```java
import java.util.HashSet;



/**
 * Esta clase modela la mano de cartas de UNO que puede tener cada jugador. Las referencias a las cartas de la mano se deben almacenar usando un array.
 *
 *
 */
public class ManoDeUno {

    private HashSet <CartaDeUNO> mano;
    private int numMano;
    private int numMaximo;

    /**
     * Constructor de la clase. La pila reciÃ©n instanciada no almacena cartas.
     * @param nÃºmeroMÃ¡ximodeCartas  NÃºmero mÃ¡ximo de cartas que puede almacenar la pila de cartas.
     */
    public ManoDeUno(int nÃºmeroMÃ¡ximodeCartas) {

        mano = new HashSet <CartaDeUNO> (nÃºmeroMÃ¡ximodeCartas);
        numMano=0;
        numMaximo = nÃºmeroMÃ¡ximodeCartas;
    }

    /**
     * Indica si la mano estÃ¡ vacÃ-a (no tiene cartas).
     * @return true si esta mano no tiene ninguna carta disponible.
     */
    public boolean estÃ¡VacÃ-a() {

        boolean manoVacia = false;

        if(mano.isEmpty() && numMano == 0) {
            manoVacia = true;
        }

        return manoVacia;
    }

    /**
     * Indica si la mano estÃ¡ llena (ya no se puede aÃ±adir cartas a la mano).
     * @return true si esta mano no tiene ninguna carta disponible.
     */
    public boolean estÃ¡Llena() {

        boolean manoLlena = false;

        if(numMano >= numMaximo) {
            manoLlena = true;
        }
        return manoLlena;
    }

    /**
     * Agrega la carta recibida como argumento a esta mano de cartas.
     * @param carta Carta que se desea agregar a esta mano.
     */

```

```
62      public void agregarCartaâ€‹(CartaDeUNO carta) {
63  
64          boolean coincide = mano.contains(carta);
65  
66          if(mano!=null && !coincide) {
67               mano.add(carta);
68               numMano++;
69          }
70      }
71  
72  
73      /**
74       * Devuelve la referencia a una carta de esta mano que se puede jugar para apilarla sobre la carta cuya referencia se ha recibido como argumento.
75       *
76       * @param cartaSobreLaQueHayQueApilar Carta para la que hay que buscar una carta de la mano que se pueda jugar.
77       * @return Referencia a una carta de esta mano que se puede jugar para apilarla sobre la carta cuya referencia se ha recibido como argumento. null si no hay carta en esta mano que se pueda jugar.
78       */
79  
80  
81      public CartaDeUNO extraerCartaApilableSobreâ€‹(CartaDeUNO cartaSobreLaQueHayQueApilar) {
82  
83          Iterator <CartaDeUNO> iter = mano.iterator();
84  
85          boolean encontrado = false;
86          CartaDeUNO aux = null;
87          CartaDeUNO entregada = null;
88  
89          while(iter.hasNext() && !encontrado) {
90  
91              aux = iter.next();
92  
93              if(aux.sePuedeApilarSobre(cartaSobreLaQueHayQueApilar)){
94  
95                  iter.remove();
96                  numMano--;
97                  encontrado = true;
98                  entregada = aux;
99              }
100 
101         }
102           return entregada;
103     }
104 
105     /**
106      * Devuelve una representaciÃ³n textual de las cartas contenidas en esta mano.
107      * @return RepresentaciÃ³n textual de las cartas contenidas en esta mano.
108      */
109     public String getMano() {
110 
111         Iterator <CartaDeUNO> iter = mano.iterator();
112         String manoCartas ="";
113 
114          while(iter.hasNext()) {
115 
116              manoCartas = manoCartas+" "+ iter.next().getIdentificador() ;
117         }
118         if(numMano ==  0) {
```

```
119             manoCartas = "Sin cartas.";
120         }
121 
122         return manoCartas;
123     }
124 
125 
126 }
127 
```