Trabajo Práctico VII: Colecciones

La clase Object
El método equals
El método hashcode
Colecciones
Tipos de colecciones: HashSet, LinkedHashSet, TreeSet, ArrayList, LinkedList, PriorityQueue.
Iterators
Excepciones comunes
Tipos genéricos
Restricción de tipos genéricos
Instanciando genericidad en una subclase
Colecciones genéricas
Herramientas para programación genérica

Objetivos de la práctica

Sobreescribir los métodos equals, hashCode de acuerdo a los requerimientos del caso.
Utilizar diferentes tipos de colecciones y comprender cual es la más indicada en cada caso.
Definir clases e interfaces con genericidad.
Definir clases con genéricos restringidos.
Extender una clase genérica haciéndola específica (de un determinado tipo de dato concreto).

Ejercicio 1

La clase Persona tiene los atributos, apellido, nombre, dirección.

Inciso a)

Escribe la clase Persona con sus getters y setters correspondientes, reescribiendo los métodos equals y hashCode utilizando como criterio los atributos nombre y apellido.

Ademas, la clase Persona implementa la interfaz Comparable, utilizando como criterio de orden el apellido, y en caso de ser iguales, el nombre.

Sobreescribe convenientemente el método toString

Inciso b)

Escribe la clase Ordenadora que tendrá el método estático ordenar:

```
public static void ordenar(Comparable[] array)
```

Inciso c)

Escribe la clase Prueba, que en su método main cree un array de 5 personas con los siguientes atributos:

	apellido	nombre	dirección
persona1	“Perez”	“Carlos”	“Colon 3212”
persona2	“Perez”	“Carlos”	“Colon 3212”
persona3	“García”	“Ana”	“Mitre 2812”
persona4	“Alvarez”	“Valeria”	“San Luis 2812”
persona5	“Garcia”	“Luis”	“Matheu 3538”

Luego compara las personas 1 y 2 entre ellas, utilizando el operado == y el método equals, para verificar que lo resultados sean los esperados.

Muestra en forma secuencial las 5 personas del array, luego ordena el array utilizando la clase Ordenadora y muestra el resultado.

Inciso d)

En la clase Prueba, agrega las 5 instancias de persona en colecciones de los siguientes tipos: HashSet, LinkedHashSet, TreeSet, ArrayList, LinkedList y PriorityQueue.

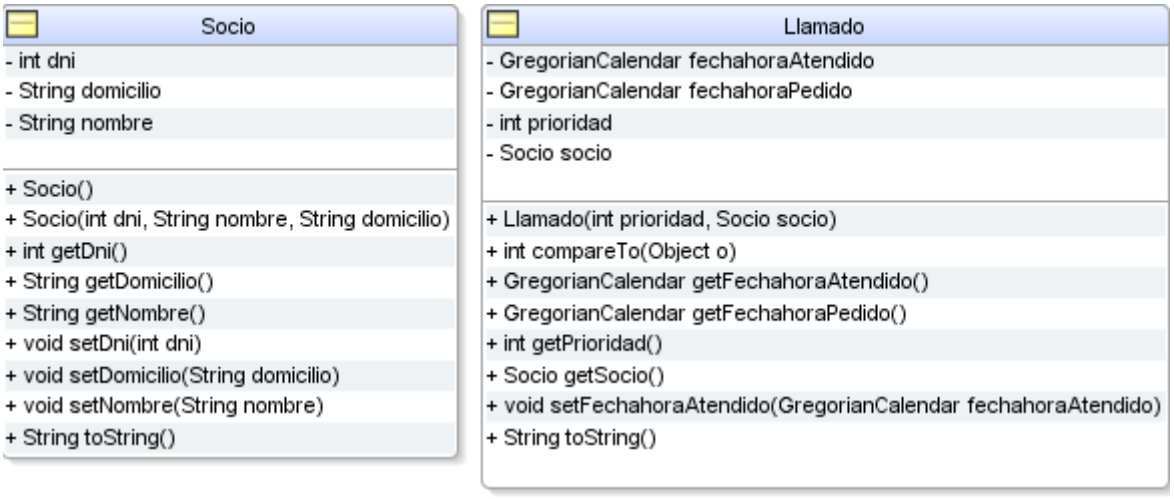
Luego utiliza un Iterator para mostrar el contenido de cada colección y analiza las diferencias entre cada uno. Agrega las instancias sin ordenarlas (En el caso de la PriorityQueue, en lugar de usar un Iterator utiliza el método poll)

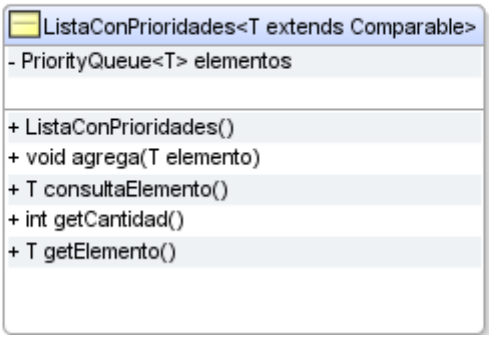
Ejercicio 2

Una empresa de emergencias médicas tiene un servicio de ambulancias a domicilio.

Para ello, es importante atender los diferentes llamados que se reciben y dar atención a los mismos de acuerdo a su orden de prioridad.

Se tienen las clases: **Socio, Llamado, ListaConPrioridades** :





- La clase **Socio** simplemente tiene atributos, geters, seters y constructores.
- La clase **Llamado** tiene un único constructor que pide como parámetros: el socio que realizo el llamado, y un entero que nos da la prioridad (cuanto mayor sea el número, más urgente es el llamado). Al llamar al constructor, el atributo fechahoraPedido, toma los valores de fecha y hora del sistema, el atributo fechahoraAtendido toma valor null. Esta clase implementa la interfaz **Comparable,** el criterio de comparación será el siguiente, un llamado será menor que otro (se ordenará primero) cuanto mayor sea prioridad, en caso de que la prioridad sea la misma, entonces se tomará el orden dado por el atributo fechahoraPedido (la clase GregorianCalendar propia de Java implementa la interfaz Comparable).
- La clase ListaConPrioridades es de tipo genérico <T extends Comparable>. El método getElemento nos devuelve el elemento con mayor prioridad, y lo remueve del atributo elementos (PriorityQueue). El método consultaElemento() nos devuelve el elemento con mayor prioridad pero mantiene intacta la PriorityQueue. Deberá tener un método que nos permita conocer la cantidad de elementos que contiene.

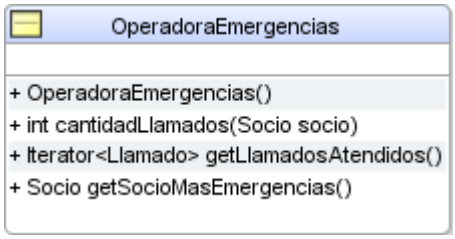
Inciso a)

Implementa las clases anteriormente descriptas, y realiza una aplicación visual sencilla que nos permita gestionar llamados de emergencia, es decir, que nos permita agregas llamados y luego recuperar el llamado más urgente. Recuerda informar visualmente los datos que creas relevantes.

Inciso b)

Escribe la clase OperadoraEmergencias que se extiende de ListaConPrioridades <Llamado>.

Esta clase agrega los métodos:



- El método cantidadLlamados, devuelve la cantidad de llamados que realizó el socio pasado como parámetro.
- El método getLlamadosAtendidos, nos devuelve un Iterator con los llamados que han sido atendidos en ese orden.
- El método getSocioMasEmergencias retornará el socio al que se le han atendido mas llamados.

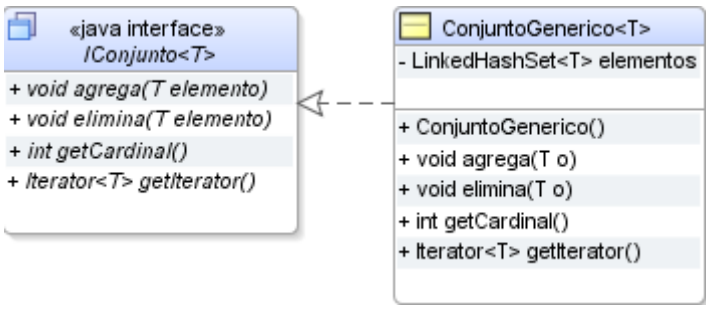
(Considera la necesidad de implementar mas tipos de colecciones para poder satisfacer estos nuevos requerimientos)

Además sobreescribe el método getElemento(), de manera que además de recuperar el llamado mas urgente, le setea el atributo fechaHoraAtendido con la fecha y hora del sistema. ¿Por qué motivo no podíamos realizar esta operación en la clase ListaConPrioridades?

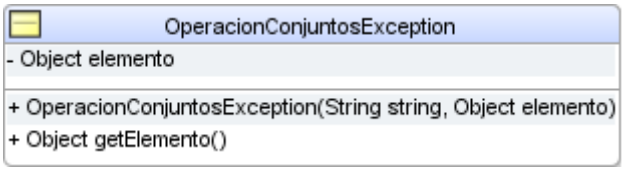
Escribe también una aplicación visual sencilla que nos permita utilizar los nuevos servicio de la clase OperadoraEmergencias.

Ejercicio 3

Escribe la interfaz IConjunto de tipo genérico <T> y la clase ConjuntoGenerico (también de tipo genérico) que implementa dicha interfaz.



En caso de pretender agregar un elemento existente o eliminar un elemento inexistente se lanzara una excepción de tipo OperacionConjuntoInvalidaException (no se visualiza en el diagrama de UML), en cuyo constructor se pasará por parámetro un String indicando el motivo de la Excepción, y el elemento que provocó la excepción:

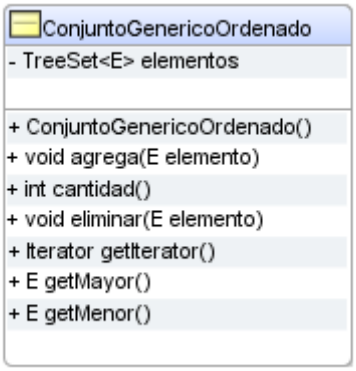


Inciso a)

Realiza una aplicación visual sencilla que nos permita agregar Personas (descriptas en el ej 1) en un ConjuntoGenerico y consultar los elementos del mismo.

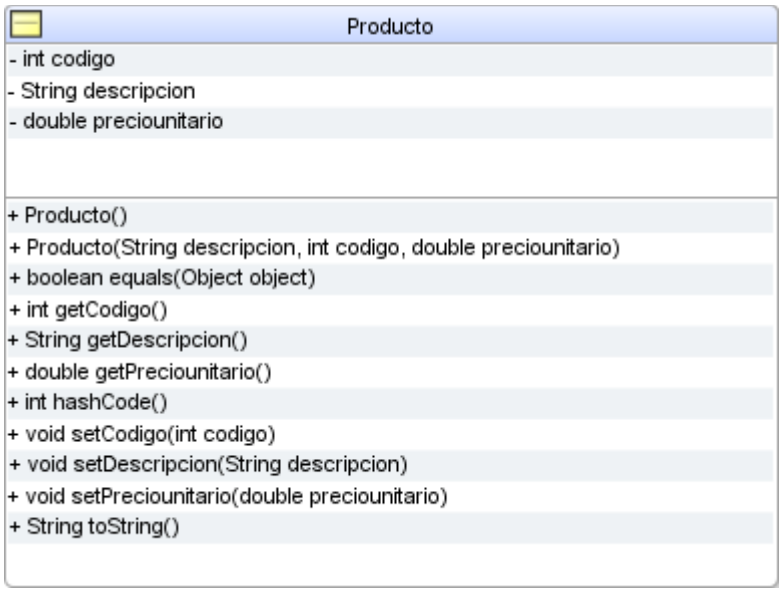
Inciso b)

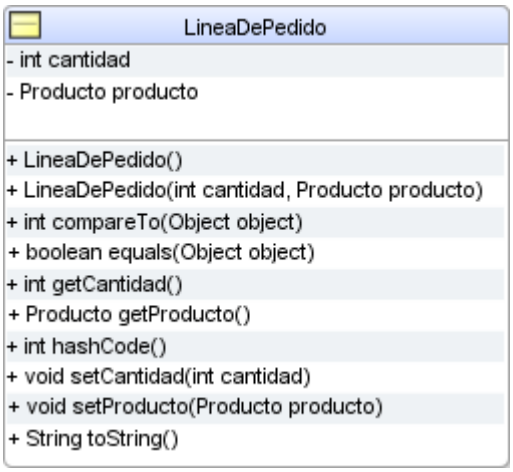
Escribe la clase **ConjuntoGenéricoOrdenado** que implementa la interfaz **IConjunto,** utilizando para su implementación la colección que considere conveniente, esta clase es genérica <T extends Comparable>.



Ejercicio 4

Modificaremos un ejercicio visto anteriormente en la guía 2: Escribe un programa que represente un pequeño negocio, construye las clases **Empleado**, **Producto** , **Pedido**,y **LineaDePedido** con sus correspondientes getters , setters, así como sus constructores.





Un pedido involucra un empleado responsable del mismo. (el cual podrá ser responsable de muchos pedidos), una fecha, y un conjunto de lineas de pedido cada línea de pedido consta de un producto especifico, y de una cantidad.

La clase **LineaDePedido** deberá sobrescribir el método equals, considerando como criterio el producto pedido. Deberá también reescribir el método toString.

La clase **Pedido** se extiende de un ConjuntoGenericoOrdenado de **LineasDePedido** deberá permitirnos obtener un detalle del conjunto de lineas de pedido y obtener el precio total de un pedido.

Realiza el diagrama de clases de las clases involucradas considerando el tipo de relación que las une.

Escribe una pequeña aplicación visual para lograr la funcionalidad del sistema.