Data de Entrega: 4.Novembro
O que deve ser entregue: 1) Deve ser enviado por email um arquivo zipado contendo os códigos fontes dos programas implementados;
Regras para entrega do trabalho prático: Este trabalho deve ser realizado individualmente. O aluno deverá enviar o trabalho em um arquivo zipado, cujo nome deve respeitar o seguinte formato: CK069_<matricula do aluno>_T<numero do trabalho pratico>, por exemplo, CK069_123456_T1 é o arquivo contendo o trabalho pratico 1 do estudante com matricula 123456. O arquivo compactado deve ser enviado pelo SIGAA Caso o aluno não respeite as regras apresentadas anteriormente, poderá sofrer redução na nota do trabalho.

Laboratório 6 – Sincronização Processos

Questão 1 (4.0 pontos) Neste laboratório você deve resolver um desafio de sincronização conforme o código abaixo. De fato, este programa implementa um exemplo de modelo produtor-consumidor. Neste programa são criados dois consumidores e um produtor. Um consumidor produz somente números pares e outro consumidor gera somente números impares. O único produtor existente, gera números pares e ímpares, indistintamente. Claramente, um número consumido deve ser eliminado logo após sua utilização. Analise o código abaixo, identifique o problema deste código e implemente a solução para o problema.

```
public class ProducerConsumerExample {

    private static boolean Even = true;
    private static boolean Odd = false;

    public static void main(String[] args) {
        Dropbox dropbox = new Dropbox();
        (new Thread(new Consumer(Even, dropbox))).start();
        (new Thread(new Consumer(Odd, dropbox))).start();
        (new Thread(new Producer(dropbox))).start();
    }
}
```

```
public class Dropbox {
    private int number;
    private boolean evenNumber = false;
    public int take(final boolean even) {
        System.out.format("%s CONSUMIDOR obtem %d.%n", even ? "PAR" : "IMPAR",
number);
        return number;
    }

    public void put(int number) {
        this.number = number;
        evenNumber = number % 2 == 0;
        System.out.format("PRODUTOR gera %d.%n", number);
    }
}
```

```
import java.util.Random;

public class Producer implements Runnable {

    private Dropbox dropbox;

    public Producer(Dropbox dropbox) {
        this.dropbox = dropbox;
    }

    public void run() {
        Random random = new Random();
        while (true) {
            int number = random.nextInt(10);
            try {
                Thread.sleep(random.nextInt(100));
                dropbox.put(number);
            } catch (InterruptedException e) { }
        }
    }
}
```

```
import java.util.Random;
public class Consumer implements Runnable {
    private final Dropbox dropbox;
    private final boolean even;
    public Consumer(boolean even, Dropbox dropbox) {
        this.even = even;
        this.dropbox = dropbox;
    }
    public void run() {
        Random random = new Random();
        while (true) {
            dropbox.take(even);
            try {
               Thread.sleep(random.nextInt(100));
            } catch (InterruptedException e) { }
        }
    }
}
```

Questão 2 (3.0 pontos) Um problema clássico de sincronização entre processos, conhecido como o problema dos Leitores e Escritores, ocorre quando um conjunto de dados é compartilhado entre um certo número de processos concorrentes de dois tipos: **Readers** – só leem o conjunto de dados; eles não realizam atualizações e **Writers** – podem ler e gravar dados. Neste exercício, o problema é permitir que múltiplos **readers** leiam ao mesmo tempo e somente um único **writer** possa acessar o conjunto de dados compartilhados em um determinado momento. Implemente duas soluções para este problema, onde a primeira solução utilize semáforos e a segunda utilize monitores.

Questão 3 (3.0 pontos) Outro problema clássico de sincronização é conhecido como o jantar dos filósofos. Neste problema, há cinco filósofos em torno de uma mesa, um garfo é colocado entre cada filósofo, cada filósofo deve,alternadamente, refletir e comer. Para que um filósofo coma, ele deve possuir dois garfos. Os dois garfos devem ser aqueles logo a sua esquerda e a sua direita. Para pegar um garfo, somente pode ser pego por um filósofo e somente pode ser pego não estiver em uso por nenhum outro filósofo. Após comer, o filósofo deve liberar o garfo que utilizou. Um filósofo pode segurar o garfo da sua direita ou o da sua esquerda assim que estiverem disponíveis, mas, só pode começar a comer quando ambos estiverem sob sua posse. Implemente duas soluções para este problema, onde a primeira solução utilize semáforos e a segunda utilize monitores.