```java

/**
 * 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 final CartaDeUNO mano[];
    private int numMano;

    /**
     * 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 CartaDeUNO[nÃºmeroMÃ¡ximodeCartas];
        numMano=0;
    }

    /**
     * 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(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 >= mano.length) {
            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.
     */

    public void agregarCartaâ€‹(CartaDeUNO carta) {

        if(mano!=null) {
            mano[numMano]=carta;
            numMano++;
```

```
61          }
62      }
63
64
65      /**
66       * 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.
67       *
68       * @param cartaSobreLaQueHayQueApilar Carta para la que hay que buscar una carta de la mano que se pueda jugar.
69       * @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.
70       */
71      public CartaDeUNO extraerCartaApilableSobreâ€‹(CartaDeUNO cartaSobreLaQueHayQueApilar) {
72
73          CartaDeUNO cartaExtraida = null;
74          boolean encontrado = false;
75          int contador = 0;
76          int  x,y,z,j;
77
78
79          for(int i = 0; i < numMano && !encontrado; i++) {
80              if(mano[i].sePuedeApilarSobre(cartaSobreLaQueHayQueApilar)){
81
82                  contador = i;
83                  encontrado = true;
84              }
85           }
86
87          if(encontrado) {
88              cartaExtraida = mano[contador];
89              numMano--;
90           }
91           else {
92               cartaExtraida = null;
93           }
94
95          x = numMano - contador;
96          y = 1;
97          z = 0;
98          j = 0;
99
100         while(encontrado && j < x) {
101          mano[contador+z] = mano[contador+y];
102          y++;
103          z++;
104          j++;
105         }
106
107          return cartaExtraida;
108     }
109
110     /**
111      * Devuelve una representaciÃ³n textual de las cartas contenidas en esta mano.
112      * @return RepresentaciÃ³n textual de las cartas contenidas en esta mano.
113      */
114     public String getMano() {
115
116         String manoCartas ="";
117
```

```
118          for(int i = 0; i < numMano; i++) {
119
120              manoCartas = manoCartas+" "+ mano[i].getIdentificador() ;
121          }
122          if(numMano ==  0) {
123              manoCartas = "Sin cartas.";
124          }
125
126          return manoCartas;
127      }
128
129
130  }
131
```