Visão Geral do Cenário:

É a rotina diária de duas crianças, Charlinhos e Joãozinho, com horários escolares. A simulação opera em um ciclo de 24 horas, onde os agentes transitam entre estados como "Dormindo", "Estudando", "Comendo" e "Caminhando". Uma diferença de seus comportamentos é o meio de transporte: Charlinhos se desloca a pé, o que aumenta seu tempo no estado "CaminhadaCharlinho" e atrasa o início de suas atividades noturnas. Em contrapartida, Joãozinho utiliza condução, retornando para casa mais cedo e iniciando sua rotina de descanso antes.

Visão Geral de Cada Agente:

Cada **personagem** age diferente dependendo das horas no dia:
Cada **personagem** tem fome e cansaço.

Charlinho - O Sofrido

- **1 h - 8 h:** Dorme mesmo com cansaço no 0.
- **8 h-12 h:** Acorda se o cansaço chegar no 0.
    - Primeira Pergunta, Tá com fome? (fome = 20)
    - Nesse caso, vão comer.
    - Se não estiver, Brincam.
- **12h - 15h:** Caminhada até a Escola
- **12h - 18h:** Estudando
- **18h - 21h:** Caminhando pra casa
- **21h - 24h:** Comer ou Brincar dependendo da fome e cansaço
    - Esse também é o período em que Charlinho dorme, dependendo do cansaço. (cansaço = 100)

Joãozin - Vizinho do Carlinho

- **1h - 8h:** Dorme mesmo com cansaço no 0.
- **8h -13h:** Acorda se o cansaço chegar no 0.
    - Primeira Pergunta, Tá com fome? (fome = 20)
    - Nesse caso, vão comer.
    - Se não estiver, Brincam.
- **13h - 14h:** Condução até a Escola
- **14h - 18h:** Estudando
- **18h - 19h:** Já chega em casa
- **19h - 24h:** Comer ou Brincar dependendo da fome e cansaço
    - Esse também é o período em que Joãozinho dorme, dependendo do cansaço. (cansaço = 100)

Tabela de Regras:

	Dormindo	**Comendo**	**Brincando**	**Estudando**	**Caminhando**
Charlinho	**21h-24h:** Só se cansaço = 100 **1h-8h:** Obrigatório	**8h -12h:** Se fome > 20 21-24: Se fome >15	**21h - 24h =** Se fome < 20 e cansaço < 100	Só Horario: **18h -21h**	Só Horario: **12h - 15h** **18h - 21h**
Joãozinho	**19h - 24h:** Só se cansaço = 100 **1h - 8h:** Obrigatório	**8h -12h:** Se fome > 20 **18h - 24h:** Se fome >15	**18h - 24h =** Se fome < 20 e cansaço < 100	Só Horario: **13h -21h**	Só Horario: **13h - 14h** **18h - 19h**

Descrição das Variáveis:

Variável	**Tipo**	**Valor Inicial**	**Uso para Troca de Estado**
hunger	`int`	`0`	Usada como condição para transições de estado. Níveis de fome controlam se o personagem entra no estado **Eating** ou continua outras atividades **(Brincando, CaminhadaCharlinho, Estudando, etc.)**. Por exemplo, em certos horários, se getHunger() > 15 ou getHunger() > 20, o estado muda para **Eating.**
fatigue	`int`	`0`	Usada como condição para transições de estado. Níveis de cansaço controlam se o personagem entra no estado Sleeping (Dormindo) ou se tem energia para atividades. Por exemplo, se getFatigue() > 100, o estado muda para Sleeping. Em outros momentos, se getFatigue() < 100, atividades como Brincando são permitidas.

Horario	`int`	`8`	Controla os ciclos de tempo e determina as janelas de regras que os personagens (Charlinho e Joaozin) devem seguir. Os estados mudam dependendo se o Horario está entre limites específicos (e.g., Manhã: > 7 e < 13/14; Tarde: > 12/13 e < 19; Noite/Madrugada: > 21 e < 24 ou > 0 e < 8).
state	`State (Interface)`	`new Sleeping(this)`	Armazena e representa o estado comportamental atual do personagem. A função update() executa o estado atual (state.execute()) e, com base nas regras (que verificam hunger, fatigue e Horario), pode chamar setState(newState) para mudar a referência do estado.
Na_Escola (Apenas Joaozin)	`boolean`	`false`	Usada especificamente pelo personagem Joaozin para verificar sua localização ou status escolar. Esta variável é usada para diferenciar o estado, mesmo se as condições de hunger forem atendidas. Por exemplo, na Tarde, se getHunger() <= 5, a transição de estado depende de Na_Escola: se false, ele está Andando; se true, ele está Estudando. Também influencia as ações noturnas.
PreviewHunger	`int`	`0` **(Implícito)**	Armazena o valor anterior de Fome. É usada no printStats() para imprimir as mudanças de um estado para outro no log.
PreviewFatigue	`int`	`0` **(Implícito)**	Armazena o valor anterior de Cansaço. É usada no printStats() para imprimir as mudanças de um estado para outro no log.

Estrutura do Código:

Main.java: Esta é a classe principal, responsável por iniciar a execução do programa. Ela contém o método **main**, onde os agentes são criados e o loop da simulação é executado para atualizar seus estados.
Juca.java: Funciona como uma classe base abstrata (ou superclasse) para todos os personagens, servindo como molde para Carlinhos e Joãozinho. Esta classe tem os

atributos fisiológicos, como os níveis de fome e fadiga, e fornece os métodos de acesso (getters e setters) para manipulá-los. Além disso, ela define a estrutura essencial para a máquina de estados, incluindo a referência ao estado atual do personagem e o método para executar o comportamento correspondente.

Carlinhos.java e Joaozinho.java: Estas são as classes filhas que herdam diretamente da classe base Juca. Elas reutilizam toda a estrutura comum fornecida pela superclasse, incluindo os atributos fisiológicos (fome, fadiga) e a mecânica de gerenciamento de estados. A principal responsabilidade de cada uma é sobrescrever o método update(), implementando a transição e a lógica de decisão que são diferentes para cada personagem, de acordo com a rotina de cada um.

State.java: É a interface que serve como uma regra/padrão para todos os estados possíveis na simulação. Qualquer classe que represente um comportamento específico como **Sleeping.java**, **Eating.java** ou **Estudando.java** deve implementar esta interface. Isso garante que todos os estados tenham as mesmas classes de base (por exemplo, o método execute()).

AbstractState.java: Garante a estrutura e fornece métodos comuns que todos os estados precisam, mas sem implementar a execução (execute()) que é exclusiva de cada estado. Também sempre mantém uma referência ao (private Juca juca) não importa qual personagem esteja sendo usado (Juca, Charlinho ou Joãozin).

Horario.java: O Relógio Virtual. Possui um get/set para as outras classes possam utilizá-lo.

Brincando. java, Estudando.java,Comendo.java, Dormindo.java: As classes de estados da máquina. Cada uma herda da classe AbstractState e é responsável por implementar a lógica e o comportamento específicos de uma única atividade. Dentro de cada classe, o método execute() é sobrescrito para realizar as ações daquele estado e verificar as condições que podem levar a uma transição para um novo estado.

Resultados Esperados:

O Log mostra o que os meninos estão fazendo de hora em hora. No primeiro caso às 23hrs, ambos estavam descansados e foram brincar. No segundo, às 23hrs, Joãozinho foi comer porque ficou com fome.

```
--------------------
São: 22:00
------Charlinho------
Limpando o Caminhão...
Fome: 8 -->9
Cansaço: 38 -->42
Bora brincar...
-------Joaozin-------
Brincando...
Fome: 8 -->11
Cansaço: 41 -->48
Bora brincar...
--------------------
São: 23:00
------Charlinho------
Brincando...
Fome: 9 -->12
Cansaço: 42 -->49
Bora brincar...
-------Joaozin-------
Brincando...
Fome: 11 -->14
Cansaço: 48 -->55
Bora brincar...
--------------------
São: 0:00
------Charlinho------
Brincando...
Fome: 12 -->15
Cansaço: 49 -->56
Bateu um sono...
-------Joaozin-------
Brincando...
Fome: 14 -->17
Cansaço: 55 -->62
Bateu um sono...
--------------------
São: 1:00
------Charlinho------
Dormindo...
Fome: 15 -->16
Cansaço: 56 -->46
Bateu um sono...
-------Joaozin-------
```

```
--------------------
São: 23:00
------Charlinho------
Brincando...
Fome: 7 -->10
Cansaço: 59 -->66
Bora brincar...
-------Joaozin-------
Brincando...
Fome: 15 -->18
Cansaço: 69 -->76
Bateu uma fome...
--------------------
```

Limitações (Opcional):

O sistema de logs poderia ser mais dinâmico. Nós queríamos ter conseguido fazer com que o java retornasse o nome da variável como string, assim não precisaria de printout próprio para cada nome de personagem. E gostaríamos de ter colocado uma interação entre os dois personagens.