**Practical-1**

**Write a program to create a robot.**

1. **With Gear:**

**Code:**

package javaapplication1;

importch.aplu.robotsim.\*;

public class JavaApplication1 {

JavaApplication1()

{

NxtRobot robot=new NxtRobot();

Gear gear=new Gear();

robot.addPart(gear);

gear.forward(400);

gear.setSpeed(30);

gear.left(800);

gear.forward(2000);

gear.right(480);

robot.exit();

}

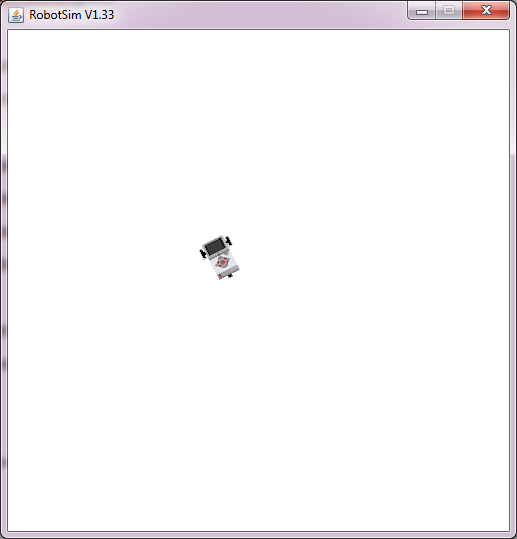
public static void main(String[] args) {

JavaApplication1 m=new JavaApplication1();

}

}

**Output:**



1. **Without Gear:**

**Code:**

packagemovewithoutgears;

importch.aplu.robotsim.\*;

public class MoveWithoutGears {

MoveWithoutGears()

{

TurtleRobot robot=new TurtleRobot();

robot.forward(100);

robot.left(45);

robot.forward(200);

robot.right(90);

robot.backward(100);

robot.exit();

}

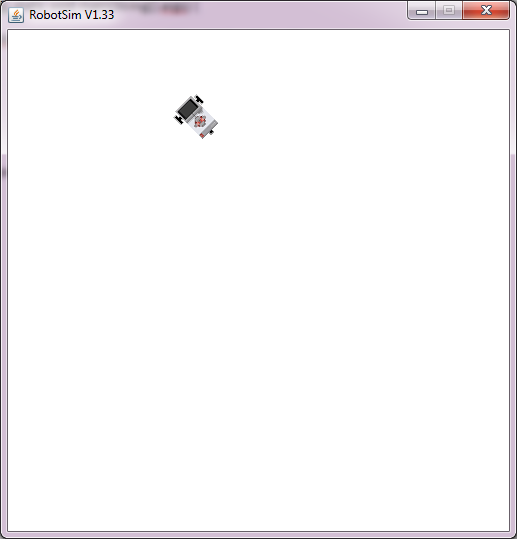
public static void main(String[] args) {

MoveWithoutGears m=new MoveWithoutGears();

}

}

**Output:**



**Practical-2**

**Write a program to create a robot with a two motor and move it forward, left, right**.

1. **Move with motors:**

**Code:**

importch.aplu.robotsim.\*;

public class MoveWithMotors

{

public MoveWithMotors()

{

NxtRobot robot=new NxtRobot();

Motor motA=new Motor(MotorPort.A);

Motor motB=new Motor(MotorPort.B);

robot.addPart(motA);

robot.addPart(motB);

motA.forward();

motB.forward();

Tools.delay(2000);

motA.stop();

Tools.delay(1050);

motA.forward();

Tools.delay(2000);

motB.stop();

Tools.delay(1050);

motB.forward();

Tools.delay(2000);

robot.exit();

}

public static void main(String args[])

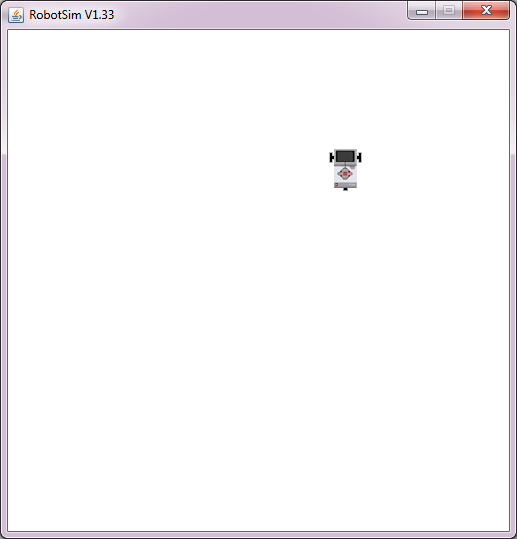
{

newMoveWithMotors();

}

}

**Output:**



**Practical-3**

**Write a program to do a square using a while loop, doing steps with a for loop, to change directions based on condition, controlling motor speed using switch case.**

**Code:**

importch.aplu.robotsim.\*;

class square

{

square()

{

NxtRobot r = new NxtRobot();

Gear g = new Gear();

r.addPart(g);

g.setSpeed(100);

g.forward(1000);

g.left(275);

g.forward(1000);

g.left(275);

g.forward(1000);

g.left(275);

g.forward(1000);

Tools.delay(2000);

r.exit();

}

public static void main(String[] args)

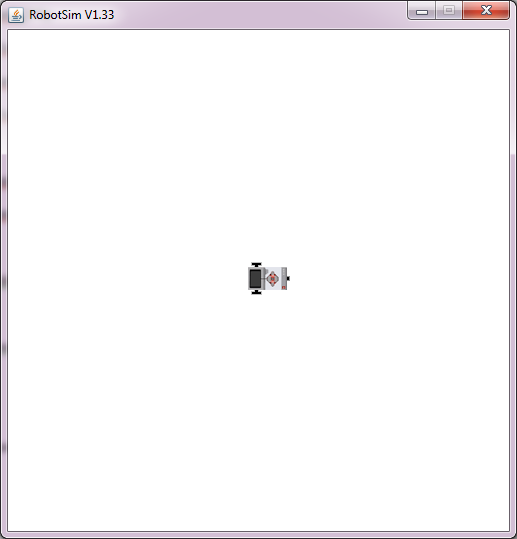
{

new square();

}

}

**Output:**



**Practical-4**

**Write a program to create a robot with light sensors to follow a line.**

**LineFollower:**

**Code:**

importch.aplu.robotsim.\*;

public class LineFollower

{

LineFollower()

{

LegoRobot robot=new LegoRobot();

Gear gear=new Gear();

LightSensorls=new LightSensor(SensorPort.S3);

robot.addPart(gear);

gear.setSpeed(20);

robot.addPart(ls);

while(true)

{

int v=ls.getValue();

if(v < 100)//black

gear.forward();

if(v > 300 && v < 750) //blue

gear.leftArc(0.05);

if(v> 800) //yellow

gear.rightArc(0.05);

}

}

public static void main(String args[])

{

newLineFollower();

}

static

{

RobotContext.setStartPosition(50,490);

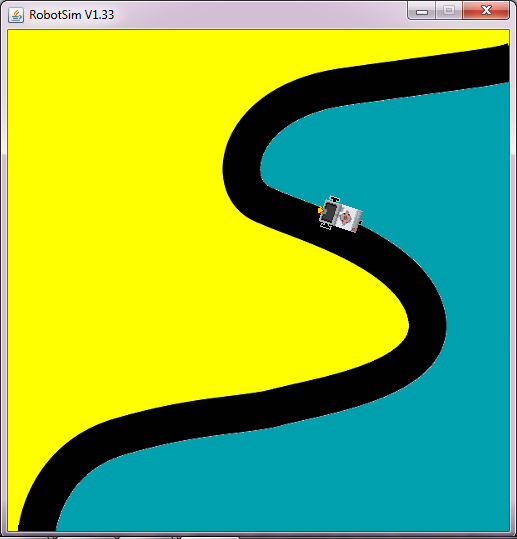
RobotContext.setStartDirection(-90);

RobotContext.useBackground("sprites/road.gif");

}

}

**Output:**



**Practical-5**

**Write a program to create a robot that does a circle using 2 motors.**

1. **Circlem:**

**Code:**

importch.aplu.robotsim.\*;

public class Circlem

{

Circlem()

{

NxtRobot robot=new NxtRobot();

Gear gear=new Gear();

robot.addPart(gear);

gear.setSpeed(60);

gear.leftArc(0.2,7000);

gear.rightArc(0.2);

Tools.delay(5000);

robot.exit();

}

public static void main(String args[])

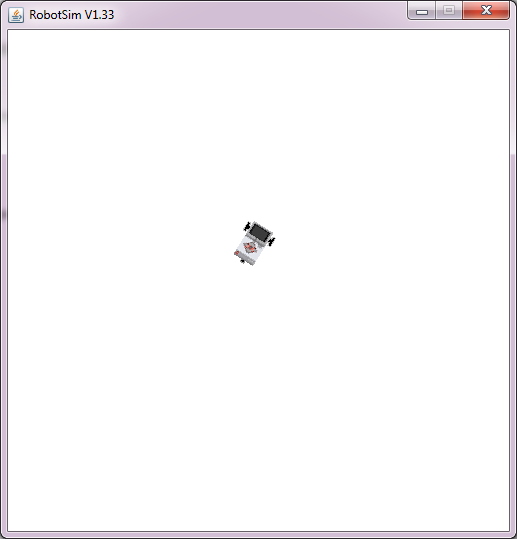
{

newCirclem();

}

}

**Output:**



1. **CircularGear:**

**Code:**

importch.aplu.robotsim.\*;

classCircularGear

{

CircularGear()

{

NxtRobot robot=new NxtRobot();

Gear gear=new Gear();

robot.addPart(gear);

gear.forward(200);

gear.setSpeed(20);

gear.leftArc(0.2,7000);

gear.forward(200);

gear.leftArc(0.2,7000);

gear.forward(200);

gear.leftArc(0.2,7000);

gear.forward(200);

gear.leftArc(0.2,7000);

gear.forward(200);

robot.exit();

}

public static void main(String args[])

{

CircularGear m=new CircularGear();

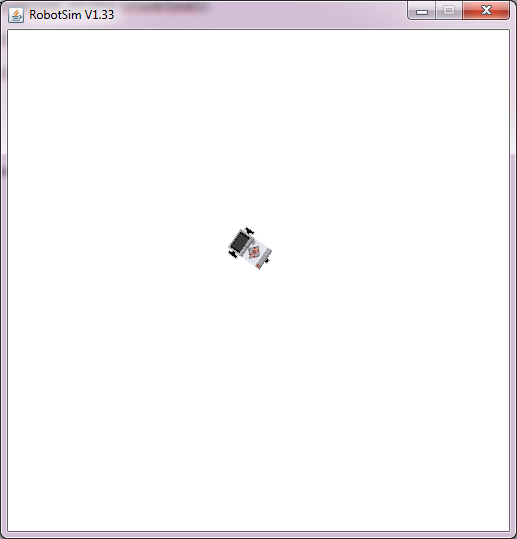
NxtContext.setStartPosition(250,200);

NxtContext.setStartDirection(90);

}

}

**Output:**



**Practical-6**

**Write a program to create a path following robot.**

**PathFinder:**

**Code:**

importch.aplu.robotsim.\*;

public class PathFinder

{

publicPathFinder()

{

NxtRobot robot=new NxtRobot();

Gear gear=new Gear();

LightSensor ls1=new LightSensor(SensorPort.S1);

LightSensor ls2=new LightSensor(SensorPort.S2);

robot.addPart(gear);

robot.addPart(ls1);

robot.addPart(ls2);

gear.forward();

while(true)

{

intrightValue=ls1.getValue();

intleftValue=ls2.getValue();

int d=rightValue - leftValue;

if(d>100)

gear.rightArc(0.1);

if(d < -100)

gear.leftArc(0.1);

if(d > -100 && d < 100 &&rightValue> 500)

gear.forward();

}

}

public static void main(String args[])

{

newPathFinder();

}

static

{

NxtContext.setStartPosition(250,490);

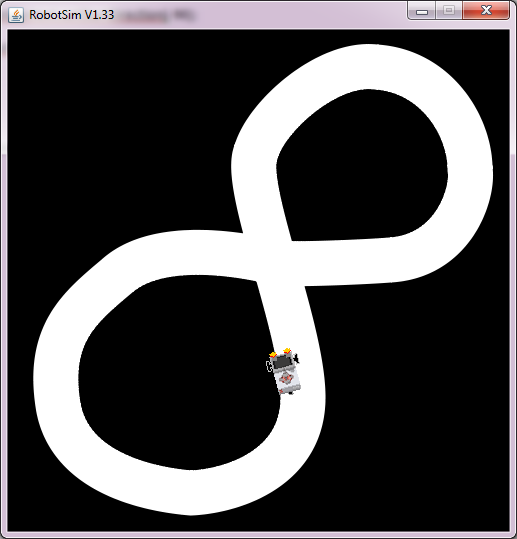
NxtContext.setStartDirection(-90);

NxtContext.useBackground("sprites/path.gif");

}

}

**Output**:



**Practical-7**

**Write a program to register obstacles**

**Resistobst:**

**Code:**

importch.aplu.robotsim.\*;

importch.aplu.util.\*;

public class resistobst

{

publicresistobst()

{

LegoRobot robot = new LegoRobot();

Gear g = new Gear();

TouchSensor ts1 = new TouchSensor(SensorPort.S1);

TouchSensor ts2 = new TouchSensor(SensorPort.S2);

robot.addPart(g);

robot.addPart(ts1);

robot.addPart(ts2);

g.forward();

while(!QuitPane.quit())

{

boolean t1 = ts1.isPressed();

boolean t2 = ts2.isPressed();

if(t1 && t2)

{

g.backward(500);

g.left(400);

g.forward();

}

else

{

if(t1)

{

g.backward(500);

g.left(400);

g.forward();

}

else

{

if(t2)

{

g.backward(500);

g.right(100);

g.forward();

}

}

}

Tools.delay(20);

}

robot.exit();

}

public static void main(String [] args)

{

newresistobst();

}

static

{

RobotContext.setLocation(10,10);

RobotContext.setStartDirection(5);

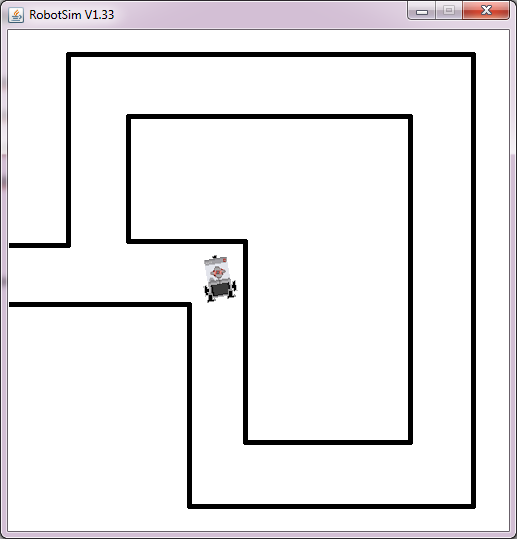
RobotContext.setStartPosition(100,240);

RobotContext.useObstacle(RobotContext.channel);

}

}

**Output:**



**Practical no 8**

**Write a program to implement Breadth First Search (BFS) algorithm for a given standard problem .**

import java.io.\*;

import java.io.BufferedReader;

class bfs

{

int i,j,k=0,l,n,sn;

int r[];

int a[][];

bfs() throws IOException

{

InputStreamReader f=new InputStreamReader(System.in);

BufferedReader br=new BufferedReader(f);

System.out.println("enter the no of nodes");

n=Integer.parseInt(br.readLine());

int a[][]=new int[n+1][n+1];

int r[]=new int[n+1];

System.out.println("enter the source node");

sn=Integer.parseInt(br.readLine());

// int e[]=new int[n];

System.out.println("enter the adjancey matrix");

for(i=1;i<=n;i++)

{

for(j=1;j<=n;j++)

{

a[i][j]=Integer.parseInt(br.readLine());

}

{

if(a[i][1]==1)

{

for(i=sn;i<=n;i++)

{

for(j=1;j<=n;j++)

{

if(a[i][j]==1)

{

if(k==0)

{r[k]=j;

k++;}

else

{

for(l=0;l<k;l++)

{

if(r[l]==j)

break;

}

if(l>=k)

{

r[k]=j;

k++;

}

System.out.print(r[k]);

}

}

}

}

System.out.println("bfs:");

for(i=0;i<k;i++)

{

System.out.print(r[i]+",");

}

}

public static void main(String args[])throws IOException

{

bfs p=new bfs();

}

}

**Output:**

enter the no of nodes

5

enter the root node

1

enter the nodes

1 2 3 4 5

enter the destination nodes

5

enter the adjancey matrix

0 1 1 0 0

1 0 0 1 0

1 0 0 0 1

0 1 0 0 0

0 0 1 0 0

adjancey matrix

1 2 3 4 5

1 0 1 1 0 0

2 1 0 0 1 0

3 1 0 0 0 1

4 0 1 0 0 0

5 0 0 1 0 0

bfs path=

1->1->2->3->4->5

**Practical-9**

**Write a program to implement Hill Climbing algorithm for a given standard problem.**

**Hillclimbing:**

**Code:**

importjava.util.\*;

classHillclimbing

{

String str[]=new String[2];

String s2="";

intno,i,j,min=1,temp,k=0;

String path[]=new String[50];

publicHillclimbing()

{

Scanner sr=new Scanner(System.in);

System.out.print("Enter the initial node:-\n");

str[0]=sr.next();

path[k]=str[0];

System.out.print("Enter the Goal Node:-\n");

str[1]=sr.next();

//Representing Cities in the tree format.

System.out.println("\t\t\tArad\n");

System.out.println("Zerind\t\t\tSibiu\t\t\tTimisoara\n");

System.out.println("\tOradea\tRimnicuVilceaFagaras\tLugoj\n");

System.out.println("\tCraiova Pitesti Bucharest\t Mehadia\n");

System.out.println(" Dobreta Pitesti Bucharest\t\t\tDobreta\n");

//Expand the appropriate node according to their minimum value.

while(min>0)

{

System.out.println(s2);

System.out.println("Enter the no. of node of" +" "+ str[0] );

no=sr.nextInt();

//System.out.println(no);

String st[][]=new String[no][2];

intarr[]=new int[no];

System.out.println("Enter the successor nodes of" +" "+ str[0]+" " +"With their straight line distance to goal node");

for(i=0;i<no;i++)

{

for(j=0;j<2;j++)

{

st[i][j]=sr.next();

}

}

//For Storing the values of various node.

for(i=0;i<no;i++)

{

arr[i]=Integer.parseInt(st[i][1]);

}

//To find the minimum value of various node.

for(i=0;i<no;i++)

{

for(j=i+1;j<no;j++)

{

if(arr[i]>arr[j])

{

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

}

min=arr[0];

//To find out the node name having minimum value.

for(i=0;i<no;i++)

{

if(Integer.parseInt(st[i][1])==min)

str[0]=st[i][0];

s2="Node"+" "+str[0]+" "+"has smallest value therefore ";

}

k++;

path[k]=str[0];

}

System.out.println("The path from initial node to goal node is:");

for(i=0;i<k;i++)

System.out.print(path[i]+"--->");

System.out.print(str[1]);

}

public static void main(String arg[])

{

Hillclimbing h=new Hillclimbing();

}

}

**Output:**

Enter the initial node:-

Arad

Enter the Goal Node:-

Bucharest

Arad

Zerind Sibiu Timisoara

Oradea RimnicuVilceaFagaras Lugoj

Craiova Pitesti Bucharest Mehadia

Dobreta Pitesti Bucharest Dobreta

Enter the no. of node of Arad

3

Enter the successor nodes of Arad With their straight line distance to goal node

Zerind 366

Sibiu 253

Timisoara 266

Node Sibiu has smallest value therefore

Enter the no. of node of Sibiu

3

Enter the successor nodes of Sibiu With their straight line distance to goal node

oradea 380

Rimnicu 180

fagaras 120

Node fagaras has smallest value therefore

Enter the no. of node of fagaras

1

Enter the successor nodes of fagarasWith their straight line distance to goal node

bucharest 0

The path from initial node to goal node is:

Arad--->Sibiu--->fagaras--->BucharestBUILD SUCCESSFUL (total time: 2 minutes 36 seconds)

**Practical-10**

**Write a program to implement A\* search algorithm for a given standard problem.**

**Astar:**

**Code:**

importjava.util.\*;

import java.io.\*;

classAstar

{

String str[]=new String[2];

String s1,s2="";

intno,i,j,min=1,temp,k=0;

String path[]=new String[50];

publicAstar()

{

Scanner sr=new Scanner(System.in);

System.out.print("Enter the initial node:-\n");

str[0]=sr.next();

path[k]=str[0];

System.out.print("Enter the Goal Node:-\n");

str[1]=sr.next();

//Representing Cities in the tree format.

System.out.println("\t\t\tArad\n");

System.out.println("Zerind\t\t\tSibiu\t\t\tTimisoara\n");

System.out.println("\tOradea\tRimnicuVilceaFagaras\tLugoj\n");

System.out.println("\tCraiova Pitesti Bucharest\t Mehadia\n");

System.out.println(" Dobreta Pitesti Bucharest\t\t\tDobreta\n");

//Expand the appropriate node according to their minimum value.

while(!str[0].equalsIgnoreCase(str[1]))

{

System.out.print(s2);

System.out.println("Enter the no. of node of" +" "+ str[0] );

no=sr.nextInt();

String st[][]=new String[no][3];

intarr[]=new int[no];

System.out.println("Enter the successor nodes of" +" "+ str[0]+" " +"With their straight line distance to goal node h(n) and path cost from start node i.e. h(n) ");

for(i=0;i<no;i++)

{

for(j=0;j<3;j++)

{

st[i][j]=sr.next();

}

}

//For Storing the values h(n) & f(n) of various node.

for(i=0;i<no;i++)

{

arr[i]=Integer.parseInt(st[i][1])+Integer.parseInt(st[i][2]);

}

//To find the minimum value of various node.

for(i=0;i<no;i++)

{

for(j=i+1;j<no;j++)

{

if(arr[i]>arr[j])

{

temp=arr[i];

arr[i]=arr[j];

arr[j]=temp;

}

}

}

min=arr[0];

//To find out the node name having minimum value.

for(i=0;i<no;i++)

{

if((Integer.parseInt(st[i][1])+Integer.parseInt(st[i][2]))==min)

str[0]=st[i][0];

s2="Node"+" "+str[0]+" "+"has smallest value therefore ";

}

k++;

path[k]=str[0];

}

System.out.println("The path from initial node to goal node is:");

for(i=0;i<k;i++)

System.out.print(path[i]+"--->");

System.out.print(str[1]);

}

public static void main(String arg[])

{

Astar a=new Astar();

}

}

**Output:**

run:

Enter the initial node:-

Arad

Enter the Goal Node:-

Bucharest

Arad

Zerind Sibiu Timisoara

Oradea RimnicuVilceaFagaras Lugoj

Craiova Pitesti Bucharest Mehadia

Dobreta Pitesti Bucharest Dobreta

Enter the no. of node of Arad

3

Enter the successor nodes of Arad With their straight line distance to goal node h(n) and path cost from start node i.e. h(n)

Sibiu 253 140

Zerind 374 45

Timisoara 329 118

Node Sibiu has smallest value therefore Enter the no. of node of Sibiu

3

Enter the successor nodes of Sibiu With their straight line distance to goal node h(n) and path cost from start node i.e. h(n)

Fagaras 176 239

Oradea 380 291

RimnicuVilcea 193 220

Node RimnicuVilcea has smallest value therefore Enter the no. of node of RimnicuVilcea

2

Enter the successor nodes of RimnicuVilcea With their straight line distance to goal node h(n) and path cost from start node i.e. h(n)

Craiova 366 160

Pitesti 317 100

Node Pitesti has smallest value therefore Enter the no. of node of Pitesti

2

Enter the successor nodes of Pitesti With their straight line distance to goal node h(n) and path cost from start node i.e. h(n)

bucharest 100 10

dobreta 125 1120

The path from initial node to goal node is:

Arad--->Sibiu--->RimnicuVilcea--->Pitesti--->BucharestBUILD SUCCESSFUL (total time: 2 minutes 49 seconds)