class Snake  
{  
private BoardSquare tail;  
Snake(BoardSquare head, BoardSquare tail)  
{  
setTail(tail);  
head.addSnake(this);  
}  
private void setTail(BoardSquare tail)  
{  
this.tail = tail;  
}  
private BoardSquare getTail()  
{  
return tail;  
}  
void movePlayerPiece(PlayerPiece counter)  
{  
System.out.println("Down the snake to " +  
getTail().getPosition());  
counter.setCurrentPosition(getTail());  
}  
}

class Ladder  
{  
private BoardSquare top;  
Ladder(BoardSquare top, BoardSquare foot)  
{  
setTop(top);  
foot.addLadder(this);  
}  
private void setTop(BoardSquare top)  
{  
this.top = top;  
}  
private BoardSquare getTop()  
{  
return top;  
}

void movePlayerPiece(PlayerPiece counter)  
{  
System.out.println("Up the ladder to " +  
getTop().getPosition());  
counter.setCurrentPosition(getTop());  
}  
}

class PlayerPiece  
{  
private BoardSquare currentPosition;  
private String color;  
PlayerPiece(String color)  
{  
setColor(color);  
}  
private void setColor(String color)  
{  
this.color = color;  
}

String getColor()  
{  
return color;  
}  
BoardSquare getCurrentPosition()  
{  
return currentPosition;  
}  
void moveTo(BoardSquare newPosition)  
{  
newPosition.movePlayerPiece(this);  
}  
void setCurrentPosition(BoardSquare newPosition)  
{  
currentPosition = newPosition;  
}  
}

class BoardSquare  
{  
// we may use one or neither of these references  
// for a particular square  
private Snake aSnake = null;  
private Ladder aLadder = null;  
private int position;  
BoardSquare(int position) {  
setPosition(position);  
}  
int getPosition() {  
return position;  
}  
private void setPosition(int position) {  
this.position = position;  
}  
// we may want to add a snake head  
void addSnake(Snake s) {  
aSnake = s;  
}  
// or add the foot of a ladder  
public void addLadder(Ladder l) {  
aLadder = l;  
}  
// methods to find out if the square has a snake or a ladder  
private boolean hasSnake() {  
return null != aSnake;  
}  
private boolean hasLadder() {  
return null != aLadder;  
}  
public void movePlayerPiece(PlayerPiece counter) {  
counter.setCurrentPosition(this);  
if (hasSnake()) {  
aSnake.movePlayerPiece(counter);  
}  
if (hasLadder()) {  
aLadder.movePlayerPiece(counter);  
}  
}  
}

class GameBoard  
{

private BoardSquare[] squares;  
private Die die;

// the array will be one square bigger than needed so that we  
// can start from array element 1, ignoring element 0  
static final int START\_SQUARE = 1;

static final int MAX\_SQUARES = 100;

// snakes and ladders  
GameBoard(){  
die = new Die();  
squares = new BoardSquare[START\_SQUARE + MAX\_SQUARES];  
for (int i = START\_SQUARE; i <= MAX\_SQUARES; i++){  
// add the next Square object to the board  
squares[i] = new BoardSquare(i);  
}  
// add the ladders  
new Ladder(squares[38], squares[1]);  
new Ladder(squares[14], squares[4]);  
new Ladder(squares[31], squares[9]);  
new Ladder(squares[42], squares[21]);  
new Ladder(squares[84], squares[28]);  
new Ladder(squares[44], squares[36]);  
new Ladder(squares[67], squares[51]);  
new Ladder(squares[91], squares[71]);  
new Ladder(squares[100], squares[80]);  
// add the snakes  
new Snake(squares[16], squares[6]);  
new Snake(squares[47], squares[26]);  
new Snake(squares[49], squares[11]);  
new Snake(squares[56], squares[53]);  
new Snake(squares[62], squares[19]);  
new Snake(squares[64], squares[60]);  
new Snake(squares[87], squares[24]);  
new Snake(squares[93], squares[73]);  
new Snake(squares[95], squares[75]);  
new Snake(squares[98], squares[78]);  
}  
// the constructor creates the squares and adds the  
BoardSquare getStartSquare()  
{  
return squares[START\_SQUARE];  
}  
// this method adjusts the counter position  
void movePlayerPiece(PlayerPiece counter)  
{  
BoardSquare current = counter.getCurrentPosition();  
int nextPosition = current.getPosition() + die.getRoll();  
if(nextPosition > MAX\_SQUARES)  
{  
System.out.println(  
"Sorry you need to land exactly on the last square to win!");  
}  
else

{  
counter.moveTo(squares[nextPosition]);  
}  
System.out.println(counter.getColor() + " counter on " +  
counter.getCurrentPosition().getPosition());  
}  
}

public class SnakesAndLadders  
{  
// reference to the GameBoard  
private GameBoard board;  
// the constructor creates the Board  
public SnakesAndLadders()  
{  
board = new GameBoard();  
}  
// this method acts as a controller for playing the game  
public void play()  
{  
PlayerPiece counter = new PlayerPiece("Red");  
counter.setCurrentPosition(board.getStartSquare());  
// iterate until we reach the end (square 100)  
while(counter.getCurrentPosition().getPosition()  
< GameBoard.MAX\_SQUARES)  
{  
board.movePlayerPiece(counter);  
}  
System.out.println(counter.getColor() +  
" counter finished on " +  
counter.getCurrentPosition().getPosition());  
}  
// 'main' creates a 'SnakesAndLadders' object and  
// starts the game  
public static void main(String[] args)  
{  
SnakesAndLadders myGame = new SnakesAndLadders();  
myGame.play();  
}  
}