EJERCICIOS SOBRE GENÉRICOS

Ejercicio 1

Crea una clase genérica llamada `Utils`, con 2 métodos estáticos que reciben un array de elementos de tipo `T` que implementa `Comparable<T>` con el siguiente comportamiento:

- `maximumElement`, devolverá el elemento máximo del array.
- `printArray`, mostrará por consola los elementos del array.

Crea las clases `Date` y `Client` e implementa en ambas el método `int compareTo(T o)`. Prueba los métodos de la clase `Utils` con arrays de `Date`, `String` y `Client`.

Nota: Comparando dos fechas será mayor la fecha posterior, y para la clase `Client` poned simplemente como atributos nombre y edad. El orden de los clientes se basa en la edad.

Ejercicio 2

Considera las siguientes clases:

```
public class A<T extends B> {
      private T t ;
      public A(T t) { this . t = t;  }
}

public class B { }
public class C extends B { }
```

¿Qué bloques de código de los que aparecen a continuación consideras que son correctos y por qué?

```
// BLOQUE 1
B b = new B ( ) ;
A<B> a1 = new A<B>(b) ;

// BLOQUE 2
C c = new C ( ) ;
A<C> a2 = new A<C>(c) ;
```

```
// BLOQUE 3
C c = new C ( ) ;
A<B> a3 = new A<>(c) ;
```

Ejercicio 3

Dado el siguiente código:

```
public class Generico<T extends P> { }

public class P {}

public class H1 extends P { }

public class H2 extends P { }
```

Indica cuáles de las siguientes instanciaciones de la clase `Generico` son correctas:

```
(a)  Generico <P> g = new Generico<P>() ;
(b)  Generico <H1> g = new Generico<H1>() ;
(c)  Generico <P> g = new Generico<H1> () ;
(d)  Generico <Object> g = new Generico<Object>() ;
```

Ejercicio 4

Considera las siguientes interfaces y clases:

```
public interface I1 { }

public interface I2 extends I1 { }

public class G<T extends I2> { }
```

Indica para qué definiciones de `B` es correcta la instrucción `G<B> g = new G<B>()`.

```
(a)  public class B implements I1 { }
(b)  public class B implements I2 { }
(c)  public class B implements I1, I2 { }
(d)  public class A implements I2 { }
(e)  public class B extends A { }
(f)  public class A implements I1 { }
(g)  public class B extends A implements I2 { }
```

Ejercicio 5

Realiza la implementación de un banco, definiendo las siguientes clases:

(a) Interfaz List<T>, con los siguientes métodos:

```
public abstract boolean addElem(T e);

public abstract boolean remove(T e);

public T getElem(int index);

public int size();
```

(b) Clase abstracta `Comparable<T>`, con métodos para representar orden entre los elementos de tipo `T`. Tiene que incluir las siguientes operaciones de orden: `less(T e)`, `greater(T e)`, `equal(T)`, `lessOrEquals(T)`, `greatOrEqual(T)`.

(c) Clase `SortedList<T extends Comparable<T>> implements List<T>`. No puedes usar ninguna colección de Java para implementar esta clase.

(d) Clase `Account`, con atributos `int accountNumber` e `int money`. Esta clase debe suministrar operaciones para añadir y eliminar dinero de la cuenta.

(e) Clase `Client`, con atributos `int id`, y `SortedList<Account> accounts`. El cliente contiene métodos para añadir y eliminar dinero de sus cuentas bancarias, así como para añadir y eliminar una cuenta.

(f) Cuando se añade dinero a una cuenta bancaria, en caso de que la cuenta no exista para ese cliente, debe crearse una nueva cuenta con saldo el dinero a añadir. De forma similar, si se quiere extraer dinero de una cuenta bancaria, pero el dinero a extraer supera el saldo actual, entonces la cuenta debe ser eliminada.

(g) Clase `Bank`, con atributos `SortedList<client> clients; String bankName`. Contiene métodos para añadir y eliminar clientes, así como para ingresar y eliminar dinero de una cuenta bancaria de un cliente. Al igual que para los clientes, si se intenta añadir dinero y el cliente no existe, entonces añadiremos al cliente el saldo a ingresar, y le insertamos en el banco como nuevo cliente. Al eliminar dinero de la cuenta bancaria de un cliente, comprobaremos que tras el reintegro el cliente sigue teniendo cuentas. En caso contrario, eliminamos al cliente del banco.

(h) Clase `Main`, que contiene el método main par probar el ejercicio.

Ejercicio 6

Dada la interfaz genérica `GenericSetInterface<T>` que representa el concepto matemático de conjunto como colección de objetos en la que no hay duplicados:

```
public interface GenericSetInterface<T> {

     public abstract boolean add(T e);

     public abstract boolean remove(Object o);
```

```
    public abstract boolean contains(Object o);

    public abstract int size();

}
```

(1) **add**
   - añade al conjunto el elemento pasado como parámetro si no pertenecía ya al conjunto.
   - devuelve true si dicho elemento no pertenecía ya al conjunto.

(2) **remove**
   - elimina el elemento pasado como parámetro si existe en el conjunto.
   - devuelve true si el conjunto contenía el elemento.

(3) **contains**.
   - devuelve true si el conjunto contiene el elemento pasado como parámetro.

(4) **size**.
   - devuelve el número de elementos en el conjunto.

Implementar la clase genérica `GenericSetNotNull<T>` que implementa la interfaz `GenericSetInterfaz` y que no acepta `null` como elemento del conjunto.

Ejercicio 7

Explica por qué el compilador de Java rechaza el siguiente código

```
List<Integer> x = new ArrayList<Integer>();
List<Number> y = x;
```

Y acepta el siguiente:

```
Integer[] x = new Integer[10];
Number[]  y = x;
```