

**PRACTICALS**

**Practical No.: 1a**

**AIM:** Write aprogram tocreate a robot with gear and move it forward, left and right.

**DESCRIPTION:**

NxtRobot() - Constructor for class ch.aplu.robotsim.NxtRobot

Gear() - Constructor for class ch.aplu.robotsim.Gear

Creates a gear instance with right motor plugged into port A, left motor plugged into port B.

addPart(Part) - Method in class ch.aplu.robotsim.LegoRobot

Assembles the given part into the robot.

setSpeed(int) - Method in class ch.aplu.robotsim.Gear

Sets the speed to the given value (arbitrary units).

forward() - Method in class ch.aplu.robotsim.Gear

Starts the forward movement.

left() - Method in class ch.aplu.robotsim.Gear

Starts to rotate left (center of rotation at middle of the wheel axes).

right() - Method in class ch.aplu.robotsim.Gear

Starts to rotate right (center of rotation at middle of the wheel axes).

**CODE:**

package robotics;

import ch.aplu.robotsim.\*;

public class movement {

movement(){

TurtleRobot t = new TurtleRobot();

Gear g = new Gear();

t.addPart(g);

g.forward(400);

g.left(180);

g.forward(400);

g.left(180);

g.forward(400);

g.right(180);

g.forward(500);

g.right(180);

g.forward(400);

t.exit();

}

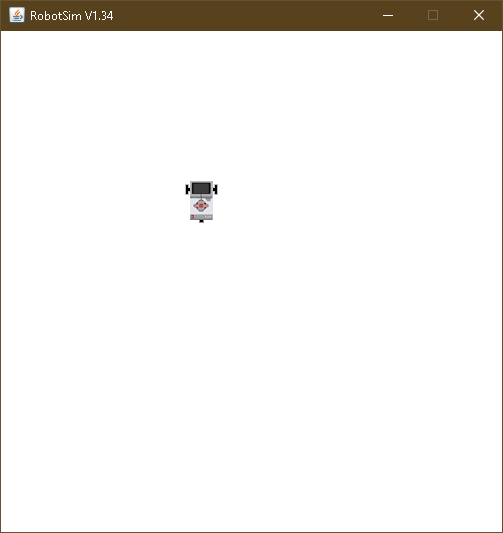
public static void main(String[] args){

new movement();

}

}

**OUTPUT:**

****

**Practical No.: 1b**

**AIM:** Writea program to create a robot without gear and move it forward, left and right.

**DESCRIPTION:**

TurtleRobot() - Constructor for class ch.aplu.robotsim.TurtleRobot

Creates a turtle robot instance.

**CODE:**

package robotics;

import ch.aplu.robotsim.\*;

public class withoutGear {

withoutGear(){

TurtleRobot t = new TurtleRobot();

t.forward(100);

t.left(90);

t.forward(100);

t.right(90);

t.forward(100);

}

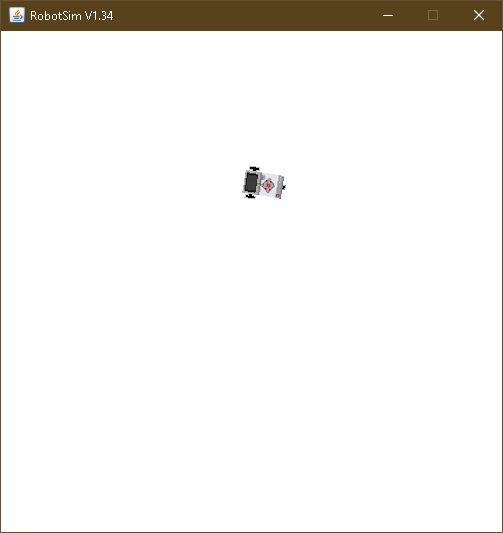
public static void main(String[] args){

new withoutGear();

}

}

**OUTPUT:**

****

**Practical No.: 2**

**AIM:** Write a program to create a robot with 2 motors and move it forward, left and right.

**DESCRIPTION:**

Motor - Class in ch.aplu.robotsim

Class that represents one of the NXT motors.

Motor(MotorPort) - Constructor for class ch.aplu.robotsim.Motor

Creates a motor instance that is plugged into given port.

Tools() - Constructor for class ch.aplu.robotsim.Tools

delay(int) - Static method in class ch.aplu.robotsim.Tools

Suspends execution of the current thread for the given amount of time (unless the game grid window is disposed).

stop() - Method in class ch.aplu.robotsim.Motor

Stops the rotation.

**CODE:**

package robotics;

import ch.aplu.robotsim.\*;

public class MoveWithMotors {

MoveWithMotors(){

NxtRobot r=new NxtRobot();

Motor m1=new Motor(MotorPort.A);

Motor m2=new Motor(MotorPort.B);

r.addPart(m1);

r.addPart(m2);

m1.forward();

Tools.delay(1000);

m2.forward();

m1.stop();

Tools.delay(200);

m1.forward();

}

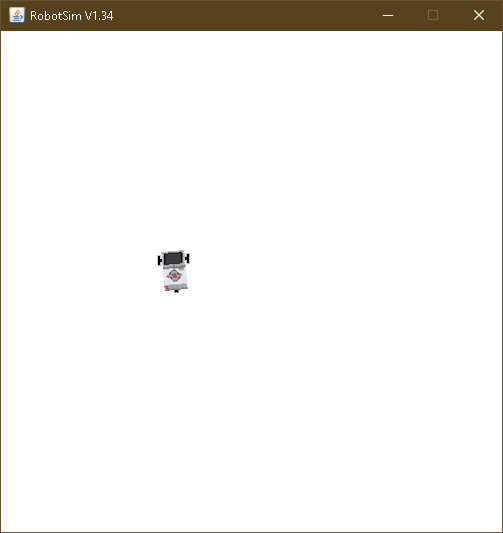
public static void main(String[] args){

new MoveWithMotors();

}

}

**OUTPUT:**

****

**Practical No.: 3**

**AIM:** Write a program to do a square using a while loop.

**CODE:**

package robotics;

import ch.aplu.robotsim.\*;

public class movementSquare {

public movementSquare(){

NxtRobot robot = new NxtRobot();

Gear g = new Gear();

robot.addPart(g);

g.setSpeed(50);

while (true){

g.forward(600);

g.left(280);

}

}

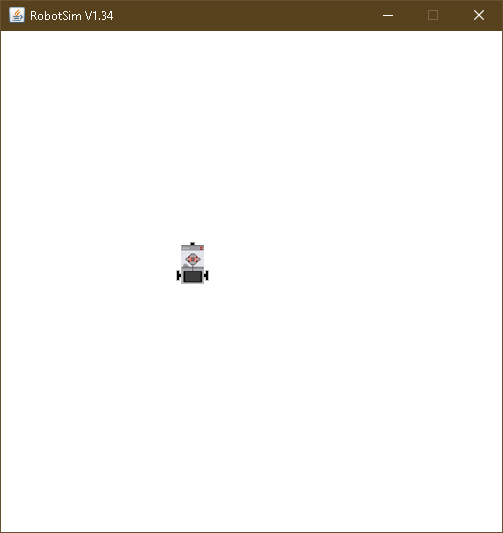
public static void main(String[] args){

new movementSquare();

}

}

**OUTPUT:**

****

**Practical No.: 4**

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

**DESCRIPTION:**

***RobotContext()*** - Constructor for class ch.aplu.robotsim.RobotContext

Creates a RobotContext instance.

***setStartPosition(int, int)*** - Static method in class ch.aplu.robotsim.RobotContext

Sets the Nxt starting position (x-y-coordinates 0..500, origin at upper left).

***useBackground(String)*** - Static method in class ch.aplu.robotsim.RobotContext

Use the given image as background (playground size 501 x 501).

***LegoRobot()*** - Constructor for class ch.aplu.robotsim.LegoRobot

Creates a robot with its playground using defaults from RobotContext.

***LightSensor(SensorPort)*** - Constructor for class ch.aplu.robotsim.LightSensor

Creates a sensor instance pointing downwards connected to the given port.

***getValue()*** - Method in class ch.aplu.robotsim.LightSensor

For sensor ports 1, 2, 3, 4: returns the brightness of the background at the current location.

***leftArc(double)*** - Method in class ch.aplu.robotsim.Gear

Starts to move to the left on an arc with given radius.

***rightArc(double)*** - Method in class ch.aplu.robotsim.Gear

Starts to move to the right on an arc with given radius.

**CODE:**

package robotics;

import ch.aplu.robotsim.\*;

public class LineFollower {

LineFollower(){

LegoRobot r =new LegoRobot();

Gear g = new Gear();

LightSensor ls = new LightSensor(SensorPort.S3);

r.addPart(g);

r.addPart(ls);

g.forward();

g.setSpeed(90);

while(true){

int v = ls.getValue();

if(v < 100){

g.forward();

}

if(v>300 && v<750){

g.leftArc(0.05);

}

if(v>800){

g.rightArc(0.05);

}

}

}

static{

RobotContext.setStartPosition(50,470);

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

}

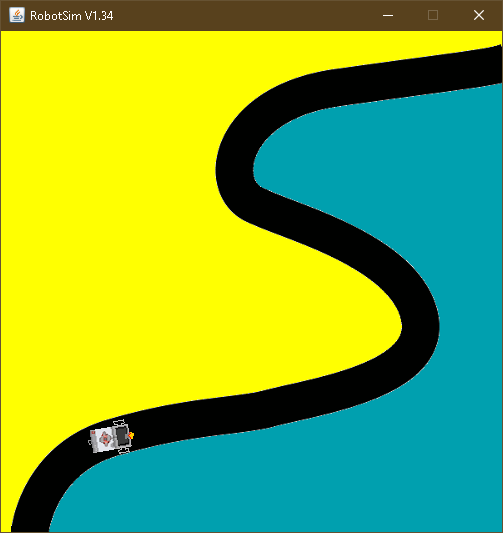
public static void main(String[] args) {

new LineFollower();

}

}

**OUTPUT:**

****

**Practical No.: 5**

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

**CODE:**

package robotics;

import ch.aplu.robotsim.\*;

public class RobotMotorCircle {

RobotMotorCircle(){

NxtRobot r = new NxtRobot();

Motor A = new Motor(MotorPort.A);

Motor B = new Motor(MotorPort.B);

r.addPart(B);

r.addPart(A);

A.setSpeed(100);

B.setSpeed(100);

A.forward();

B.forward();

while (true){

Tools.delay(200);

A.stop();

Tools.delay(200);

A.forward();

}

}

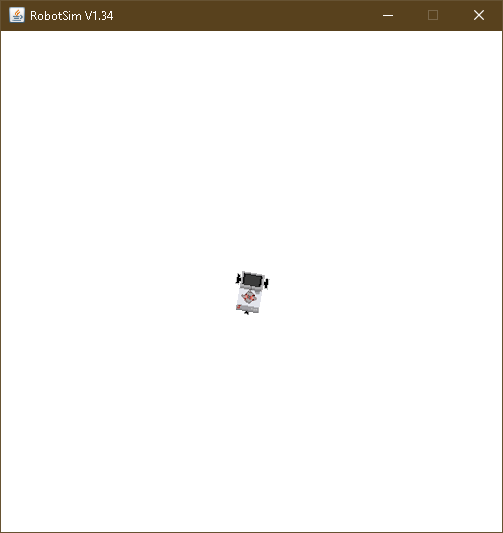
public static void main(String arg[]) {

new RobotMotorCircle();

}

}

**OUTPUT:**

****

**Practical No.: 6**

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

**DESCRIPTION:**

***NxtContext()*** - Constructor for class ch.aplu.robotsim.NxtContext

***setStartDirection(double)*** - Static method in class ch.aplu.robotsim.RobotContext

Sets the Nxt starting direction (zero to EAST).

**CODE:**

import ch.aplu.robotsim.\*;

public class PathFollowingRobot {

PathFollowingRobot(){

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();

gear.setSpeed(100);

while(true)

{

int rightValue=ls1.getValue();

int leftValue=ls2.getValue();

if(leftValue< 10)

gear.rightArc(0.05);

if(rightValue< 10)

gear.leftArc(0.05);

if(leftValue> 10 &&rightValue> 10)

gear.forward();

}

}

public static void main(String args[])

{

new PathFollowingRobot();

}

static

{

NxtContext.setStartPosition(267,232);

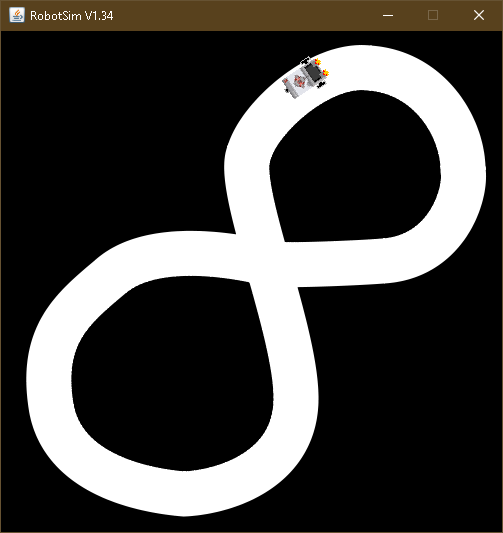
NxtContext.setStartDirection(-90);

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

}

}

**OUTPUT:**

****

**Practical No.: 7**

**AIM:** Write a program to resist obstacles.

**DESCRIPTION:**

***TouchSensor(SensorPort)*** - Constructor for class ch.aplu.robotsim.TouchSensor

Creates a sensor instance connected to the given port.

***isPressed()*** - Method in class ch.aplu.robotsim.TouchSensor

Polls the touch sensor and returns true, if there is a collision with any of the collision obstacles.

***backward()*** - Method in class ch.aplu.robotsim.TurtleRobot

Starts moving backward and returns immediately.

***useObstacle(Obstacle)*** - Static method in class ch.aplu.robotsim.RobotContext

Defines the given obstacle to be used as touch obstacle.

***channel*** - Static variable in class ch.aplu.robotsim.RobotContext

**CODE:**

package robotics;

import ch.aplu.robotsim.\*;

public class Obstacles {

Obstacles(){

LegoRobot r=new LegoRobot();

Gear g = new Gear();

TouchSensor t1= new TouchSensor(SensorPort.S1);

TouchSensor t2 = new TouchSensor(SensorPort.S2);

r.addPart(g);

r.addPart(t1);

r.addPart(t2);

g.forward();

g.setSpeed(50);

while(true){

Boolean b1 = t1.isPressed();

Boolean b2 = t2.isPressed();

if(b1 && b2){

g.backward(150);

g.right(400);

g.forward();

}

if(b1){

g.backward(150);

g.left(200);

g.forward();

}

if(b2){

g.backward(150);

g.right(200);

g.forward();

}

}

}

static {

RobotContext.setStartPosition(100,250);

RobotContext.useObstacle(RobotContext.channel);

}

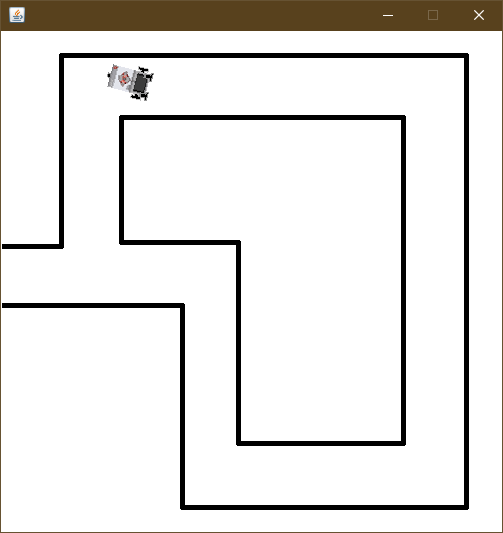
public static void main(String args[]){

new Obstacles();

}

}

**OUTPUT:**

****

**Practical No.: 8**

**AIM:** ULTRASONIC SENSOR.

**DESCRIPTION:**

UltrasonicSensor(SensorPort) - Constructor for class ch.aplu.robotsim.UltrasonicSensor

The port selection determines the position of the sensor and the direction of the beam axis.

setBeamAreaColor(Color) - Method in class ch.aplu.robotsim.UltrasonicSensor

Sets the color of the beam area (two sector border lines and axis).

setProximityCircleColor(Color) - Method in class ch.aplu.robotsim.UltrasonicSensor

Sets the color of the circle with center at sensor location and radius equals to the current distance value.

getDistance() - Method in class ch.aplu.robotsim.UltrasonicSensor

Returns the distance to the nearest target object.

useTarget(String, Point[], int, int) - Static method in class ch.aplu.robotsim.RobotContext

Creates a target for the ultrasonic sensor using the given sprite image.

**CODE:**

import ch.aplu.robotsim.\*;

import java.awt.Color;

import java.awt.Point;

public class Practical\_8 {

Practical\_8() {

LegoRobot robot = new LegoRobot();

Gear gear = new Gear();

robot.addPart(gear);

UltrasonicSensor us = new UltrasonicSensor(SensorPort.S1);

robot.addPart(us);

us.setBeamAreaColor(Color.green);

us.setProximityCircleColor(Color.lightGray);

double arc = 0.5;

gear.setSpeed(50);

gear.rightArc(arc);

booleanisRightArc = true;

int oldDistance = 0;

while (true)

{

Tools.delay(100);

int distance = us.getDistance();

if (distance == -1)

continue;

if (distance <oldDistance)

{

if (isRightArc)

{

gear.leftArc(arc);

isRightArc = false;

}

else

{

gear.rightArc(arc);

isRightArc = true;

}

}

oldDistance = distance;

}

}

static{

Point[] mesh\_bar =

{

new Point(10, 200), new Point(-10, 200),

new Point(-10, -200), new Point(10, -200)

};

RobotContext.useTarget("sprites/bar1.gif", mesh\_bar, 200, 250);

RobotContext.useTarget("sprites/bar1.gif", mesh\_bar, 300, 250);

RobotContext.setStartPosition(250, 460);

}

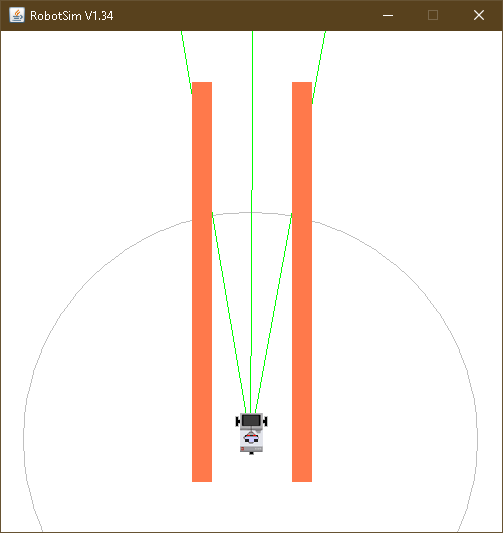
public static void main(String[] args) {

new Practical\_8();

}

}

**OUTPUT:**

****

**ASSIGNMENTS**

**Assignment 1**

**AIM:**Write a program to make use of track image and move in white area.

**CODE:**

package robotics;

import ch.aplu.robotsim.\*;

public class RobotSensorTrackFollower {

static {

RobotContext.setStartPosition(80, 438);

RobotContext.useBackground("sprites/track.png");

}

public RobotSensorTrackFollower() {

LegoRobotlegoRobot = new LegoRobot();

Gear gearBox = new Gear();

LightSensorlightSensor = new LightSensor(SensorPort.S3);

legoRobot.addPart(gearBox);

legoRobot.addPart(lightSensor);

gearBox.forward();

gearBox.setSpeed(100);

while (true) {

if(lightSensor.getValue() > 10){

gearBox.forward();

}

else{

gearBox.rightArc(0.03);

}

}

}

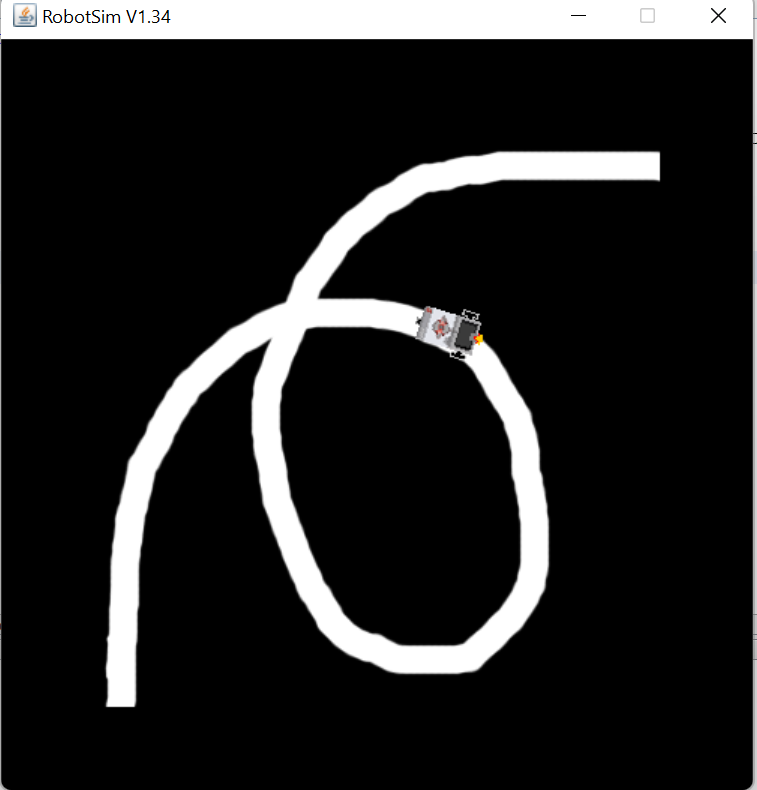
public static void main(String[] args) {

new RobotSensorTrackFollower();

}

}

**OUTPUT:**



**Assignment 2**

**AIM:**Write a program to create a robot with gear that does a circle.

**CODE:**

package robotics;

import ch.aplu.robotsim.\*;

public class MoveWithGearCircle {

public MoveWithGearCircle(){

NxtRobot robot = new NxtRobot();

Gear g = new Gear();

robot.addPart(g);

for(int i =1;i!=0;i++){

g.forward(200);

g.right(200);

}

}

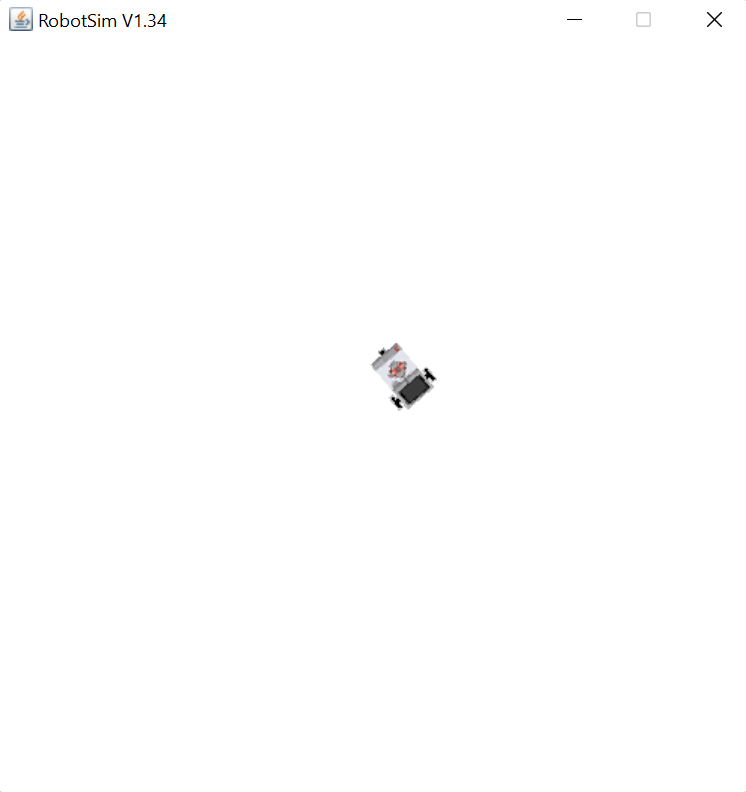
public static void main(String []args){

new MoveWithGearCircle();

}

}

**OUTPUT:**



**Assignment 3**

**AIM:**Write a program to create a robot that does a rectangle.

**CODE:**

package robotics;

import ch.aplu.robotsim.\*;

public class MoveWithGearRect {

public MoveWithGearRect(){

NxtRobot robot = new NxtRobot();

Gear g = new Gear();

robot.addPart(g);

g.right(550);

g.forward(2500);

g.left(550);

g.forward(1000);

g.left(550);

g.forward(2500);

g.left(550);

g.forward(1000);

robot.exit();

}

public static void main (String [] args){

new MoveWithGearRect();

}

}

**OUTPUT:**

