Na implementação das soluções para as questões seguintes **ignore o tratamento de excepções**.

São valorizadas as soluções que tenham **cuidados de eficiência minimizando o uso de reflexão, sempre que possível**.

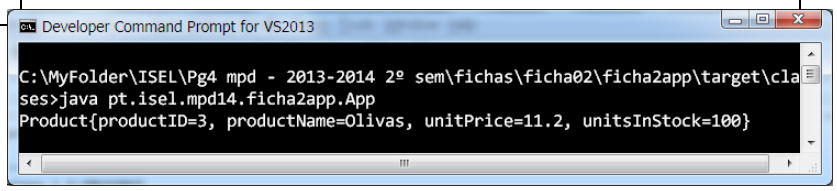
Pode usar expressões lambda e *inner* classes na implementação das soluções, mas **NÃO *inner* classes anónimas**.

1. [4] Dada a interface interface `Predicate<T> {boolean test(T t);}` implemente a classe `ReflectPredicate<T>` que:
   - implementa a interface `Predicate<T>`
   - tem um único construtor com parâmetros `String fieldName` e `Object expected`
   - verifica se o argumento testado tem um campo com o nome `fieldName` de valor igual a `expected`

   Considere o seguinte exemplo de utilização da interface anterior

```
List<Product> prods = new LinkedList<>();
prods.add(new Product(1, "Atum", 5.2, 100));
prods.add(new Product(2, "Sardinha ", 15.2, 10));
prods.add(new Product(3, "Olivas", 11.2, 100));
prods.add(new Product(4, "Batatas", 13.7, 75));
prods.add(new Product(5, "Eliecticos", 9.45, 100));
```

```
prods.stream()
   .filter(new ReflectPredicate<>("unitsInStock", 100))
   .filter(new ReflectPredicate<>("productName", "Olivas"))
   .forEach(System.out::println);
```



2. [2] Implemente o método estático: `<T> Predicate<T> and(Predicate<T> p1, Predicate<T> p2)`, que retorna um novo predicado representando o operador lógico AND entre os dois predicados recebidos por parâmetro (p1 e p2), tal que a execução do código seguinte sobre a mesma lista de produtos tem o mesmo resultado da questão 1.

```
Predicate<Product> pred = and(
  p -> p.unitsInStock == 100,
  new ReflectPredicate<>("productName", "Olivas"));
```

```
prods.stream()
     .filter(pred)
     .forEach(System.out::println);
```

3. [8] Faça uma implementação EAGER do método estático: `<T> ReflectIterable<T> reflect(Iterable<T> itens)`, considerando que `ReflectIterable` obedece à definição da seguinte interface.
   Deve usar a classe `ReflectPredicate` e o método `and` na implementação desta solução.

```
interface ReflectIterable<T> extends Iterable<T>{
    ReflectIterable<T> filter(String fieldName, Object expected);
}
```

   Considere o seguinte exemplo de utilização que tem o mesmo resultado do exemplo da questão 1:

```
reflect(prods)
  .filter("unitsInStock", 100)
  .filter("productName", "Olivas")
  .forEach(System.out::println);
```

4. [6] Faça uma nova implementação do método da alínea anterior mas com comportamento *Lazy*.