

## TIPS para la implementación del Trabajo Práctico VÍBORA

- ✓ El agente no puede acceder a todo el ambiente directamente ya que no lo conoce. Esto implica que la clase que representa al agente no puede tener acceso a la variable de estado del ambiente:

*celda = ambiente[x][y]; // No hacer esto*

*celda = ambiente.getCelda(x,y) // No hacer esto*

- ✓ El agente puede estar simultáneamente en una misma celda con la comida (excepto en la posición inicial).
- ✓ Cuando se llama un método con parámetros de objeto, las variables que se pasan en el cuerpo del método son pasadas por referencia, lo que significa que cualquier cosa que se haga a los objetos del método, también afecta a los objetos originales. Esto incluye también a los arreglos y a todos los objetos que contienen los arreglos. Los tipos primitivos son pasados por valor.
- ✓ Consejos:

- Clonar el objeto (generar una copia) antes de utilizarlo como argumento en un método. El método clonar, debe también clonar todos los objetos anidados dentro del objeto original.
- Armar los árboles de búsqueda resultantes del proceso de toma de decisión del agente en cadenas anidadas entre corchetes:

Ej: [Nodo: 1

[Nodo: 2[Nodo: 5] [Nodo: 6]]

[Nodo: 3[Nodo: 7]]

[Nodo: 4]

]

- O en formato XML:

```

2 public class Nodo implements Cloneable {
3     ...
4     ...
5     ...
6     public String generarXml(){
7
8         String str = "";
9
10        str = "<Nodo";
11        str = str + " idNodo=\"" + idNodo + "\"";
12        str = str + " posicion=\"(" + state.agentX + ", " + state.agentY + ")\"";
13        str = str + " orientacion=\"" + Direction.toString(state.getAgentDir()) + "\"";
14        str = str + " accion=\"" + action + "\"";
15        str = str + " distancia_real=\"" + realDistance + "\"";
16        str = str + " habitaciones_no_visitadas=\"" + unknownRooms + "\"";
17        str = str + " funcion_heuristica=\"" + heuristicFunction() + "\"";
18        str = str + ">";
19        str = str + toString();
20        for (int i=0; i<getCantidadHijos()+1; i++) {
21            str = str + hijos[i].generarXml();
22        }
23        str = str + "</Nodo>";
24
25        return str;
26
27    }

```

- ✓ Links de interés:

- Blog de la materia: <http://iacatedra.blogspot.com>

- Artificial Intelligence, A Modern Approach: <http://aima.cs.berkeley.edu/>
- Search and game playing: <http://aima.cs.berkeley.edu/ai.html#search>
- Ejemplos y estructuras de datos: <http://www.java2s.com/>
- Java Programming Style Guidelines: <http://geosoft.no/development/javastyle.html>
- Robocup: <http://www.robocup.org/>

### Interface de la clase Calculador

#### Constructores:

**public** *Calculador(String nombreGrupo)*

Genera una instancia de la clase calculador y asigna el parámetro *nombreGrupo* como nombre del grupo. Este nombre será utilizado en el simulador para identificar al grupo.

**public** *Calculador()*

Genera una instancia de la clase calculador sin asignar un nombre al grupo.

#### Métodos de instancia:

**public** *Vector getComida()*: debe invocarse para obtener las posiciones de los alimentos. El vector que se retorna contiene instancias de Pair indicando las posiciones x,y donde hay alimento en el mundo del juego. La longitud del vector puede variar.

**public** *Pair getDimensiones()*: retorna un par indicando la cantidad de filas (x) y columnas(y) del mundo.

**public** *Pair getPosicionInicial()*: retorna un par indicando la posición x,y inicial del agente-víbora.

**public** *void reportarAccion(String a)*: El argumento es un string que describe la acción que el agente realiza. Para ello se definieron las siguientes variables de clase:

AVANZAR

COMER

GIRAR

Todas aquellas acciones que sean “avanzarDerecha” o “avanzarEste”, etc., son representadas para el calculador por la variable de clase “AVANZAR”. Idem para COMER y GIRAR. El argumento de este método deben ser los nombres de estas variables de clase invocadas como p.e. *Calculador.AVANZAR*.

**public** *int getPerformance()*: retorna un entero correspondiente a la performance del agente víbora cuando termina la simulación. Se debe llamar sólo al finalizar, ya que una vez que es invocado, en las siguientes invocaciones retornará -1.

### Ayuda para el uso del “calculador.jar”

#### Cómo agregar un archivo .jar a un proyecto en Eclipse.

Se debe tener un Proyecto Java abierto. Para importar el archivo *calculador.jar*, ir al menú *Project* → *Properties* como muestra la figura 1.

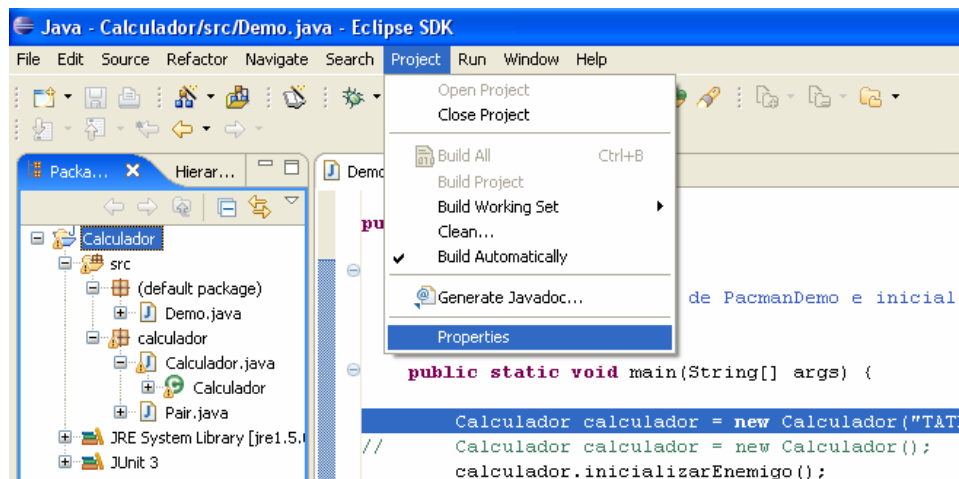


Figura 1: propiedades de un proyecto en Eclipse.

Luego en la sección “*Java Build Path*” hacer clic en “*Add External JARs*” y seleccionar la ubicación del archivo (Figura 2). Hacer clic en *Ok* y listo.

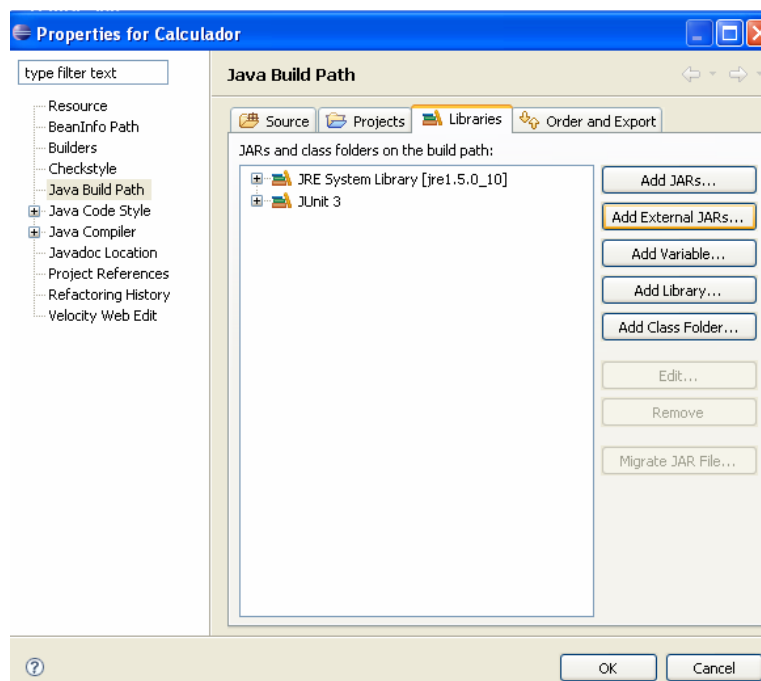


Figura 2: cómo agregar un archivo .jar a un proyecto en Eclipse.

### Cómo generar una instancia de la clase *Calculador*:

La siguiente línea de código permite generar una instancia de la clase *Calculador*.

```
Calculador calculador = new Calculador("Viborita");
```

La cadena pasada como parámetro representa el nombre asignado al grupo que será mostrado en la simulación.

### Cómo inicializar el mundo del agente *Víbora*:

Una vez generada la instancia se debe inicializar el mundo en el que se desenvuelve el agente *Víbora*. Para lograr esto:

```
Vector comida = calculador.getComida();
```

La variable *comida* es un vector que contiene elementos de tipo Pair (fila, columna) que representan las posiciones donde se encuentra la comida.

**IMPORTANTE!!** Los elementos de tipo Pair están compuestos por (filas, columnas), por lo tanto el método `par.x()` representa la fila y el método `par.y()` representa la columna. Las coordenadas de la grilla se muestran en la siguiente tabla:

1,1	1,2	1,3	1,4
2,1	2,2	2,3	2,4
3,1	3,2	3,3	3,4
4,1	4,2	4,3	4,4

### **Cómo obtener la posición inicial del agente Víbora:**

Para obtener la posición inicial se debe ejecutar el método `getPosicionInicial()`:

```
Pair pos = calculador.getPosicionInicial();
```

Este método retorna un Pair con la posición inicial del agente.

Otra opción puede ser la siguiente:

```
int posFila = calculador.getPosicionInicial().x();  
int posCol = calculador.getPosicionInicial().y();
```

### **Cómo reportar la acción emprendida por el agente para el cálculo de la performance:**

Luego de ejecutar una acción ésta debe ser informada de la siguiente manera, p.e.:

```
calculador.reportarAccion(Calculador.AVANZAR);
```

Este método debe ser invocado luego que la estrategia de búsqueda haya determinado la acción más adecuada para la situación actual. Con esta acción el calculador va calculando la performance del agente de cada grupo.

### **Cómo obtener la performance del agente Víbora:**

La performance debe ser obtenida al finalizar la ejecución del agente:

```
int performance = calculador.getPerformance();
```

Este método puede ser ejecutado sólo una vez mientras dure la instancia de la clase Calculador. En caso de ser ejecutado más de una vez retornará -1.