**Caso de Estudio: Conductor**

**Dado el siguiente modelo, codificar:**



**Ejemplo de Uso**

|  |
| --- |
| *public static void main(String args[]) {*  *int n =0;*  *System.out.println(“Cantidad de Censores”);*  *Conductor c = new Conductor(n);*  *c.avanzar();*  *}* |

**Tareas Para completar el modelo:**

* mover(). Abstracto. Recibe un argumento Integer, que indica según sea el caso, la velocidad de avance o de giro del robot, y modifica el valor del atributo respectivo.
* parar(). No recibe argumentos y pone velocidad en 0.
* buscarPaso(). No recibe argumentos. En caso de que alguno de los censores haya detectado un obstáculo activa parar() e incrementa el valor de dirección en 1. Si no hay obstáculos activa mover(), pasándole el equivalente a 2 como argumento. El método regresa el valor 2 como distancia avanzada.
* avanzar(). No recibe argumentos, ni regresa valor. Lo que hace es buscarPaso(), incrementando el valor de distRecorrida mientras sea menor a 100.
* detectarObstaculo(). No recibe argumentos. Regresa true si alguno de los censores detectó un obstáculo a menos de 2 mts, sino devuelve false.
* distancia(). No recibe argumentos. Regresa el valor de la distancia existente hasta un obstáculo. Simular con return (int(Math random() \* 10 ).
* La clase Giro no hace falta implementarla.
* La dirección y velocidad de los motores inicialmente están en 0.

**Dudas:**

En el método mover() , nos pide indicar la velocidad y modifica su valor , duda sobre dieseño del código:

public void mover(Integer n){

velocidad=this.velocidad+n;

}

-Duda sobre diseño del código del método buscarPaso() y detectarObstaculo()

-En el método distancia no sabia si era correcto el diseño que realice:

public int distancia(){

return (int)(Math.random()\*10);

}

-Duda sobre los parámetros que deberían recibir los constructores de Censor y Buscador

**Respuesta a la Duda, e Implementacion de algunas Clases:**

|  |
| --- |
| *class MotorAvance extends Motor{*  *public MotorAvance() //constructor*  *{*  *super(0); //activar constructor de Motor*  *}*  *public void mover(Integer veloc) {*  *velocidad = veloc;*  *}*  *}* |

|  |
| --- |
| *class Conductor {*  *private Buscador busca;*  *public final MotorAvance ctrlAvance;*  *prívate Integer direccion;*  *prívate int distRecorrida = 0;*  *public final MotorGiro ctrlGiro;*    *public Constructor( int cantidad) //constructor*  *{*  *busca = new Buscador(this, cant);*  *direccion = new Integer(0);*  *ctrlGiro = new MotorGiro();*  *ctrlAvance = new MotorAvance();*  *}*  *public void avanzar(){*  *while( distRecorrida < 100) {*  *distRecorrida += busca.buscarPaso();*  *}*  *}*    *//…..agregar métodos get y set para atributos privados*  *}* |

|  |
| --- |
| *class Buscador {*    *private Censor sensa[];*  *private Conductor conduce;*  *public Buscador(Conductor conductor, int n) {*  *censa = new Censor[ n ]; //utilizo constructor de array*  *conduce = condutor;*  *//inicializar arreglo*  *for(int i=0; i<censa.length; i++) {*  *censa[ I ] = new Censor( this ); //cada censor conoce su conductor*  *}*  *}*  *public int buscaPaso(){*  *if( detectarObstaculo() ) {*  *conduce.ctrlAvance.parar();*  *conduce.setDireccion( conduce.getDireccion() + 1 );*  *}*  *else {*  *conduce.ctrlAvance.mover( new Integer(2) );*  *}*  *return 2;*  *}*  *public boolean detectarObstaculo() {*  *boolean resultado = true; //suponemos primero que no hay problemas*    *for(int i=0; i<censa.length && resultado ; i++) {*  *if ( censa[k].distancia() < 2) //si la distancia es menor…*  *resultado = false;*  *else*  *resultado = true;*  *}*  *return resultado;*  *}*  *}* |

|  |
| --- |
| *class Censor {*  *prívate Buscador busca;*    *public Censor( Buscador buscador ){ //constructor.*  *busca = buscador; //el censor “conoce” al buscador*  *}*  *public int distancia(){*  *return (int) (Math.random() \* 10 ) ;*  *}*  *//El método estático random(), de la clase Math, genera un número aleatorio entre 0 y 1.*  *}* |

*Completar el resto…*