Candidato:
Data: **Início:** **Término:**
Modelo: Desenvolvedor Java Sr/Especialista em Engenharia de Sistemas

Orientações Gerais:

- A avaliação será feita com base na **qualidade** das respostas. Uma questão repondida por completo, com boa estrutura e refinamento, terá mais valor que a soma de várias respostas incompletas
- A prova aborda de maneira eclética diferentes temas do ecossistema Java e tem por objetivo avaliar o nível de familiaridade do avaliado com relação aos mesmos.
- Assuma visibilidade pública de classes, métodos e atributos, quando não especificado
- Modificadores de visibilidade ou outros, exceto ***volatile***, quando aplicável, são dispensáveis na resolução dos problemas, mesmo quando implementando interfaces.
- Evite o uso de classes de frameworks ou bibliotecas específicas (e.g. org.apache.commons.lang.StringUtils) a menos quando solicitado no enunciado.

Conceitos Básicos

1. Complete o corpo do método a seguir, utilizando apenas construções básicas da linguagem (nada além do contido no pacote java.lang),

```
byte[] split(long val){

}
```

de modo que o retorno seja a representação *little-endian* do valor da variável val.

2. Considere uma lista imutável de objetos do tipo T. Assumindo que exista uma função de hash perfeita para esse conjunto, crie uma estrutura de dados que realize a associação T => (posição de T na lista), com complexidade O(1).

3. Descreva a diferença entre os algoritmos de *Garbage Collection* CMS (*Concurrent Mark Sweep*) e *Parallel*. De exemplos de aplicações nas quais é mais indicado o uso de CMS ao invés de *Parallel* e vice-versa.

Streams

4. Considere o seguinte problema: Dados dois conjuntos ordenados $A \subseteq X\_32$ e $B \subseteq X\_32$, onde $X\_32=[2^{16},2^{16})$, construa métodos que retornem uma representação de um conjunto $C=A \cap B$, considerando as seguintes abordagens:

a) A implementação deve retornar uma representação *eager*. Nesse caso, descrever qual a complexidade do algoritmo utilizando a notação *Big-O*
b) A implementação deve retornar uma representação *lazy* e deve poder ser representada como um Stream.

Descreva uma situação em que é preferível utilizar a implementação *eager* ao invés da *lazy*.

5. Considere a seguinte classe

```
class A {

   int id;
   String label;
   LocalDate day;
   short offset;

}
```

Considerando que os valores de offset se encontram no intervalo [0,1440), dada uma coleção de elementos A, implementar um método

```
public Stream<A> withGaps(Collection<A> source,LocalDate begin,LocalDate end){
//...
}
```

, de forma que a iteração nesse Stream através do método forEach, satisfaça:

- A ordem de encontro dos elementos deverá ser ascendente, de acordo com o valor de day e em seguida de offset. Assuma que a coleção source não apresenta valores de offset duplicados para um mesmo valor de day.
- Os elementos deverão ter valor de **day** superior ou igual ao parâmetro begin e inferior ao parâmetro end.
- Para cada **day**, espera-se que haja 1440 elementos. Se para um dado **day** a coleção original não possuir **offsets** suficientes para completar 1440 elementos, estes deverão ser apresentados como instâncias “sintéticas”, com **id=**-1, **label=**”FAKE” e **offsets** que completem os elementos ausentes. E.g., se para determinado valor de **day** houver 3 instâncias de A com **offsets**, 4, 121 e 256, espera-se que na iteração dos 4 primeiros elementos (offsets 0 a 3) sejam apresentadas instâncias sintéticas, assim como na iteração dos elementos com offsets em [5 ,120], [122,255] e [257,1440).
- O Stream deve ser ***lazy***, com respeito à offsets, isto é, instâncias sintéticas deverão ser criadas por demanda durante uma possível iteração por consumidores do Stream.

Concorrência e Estruturas de dados compartilhadas

6. Assuma que existe uma sequência de objetos armazenados em um array. Será necessário processar todos esses objetos, utilizando operações bloqueantes e independentes, isto é, o processamento do elemento na posição 0 não interfere no elemento associado à posição 1 e assim sucessivamente. Descreva uma estratégia para paralelizar este processamento através da API de Streams, de forma que o tempo decorrido do processamento em paralelo seja o mais próximo possível de *T*/*N*, onde *T corresponde ao* tempo que seria necessário para processar o array sequencialmente *e N* é o número de processadores disponíveis.

7. Considere 2 instâncias de JVM, rotuladas *C* (consumidor) e *P* (produtor), executando simultaneamente em um mesmo servidor. Implementar um mecanismo para que a instância *C* seja capaz de consumir mensagens publicadas por *P* em *N* "filas" (N >=1), utilizando memória compartilhada, observando as seguintes exigências:

- O Consumidor deverá dedicar uma Thread para cada "fila" em períodos *T* fixos.
- O Produtor publicará mensagens nas filas em uma única Thread, de maneira *round-robin*. Considere que as mensagens produzidas por *P* são objetos advindos de um mecanismo de notificação em memória através de um componente pré-existente, isto é, *P* será estimulado por um método na forma

  ```
  <T extends IMessage & Externalizable> void onMessage(T msg){
          ...
  }
  ```

onde

```
public interface IMessage {

   long seq();

}
```

Frameworks

8. Considere as entidades mapeadas a seguir, utilizando framework hibernate.

```
@Entity
@Cache(usage=CacheConcurrencyStrategy.READ_WRITE)
public class User {

  @Id
  Integer id;
```

```
  @OneToMany
  Set<Purchase> purchases;

  @Version
  Integer version;
}

@Entity
public class Purchase {

  @Id
  Integer id;

  String label;

  double value;

}
```

Completar o mapeamento de forma que nenhuma consulta ao banco seja realizada após uma instância de User com determinado id tenha sido carregada uma vez.

9. Considere que a classe a seguir

```
public class DynService {

  @Autowired
  HibernateOperations ops;

  int maxPageSize;
}
```

Descreva como decorar a classe, compilá-la e configurar o arquivo de inicialização do Spring (xml) de modo que o atributo ops seja atribuído através de injeção de dependências quando realizando instanciação programática da classe DynService (new DynService());

Descreva situações em que o uso de singleton beans não são a abordagem correta para obtenção de serviços.

10. Suponha que você está usando o produto ElasticSearch como mecanismo de armazenamento para sua aplicação. O modelo de dados ainda não está estável, e.g., alguns campos podem ser removidos e outros possivelmente criados com certa frequência. Descreva como você configuraria o índice de forma a minimizar o overhead causado pela adição/remoção de campos ao modelo, considerando que apenas uma pequena fração dos campos de fato faça uso de buscas do tipo fulltext.

11. Considere que existe uma funcionalidade bloqueante em determinado sistema que em média leva 3 minutos para completar o processamento de uma informação.
Foi solicitado que essa funcionalidade seja disponibilizada através de uma api REST e é esperado que o número de clientes desta API seja consideravelmente expressivo.

Baseado nessas informações, descrever como você faria para implementar esta API, apontando para vantagens, desvantagens e riscos de sua implementação, e, se aplicável, possíveis formas de mitigá-los.

Por fim, codifique o esqueleto dos Endpoints e elabore um diagrama de sequência com o fluxo principal de sua solução.