Software Development Test

Before inviting you in for interview, please could you complete our short coding test. We find this a really useful, practical tool for understanding how you solve problems as a developer and we use it as the basis for the first-round of face to face interviews. Tests are anonymised and marked independently of a candidate's application, by several members of the hiring team.

Please keep your answers as simple and concise as possible, giving an answer to both questions two and three separately and providing your submission via an e-mail attachment.

Question One

Design and implement a static method `map` that takes two arguments, a "function" *f* and a list *l*, and returns a new list generated from applying *f* to each element of *l*. The original list *l* and its elements should remain unchanged.

Provide example code which uses your `map` implementation and shows how to implement the function `plusOne` which takes a number *n* and returns a new number *n+1*.

If `three` is the list containing the numbers 1, 2 and 3, then:

```
        map(plusOne, three)
```

Will return a new list containing the numbers 2, 3 and 4.

Question Two

You are working on a system, which receives prices from an external component at a frequent rate. The system needs to use the latest price received when required. You are tasked with implementing the component that will provide the latest price for an entity. The external system produces prices for several entities, interleaved into a single sequence of the form:

$$
\begin{array}{lllll}
\texttt{Entity}_A\texttt{:} & p^a{}_1, & p^a{}_2, & & p^a{}_3\ldots \\
\texttt{Entity}_B\texttt{:} & \quad p^b{}_1, & \quad p^b{}_2, & p^b{}_3\ldots &
\end{array}
$$

Time ► ► ►

For example, if during the time taken to process price $p^a{}_1$ the prices $p^a{}_2$, $p^a{}_3$ and $p^a{}_4$ arrive, then the next price the application should process is $p^a{}_4$ and all previous prices should be ignored. Prices for other entities (e.g. $p^b{}_i$) do not affect the latest price for entity a, but are processed independently in the same way with respect to entity b.

Below is the skeleton code for a component that manages prices in this manner. The component needs to be thread-safe (i.e. work without error with concurrent access to update and get prices for the same or different entities, since prices may be updated and accessed by multiple different threads). Complete the code on the following page, implementing the constructor and three methods which are documented.

The following example shows how the component may be used (although single threaded) and the expected output.

```
Example                                            Output
ph.putPrice("a", new BigDecimal(10));
System.out.println(ph.getPrice("a"));              10
ph.putPrice("a", new BigDecimal(12));
System.out.println(ph.hasPriceChanged("a"));       true
ph.putPrice("b", new BigDecimal(2));
ph.putPrice("a", new BigDecimal(11));
System.out.println(ph.getPrice("a"));              11
System.out.println(ph.getPrice("a"));              11
System.out.println(ph.getPrice("b"));              2
```

```java
public final class PriceHolder {

    // TODO Write this bit

    public PriceHolder() {
        // TODO Write this bit
    }

    /** Called when a price 'p' is received for an entity 'e' */
    public void putPrice(String e, BigDecimal p) {
        // TODO Write this bit
    }

    /** Called to get the latest price for entity 'e' */
    public BigDecimal getPrice(String e) {
        // TODO Write this bit
    }

    /**
     * Called to determine if the price for entity 'e' has
     * changed since the last call to getPrice(e).
     */
    public boolean hasPriceChanged(String e) {
        // TODO Write this bit
    }
}
```

Question Three

Extend `PriceHolder` to provide a method that waits for a price change:

```java
/**
 * Returns the next price for entity 'e'. If the price has changed since the last
 * call to getPrice() or waitForNextPrice(), it returns immediately that price.
 * Otherwise it blocks until the next price change for entity 'e'.
 */
public BigDecimal waitForNextPrice(String e) throws InterruptedException;
```

Copy your answer to question two, add this new method and make any necessary modifications to your existing code.