Exercícios

1 – Com a utilização de um HashMap, crie uma solução que represente uma agenda telefônica. Sua solução deve conter:

- Uma classe que represente a agenda
- Uma classe que represente o armazenamento dos dados da agenda
- Uma classe para testes da solução

Agora você deve criar uma outra implementação da classe de armazenamento dos dados da agenda, faça uma implementação que utilize um ArrayList.

- Qual o impacto na mudança da implementação?
- Qual das duas estruturas atendem melhor o problema?

2 - Crie um código que insira 30 mil números numa ArrayList e pesquise-os. Vamos usar um método de System para cronometrar o tempo gasto:

```
    public static void main(String[] args) {
        System.out.println("Iniciando...");
        Collection<Integer> teste = new ArrayList<>();
        long inicio = System.currentTimeMillis();

        int total = 30000;

        for (int i = 0; i < total; i++) {
            teste.add(i);
        }

        for (int i = 0; i < total; i++) {
            teste.contains(i);
        }

        long fim = System.currentTimeMillis();
        long tempo = fim - inicio;
        System.out.println("Tempo gasto: " + tempo);
    }
}
```

Troque a ArrayList por um HashSet e verique o tempo que vai demorar:

```
Collection<Integer> teste = new HashSet<>();
```

O que é lento? A inserção de 30 mil elementos ou as 30 mil buscas? Descubra computando o tempo gasto em cada for separadamente. A diferença é mais que gritante. Se você passar de 30 mil para um número maior, como 50 ou 100 mil, verá que isso inviabiliza por completo o uso de uma List, no caso em que queremos utilizá-la essencialmente em pesquisas.

3 - Repare que, se você declarar a coleção e der new assim:

```
Collection<Integer> teste = new ArrayList<>();
```

em vez de:

```
ArrayList<Integer> teste = new ArrayList<>();
```

É garantido que vai ter de alterar só essa linha para substituir a implementação por HashSet. Estamos aqui usando o polimorfismo para nos proteger que mudanças de implementação venham nos obrigar a alterar muito código. Mais uma vez: programe voltado a interface, e não à implementação!

Esse é um excelente exemplo de bom uso de interfaces, afinal, de que importa como a coleção funciona? O que queremos é uma coleção qualquer, isso é suciente para os nossos propósitos! Nosso código está com baixo acoplamento em relação a estrutura de dados utilizada: podemos trocá-la facilmente.

Esse é um código extremamente elegante e extensível. Com o tempo você vai reparar que as pessoas tentam programar sempre se referindo a essas interfaces menos específicas, na medida do possível: métodos costumam receber e devolver Collections, Lists e Sets em vez de referenciar diretamente uma implementação. É o mesmo que ocorre com o uso de InputStream e OutputStream: eles são o suciente, não há porque forçar o uso de algo mais específico.

Obviamente, algumas vezes não conseguimos trabalhar dessa forma e precisamos usar uma interface mais especíca ou mesmo nos referir ao objeto pela sua implementação para poder chamar alguns métodos. Por exemplo, TreeSet tem mais métodos que os definidos em Set, assim como LinkedList em relação à List.

4 - Gere todos os números entre 1 e 1000 e ordene em ordem decrescente utilizando um TreeSet.

5 - Gere todos os números entre 1 e 1000 e ordene em ordem decrescente utilizando um ArrayList.