

Programa oficina

Descrição do programa:

Este será um sistema de check-list de conclusão de manutenção automotiva.

Quando for concluído a manutenção em um veículo, o responsável pela qualidade deverá utilizar esse programa para fazer as verificações de qualidade e segurança antes da liberação do veículo para o cliente.

O programa em si.

O programa deverá rodar em smartphones por meio de instalação.

Ao entrar no programa, o usuário deverá preencher o seu nome e a placa do veículo, (a data será instanciada automaticamente). Após isso, o programa solicitará ao usuário para informar em quais sistemas o veículo sofreu intervenção (a solicitação poderá ser por meio de tickagem).

Após isso, o programa inicia um método de tickagem para o usuário confirmar as verificações para serem feitas no veículo.

Caso o usuário deseje visualizar a lista completa de itens, deverá marcar um campo afirmando estar ciente dos riscos e assumindo a responsabilidade pela verificação “manual”, então o programa mostrará a lista de todos os itens de verificação sem mesmo terem sido tickados.

Estrutura do programa:

Classes:

▼ Program

- Program
 - Logica para iniciar o programa com o nome do usuario, data da inspecão e placa do veículo, e armazenar os dados em uma lista, que será armazenada no sistema, para consultas futuras.

- Logica para dar sequencia no programa com as informações dos sistemas intervidos e armazenar em uma lista, que será armazenada no sistema, para consultas futuras.
- Lógica que mostra ao usuario um o check-list (item por item) que deverá ser inspecionado no veiculo, e marcado como verificado, e o programa armazena em uma lista, que será armazenada no sistema, para consultas futuras.
- Logica que mostra para o usuario uma mensagem que se ele quiser pular para ver todos os itens de check-list sem mesmo ter clicado em verificados, deve concordar com os termos de responsabilidade, e isso também será armazenada no sistema, para consultas futuras.
- Logica para mostrar todos os dados no fim programa.
- logica para manter o usuario no programa, caso queira sair, pressionar o botao sair.

▼ Checkin

- Check-in
 - nome;
 - dataDaInspecao
 - placaDoVeiculo

▼ SistemasIntervidos

- SistemasIntervidos
 - motor
 - suspensao
 - freio
 - cambio
 - arrefecimento
 - injecaoEletronica
 - abs
 - airbag

▼ Verificacoes

- Verificacacoes
 - lista com itens para check

LOGICAS:

- Criar loop para permitir o usuario ficar no sistema, ou saia quando desejar.
- Criar tratamento de exceções e formatação padrão para cada campo de entrada de informação digitada..
- Checklist adaptado de acordo com cada tipo de intervenção realizada no veiculo.

▼ Lista de itens CheckList:

▼ Lista:

- Nivel do oleo do motor
- nivel do liquido de arrefecimento
- nivel do fluido da direção hidraulica
- nivel do fluido de freio
- check-up no cofre do motor
- aperto das rodas
- aperto do bujão do carter
- aperto do filtro de oleo
- lampadas

▼ Primeira versão pronta do projeto:

! Todo o código foi escrito no programa principal, sem utilização de orientação a objetos, herança e polimorfismo.

▼ Código da primeira versão pronta:

```
package application;

import entites.Checkin;
import entites.SistemasIntervidos;

import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

public class Program {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Deseja iniciar o programa? 1 SIM / 2 NAO");

        int a = sc.nextInt();

        System.out.println("\nPara sair, digite 0 a qualquer momento.");
        if (a == 1) {
            int x = 1;
            List<Checkin> listCheckin = new ArrayList<>(1);
            for (int i = 0; i < x; i++) {
                System.out.println("\nPREENCHA AS INFORMACOES: ");
                System.out.println("\nNome do tecnico responsavel: ");
                sc.nextLine();
                String nome = sc.nextLine();
                System.out.println("Placa do veiculo a ser inspecionado:
```

```

    ");
    String placa = sc.nextLine();
    LocalDateTime dataDaInspecao = LocalDateTime.now();
    listCheckin.add(new Checkin(nome, dataDaInspecao, placa));
}
for (Checkin checkin : listCheckin) {
    System.out.println(listCheckin);
}
List<String> listaSistemas = new ArrayList<>();

System.out.println("\nConfirme os sistemas que sofreram
intervencao no veiculo com S/N: ");

System.out.println("\nMotor: S/N ");
if (sc.next().equalsIgnoreCase("S")) {
    listaSistemas.add("motor");
}
System.out.println("Suspensao: S/N ");
if (sc.next().equalsIgnoreCase("S")) {
    listaSistemas.add("suspencao");
}
System.out.println("Freio: S/N ");
if (sc.next().equalsIgnoreCase("S")) {
    listaSistemas.add("freio");
}
System.out.println("Cambio: S/N ");
if (sc.next().equalsIgnoreCase("S")) {
    listaSistemas.add("cambio");
}
System.out.println("Arrefecimento: S/N ");
if (sc.next().equalsIgnoreCase("S")) {
    listaSistemas.add("arrefecimento");
}
System.out.println("Injecao Eletronica: S/N ");
if (sc.next().equalsIgnoreCase("S")) {
    listaSistemas.add("injecao Eletronica");
}

```

```

        System.out.println("ABS: S/N ");
        if (sc.next().equalsIgnoreCase("S")) {
            listaSistemas.add("ABS");
        }
        System.out.println("Airbag: S/N ");
        if (sc.next().equalsIgnoreCase("S")) {
            listaSistemas.add("airbag");
        }

List<String> listaCheckup = new ArrayList<>();

System.out.println("\n====CHECK-UP DE VEICULO====");

System.out.println("\nFaca o check-up nos itens a seguir e
marque S para OK e N para sair: ");

System.out.println("\nNivel do oleo do motor: S/N ");
if (sc.next().equalsIgnoreCase("S")) {
    listaCheckup.add("Nivel do oleo do motor OK");
}

System.out.println("nivel do liquido de arrefecimento: S/N
");
if (sc.next().equalsIgnoreCase("S")) {
    listaCheckup.add("nivel do liquido de arrefecimento O
K");
}

System.out.println("nivel do fluido da direcao hidraulica: S/
N ");
if (sc.next().equalsIgnoreCase("S")) {
    listaCheckup.add("nivel do fluido da direcao hidraulica
OK");
}

System.out.println("nivel do fluido de freio: S/N ");
if (sc.next().equalsIgnoreCase("S")) {

```

```

        listaCheckup.add("nivel do fluido de freio OK");
    }

    System.out.println("check-up no cofre do motor ");
    if (sc.next().equalsIgnoreCase("S")) {
        listaCheckup.add("check-up no cofre do motor OK");
    }

    System.out.println("aperto das rodas: S/N ");
    if (sc.next().equalsIgnoreCase("S")) {
        listaCheckup.add("aperto das rodas OK");
    }

    System.out.println("aperto do bujao do carter: S/N ");
    if (sc.next().equalsIgnoreCase("S")) {
        listaCheckup.add("aperto do bujao do carter OK");
    }

    System.out.println("aperto do filtro de oleo: S/N ");
    if (sc.next().equalsIgnoreCase("S")) {
        listaCheckup.add("aperto do filtro de oleo OK");
    }

    System.out.println("lampadas: S/N ");
    if (sc.next().equalsIgnoreCase("S")) {
        listaCheckup.add("lampadas OK");
    }

    System.out.println("\n====RESUMO DO CHECK-UP: ====");
    System.out.println("\nSistemas que sofreram intervencao:
" + listaSistemas);
    System.out.println("\nItens verificados no check-up: " + listaCheckup);

} else {
    System.out.println("Fora do sistema!");
}

```

```
    sc.close();
}

}
```

▼ **Link repository-Git:**

<https://github.com/Varga-tech-Java/Checkout>

▼ **Segunda versão do projeto:**



- *Criar uma nova branch no Git.*
- *Reorganizar os pacotes, adicionar as informações nas classes, modificar a estrutura do programa, criar classe mãe Check-list e subclasses dos sistemas intervidos.*

▼ Classes:

▼ Super classe:

- ChecklistPadrao

▼ Subclasses:

- motor
- suspensao
- freio
- cambio
- arrefecimento
- injecaoEletronica
- abs

- airbag

▼ FLUXOGRAMA CLASSES

```
graph TD
    A[Classes] --> B[CLASSE MÃE]
    B --> C[CheckListPadrao]
    C --> E[SUBCLASSES]
    E --> F[Suspensao]
    E --> G[Freio]
    E --> H[Cambio]
    E --> I[Arrefecimento]
    E --> J[InjecaoEletronica]
    E --> K[Abs]
    E --> L[Airbag]
```

▼ Descrição das Etapas do Fluxograma:

1. Entendimento do Problema:

- Leia com atenção e identifique o que o problema pede.

2. Entradas e Saídas:

- Determine os dados necessários e os resultados esperados.

3. Testar com Exemplos Simples:

- Use alguns valores fictícios para entender o problema na prática.

4. Planejamento:

- Crie um plano com etapas lógicas claras.

5. Divisão em Partes Menores:

- Separe o problema nas etapas: entrada, processamento e saída.

6. Escrever Pseudocódigo:

- Detalhe os passos como um rascunho.

7. Implementação e Testes Iniciais:

- Codifique pequenas partes e teste cada uma para evitar erros acumulados.

8. Correções e Ajustes:

- Analise os erros e ajuste a lógica ou a implementação.

9. Refinamento:

- Revise a eficiência do código e deixe-o mais claro.

10. Conclusão:

- Resolva o problema e registre o aprendizado adquirido!

Estudos para melhorar o programa:

▼ Git

- "o que são branches no git"
- "git workflow para iniciantes"
- "como criar uma nova branch no git e github"

▼ Convenção de Pacotes Java (Java Package Naming Convention):

- "convenção de nomes de pacotes em java"
- "como organizar pacotes em um projeto java"
- "java project structure best practices"

▼ Injeção de Dependência (conceito inicial)

▼ Composição vs. Herança

- "herança vs composição java quando usar"
- "composition over inheritance java"

▼ Polimorfismo

- "o que é polimorfismo em java"
- "exemplo prático de polimorfismo em java"
- "java instanciar objeto de acordo com variável"

▼ Composição sobre Herança

▼ Herança

- "herança em java extends"
- "java superclass e subclass exemplo"
- "reutilização de código com herança em java"

▼ Método `toString()`

- "java override tostring exemplo"
- "como formatar a saída de um objeto no system.out.println"
- "sobrescrever método `toString` na classe"

▼ Arquitetura em Camadas (Layered Architecture):

- "arquitetura em 3 camadas java"
- "model view controller mvc java simples" (MVC é um padrão um pouco mais avançado, mas a ideia de separar responsabilidades é a mesma).

▼ Princípio da Responsabilidade Única (SRP)

▼ POJO (Plain Old Java Object)

▼ Separação de Preocupações

▼ Classes de Serviço

▼ Iteração sobre Coleções (Looping through Collections):

- "java for each loop com arraylist"
- "como percorrer uma lista em java"
- "refatorar código repetido com loop em java" (A palavra **refatorar** significa melhorar o código sem alterar seu comportamento externo. É exatamente o que você quer fazer).