```java
public class Trabajo {

   int id;
   int hora;
   int tamano;
   int izquierda;
   int derecha;

   //la hora en la que el trabajo empieza realmente en un horario
   int empieza;

   public Trabajo (int id, int hora, int tamano, int izquierda, int derecha) {
      this.id = id;
      this.hora=hora;
      this.tamano=tamano;
      this.izquierda=izquierda;
      this.derecha=derecha;
      this.empieza = 0;
   }
   public Trabajo (Trabajo viejo) {
      id=viejo.id;
      hora=viejo.hora;
      tamano=viejo.tamano;
      izquierda=viejo.izquierda;
      derecha=viejo.derecha;
      empieza=viejo.empieza;
   }

   public int getId() {
      return this.id;
   }
   public int getHora(){
      return this.hora;
   }
   public int getTamano(){
      return this.tamano;
   }
   public int getIzquierda(){
      return this.izquierda;
   }
   public int getDerecha(){
      return this.derecha;
   }
   public int getEmpieza(){
      return this.empieza;
   }
   public void setEmpieza(int hora){
      this.empieza = hora;
   }

   public boolean esIgual(Trabajo trabajo){
      return (this.id == trabajo.id);
   }

   public String toString(){
      return(id + ":" +
            hora + " " + tamano + "[" + izquierda + "," + derecha + "]" +
            empieza);
   }
}
```

```
import java.util.ArrayList;
import java.lang.StringBuffer;

public class Horario{

   ArrayList trabajos; // ordenada por hora del inicio real

   int [] horasLibres = new int [168];  //las horas son de 0 a 167
   // -1 indica que la hora está libre,
   // un valor >= 0 indica el número de trabajo

   public Horario(){
      trabajos = new ArrayList();
      for(int i=0; i<horasLibres.length; i++){
         horasLibres[i]=-1;
      }
   }


   // Da un nuevo horario que es la copia de otro
   public Horario(Horario orig){
      trabajos = new ArrayList();
      for (int i=0; i<orig.trabajos.size(); i++) {
         trabajos.add( (Trabajo) orig.trabajos.get(i) );
      }

      for(int i=0; i<horasLibres.length; i++){
         horasLibres[i]=orig.horasLibres[i];
      }
   }

   public int numTrabajos(){
      return trabajos.size();
   }

   public double[] periodosLibres(){
      /*
         Devuelve un array con:
         [0]: número de horas libres en el horario
         [1]: longitud media de los períodos de horas libres
      */
      //el número de horas libres en el período actual de horas libres
      int acc=0;
      //el número de períodos de horas libres
      int numPer=0;

      //  (para calcular la media):
      // la suma de las longitudes de todos los períodos de horas libres
      int sumLong=0;

      //el número de horas libres
      int numHL=0;

      for (int i = 0; i<horasLibres.length; i++){
         if(horasLibres[i]==-1){
            numHL++;
            acc++;
         }
         else if (acc!=0){
            sumLong += acc;
            numPer++;
            acc = 0;
         }
      }
      double[] result = new double[2];
      result [0] = numHL;
```

```
        result [1] = (double)sumLong / numPer;

        return result;

    }

    public boolean contiene(Trabajo trabajo){

        for(int i=0; i<horasLibres.length; i++){
            if (trabajo.id == horasLibres[i]) return true;
        }
        return false;
    }

    public boolean compruebaLimites(Trabajo trabajo, int desplazamiento){
        // Devuelve falso si el trabajo queda fuera del horario
        if (trabajo.hora + desplazamiento < 1) return false;
        if (trabajo.hora + desplazamiento + trabajo.tamano > 168) return false;
        //167+0 1  168 > 168
        return true;
    }

    public boolean compruebaTolerancia(Trabajo trabajo, int desplazamiento){
        //   Comprueba si el desplazamiento está permitido
        // por la tolerancia del trabajo

        if (desplazamiento < 0){
            if (  Math.abs( desplazamiento ) > trabajo.izquierda  ) {
                return false;
            }
        }
        else {
            if ( desplazamiento > trabajo.derecha ) {
                return false;
            }
        }
        return true;
    }

    public boolean compruebaHorasLibres(Trabajo trabajo, int desplazamiento){

        // a: hora a la que debe empezar el trabajo que nos han pasado
        // b: hora a la que debe acabar
        int a=trabajo.hora+desplazamiento;
        int b= a + trabajo.tamano;

        a--; b--; //porque empieza en 0

        for(int i=a; i<b; i++){
            if(horasLibres[i] != -1) {
                //System.out.println("Hora i="+i+" ocupada. a="+a+" b="+b);
                return false;
            }
        }

        return true;
    }




    public boolean sePuedePoner(Trabajo trabajo, int desplazamiento){


        //comprueba límites
        if (!compruebaLimites(trabajo,desplazamiento))
```

```
            return false;

        //comprueba si el desplazamiento está permitido
        if (!compruebaTolerancia(trabajo,desplazamiento))
            return false;

        //si el horario está vacío, se puede poner
        if ( trabajos.size() == 0) return true;

        //comprueba si el trabajo está puesto
        if (contiene(trabajo))
            return false;

        //comprueba si las horas están libres
        if (!compruebaHorasLibres(trabajo,desplazamiento))
            return false;

        return true;
    }

    public boolean sePuedeQuitar(Trabajo trabajo){
        return contiene(trabajo);
    }
    public boolean sePuedeAjustar(Trabajo trabajo, int desplazamiento){
        if (!contiene(trabajo))
            return false;
        if (!compruebaLimites(trabajo,desplazamiento))
            return false;
        if (!compruebaTolerancia(trabajo, desplazamiento)){
            return false;
        }
        int a = trabajo.hora + desplazamiento;
        int b = a + trabajo.tamano;

        a--; b--;  //porque empieza en 0

        //   Un trabajo se puede ajustar si la nueva posición está libre
        // u ocupada por el mismo trabajo
        //
        for (int i=a; i<b; i++){
            if (!(horasLibres[i]==-1 || horasLibres[i]==trabajo.id)) return false;
        }

        return true;
    }

    public boolean sePuedeIntercambiar(Trabajo viejo, Trabajo nuevo){
        // cambiar viejo (dentro del horario) por nuevo (aún sin poner)
        //
        if (!contiene(viejo))
            return false;
        if (contiene(nuevo))
            return false;
        Horario h = new Horario(this);
        h.quitar(viejo);
        return h.sePuedePoner(nuevo,0);
    }


    public void poner(Trabajo trabajo, int desplazamiento){
        int inicio = 0;
        int i = 0;
        int total = trabajos.size();
        Trabajo nuevo= new Trabajo (trabajo);
        nuevo.setEmpieza( trabajo.hora + desplazamiento );
```

```
        while (i<total){
            inicio = ((Trabajo)(trabajos.get(i))).empieza;
            if (inicio > nuevo.getEmpieza() ) break;
            i++;
        }
        trabajos.add(i,nuevo);
        // −1 porque trabajo.hora empieza en 1 y no en 0
        for(  i = nuevo.getEmpieza()−1;
            i<nuevo.getEmpieza()+nuevo.getTamano()−1;
            i++
            ){
            horasLibres[i]=nuevo.getId();
        }
        //System.out.println(nuevo);
        //System.out.println(horasLibresToString());
    }

    public void quitar (Trabajo trabajo){
        int inicio = 0;
        int i = 0;
        int total = trabajos.size();

        while (i<total){
            if (((Trabajo)(trabajos.get(i))).esIgual(trabajo)) break;
            i++;
        }

        trabajos.remove(i);
        // −1 porque trabajo.hora empieza en 1 y no en 0
        for(i = trabajo.empieza−1; i<trabajo.empieza+trabajo.tamano−1; i++){
            horasLibres[i]=−1;
        }
    }

    public void ajustar (Trabajo trabajo, int desplazamiento) {
        quitar(trabajo);
        poner(trabajo, desplazamiento);
    }

    public void intercambiar (Trabajo viejo, Trabajo nuevo) {
        quitar(viejo);
        poner(nuevo,0);
    }


    public String toString(){
        StringBuffer buf=new StringBuffer();
        buf.append( "Horario de " + trabajos.size() + " trabajos:\n");
        int i=0;
        while ( i<trabajos.size() ) {
            buf.append( ((Trabajo)trabajos.get(i)).toString() + "\n");
            i++;
        }
        buf.append(horasLibresToString()+"\n");

        return buf.toString();
    }

    public String horasLibresToString(){
        StringBuffer buf = new StringBuffer();

        buf.append("Horas ocupadas: \n");
        int j = 0;
        int id = −1;
```

```
        //-1 porque en la última hora no puede empezar un trabajo nuevo
        int n = horasLibres.length-1;

        for(int i=0; i<n; i++) {

            if(i%10==0) buf.append(" ");
            if (i%40==0) buf.append("\n");
            if (horasLibres[i] !=-1){
                if(id != horasLibres[i]){
                    buf.append("X");
                }
                else{
                    buf.append("x");
                }
                id = horasLibres[i];
                j++;
            }
            else{
                buf.append("_");
            }
        }

        buf.append("\n En el horario hay " + j + " horas ocupadas.\n"+
            " En el horario hay " + (n-j) + " horas libres.");
        return buf.toString();
    }

}
```

```
import java.util.List;
import java.util.ArrayList;
import aima.search.framework.Successor;
import aima.search.framework.SuccessorFunction;

public class Sucesores implements SuccessorFunction {

   public List getSuccessors(Object state) {
      Horario h = (Horario) state;
      List sucesores = new ArrayList();


      // recorrido por todos los trabajos ya puestos
      for (int i=0; i<h.trabajos.size(); i++) {
         Trabajo trabajo= (Trabajo) h.trabajos.get(i);


         //// Operador: quitar(trabajo) ////

         if ( h.sePuedeQuitar(trabajo) ) {
            Horario result = new Horario(h);
            result.quitar(trabajo);
            String mensaje = "quitar tr. "+trabajo.id;
            sucesores.add( new Successor(mensaje, result) );
         }



         //// Operador: ajustar(trabajo,desplazamiento) ////

         // probar cada desplazamiento permitido por la tolerancia
         for (int d = -5; d <= 5; d++) {
            if ( h.sePuedeAjustar(trabajo,d) ) {
               Horario result = new Horario(h);
               result.ajustar(trabajo,d);
               String mensaje = "ajustar tr. "+trabajo.id+" a despl. "+d;
               sucesores.add( new Successor(mensaje, result) );

            }
         }



         //// Operador: intercambiar(trab_viejo,trab_nuevo) ////

         for (int j=0; j<HorarioTrabajo.todosTrabajos.size(); j++) {
            Trabajo trabajoNuevo= (Trabajo) HorarioTrabajo.todosTrabajos.get(j);
            if ( h.sePuedeIntercambiar(trabajo,trabajoNuevo) ) {
               Horario result = new Horario(h);
               result.intercambiar(trabajo,trabajoNuevo);
               String mensaje =
                        "intercambiar tr. "+trabajo.id+" con el tr. "+trabajoNuevo.id;
               sucesores.add( new Successor(mensaje, result) );
            }
         }




      }
      //System.out.println("sucesores, paso intermedio");

      // recorrido por todos los trabajos disponibles para poner
      for (int i=0; i<HorarioTrabajo.todosTrabajos.size(); i++) {
```

```
         Trabajo trabajo= (Trabajo) HorarioTrabajo.todosTrabajos.get(i);



         //// Operador: poner(trabajo,desplazamiento) ////

         // probar cada desplazamiento
         for (int d = -5; d <= 5; d++) {
            if ( h.sePuedePoner(trabajo,d) ) {
               Horario result = new Horario(h);

               result.poner(trabajo,d);
               String mensaje = "poner tr. "+trabajo.id+" con despl. "+d;
               sucesores.add( new Successor(mensaje, result) );

            }
         }


      }

      System.out.println("Generados "+sucesores.size()+" sucesores "+
                         "de un horario con "+h.numTrabajos()+" trabajos.");

      return sucesores;
   }

}
```

```java
import aima.search.framework.HeuristicFunction;

public class Heuristica implements HeuristicFunction {

   public int getHeuristicValue (Object state) {
      Horario h = (Horario) state;
      return Heuristica2(h);
   }

   public int Heuristica1(Horario h) {
      return -h.numTrabajos();
   }

   public int Heuristica2(Horario h) {

      /*
         n: número de trabajos
         s: longitud media de los períodos de horas libres
         hl: número de horas libres

         f(n,s,h) = 168^2 * n  +  168 * s  +  hl
      */

      int n = h.numTrabajos();
      double [] result = h.periodosLibres();
      int hl = (int) result [0];
      double s = result [1];

      return  - ( 168*168*n + (int)(168*s) + hl );

   }

   public int Heuristica3(Horario h){
      /*
         n: número de trabajos
         hl: número de horas libres

         f(n,h) = 168 * n + hl
      */

      int n = h.numTrabajos();
      double [] result = h.periodosLibres();
      int hl = (int) result [0];

      return  - ( 168*n + hl );

   }

   public int Heuristica4(Horario h) {

      /*
         n: número de trabajos
         s: longitud media de los períodos de horas libres

         f(n,s) = 168 * n  +  s
      */

      int n = h.numTrabajos();
      double [] result = h.periodosLibres();
      double s = result [1];

      return  - ( 168*n + (int)s );

   }

}
```

```java
import aima.search.framework.GoalTest;

public class ComprFin implements GoalTest {

   public boolean isGoalState(Object state) {
      Horario h = (Horario) state;
      return h.numTrabajos() == 168;
   }

}
```

```java
import java.util.ArrayList;
import java.util.Iterator;
import java.util.List;
import java.util.Properties;
import java.util.Random;

import aima.search.framework.Problem;
import aima.search.framework.Search;
import aima.search.framework.SearchAgent;
import aima.search.informed.HillClimbingSearch;
import aima.search.informed.SimulatedAnnealingSearch;

public class HorarioTrabajo{

   public static ArrayList todosTrabajos;
   public static Horario h;

   public static void llenarLista(){
      todosTrabajos = new ArrayList();
      Random rand=new Random(1);

      int cuantos = 500;
      for(int i=0; i<cuantos; i++){
         int hora = rand.nextInt(167)+1; // 1 a 167
         int tamano = rand.nextInt(5)+1; // 1 a 5
         int izquierda = rand.nextInt(6); // 0 a 5
         int derecha = rand.nextInt(6); // 0 a 5

         Trabajo dummy =
            new Trabajo(i, hora, tamano, izquierda, derecha);
         todosTrabajos.add(dummy);
      }
   }

   public static void imprimir(){
      System.out.println("La lista de trabajos con "+
            todosTrabajos.size()+"\n");
      for(int i=0; i<todosTrabajos.size(); i++){
         System.out.println((Trabajo)todosTrabajos.get(i));
      }
   }

   // Crea un horario con bastantes trabajos al azar, sacados de todosTrabajos
   // Es un posible estado inicial
   public static Horario horarioAleatorio(){
      h = new Horario();
      Trabajo t;
      for(int i=0; i<todosTrabajos.size(); i++){
         t = (Trabajo)todosTrabajos.get(i);
         if (h.sePuedePoner(t,0)) h.poner(t,0);
      }
      return h;
   }

   public static void main(String[]args){
      llenarLista();
      //imprimir();
      //System.exit(0);

      h = new Horario();

      //h = horarioAleatorio();

      //System.out.println(  h  );
```

```java
      //horarioTrabajoHillClimbingDemo();
      horarioTrabajoSimulatedAnnealingDemo();

   }

   public static void horarioTrabajoHillClimbingDemo() {
      System.out.println("\nHorarioTrabajo con el algoritmo Hill Climbing −−>");
      try {
         Problem problem = new Problem(h,
               new Sucesores(),
               new ComprFin(),
               new Heuristica());

         Search search = new HillClimbingSearch();
         SearchAgent agent = new SearchAgent(problem, search);
         printActions(agent.getActions());
         printInstrumentation(agent.getInstrumentation());

      } catch (Exception e) {
         e.printStackTrace();
      }

   }

   private static void horarioTrabajoSimulatedAnnealingDemo() {
      System.out.println("\nHorarioTrabajo con el algoritmo Simulated Annealing −−>");
      try {
         Problem problem = new Problem(h,
               new Sucesores(),
               new ComprFin(),
               new Heuristica());

      // SimulatedAnnealingSearch(int steps, int stiter, int k, double lamb)
      //                            10000      100        20     0.005

         Search search =
            new SimulatedAnnealingSearch(10000,100,20,0.005);

         SearchAgent agent = new SearchAgent(problem, search);

         printActions(agent.getActions());
         printInstrumentation(agent.getInstrumentation());
      } catch (Exception e) {
         e.printStackTrace();
      }

   }

   private static void printInstrumentation(Properties properties) {
      Iterator keys = properties.keySet().iterator();
      while (keys.hasNext()) {
         String key = (String) keys.next();
         String property = properties.getProperty(key);
         System.out.println(key + ":" + property);
      }

   }

   private static void printActions(List actions) {
      for (int i = 0; i < actions.size(); i++) {
         String action = (String) actions.get(i);
         System.out.println(action);
      }
   }
}
```