Java

Tema 6: Uso avanzado de clases

Ampliación de clases

Fase III

Contenidos

1. Trocear un texto
2. Interfaces
   - Comparable
   - Comparator

Java

Trocear un texto

La clase StringTokenizer

- **StringTokenizer** es otra clase de cadenas de caracteres, permite trocear un String en varias subcadenas (tokens). Por defecto usa como delimitador " \t\n", el espacio, tabulador, salto de linea.

```
StringTokenizer s = new StringTokenizer("El precio es 20 euros");
 System.out.println( "Total trozos: " + s.countTokens() );
  while( s.hasMoreTokens() == true ){  //si hay mas trozos
      System.out.println( s.nextToken() );  //siguiente trozo
 }
```

- Usar otro delimitador StringTokenizer(texto, **delimitador**)

```
String txt = "El precio es 20 euros. Compra dos unidades";
StringTokenizer s = new StringTokenizer( txt , "." );
System.out.println( "Total trozos: " + s.countTokens() );
 while( s.hasMoreTokens() == true ){  //si hay mas trozos
    System.out.println( s.nextToken() );  //siguiente trozo
}
```

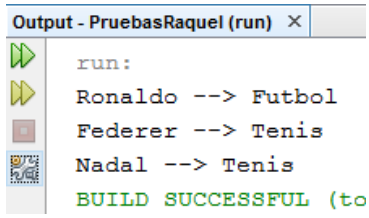
La clase StringTokenizer

- Usar varios delimitadores StringTokenizer(texto, **delimitadores**)

```
String datos = "Ronaldo=Futbol;Federer=Tenis;Nadal=Tenis;";
StringTokenizer st = new StringTokenizer(datos, "=;");

while (st.hasMoreTokens()) {
    String jugador = st.nextToken();
    String deporte = st.nextToken();
   System.out.println(jugador + " " + deporte);

}
```



Ejercicio: Crea una clase Jugador (con nombre y deporte), crea un objeto con cada uno de los datos recogidos y añádelos a un Set<Jugador>. Lista esa información

5

Método split()

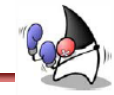
- Se puede hacer lo mismo con el método **split**() de la clase String

```
String s = new String("El precio es 20 euros");

/* \\s representa a un espacio en blanco */
  String[] trozos = s.split("\\s");

  for(String t: trozos){
       System.out.println( t );
  }
```

6



Java

Interfaces

Interfaces

- Un interface es una clase formada por una **lista de declaraciones de métodos no implementados** (todos abstractos, es decir, no tienen código, sin necesidad de añadir la palabra *abstract*).
- Es una **clase abstracta pura**, es decir, no hay ninguna implementación de sus métodos.
- También puede incluir constantes, es decir, *atributos estáticos* (propios de toda la interfaz, por tanto NO se pueden heredar) y *finales* (no se puede modificar su valor) → public static final …. No es obligatorio ponerlo
- Las interfaces **NO SE PUEDEN INSTANCIAR**, es decir no se pueden crear objetos mediante new.

Interfaces

- Declaración de un interface:

```
public interface nombre_interface {
        . . .
}
```

- Ejemplo de interface para un teléfono móvil

```
public interface APIMovil {
    int active=1;    //es como public static final
    int inactive=0;  //es como public static final

    public boolean isAlive();
    public String getModel();
    public int getBatteryCharge();
    public String getIMEI();
    public ActivityStatus getActivityStatus();
    public void sendSMS(String numTelefono,String msg);
    public void dialCommand(String numTelefono);
    public ArrayList<Message> getMessajes(SMSType smsType);
}
```

Implementación de Interfaces

- Una vez definida una interface, **una o más clases pueden implementarla**, sobrescribiendo sus métodos (**@Overrride**)
- Una clase que implemente una interface deberá implementar todos sus métodos.
- Declaración de una clase que implementa una interface:

```
public class nombre_clase implements interface {
        . . .
}
```

- Una clase puede heredar de una sola clase, pero puede implementar varias interfaces.

```
public class claseA extends clasePadre
                    implements interface1, interface2 {
    . . .
}
```

Interfaces

Ejemplo: Clase que implementa la interfaz del teléfono móvil

```
public class MovilNokia implements APIMovil {
    public boolean isAlive(){
     ……..
    };

    public int getNumEntries(MemoryStorage memoryStorage){
      ……..
    }

     ……..
}
```

Debe implementar TODOS los métodos de la interfaz

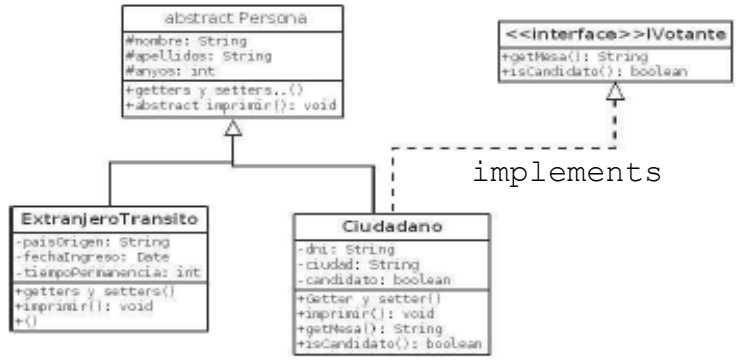
Interfaces

- Entonces, si no se pueden crear objetos (al ser abstracta), no se pueden heredar sus atributos (al ser estáticos) y no se hereda código (al tener todos sus métodos abstractos) ¿Para qué sirve una interfaz?

- Una interfaz es una especie de contrato, las clases que implementan una interfaz se comprometen a definir/implementar una serie de métodos.

Interfaces: ejemplo

- La manera de definir mediante UML una interfaz es la siguiente:



Interfaces: ejemplo

- Tenemos una clase abstracta Persona, de la cual heredan tanto ExtranjeroTransito como Ciudadano. Toda persona es o ciudadano o extranjero en transito.
- La clase Ciudadano implementa la interfaz Votante, que tiene dos métodos abstractos, getMesa() y isCandidato(). Al implementar la interfaz, Ciudadano tiene que definir el código de todos los métodos.
- Por tanto,
  - Un Ciudadano es un persona
  - Un extranjero en transito tambien es una persona
  - Un ciudadano tiene la capacidad de comportarse como un votante.
- Ver en proyecto adjunto

Polimorfismo en Interfaces

- Una interface se trata como cualquier otra clase.
- Un objeto de una clase puede ser referenciado/apuntado por una interface, siempre y cuando esa clase o su superclase implemente dicho interface.

```
//interface
APIMovil apiMovil;

//class
MovilNokia movilNokia = new movilNokia();

// polimorfismo, referencia a un objeto de otra clase diferente
apiMovil = movilNokia;
```

15

Interface Comparable

- Existe una interfaz del api de java, llamada **Comparable,** que tiene un método *compareTo()* que sirve para marcar el criterio para ordenar los elementos de una misma clase.
- compareTo(objeto) devuelve -1 si this se sitúa delante del objeto, devuelve 1 si se sitúa detrás y 0 si son iguales .
- compareTo() es invocada automáticamente por el método de ordenación sort().

16

Interface Comparable: ejemplo

```
public  class Persona implements Comparable<Persona>{
    private String nombre;
    private String apellido;
    private Integer anyos;
…
    @Override
    public int compareTo(Persona otro) {
        int comparacion = 0;
        if (this.anyos < otro.getAnyos())
            comparacion = -1;
        else if (this.anyos > otro.getAnyos())
            comparacion = 1;

        return comparacion;
//        OTRA FORMA
//        return  anyos.compareTo(otro.anyos);
    }
```


Interface Comparable: ejemplo

```
public static void main(String[] args) {
    List<Persona> lista = new ArrayList<>();
    Persona p1 = new Persona("Paco",60);
    Persona p2 = new Persona("Raquel",30);
    Persona p3 = new Persona("Brad",45);
    lista.add(p1);
    lista.add(p2);
    lista.add(p3);

    System.out.println(lista);

    Collections.sort(lista);//invoca a compareTo()

    System.out.println("Ordenada asc por edad: " + lista);

    }
```

Interface Comparator

- Existe una interfaz del api de java, llamada **Comparator,** que tiene un método *compare()* que sirve para marcar el criterio para ordenar los elementos de una misma clase.
- compare(objeto1, objeto2) devuelve -1 si objeto1 se sitúa delante de objeto2, si va detrás devuelve 1 y si son iguales devuelve 0.
- Nos servirá como argumento al invocar al método de ordenación sort().
- Para crear varios criterios de ordenación, definiremos varias clases que implementen esta interfaz

Interface Comparator: ejemplo

```
public  class Persona {
    private String nombre;
    private String apellido;
    private Integer anyos;
……..
}

public class OrdenEdad implements Comparator<Persona>{
    @Override
    public int compare(Persona o1, Persona o2) {
         int comparacion = 0;
        if (o1.getAnyos < o2.getAnyos())
            comparacion = -1;
        else if (o1.getAnyos > o2.getAnyos())
            comparacion = 1;
        return comparacion;
    }
}
```

Interface Comparator: ejemplo

```
public class OrdenNombre implements Comparator<Persona>{

    @Override
    public int compare(Persona o1, Persona o2) {
        int comparacion = 0;
        if (o1.getNombre() < o2.getNombre())
            comparacion = -1;
        else if (o1.getNOmbre() > o2.getNombre())
            comparacion = 1;

        return comparacion;
    }
}
```


Interface Comparator: ejemplo

```
public static void main(String[] args) {
      List<Persona> lista = new ArrayList<>();
      Persona p1 = new Persona("Paco",60);
      Persona p2 = new Persona("Raquel",30);
      Persona p3 = new Persona("Brad",45);
      lista.add(p1);
      lista.add(p2);
      lista.add(p3);

      System.out.println(lista);

      Collections.sort(lista, new OrdenEdad());
      System.out.println("Orden asc por edad: " + lista);

      Collections.sort(lista, new OrdenNombre());
      System.out.println("Orden asc por nombre:" + lista);
    }
```