1) [Deitel-Cap. 10] Utilizando interfaces (Java) você pode especificar comportamentos semelhantes para classes possivelmente não relacionadas ou díspares. Há uma preocupação atual com as pegadas de carbono (carbono *footprints*, emissões anuais de gás carbônico na atmosfera) a partir de instalações que queimam vários tipos de combustíveis para aquecimento, veículos que queimam combustíveis para se mover, e assim por diante. Nesse cenário:

a) Crie três pequenas classes Building, Car, e Bicycle.
De a cada classe alguns atributos e comportamentos (métodos) únicos que ela não tem em comum com as outras classes.
Sugestões:
i. Building: número de pessoas (int), uso de energia renovável (boolean), número de lâmpadas (int), uso de ar-condicionado (boolean).
ii. Car: combustível (string), cilindrada (float).

b) Escreva uma interface CarbonFootprint com um método getCarbonFootprint. Faça cada uma das suas classes implementar essa interface, para que seu método getCarbonFootprint
calcule uma pegada de carbono apropriada a cada classe (usando os atributos sugeridos ou
outros).

c) Escreva um programa que crie 2 objetos de cada uma das três classes. Crie um Array do tipo CarbonFootprint e insira as referências dos objetos instanciados nessa coleção. Finalmente, itere pela coleção, chamando o método getCarbonFootprintde cada objeto.

2) Modifique o código do exercício 1, tornando Building uma classe abstrata, e implementando duas novas subclasses concretas House e School.

a) O aplicativo que cria a coleção de objetos vai continuar funcionando após a modificação na estrutura das classes?

b) Modifique o aplicativo para que passe a instanciar diretamente objetos House e School, incluindo-os na coleção.

3) Na Java Collection Framework, existem dois tipos abstratos de dados para verificar se um objeto é maior, igual ou inferior a um outro objeto: java.lang.Comparable e java.util.Comparator. Comparable é uma interface que deve ser implementada pela própria classe do objeto que se deseja comparar. Nesse caso, apenas uma ordem para esses objetos existem. Às vezes, se deseja implementar ordenações diferentes para a mesma classe. Neste caso você pode implementar para cada ordenação a interface Comparator em uma classe separada.

- Comparable define o método int compareTo (Object obj) que retorna 0 se o objeto corrente for igual ao objeto obj, um valor negativo se obj é menor ou um valor positivo se obj for maior que o objeto para a qual este método foi invocado. A classe String implementa comparável, por exemplo
- Comparator define o método int compareTo (Object o1, Object o2). Ele retorna 0 se o1 é igual a o2, um valor negativo se o1 for menor que o2, pe um valor positivo se o1 é maior que o2

Um professor poderia desejar que os alunos de uma turma fossem ordenados pela suas notas. Esse será o objetivo deste exercício. Para tanto, siga as seguintes etapas.

I. Primeiramente, crie a classe Aluno. Cada Aluno terá um nome (String) e uma nota (double).

II. Crie a classe Disciplina que guarde os Alunos matriculados em uma turma num atributo array interno. Crie também os métodos de inserção e remoção de alunos nesta turma e o método para exibir os alunos naquela turma na ordem em que se encontram no array.

III. Em um método main de uma classe qualquer, crie um programa que leia nomes de alunos e notas do console (teclado). Para cada par nome e nota, crie um objeto Aluno e o insira no objeto Disciplina (deve ser criado apenas um objeto Disciplina).

IV. Crie o método ordena na classe Disciplina. Esse método deve apenas chamar o método Arrays.sort (Object[] array) para ordenar o atributo array de alunos.

V. Chame o método ordena do objeto Disciplina no método main e após chame o método que exibe os alunos daquela disciplina. Observe o resultado.

VI. Faça a classe Aluno implementar a interface Comparable e implementar o método int compareTo (Object obj). Considere que um aluno está a frente de outro se sua nota é maior.

VII. Execute novamente o método main e observe a ordem em que serão exibidos os alunos