Requerimientos

Nombre:	RF1: Verificar si el orden es impar
Resumen:	El programa deberá verificar que el valor ingresado como orden del cuadrado, sea un número impar
Entradas:	-
Resultados	boolean

Nombre:	RF2: Calcular la constante
Resumen:	El programa deberá calcular la constante del cuadrado mágico
Entradas:	-
Resultados	int

Nombre:	RF3: Generar número 1
Resumen:	El programa deberá generar el número 1, en la posición del cuadrado mágico que sea ingresada
Entradas:	String
Resultados	void

Nombre:	RF4: Recorrer cuadrado mágico noreste, posición arriba
Resumen:	El programa deberá llenar el cuadrado mágico cuando el número 1 esté ubicado arriba, siguiendo el patrón arriba a la derecha
Entradas:	-
Resultados	Void

Nombre:	RF5: Recorrer cuadrado mágico noreste, posición derecha
Resumen:	El programa deberá llenar el cuadrado mágico cuando el número 1 esté ubicado en la derecha, siguiendo el patrón arriba a la derecha
Entradas:	-
Resultados	void

Nombre:	RF6: Recorrer cuadrado mágico noroeste, posición arriba
Resumen:	El programa deberá llenar el cuadrado mágico cuando el número 1 esté ubicado arriba, siguiendo el patrón arriba a la izquierda
Entradas:	-
Resultados	void

Nombre:	RF7: Recorrer cuadrado mágico noroeste, posición izquierda
Resumen:	El programa deberá llenar el cuadrado mágico cuando el número 1 esté ubicado a la izquierda, siguiendo el patrón arriba a la izquierda
Entradas:	-
Resultados	void

Nombre:	RF8: Recorrer cuadrado mágico sureste, posición abajo
Resumen:	El programa deberá llenar el cuadrado mágico cuando el número 1 esté ubicado abajo, siguiendo el patrón abajo a la derecha
Entradas:	-
Resultados	Void

Nombre:	RF9: Recorrer cuadrado mágico sureste, posición derecha
Resumen:	El programa deberá llenar el cuadrado mágico cuando el número 1 esté ubicado a la derecha, siguiendo el patrón abajo a la derecha
Entradas:	-
Resultados	Void

Nombre:	RF10: Recorrer cuadrado mágico suroeste, posición abajo
Resumen:	El programa deberá llenar el cuadrado mágico cuando el número 1 esté ubicado abajo, siguiendo el patrón abajo a la izquierda
Entradas:	-
Resultados	void

Nombre:	RF11: Recorrer cuadrado mágico suroeste, posición izquierda
Resumen:	El programa deberá llenar el cuadrado mágico cuando el número 1 esté ubicado abajo, siguiendo el patrón abajo a la derecha
Entradas:	-
Resultados	void

Nombre:	RF12: Sumar filas
Resumen:	El programa deberá sumar todos los valores de las filas y verificar si el resultado es igual a la constante
Entradas:	int
Resultados	boolean

Nombre:	RF13: Sumar columnas
Resumen:	El programa deberá sumar todos los valores de las columnas y verificar si el resultado es igual a la constante
Entradas:	Int
Resultados	boolean

Nombre:	RF14: Sumar diagonales
Resumen:	El programa deberá sumar todos los valores de las diagonales y verificar si el resultado es igual a la constante
Entradas:	int
Resultados	boolean

Nombre:	RF15: Verificar si es cuadrado mágico
Resumen:	El programa deberá verificar si el cuadrado es cuadrado mágico
Entradas:	boolean, boolean, boolean
Resultados	boolean

Nombre:	RNF1: Implementar en JavaFX
Resumen:	El programa deberá implementarse en JavaFX

Nombre:	RNF2: Implementar tamaño
Resumen:	El programa deberá tener un tamaño de 600 x 400

Trazabilidad

Requerimiento	Método
RF1: Verificar si el orden es impar	`public NotOddException(int number) { super("El orden de la matrix no es impar"); order = number; isOdd(number); } public boolean isOdd(int order) { boolean odd = true; if (order%2 == 1) { odd = true;`

	} else {
	odd = false;
	}
	
	return odd;
	}
	
	public void setOrder(int order) throws NotOddException, NumberFormatException{
	if (order%2!=1) {
	throw new NotOddException(order);
	} else {
	this.order = order;
	}
	
	}

RF2: Calcular la constante	public int findConstant() {
	int constant;
	
	constant = (order*((order*order) + 1)) /2;
	
	return constant;
	}

RF3: Generar número 1	public void generateNumberOne(String option) {
	int position;
	position = order/2;
	switch (option) {
	case "Arriba":
	
	matrix[0][position] = 1;
	break;
	case "Abajo":
	
	matrix[matrix.length-1][position] = 1;
	break;
	case "Derecha":
	
	matrix[position][matrix.length-1] = 1;
	break;
	case "Izquierda":
	
	matrix[position][0] = 1;
	break;
	default:
	break;
	}

	}

RF4: Recorrer cuadrado mágico noreste, posición arriba	public void northeastUp() {
	int lastCol = 0;
	int lastRow = 0;
	int nextRow = 0;
	int nextCol = order/2;
	int one = 1;
	
	for (int i = 0; i < order; i++){
	for (int j = 0; j < order; j++){
	matrix[nextRow][nextCol] = one;
	
	lastCol = nextCol;
	lastRow= nextRow;
	
	nextRow = lastRow-1;
	nextCol = lastCol+1;
	
	if(nextRow < 0){
	nextRow = order - 1;
	}
	if(nextCol >= order){
	nextRow = lastRow - 1;
	nextCol = 0;
	}
	
	if(nextRow == -1 ‖
	matrix[nextRow][nextCol] != 0){
	nextRow = lastRow + 1;
	nextCol = lastCol ;
	}
	one++;
	}
	}
	
	}

RF5: Recorrer cuadrado mágico noreste, posición derecha	public void northeastRight() {
	int lastCol = 0;
	int lastRow = 0;
	int nextRow = order/2;
	int nextCol = matrix.length-1;
	int one = 1;
	
	for (int i = 0; i < order; i++){
	for (int j = 0; j < order; j++){
	matrix[nextRow][nextCol]
	= one;
	
	lastCol = nextCol;
	lastRow= nextRow;
	
	nextRow = lastRow-1;

	nextCol = lastCol+1;
	
	if(nextRow < 0){
	nextRow = order - 1;
	}
	if(nextCol >= order){
	nextRow = lastRow - 1;
	
	nextCol = 0;
	}
	
	if(nextRow == -1 || matrix[nextRow][nextCol] != 0){
	nextRow = lastRow;
	nextCol = lastCol - 1;
	}
	one++;
	}
	}
	}

RF6: Recorrer cuadrado mágico noroeste, posición arriba	public void northwestUp() {
	int lastCol = 0;
	int lastRow = 0;
	int nextRow = 0;
	int nextCol = order/2;
	int one = 1;
	
	for (int i = 0; i < order; i++){
	for (int j = 0; j < order; j++){
	matrix[nextRow][nextCol] = one;
	
	lastCol = nextCol;
	lastRow= nextRow;
	
	nextRow = lastRow - 1;
	nextCol = lastCol - 1;
	
	if(nextRow < 0){
	nextRow = order - 1;
	}
	if(nextCol < 0){
	nextRow = lastRow - 1;
	
	nextCol = order - 1;
	}
	
	if(nextRow == -1 || matrix[nextRow][nextCol] != 0){
	nextRow = lastRow + 1;

	nextCol = lastCol;
	}
	one++;
	}
	}
	}

RF7: Recorrer cuadrado mágico noroeste, posición izquierda	public void northwestLeft() {
	int lastCol = 0;
	int lastRow = 0;
	int nextRow = order/2;
	int nextCol = 0;
	int one = 1;
	
	for (int i = 0; i < order; i++){
	for (int j = 0; j < order; j++){
	matrix[nextRow][nextCol] = one;
	
	lastCol = nextCol;
	lastRow= nextRow;
	
	nextRow = lastRow - 1;
	nextCol = lastCol - 1;
	
	if(nextRow < 0){
	nextRow = order - 1;
	}
	if(nextCol < 0){
	nextRow = lastRow - 1;
	
	nextCol = order - 1;
	}
	
	if(nextRow == -1 || matrix[nextRow][nextCol] != 0){
	nextRow = lastRow;
	nextCol = lastCol + 1;
	
	}
	one++;
	}
	}
	}

RF8: Recorrer cuadrado mágico sureste, posición abajo	public void southeastBottom() {
	int lastCol = 0;
	int lastRow = 0;
	int nextRow = matrix.length-1;
	int nextCol = order/2;
	int one = 1;
	
	for (int i = 0; i < order; i++){

	for (int j = 0; j < order; j++){
	matrix[nextRow][nextCol] = one;
	
	lastCol = nextCol;
	lastRow= nextRow;
	
	nextRow = lastRow+1;
	nextCol = lastCol+1;
	
	if(nextRow >= order){
	nextRow = 0;
	}
	if(nextCol >= order){
	nextRow = lastRow + 1;
	
	nextCol = 0;
	}
	
	if(nextRow == order || matrix[nextRow][nextCol] != 0){
	nextRow = lastRow - 1;
	
	nextCol = lastCol;
	}
	one++;
	}
	}
	}

RF9: Recorrer cuadrado mágico sureste, posición derecha	public void southeastRight() {
	int lastCol = 0;
	int lastRow = 0;
	int nextRow = order/2;
	int nextCol = matrix.length-1;
	int one = 1;
	
	for (int i = 0; i < order; i++){
	for (int j = 0; j < order; j++){
	matrix[nextRow][nextCol] = one;
	
	lastCol = nextCol;
	lastRow= nextRow;
	
	nextRow = lastRow+1;
	nextCol = lastCol+1;
	
	if(nextRow >= order){
	nextRow = 0;
	}
	if(nextCol >= order){

	nextRow = lastRow + 1;
	nextCol = 0;
	}
	
	if(nextRow == order || matrix[nextRow][nextCol] != 0){
	nextRow = lastRow;
	nextCol = lastCol - 1;
	}
	one++;
	}
	}
	}

RF10: Recorrer cuadrado mágico suroeste, posición abajo	public void southwestBottom() {
	int lastCol = 0;
	int lastRow = 0;
	int nextRow = matrix.length-1;
	int nextCol = order/2;
	int one = 1;
	
	for (int i = 0; i < order; i++){
	for (int j = 0; j < order; j++){
	matrix[nextRow][nextCol] = one;
	
	lastCol = nextCol;
	lastRow= nextRow;
	
	nextRow = lastRow+1;
	nextCol = lastCol-1;
	
	if(nextRow >= order){
	nextRow = 0;
	}
	if(nextCol < 0){
	nextRow = lastRow + 1;
	nextCol = order - 1;
	}
	
	if(nextRow == order || matrix[nextRow][nextCol] != 0){
	nextRow = lastRow - 1;
	nextCol = lastCol;
	}
	one++;
	}
	}
	}

RF11: Recorrer cuadrado mágico suroeste, posición izquierda	`public void southwestLeft() {`
	`int lastCol = 0;`
	`int lastRow = 0;`
	`int nextRow = order/2;`
	`int nextCol = 0;`
	`int one = 1;`
	
	`for (int i = 0; i < order; i++){`
	`for (int j = 0; j < order; j++){`
	`matrix[nextRow][nextCol] = one;`
	
	`lastCol = nextCol;`
	`lastRow= nextRow;`
	
	`nextRow = lastRow+1;`
	`nextCol = lastCol-1;`
	
	`if(nextRow >= order){`
	`nextRow = 0;`
	`}`
	`if(nextCol < 0){`
	`nextRow = lastRow + 1;`
	`nextCol = order - 1;`
	`}`
	
	`if(nextRow == order || matrix[nextRow][nextCol] != 0){`
	`nextRow = lastRow;`
	`nextCol = lastCol + 1;`
	`}`
	`one++;`
	`}`
	`}`
	`}`

RF12: Sumar filas	`public boolean plusRows(int constant) {`
	`boolean tr = true;`
	`int pRows = 0;`
	`int k = 0;`
	`while (k < order && tr) {`
	`for (int i = 0; i < order; i++){`
	`pRows += matrix[k][i];`
	`}`
	`if (pRows != constant) {`
	`tr = false;`
	`}`
	`k++;`
	`pRows = 0;`
	`}`

	`return tr;` `}`

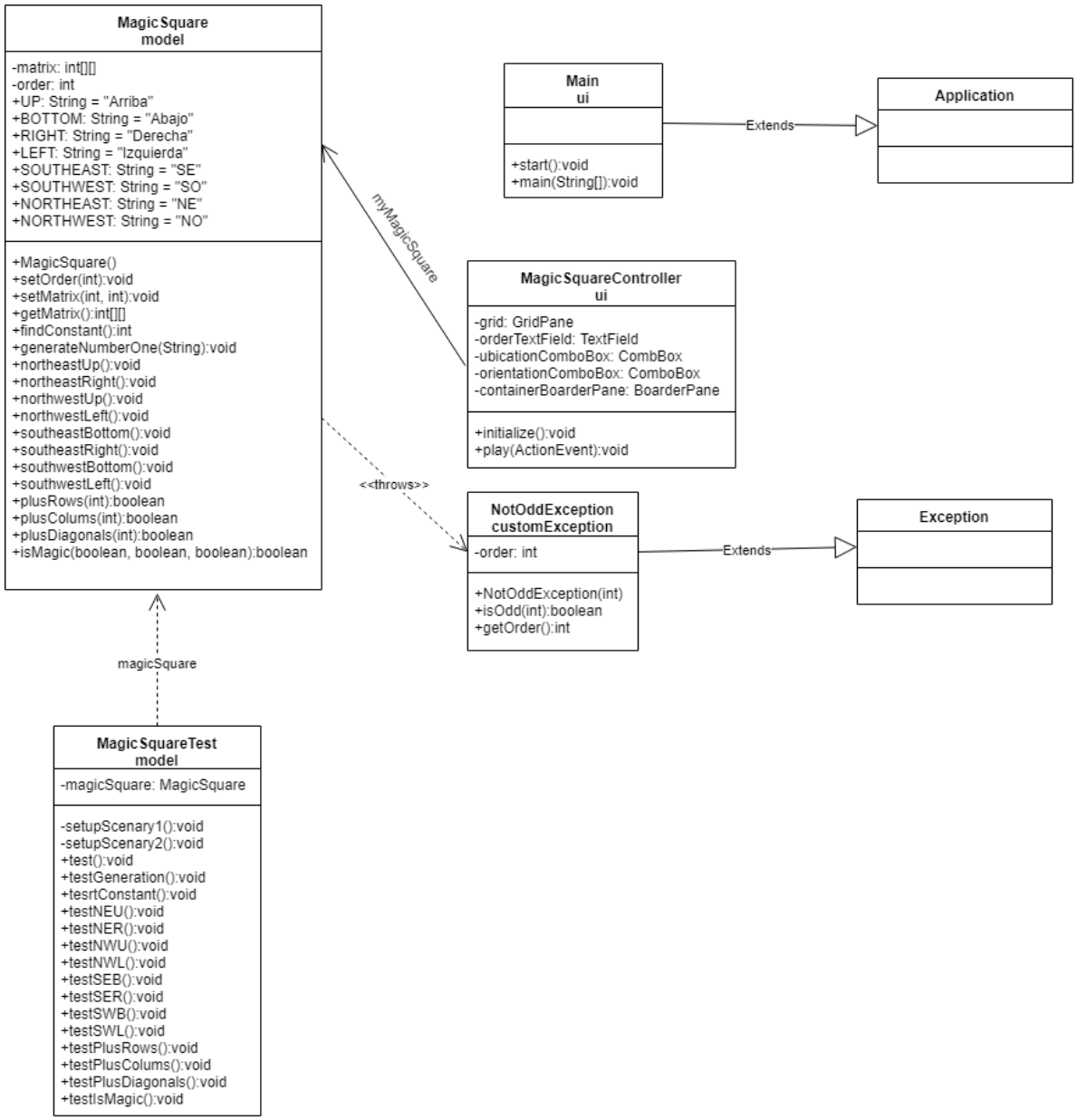
RF13: Sumar columnas	`public boolean plusColums(int constant){`
	`boolean tc = true;`
	`int pCols = 0;`
	`int k = 0;`
	`while (k < order && tc) {`
	`for (int i = 0; i < order; i++){`
	`pCols += matrix[k][i];`
	`}`
	`if (pCols != constant) {`
	`tc = false;`
	`}`
	`k++;`
	`pCols = 0;`
	`}`
	
	`return tc;`
	`}`

RF14: Sumar diagonales	`public boolean plusDiagonlas(int constant){`
	`boolean td = true;`
	`int pDiag = 0;`
	`int k = 0;`
	`int l = 0;`
	`while (k < order && l < order) {`
	`pDiag += matrix[k][l];`
	`k++;`
	`l++;`
	`}`
	
	`if (pDiag != constant) {`
	`td = false;`
	`} else {`
	`int pDiag1 = 0;`
	`int i = 0;`
	`int j = order - 1;`
	`while (i < order && j >= ){`
	`pDiag1 += matrix[i][j];`
	`i++;`
	`j--;`
	`}`
	`if (pDiag1 != constant) {`
	`td = false;`
	`}`
	`}`
	
	`return td;`
	`}`

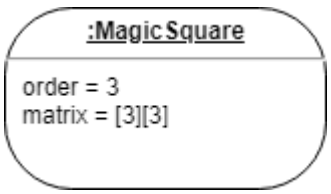
RF15: Verificar si es cuadrado mágico	`public boolean isMagic(boolean pRows, boolean pCols, boolean pDiag) {`

	`boolean magic = true;`
	
	`if (pRows == true && pCols == true && pDiag == true) {`
	`magic = true;`
	`}else {`
	`magic = false;`
	`}`
	
	`return magic;`
	
	`}`

Diagrama de clases



Pruebas

Nombre	Clase	Escenario
setupScenary1	MagicSquare Test	vacío
setupScenary2	MagicSquareTest	

Objetivo de la prueba: Verificar la correcta creación de una matriz dado un orden				
Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
MagicSquare	MagicSquare	setupScenary1	order = 5 matrix = [5][5]	*True* Se creo correctamente una nueva matriz de 5x5

Objetivo de la prueba: Corroborar el correcto cálculo de la constante				
Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
MagicSquare	findConstant	setupScenary2		*True* La constante es igual a 15

Objetivo de la prueba: Comprobar la correcta asignación del número según la opción esocogida				
Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
MagicSquare	generateNumberOne	setupScenary2		*True* Según la opción elegida se debe crear el número en el lugar determinado

Objetivo de la prueba: Corroborar el correcto cálculo de la constante				
Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
MagicSquare	northeastUp	setupScenary2		*True* El recorrido es exitoso

Objetivo de la prueba: Corroborar el correcto cálculo de la constante				
Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
MagicSquare	northeastRight	setupScenary2		*True* El recorrido es exitoso

Objetivo de la prueba: Corroborar el correcto cálculo de la constante				
Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
MagicSquare	northwestUp	setupScenary2		*True* El recorrido es exitoso

Objetivo de la prueba: Corroborar el correcto cálculo de la constante				
Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
MagicSquare	northwestLeft	setupScenary2		*True* El recorrido es exitoso

Objetivo de la prueba: Corroborar el correcto cálculo de la constante				
Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
MagicSquare	southeastBottom	setupScenary2		*True* El recorrido es exitoso

Objetivo de la prueba: Corroborar el correcto cálculo de la constante				
Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
MagicSquare	southeastRight	setupScenary2		*True* El recorrido es exitoso

Objetivo de la prueba: Corroborar el correcto cálculo de la constante				
Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
MagicSquare	southwestBottom	setupScenary2		*True* El recorrido es exitoso

Objetivo de la prueba: Corroborar el correcto cálculo de la constante				
Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
MagicSquare	southwestLeft	setupScenary2		*True* El recorrido es exitoso

Objetivo de la prueba: Corroborar el correcto cálculo de la constante				
Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
MagicSquare	plusRows	setupScenary2		*True* La suma de las filas del cuadrado mágico es igual a la constante

Objetivo de la prueba: Corroborar el correcto cálculo de la constante				
Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
MagicSquare	plusColums	setupScenary2		*True*

				La suma de las columnas del cuadrado mágico es igual a la constante

Objetivo de la prueba: Corroborar el correcto cálculo de la constante				
Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
MagicSquare	plusDiagonals	setupScenary2		*True* La suma de las diagonales del cuadrado mágico es igual a la constante

Objetivo de la prueba: Corroborar el correcto cálculo de la constante				
Clase	**Método**	**Escenario**	**Valores de entrada**	**Resultado**
MagicSquare	plusRows	setupScenary2		*True* La suma de las filas, las columnas y las diagonales del cuadrado mágico es igual a la constante