Pregunta 1 (5 puntos)

Se desea definir una clase para poder representar conjuntos de enteros dentro de un rango dado. Dicho rango vendrá indicado por dos números:

- **`min`**: los elementos del conjunto serán todos **`>= min`**
- **`max`**: los elementos del conjunto serán todos **`< max`**

Es decir, representaremos un conjunto de enteros dentro del intervalo **[min, max)**

Los métodos de esta clase, que llamaremos **`RangeSet`**, serán:

- `new RangeSet(int min, int max)`:
  - **tendremos garantizado que `min <= max`**
  - si `min == max` el rango de valores será vacío
  - NOTA: el número máximo de valores en el conjunto es (`max - min`) y en Java no hay ningún problema en crear un array de tamaño 0
- `boolean add(int elem)`
  - intenta añadir `elem` al conjunto
  - si `elem` no está dentro del rango la operación devuelve `false` (y no hace nada más)
  - si `elem` está dentro del rango, la operación añade `elem` al conjunto y devuelve `true`
- `boolean contains(int elem)`
  - devuelve un booleano correspondiente al hecho de si el elemento pertenece al conjunto
  - fijaos en que para enteros fuera del rango, siempre devuelve `false`
- `static RangeSet union(RangeSet rs1, RangeSet rs2)`
  - devuelve el rango correspondiente a la unión de ambos conjuntos
  - el rango del resultado (sus valores `min` y `max`) deberá incluir los rangos de los conjuntos que se unen
  - NOTA: podéis usar `add` y `contains`
- `int size()`
  - devuelve el número de elementos que tiene el conjunto

Ejemplo de uso:

```
RangeSet rs = new RangeSet(21, 31); // Rango de 10 valores posibles {21, 22, ... , 30}
rs.add(25);    // Añade 25 y devuelve true
rs.add(40);    // No puede añadirlo ya que está fuera del rango, y devuelve false
rs.add(28);    // Añade 28 y devuelve true
RangeSet rs2 = new RangeSet(25, 32); // Rango de 7 valores posibles {25, 26, ..., 31}
rs2.add(28);   // añade 28 y devuelve true
rs2.add(30);   // Añade 30 y devuelve true
RangeSet rs3 = RangeSet.union(rs1, rs2);  // El rango resultado tendrá que incluir los
                                          // rangos [21, 31) y [25, 32) por lo que tendrá
                                          // que ser [21, 32) e incluirá los elementos
                                          // {25, 28, 30}
rs3.contains(25);  // Devuelve true
rs3.contains(27);  // Devuelve false (aunque está en el rango no está en el conjunto)
rs3.contains(40);  // Devuelve false (no está en el rango)
rs3.size();        // Devuelve 3
```

Pregunta 2 (5 puntos)

Un Entrenador Pokémon registra en su Pokédex los Pokémon que ha ido cazando/conociendo durante su larga y dura formación. Ash, nuestro entrenador Pokémon predilecto, ha sido atacado por el malvado Team Rocket y, aunque no han podido robarle su valioso Pikachu, le han sustraído su Pokédex. Por suerte, Ash es un entrenador precavido y dispone de una copia de seguridad de su Pokédex y de todos los Pokémon que ha ido cazando escrita en un papel. Debemos ayudar a Ash

con un programa para restablecer su Pokédex; de este modo podrá conocer la cantidad total de Pokémon **diferentes** de cada generación que ha conseguido o conocido hasta la fecha, así como el total de "dorsales" (números) **diferentes** que contiene.

Para simplificar un poco el problema, consideraremos que existen **7 generaciones** de Pokémon (**desde la 1 a la 7**) y, en cada generación, los Pokémon tienen **números de dorsal del 1 al 100**. De esta manera, cada Pokémon puede representarse por una cadena con el formato "Generación#Dorsal". Por ejemplo, la cadena "2#40" representa el Pokémon con número 40 de la segunda generación.

De esta manera, el programa leerá una cadena formada por descripciones de Pokémon como las descritas en el párrafo anterior, que podéis considerar que no contienen errores de formato. Eso sí, puede tener Pokémon repetidos. Para ir contando "cosas diferentes" usaréis instancias de la clase RangeSet del problema anterior.

El programa escribirá, para cada una de las generaciones con al menos un Pokémon, el número de Pokémon diferentes de esa generación que tiene y, finalmente, el número de dorsales diferentes que ha conseguido, teniendo en cuenta todas las generaciones.

Por ejemplo, dada la cadena de Pokémon:

```
2#12 5#6 2#7 2#12 3#4 5#6 3#2 1#12 6#4
```

El programa escribiría:

```
Generación 1: 1
Generación 2: 2
Generación 3: 2
Generación 5: 1
Generación 6: 1
Dorsales diferentes: 5
```

PISTAS:

- para no tener que ir pasando parámetros en todas las funciones, podéis considerar que la variable sobre la que acumuláis el resultado es una variable de instancia de la clase del programa principal
- si no tenéis claro cómo separar la información correspondiente a un Pokémon (la cadena que tiene como formato "Generación#Dorsal"), recordad que podéis suponer que tenéis funciones privadas como:
  - `private static int generation(String pokemon)`
  - `private static int numInGeneration(String pokemon)`

  que podéis usar, incluso sin saberlas programar, para expresar el resto de la solución del problema.

- `class ConsoleProgram`
  - `String readLine(String message)`
  - `void println(String str) / void println(int n)`
  - `void print(String str)  / void print(int n)`
- `class Integer:`
  - `static int parseInt(String str)`
- `class String:`
  - `char charAt(int index)`
  - `int length()`
  - `boolean equals(String other) / boolean equalsIgnoreCase(String other)`
  - `int compareTo(String other)`
  - `int indexOf(char c) / int indexOf(String s)`
  - `String substring(int p1, int p2) / String substring(int p1)`
  - `String concat(String s) / También podéis usar + para concatenar Strings`
  - `String trim()`
  - `static String valueOf(int n)`
- `class StringTokenizer:`
  - `new StringTokenizer(String str)`
  - `new StringTokenizer(String str, String delims)`
  - `new StringTokenizer(String str, String delims, boolean returnDelims)`
  - `boolean hasMoreTokens()`
  - `String nextToken()`
- `class Math`
  - `static int max(int n1, int n2) / static int min(int n1, int n2)`