

# TRABALHO PRÁTICO 4

---

# Objetivo do Trabalho

- O objetivo do trabalho é implementar a estratégia de controle de concorrência conhecida como Timestamp Ordering
- A classe que implementa o estratégia se chama **XXXTimestampOrderingConcurrentyManager**
- O aluno deve
  - Trocar XXX pelo seu nome
  - Implementar as funções que estão faltando

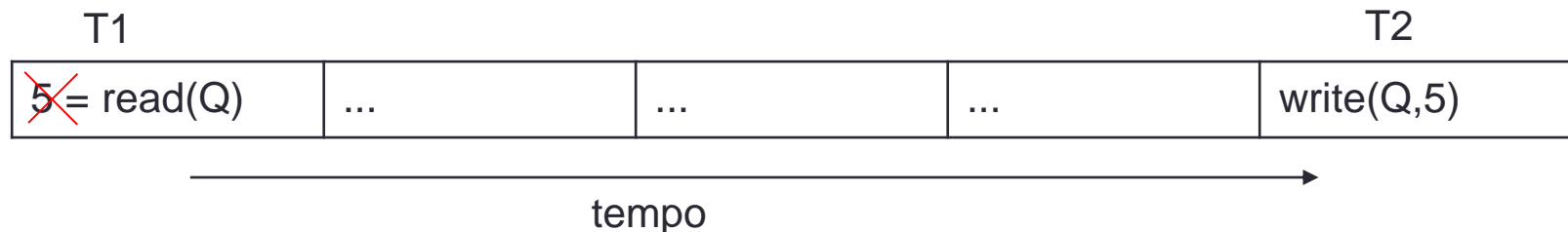
# Timestamp Ordering

- Cada transação recebe um timestamp TS, que indica a ordem de criação
  - Se a transação T1 iniciou antes de T2, então  $TS(T1) < TS(T2)$
- Durante a execução, uma transação só pode executar um read ou um write se as regras do protocolo forem satisfeitas
  - Caso contrário, a transação deve abortar

# Timestamp Ordering

- Quando T1 efetuar um read(Q)
  - Se  $ts(T1) < ts-w(Q)$ 
    - T1 não deveria poder ler um valor que foi gerado por uma transação mais velha
      - Seria como ler um valor que ainda não existe
    - Por isso, T1 aborta
  - Caso contrário
    - T1 executa
    - Se  $ts(T1) > ts-r(Q)$ 
      - $ts-r(Q)$  é atualizado

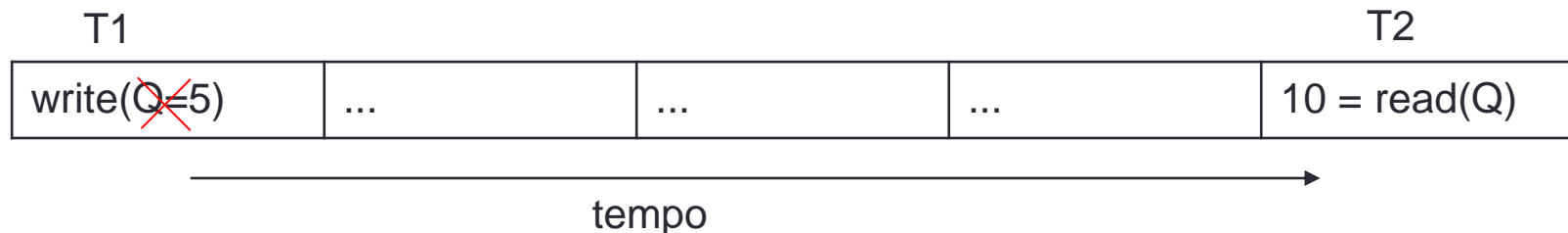
Ex. se t1 for mais velha, e  $Q = 100$ , t1 não poderia ler o valor 5 que, em um schedule serial, nem teria sido gerado ainda



# Timestamp Ordering

- Quando T1 efetuar um write(Q)
  - Se  $ts(T1) < ts-r(Q)$ 
    - Uma transação mais nova já leu um valor de Q. Então, o valor que a transação mais velha está propondo não pode ser aceito.
      - Seria como voltar no tempo para modificar algo
    - Por isso, T1 aborta
  - Caso contrário
    - Se  $ts(T1) > ts-w(Q)$ 
      - T1 executa
      - $ts-w(Q)$  é atualizado

Ex. se t1 for mais nova, e já conheceu uma realidade em que  $Q = 10$ , então a transação mais velha não pode alterar esse valor



[illegible]

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);
---	-----------------------------	------------------------------

## Controle

item	ts-r	ts-w
A	-1	-1
B	-1	-1
C	-1	-1

Timestamp: 0

transação	ts
T1	
T2	
T3	

[illegible]

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);
---	-----------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	-1	-1
B	-1	-1
C	-1	-1

transação	ts
T1	1
T2	
T3	

Timestamp: 1

## Definido o timestamp de T1

[illegible]

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);
---	-----------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	-1	-1
B	-1	-1
C	-1	-1

transação	ts
T1	1
T2	
T3	

Timestamp: 1

$$\text{ts-r}(B) < \text{ts}(T1).$$

OK!



T1	T2	T3	T4
W(B)			

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);
---	-----------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	-1	-1
B	-1	1
C	-1	-1

transação	ts
T1	1
T2	
T3	

Timestamp: 1

Realiza a escrita e atualiza ts-w

[illegible]

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);
---	-----------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	-1	-1
B	-1	1
C	-1	-1

transação	ts
T1	1
T2	2
T3	

Timestamp: 2

## Definido o timestamp de T2

T1	T2	T3	T4
W(B)			

## Transações

T1:	T2:	T3:
write(B);	read(B);	read(C);
write(C);	read(A);	write(A);
read(A);		



## Controle

item	ts-r	ts-w
A	-1	-1
B	-1	1
C	-1	-1

transação	ts
T1	1
T2	2
T3	

Timestamp: 2

ts-w(B) < ts(T2).

OK!



[illegible]

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);
---	-----------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	-1	-1
B	2	1
C	-1	-1

transação	ts
T1	1
T2	2
T3	3

Timestamp: 3

## Definido o timestamp de T3

T1	T2	T3	T4
W(B)			
	R(B)		

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);
---	-----------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	-1	-1
B	2	1
C	-1	-1

transação	ts
T1	1
T2	2
T3	3

Timestamp: 3

ts-w(C) < ts(T3).

OK!

T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);
---	-----------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	-1	-1
B	2	1
C	3	-1

transação	ts
T1	1
T2	2
T3	3

Timestamp: 3

Realiza a leitura e atualiza ts-r

T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	

# Transações

T1: write(B); <b>write(C);</b> read(A);	T2: read(B); read(A);	T3: read(C); write(A);
--	-----------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	-1	-1
B	2	1
C	<b>3</b>	-1

transação	ts
T1	<b>1</b>
T2	2
T3	3

Timestamp: 4

ts-r(C) > ts(T1).

T1 deve abortar!













[illegible]

## Transações

<b>T1:</b> write(B); write(C); read(A);	<b>T2:</b> read(B); read(A);	<b>T3:</b> read(C); write(A);
--	------------------------------------	-------------------------------------



## Controle

item	ts-r	ts-w
A	2	3
B	2	1
C	3	-1

transação	ts
T1	7
T2	2
T3	3

Timestamp: 7

T1 foi reiniciada.

Definido o novo timestamp de T1



T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);
---	-----------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	2	3
B	2	7
C	3	-1

transação	ts
T1	7
T2	2
T3	3

Timestamp: 7

Realiza a escrita e atualiza ts-w



T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);
---	-----------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	2	3
B	2	7
C	3	-1

transação	ts
T1	7
T2	2
T3	3

Timestamp: 8

T2 comita



T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);
---	-----------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	2	3
B	2	7
C	3	-1

transação	ts
T1	7
T2	2
T3	3

Timestamp: 9

T3 comita

[illegible]

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
---	-----------------------------	------------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	2	3
B	2	7
C	3	-1

Timestamp: 10

transação	ts
T1	7
T2	2
T3	3
T4	10

A transação T4 iniciou

## Definido o timestamp de T4

[illegible]

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
---	-----------------------------	------------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	2	3
B	2	7
C	3	-1

Timestamp: 10

transação	ts
T1	7
T2	2
T3	3
T4	10

$$ts-w(B) < ts(T4).$$

OK!

T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	
			R(B)

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
---	-----------------------------	------------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	2	3
B	10	7
C	3	-1

Timestamp: 10

transação	ts
T1	7
T2	2
T3	3
T4	10

Realiza a leitura e atualiza ts-r



T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	
			R(B)
W(C)			

## Transações

T1: write(B); <b>write(C);</b> read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
--	-----------------------------	------------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	2	3
B	10	7
C	3	<b>7</b>

Timestamp: 11

transação	ts
T1	<b>7</b>
T2	2
T3	3
T4	10

Realiza a escrita e atualiza ts-w



T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	
			R(B)
W(C)			

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
---	-----------------------------	------------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	2	3
B	10	7
C	3	7

Timestamp: 12

transação	ts
T1	7
T2	2
T3	3
T4	10

ts-r(A) < ts(T4).

OK!

T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	
			R(B)
W(C)			
			W(A)

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
---	-----------------------------	------------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	2	10
B	10	7
C	3	7

Timestamp: 12

transação	ts
T1	7
T2	2
T3	3
T4	10

Realiza a escrita e atualiza ts-w

T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	
			R(B)
W(C)			
			W(A)

## Transações

T1: write(B); write(C); <b>read(A);</b>	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
--	-----------------------------	------------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	2	<b>10</b>
B	10	7
C	3	7

Timestamp: 13

transação	ts
T1	<b>7</b>
T2	2
T3	3
T4	10

ts-w(A) > ts(T1).

T1 deve abortar

T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	
			R(B)
W(C)			
			W(A)
abort			

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
---	-----------------------------	------------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	2	10
B	10	7
C	3	7

Timestamp: 13

transação	ts
T1	
T2	2
T3	3
T4	10

T1 abortou e seu timestamp foi limpo

T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	
			R(B)
W(C)			
			W(A)
abort			
			commit

Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
---	-----------------------------	------------------------------	------------------------------



Controle

item	ts-r	ts-w
A	2	10
B	10	7
C	3	7

Timestamp: 14

transação	ts
T1	
T2	2
T3	3
T4	10

T4 comitou

T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	
			R(B)
W(C)			
			W(A)
abort			
			commit

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
---	-----------------------------	------------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	2	10
B	10	7
C	3	7

Timestamp: 14

transação	ts
T1	
T2	2
T3	3
T4	10

Perceba o schedule gerado é irrecuperável: T4, que acabou de comitar, usou um valor gerado por uma transação que abortou (T1).

T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	
			R(B)
W(C)			
			W(A)
abort			
			commit

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
---	-----------------------------	------------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	2	10
B	10	7
C	3	7

Timestamp: 15

transação	ts
T1	15
T2	2
T3	3
T4	10

T1 reiniciou.

Definido o novo timestamp de T1

T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	
			R(B)
W(C)			
			W(A)
abort			
			commit

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
---	-----------------------------	------------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	2	10
B	10	7
C	3	7

Timestamp: 15

transação	ts
T1	15
T2	2
T3	3
T4	10

ts-r(B) < ts(T1).

OK!



T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	
			R(B)
W(C)			
			W(A)
abort			
			commit
W(B)			

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
---	-----------------------------	------------------------------	------------------------------



## Controle

item	ts-r	ts-w
A	2	10
B	10	15
C	3	7

Timestamp: 15

transação	ts
T1	15
T2	2
T3	3
T4	10

Realiza a escrita e atualiza ts-w

T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	
			R(B)
W(C)			
			W(A)
abort			
			commit
W(B)			

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
---	-----------------------------	------------------------------	------------------------------

## Controle

item	ts-r	ts-w
A	2	10
B	10	15
C	3	7

Timestamp: 16

transação	ts
T1	15
T2	2
T3	3
T4	10

ts-r(C) < ts(T1).

OK!

T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	
			R(B)
W(C)			
			W(A)
abort			
			commit
W(B)			
W(C)			

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
---	-----------------------------	------------------------------	------------------------------

## Controle

item	ts-r	ts-w
A	2	10
B	10	15
C	3	15

Timestamp: 16

transação	ts
T1	15
T2	2
T3	3
T4	10

Realiza a escrita e atualiza ts-w

T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	
			R(B)
W(C)			
			W(A)
abort			
			commit
W(B)			
W(C)			

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
---	-----------------------------	------------------------------	------------------------------

## Controle

item	ts-r	ts-w
A	2	10
B	10	15
C	3	15

Timestamp: 17

transação	ts
T1	15
T2	2
T3	3
T4	10

ts-w(A) < ts(T1).

OK!

T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	
			R(B)
W(C)			
			W(A)
abort			
			commit
W(B)			
W(C)			
R(A)			

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
---	-----------------------------	------------------------------	------------------------------

## Controle

item	ts-r	ts-w
A	15	10
B	10	15
C	3	15

Timestamp: 17

transação	ts
T1	15
T2	2
T3	3
T4	10

Realiza a leitura e atualiza ts-r

T1	T2	T3	T4
W(B)			
	R(B)		
		R(C)	
abort			
	R(A)		
		W(A)	
W(B)			
	commit		
		commit	
			R(B)
W(C)			
			W(A)
abort			
			commit
W(B)			
W(C)			
R(A)			
commit			

## Transações

T1: write(B); write(C); read(A);	T2: read(B); read(A);	T3: read(C); write(A);	T4: read(B); write(A);
---	-----------------------------	------------------------------	------------------------------

## Controle

item	ts-r	ts-w
A	15	10
B	10	15
C	3	15

Timestamp: 18

transação	ts
T1	15
T2	2
T3	3
T4	10

T1 comita

# Timestamp Ordering

- O protocolo apresentado possui algumas limitações
  - Pode gerar schedules irrecuperáveis
  - Pode provocar starvation
- Outra das suas limitações é
  - o protocolo não limpa timestamps de leitura e escrita de itens referentes a transações que abortaram.
- Se fosse feita a limpeza, alguns aborts seriam evitados
  - Para isso, é necessário guardar um histórico contendo todos os acessos feitos a cada item
    - O exemplo a seguir mostra como

[illegible]

## Transações

T1: read(B); read(A); read(C);	T2: write(A); read(B);	T3: write(B); read(A); write(C)
---	------------------------------	--

## Controle

item	ts-r	ts-w	transação	ts
A	-1	-1	T1	
B	-1	-1	T2	
C	-1	-1	T3	

Timestamp: 0

## Realiza a leitura e atualiza ts-r



[illegible]

## Transações

T1: read(B); read(A); read(C);	T2: write(A); read(B);	T3: write(B); read(A); write(C)
---	------------------------------	--



## Controle

item	ts-r	ts-w	transação	ts
A	-1	-1	T1	1
B	-1	-1	T2	
C	-1	-1	T3	

Timestamp: 1

T1 inicia e recebe um timestamp



[illegible]

## Transações

T1: read(B); read(A); read(C);	T2: write(C); read(B);	T3: write(B); read(A); write(C)
---	------------------------------	--



## Controle

item	ts-r	ts-w	transação	ts
A	-1	-1	T1	1
B	T1	-1	T2	2
C	-1	-1	T3	

Timestamp: 2

T2 inicia e recebe um timestamp

























[illegible]

## Transações

<b>T1:</b> read(B); read(A); read(C);	<b>T2:</b> write(C); read(B);	<b>T3:</b> write(B); read(A); write(C)
--	-------------------------------------	---



# Controle

item	ts-r	ts-w	transação	ts
A	T3, T1	-1	T1	1
B	T1	T3	T2	8
C	T1	T2	T3	3

Timestamp: 10

## T1 comita





T1	T2	T3
R(B)		
	W(C)	
		W(B)
R(A)		
	abort	
		R(A)
R(C)		
	W(C)	
		W(C)
commit		
	R(B)	
		commit

## Transações

T1: read(B); read(A); read(C);	T2: write(C); read(B);	T3: write(B); read(A); write(C)
---	------------------------------	--



## Controle

item	ts-r	ts-w	transação	ts
A	T3, T1	-1	T1	1
B	T2, T1	T3	T2	8
C	T1	T2	T3	3

Timestamp: 12

T3 comita

T1	T2	T3
R(B)		
	W(C)	
		W(B)
R(A)		
	abort	
		R(A)
R(C)		
	W(C)	
		W(C)
commit		
	R(B)	
		commit
	commit	

## Transações

T1: read(B); read(A); read(C);	T2: write(C); read(B);	T3: write(B); read(A); write(C)
---	------------------------------	--



## Controle

item	ts-r	ts-w	transação	ts
A	T3, T1	-1	T1	1
B	T2, T1	T3	T2	8
C	T1	T2	T3	3

Timestamp: 13

T2 comita

# Objetivo do Trabalho

- A classe XXXOptimisticConcurrencyManager deve implementar três funções
  - **processInstruction**: função que processa a próxima instrução de uma transação
  - **getTransactionStartTime**: recupera o timestamp de uma transação e define um em caso de necessidade
  - **abort**: função que se encarrega de abortar uma transação

# Objetivo do Trabalho

@Override

```
public List<Record> processInstruction(Transaction t) throws Exception {
```

```
    boolean ignore = false;
```

```
    currentTime++;
```

```
    Integer tStart = getTransactionStartTime(t);
```

```
    //puts the transaction in the recovery log if its  
    logTansactionStart(t);
```

```
    Instruction i = t.getCurrentInstruction();
```

```
    if (i instanceof SingleUpdateInstruction) {
```

```
        ...
```

```
    } else if (i instanceof SingleReadInstruction) {
```

```
        ...
```

```
    }
```

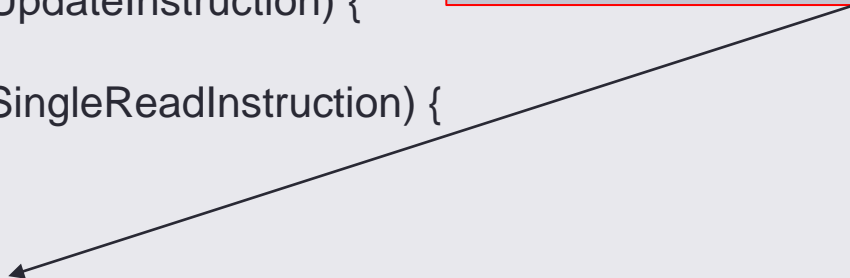
```
    return processCurrentInstruction(t, ignore);
```

```
}
```

A implementação atual processa todas as instruções.

O objetivo é estender essa implementação e realizar a validação das regras.

Em caso de falha, deve ser provocado o abort, antes de chegar nesse último comando



# Objetivo do Trabalho

@Override

```
public List<Record> processInstruction(Transaction t) throws Exception {
```

```
    boolean ignore = false;  
    currentTime++;  
    Integer tStart = getTransactionStartTime(t);
```

A validação das regras também poderá fazer com que a instrução seja aceita, mas não processada

```
    //puts the transaction in the recovery log if its not already there  
    logTansactionStart(t);
```

```
    Instruction i = t.getCurrentInstruction();
```

```
    if (i instanceof SingleUpdateInstruction) {  
        ...  
    } else if (i instanceof SingleReadInstruction) {  
        ...  
    }
```

```
    return processCurrentInstruction(t, ignore);
```

```
}
```

# Objetivo do Trabalho

@Override

```
public List<Record> processInstruction(Transaction t) throws Exception {
```

```
    boolean ignore = false;
```

```
    currentTime++;
```

```
    Integer tStart = getTransactionStartTime(t);
```

Esta função também precisa ser implementada. Cabe a ela retornar o timestamp de início da transação, e definir um novo, caso seja necessário.

```
    //puts the transaction in the recovery log if its not already there  
    logTansactionStart(t);
```

```
    Instruction i = t.getCurrentInstruction();
```

```
    if (i instanceof SingleUpdateInstruction) {
```

```
        ...
```

```
    } else if (i instanceof SingleReadInstruction) {
```

```
        ...
```

```
    }
```

```
    return processCurrentInstruction(t, ignore);
```

```
}
```

# Objetivo do Trabalho

@Override

```
public List<Record> processInstruction(Transaction t) throws Exception {
```

```
    boolean ignore = false;
```

```
    currentTime++;
```

```
    Integer tStart = getTransactionStartTime(t);
```

```
    //puts the transaction in the recovery log if its not already there  
    logTansactionStart(t);
```

```
    Instruction i = t.getCurrentInstruction();
```

```
    if (i instanceof SingleUpdateInstruction) {
```

```
        ...
```

```
    } else if (i instanceof SingleReadInstruction) {
```

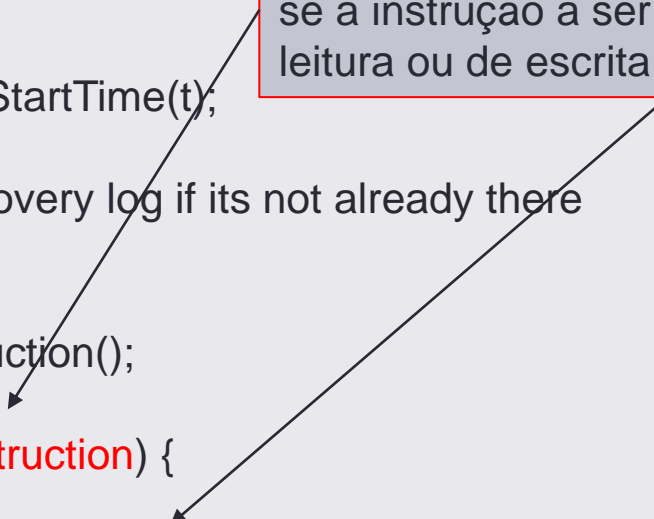
```
        ...
```

```
    }
```

```
    return processCurrentInstruction(t, ignore);
```

```
}
```

Para validar as regras, deve-se verificar se a instrução a ser processada é de leitura ou de escrita



# Objetivo do Trabalho

@Override

```
public List<Record> processInstruction(Transaction t) throws Exception {
```

```
    boolean ignore = false;
```

```
    currentTime++;
```

```
    Integer tStart = getTransactionStartTime(t);
```

```
    //puts the transaction in the recovery log if its  
    logTansactionStart(t);
```

```
    Instruction i = t.getCurrentInstruction();
```

```
    if (i instanceof SingleUpdateInstruction) {
```

```
        ...
```

```
    } else if (i instanceof SingleReadInstruction) {
```

```
        ...
```

```
    }
```

```
    return processCurrentInstruction(t, ignore);
```

```
}
```

Aqui será realizada boa parte da implementação, que envolve atualizar as listas referentes às leituras e escritas de cada item e validar as regras.

Se julgar necessário, crie funções auxiliares para modularizar o código.



# Objetivo do Trabalho

@Override

```
public List<Record> processInstruction(Transaction t) throws Exception {
```

```
    boolean ignore = false;
```

```
    currentTime++;
```

```
    Integer tStart = getTransactionStartTime(t);
```

```
    //puts the transaction in the recovery log if its not already there  
    logTansactionStart(t);
```

```
    Instruction i = t.getCurrentInstruction();
```

```
    if (i instanceof SingleUpdateInstruction) {
```

```
        ...
```

```
    } else if (i instanceof SingleReadInstruction) {
```

```
        ...
```

```
    }
```

```
    return processCurrentInstruction(t, ignore);
```

```
}
```

Caso se verifique a necessidade de abort, esse é o trecho que deve ser inserido

```
    abort(t);  
    return null;
```

# Objetivo do Trabalho

- A função **abort** vem com uma implementação padrão
- O objetivo é incluir os processos de limpeza dos dados de controle para a transação que está abortando

```
@Override
protected void abort(Transaction t) throws Exception {
    super.abort(t);

    //inclua a limpeza aqui
    ...
}
```

# Dicas

- As regras exigem que informações referentes às transações sejam mantidas em estruturas de controle
  - Timestamp de cada transação
  - Lista de transações que acessaram cada item
- Investigue quais estruturas de dados do Java mais se adequam para resolver o problema proposto
- Obs. limite-se aos tipos disponibilizados pelo Java
  - Não use bibliotecas de terceiros, já que apenas a classe do gerenciador será entregue

# Testes

- Para testar, é necessário trocar o gerenciador de acesso concorrente usado pelo simulador

```
public void run(int error) throws Exception {  
    Main1 m = new Main1();  
    m.test2(new XXXTimestampOrderingConcurrencyManager());  
    ...  
}
```

- Teste com diferentes cenários para garantir que o controle seja realizado de forma consistente
  - O resultado de alguns testes está publicado no moodle

# Avaliação

- Os principais aspectos que serão analisados para a avaliação do trabalho são
  - O código deve estar funcional
  - Mensagens de depuração foram omitidas
  - As funções complementares (se necessárias) foram criadas como privadas
  - O arquivo correto foi enviado (.java)
  - O arquivo implementa a classe especificada
  - O pacote da classe foi especificado da forma correta
    - Crie o pacote `timestamp.ordering` dentro de `ibd.transaction.concurrency` para evitar problemas de compilação
  - O schedule gerado é o correto

# Entrega

- Entrega pelo moodle
- Não entregue o projeto inteiro
  - Apenas a classe java solicitada
- O trabalho é **individual**
  - O compartilhamento de código entre alunos leva à anulação da nota

# Entrega

- A nota máxima possível depende do dia em que for feita a entrega

Prazo	Nota máxima
24/11 23h59min (sexta)	100%
25/11 23h59min (sábado)	80%
26/11 23h59min (domingo)	60%
27/11 23h59min (segunda)	40%

- Entregas feitas após o dia 27/11 não serão avaliadas