Lista de exercícios

Varargs, enums, Generics, etc.

1) **Enums** podem ser definidas com parâmetro de tipo? Responda incluindo um código de teste e mensagens de execução/compilação para comprovar sua afirmação de possibilidade ou limitação.
2) Escreva um programa Java para criar um método genérico que receba uma lista de números e retorne a soma de todos os Z^+ (inteiros positivos) que sejam primos.Veja que será preciso usar ? (wildcard) para selecionar o conjunto de números permitidos.
3) Escreva um programa Java para criar um método genérico que recebe dois arrays do mesmo tipo e verifica se eles têm os mesmos elementos na mesma ordem.
4) Incremente o método da questão anterior para que retorne **0**, se e somente se os dois arrays possuem os mesmos objetos (na mesma ordem); **+1**, se e somente se os objetos dos dois arrays são idênticos e estejam na mesma ordem; **-1**, se os dois vetores armazenam objetos diferentes ou desordenados entre si.

5) Escreva um programa Java para criar um método genérico que recebe uma lista de qualquer tipo e um elemento para pesquisa. O método deve retornar o índice da primeira ocorrência do elemento de pesquisa na lista. Retornará -1 se o elemento de não existir na lista genérica. Obs.: considere receber uma List<?> parametrizada de um tipo qualquer, ou seja, um tipo coringa irrestrito.
6) Escreva um programa Java para criar um método genérico que recebe um mapa de qualquer tipo e imprime cada par chave-valor do mesmo. Use tipos parametrizados da API Java.
7) Escreva um método genérico que receba 2 arrays de mesmo tipo, ordene-os usando o método já implementado para ordenação (selection sort, exec1. slides Generics - parte 2). Em seguida, crie um novo vetor que alterne apropriadamente os elementos de ambos os vetores, e retorno uma lista concatenada ordenada. Nesse último passo, não é necessário invocar nova ordenação, apenas realizar uma adição ao vetor destino, em ordem.

8) É possível usar parâmetros de tipo em argumentos para se obter métodos com número variado de argumentos (varargs)? Responda incluindo um código de teste e mensagens de execução/compilação para comprovar sua afirmação de possibilidade ou limitação. O que justifica esta possibilidade ou limitação?

9) Ambos os códigos dos métodos genéricos abaixo são compilados e executados sem erro? Justifique. Considerando que você é um programador Java experiente, qual das versão é mais adequada para a API que esteja desenvolvendo, e por qual motivo?

```
public <T extends Animal> void printCollectionTyped(Collection<T> animals) {
    animals.forEach(animal -> System.out.println(animal.getClass()));
}

public void printCollectionWildcard(Collection<? extends Animal> animals) {
    animals.forEach(animal -> System.out.println(animal.getClass()));
}
```

10) Seguindo a mesma premissa da questão 9, em relação à justificativa de API mais adequada, observe o código abaixo. Aponte se seguem um padrão de codificação adequado e justifique.

```
public <T extends Animal> Collection<T> createNewListAndAdd(T toAdd) {
    List<T> list = new LinkedList<>();
    list.add(toAdd);
    return list;
}
```

11) O código abaixo está correto? Inclua justificativa baseada em teste do código com mensagens de compilação/execução.

```
public Collection<?> createNewListAndAdd(? toAdd)
```

12) Verifique se o código a seguir compila, e inclua a comprovação do seu teste! Em seguida, analise criticamente esta alternativa: "explique o que pode ocorrer com quem chamar esse método"?

```
public Collection<?> addInto(Object toAdd) {
    List<Object> list = new LinkedList<>();
    list.add(toAdd);
    return list;
}
```

13) Verifique as interfaces **Comparator** e **Comparable** da API. Use-as para sortear uma List<Car>. Car é um classe que representa carros e deve ser ordenada pelo atributo valor do carro. Use os métodos **Collection.sort()** para ordenar a lista usando as versões sobrecarregadas de Comparator e Comparable. Responda também, qual dessas interfaces é apropriada se seu código for apenas um 'consumidor' do código da classe dos elementos da lista.