

Centro Federal de Educação Tecnológica de Minas Gerais
Campus VII - Unidade Timóteo - Engenharia da Computação
Trabalho Prático 5

Trabalho Prático V: Aplicação Multithread

Arthur Moraes Pimentel

Professor: Lucas Pantuza Amorim

Timóteo, Dezembro de 2022

1 Introdução

Neste trabalho será apresentado uma proposta de solução para o problema do compartilhamento de um recurso comum entre várias threads. Foi desenvolvido uma aplicação multithread, onde dado um cenário de propagação de notícias, teve-se como principal foco abordar os conteúdos estudados em sala de aula, bem como exercitar o conceito de várias threads acessando ao mesmo tempo um recurso compartilhado, que se trata das regiões críticas. O problema trata de duas versões para este cenário, que busca abordar uma variação na quantidade de produtores e na quantidade de consumidores.

2 Metodologia

Para desenvolvimento do trabalho foi utilizado a linguagem de programação Java, em que a JVM permite que a aplicação tenha várias threads de execução em execução simultaneamente, onde cada uma delas tem uma prioridade respectiva.

Para usar as threads é preciso estendê-las na classe que irá comportar como uma, é preciso também criar um método, sobrescrito, que é o método run, ele é responsável por executar o trecho de código correspondente aquela thread. O método run só será executado após ser executado o start, que é um método que tem como funcionalidade inicializar a thread que no caso é estendida em uma classe java.

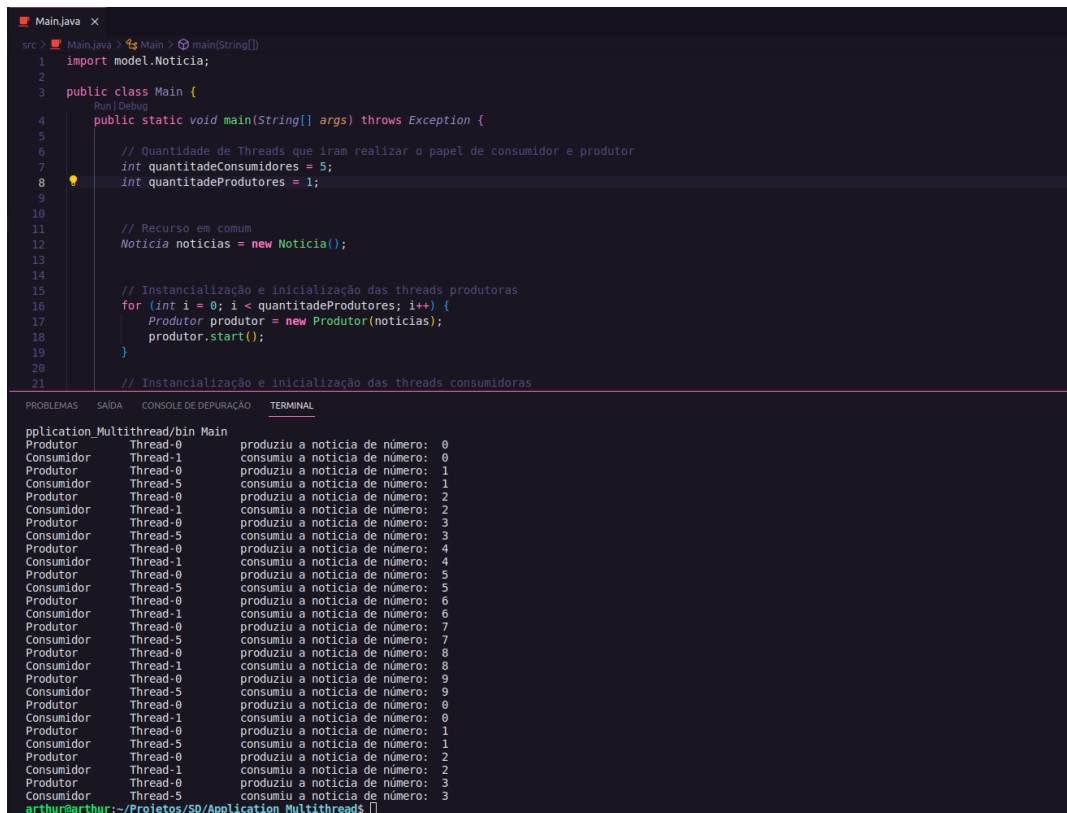
A IDE utilizada foi o VSCode, que é um editor de código bem completo, permitindo criar e testar o projeto java.

3 Resultados

Foi realizado três testes de execução da aplicação, na qual corresponde a quantidade de threads para consumir e produzir uma notícia.

Para ambos os testes foram gerados logs no terminal indicando quem está atuando, produtor ou consumidor, identificação da thread correspondente, e por fim a informação de produzir ou consumir a notícia que se identifica pelo número que representa o seu índice no buffer de notícias.

No primeiro caso, temos 1 thread atuando como produtora, e 5 threads atuando como consumidoras. O resultado obtido pode ser visualizado na seguinte captura de tela 1. Pode se observar pela segunda coluna os identificadores de cada thread, para este caso as thread consumidoras assumiram um comportamento de espera em maior parte do tempo, pois o buffer tende a ficar vazio, uma vez que a cada notícia produzida pela única thread produtora, uma thread consumidora já está aguardando para consumir.



```
src > Main.java > Main > main(String[])
1 import model.Noticia;
2
3 public class Main {
4     public static void main(String[] args) throws Exception {
5
6         // Quantidade de Threads que iram realizar o papel de consumidor e produtor
7         int quantidadeConsumidores = 5;
8         int quantidadeProdutores = 1;
9
10
11         // Recurso em comum
12         Noticia noticias = new Noticia();
13
14         // Instancionalização e inicialização das threads produtoras
15         for (int i = 0; i < quantidadeProdutores; i++) {
16             Produtor produtor = new Produtor(noticias);
17             produtor.start();
18         }
19
20         // Instancionalização e inicialização das threads consumidoras
21     }
```

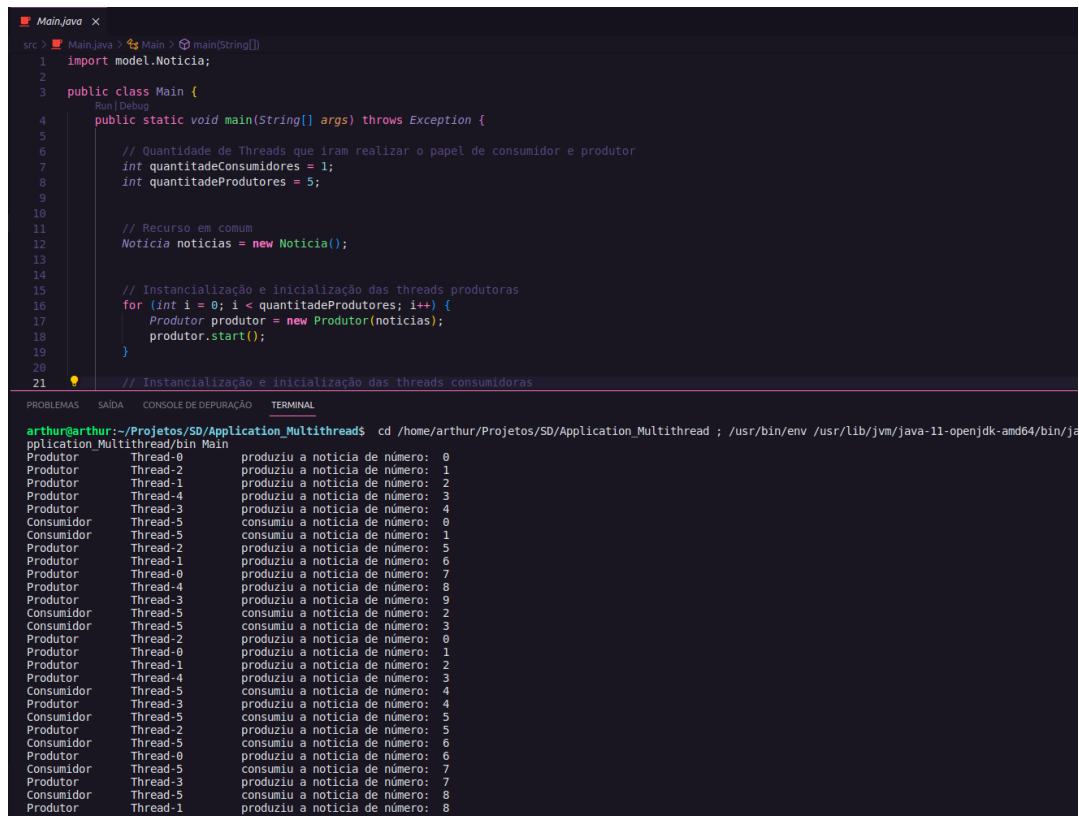
pplication_Multithread/bin Main

Thread	Ação	Número
Thread-0	produziu a notícia de	0
Thread-1	consumiu a notícia de	0
Thread-0	produziu a notícia de	1
Thread-5	consumiu a notícia de	1
Thread-0	produziu a notícia de	2
Thread-1	consumiu a notícia de	2
Thread-0	produziu a notícia de	3
Thread-5	consumiu a notícia de	3
Thread-0	produziu a notícia de	4
Thread-1	consumiu a notícia de	4
Thread-0	produziu a notícia de	5
Thread-5	consumiu a notícia de	5
Thread-0	produziu a notícia de	6
Thread-1	consumiu a notícia de	6
Thread-0	produziu a notícia de	7
Thread-5	consumiu a notícia de	7
Thread-0	produziu a notícia de	8
Thread-1	consumiu a notícia de	8
Thread-0	produziu a notícia de	9
Thread-5	consumiu a notícia de	9
Thread-0	produziu a notícia de	0
Thread-1	consumiu a notícia de	0
Thread-0	produziu a notícia de	1
Thread-5	consumiu a notícia de	1
Thread-0	produziu a notícia de	2
Thread-1	consumiu a notícia de	2
Thread-0	produziu a notícia de	3
Thread-5	consumiu a notícia de	3

arthur@arthur:~/Projetos/SD/Application_Multithreads []

Figura 1: 1 Produtor 5 Consumidor

No segundo caso, temos 1 thread atuando como consumidora, e 5 threads atuando como produtora. O resultado obtido pode ser visualizado na seguinte captura de tela 2. Pode se observar pela segunda coluna os identificadores de cada thread, para este caso as threads produtoras assumiu um comportamento de produção bem rápido inicialmente, e depois de o buffer está praticamente cheio elas passaram a assumir um comportamento de produção mais lento, isso pelo efeito de termos mais de uma thread produzindo, oque acarretou em uma produção maior para um consumo menor. A tendência é que o buffer venha encher mais rápido do que esvaziar.

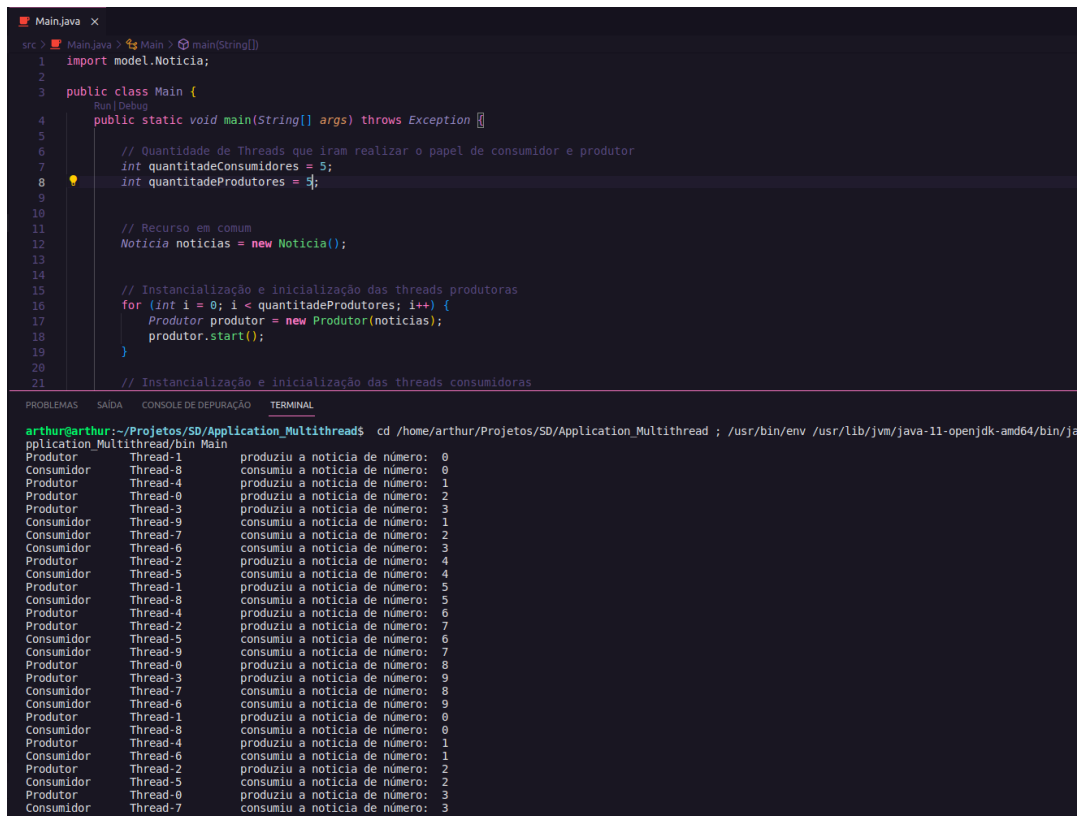


```
src > Main.java > Main > main(String[])
1 import model.Noticia;
2
3 public class Main {
4     // Quantidade de Threads que iram realizar o papel de consumidor e produtor
5     // Recurso em comum
6     // Instanciacao e inicializacao das threads produtoras
7     // Instanciacao e inicializacao das threads consumidoras
8
9     public static void main(String[] args) throws Exception {
10
11         // Quantidade de Threads que iram realizar o papel de consumidor e produtor
12         int quantidadeConsumidores = 1;
13         int quantidadeProdutores = 5;
14
15         // Recurso em comum
16         Noticia noticias = new Noticia();
17
18         // Instanciacao e inicializacao das threads produtoras
19         for (int i = 0; i < quantidadeProdutores; i++) {
20             Produtor produtor = new Produtor(noticias);
21             produtor.start();
22         }
23
24         // Instanciacao e inicializacao das threads consumidoras
25         Consumidor consumidor = new Consumidor(noticias);
26         consumidor.start();
27     }
28 }
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

```
PROBLEMAS SAIDA CONSOLE DE DEPURACAO TERMINAL
arthur@arthur:~/Projetos/SD/Application_Multithreads$ cd /home/arthur/Projetos/SD/Application_Multithread ; /usr/bin/env /usr/lib/jvm/java-11-openjdk-amd64/bin/java -cp ./bin Main
application_Multithread/bin Main
Produtor Thread-0 produziu a noticia de numero: 0
Produtor Thread-2 produziu a noticia de numero: 1
Produtor Thread-1 produziu a noticia de numero: 2
Produtor Thread-4 produziu a noticia de numero: 3
Produtor Thread-3 produziu a noticia de numero: 4
Consumidor Thread-5 consumiu a noticia de numero: 0
Produtor Thread-2 produziu a noticia de numero: 5
Produtor Thread-1 produziu a noticia de numero: 6
Produtor Thread-0 produziu a noticia de numero: 7
Produtor Thread-4 produziu a noticia de numero: 8
Produtor Thread-3 produziu a noticia de numero: 9
Consumidor Thread-5 consumiu a noticia de numero: 2
Consumidor Thread-5 consumiu a noticia de numero: 3
Produtor Thread-2 produziu a noticia de numero: 0
Produtor Thread-0 produziu a noticia de numero: 1
Produtor Thread-1 produziu a noticia de numero: 2
Produtor Thread-4 produziu a noticia de numero: 3
Consumidor Thread-5 consumiu a noticia de numero: 4
Produtor Thread-3 produziu a noticia de numero: 4
Consumidor Thread-5 consumiu a noticia de numero: 5
Produtor Thread-2 produziu a noticia de numero: 5
Consumidor Thread-5 consumiu a noticia de numero: 6
Produtor Thread-0 produziu a noticia de numero: 6
Consumidor Thread-5 consumiu a noticia de numero: 7
Produtor Thread-3 produziu a noticia de numero: 7
Consumidor Thread-5 consumiu a noticia de numero: 8
Produtor Thread-1 produziu a noticia de numero: 8
```

Figura 2: 5 Produtor 1 Consumidor

No terceiro caso, temos 5 thread atuando como consumidora, e 5 threads atuando como produtora. O resultado obtido pode ser visualizado na seguinte captura de tela 3. Pode se observar pela segunda coluna os identificadores de cada thread, para este caso houve uma grande variação de threads, uma vez que com a criação de 5 para cada funcionalidade, a prioridade acabou se normalizando, isso podemos ver pela distribuição das thread ao produzir e consumir, ocorreu uma grande variação de uma pra outra.



```
src > Main.java > Main > main(String[])
1 import model.Noticia;
2
3 public class Main {
4     public static void main(String[] args) throws Exception {}
5
6     // Quantidade de Threads que iram realizar o papel de consumidor e produtor
7     int quantidadeConsumidores = 5;
8     int quantidadeProdutores = 5;
9
10
11     // Recurso em comum
12     Noticia noticias = new Noticia();
13
14     // Instanciacao e inicializacao das threads produtoras
15     for (int i = 0; i < quantidadeProdutores; i++) {
16         Produtor produtor = new Produtor(noticias);
17         produtor.start();
18     }
19
20     // Instanciacao e inicializacao das threads consumidoras
21 }
```

```
artur@arthur:~/Projetos/SD/Application_Multithreads$ cd /home/artur/Projetos/SD/Application_Multithread ; /usr/bin/env /usr/lib/jvm/java-11-openjdk-amd64/bin/java
Application_Multithread/bin Main
Produtor Thread-1 produziu a noticia de número: 0
Consumidor Thread-8 consumiu a noticia de número: 0
Produtor Thread-4 produziu a noticia de número: 1
Produtor Thread-0 produziu a noticia de número: 2
Consumidor Thread-9 consumiu a noticia de número: 1
Consumidor Thread-7 consumiu a noticia de número: 2
Produtor Thread-2 produziu a noticia de número: 3
Consumidor Thread-6 consumiu a noticia de número: 3
Produtor Thread-5 consumiu a noticia de número: 4
Produtor Thread-1 produziu a noticia de número: 5
Consumidor Thread-8 consumiu a noticia de número: 4
Produtor Thread-4 produziu a noticia de número: 6
Consumidor Thread-5 consumiu a noticia de número: 5
Produtor Thread-2 produziu a noticia de número: 7
Consumidor Thread-9 consumiu a noticia de número: 6
Produtor Thread-0 produziu a noticia de número: 8
Consumidor Thread-7 consumiu a noticia de número: 7
Produtor Thread-3 produziu a noticia de número: 9
Consumidor Thread-6 consumiu a noticia de número: 8
Produtor Thread-1 produziu a noticia de número: 0
Consumidor Thread-8 consumiu a noticia de número: 9
Produtor Thread-4 produziu a noticia de número: 1
Consumidor Thread-5 consumiu a noticia de número: 0
Produtor Thread-2 produziu a noticia de número: 2
Consumidor Thread-9 consumiu a noticia de número: 1
Produtor Thread-0 produziu a noticia de número: 3
Consumidor Thread-7 consumiu a noticia de número: 2
Consumidor Thread-6 consumiu a noticia de número: 3
```

Figura 3: 5 Produtor 5 Consumidor

4 Análise

O conceito de trabalhar em cima de um recurso compartilhado permite que várias threads acessem e operem em cima do mesmo conteúdo. O fato de ter disponivel um mesmo conteúdo para várias threads nem sempre é uma boa opção se não houver um tratamento, isso pois pode ocorrer o que chamamos de região crítica que trata-se de uma área do código que acessa um recurso compartilhado que não pode ser acedido concorrentemente por mais de uma linha de execução. No trabalho proposto é abordado uma possível solução de contornar a região crítica.

O recurso compartilhado trata-se de um objeto Noticia, que possui um array de StringBuffer, limitado por um tamanho de 10 posições, onde será acessado pelas várias threads criadas. Inicialmente se não houvesse uma tratativa, as threads acessariam desordenadamente as posições do array. Teriamos cenários como uma thread acessando uma notícia que talvez já foi consumida por outra thread, ou então, mais de uma tentar consumir ou produzir uma noticia em certa posição.

A forma de solucionar a zona critica, foi garantir que uma certa noticia, posição do array, fosse acessada por uma única thread no momento de consumo ou produção. Desta maneira foi aprofundado o uso do `synchronized`, que permite que o método de consumo quanto de produção seja acessada no ato de consumir ou produzir por uma única thread, desta forma, não corremos o risco de mais de uma thread tentar produzir uma noticia para a mesma posição do array, ou então, mais de uma thread tentar consumir uma mesma noticia em uma posição `x` do array.

Apesar do uso do `synchronized`, ainda sim foi preciso realizar outras tratativas para a aplicação em si. O array de noticias, possui um tamanho limitado, ou seja, pode possuir apenas `x` noticias. Deste modo foi criado um contador, que a cada nova noticia produzida ele sofreria de um acrescimo, e a cada noticia consumida ele sofreria de um decrescimo.

Esse contador (`quantidadeNoticias`) foi de tal importância pois quando alcançado um valor `x`, correspondente ao limite do array, as threads produtoras entrarão em um estado de espera, portanto não produzirão até que o array tenha espaço liberado. Caso esse contador tenha um valor respectivo a zero, ou seja, indica que o array de noticias não possui noticias, as threads consumidoras entrarão em estado de espera por não haver noticias no array.

Outro tratamento realizado, para poder evitar a região critica, foi o uso de dois indices, `proximaNoticiaConsumir` e `proximaNoticiaProduzir`, estes indices são responsáveis por informar a posição a ser acessado pela thread consumidora ou produtora. Sempre que uma thread consumidora consome uma noticia, ela atualiza este indice, para que a proxima consumidora não acesse a posição consumida que no caso estaria vazia. De mesma forma, sempre que uma thread produtora produz uma noticia, ela atualiza o indice correspondente, `proximaNoticiaProduzir`, que indica para proxima thread produtora a posição a ser acessada, para que não ocorra de produzir em cima de uma noticia que ja foi produzida.

5 Conclusão

Neste trabalho foi possível ver e experimentar na prática o uso de multithreads atuando em um único recurso compartilhado, bem como visualizar e entender melhor os métodos padrão que uma thread pode ter, como o `wait` que a faz esperar, e o `notifyAll` que notifica todas as demais threads.

A região critica ficou mais clara de entender, bem como as diversas soluções possíveis para trata-lás.

6 Código

Todo o código fonte se encontra disponível em [GitHub](#).

Segue o código comentado:

6.1 Aplicação Multithread

6.1.1 Main

Onde será inicializado a aplicação, definido a quantidade de threads consumidoras e a quantidade de threads produtoras, bem como inicia-lás pelo metodo start. E definido também o objeto Noticia que será o recurso compartilhado entre as threads.

```
1 import model. Noticia ;
2
3 public class Main {
4     public static void main( String[] args ) throws Exception {
5
6         // Quantidade de Threads que iram realizar o papel de consumidor e
           produtor
7         int quantidadeConsumidores = 1;
8         int quantidadeProdutores = 5;
9
10
11         // Recurso em comum
12         Noticia noticias = new Noticia ();
13
14
15         // Instancia o e inicializa o das threads produtoras
16         for (int i = 0; i < quantidadeProdutores; i++) {
17             Produtor produtor = new Produtor(noticias);
18             produtor.start();
19         }
20
21         // Instancia o e inicializa o das threads consumidoras
22         for (int i = 0; i < quantidadeConsumidores; i++) {
23             Consumidor consumidor = new Consumidor(noticias);
24             consumidor.start();
```

```
25     }  
26  
27     }  
28 }
```

codes/Main/Main.java

6.1.2 Consumidor

A classe Consumidor extend a Thread, pois a mesma atuará como uma, sendo assim necessário implementar o método run que será usado pela thread ao ser realizado a sua inicialização. A thread consumidora irá consumir infinitamente, até que a aplicação seja encerrada. Ao consumir a thread aguarda 2000 mls para poder continuar a consumir.

```
1 import model. Noticia ;  
2  
3 public class Consumidor extends Thread {  
4     Noticia noticias ;  
5  
6     public Consumidor( Noticia noticias ) {  
7         this . noticias = noticias ;  
8     }  
9  
10    @Override  
11    public void run () {  
12        while ( true ) {  
13            this . noticias . consumir ( this . getName () ) ;  
14            try {  
15                sleep ( 1000 ) ;  
16            } catch ( Exception e ) {  
17                e . printStackTrace () ;  
18            }  
19        }  
20    }  
21  
22 }
```

codes/Main/Consumidor.java

6.1.3 Produtor

A classe Produtor extend a Thread, pois a mesma atuará como uma, sendo assim necessário implementar o método run que será usado pela thread ao ser realizado a sua inicialização. A thread produtora irá produzir infinitamente, até que a aplicação seja encerrada. Ao produzir a thread aguarda 1000 mls para poder continuar a produzir.

```
1 import model. Noticia ;
2
3 public class Produtor extends Thread{
4     Noticia noticias ;
5
6     public Produtor( Noticia noticias ){
7         this. noticias = noticias ;
8     }
9
10    @Override
11    public void run () {
12        while ( true ) {
13            noticias . produzir ( this . getName () ) ;
14            try {
15                sleep ( 2000 ) ;
16            } catch ( Exception e ) {
17                e . printStackTrace () ;
18            }
19        }
20    }
21 }
22 }
```

codes/Main/Produtor.java

6.2 Model

6.2.1 Noticia

A classe Noticia representa o recurso compartilhado, portanto é onde se encontra o array que será acessado pelas threads, bem como os índices, e os métodos produzir e consumir que serão executados pelas respectivas threads.

```
1 package model ;
```

```
2
3 import java.io.BufferedReader;
4 import java.io.File;
5 import java.io.FileReader;
6 import java.io.IOException;
7
8 public class Noticia {
9     // Array de Dados
10    private StringBuffer[] noticias;
11    // Limite do array
12    private int limite = 10;
13    // Quantidade de dados presente no array
14    private int quantidadeNoticias = 0;
15    // Indice da proxima posi o a ser consumida
16    private int proximaNoticiaConsumir = 0;
17    // Indice da proxima posi o a ser produzida
18    private int proximaNoticiaProduzir = 0;
19
20    public Noticia() {
21        this.noticias = new StringBuffer[limite];
22    }
23
24    public synchronized void consumir(String idThread) {
25
26        while (this.quantidadeNoticias == 0) {
27            try {
28                // Coloca as demais threads consumidoras a dormir
29                this.wait();
30            } catch (InterruptedException e) {
31                e.printStackTrace();
32            }
33        }
34
35        // Consume a noticia
36        System.out.println("Consumidor \t" + idThread + "\t consumiu a
37        noticia de n mero: \t" + proximaNoticiaConsumir);
38        this.noticias[proximaNoticiaConsumir] = null;
39        quantidadeNoticias--;
40
41        // Atualiza o indice da proxima noticia a ser consumida
42        if (proximaNoticiaConsumir + 1 == limite)
43            proximaNoticiaConsumir = 0;
44        else
45            proximaNoticiaConsumir++;
46
47        // Notifica as demais threads
48        notifyAll();
49    }
50 }
```

```
49
50 public synchronized void produzir(String idThread) {
51     while (quantidadeNoticias == limite) {
52         try {
53             // Coloca as threads produtoras para dormir
54             this.wait();
55         } catch (InterruptedException e) {
56             e.printStackTrace();
57         }
58     }
59
60     // Gera uma nova noticia
61     StringBuffer novaNoticia = gerarNovaNoticia();
62
63     noticias[proximaNoticiaProduzir] = novaNoticia;
64
65     System.out.println("Produtor \t" + idThread + "\t produziu a
66                       noticia de n mero: \t" + proximaNoticiaProduzir);
67
68     // Atualiza o indice que implica na proxima posi o
69     // a ser utilizada para armazenar a nova noticia
70     if (proximaNoticiaProduzir + 1 == limite)
71         proximaNoticiaProduzir = 0;
72     else
73         proximaNoticiaProduzir++;
74
75     quantidadeNoticias++;
76
77     // Notifica as demais threads
78     notifyAll();
79
80 }
81
82 // Realiza a leitura de um arquivo TXT
83 // Gera um StringBuffer
84 private StringBuffer gerarNovaNoticia() {
85     File file = new File("src/Util/Texto.txt");
86     StringBuffer texto = new StringBuffer();
87     try {
88         file.createNewFile();
89
90         FileReader fileReader = new FileReader(file);
91         BufferedReader bufferedReader = new BufferedReader(fileReader)
92             ;
93
94         while (bufferedReader.ready()) {
95             texto.append(bufferedReader.readLine() + "\n");
96         }
97     } catch (IOException e) {
98         e.printStackTrace();
99     }
100 }
```

```
95         }
96
97         bufferedReader.close();
98         fileReader.close();
99     } catch (IOException erro) {
100         System.out.printf("Erro: %s", erro.getMessage());
101     }
102     return texto;
103 }
104 }
```

codes/Model/Noticia.java

6.3 Util

6.3.1 Texto

Este arquivo de texto simulada uma noticia, que será utilizado pelas threads produtoras.

```
1 Lorem Ipsum is simply dummy text of the printing and typesetting industry.
2 Lorem Ipsum has been the industry's standard dummy text ever since the
  1500s,
3 when an unknown printer took a galley of type and scrambled it to make a
  type specimen book.
4 It has survived not only five centuries, but also the leap into electronic
  typesetting, remaining essentially unchanged.
5 It was popularised in the 1960s with the release of Letraset sheets
  containing Lorem Ipsum passages,
6 and more recently with desktop publishing software like Aldus PageMaker
  including versions of Lorem Ipsum.
```

codes/Util/Texto.txt
