```java
import java.util.HashSet;

/**
 * Simple class to experiment with the Java Collections Framework and how it
 * compares with the OSU CSE collection components.
 *
 * @author Put your name here
 *
 */
public final class JCFExplorations {

    /**
     * Private constructor so this utility class cannot be instantiated.
     */
    private JCFExplorations() {
    }

    /**
     * Raises the salary of all the employees in {@code map} whose name starts
     * with the given {@code initial} by the given {@code raisePercent}.
     *
     * @param map
     *            the name to salary map
     * @param initial
     *            the initial of names of employees to be given a raise
     * @param raisePercent
     *            the raise to be given as a percentage of the current salary
     * @updates map
     * @requires [the salaries in map are positive] and raisePercent > 0
     * @ensures <pre>
     * DOMAIN(map) = DOMAIN(#map)  and
     * [the salaries of the employees in map whose names start with the given
     *  initial have been increased by raisePercent percent (and truncated to
     *  the nearest integer); all other employees have the same salary]
     * </pre>
     */
    public static void giveRaise(components.map.Map<String, Integer> map,
            char initial, int raisePercent) {
        assert map != null : "Violation of: map is not null";
        assert raisePercent > 0 : "Violation of: raisePercent > 0";

        Iterator<Pair<String, Integer>> itr = map.iterator();
        Pair<String, Integer> object = null;

        while (itr.hasNext()) {
            object = itr.next();

            if (object.key().charAt(0) == initial) {
                int newValue = object.value() + (object.value() * raisePercent / 100);
                map.replaceValue(object.key(), newValue);
            }
        }

    }

    /**
     * Raises the salary of all the employees in {@code map} whose name starts
```

```
63      * with the given {@code initial} by the given {@code raisePercent}.
64      *
65      * @param map
66      *            the name to salary map
67      * @param initial
68      *            the initial of names of employees to be given a raise
69      * @param raisePercent
70      *            the raise to be given as a percentage of the current salary
71      * @updates map
72      * @requires <pre>
73      * [the salaries in map are positive] and raisePercent > 0 and
74      * [the dynamic types of map and of all objects reachable from map
75      *  (including any objects returned by operations (such as
76      *  entrySet() and, from there, iterator()), and so on,
77      *  recursively) support all optional operations]
78      * </pre>
79      * @ensures <pre>
80      * DOMAIN(map) = DOMAIN(#map)  and
81      * [the salaries of the employees in map whose names start with the given
82      *  initial have been increased by raisePercent percent (and truncated to
83      *  the nearest integer); all other employees have the same salary]
84      * </pre>
85      */
86     public static void giveRaise(java.util.Map<String, Integer> map,
87             char initial, int raisePercent) {
88         assert map != null : "Violation of: map is not null";
89         assert raisePercent > 0 : "Violation of: raisePercent > 0";
90
91         java.util.Iterator<Entry<String,Integer>> itr = map.entrySet().iterator();
92
93         while (itr.hasNext()) {
94             Entry<String, Integer> object = itr.next();
95
96             if (object.getKey().charAt(0) == initial) {
97                 int newValue = object.getValue()
98                         + (object.getValue() * raisePercent / 100);
99                 map.replace(object.getKey(), newValue);
100            }
101        }
102    }
103
104    /**
105     * Increments by 1 every element in the given {@code Set}.
106     *
107     * @param set
108     *            the set whose elements to increment
109     * @updates set
110     * @ensures <pre>
111     * DOMAIN(map) = DOMAIN(#map)  and
112     * [set is the set of integers that are elements of #set incremented by 1]
113     * </pre>
114     */
115    public static void incrementAll(components.set.Set<NaturalNumber> set) {
116        assert set != null : "Violation of: set is not null";
117
118        NaturalNumber currentNum = null;
119        components.set.Set<NaturalNumber> newSet = set.newInstance();
```

```
120         int setSize = set.size();
121
122         for (int i = 0; i < setSize; i++) {
123             currentNum = set.removeAny();
124             currentNum.increment();
125             newSet.add(currentNum);
126         }
127
128         set.transferFrom(newSet);
129     }
130
131     /**
132      * Increments by 1 every element in the given {@code Set}.
133      *
134      * @param set
135      *            the set whose elements to increment
136      * @updates set
137      * @requires <pre>
138      * [the dynamic types of set and of all objects reachable from set
139      *  (including any objects returned by operations (such as iterator()), and
140      *  so on, recursively) support all optional operations]
141      * </pre>
142      * @ensures <pre>
143      * DOMAIN(map) = DOMAIN(#map)  and
144      * [set is the set of integers that are elements of #set incremented by 1]
145      * </pre>
146      */
147     public static void incrementAll(java.util.Set<NaturalNumber> set) {
148         assert set != null : "Violation of: set is not null";
149
150         Iterator<NaturalNumber> itr = set.iterator();
151         java.util.Set<NaturalNumber> newSet = new HashSet<NaturalNumber>();
152         int setSize = set.size();
153         NaturalNumber currentNum = null;
154
155         for (int i = 0; i < setSize; i++) {
156             currentNum = itr.next();
157             currentNum.increment();
158             newSet.add(currentNum);
159         }
160
161         set = newSet;
162
163     }
164
165 }
166
```