**Exercício 1:** Explique os conceitos de Abstração, Encapsulamento, Herança e Polimorfismo em suas próprias palavras e dê exemplos de como cada conceito pode ser aplicado em um sistema de gerenciamento de biblioteca (por exemplo, classes como Livro, Usuário, Bibliotecário etc.).

R=

Abstração🡪 é a prática de expor apenas os detalhes essenciais de uma classe, escondendo a complexidade.

Exemplo: Em vez de mostrar todos os detalhes internos de como os dados do livro são armazenados e manipulados, a classe Livro pode expor métodos simples como emprestar(), devolver().

Encapsulamento🡪 refere-se à prática de esconder os detalhes internos de um objeto, expondo apenas o que é necessário através de métodos.

Exemplo: na classe Livro, os atributos como título, autor, e disponível podem ser encapsulados para que apenas métodos internos da classe possam alterá-los diretamente.

Herança🡪 A capacidade de uma classe derivada herdar características (métodos e propriedades) de sua classe base.

Exemplo: Em um sistema de biblioteca, a classe base Usuário pode ter classes derivadas como Bibliotecário e Leitor, que herdam características comuns e adicionam funcionalidades específicas.

Polimorfismo🡪 permite tratar objetos de diferentes classes derivadas de uma mesma classe base de forma uniforme, utilizando métodos sobrescritos. A capacidade de uma classe se comportar de várias maneiras com base no contexto em que é usada.  
Exemplo: Na classe Usuário, tanto o Bibliotecário quanto o Leitor têm o método mostrar Detalhes, mas cada classe pode implementá-lo de maneira diferente.

**Exercício 2:** Quais são as principais diferenças entre a Programação Orientada a Objetos (POO) e a Programação Funcional? Cite vantagens e desvantagens de cada abordagem.

R=

POO tem um estrutura baseada em objetos e classes, tem como vantagens a modularidade abstração e manutenção, e de desvantagens a complexidade e o desempenho.

Programação Funcional tem com sua estrutura baseada em funções puras, tem como vantagem a concorrência, depuração e previsibilidade, e de desvantagens a curva de aprendizado e o desempenho.

Exercício 3 - Criando uma Classe em Dart: Crie uma classe chamada Pessoa em Dart com as seguintes propriedades: nome (String), idade (int) e endereço (String). Adicione um método chamado exibirDetalhes() que imprime os detalhes da pessoa no console.  
    Instruções: Defina a classe Pessoa e crie um objeto dessa classe em sua função principal. Chame o método exibirDetalhes() para exibir as informações da pessoa no console.

R=

void main() {  
   
  Pessoa novaPessoa = Pessoa('Maria', 20, 'rua X');  
  novaPessoa.exibirDetalhes();  
}  
  
class Pessoa {  
  String nome;  
  int idade;  
  String endereco;

  Pessoa(this.nome, this.idade, this.endereco);  
   
  void exibirDetalhes(){  
    print('Nome: $nome');  
    print('Idade: $idade');  
    print('Endereço: $endereco');  
  }  
}

Exercício 4 - Herança e Polimorfismo: Tarefa: Crie uma classe Animal com um método abstrato fazerSom(). Em seguida, crie duas subclasses, Cachorro e Gato, que estendem Animal e implementam o método fazerSom(), fazendo com que cada uma emita um som diferente (por exemplo, "Au Au" para Cachorro e "Miau" para Gato).  
    Instruções: Implemente as classes e crie uma lista de Animal que contenha objetos de Cachorro e Gato. Percorra a lista chamando o método fazerSom() para cada animal.

R=

void main() {

Animal cachorro = Cachorro();

Animal gato = Gato();

List<Animal> animais = [cachorro, gato]

for (var animal in animais) {

animal.fazerSom();

}

}

abstract class Animal {

void fazerSom();

}

class Cachorro extends Animal {

@override

void fazerSom() {

print('Au Au Au Au');

}

}

class Gato extends Animal {

@override

void fazerSom() {

print('Meau Meau');

}

}

class Boi extends Animal {

@override

void fazerSom() {

print('Muuuuuuuuuu');

}

}

Exercício 5 - Encapsulamento e Manipulação de Dados: Crie uma classe ContaBancaria com as propriedades saldo (double) e numeroConta (String). Implemente métodos para depositar() e sacar() dinheiro. Adicione uma verificação no método sacar() para garantir que o saque não seja maior do que o saldo disponível.  
    Instruções: Implemente a classe e crie um objeto de ContaBancaria. Realize algumas operações de depósito e saque, e exiba o saldo final.

R=   
void main() {

ContaBancaria conta = ContaBancaria('18572', 2360.0);

print('Saldo inicial: R\$${conta.saldo}');

conta.depositar(423.0);

conta.sacar(22.0);

conta.sacar(1500.0);

conta.depositar(420.0);

print('Saldo final: R\$${conta.saldo}');

}

class ContaBancaria {

double \_saldo;

String \_numeroConta;

ContaBancaria(this.\_numeroConta, this.\_saldo);

double get saldo => \_saldo;

String get numeroConta => \_numeroConta;

void depositar(double valor) {

\_saldo += valor;

print('Depósito de \$${valor} realizado com sucesso.');

}

void sacar(double valor) {

if (valor <= \_saldo) {

\_saldo -= valor;

print('Saque de R\$${valor} realizado com sucesso.');

} else {

print('Saldo insuficiente para saque.');

}

}

}