CONTEXTO

FPickr es una aplicación Web para gestionar álbumes de fotos de usuarios, al estilo de Flickr, Picasa, etc. Dicho sistema permite a los usuarios subir fotos que se organizan en álbumes. Un álbum es una secuencia de fotos. Cada foto puede etiquetarse con palabras ("Sevilla", "Vacaciones", "Playa"…) y recibe visitas que FPickr contabiliza. Las fotos se identifican por su URL, y los usuarios se registran en el sistema mediante un login (o nombre corto de usuario) y una contraseña. El sistema de tipos es el siguiente:

Tipo **Persona,** subtipo de Comparable<Persona>:
Propiedades:
- nombre, de tipo String, consultable y modificable.
- apellidos, de tipo String, consultable y modificable.
- fecha de nacimiento, de tipo LocalDate, consultable y modificable.

Orden natural: por apellidos y a igualdad de apellidos por nombre.
Criterio de Igualdad: por apellidos y nombre.
Representación de como cadena: Los apellidos seguidos del nombre, separados ambos por una coma.
Constructores: uno con un parámetro por cada propiedad básica del tipo. Tenga en cuenta que pueden aparecer espacios entre cada uno de los campos.

Tipo **Foto:**
Propiedades:
- titulo, de tipo String, consultable y modificable.
- url, de tipo String, consultable.
- propietario, de tipo String, consultable. Almacena el login del usuario propietario de la foto.
- etiquetas, de tipo SortedSet<String>, consultable.
- numVisitas, de tipo Integer, consultable.

Criterio de igualdad: por url.
Criterio de orden natural: por url.
Representación como cadena: la url seguida del número de visitas entre paréntesis.
Constructores: un constructor que recibe como parámetros título, url y login del propietario y número de visitas (las etiquetas se inicializan con un conjunto vacío). También un constructor a partir de String, que recibe una cadena con el título, la url, el propietario y el número de visitas separados por comas (las etiquetas se inicializan con un conjunto vacío).
Restricciones:
- El login no puede ser la cadena vacía.
- La url debe comenzar por un identificador de protocolo (cadena de al menos 3 caracteres), seguido de la cadena "://", y a continuación una cadena no vacía. Por ejemplo: "file://mifoto.jpg".
- Número de visitas debe ser mayor o igual que 0.

Otras operaciones:
- *void añadirEtiqueta(String etq)*. Añade la etiqueta etq a la lista de etiquetas asociadas a la foto.
- *void incrementarVisitas()*. Incrementa en una unidad el número de visitas de la foto.

Tipo **Usuario,** subtipo de Persona:
Propiedades:
- login, de tipo String, consultable.
- password, de tipo String, consultable y modificable.
- álbumes, de tipo List<Album>, consultable.
- numVisitas, de tipo Integer, consultable. Es la suma de visitas de todas las fotos de todos los álbumes del usuario.

Orden natural: el mismo de Persona.
Criterio de igualdad: el mismo de Persona.
Representación como cadena: el login, seguido de la representación como cadena de Persona entre paréntesis.
Constructores: uno que recibe los mismos parámetros que el constructor de Persona, y además el login y el password. La lista de álbumes la inicializa vacía.
Restricciones:
- El login no puede ser la cadena vacía.
- El usuario debe tener al menos 18 años.

Otras operaciones:
- *SortedSet<String> todasLasEtiquetas(). Devuelve el conjunto ordenado de todas las etiquetas de todas las fotos del usuario.*

Tipo **Album:**
Propiedades:
- nombre, de tipo String, consultable y modificable.
- fotos, de tipo List<Foto>, consultable.
- numVisitas, de tipo Integer, consultable. Corresponde a la suma de las visitas de todas las fotos del álbum.

Criterio de igualdad: por nombre y por fotos (las mismas y en el mismo orden).
Representación como cadena: el nombre seguido del n° de visitas entre paréntesis y a continuación la lista de fotos.
Constructores: uno que recibe el nombre del álbum y una lista de fotos.
Restricciones:
- La lista de fotos no puede estar vacía.

Otras operaciones:
- *void añadirFoto(Foto f)*
- *SortedSet<String> etiquetasComunes()*
- Foto masVisitas()
- Set<Foto> conEtiqueta(String etq)
- Double porcentajeMenosVisitas(Integer n)

	FUNDAMENTOS DE PROGRAMACIÓN	**Curso: 2015/16**
	EXAMEN DE DICIEMBRE. 9 de diciembre de 2015	**Versión: 1.0.6**

EJERCICIO 1 (1.5 puntos)

Escriba, para la clase `FotoImpl`, el código de los siguientes elementos. Lance excepciones de tipo IllegalArgumentException para informar del incumplimiento de restricciones.

a) La cabecera y los atributos de la clase.
b) El constructor con parámetros.
c) El constructor a partir de String.
d) Los métodos consultore y modificador de la propiedad url.
e) Los métodos `equals` y `compareTo`.

EJERCICIO 2 (1.5 puntos)

Implemente el tipo Usuario mediante la clase UsuarioImpl. Se pide el código completo de la clase, excepto los métodos de consulta y modificación de las nuevas propiedades de Usuario. **No se permite el uso de la clase Stream de Java 8, ni más de un bucle por método.**

EJERCICIO 3 (2 puntos)

Implemente las siguiente operaciones del tipo Album, sin usar la clase Stream de Java 8, y **no usando más de un bucle por método**. Se penalizará el uso de métodos auxiliares innecesarios.

a) **Implemente el método `etiquetasComunes`, que dado un álbum devuelve un conjunto ordenado por el orden natural de las cadenas que contiene las etiquetas comunes a todas las fotos del álbum. Si el álbum no contiene ninguna foto se devuelve un conjunto ordenado vacío.** En este apartado no se permite utilizar ningún método auxiliar.

b) Implemente el método **`masVisitas`**, que devuelve la foto con más número de visitas.
c) Implemente el método `conEtiqueta`, que devuelve el subconjunto de aquellas fotos que contengan la etiqueta pasada como parámetro.
Implemente el método `porcentajeFotosMenosVisitas`, que devuelve el tanto por ciento de fotos del álbum que tengan un número de visitas menor que un valor pasado como parámetro.

EJERCICIO 4 (1.5 puntos)

Se desea implementar una factoría de objetos de tipo Foto. Para ello, en una nueva clase de utilidad `Fotos`, implemente las siguientes propiedades poblacionales:

- Login del usuario con más fotos (null en caso de no existir aún ninguna foto). Para consultar esta propiedad cree el método `getUsuarioMasFotos`. Si aún no existiera ninguna foto, se devolvería null.
- Conjunto de fotos de un usuario determinado. Para consultar esta propiedad cree el método `getFotosUsuario` que recibe el login de un usuario como parámetro. Si no hubiese ningún usuario con tal login se devolverá el conjunto vacío.

En la misma clase `Fotos`, implemente dos métodos creacionales para el tipo `Foto`: un método creacional copia y otro a partir de `String`. Llame a ambos métodos `createFoto`. **NOTA**: **NO utilice bucles ni métodos de la clase Stream de Java8.**

EJERCICIO 5 (2 puntos)

Implemente en la clase de utilidad `Albumes` los siguientes métodos. **Los apartados b), c), d) y e) deben resolverse utilizando exclusivamente la clase Stream de Java 8.**

a) Un método `cuentaVisitasCadaFoto` que recibe un `Album` y devuelve un `Map<Foto, Integer>` en el que las claves son las fotos del álbum y los valores asociados el número de visitas de cada foto. **En este apartado no se permite utilizar la clase Stream de Java 8**.

b) Un método `rankingUsuariosMasVisitas` que dado un conjunto de usuarios y un entero `n` devuelva la lista de los `n` usuarios con mayor número de visitas, ordenados por número de visitas de mayor a menor.
c) Un método `fotoPorURL` que dado un conjunto de fotos devuelva un `Map` que asocie a cada URL de una foto el título de la misma.
d) Un método `comprobarNumeroVisitas` que dado un conjunto de usuarios compruebe si todos aquellos usuarios del conjunto que no tienen ningún álbum, no tienen ninguna visita.
e) Un método `incrementarVisitasUsuario` que dado un usuario incremente en una unidad el número de visitas asociadas a cada una de las fotos que el usuario tiene en sus álbumes. Tenga en cuenta que si una foto aparece en más de un álbum, sólo debe incrementar su número de visitas en una unidad.

EJERCICIO 6 (Lenguaje C) (1.5 puntos)

Dado el tipo Cadena, capaz de almacenar un máximo de 255 caracteres, responda a las siguientes cuestiones:

a) Defina en lenguaje C:
- Un tipo **GrupoEtiquetas**, con las siguientes propiedades:
  - etiquetas, que es un array para almacenar hasta 8 elementos de tipo Cadena.
  - nEtiquetas, de tipo entero que almacena el número real de etiquetas.
- Un tipo **Foto**, con las siguientes propiedades:
  - titulo (sin tilde), de tipo Cadena.
  - url, de tipo Cadena.
  - propietario, de tipo Cadena.
  - etiquetado, de tipo GrupoEtiquetas.
  - numeroVisitas (sin tilde), de tipo entero.
- Un puntero **PGrupoEtiqueta** al tipo GrupoEtiquetas.
- Un puntero **PFoto** al tipo Foto.
- Un array de fotos **ArrayFotos** para almacenar hasta 500 fotos.

NOTA. Para los ejercicios siguientes no se preocupe de usar la función fflush, ni de que queden caracteres en el buffer de entrada o de salida; pero tenga en cuenta que las propiedades de tipo Cadena pueden contener espacios en blanco, por ejemplo: "Tomando el sol".

b) Escriba una función **void leeEtiquetadoTeclado (…)** que devuelva en un parámetro de salida de tipo GrupoEtiquetas los datos de etiquetado tecleados para una foto. Tenga en cuenta lo siguiente:
- Si el número de etiquetas que se desea introducir está fuera del rango del 1 al MAX_ETIQUETAS no se introducirán etiquetas, es decir, en este caso el número de etiquetas será cero. En otro caso las etiquetas se piden de la siguiente forma:

  Etiqueta [1]: <*aquí se tecleará la denominación de la primera etiqueta*>
  Etiqueta [2]: <*aquí se tecleará la denominación de la segunda etiqueta*>
  …

c) Escriba una función **int leeFotosFichero(…)** que recibiendo como parámetro de entrada el nombre de un fichero, devuelva en un parámetro de tipo ArrayFotos los datos de determinado número de fotos que están almacenadas en el fichero. Tenga en cuenta que la propia función devolverá alguna de estas tres posibilidades:
   a. El número de leídas, o
   b. Cero (0) si no se puede abrir el fichero, o
   c. Si el número de fotos del fichero supera la constante MAX_FOTOS, deben leerse justo este número fotos y por tanto devolver el valor de MAX_FOTOS

Suponga ya implementada una función void leeFotoFichero (FILE *, PFoto) que le ayudará a leer los datos completos de cada foto.

Anexo: clases e interfaces de la API de Java

```
public class Collections {
  public static void  <T extends Comparable<? super T>>
    sort ( List<T> l);
  public static void sort( List<T> l, Comparator<? super T> c);
  public static void reverse (List<T> list);
  public static List<T> nCopies(int n, T o); }
```

```
public interface SortedSet<E> extends Set<E> {
  SortedSet<E> headSet(E toElement);
  SortedSet<E> tailSet(E fromElement);
  SortedSet<E> subSet(E fromElement, E toElement);
  E first();
  E last(); }
```

```
public class LocalDate {
  static LocalDate now();
  static LocalDate of(int year, int month, int dayOfMonth);
  static LocalDate parse(String date, DateTimeFormatter f);
  int getYear(); int getMonth(); int getDayOfMonth();
  boolean isBefore(LocalDate d);
}
public class DateTimeFormatter {
  String format(TemporalAccessor temporal)
  static DateTimeFormatter  ofPattern(String pattern); }
```

```
public class String {
  char charAt (int index);
  boolean contains (CharSequence s);
  int indexOf (char c);
  boolean isEmplty();
  int length();
  Boolean startsWith(String s);
  String[] split(String sep);
  String toUpperCase();
}
```

```
public interface Map<K,V> {
  V put(K key, V value);
  V get(Object key);
  V remove(Object key);
  boolean containsKey(Object key);
  boolean containsValue(Object value);
  int size();
  boolean isEmpty();
  Set<K> keySet();
  Collection<V> values(); }
```

```
public interface Comparator<T> {
  int compare(T o1, T o2);
  static <T,U extends Comparable<? super U>> Comparator<T>
    comparing(Function<? super T, ? extends U> keyExtractor);
  static <T extends Comparable<? super T>> Comparator<T>
    naturalOrder();
  default Comparator<T> reversed();
  default Comparator<T> thenComparing(Comparator<? super T>
    other); }
```

```
public class Optional<T> {
  T get();
  boolean isPresent();
  static <T> Optional<T> of(T value);
  T orElse (T other);
  T orElseThrow (Supplier<? extends X>exceptionSupplier); }
```

```
public class Period {
  static Period between(
    LocalDate startDateInclusive, LocalDate endDateExclusive);
  int getDays();
  int getMonths();
  int getYears(); }
```

```
public interface Stream<T> {
  T reduce(T identity, BinaryOperator<T> accumulator);
  Optional<T> reduce(BinaryOperator<T> accumulator);
  <U> U reduce(U identity, BiFunction<U,? super T,U>
    accumulator, BinaryOperator<U> combiner);
  <R> R collect(Supplier<R> resultFactory, BiConsumer<R,?
    super T> accumulator, BiConsumer<R,R> combiner);
  <R,A> R collect(Collector<? super T,A,R> collector);
  boolean allMatch(Predicate<? super T> predicate);
  boolean anyMatch(Predicate<? super T> predicate);
  boolean noneMatch(Predicate<? super T> predicate);
  Optional<T> min(Comparator<? super T> comparator);
  Optional<T> max(Comparator<? super T> comparator);
  long count();
  Optional<T> findFirst();
  Optional<T> findAny();
  void forEach(Consumer<? super T> action);
  Stream<T> filter(Predicate<? super T> predicate);
  DoubleStream mapToDouble(ToDoubleFunction<? super T>
    mapper);
  IntStream mapToInt(ToIntFunction<? super T> mapper);
  LongStream mapToLong(ToLongFunction<? super T> mapper);
  <R> Stream<R> map(Function<? super T,? extends R> mapper);
  IntStream mapToInt(ToIntFunction<? super T> mapper);
  DoubleStream mapToDouble(ToDoubleFunction<? super T>
    mapper);
  <R> Stream<R> flatMap(Function<? super T, ? extends Stream<?
    extends R>> mapper);
  Stream<T> distinct();
  Stream<T> sorted();
  Stream<T> sorted(Comparator<? super T> comparator);
}
```

```
public class Collectors {
  public static <T> Collector<T,?,Set<T>> toSet();
  public static <T> Collector<T,?,List<T>> toList();
  public static Collector< <T,C extends Collection<T>>> toCollection(Supplier<C> collectionFactory);
  static<T,K,U>Collector<T,?Map<K,U>>toMap(Function<?super T,?extends K>keyMapper,Function<?super T,?extends U>valueMapper);
  static<T,K,U>Collector<T,?Map<K,U>>toMap(Function<?super T,?extends K>keyMapper,Function<?super T,?extends U>valueMapper,
    BinaryOperator<U> mergeFunction)
  static<T,K,U,M extends Map<K,U>>Collector<T,?,M>toMap(Function<?super T,?extends K>keyMapper,Function<?super T,?extends U>
    valueMapper, BinaryOperator<U> mergeFunction, Supplier<M> mapSupplier);
  static<T,K>Collector<T,?,Map<K,List<T>>>groupingBy(Function<?super T,?extends K>classifier);
  static<T,K,A,D>Collector<T,?,Map<K,D>>groupingBy(Function<?super T,?extends K>classifier,Collector<? super T,A,D>downstream)
}
```

```
public interface DoubleStream {
  OptionalDouble average();
  double sum(); }
```

```
public interface IntStream {
  OptionalDouble average();
  int sum(); }
```

```
public interface LongStream {
  OptionalDouble average();
  Long sum(); }
```