Nome do Candidato: jose viera flores

Avaliação conhecimento

Espero que envie o código em JAVA:

1- Escreva um algoritmo em Java para, dado um determinado número inteiro N maior que zero, verificar se N é um número primo.

```
//package com.avaliacao.test;

import java.util.Scanner;
public class Main{
	public static void main(String[] args){
		Scanner scan = new Scanner(System.in);
		System.out.print("Qual e o seu numero");
		int N = scan.nextInt();
		int cont =0;
		for(int i=1; i<=N ; i++){
			if((N%i)==0){
				cont +=1;
				//System.out.print(cont);
			}
		}
		final String msg = cont ==2
			? "O numero é primo"
			: "O numero NÃO e primo";
		System.out.print(msg);
	}
}
```

2- implementação de um método para acessar a tabela de CLIENTE, buscar a lista dos clientes que possuem o tipo VIP e imprimir numa tela.

a) Primeiro criamos a base de dados em mysql

```
create database cliente;
```

b) Criamos nosso projeto spring

```
mysql-connector-java
spring-boot-starter-data-jpa
spring-boot-starter-web
```

c) **Configuramos as propriedades para a conexão com a base de dados**

```
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://${DB_HOST:localhost}:3306/cli
ente
spring.datasource.username=root
spring.datasource.password=2812
spring.jpa.show-sql=true
spring.jpa.hibernate.ddl-auto=update
spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.My
SQL5Dialect
```

d) **Vamos a trabalhar no modelo MVC porem criamos o modelo**

```
import javax.persistence.*;

@Entity
@Table(name = "TavelaCliente")
Public class Cliente{

     @Id
     @GeneratedValue(strategy = Generation.Type.AUTO)
     private Long id;

     private String nome;
     private String telephone;
     private String tipo;


     //Podemos usar Lombok ( ums dos beneficios de spring
eliminaríamos todos istos getters and setters)

     public Long getId(){
          return id;
     }

     public void  setId(Long id){
          this.id =  id;
     }

     public Long getNome(){
          return nome;
     }

     public void  setNome(String nome){
          this.nome =  nome;
     }
```

```
      public Long getTelephone(){
            return telephone;
      }

      public void  setTelephone(String telephone){
            this. telephone =  telephone;
      }

      public Long getTipo(){
            return tipo;
      }

      public void  setTipo(String tipo){
            this.tipo =  tipo;
      }

}
```

e) **Criamos nosso repositorio, conjunto de métodos a ser aplicados**

```
import com.avaliacao.modelo.Cliente;
import org.springframework.data.jpa.repository.JpaRepository;

import java.util.List;

public interface ClienteRepository extends
JpaRepository<Cliente, Long> {
    List<Cliente> findByTipo(String tipo);

}
```

f) **Desenhamos nossos serviços, são os encarregados de se comunicar com a base de dados**

```
import com.avaliacao.modelo.Cliente;
import com.avaliacao.repository.ClienteRepository;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;

import java.util.List;

@Service
public class ClienteService {

    @Autowired
```

```
    private ClienteRepository repository;

    public Cliente saveCliente(Cliente cliente){
        return repository.save(cliente);
    }

    public List<Cliente> saveClientes(List<Cliente> clientes){
        return repository.saveAll(clientes);
    }

    public List<Cliente> getClientes(){
        return repository.findAll();
    }

    public Cliente getClienteById(Long id){
        return repository.findById(id);
    }

    public List<Cliente> getClienteByTipo(String tipo){
        return repository.findByTipo(tipo);
    }

    public String deleteCliente(int id){
        repository.deleteById(id);
        return "cliente removido !! " + id;
    }


}
```

Criamos nosso controller, encarregado de comunicar o cliente com a API

```
import com.avaliacao.modelo.Cliente;
import com.avaliacao.service.ClienteService;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.web.bind.annotation.*;

import java.util.List;

@RestController
public class ClienteController {

    @Autowired
    private ClienteService service;

    @PostMapping("/addCliente")
    public Cliente addCliente(@RequestBody Cliente cliente){
```

```
        return service.saveCliente(Cliente);
    }

    @PostMapping("/addClientes")
    public List<Cliente> addClientes(@RequestBody List<Cliente>
clientes){
        return service.saveClientes(clientes);
    }

    @GetMapping("/clientes")
    public List<Cliente> findAllClientes(){
        return service.getClientes();
    }

    @GetMapping("/clienteById/{id}")
    public Cliente findClienteById(@PathVariable Long id){
        return service.getClienteById(id);
    }

    @GetMapping("/clienteByTipo/{tipo}")
    public List<Cliente> findByTipo(@PathVariable String
tipo){return service.getClienteByTipo(tipo);  }


    @DeleteMapping("/delete/{id}")
    public String deleteCliente(@PathVariable Long id){
        return service.deleteCliente(id);
    }

}
```

Dissertativas:

1- Faça uma descrição breve do que você entende por:

- Spring: e um framework que facilita o desenvolvimento de API, em especial serviços RESTfull, mediante a configuração do ambiente, aplicação de anotações que permite a redução de código e maior legibilidade, e construção de microservicos que tira a construção monolítica para poder realizar serviços escaláveis de fácil manutenção sendo que podemos para um serviço especifico sem deter toda a API. (Exemplo cliente encima)

- JMS:e uma API que permite que duas aplicações se comuniquem entre si a traves de uma lenguajem que pode ser JSON, XML, mediante um sistema de colas, o q quer dizer que cada elemento enviado fica no ultimo da fila esperando seu momento para ser enviada, estos envios são chamados de mensagem. Os concorrentes de esta tecnologia são por exemplo rabbit mq

- Hibernate: e um framework (Object Relational Model) que permite operações com bancos de dados relacionais como se fossem banco de dados orientado a objetos. E dizer, os elementos na base de dados são construídos em base a classes que instanciam os diferentes elementos nas tabelas. A configuração de hibernate e feita a traves do arquivo persistence.xml

- JPA(java persistence api): e simplesmente uma camada que descreve uma interface em java, esta e implementada por hibernate ou outro framework.
https://docs.oracle.com/javaee/7/api/javax/persistence/package-summary.html

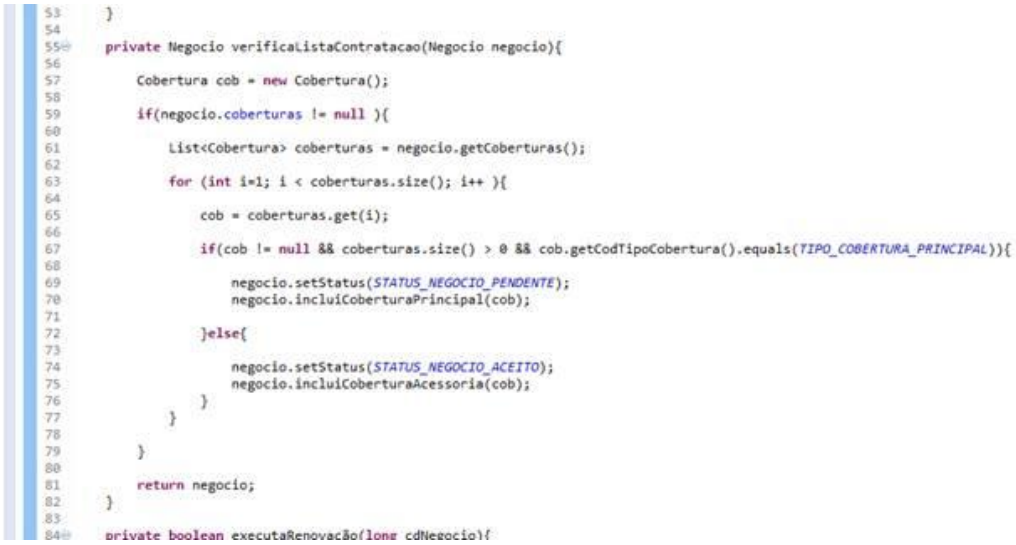
2- Considerado os trechos de código abaixo e o log de erro da aplicação, a qual conclusão podemos chegar?

Na linha 57 se inicializo *Cobertura cob = new Cobertura();* mas não se assinaram valores porem os valores para a variáveis da classe cobertura são nulas, e mostra o null pointer exception. Esta apontando na direção de null.

```
java.lang.NullPointerException
at br.com.tokiomarine.ssv.comuns.emissao.util.ExecutaRenovacao.verificaListaContratacao(ExecutaRenovacao.java:67)
at br.com.tokiomarine.ssv.comuns.emissao.util.ExecutaRenovacao.executaRenovação(ExecutaRenovacao.java:88)
at br.com.tokiomarine.ssv.comuns.emissao.util.ExecutaRenovacao.trataNegocio(ExecutaRenovacao.java:99)
at br.com.tokiomarine.ssv.comuns.emissao.util.ExecutaRenovacao.iniciaProcesso(ExecutaRenovacao.java:95)
at br.com.tokiomarine.seguradora.ssv.prerecepcaoauto.controller.Consumer.transmitirLog(ExecutaRenovacao.java:126)
at br.com.tokiomarine.seguradora.ssv.prerecepcaoauto.controller.Consumer.gravarLogInicioProcessamento(Consumer.java:216)
at br.com.tokiomarine.seguradora.ssv.prerecepcaoauto.controller.Consumer.onMessage(Consumer.java:90)
at sun.reflect.GeneratedMethodAccessor107.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:606)
at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:317)
at org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint(ReflectiveMethodInvocation.java:183)
at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java:150)
at org.springframework.aop.aspectj.MethodInvocationProceedingJoinPoint.proceed(MethodInvocationProceedingJoinPoint.java:
at br.com.tokiomarine.seguradora.ssv.prerecepcaoauto.aspect.CapturaMetrica.executar(CapturaMetrica.java:48)
at sun.reflect.GeneratedMethodAccessor92.invoke(Unknown Source)
at sun.reflect.DelegatingMethodAccessorImpl.invoke(DelegatingMethodAccessorImpl.java:43)
at java.lang.reflect.Method.invoke(Method.java:606)
```

```
public class Negocio {

    Long codigoNegocio;

    String statusNegocio;

    Long codigoCliente;

    List<Cobertura> coberturas;

    Integer qtdParcela;
```

```
public class Cobertura {

    Long codTipoCobertura;

    Double valorContratado;

    Double valorPremio;
```



```
53      }
54
55      private Negocio verificaListaContratacao(Negocio negocio){
56
57          Cobertura cob = new Cobertura();
58
59          if(negocio.coberturas != null ){
60
61              List<Cobertura> coberturas = negocio.getCoberturas();
62
63              for (int i=1; i < coberturas.size(); i++ ){
64
65                  cob = coberturas.get(i);
66
67                  if(cob != null && coberturas.size() > 0 && cob.getCodTipoCobertura().equals(TIPO_COBERTURA_PRINCIPAL)){
68
69                      negocio.setStatus(STATUS_NEGOCIO_PENDENTE);
70                      negocio.incluiCoberturaPrincipal(cob);
71
72                  }else{
73
74                      negocio.setStatus(STATUS_NEGOCIO_ACEITO);
75                      negocio.incluiCoberturaAcessoria(cob);
76                  }
77              }
78
79          }
80
81          return negocio;
82      }
83
84      private boolean executaRenovacao(long cdNegocio){
```

3- O preço de uma mercadoria foi reduzido em 25%. Se quisermos obter novamente o preço original, o novo preço deve ser aumentado de:

a. 20%
b. 25%
c. 33,3%
d. 40%
e. 50%