e verifica se a posição for negativa ou se for maior que this.qtd-1**; como eu começo a guardar do zero, this.qtd-1 seria a última posição em que tenho algo guardado. Se eu tenho 10 coisas guardadas, vai de 0 a 9. Então, se a posição entrar nesse if das posições, eu lanço a exceção java.lang.ArrayIndexOutOfBounds (posição); o construtor da classe ArrayIndexOutOfBounds tem como parâmetro posição, pois uma exceção é uma classe. Uma exceção é um objeto de uma classe instanciada por throw new (nome da exceção). Então, o construtor da exceção aceita como parâmetro um int que deve ser a posição inválida que estou tentando acessar. Caso não tenha dado erro, eu vou pegar o vetor, o que tiver na posição indicada vou retornar em elem.

**Em seguida do método get, temos o método remove. O método remove também tem um parâmetro chamado posição. Vou remover o que tiver nessa posição indicada. Primeiro vou validar se a posição é inválida (da mesma forma que no método get, pois a exceção é a mesma).** Não estando errado, entro na repetição de for. O for tem a variável i começando na posição que quero remover +1 (posição seguinte daquela que eu quero remover); eu pego o que tiver na posição adiante e guarda na posição atrás (que é a posição que foi passada no parâmetro). Então ele pega o que está lá na frente e guarda lá trás bem onde eu queria remover. E removeu porque eu guardei por cima. Aumenta o i. O for faz sempre a mesma lógica descrita, enquanto o i for menor que this.qtd.

**Imagina que no meu vector eu tenha guardado: C (posição 0), C++ (posição 1), Java (posição 2). Imaginem que eu faça como no programa que faz remove (1). O método remove vai receber 1.** Então, eu quero remover C++. Primeiro ele valida: posição é menor que zero? Não porque posição é 1. Posição é maior que a quantidade de coisas menos 1? Não porque qtd é 3 ; e qtd-1 = 2; então o limite é a posição 2 e eu estou na posição 1. Então, eu não fiz nada errado e não estou na exceção.

**Para o for, i começa valendo posição +1 , então i =1+1 = 2. Ele vai pegar o tiver na posição 2 (Java) e vai jogar na posição 1 (2-1=1) .** Então, Java vai ser jogado no lugar de C++; e eliminou C++. Só que o meu for aumenta o i, i foi pra 3. Lembrando que agora, eu teria:

0 1 2 3

C Java Java Phyton

O meu vetor pega o que tem na posição 3 (Phyton) e vai jogar na posição 2 (Java). Então:

0 1 2 3

C Java Phyton Phyton

**Então ele joga da frente pra trás , até chegar na quantidade que tem no vetor. No nosso caso já chegou porque o 3 é o último. Parou o for.** Depois que parou o for, ficou o último elemento (3) repetido, porque o último foi jogado pra trás, mas a posição 3 não foi eliminada. Então, qtd =4. (4 coisas guardadas). Então, diminuo qtd em 1 unidade, então passa de 4 para 3. Vou na posição 3 e coloco nulo para tirar o Phyton da posição 3. Logo, em qtd teremos 3 coisas guardadas. O método vai até qtd. Se qtd é 3 tenho coisa guardada até 2.

**Depois que o método remove removeu a minha posição, eu entro em um if esquisito. Ele vai ver se o comprimento físico do vetor é maior que o meu tamanho inicial que é 10, ou seja, se o meu vetor já aumentou.** E tenho que ver se a quantidade de coisas que tenho guardadas é menor ou igual a 0.25\* o comprimento físico do vetor. Todo mundo sabe que se eu pegar o comprimento físico do vetor e multiplicar por 0,25, eu vou estar pegando 25% do tamanho; pode ser que dê um valor quebrado. Por exemplo, se meu vetor for 4 e pegar 25% dará 1. Mas se o meu vetor for 5 e pego 25%, vai dar valor quebrado acima de 1; por isso que usa float. O math.round é para arredondar para o inteiro mais próximo. Então, se a conta der 1,3 arredonda para 1; se der 1,8 arredonda para 2.

**Logo, estou vendo se a quantidade de coisas que eu tenho guardadas no vetor é menor ou igual a 25% do tamanho do vetor (arredondado).** Se isso acontecer significa que estou ocupando apenas 25% do tamanho ou menos do vetor; o que sobre 75% sem uso. Isso significa que o meu vetor está muito grande. Então, eu diminuo o tamanho para 50% com o método redimensioneSe, evitando desperdício de memória.

**Então, reparem que quando eu adiciono e não cabe, eu dobro. Mas quando eu removo e depois de remover eu ocupo até 25% do vetor, eu diminuo para a metade** (só se o comprimento for maior que o tamanho inicial). O vetor nunca fica menor que o tamanho inicial.

**O método redimensioneSe recebe um número real float que eu chamo de taxaDeRedim. Esse número real será usado para dobrar ou diminuir o vetor na metade. O método aceita a taxa que eu quiser passar**. Esse método é private, somente eu tenho acesso. Os vetores em java, assim como os objetos são ponteiros (variáveis capazes de guardar endereços de memória). Então, foi declarado dentro do método redimensioneSe o vetor X com nome novo. Para instanciar esse vetor, eu crio novo com new Object que tem como tamanho: a taxaDeRedim multiplicado pelo tamanho do vetor (elem) que dará um número real. Para dar um número inteiro, o resultado dessa multiplicação eu multiplico por Math.Round para arredondar pro mais próximo e achar em int. E crio um novo vetor de new Object com o Math Round. Então, o que foi guardado no novo é o endereço de memória de X (após a conversão) onde tem o novo para não ser nulo.

**Agora, eu transporto através do for que começa com i valendo 0, vai aumentando i até que se chegue na quantidade de coisas guardadas. O for pega uma coisa guardada no atributo elem na posição i (que já tenho hoje) e passo para o vetor novo na posição i;** faz elemento a elemento: pega do vetor velho que precisa de redimensionamento e passa pro vetor novo com o tamanho já redimensionado. Depois eu pego o endereço do vetor já redimensionado (o endereço do novo) e guardo no lugar do vetor que precisava de redimensionamento (elem). E agora, elem é um vetor redimensionamento; porque já tem o endereço de novo, tem o tamanho do novo e foi transportado os elementos do vetor velho para o novo.

**Agora, para ver o método obrigatório toString que tem como parâmetro o tipo String. Eu começo declarando a variável ret do tipo String valendo um abre colchete e no fim será retornado o fecha colchete.** Depois do abre colchete, eu percorro o meu vetor com for, onde i vale zero, aumenta de 1 em 1 até chegar na quantidade-1 (até chegar no penúltimo). Eu pego o que tem no ret, concateno com elemento do vetor na posição i que estou e acrescento uma vírgula a direita da posição. É por causa da vírgula que estou indo até o penúltimo; pois até o penúltimo coloco uma vírgula depois do elemento. No último, não terá vírgula.

**Quando chegar no último, chego no if this.qtd>0, onde eu pego o ret, concateno com o último (se qtd é 10, vai até 9).** E na hora de retornar, concateno com fecha colchete.

**O método equals recebe Object obj, vai ver se this é igual a obj e retorna true. Se o obj é igual a null retorna false. Depois eu checo se o tipo do this é diferente do tipo de obj usando o método getClass (descobre a classe do objeto).** Então se a classe do this for diferente da classe obj, eu retorno falso. Significa que estou comparando vector com algo diferente de vector, o que dá falso. Se não entrar no primeiro nem segundo if, é porque estou comparando dois vector e nenhum dos dois é nulo e estão em endereços de memória diferente. Agora tenho que comparar conteúdos. Lembre-se que java acha que obj é do tipo Object ; eu não posso pegar obj.elem porque pro java coisas do tipo Object não tem atributo elem.

**Mas o teste do getClass já me deixou convencido que obj é um vector, porque estou no equals de vector; pois eu chamei um objeto qualquer (o this) .equals. Se obj tiver um tipo diferente de vector dará falso.** Porém, para o java obj foi declarado como Object. Para isso, convenço o java de que obj é um Vector <X> (faço um typecast). Coloca o endereço de memória obj, converto para Vector<X> e guardo na variável vec e ambos terão o mesmo endereço (vec e obj) e terão o mesmo tipo Vector<X>; dessa forma, obj será visto como Vector <X>.

**Logo, o vec entrará em um for onde eu percorro desde i valendo zero até this.qtd e sempre aumento i em 1 unidade. Vou ver se na posição i do this.elem tem alguma coisa diferente a vec.elem [i]; ele retorna falso** (é falso que elem e vec são iguais). Se o meu obj não entrou em nenhum if, eu retorno true.

**9.1.Lista de Arquivos usados na aula 9**

**9.1.1.Arquivo: Vector.java**

public class Vector <X>

{

private static final int TAMANHO\_INICIAL = 10;

private X[] elem;

private int qtd;

public Vector ()

{

//this.elem = new X [Vector.TAMANHO\_INICIAL]; <- não compila

this.elem = (X[])new Object [Vector.TAMANHO\_INICIAL];

this.qtd = 0;

}

public int size ()

{

return this.qtd;

}

private void redimensioneSe (float taxaDeRedim)

{

X[] novo;

//novo = new X [Math.round(this.elem.length\*taxaDeRedim)]; <- não compila

novo = (X[])new Object [Math.round(this.elem.length\*taxaDeRedim)];

for (int i=0; i<this.qtd; i++)

novo[i] = this.elem[i];

this.elem = novo;

}

public void add (X x)

{

if (this.qtd==this.elem.length)

this.redimensioneSe (2.0F);

this.elem[this.qtd] = x;

this.qtd++;

}

public X get (int posicao) // posicao vai de 0 a this.qtd-1

{

if (posicao<0 || posicao>this.qtd-1)

throw new java.lang.ArrayIndexOutOfBoundsException (posicao);

return this.elem[posicao];

}

public void remove (int posicao) throws ArrayIndexOutOfBoundsException

{

if (posicao<0 || posicao>this.qtd-1)

throw new java.lang.ArrayIndexOutOfBoundsException (posicao);

for (int i=posicao+1; i<this.qtd; i++)

this.elem[i-1] = this.elem[i];

this.qtd--;

this.elem[this.qtd] = null;

if (this.elem.length>Vector.TAMANHO\_INICIAL &&

this.qtd<=Math.round(this.elem.length\*0.25F))

this.redimensioneSe (0.5F);

}

public String toString ()

{

String ret="[";

for (int i=0; i<this.qtd-1; i++)

ret = ret+this.elem[i]+", ";

if (this.qtd>0)

ret = ret+this.elem[this.qtd-1];

return ret+"]";

}

public boolean equals (Object obj)

{

if (this==obj)

return true;

if (obj==null)

return false;

if (this.getClass()!=obj.getClass())

return false;

Vector<X> vec = (Vector<X>)obj;

for (int i=0; i<this.qtd; i++)

if (!this.elem[i].equals(vec.elem[i]))

return false;

return true;

}

}

++++++++++++++++++++++++++++++++++++++++++

**9.1.2.Arquivo: Programa.java**

public class Programa

{

public static void main (String[] args)

{

java.util.Vector<String> v1 =

new java.util.Vector<String> ();

v1.add("C");

v1.add("C++");

v1.add("Java");

System.out.println (v1);

v1.remove(1);

System.out.println (v1);

System.out.println ();

Vector<String> v2 =

new Vector<String> ();

v2.add("C");

v2.add("C++");

v2.add("Java");

System.out.println (v2);

v2.remove(1);

System.out.println (v2);

}

}

**10. Pasta: Aula 28-10-2021**

Pasta: 2021-10-28.A

**Arquivo: Sincronismo\_Aula28102021\_DS201.mp4**

**AULA 10 – Sincronismo ( Dia 28 Out 2021)**

**9.1.Lista de Arquivos usados na aula 10**

**9.1.1.Arquivo: Vector,java**

Public class Vector <x>

{

private static final int TAMANHO\_INICIAL = 10;

private x [] elem;

private int qtd;

public Vector ()

{

//this.elem = new X [Vector.TAMANHO\_INICIAL]; ← não compila

this.elem = (X[] new Object [Vector.TAMANHO\_INICIAL];

this.qtd = 0;

}

public int size ()

{

return this.qtd;

}

private void redimensioneSe (float taxaDeRedim)

{

Olá pessoal, **eu gostaria nesse vídeo aula de explicar para vocês o conceito de sincronismo** e mais do que explicar o conceito de sincronismo. Eu gostaria de fazer você compreender, quando que o sincronismo é necessário e quando ele não é. E também comentar com vocês e mostrar para vocês como que o sincronismo pode ser implementado. Para compreender a ideia nós vamos voltar na classe Vector que nós tínhamos feito, e sobre a qual nós tínhamos comentado na vídeo-aula anterior.

**Todos devem se lembrar que já existe uma classe Vector que inclusive foi usada nos nossos dois últimos exemplos de implementação de programas multitarefa.** E então nós sabemos que existe uma classe Vector pronta na linguagem Java. Já que ela faz parte da biblioteca java.util e ela foi usada na implementação conforme eu já disse, na implementação dos nossos dois últimos programas multitarefa que foram apresentados em vídeo aulas; não na passada, mas na retrasada e na anterior a retrasada. Nós fizemos essa classe Vector e a gente fez um programa tem esse programa aqui (ver arquivo: Programa.java da pasta meu vector sincronizado):

**10.1.Lista dos arquivos usados na aula 10**

**10.1.1.Arquivo: Programa.java**

Public class Programa

{

public static void main (string[] args)

{

java.util.Vector<String> v1 =

new java.util.Vector<String> ();

v1.add(“C”);

v1.add(“C++”);

v1.add(“Java”);

System.out.println (v1);

v1.remove(1)

System.out.println (v1);

System.out.println ();

Vector<String> v2 = new Vector<String> () ;

v2.add(“C”);

v2.add(“C++”);

v2.add(“Java”);

System.out.println (v2);

v2.remove(1)

**Ao qual usava a classe Vector que já vinha pronta no Java e também usava a classe Vector que nós fizemos.** A gente rodou o programa exemplo, viu que programa funcionava e que aparentemente ambas as classes: a que vinha pronta e também a que a gente fez funcionavam da mesma maneira.

**Tirando o fato de que classe Vector (que já está pronta) tem vários métodos que a gente nem implementou aqui; a gente implementou apenas alguns.** Mas, eu gostaria de dizer para vocês que se eu pegasse essa classe Vector que nós fizemos; e ao invés de usá-la nesse programa de testes que a gente fez. Se eu usasse nossa classe Vector naqueles programas multitarefas que compartilhavam uma estrutura de dados e a gente provavelmente iriam verificar mau funcionamento.

**Desta forma, a gente provavelmente ia verificar erros no nosso programa.; porque isso já que aparentemente a classe Vector que a gente fez funciona do mesmo jeito que a classe Vector que já está pronta.** Porque a nossa provavelmente causaria mal funcionamento naqueles programas nos quais havia multitarefas e havia compartilhamento de uma estrutura de dados. Que inclusive era um Vector e que se chamava armazenamento.

**Porque essa classe causaria problemas e a classe pronta da biblioteca não causaria. Vamos tentar entender isso então agora. Para facilitar nossa compreensão do que é que pode acontecer quando a gente usar a nossa classe, lá naquele programa multitarefas**. Eu gostaria de que vocês imaginassem a seguinte situação: imaginem um computador que só tenha um núcleo de processamento. Então as várias tarefas que eu colocar para rodar ao mesmo tempo, elas vão se revezar na execução e vão se revezar no uso desse processador único; que nós estamos imaginando que exista.

**Então eu vou criar várias tarefas, e as tarefas vão funcionar da seguinte maneira: uma dela entra em execução e executa durante um certo tempo. Depois, ela para e entra outra para executar um certo tempo;** a segunda para e entra uma terceira tarefa para executar ao mesmo tempo. E assim sucessivamente, até que pare e entre a última para executar. E quando essa última executar um certo tempo ela vai parar e a primeira vai voltar a executar.

**Vamos imaginar essa situação que nos ajuda a compreender o problema. Mas o problema não se manifesta só nessa situação, o problema pode se manifestar em qualquer situação;** tanto faz se a gente tem um núcleo de processamento, 2 núcleos, 4 núcleos ou outro núcleos: isso não faz diferença. O problema pode acontecer de qualquer maneira, mas é mais fácil gente entender o problema quando a gente pensa nesse revezamento de tarefas para serem executadas no processador.

**Fica mais fácil do que pensar que as tarefas estão exatamente ao mesmo tempo sendo executados. Então, vamos pensar desse jeito que eu propus:**  tem só um processador e as tarefas vão se revezar para serem executadas no processador. Imaginem que as tarefas usem um objeto da classe Vector.

**Podemos pensar nas tarefas que já foram implementadas e compartilhavam um objeto da classe Vector chamado armazenamento.** Podemos pensar nas tarefas que a gente já conhece mesmo. Imagine que uma dessas tarefas, num certo momento, chame o método ADD do Vector armazenamento. Verifique o arquivo Vector.java:

public void add (X x)

{

if (this.qtd==this.elem.length)

this.redimensioneSe (2.0F);

this.elem[this.qtd] = x;

this.qtd++;

}

**Quando o método add que pertence a classe armazenamento for chamado, ele vai verificar se quantidade de coisas guardadas no Vector é igual ao comprimento que fisicamente tem o vetor elem do Vector.** Se for igual vai acontecer um redimensionamento; senão for igual não vai acontecer esse redimensionamento. Agora imagine vocês que, logo em seguida, o método add por chamado por aquela tarefa, vai lá no Vector. Ele pega o atributo elem, indexa com atributo qtd e guarda x. Então x foi guardado no vector. Agora, deveríamos aumentar o atributo quantidade. Mas, digamos que não deu tempo de aumentar; deu tempo de guardar o x, mas não deu tempo de aumentar a quantidade.

**Porque acabou o tempo daquela tarefa. Aquela tarefa tinha um tempo para ficar executando processador.** Digamos que justamente agora, por azar, acabou o tempo. Deu tempo de guardar o x, mas não deu tempo de ir aumentar a quantidade. Aquela tarefa para, porque o tempo dela acabou. E outra tarefa entra em execução.

**Se essa outra tarefa executar coisas que não envolvam o Vector, não vamos ter nenhum problema. Mas, se essa outra tarefa também chamar o método add. O que vai acontecer?** Vai verificar a necessidade de redimensionamento. Se houver necessidade, vai ser feito o redimensionamento e vai ser guardado x no vetor chamado elem, do Vector, na posição indicada pela quantidade.

**Note que a outra thread, aquela que não teve tempo de aumentar a quantidade, teve tempo de guardar o x; mas não teve tempo de aumentar a quantidade.** Agora essa tarefa que está em execução, vai lá e guarda um x na posição indicada pela quantidade, que precisava ter sido aumentada pela outra tarefa. Mas não deu tempo.

**Qual vai ser a consequência? Esse x (da linha this.elem[this.qtd]) vai ser guardado no vetor, na mesma posição e por cima do que a outra tarefa havia guardado.** E não numa posição seguinte. Porque não deu tempo da outra tarefa aumentar a quantidade. Então, a quantidade continua com o mesmo valor; e essa tarefa que está executando agora, vai guardar o x na mesma posição lá no vetor elem. Ou seja, vai guardar por cima do que a tarefa, que foi interrompido. Porque acabou tempo, na mesma posição que aquela tarefa tinha guardado o que ela precisava guardar. Ou seja, vai acontecer perda de dados; uma tarefa vai lá e guarda uma coisa, enquanto que a outra vai guardar por cima, vai estragar o que a primeira tinha guardado.

**Como se isso não bastasse, essa tarefa que está em execução; depois de armazenar por cima e provocar essa perda de dados. Digamos que ela teve tempo de aumentar o qtd.** Então aumentou, ocorreu perda de dados. Depois, essa tarefa executa mais um pouco e acabou o tempo dela. Digamos que, aquela que foi interrompida, porque tinha acabado tempo antes primeiro; digamos que ela volta a ser executada.

**Na hora em que ela a volta ser executada, lembrem-se que ela conseguiu guardar o x; mas ela não conseguiu aumentar o qtd . Quando ela volta a ser executada; o que ela faz?** Ela aumenta o qtd, porque foi aqui que ela foi interrompida. Ela foi interrompida na eminência de aumentar qtd, já que tinha dado tempo de ela guardar o x. Então, não deu tempo de aumentar qtd; agora, quando ela volta executar ela aumenta qtd.

**Moral da história: as duas coisas foram guardadas uma por cima da outra e o qtd foi aumentado duas vezes. Portanto ficou uma posição inutilizada é a posição nula dentro do meu vetor; porque aumentou uma vez e aumentou de novo.** Então ficou uma posição nula ali no meio, como se tivesse alguma coisa guardada ali; mas de fato não tem nada guardado ali.

**E esse tipo de problema pode acontecer também no get; não precisa ser só no add. Esse tipo de problema pode acontecer no remove, ele está aqui: duas tarefas estão em execução.** Como na verdade, só tem um processador, ele executa uma. Uma chama o método remove e ele executa um monte de coisa, removeu o que tinha que ser removido do vetor. Mas não deu tempo de diminuir qtd.

**Depois ele chama outro remove. o qtd vai estar inconsistente, porque qtd vai estar indicando uma quantidade que já não tem mais.** Porque ele já removeu da tarefa anterior, mas quando ele ia diminuir a quantidade não deu tempo.

**E depois, outra tarefa entra em execução: aquela primeira interrompido que não deu tempo de diminuir o qtd . A próxima que entra em execução, vai encontrar o nosso vetor numa situação inconsistente, numa situação onde eu já tirei alguma coisa do vetor, mas não contabilizei a retirada.** É como se eu tivesse contabilizando que tem 10 elementos guardados, mas na verdade só tem nove. Ou seja, ele vai estar inconsistente.

**Então essa problemática que eu estou mostrando pra vocês, ela não acontece só no add : pode acontecer no get, remove.**

**O método remove pode ser visto abaixo:**

public void remove (int posicao) throws ArrayIndexOutOfBoundsException

{

if (posicao<0 || posicao>this.qtd-1) throw new java.lang.ArrayIndexOutOfBoundsException (posicao);

for (int i=posicao+1; i<this.qtd; i++)

this.elem[i-1] = this.elem[i];

this.qtd--;

this.elem[this.qtd] = null;

if (this.elem.length>Vector.TAMANHO\_INICIAL && this.qtd<=Math.round(this.elem.length\*0.25F)) this.redimensioneSe (0.5F);

}

**Ela pode acontecer numa mistura onde add e a outra das tarefas faz get. Mas a tarefa que faz get executou e parou em this.elem[this.qtd] =x, onde deu tempo dela guardar, mas não deu tempo de aumentar.** Na hora em que ela é interrompida e a outra tarefa entra em execução, e for fazer um get, vai estar numa situação inconsistente. Porque ele vai estar pegando alguma coisa de uma posição do vetor.

**Por exemplo, digamos que o vetor tinha 10 coisas guardadas. E aquela primeira que foi interrompida lá, estava guardando e essa coisa que está sendo guardada, seria a décima primeira coisa guardada.**  E ele guardou a décima primeira coisa guardada, mas não deu tempo de aumentar this.qtd para 11. E foi interrompida, deu tempo de guardar, mas não deu tempo de aumentar para 11.

**Na hora que eu fizer um get, querendo a posição 10, querendo última posição; ele vai dar exceção falando que a posição não existe. Mas foi guardado uma décima primeira coisa lá, mas não foi contabilizada.** Porque guardou e não deu tempo de contabilizar. Então, se eu fizer um get, ele pode falar que eu estou querendo pegar uma posição que não existe, mas na verdade é uma posição que já existe, já foi guardado alguma coisa ali; só não foi contabilizada no qtd.

**Então às vezes uma mistura: uma tarefa está usando add, outra tarefa está usando get. Uma tarefa está usando add enquanto que outra tarefa está usando remove.** Pode ter uma mistura de métodos, uma tarefa usando uma coisa e outra tarefa usando outra coisa e dar problema.

**Por exemplo, imagina que uma das tarefas que está em execução usa o método add e executa até dar tempo de guardar o x no vetor, mas não deu tempo de aumentar a quantidade.** Então, a outra tarefa essa execução, porque essa foi interrompido (a do método add) e a outra quando entrar vai fazer um remove. Quando ela for fazer um remove, ela vai fazer um for até qtd, mas o qtd está errado; porque a que estava executando antes não teve tempo de atualizar o qtd.

**O qtd está errado, ou seja, o remove não vai funcionar direito; vai bagunçar todo o funcionamento das nossas tarefas. Espero que vocês tenham entendido por quê que essa bagunça iria acontecer, por conta de interrupções em momentos estratégicos**. Uma interrupção no momento onde, deu tempo de atualizar um atributo, mas não deu tempo de atualizar o outro. Deu tempo de eu tirar uma coisa do vetor, deslocar os elementos do vetor, mas não deu tempo de atualizar a quantidade. Deu tempo de eu guardar alguma coisa no vetor, mas não deu tempo de contabilizar; que agora a quantidade aumentou.

**Então, quando há interrupção, eu estou com threads rodando, quando acontece e estamos imaginando só tem um processador; então elas se revezam. Tem uma execução e as outras estão esperando.** Quando acontece dessa uma que está execução, ser interrompida no momento estratégico: como esse que eu estou dizendo, deu tempo de guardar, mas não deu tempo de contabilizar o que foi guardado.

**Deu tempo de remover o que tinha no vetor, deslocar todos os elementos para esquerda, para sumir com aquilo que eu queria remover. Mas não deu tempo de contabilizar que a remoção agora, fez o meu Vector ter uma quantidade um a menos.** Então, quando acontece nesses momentos estratégicos; a gente enfrenta a possibilidade de erros.

**Porque a nossa tarefa foi interrompida enquanto estava em execução, no momento estratégico: em que deu tempo de ela excluir do vetor, mas não deu tempo de ela diminuir a quantidade. Ou então, deu tempo para ela guardar no vetor, mas não deu tempo dela aumentar a quantidade.** Então, quando acontece a interrupção nesses momentos estratégicos; a gente vai acabar tendo problemas. Se a outra tarefa que entrar em execução, for fazer alguma coisa envolvendo o vector. Porque ela vai fazer alguma coisa usando vector, e o Vector está numa situação de inconsistência. O Vector está numa situação onde uma coisa foi guardada, mas não foi aumentado a quantidade; ou então uma coisa foi removida, mas não foi diminuída a quantidade. Ou seja, o vector está em uma situação inconsistente, porque a tarefa que estava usando-o; começou a fazer alguma coisa, não chegou acabar e foi interrompida. E a próxima tarefa que entra em execução acaba se for usar Vector; acaba usando um vector que está nessa situação de inconsistência.

**Logo, acaba tendo erros de execução. Agora se s acontece de uma tarefa ser interrompida, mas ela foi interrompida no momento em que ela não estava usando o vector. Vai dar problema nenhum para os outros.** Digamos que uma tarefa esteja usando Vector seja interrompida no momento estratégico, mas as tarefas que vão executar depois não vão executar coisas que não vão precisar do vector; também não vamos ter problemas.

**Então, não é determinístico que eu vou até um problema; não é certeza que eu vou ter um problema. (20min56s). Existe chance de eu ter um problema. Quando isso pode acontecer?** Quando uma tarefa é interrompida no momento estratégico, ela estava guardando e não acabou de fazer tudo que tinha que fazer para guardar; e foi interrompida bem no meio. Assim, deu tempo de fazer uma parte. mas não deu tempo de fazer o resto.

**Ou então, deu tempo de remover do vetor, mas não deu tempo de fazer o resto. Então, quando uma tarefa está em execução, ela está usando o vector e ela é interrompida no momento estratégico.** Se a tarefa que executar depois que ela foi interrompida, não usar o Vector, não vai dar problema nenhum. Mas, se usar vai dar problema, porque vai usar um vector que está numa situação inconsistente: uma coisa foi guardada mas não foi contabilizada, que está em uma situação, onde uma coisa foi removida mas não foi contabilizada .

**Ou seja, está inconsistente. Logo, a outra tarefa vai ter erros de execução porque está usando vector que estava bagunçado, ele não está ok, ele não está consistente.** Ele está numa situação anômala, por isso o problema acontece.

**E porque que esse problema não acontece com a classe Vector já vem pronta? O que é classe Vector que já vem pronta tem, que essa nossa classe Vector não tem?** Vamos falar sobre isso agora. E exatamente o que, faz com que a nossa classe Vector tenha essa diferença de funcionamento; se a gente comparar com a classe Vector que nos é oferecida, pela própria linguagem Java na biblioteca java.util?

**Porque que, se eu usar lá nas minhas tarefas; porque se eu compartilhar entre as minhas várias tarefas, uma estrutura de dados do tipo Vector;** mas do tipo Vector que já tá pronto em Java na biblioteca java.util . Porque que eu não vou enfrentar nenhum problema?

**Mas, se eu compartilhar entre as minhas várias tarefas, um objeto na estrutura de dados da classe Vector que eu fiz; eu vou enfrentar problemas?** Porque a classe que eu fiz tem de diferente da classe Vector que já está pronta? Ou então, porque a classe vector, que já está pronta, tem de diferente dessa minha? O que a outra tem que a minha não tem ou vice-versa ?

**Basicamente o que acontece é que os métodos da minha classe** **Vector que já está pronta, os métodos públicos da classe Vector que já tá pronta.** Eles têm essa palavrinha modificadora synchronized, que usa em métodos públicos. No método private void redimensioneSe (float taxaDeRedim) não precisa.

**Nos métodos adiante precisa de synchronized.**

public synchronized void add (X x)

{

if (this.qtd==this.elem.length)

this.redimensioneSe (2.0F);

this.elem[this.qtd] = x;

this.qtd++;

}

public synchronized X get (int posicao) // posicao vai de 0 a this.qtd-1

{

if (posicao<0 || posicao>this.qtd-1)

throw new java.lang.ArrayIndexOutOfBoundsException (posicao);

return this.elem[posicao];

}

public synchronized void remove (int posicao) throws ArrayIndexOutOfBoundsException

{

if (posicao<0 || posicao>this.qtd-1)

throw new java.lang.ArrayIndexOutOfBoundsException (posicao);

for (int i=posicao+1; i<this.qtd; i++)

this.elem[i-1] = this.elem[i];

this.qtd--;

this.elem[this.qtd] = null;

if (this.elem.length>Vector.TAMANHO\_INICIAL && this.qtd<=Math.round(this.elem.length\*0.25F)) this.redimensioneSe (0.5F);

}

public synchronized String toString ()

{

String ret="[";

for (int i=0; i<this.qtd-1; i++)

ret = ret+this.elem[i]+", ";

if (this.qtd>0)

ret = ret+this.elem[this.qtd-1];

return ret+"]";

}

public synchronized boolean equals (Object obj)

{

if (this==obj)

return true;

if (obj==null)

return false;

if (this.getClass()!=obj.getClass())

return false;

Vector<X> vec = (Vector<X>)obj;

for (int i=0; i<this.qtd; i++)

if (!this.elem[i].equals(vec.elem[i]))

return false;

return true;

}

**Então, os métodos públicos da classe Vector que já estão prontos tem essa palavrinha modificadora synchronized. Os construtores não levam essa palavrinha nunca, mas os demais métodos públicos sim.** E o que essa palavrinha faz, qual é o efeito dessa palavrinha? Basicamente o que acontece é o seguinte: tendo a palavrinha. como de fato agora nós temos, estamos indicando que não é para permitir que uma tarefa use nenhum método. Então, nenhuma tarefa pode usar o add, size, get, remove, toString e nem o equals.

**Logo,** **nenhuma tarefa pode usar um método synchronize, se outra tarefa começou a executar um método e parou no meio sem ter terminado o método. Então, se uma tarefa começou a executar esse método. executou até um certo pedaço e foi interrompida; não acabou de executar o método ainda.** Nenhuma outra tarefa poderá executar um método, nem o mesmo e nem outro. Se tem uma tarefa que começou a executar um método e não acabou, um método daquele objeto; no nosso caso aquele objeto compartilhado entre as várias tarefas. No nosso exemplo, era aquele objeto armazenamento, que era o objeto compartilhado pelas várias tarefas. Dessa forma, você vai ver que no exemplo tem tarefas, compartilhando uma estrutura de dados, do objeto da classe Vector.

**E eu afirmei para vocês que, ali nenhum problema iria acontecer; porque ali o objeto compartilhado da classe Vector que se chama armazenamento era um objeto da classe Vector já tem pronta em java. (da biblioteca java.io útil).** E nessa biblioteca, tem a palavra synchronized para métodos públicos.

**Por isso, lá naquele nosso exemplo. onde tinha as estruturas de dados compartilhados. Se eu, na tarefa do tipo um, eu começasse executar o add.** Executasse o add até um certo ponto e fosse interrompida a execução; outra tarefa, por exemplo tarefa do tipo 2 que entrasse em execução, se ela tentasse executar qualquer método: size, add, get, remove ou qualquer método.

**Se essa outra que entrou em execução agora, tentar executar qualquer método synchronize; ela não vai conseguir.** Porque tem aquela tarefa anterior que tinha começado a executar o add, ela executou até um certo e foi interrompida. Ou seja, nenhuma tarefa consegue executar um método synchronize; se tem uma outra tarefa que começou a executar um método synchronize e não terminou.

**Com isso, os problemas desaparecem. Então, a classe Vector que já está pronta, tem a palavra synchronize nos métodos públicos. E a nossa classe não tinha**. Então, a nossa classe iria causar problemas naquele programa onde tinham tarefas compartilhando uma estrutura de dados. Mas, a classe pronta que está na biblioteca Java.útil, ela não causaria problemas; porque ela tinha synchronize. E a nossa agora, que a gente colocou synchronized, não vai causar problema nenhum se fosse usado naquele programa para declarar o objeto compartilhado chamado armazenamento. Se fosse a nossa classe, agora a classe usada para declarar aquele armazenamento não teria nenhum problema. Porque, agora a gente colocou synchronize na nossa classe.

**Será que todas as estruturas de dados que existem prontas na linguagem Java, nas diversas bibliotecas que existem na linguagem Java. Será que todas elas têm synchronized nos métodos públicos?** Ou será que foi sorte a classe Vector tinha, mas não são todas que tem? Por sorte, usamos a classe Vector. Então por sorte deu tudo certo? Como é que será que funciona?

**Nós vamos ter que contar com a sorte? Ou será que a gente vai poder se basear em alguma coisa, para saber se a gente vai ter problemas ou não; quando a gente precisar compartilhar uma estrutura de dados entre diversas tarefas?** Vai depender da sorte? Vamos podemos basear em alguma coisa, ou como é que vai ser?

**Para isso, existe a documentação da classe java; aqui nessa janela de navegador eu digitei Java 13 api Vector e dei enter.** Entrou em: (https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/util/Vector. html).

**Ele entrou na página onde tem a documentação da classe Vector. E se eu ler a documentação eu vou acabar me deparando com esse pedacinho: Vector is synchronized.**

**Eu vou me deparar lendo a documentação com esses dizeres, com essa instrução :Vector is synchronized.** E, com isso eu vou ficar tranquilo, pois sei que vou poder compartilhar estruturas de dados do tipo Vector, com várias tarefas em execução e simultânea ou aparentemente simultânea; e não vou ter problemas.

**Porque essa documentação está me dizendo isso: Vector is synchronized.** Então, pode ficar tranquilo que você não vai enfrentar problemas; se usar o objeto da classe Vector para compartilhar: entre várias tarefas no programa multitarefas.

**Digamos que eu tivesse a intenção de usar um objeto da classe arraylist; que é uma outra classe que existe pronto em Java, inclusive na mesma biblioteca Java.util**. Se eu vier a documentação de arraylist, duplico a guia e digito Java api Java 13 arraylist, na qual abre o link: t(https://docs.oracle.com/en/java/javase/13/docs/api/java.base/java/util/Array List.html).

**Quando eu consultar a documentação do ArrayList, no certo ponto da minha leitura, eu vou encontrar esse Alerta: Note that this implementation is not synchronized. Ou seja, o arraylist não é synchronized, isto é essa implementação não é sinchronized.** Então, olhando a documentação sabemos que vai ter problema se pegar um objeto da classe Arraylist; e compartilhá-lo entre as várias tarefas que você vai colocar em execução simultânea ou aparentemente simultânea.

**O alerta deixa isso claro: se múltiplas tarefas acessam um Arraylist, acessam uma instância de Arraylist;** você vai ter que sincronizar externamente. Tem que ser sincronizado externamente.

**Então, ainda tem esperança para o Arraylist? Sim, ainda tem esperança. Eu até posso usar uma Arraylist para compartilhar entre várias tarefas.** Mas, eu não posso fazer isso tão tranquilamente quanto eu fiz com vector. Porque o vector, eu fui lá. declarei, instanciei e sai usando; não me preocupei com essa questão do sincronismo. Se eu fizesse o uso de uma Arraylist; eu teria que me preocupar com isso, teria que estar atento ao fato de que o Arraylist não é sincronizado.

**Se você quiser usar arraylists em múltiplas tarefas, você tem que sincronizar externamente. Agora, a gente quer saber como é que essa história de sincronizar externamente.** Sincronizar internamente é o que acontece no Vector; pois dentro do Vector já tem a palavrinha sinchronized e, portanto, o Vector já está preparado para funcionar, numa situação de ser compartilhado com várias tarefas de um programa multitarefas.

**Então, o Arraylist não está preparado, porque eles não têm essa palavrinha nos seus métodos.** O Vector tem e ele (vector) já está preparado para poder ser usado numa situação de compartilhamento antes várias tarefas no programa multitarefas.

**Aqui está dizendo que o arraylist não é sincronizado. Mas, ainda dá para usar um Arraylist? Sim, porém, como eles não é sincronizado em si mesmo;** como é o caso do vector que é sincronizado na implementação dele mesmo. Logo, para o arraylist, vou ter que fazer sincronismo externo. E isso é uma coisa que eu ainda vou mostrar para vocês daqui a pouco: nós vamos pegar aquele exemplo onde a gente usou Vector, e nós vamos parar de usar Vector e para conseguir usar o Arraylist.

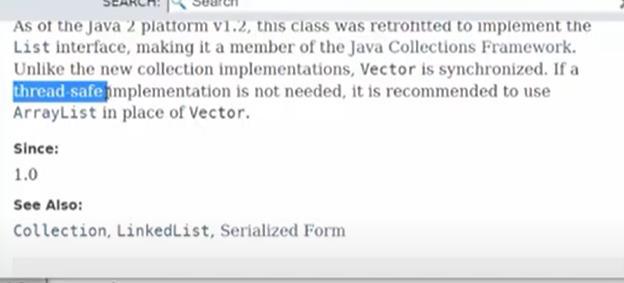
**Eu vou mostrar para vocês, o que eu vou ter que fazer para usar o Arraylist; já que o Arraylist não é como um Vector, pois o Vector é sincronizado(is synchronized), passo para realese is not synchronized.** Então, a gente precisa a aprender agora, o que fazer quando eventualmente a gente desejar ter uma estrutura de dados compartilhadas entre várias tarefas no programa multitarefas. Mas, a mistura de dados não é sincronizada. A minha estrutura de dados vai precisar de sincronismo externo; vamos aprender fazer isso e saber que é sincronismo externo e como se faz.

**Eu deixei a documentação bem no lugarzinho aqui, para mostrar o que eu queria mostrar. Mas só pra você saber, isso realmente é a documentação do vector.** E vai ver que a documentação da classe Vector implementa um vetor de objetos, que cresce mais não é fixo, ela pode aumentar. Como um vetor que contém componentes que vão ser acessados usando indices, usando posições inteiras.

**Ou seja, estamos mesmo na classe Vector, e já vimos que Vector is sincronized.**

**E na classe arraylist, já vimos aquele pedacinho onde eu tinha deixado navegador inicialmente; antes de começar o vídeo;** Nós vemos que o arraylist is not sinchronized, ou seja, você pretende usar o arraylist em compartilhamento de tarefas no programa multi tarefa. Você vai ter que tomar suas providências, fazendo o sincronismo externo.

 Eu quero mostrar para vocês que as vezes a gente encontra isso:



**Essa forma de se referir a ser sincronizado ou não ser sintonizado. As vezes a gente está falando thread-safe, ou seja, seguro para threads / tarefas.** Aqui, está falando o que ser uma implementação segura para tarefas, se não for necessário você pode usar Arraylist em vez de vector.

**Então, as vezes a gente fala que Vector é sincronizado; as vezes diz que Vector é thread safe.** As vezes diz que arraylist não é sincronizado, é sinônimo de não ser thread safe, e isso faz com que pode ter problemas, exceto se fizer o sincronismo externo.

**Agora pode pensar em como fazer o sincronismo externo. Pesquisou como Java 13 api vector e java 13 api arraylist no google para achar a documentação.** Isto é, estou perguntando sobre as bibliotecas de java na versão 13, mais atualizada. (45min18s). Assim sendo, eu entro nas documentações das classes vector e arraylist respectivamente. E esse mecanismo de pesquisa vale para achar qualquer classe que tenha interesse.

**Agora, na segunda parte da aula; vou explicar como faria para pegar um daqueles programas que fez no começo, que compartilhavam vector.** Se quisesse modificar para, ao invés de compartilhar vector, compartilhar arraylist. Sabendo que arraylist, não são thread safe nem sincronizados, o que vou ter que fazer caso eu deseje compartilhar um arraylist?

**No exemplo a seguir, é o exemplo no qual foi explicado semáforos (TarefaDoTipo4.java da aula de compartilhar ed com limites), mas agora foi mudado usando o import.java.util.Arraylist no lugar da biblioteca de vector.** O restante do exemplo é o mesmo. (46min59s); os métodos que usam de arraylist são os mesmos de vector, ou seja, ambos contêm add, size, get, remove. Porque o arraylist eles não tem os mesmos métodos que vector, porém os métodos supracitados têm em ambas as classes.

**Logo, tive que fazer poucas modificações de vector para arraylist. Eu alterei de Vector para ArrayList ( naquele programa onde tem a main com semaphore**).

Logo, eu faço:

Import.java.util.ArrayList;

Import.java.util.concurrent.Semaphore;

public class Programa

{

public static void main (String[] args)

{

try

{

ArrayList<Character> armazenamento = new ArrayList<Character> ();

Semaphore livre = new Semaphore (1024,true);

Semaphore ocupado = new Semaphore (0,true);

System.out.println ("Tecle ENTER para ativar as tarefas e");

System.out.println ("Tecle novamente ENTER para terminar o programa.");

Teclado.getUmString();

TarefaDoTipo1 t1 = new TarefaDoTipo1 (armazenamento, livre, ocupado);

t1.start ();

TarefaDoTipo2 t2 = new TarefaDoTipo2 (armazenamento, livre, ocupado);

t2.start ();

TarefaDoTipo3 t3 = new TarefaDoTipo3 (armazenamento, livre, ocupado);

t3.start ();

TarefaDoTipo4 t4 = new TarefaDoTipo4 (armazenamento, livre, ocupado);

t4.start ();

Teclado.getUmString();

t4.morra ();

t3.morra ();

t2.morra ();

t1.morra ();

/\*

t1.join();

t2.join();

t3.join();

t4.join();

\*/

System.out.println ("Execucao do programa finalizada.");

}

catch (Exception erro)

{} // sei que não passei null para o construtor de nenhuma das tarefas

}

}

**Desta forma, a gente declara e instancia na main o objeto armazenamento sendo um ArrayList de Caracteres.** Na hora de passar as tarefas que serão iniciadas pelo armazenamento, eu passo o armazenamento de arraylist. (Para tarefas dos tipos 1,2,3,4).

**Por consequência, nas tarefas dos tipos 1,2,3,4 no construtor (exemplo: TarefaDoTipo1.java):**

import java.util.ArrayList;

import java.util.concurrent.Semaphore;

public class TarefaDoTipo1 extends Thread

{

ArrayList <Character> armazenamento;

Semaphore livre;

Semaphore ocupado;

public TarefaDoTipo1 (ArrayList<Character> armz, Semaphore lvr, Semaphore ocp) throws Exception

{

if (armz==null)

throw new Exception ("Armazenamento ausente");

if (lvr==null)

throw new Exception ("Livre ausente");

if (ocp==null)

throw new Exception ("Ocupado ausente");

this.armazenamento = armz;

this.livre = lvr;

this.ocupado = ocp;

}

**O construtor de cada tarefa só muda o número do tipo; pois todas as tarefas importam arraylist e também tem como primeiro método o construtor.** Como anteriormente elas usavam vector, tem que mudar na lógica no método run:

Então, pegando como exemplo o arquivo: TarefaDoTipo1.java:

public void run () // vale para as tarefas dos tipos 1,2 e 3

{

char caractere='a';

while (!this.fim)

{

this.livre.acquireUninterruptibly();

synchronized (this.armazenamento)

{

this.armazenamento.add (caractere);

}

this.ocupado.release();

try { this.sleep (150); } catch (Exception erro) {}

if (caractere=='z')

caractere = 'a';

else

caractere = (char)(((int)caractere)+1);

}

}

**No meu método run não mudou a tarefa; pois usa o mesmo add quando era vector. E isso vale para todas as tarefas.** O atributo não é vector e sim arraylist; eu importo arraylist da biblioteca java.util e o construtor recebe um arraylist de armazenamento. O restante da lógica continua a mesma coisa. Agora sabemos que estamos usando uma classe que não é thread-safe em ambiente de multitarefas; mas é possível se tiver algumas cautelas.

**E quais são os cuidados? São bem simples.** Tenho que escrever, na hora de escrever qualquer método. Por exemplo, na TarefaDoTipo4.java, sempre que usar qualquer método daquela estrutura de dados compartilhadas que não é thread safe (como o armazenamento do tipo ArrayList), tenho que chamar o método dentro do comando synchronized (nome do objeto compartilhado que não é do thread-safe) {nome do objetochamante.metodo ();}

**Essa forma de usar o comando synchronized vai acontecer para todas as tarefas. Perceba que, quando tinha a classe Vector, o synchronized ia no cabeçalho público dentro de cada método, na qualificação do nosso método;** como usamos arraylist, o synchronized vem como linha de comando dentro de um método maior que eu esteja usando (como se fosse um try).

Para a tarefa do tipo 4, no método run:

public void run ()

{

while (!this.fim)

{

this.ocupado.acquireUninterruptibly();

char caractere;

synchronized (this.armazenamento)

{

caractere = this.armazenamento.get(0);

this.armazenamento.remove(0);

}

this.livre.release();

System.out.println (caractere);

try { this.tarefa.sleep (200); } catch (Exception erro) {}

}

while (this.armazenamento.size()!=0)

{

this.ocupado.acquireUninterruptibly();

char caractere;

synchronized (this.armazenamento)

{

caractere = this.armazenamento.get(0);

this.armazenamento.remove(0);

}

this.livre.release();

System.out.println (caractere);

try { this.tarefa.sleep (600); } catch (Exception erro) {}

}

(Parei em 55min30s)

**Então, sempre que quiser usar o objeto armazenamento: faz synchronized (this.armazenamento) e dentro do abre-fecha colchete coloca todos os métodos que tem o armazenamento como objeto chamante, a vontade.** Então, é assim que se faz o sincronismo externo.

**Qual o sentido de usar synchronized? Eu estou na TarefaDoTIpo4; quando escreve synchronized (this.armazenamento) sendo armazenamento o nome do objeto que está sendo compartilhado com as várias tarefas.** O sentido é que a TarefaDoTipo4, nesse momento, manifesta a intenção de usar o objeto armazenamento. Mas ela quer usar esse objeto com exclusividade, quer ser a única tarefa a usar esse objeto. Então, se já tinha uma tarefa usando que não acabou de usar ainda, a TarefaDoTIpo4 para e espera porque essa última tarefa (tipo 4) quer usar o objeto sozinha, sem nenhuma outra tarefa usar ao mesmo tempo.

**Agora (para a TarefaDoTipo4), senão tiver nenhuma outra tarefa usando, ela manifesta o desejo de usar sozinha, consegue usar; entra no abre-fecha chave e consegue usar. Vejamos a TarefaDoTipo3, executar synchronized; como a TarefaDoTIpo4 está usando o armazenamento.** Logo, a TarefaDoTIpo3 manifesta o desejo de usar o armazenamento sozinha, sem nenhuma outra tarefa usando junto. A TarefaDoTipo3 não consegue usar sozinha, porque a TarefaDoTIpo4 já está usando; então a TarefaDoTIpo3 será interrompida, para de executar e fica esperando a TarefaDoTipo4 terminar. A TarefaDoTipo3 só volta quando for possível executar o armazenamento sozinha.

**Se a TarefaDoTipo2 quiser usar o armazenamento, ela espera a TarefaDoTipo3 terminar de usar o objeto armazenamento.** Se a TarefaDoTipo1 quiser usar, vai esperar a TarefaDoTipo2 terminar.

**Os métodos dentro do comando synchronized teve ter as chamadas do método compartilhado, que eu quero acessar com exclusividade;** deve ser o mais breve possível. Mesmo que tiver que colocar mais de um synchronized.

**Imagina que, dentro da TarefaDoTIpo4, no método run. Estou dentro do synchronized e entre as linhas do get e remove tenha um monte de contas que faço e guardo em uma variável que não tem relação nenhuma com o objeto armazenamento.** Eu faria um primeiro synchronized com get; passaria fora do synchronized a variável das contas. Depois faria um segundo synchronized com o remove. O synchronized limita a quantidade de tarefas a serem usadas; ficam bloqueadas.

**Uma boa prática de programação é manter o objeto armazenamento apenas com as chamadas do método que realmente se relacionam com o objeto dentro de um synchonized para a tarefa ter exclusividade.** O restante (que não precisa de exclusividade) fica para fora, sem bloqueio.

**10.2) Pasta: Meu Vector Sincronizado**

**10.2.1.Arquivo: Programa.java**

public class Programa

{

public static void main (String[] args)

{

java.util.Vector<String> v1 =

new java.util.Vector<String> ();

v1.add("C");

v1.add("C++");

v1.add("Java");

System.out.println (v1);

v1.remove(1);

System.out.println (v1);

System.out.println ();

Vector<String> v2 =

new Vector<String> ();

v2.add("C");

v2.add("C++");

v2.add("Java");

System.out.println (v2);

v2.remove(1);

System.out.println (v2);

}

}

++++++++++++++++++++++++++++++++++++

**10.2.2.Arquivo: Vector.java**

public class Vector <X>

{

private static final int TAMANHO\_INICIAL = 10;

private X[] elem;

private int qtd;

public Vector ()

{

//this.elem = new X [Vector.TAMANHO\_INICIAL]; <- não compila

this.elem = (X[])new Object [Vector.TAMANHO\_INICIAL];

this.qtd = 0;

}

public synchronized int size ()

{

return this.qtd;

}

private void redimensioneSe (float taxaDeRedim)

{

X[] novo;

//novo = new X [Math.round(this.elem.length\*taxaDeRedim)]; <- não compila

novo = (X[])new Object [Math.round(this.elem.length\*taxaDeRedim)];

for (int i=0; i<this.qtd; i++)

novo[i] = this.elem[i];

this.elem = novo;

}

public synchronized void add (X x)

{

if (this.qtd==this.elem.length)

this.redimensioneSe (2.0F);

this.elem[this.qtd] = x;

this.qtd++;

}

public synchronized X get (int posicao) // posicao vai de 0 a this.qtd-1

{

if (posicao<0 || posicao>this.qtd-1)

throw new java.lang.ArrayIndexOutOfBoundsException (posicao);

return this.elem[posicao];

}

public synchronized void remove (int posicao) throws ArrayIndexOutOfBoundsException

{

if (posicao<0 || posicao>this.qtd-1)

throw new java.lang.ArrayIndexOutOfBoundsException (posicao);

for (int i=posicao+1; i<this.qtd; i++)

this.elem[i-1] = this.elem[i];

this.qtd--;

this.elem[this.qtd] = null;

if (this.elem.length>Vector.TAMANHO\_INICIAL &&

this.qtd<=Math.round(this.elem.length\*0.25F))

this.redimensioneSe (0.5F);

}

public synchronized String toString ()

{

String ret="[";

for (int i=0; i<this.qtd-1; i++)

ret = ret+this.elem[i]+", ";

if (this.qtd>0)

ret = ret+this.elem[this.qtd-1];

return ret+"]";

}

public synchronized boolean equals (Object obj)

{

if (this==obj)

return true;

if (obj==null)

return false;

if (this.getClass()!=obj.getClass())

return false;

Vector<X> vec = (Vector<X>)obj;

for (int i=0; i<this.qtd; i++)

if (!this.elem[i].equals(vec.elem[i]))

return false;

return true;

}

}

++++++++++++++++++++++++++++++++++++++++++++

**10.3) Pasta: Threads com compartilhamento de ED (sincronismo externo)**

**10.3.1.Arquivo: Programa.java**

import java.util.ArrayList;

import java.util.concurrent.Semaphore;

public class Programa

{

public static void main (String[] args)

{

try

{

ArrayList<Character> armazenamento = new ArrayList<Character> ();

Semaphore livre = new Semaphore (4,true);

Semaphore ocupado = new Semaphore (0,true);

System.out.println ("Tecle ENTER para ativar as tarefas e");

System.out.println ("Tecle novamente ENTER para terminar o programa.");

Teclado.getUmString();

TarefaDoTipo1 t1 = new TarefaDoTipo1 (armazenamento, livre, ocupado);

t1.start ();

TarefaDoTipo2 t2 = new TarefaDoTipo2 (armazenamento, livre, ocupado);

t2.start ();

TarefaDoTipo3 t3 = new TarefaDoTipo3 (armazenamento, livre, ocupado);

t3.start ();

TarefaDoTipo4 t4 = new TarefaDoTipo4 (armazenamento, livre, ocupado);

t4.start ();

Teclado.getUmString();

t4.morra ();

t3.morra ();

t2.morra ();

t1.morra ();

t1.join();

t2.join();

t3.join();

t4.join();

System.out.println ("Execucao do programa finalizada.");

}

catch (Exception erro)

{} // sei que não passei null para o construtor de nenhuma das tarefas

}

}

+++++++++++++++++++++++++++++++++++++++++++++

**10.3.2.Arquivo: TarefaDoTipo1.java**

import java.util.ArrayList;

import java.util.concurrent.Semaphore;

public class TarefaDoTipo1 extends Thread

{

ArrayList<Character> armazenamento;

Semaphore livre;

Semaphore ocupado;

public TarefaDoTipo1 (ArrayList<Character> armz, Semaphore lvr, Semaphore ocp) throws Exception

{

if (armz==null)

throw new Exception ("Armazenamento ausente");

if (lvr==null)

throw new Exception ("Livre ausente");

if (ocp==null)

throw new Exception ("Ocupado ausente");

this.armazenamento = armz;

this.livre = lvr;

this.ocupado = ocp;

}

private boolean fim = false;

public void morra ()

{

this.fim=true;

}

public void run ()

{

char caractere='a';

while (!this.fim)

{

this.livre.acquireUninterruptibly();

synchronized (this.armazenamento)

{

this.armazenamento.add (caractere);

}

this.ocupado.release();

try { this.sleep (150); } catch (Exception erro) {}

if (caractere=='z')

caractere = 'a';

else

caractere = (char)(((int)caractere)+1);

}

}

}

++++++++++++++++++++++++++++++++++++++++++++++++++

**10.3.3.Arquivo: TarefaDoTipo2.java**

import java.util.ArrayList;

import java.util.concurrent.Semaphore;

public class TarefaDoTipo2 extends Thread

{

ArrayList<Character> armazenamento;

Semaphore livre;

Semaphore ocupado;

public TarefaDoTipo2 (ArrayList<Character> armz, Semaphore lvr, Semaphore ocp) throws Exception

{

if (armz==null)

throw new Exception ("Armazenamento ausente");

if (lvr==null)

throw new Exception ("Livre ausente");

if (ocp==null)

throw new Exception ("Ocupado ausente");

this.armazenamento = armz;

this.livre = lvr;

this.ocupado = ocp;

}

private boolean fim = false;

public void morra ()

{

this.fim=true;

}

public void run ()

{

char caractere='A';

while (!this.fim)

{

this.livre.acquireUninterruptibly();

synchronized (this.armazenamento)

{

this.armazenamento.add (caractere);

}

this.ocupado.release();

try { this.sleep (300); } catch (Exception erro) {}

if (caractere=='Z')

caractere = 'A';

else

caractere = (char)(((int)caractere)+1);

}

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++

**10.3.4.Arquivo: TarefaDoTipo3.java**

import java.util.ArrayList;

import java.util.concurrent.Semaphore;

public class TarefaDoTipo3 implements Runnable

{

ArrayList<Character> armazenamento;

Semaphore livre;

Semaphore ocupado;

public TarefaDoTipo3 (ArrayList<Character> armz, Semaphore lvr, Semaphore ocp) throws Exception

{

if (armz==null)

throw new Exception ("Armazenamento ausente");

if (lvr==null)

throw new Exception ("Livre ausente");

if (ocp==null)

throw new Exception ("Ocupado ausente");

this.armazenamento = armz;

this.livre = lvr;

this.ocupado = ocp;

}

private Thread tarefa = new Thread (this);

public void start ()

{

this.tarefa.start();

}

public void join () throws InterruptedException

{

this.tarefa.join();

}

private boolean fim = false;

public void morra ()

{

this.fim=true;

}

public void run ()

{

char caractere='0';

while (!this.fim)

{

this.livre.acquireUninterruptibly();

synchronized (this.armazenamento)

{

this.armazenamento.add (caractere);

}

this.ocupado.release();

try { this.tarefa.sleep (450); } catch (Exception erro) {}

if (caractere=='9')

caractere = '0';

else

caractere = (char)(((int)caractere)+1);

}

}

}

+++++++++++++++++++++++++++++++++++++++++++++++++++++

**10.3.5.Arquivo: TarefaDoTipo4.java**

import java.util.ArrayList;

import java.util.concurrent.Semaphore;

public class TarefaDoTipo4 implements Runnable

{

ArrayList<Character> armazenamento;

Semaphore livre;

Semaphore ocupado;

public TarefaDoTipo4 (ArrayList<Character> armz, Semaphore lvr, Semaphore ocp) throws Exception

{

if (armz==null)

throw new Exception ("Armazenamento ausente");

if (lvr==null)

throw new Exception ("Livre ausente");

if (ocp==null)

throw new Exception ("Ocupado ausente");

this.armazenamento = armz;

this.livre = lvr;

this.ocupado = ocp;

}

private Thread tarefa = new Thread (this);

public void start ()

{

this.tarefa.start();

}

public void join () throws InterruptedException

{

this.tarefa.join();

}

private boolean fim = false;

public void morra ()

{

this.fim=true;

}

public void run ()

{

while (!this.fim)

{

this.ocupado.acquireUninterruptibly();

char caractere;

synchronized (this.armazenamento)

{

caractere = this.armazenamento.get(0);

this.armazenamento.remove(0);

}

this.livre.release();

System.out.println (caractere);

try { this.tarefa.sleep (100); } catch (Exception erro) {}

}

while (this.armazenamento.size()!=0)

{

this.ocupado.acquireUninterruptibly();

char caractere = this.armazenamento.get(0);

this.armazenamento.remove(0);

this.livre.release();

System.out.println (caractere);

try { this.tarefa.sleep (600); } catch (Exception erro) {}

}

}

}

++++++++++++++++++++++++++++++++++++++++++++

**10.3.6.Arquivo: Teclado.java**

//import java.io.BufferedReader;

//import java.io.InputStreamReader;

//import java.io.IOException;

import java.io.\*;

public class Teclado

{

private static BufferedReader teclado =

new BufferedReader (

new InputStreamReader (

System.in));

public static String getUmString ()

{

String ret=null;

try

{

ret = teclado.readLine ();

}

catch (IOException erro)

{} // sei que nao vai dar erro

return ret;

}

public static byte getUmByte () throws Exception

{

byte ret=(byte)0;

try

{

ret = Byte.parseByte (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Byte invalido!");

}

return ret;

}

public static short getUmShort () throws Exception

{

short ret=(short)0;

try

{

ret = Short.parseShort (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Short invalido!");

}

return ret;

}

public static int getUmInt () throws Exception

{

int ret=0;

try

{

ret = Integer.parseInt (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Int invalido!");

}

return ret;

}

public static long getUmLong () throws Exception

{

//long ret=(long)0;

//long ret=0;

long ret=0L;

try

{

ret = Long.parseLong (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Long invalido!");

}

return ret;

}

public static float getUmFloat () throws Exception

{

//float ret=0;

//float ret=(float)0.0;

float ret=0.0F;

try

{

ret = Float.parseFloat (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Float invalido!");

}

return ret;

}

public static double getUmDouble () throws Exception

{

//double ret=0;

//double ret=(long)0;

//double ret=0L;

double ret=0.0;

try

{

ret = Double.parseDouble (teclado.readLine ());

}

catch (IOException erro)

{} // sei que nao vai dar erro

catch (NumberFormatException erro)

{

throw new Exception ("Double invalido!");

}

return ret;

}

public static char getUmChar () throws Exception

{

char ret=' ';

try

{

String str = teclado.readLine ();

if (str==null)

throw new Exception ("Char invalido!");

if (str.length() != 1)

throw new Exception ("Char invalido!");

ret = str.charAt(0);

}

catch (IOException erro)

{} // sei que nao vai dar erro

return ret;

}

public static boolean getUmBoolean () throws Exception

{

boolean ret=false;

try

{

String str = teclado.readLine ();

if (str==null)

throw new Exception ("Boolean invalido!");

if (!str.equals("true") && !str.equals("false"))

throw new Exception ("Boolean invalido!");

ret = Boolean.parseBoolean (str);

}

catch (IOException erro)

{} // sei que nao vai dar erro

return ret;

}

}

++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++