

LABORATORIO #3

JULIAN CAMILO LOPEZ BARRERO

JUAN SEBASTIAN PUENTES JULIO

ESCUELA COLOMBIANA DE INGENIERÍA JULIO GARAVITO

POOB

BOGOTÁ, COLOMBIA

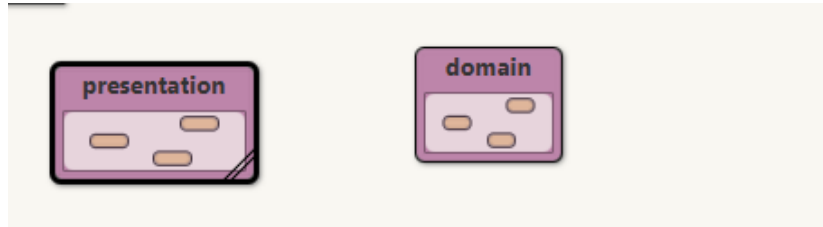
14 DE FEBRERO DE 2025

Conociendo [En lab03.doc y schelling.asta]

1. En el directorio descarguen los archivos contenidos en schelling.zip. Revisen el código:

a) ¿Cuántos paquetes tiene?

Tiene 2 paquetes.



d) ¿Cuál es el propósito del paquete presentación?

Preparar el entorno en el que nos vamos a desenvolver. En este caso se refiere a una ciudad. Ejecutar el programa y la visualización de esto.

e) ¿Cuál es el propósito del paquete dominio?

Definir diferentes comportamientos como las interfaces que hay , la herencia que hay entre persona y agente etc...

2. Revisen el paquete de dominio

a)¿Cuáles son los diferentes tipos de componentes de este paquete?

Se tiene una clase abstracta y una interfaz, además existen otras que las implementan para definir los diferentes comportamientos (las clases concretas).

b) ¿Qué implica cada uno de estos tipos de componentes?

Hay diferentes tipos de relaciones. Person implementa Ítem y extiende Agent además que este los usa y adicionalmente usa la clase City.

3. Revisen el paquete de presentación

a) ¿Cuántos componentes tiene?

Tiene un componente.

b)¿Qué métodos públicos propios (no heredados) ofrece?

Ofrece gettheCity y paintComponent()

4. Para ejecutar un programa en java

¿Qué método se debe ejecutar?

Se debe ejecutar el main.

¿En qué clase se encuentra?

Se encuentra en la clase CityGUI.

5. Ejecuten el programa.

¿Qué funcionalidades ofrece? ¿Qué hace actualmente?

Al ejecutar el programa se abre una interfaz con diferentes cuadros en ella. Además, existe una opción para hacer click llamada Tic-Tac. Esta última no deja hacer nada.

¿Por qué?

Porque no está implementada aun,

Arquitectura general. [En lab03.doc y schelling.asta]

1. Consulte el significado de las palabras package e import de java.

Package: los paquetes en Java son contenedores que posibilitan agrupar **clases** e **interfaces** relacionadas entre sí.

Import: La palabra clave import se utiliza para importar clases o paquetes completos en tu código, lo que te permite utilizar esas clases sin tener que escribir sus nombres completos.

¿Qué es un paquete?

Los paquetes son el mecanismo que usa Java **para facilitar la modularidad del código**

¿Para qué sirve?

Sirve para para facilitar la modularidad del código igualmente promueve la encapsulación y la seguridad del código

¿Para qué se importa?

-Al importar paquetes, se pueden reutilizar clases y métodos ya existentes sin tener que escribirlos desde cero.

-Los paquetes ayudan a organizar el código en módulos lógicos. Esto hace que el código sea más fácil de mantener y entender.





Explique su uso en este programa.

Para este programa tenemos dos paquetes que son el de presentation y el de domain que respectivamente tienen como uso la interfaz en la que se va a envolver el programa y el dominio que va a ser donde se van a realizar las distintas operaciones sobre los agentes, las personas, los items y las ciudades.

2. Revise el contenido del directorio de trabajo y sus subdirectorios. Describa su contenido. ¿Qué coincidencia hay entre paquetes y directorios?

La coincidencia que se evidencia entre paquetes y directorios es que este último guarda de la misma manera que los paquetes.

En los directorios se encuentran los respectivos paquetes con sus archivos.java así como en el archivo doc que presenta contenido variado como extensiones html, algunos java script y carpetas como application, domain, presentacion, presentation y resources, cada una de las anteriores tienen archivos que se caracterizan por su extensión html.

 doc	-	233 KB
 domain	-	13 KB
 presentation	-	14 KB
 package.bluej	12 mar 2025	799 bytes

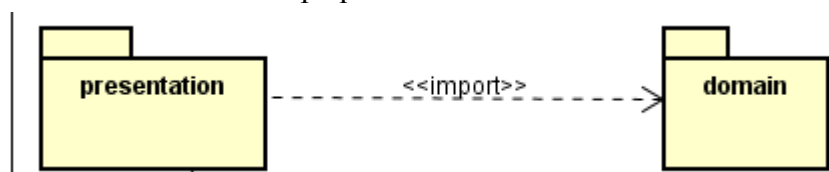
	Nombre	Última modificación	Tamaño del archivo
≡	Agent.class	12 mar 2025	988 bytes
📄	Agent.ctxt	12 mar 2025	887 bytes
📄	Agent.java	12 mar 2025	1 KB
≡	City.class	12 mar 2025	2 KB
📄	City.ctxt	12 mar 2025	637 bytes
📄	City.java	12 mar 2025	1 KB
≡	Item.class	12 mar 2025	697 bytes
📄	Item.ctxt	12 mar 2025	419 bytes
📄	Item.java	12 mar 2025	519 bytes
📄	package.bluej	12 mar 2025	1 KB
≡	Person.class	12 mar 2025	1 KB
📄	Person.ctxt	12 mar 2025	848 bytes
📄	Person.java	12 mar 2025	1 KB

	Nombre	Última modificación	Tamaño del archivo
≡	CityGUI\$1.class	12 mar 2025	703 bytes
≡	CityGUI.class	12 mar 2025	2 KB
📄	CityGUI.ctxt	12 mar 2025	486 bytes
📄	CityGUI.java	12 mar 2025	4 KB
📄	package.bluej	12 mar 2025	605 bytes
≡	PhotoAManufacturing.class	29 sept 2022	2 KB
≡	PhotoAutomata.class	10 mar 2022	2 KB
≡	PhotoCity.class	12 mar 2025	2 KB

3. Adicione al diseño la arquitectura general con un diagrama de paquetes en el que se presente los paquetes y las relaciones entre ellos. Consulte la referencia en moodle.

En astah, crear un diagrama de clases (cambiar el nombre por Package Diagram)

Relación de uso de los paquetes.



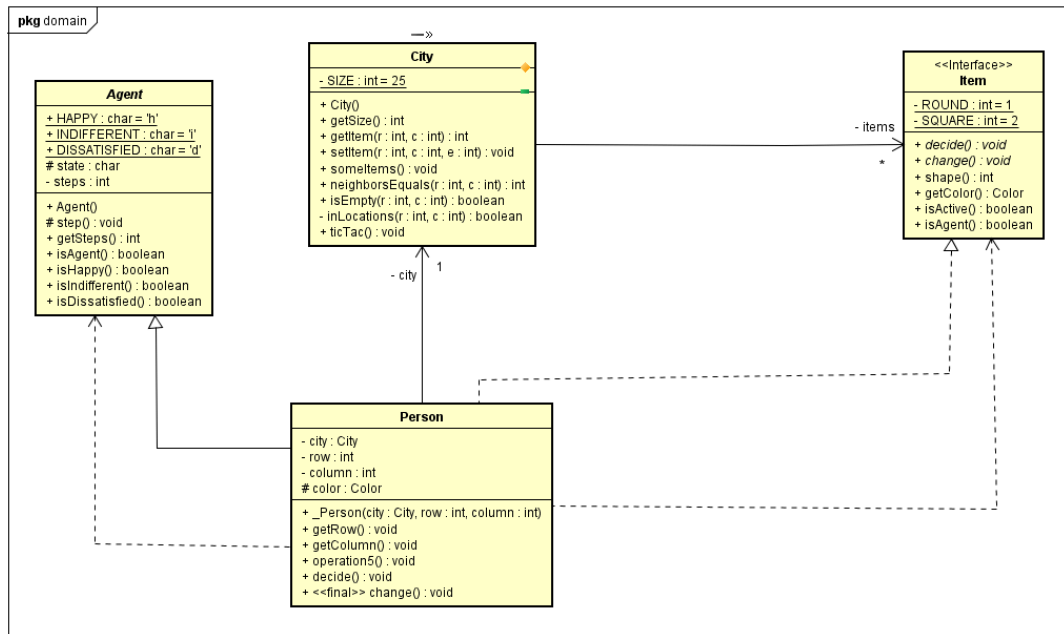
Arquitectura detallada. [En lab03.doc y schellingasta]

1. Para preparar el proyecto para BDD. Completen el diseño detallado del paquete de dominio. Adicionen el diagrama de clases en el paquete correspondiente.

a) ¿Qué componentes hacían falta?

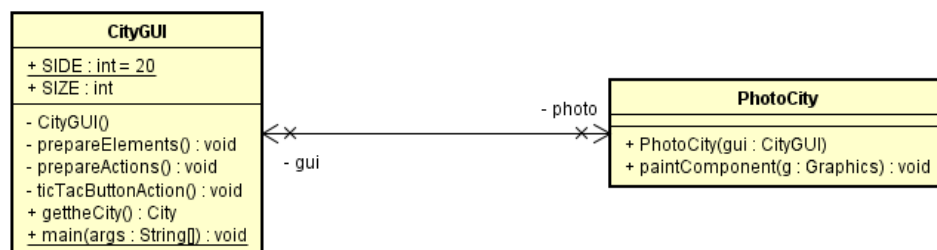
Hacían Falta Ítem y agent

Lo podemos observar así:



2. Completen el diseño detallado del paquete de presentación. Adicionen el diagrama de clases al paquete correspondiente.

Diagrama:



a) ¿Por qué hay dos clases y un archivo .java?

Hay dos clases porque la presentación usa el CityGUI para crear la interfaz y photoCity que se extiende dentro de JPanel en la clase de CityGUI, el archivo .java es CityGUI.

3. Adicione la clase de pruebas unitarias necesaria para BDD en un paquete independiente de test. (No lo adicione al diagrama de clases)

¿Qué paquete debe usar?

Debe usar el paquete dominio.

¿Por qué?

Porque allí se encuentra la lógica que luego se usa para crear el juego.

¿Asociado a qué clase?

A la clase city.

¿Por qué?

Porque la instancia de esta clase es con la que vamos a interactuar durante las pruebas.

Ciclo 1. Iniciando con las personas normales [En lab03.doc y *.java]

(NO OLVIDE BDD – MDD)

1. Estudie la clase City ¿Qué tipo de colección usa para albergar cosas?

Para albergar las cosas usa la colección de vectores 2D.

¿Puede recibir personas?

Puede recibir personas.

¿Por qué?

Porque implementa la interfaz ítem que se llama locations.

2. Estudie el código asociado a la clase Person

¿En qué estado se crea?

Se crea en estado DISSATISFIED debido a que la persona hereda todo lo que tiene la clase agente.

¿Qué forma usa para pintarse?

La forma que usa para pintarse es ROUND.

¿Cuándo aumenta su tiempo?

Cuando este se mueve, entonces suma con el método step().

¿Qué clases definen la clase Person ? Justifique sus respuestas.

Las clases que definen a Person son Item y Agent porque respectivamente Item tiene la forma que esta va a tener y el Agent nos va a definir el estado de estos.

3. Person por ser un Agent /Justifique sus respuestas.

¿Qué atributos tiene?

Tiene una ciudad como atributo, el número de fila y columna donde se encuentra, el color y así mismo el estado y los steps que estos dan. Además, de un char que define happy, indifferent y dissatisfied.

¿Qué puede hacer (métodos)?

Puede hacer un step, obtener los steps que ha hecho, también saber si es un agente y su estado.

¿Qué decide hacer distinto?

Por ser un agente no puede hacer nada distinto.

¿Qué no puede hacer distinto a todos los agentes?

Por ser un agente no puede hacer nada diferente a las cosas que tienen final.

¿Qué debe aprender a hacer?

Person debería aprender a decidir de su estado según su estado emocional y debería definir su comportamiento para moverse dentro de city

Justifique sus respuestas.

4. Por comportarse como un Ítem Justifique sus respuestas.

¿qué sabe hacer?

Persona por comportarse como un ítem sabe retornar el color, isActive, isAgent, shape, change y decide.

¿qué decide hacer distinto?

Decide hacer distinto, decide y change ya que son métodos que debemos implementar para cada persona y del modo modo debe aprender a hacer estos.

¿qué no puede hacer distinto?

No puede cambiar su forma porque según la clase de Person no se sobrescribe o se cambia esto y su valor se queda en round y si es activo o es agente están definidos respectivamente *como true y false*

¿qué debe aprender a hacer?

Debe aprender a cambiar su forma el mismo sobrescribiendo a el valor SQUARE y sus valores booleanos si es activo o no

5. De acuerdo a lo anterior una Person, ¿Cómo actúa (decide+cambia)?

Cada vez que se llama a decidir lo que hace la persona es establecer su estado emocional dependiendo del número de pasos que de y en la casilla que quede y como actúa con change es que este va a incrementar el número de pasos llamando al método step().

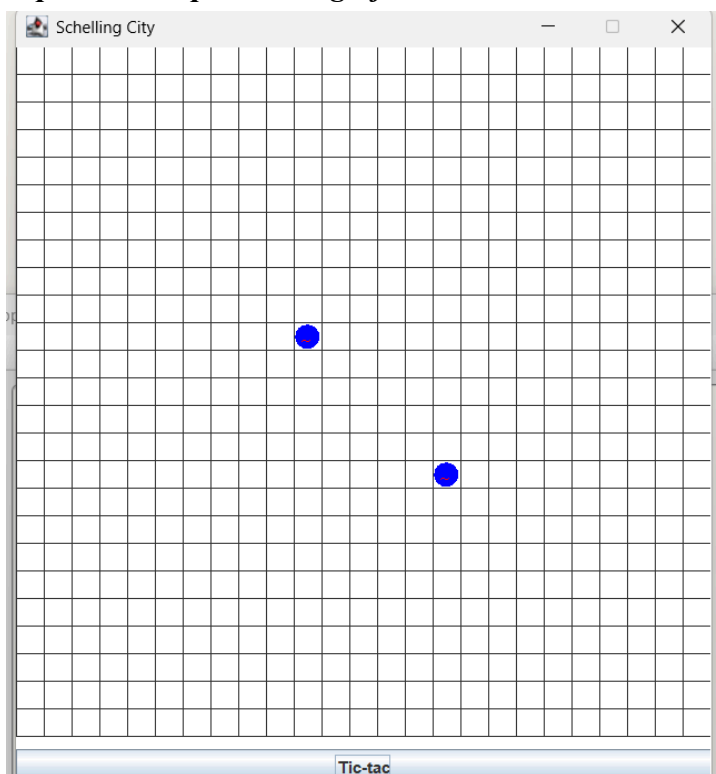
6. Ahora vamos a crear dos personas en diferentes posiciones (10,10) (15,15) llámelas adan y eva usando el método someItems() .

Ejecuten el programa,

¿Qué pasa con las personas? ¿Por qué?

Aparecen personas con estado de ánimo insatisfechos debido a que es el estado que tienen por defecto.

Capturen una pantalla significativa.



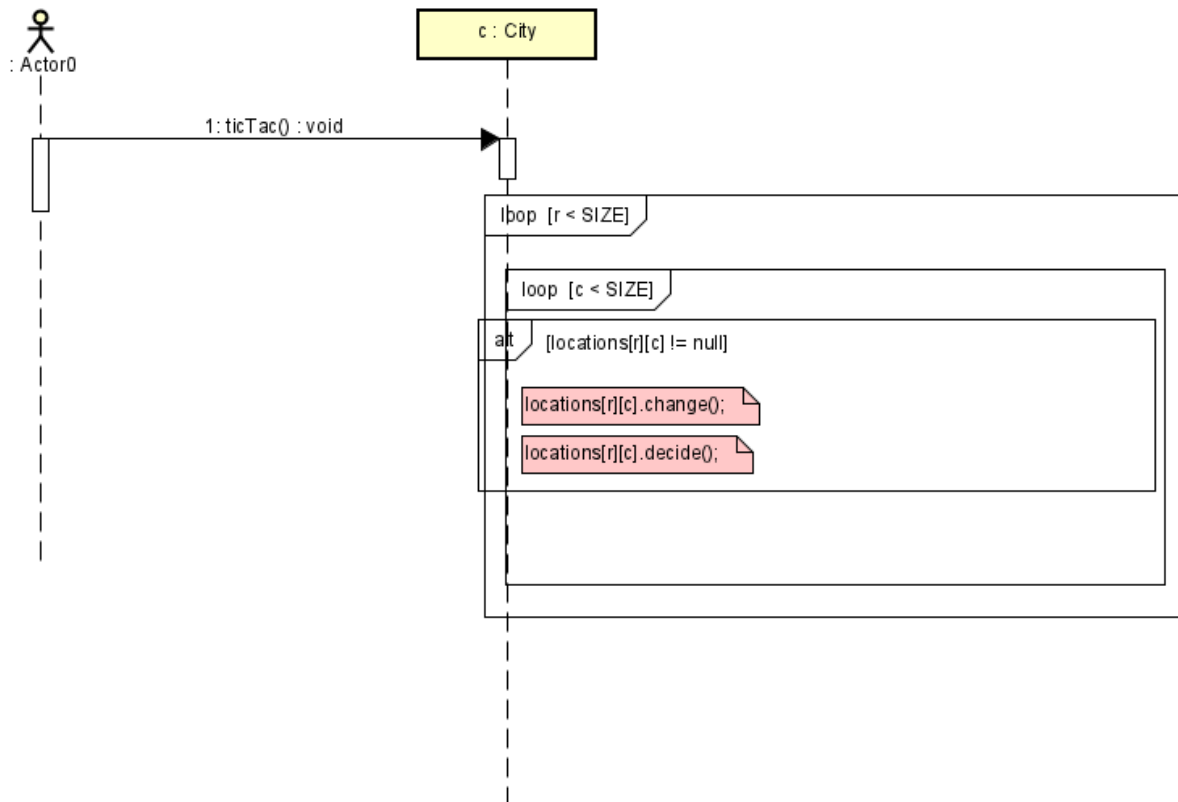
7. Diseñen, construyan y prueben el método llamado ticTac() de la clase City.

```

/**
 * Logic for tic tac button where items change and decide.
 */
public void ticTac(){
    for (int r = 0; r < SIZE; r++) {
        for (int c = 0; c < SIZE; c++) {
            if (locations[r][c] != null) {
                locations[r][c].change();
                locations[r][c].decide();
            }
        }
    }
}

```

El diseño se encuentra en astah.

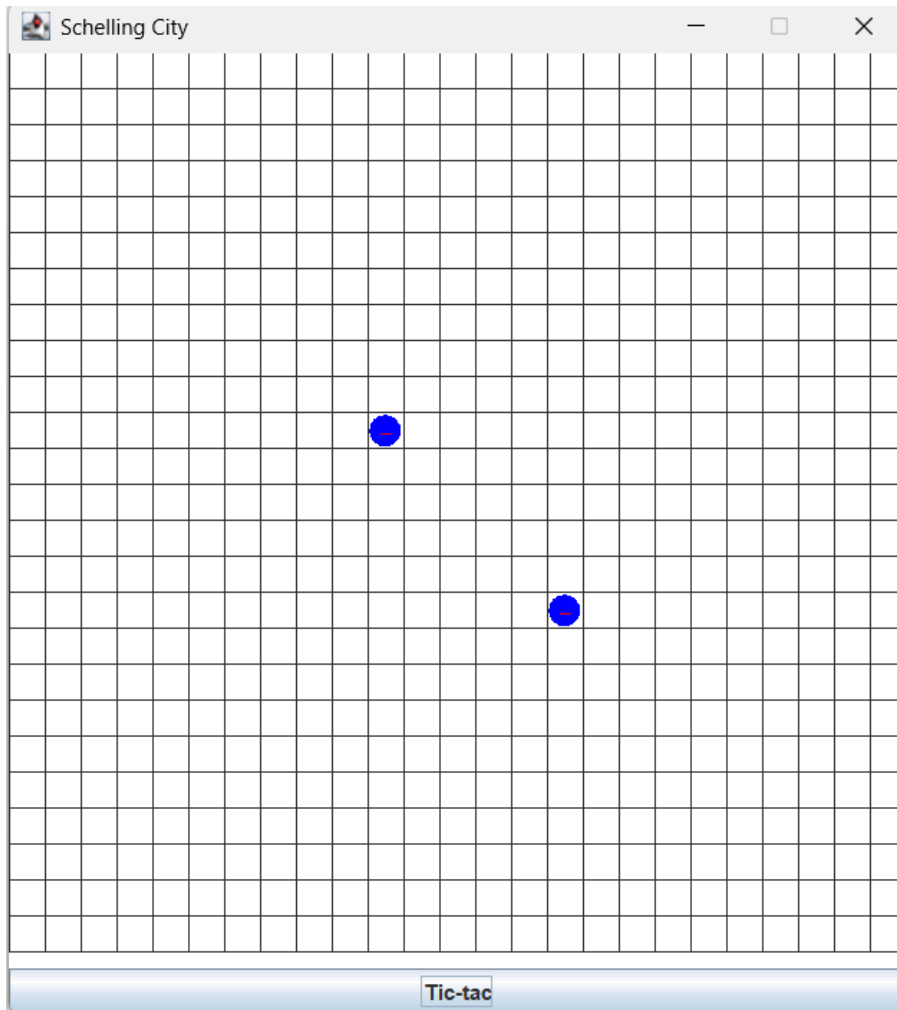


8. ¿Cómo quedarían adan y eva después de uno, dos, cuatro y seis Tic-tac? Ejecuten el programa. Después de 1 indiferente, 2 Insatisfecho, 4 Indiferente y 6 HAPPY. Capturan pantallas significativas en momentos correspondientes. ¿Es correcto?

Un tic-tac

Indiferente

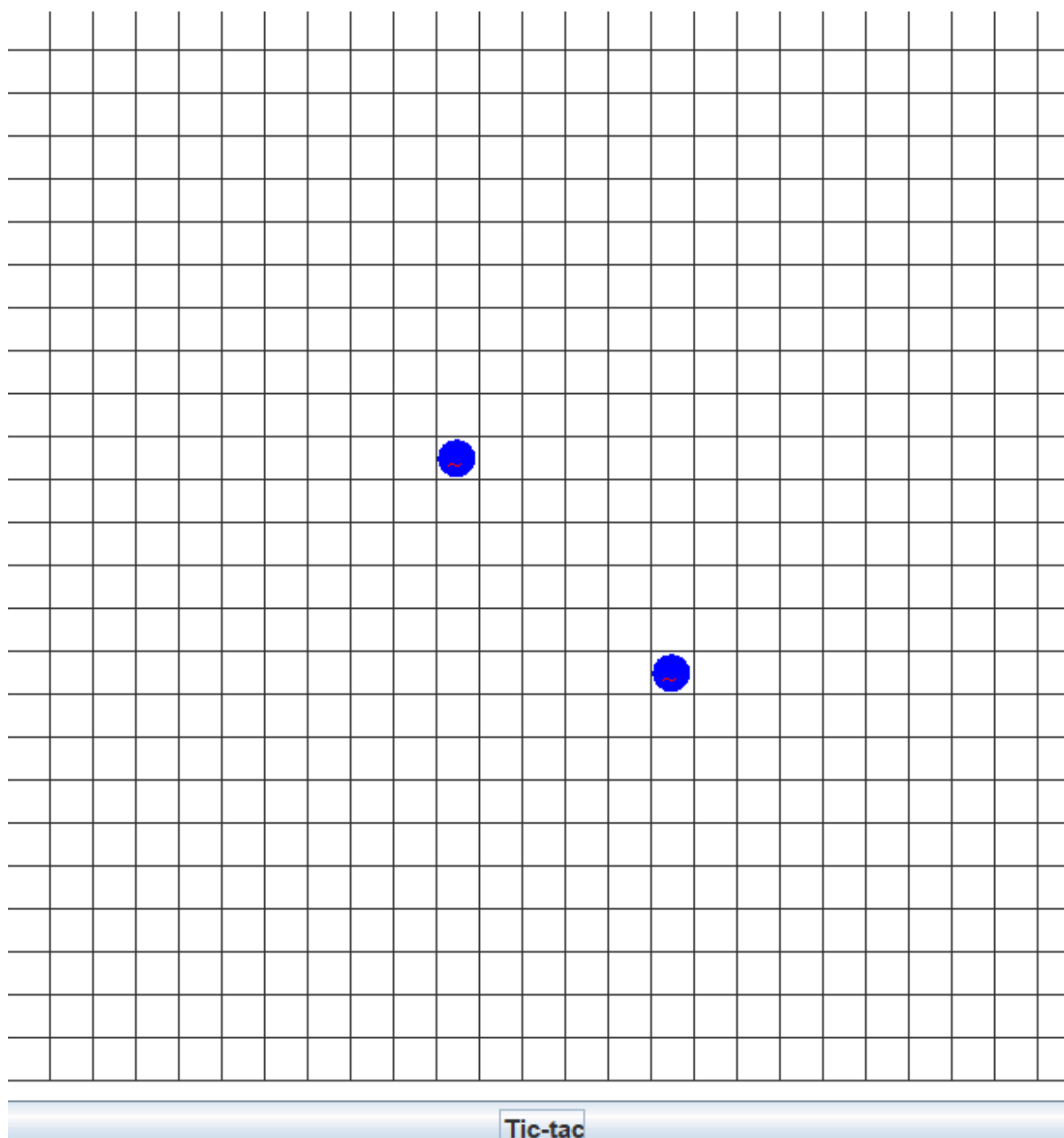
$1 \bmod 3 = 1 \rightarrow \text{Agent.INDIFFERENT}$



Dos tic-tac

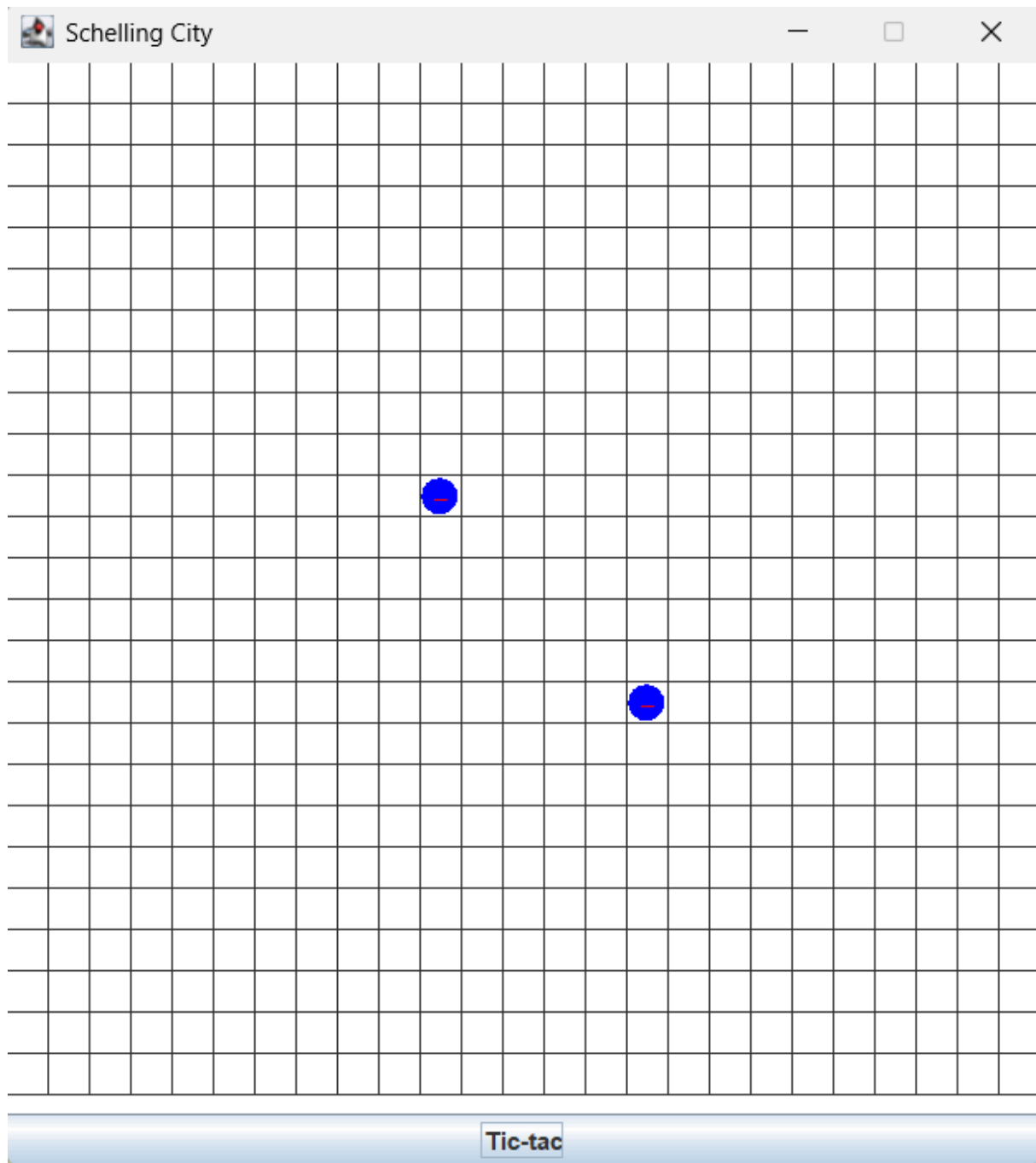
Insatisfecho

$2 \bmod 3 = 2 \rightarrow \text{Agent.DISSATISFIED}$



Cuatro tic-tac
Indiferente

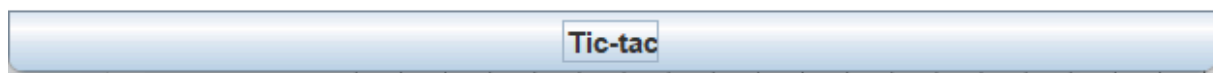
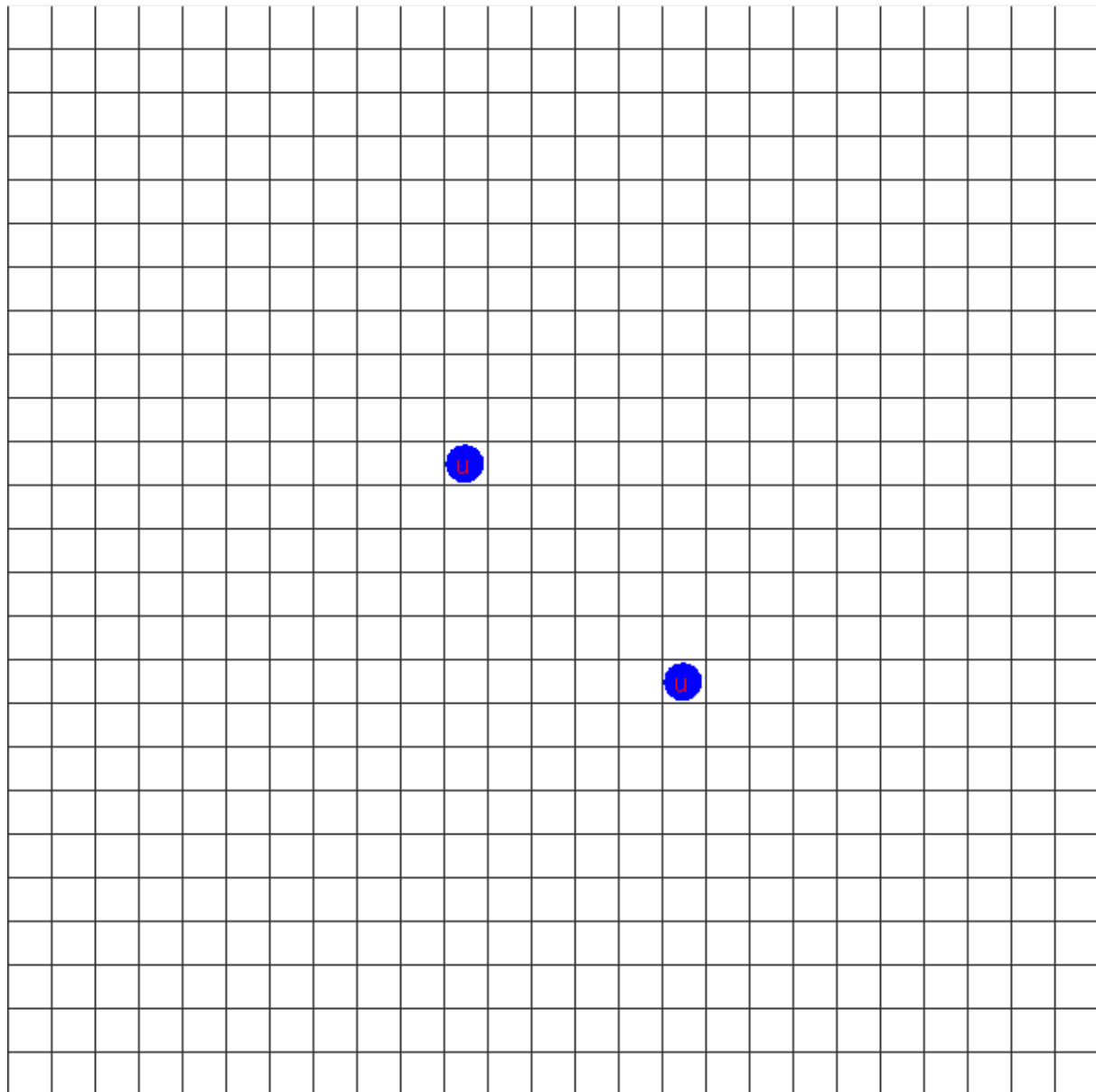
$4 \bmod 3 = 1 \rightarrow \text{Agent.INDIFFERENT}$



Seis tic-tac

Feliz

$6 \bmod 3 = 0 \rightarrow \text{Agent.HAPPY}$



Es correcto porque cada que nosotros hacemos un tic lo que va a pasar es que se aumentan los pasos y ahí se va a pasar al método de decidir que lo que hace es que a partir de los steps haga el módulo y defina qué estado de ánimo tiene que es la condición que tenemos en el método decidir dentro de Person.

Ciclo 2. Incluyendo a los caminantes [En lab03.doc y schellingasta]

(NO OLVIDE BDD – MDD)

El objetivo de este punto es permitir recibir personas caminantes . Ellas (i) son rectángulos de color verde; (ii) inician indiferentes; (ii) se mueven hacia el norte,

, (iii) si quedan vecinos a un ítem, se ponen felices; (iv) si no logran moverse al sitio que querían, quedan insatisfechos.

1. Para implementar esta nueva persona Walker ¿cuáles métodos se sobre-escriben (overriding)?

Se deben sobrescribir los métodos decide() y shape() para que estos cambien su figura que en su caso es un cuadrado.

2. Diseñen, construyan y prueben esta nueva clase. (Mínimo dos pruebas de unidad).

Se encuentra el respectivo diseño en Astah y la implementación en BlueJ con las pruebas de unidad.

```
public void decide(){
    checkItemsAround();
    move();
}

/**
 * Checks if there exists items around the walker. If exists then his state turns happy.
 */
public void checkItemsAround(){
    for(int dr=-1; dr<2;dr++){
        for (int dc=-1; dc<2;dc++){
            if( dr == 0 && dc == 0){
                continue;
            }
            int checkRow = row + dr;
            int checkColumn = column + dc;
            if(city.inLocations(checkRow,checkColumn) && city.getItem(checkRow,checkColumn) != null){
                state = HAPPY;
            }
        }
    }
}

/**
 * Moves the walker only to the north.
 */
public void move(){
    if (row > 0) {
        int newRow = row - 1;
        if (city.isEmpty(newRow, column)) {
            city.setItem(row, column, null);
            row = newRow;
            city.setItem(row, column, this);
        }else{
            state = DISSATISFIED;
        }
    }
}
```

```

public class Walker extends Person
{
    /**
     * Constructor for objects of class Walker
     */
    public Walker(City city,int row,int column){
        super(city,row,column);
        color = color.green;
        state = INDIFFERENT;
    }

    /**
     * Returns the Walker Shape.
     * @return SQUARE int.
     */
    @Override
    public int shape(){
        return SQUARE;
    }

    /**
     * Checks items around and then moves.
     */
    @Override
    public void decide(){
        checkItemsAround();
        move();
    }
}

```

```

public void shouldWalkerBeHappy(){
    Person eva = new Person(city,15,15);
    Walker messner = new Walker(city,18,16);

    city.ticTac();
    city.ticTac();
    city.ticTac();

    char expectedState = 'h';
    char actualState = messner.getState();
    assertEquals(expectedState,actualState);
}

```

```

@Test
public void shouldWalkerBeDissatisfied(){
    Person adan = new Person(city,10,10);
    Walker kukuczka = new Walker(city,20,10);

    city.ticTac();
    city.ticTac();
    city.ticTac();
    city.ticTac();
    city.ticTac();
    city.ticTac();
    city.ticTac();
    city.ticTac();
    city.ticTac();
    city.ticTac();

    char expectedState = 'd';
    char actualState = kukuczka.getState();
    assertEquals(expectedState,actualState);
}

```

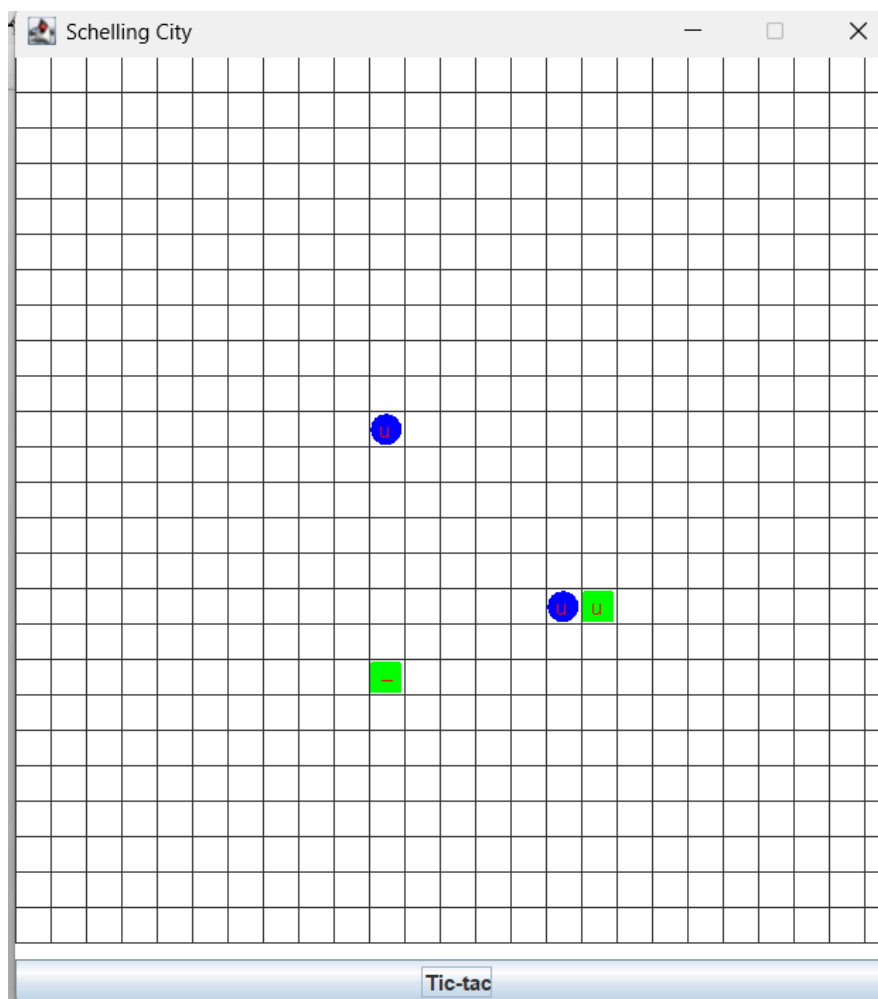
```
@Test
public void shouldWalkerBeIndifferent(){
    Person adan = new Walker(city,10,10);
    char expectedState = 'i';
    char actualState = adan.getState();
    assertEquals(expectedState,actualState);
}
```

3. Adicione una pareja de caminantes, llámese messner y kukuczka, (a) ¿Cómo quedarían después de tres Tic-tac?

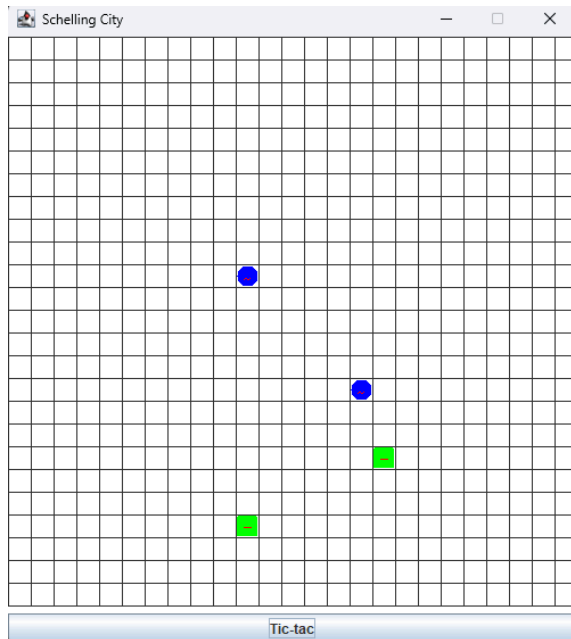
Después de tres Tic- Tac deberían moverse tres unidades al norte y uno de ellos cambiar a estado HAPPY debido a que tiene items alrededor de él.

Ejecuten el programa y hagan tres clics en el botón. Capturen una pantalla significativa. (b) ¿Es correcto?

Después de 3 tic-tac:



Es correcto debido a que su estado inicial era el siguiente:



Ciclo 3. Adicionando semáforos [En lab03.doc, schelling.asta y *.java]

El objetivo de este punto es incluir semáforos (sólo vamos a permitir el tipo básico de semaforos) los semaforos son redondos, siempre activos y van cambiando de color: rojo, amarillo, verde, amarillo, rojo, etc

(NO OLVIDE BDD – MDD)

1. Para poder adicionar semáforos, ¿debe cambiar en el código de City en algo? ¿por qué?

En nuestro caso, no fue necesario cambiar el código de city ya que se definió una clase semaforo que implementa el ítem.

2. Diseñen , construyan y prueben esta nueva clase. (Mínimo dos pruebas de unidad)

Se encuentra el respectivo diseño en astah y la implementación en BlueJ con las pruebas de unidad.


```

@Test
public void shouldBeRedLightAfter4TicTacs(){
    Light alert = new Light(city,0,0);

    city.ticTac();
    city.ticTac();
    city.ticTac();
    city.ticTac();

    char expectedState = 'r';
    char actualState = alert.getLightState();

    assertEquals(expectedState,actualState);
}

```

```

@Test
public void shouldBothLightsHaveTheSameStateAfter3TicTacs(){
    Light alert = new Light(city,0,0);
    Light alarm = new Light(city,0,24);

    city.ticTac();
    city.ticTac();
    city.ticTac();

    char actualState = alert.getLightState();
    char actualState1 = alarm.getLightState();

    assertEquals(actualState,actualState1);
}

```

```

/**
 * Change Color of the Light.
 */
@Override
public void decide(){
    changeColor();
}

```

```

/**
 * Returns color of the item.
 * @return Color color.
 */
public Color getColor(){
    return color;
}

```

```

/**
 * Change the color and states of the item.
 */
public void changeColor() {
    if (state == RED && color == Color.red) {
        state = YELLOW;
        color = Color.yellow;
        lastColor = RED;
    } else if (state == YELLOW && color == Color.yellow && lastColor == RED) {
        state = GREEN;
        color = Color.green;
        lastColor = YELLOW;
    } else if (state == GREEN && color == Color.green) {
        state = YELLOW;
        color = Color.yellow;
        lastColor = GREEN;
    } else if (state == YELLOW && color == Color.yellow && lastColor == GREEN){
        state = RED;
        color = Color.red;
        lastColor = YELLOW;
    }
}

```

3. Adicionen dos semáforos en las esquinas superiores de la ciudad, llámenlos alarm y

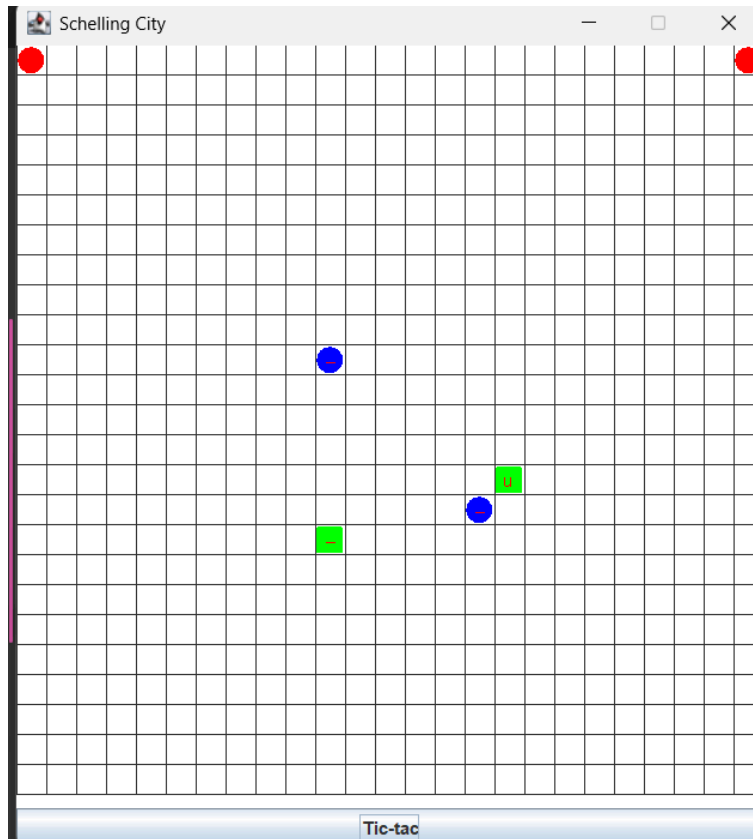
alert,

(a) ¿Cómo quedarían después de cuatro Tic-tac?

Luego de cuatro tic-Tac su color debe ser rojo nuevamente.

Ejecuten el programa y

hagan cuatro clics en el botón. Capturen una pantalla significativa.



(b) ¿Es correcto?

Es correcto pues el orden en el cual los colores del semaforo cambian es rojo, amarillo, verde, amarillo y rojo y así sucesivamente. Luego de cuatro tic-Tac será rojo nuevamente.

Ciclo 4. Nueva persona: Proponiendo y diseñando

El objetivo de este punto es permitir recibir en un nuevo tipo de Item.

(NO OLVIDE BDD – MDD)

1. Propongan, describan e implementen el nuevo tipo de persona.

(Mínimo dos pruebas de unidad)

El nuevo tipo de persona es katalan, este solo se moverá diagonalmente a la derecha. Estos son rosados, su forma es un círculo y si alrededor de ellos hay un ítem se ponen felices y se dejan de mover. Su estado inicial es insatisfecho.

2. Considerando una pareja de ellas con el apellido de ustedes.

(a) Piensen en otra prueba significativa y expliquen la intención.

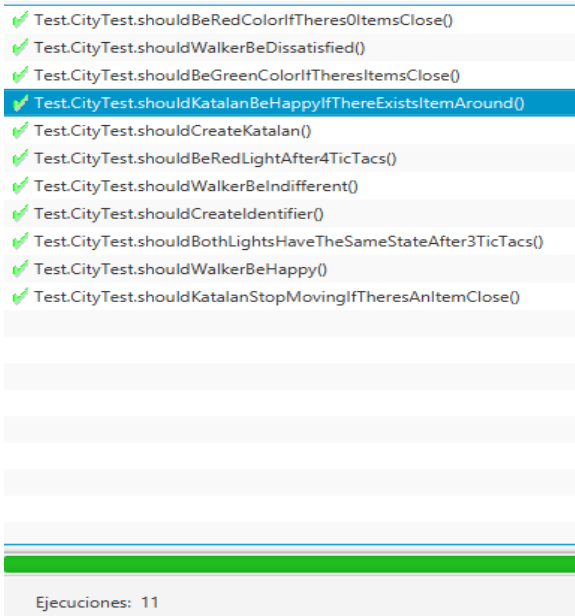
Katalan debe cambiar su estado a HAPPY si tiene ítems alrededor. Se quiere verificar si el estado de ánimo de estos realmente cambia.

(b) Codifiquen la prueba de unidad correspondiente y capturen la pantalla de resultados de ejecución de la prueba.

```
@Test
public void shouldKatalanBeHappyIfThereExistsItemAround(){
    Katalan pizza = new Katalan(city,18,13);
    Person eva = new Person(city,15,15);

    city.ticTac();
    city.ticTac();
    city.ticTac();

    char expectedState = 'h';
    char actualState = pizza.getState();
    assertEquals(expectedState,actualState);
}
```

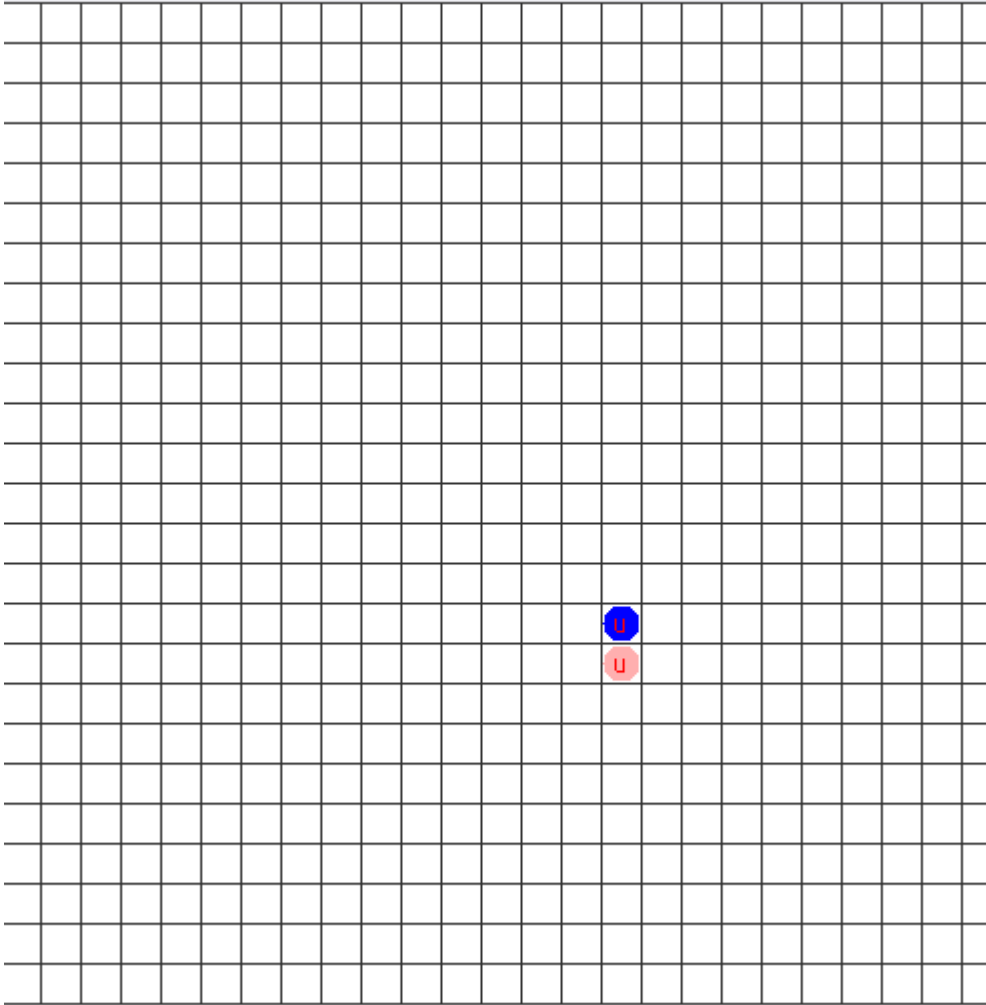


(c) Ejecute el programa con esa prueba como prueba de aceptación y capturen las pantallas correspondientes.

Luego de tres tic-Tac, el Katalan se mueve y cómo identifica ítems a su alrededor cambió su estado a HAPPY y dejó de moverse.



Schelling City



Tic-tac

Ciclo 5. Nuevo ítem: Proponiendo y diseñando

El objetivo de este punto es permitir recibir un nuevo ítem (no persona) en la ciudad

(NO OLVIDE BDD – MDD)

1. Propongan, describan e implementen el nuevo tipo de ítem.

(Mínimo dos pruebas de unidad)

El nuevo ítem es Identifier, este es un círculo e inicialmente su color es gris, si no tiene ítems alrededor su color pasa a ser rojo. Si un ítem se acerca a él pasa a ser verde. Si es verde y al verificar que no tiene ítems alrededor nuevamente pasa a ser rojo.

2. Considerando un par de ellos con el nombre de ustedes.

(a) Piensen en otra prueba significativa y expliquen la intención.

Verificar que esta pasa a ser de color rojo si no tiene ítems alrededor. Se realiza esta prueba para verificar que efectivamente cambia de estado y así mismo su color.

(b) Codifiquen la prueba de unidad correspondiente y capturen la pantalla de resultados de ejecución de la prueba.

```
@Test
public void shouldBeRedColorIfTheres0ItemsClose(){
    Identifier prueba = new Identifier(city,0,24);

    city.ticTac();

    char actualColor = prueba.getIdentfierState();
    char expectedColor = 'r';

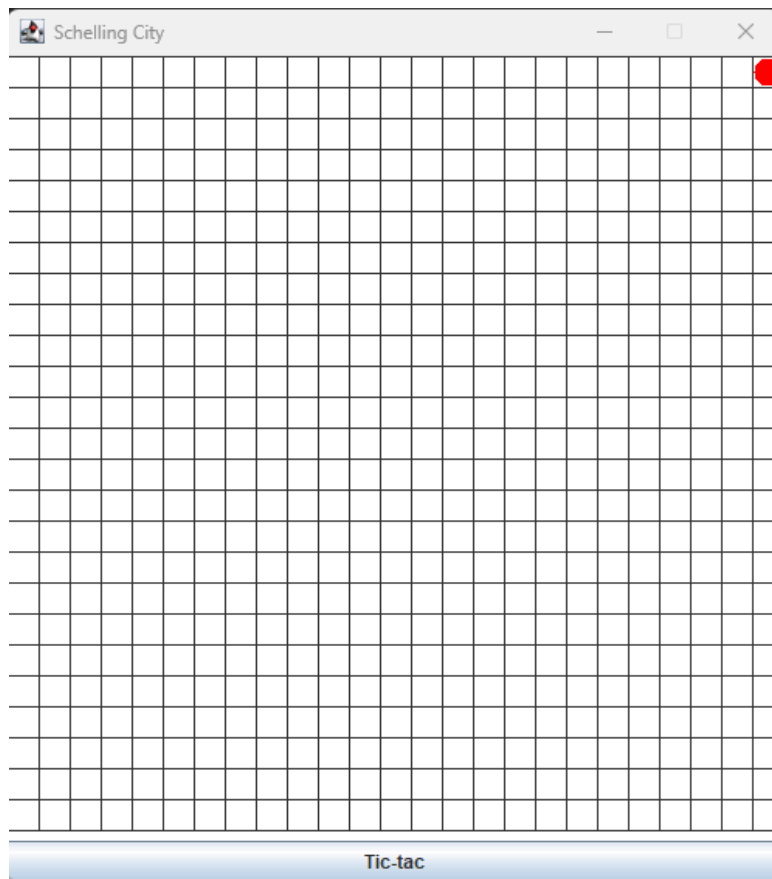
    assertEquals(expectedColor, actualColor);
}
```

✔ Test.CityTest.shouldBeRedColorIfTheres0ItemsClose()

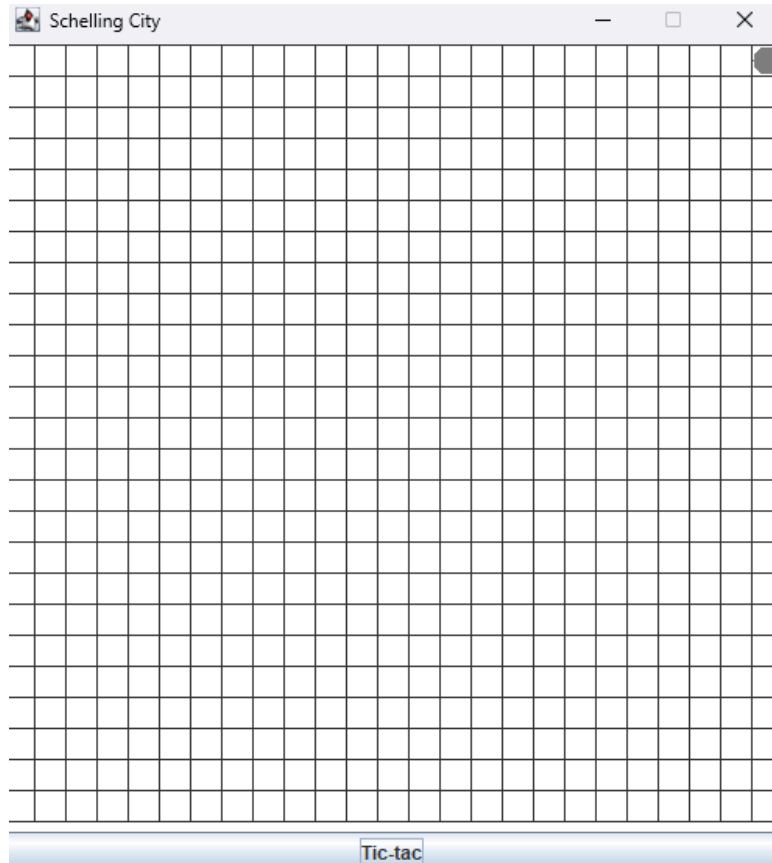
✔ Test.CityTest.shouldWalkerBeDissatisfied()

✔ Test.CityTest.shouldBeGreenColorIfTheresItemsClose()

(c) Ejecute el programa con esa prueba como prueba de aceptación y capturen las pantallas correspondientes.

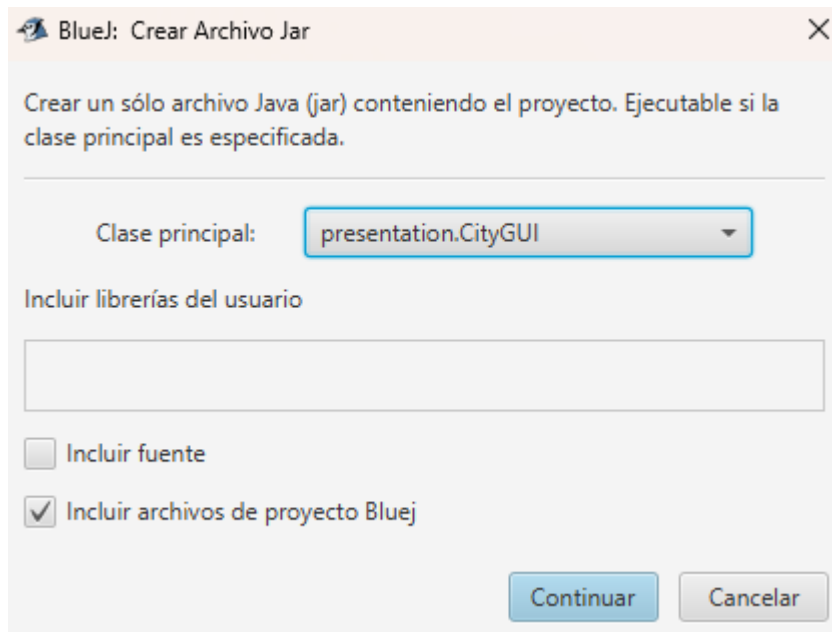


Es correcto puesto que inicialmente este era:

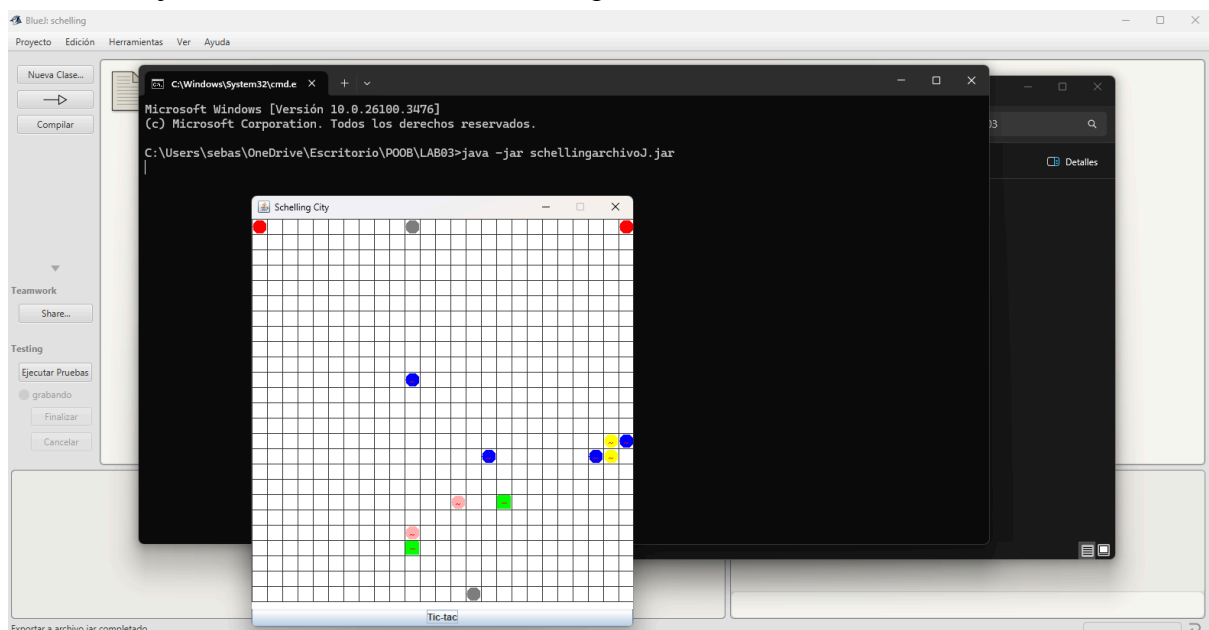


Empaquetando la versión final para el usuario. [En lab03.doc, schelling.asta , *.java, schelling.jar]

1. Revise las opciones de BlueJ para empaquetar su programa entregable en un archivo .jar. Genere el archivo correspondiente.



2. Consulte el comando java para ejecutar un archivo jar. ejecuten lo ¿qué pasa? Cuando se ejecuta se abre la cuadrícula correspondiente:



3. ¿Qué ventajas tiene esta forma de entregar los proyectos? Explique claramente.

Facilita que el cliente tenga que manejar diferentes archivos .class, además de facilitar el manejo del programa porque este contiene todos los archivos. Finalmente, permite ejecutar el programa con mayor facilidad mediante la consola o presionando doble click sobre el archivo .jar correspondiente.

DE BLUEJ A CONSOLA

En esta sección del laboratorio vamos a aprender a usar java desde consola. Para esto se va a trabajar con el proyecto del punto anterior..

Comandos básicos del sistema operativo [En lab03.doc]

Antes de iniciar debemos repasar los comandos básicos del manejo de la consola.

1. Investiguen los comandos para moverse en la estructura de directorios: crear, borrar, listar su contenido y copiar o eliminar un archivo.

Moverse en directorios

cd -> ver el directorio en el que estamos

cd NombreDirectorio -> cambiar a el directorio NombreDirectorio

cd .. -> regresar al directorio anterior

cd ruta de archivo -> ir a una ruta específica

cd/ -> directorio raíz

Creación De Directorios

mkdir "Nombre Directorio" -> crear directorio

Borrar Un Directorio

rmdir "Nombre Directorio" -> eliminar el directorio

rmdir /s /q "Nombre Directorio" -> elimina un directorio con:

/s → Elimina todos los archivos y subdirectorios dentro. /q → Sin confirmacion

Listar el contenido

dir

Solo Carpetas dir /ad

Archivos ocultos dir /a

Copiar Archivos

copy archivo.txt copia_archivo.txt

copy archivo.txt C:\Destino\

Eliminar archivo

del archivo.txt

Eliminar todos los archivos dentro de un directorio

del /s /q *.*

2. Organicen un nuevo directorio con la estructura propuesta para probar desde allí su habilidad con los comandos de consola.

Consulten y capturen el contenido de su directorio:

schelling

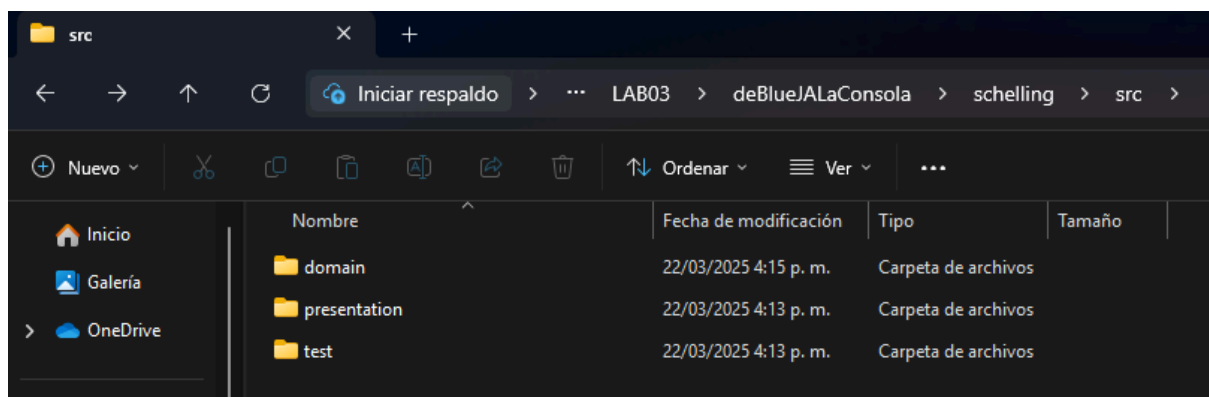
src

domain

presentation

test

```
C:\Users\julia\Downloads\LAB03\LAB03\DirectorioLab03>mkdir schelling
C:\Users\julia\Downloads\LAB03\LAB03\DirectorioLab03>cd schelling
C:\Users\julia\Downloads\LAB03\LAB03\DirectorioLab03\schelling>mkdir src
C:\Users\julia\Downloads\LAB03\LAB03\DirectorioLab03\schelling>cd src
C:\Users\julia\Downloads\LAB03\LAB03\DirectorioLab03\schelling\src>mkdir domain
C:\Users\julia\Downloads\LAB03\LAB03\DirectorioLab03\schelling\src>mkdir presentation
C:\Users\julia\Downloads\LAB03\LAB03\DirectorioLab03\schelling\src>mkdir test
C:\Users\julia\Downloads\LAB03\LAB03\DirectorioLab03\schelling\src>
```



3. En el directorio copien únicamente los archivos *.java del paquete de aplicación .

Consulte y capture el contenido de src/domain

```
Directorio de C:\Users\sebas\OneDrive\Escritorio\P00B\LAB03\deBlueJALaConsola\schelling\src\domain
21/03/2025 11:36 p. m. 1.298 Agent.java
22/03/2025 02:18 p. m. 4.190 City.java
21/03/2025 11:38 p. m. 1.746 Identifier.java
21/03/2025 11:39 a. m. 521 Item.java
21/03/2025 11:34 p. m. 1.771 Katalan.java
21/03/2025 11:41 p. m. 1.841 Light.java
22/03/2025 02:24 p. m. 1.310 Person.java
21/03/2025 11:05 p. m. 1.668 Schelling.java
21/03/2025 11:35 p. m. 1.697 Walker.java
9 archivos 16.042 bytes
0 dirs 164.123.193.344 bytes libres
```

Estructura de proyectos java [En lab03.doc]

En java los proyectos se estructuran considerando tres directorios básicos.

schelling

src : en src debe ir el código fuente del proyecto en .java organizado por paquetes.

bin: son los archivos .class compilados desde scr.

docs: es la documentación generada con javadoc.

1. Investiguen los archivos que deben quedar en cada una de esas carpetas y la organización interna de cada una de ellas.

-scr/

domain

archivos.java

presentation

.class que tiene el main

bin/

domain

archivos.class

presentation

.class que tiene el main

docs/

index.html

allclasses.html

main/

App.html

models/

Objetos.html

utils/

FileManager.html

2. ¿Qué archivos debería copiar del proyecto original al directorio bin? ¿Por qué?

Cópielos y consulte y capture el contenido del directorio que modificó.

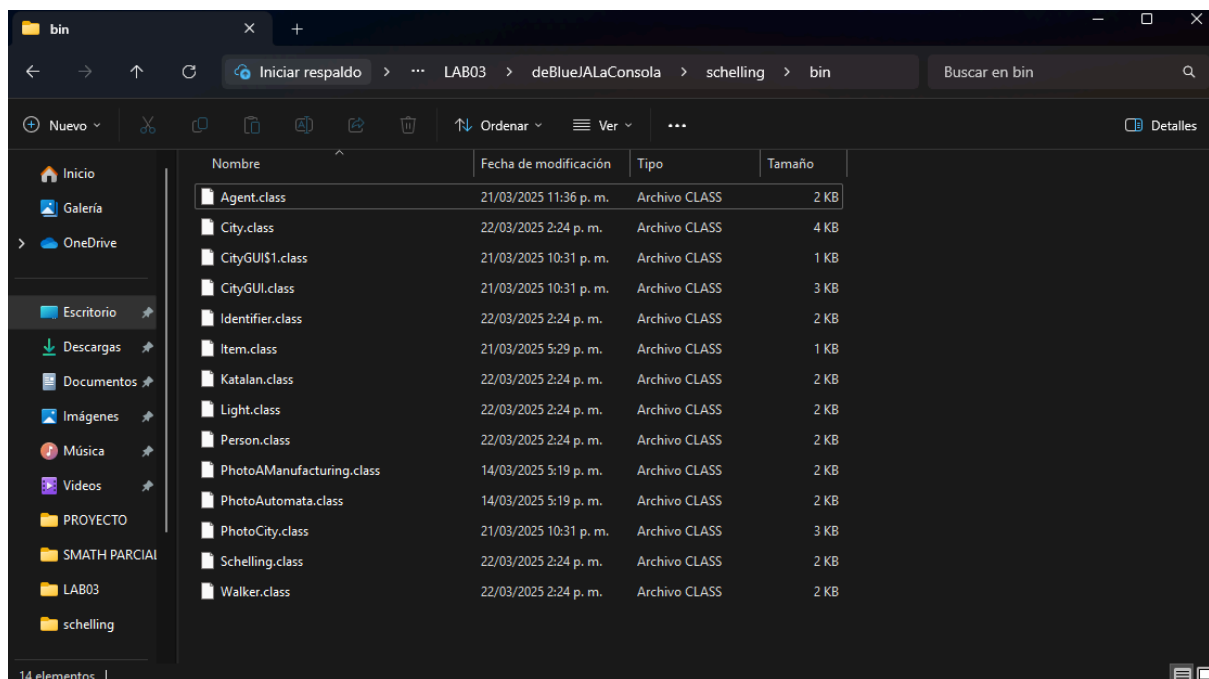
Los archivos que debería copiar del proyecto original son todos los que tienen la extensión .class porque son los archivos compilados que genera javac a partir de los .java

```
C:\Windows\System32\cmd.e x + v
C:\Users\sebas\OneDrive\Escritorio\POOB\LAB03\deBlueJALaConsola\schelling>cd ..
C:\Users\sebas\OneDrive\Escritorio\POOB\LAB03\deBlueJALaConsola\schelling>cd bin
C:\Users\sebas\OneDrive\Escritorio\POOB\LAB03\deBlueJALaConsola\schelling\bin>dir *
El volumen de la unidad C es Acer
El número de serie del volumen es: CE5F-EA33

Directorio de C:\Users\sebas\OneDrive\Escritorio\POOB\LAB03\deBlueJALaConsola\schelling\bin

22/03/2025 04:07 p. m. <DIR> .
22/03/2025 03:59 p. m. <DIR> ..
21/03/2025 11:36 p. m. 1.066 Agent.class
22/03/2025 02:24 p. m. 3.316 City.class
21/03/2025 10:31 p. m. 760 CityGUI$1.class
21/03/2025 10:31 p. m. 2.093 CityGUI.class
22/03/2025 02:24 p. m. 1.513 Identifier.class
21/03/2025 05:29 p. m. 697 Item.class
22/03/2025 02:24 p. m. 1.372 Katalan.class
22/03/2025 02:24 p. m. 1.436 Light.class
22/03/2025 02:24 p. m. 1.142 Person.class
14/03/2025 05:19 p. m. 1.984 PhotoAManufacturing.class
14/03/2025 05:19 p. m. 1.917 PhotoAutomata.class
21/03/2025 10:31 p. m. 2.277 PhotoCity.class
22/03/2025 02:24 p. m. 1.259 Schelling.class
22/03/2025 02:24 p. m. 1.382 Walker.class
14 archivos 22.214 bytes
2 dirs 164.124.741.632 bytes libres

C:\Users\sebas\OneDrive\Escritorio\POOB\LAB03\deBlueJALaConsola\schelling\bin>
```



Comandos de java [En lab03.doc]

1. Consulte para qué sirven cada uno de los siguientes comandos:

javac: compila los archivos .java a .class.

java: ejecuta archivos .class en la máquina virtual de java.

javadoc: genera la documentación en HTML a partir de los comentarios que están en el código.

jar: crea archivos .jar(Java Archive) para distribuir aplicaciones.

2. Cree una sesión de consola y consulte en línea las opciones de los comandos java y javac. Capture las pantallas.

```
C:\Users\julia>java --help
Usage: java [options] <mainclass> [args...]
        (to execute a class)
or java [options] -jar <jarfile>.jar [args...]
        (to execute a jar file)
or java [options] -m <module>[/<mainclass>] [args...]
   java [options] --module <module>[/<mainclass>] [args...]
        (to execute the main class in a module)
or java [options] <sourcefile>.java [args]
        (to execute a source-file program)

Arguments following the main class, source file, -jar <jarfile>.jar,
-m or --module <module>/<mainclass> are passed as the arguments to
main class.

where options include:

-cp <class search path of directories and zip/jar files>
-classpath <class search path of directories and zip/jar files>
--class-path <class search path of directories and zip/jar files>
        A ",-"separated list of directories, JAR archives,
        and ZIP archives to search for class files.
-p <module path>
--module-path <module path>...
        A ",-"separated list of elements, each element is a file path
        to a module or a directory containing modules. Each module is either
        a modular JAR or an exploded-module directory.
--upgrade-module-path <module path>...
        A ",-"separated list of elements, each element is a file path
        to a module or a directory containing modules to replace
        upgradeable modules in the runtime image. Each module is either
        a modular JAR or an exploded-module directory.
--add-modules <module name>[,<module name>...]
        root modules to resolve in addition to the initial module.
```

```
C:\Users\julia>javac --help
Usage: javac <options> <source files>
where possible options include:
@<filename>          Read options and filenames from file
-Akey[=value]         Options to pass to annotation processors
--add-modules <module>(<module>)*
        Root modules to resolve in addition to the initial modules,
        or all modules on the module path if <module> is ALL-MODULE-PATH.
--boot-class-path <path>, -bootclasspath <path>
        Override location of bootstrap class files
--class-path <path>, -classpath <path>, -cp <path>
        Specify where to find user class files and annotation processors
-d <directory>        Specify where to place generated class files
-deprecation
        Output source locations where deprecated APIs are used
--enable-preview
        Enable preview language features.
        To be used in conjunction with either -source or --release.
-encoding <encoding>  Specify character encoding used by source files
-endorseddirs <dirs>  Override location of endorsed standards path
-extdirs <dirs>       Override location of installed extensions
-g                   Generate all debugging info
-g:{lines,vars,source}
                    Generate only some debugging info
-g:none             Generate no debugging info
-h <directory>       Specify where to place generated native header files
--help, -help, -?    Print this help message
--help-extra, -X     Print help on extra options
-implicit:{none,class}
                    Specify whether to generate class files for implicitly referenced files
```

3. Busque la opción que sirve para conocer la versión a qué corresponden estos dos comandos. Documente el resultado.

```
C:\Users\julia>java --version
java 24 2025-03-18
Java(TM) SE Runtime Environment (build 24+36-3646)
Java HotSpot(TM) 64-Bit Server VM (build 24+36-3646, mixed mode, sharing)
```

Compilando [En lab03.doc]

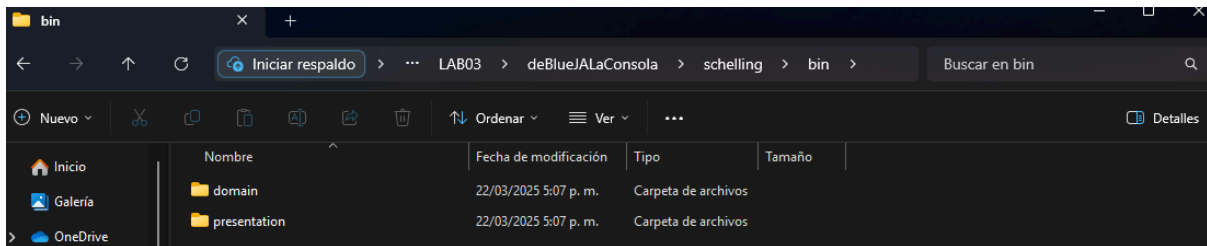
1. Utilizando el comando javac, desde el directorio raíz (desde schelling con una sola instrucción), compile el proyecto. ¿Qué instrucción completa tuvo que dar a la consola para compilar TODO el proyecto? Tenga presente que se pide un único

comando y que los archivos compilados deben quedar en los directorios respectivos.

- `javac -d bin -sourcepath src src/presentation/ *.java`

```
C:\Users\sebas\OneDrive\Escritorio\POOB\LAB03\deBlueJALaConsola\schelling>javac -d bin -sourcepath src src/presentation/ *.java
```

2. Revise de nuevo el contenido del directorio de trabajo y sus subdirectorios. ¿Cuáles ¿Qué nuevos archivos aparecen ahora y dónde se ubican?



Dentro de la carpeta bin se crean nuevos archivos que contiene los respectivos .class.

Documentando [En lab03.doc]

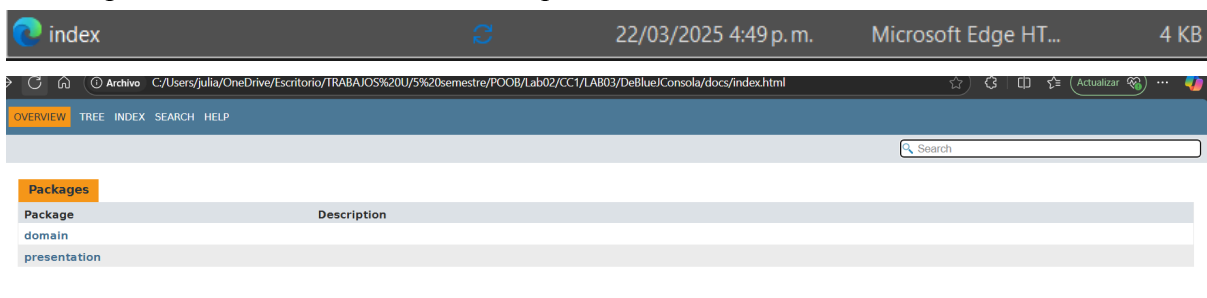
1. Utilizando el comando javadoc, desde el directorio raiz, genere la documentación (API) en formato html, en este directorio. ¿cuál es el comando completo para generar esta documentación?

El comando es:

- `javadoc -d docs -cp src -subpackages presentation domain`

2. ¿Qué archivo hay que abrir para empezar a navegar por la documentación? Ábralo y capture la pantalla.

Para empezar la documentación tenemos que meternos a **index.html**

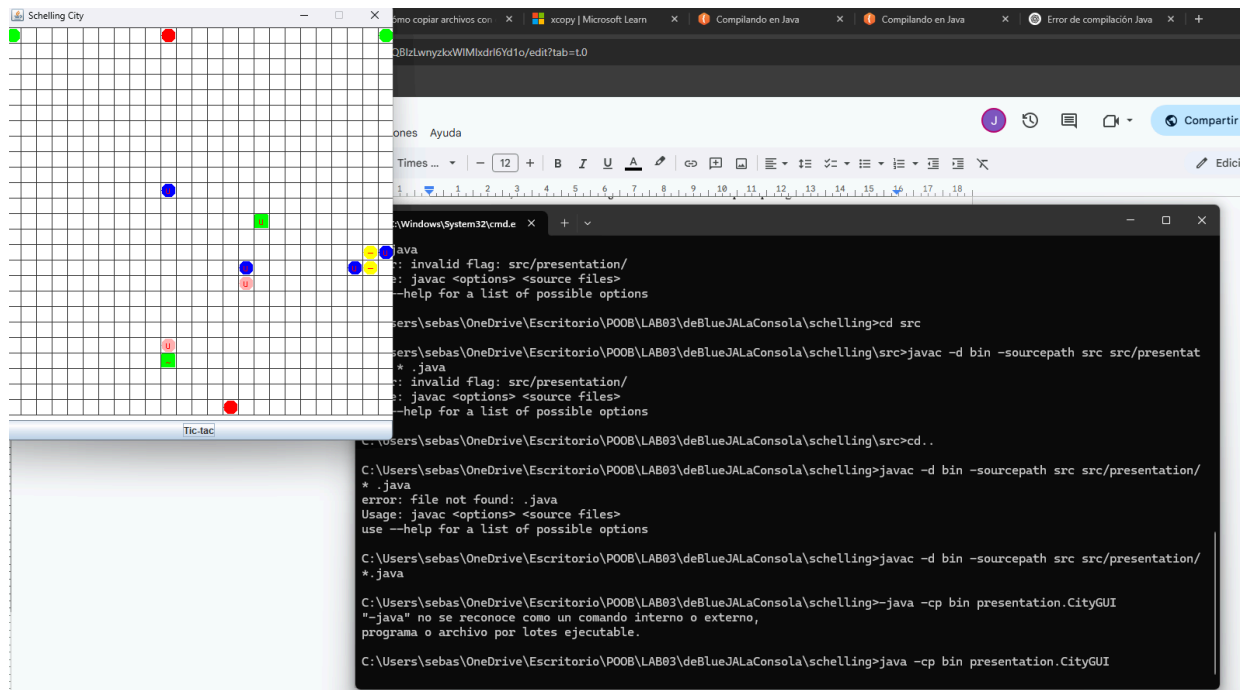


Ejecutando [En lab03.doc]

4. Empleando el comando java, desde el directorio raiz, ejecute el programa. ¿Cómo utilizó este comando?

Se utilizó:

- `java -cp bin presentation.CityGUI`



Probando [En lab03.doc]

1. Adicione ahora los archivos del directorio pruebas y trate de compilar nuevamente el programa.

Tenga en cuenta que estas clases requieren la librería junit 4.8.

¿Cómo se incluye un paquete para compilar? ¿Qué instrucción completa tuvo que dar a la consola para compilar?

`javac -d bin -cp "bin;lib/*" src/test/*.java`

2. Ejecute desde consola las pruebas . ¿Cómo utilizó este comando? Puede ver ejemplos de cómo ejecutar el “test runner” en How to run JUnit test cases from the command line.

`java -cp "bin;lib/*" org.junit.platform.console.ConsoleLauncher --class-path bin
--scan-class-path`

3. Pegue en su documento el resultado de las pruebas

Thanks for using JUnit! Support its development at <https://junit.org/sponsoring>

```
+-- JUnit Jupiter [OK]
+-- CityTest [OK]
|   +-- shouldBeRedColorIfTheres0ItemsClose() [OK]
|   +-- shouldWalkerBeDissatisfied() [OK]
|   +-- shouldBeGreenColorIfTheresItemsClose() [OK]
|   +-- shouldKatalanBeHappyIfThereExistsItemAround() [OK]
|   +-- shouldCreateKatalan() [OK]
|   +-- shouldBeRedLightAfter4TicTacs() [OK]
|   +-- shouldWalkerBeIndifferent() [OK]
|   +-- shouldCreateIdentifier() [OK]
|   +-- shouldBothLightsHaveTheSameStateAfter3TicTacs() [OK]
|   +-- shouldWalkerBeHappy() [OK]
|   +-- shouldKatalanStopMovingIfTheresAnItemClose() [OK]
+-- JUnit Vintage [OK]
+-- JUnit Platform Suite [OK]
```

```
Test run finished after 201 ms
[ 4 containers found ]
[ 0 containers skipped ]
[ 4 containers started ]
[ 0 containers aborted ]
[ 4 containers successful ]
[ 0 containers failed ]
[ 11 tests found ]
[ 0 tests skipped ]
[ 11 tests started ]
[ 0 tests aborted ]
[ 11 tests successful ]
[ 0 tests failed ]
```

C:\Users\julia\OneDrive\Escritorio\TRABAJOS U\5 semestre\POOB\Lab02\CC1\LAB03\DeBlueJConsola\schelling>

Empaquetando [En lab03.doc]

1. Consulte cómo utilizar desde consola el comando jar para empaquetar su programa entregable en un archivo .jar, que contenga los archivos bytecode necesarios (no las fuentes ni las clases de prueba), y que se pueda ejecutar al instalarlo en cualquier directorio, con solo tener la máquina virtual de java y su entorno de ejecución (JRE).

¿Cómo empaquetar jar ?

EMPAQUETAR

jar cfe schelling.jar presentation.CityGUI -C bin .

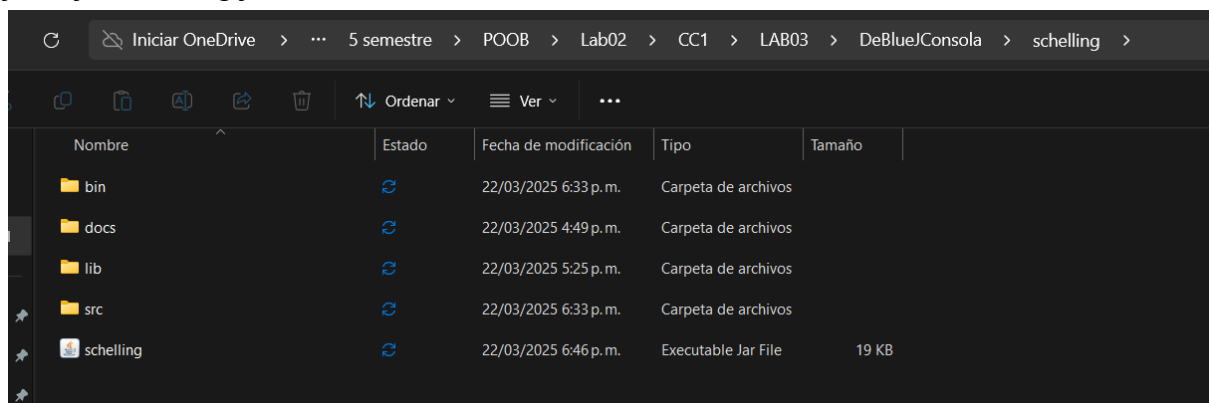
VERIFICAR

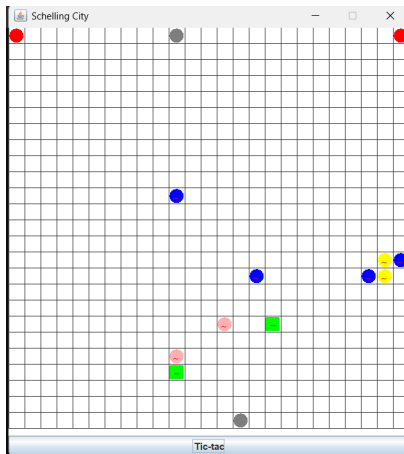
jar tf schelling.jar

2. ¿Cómo se ejecuta el proyecto empaquetado?

EJECUTAR

java -jar schelling.jar





domain/Schelling.class
domain/Schelling.java
domain/Walker.class
domain/Walker.java
presentation/
presentation/CityGUI1.class
presentation/CityGUI.class
presentation/PhotoCity.class

```
C:\Users\julia\OneDrive\Escritorio\TRABAJOS U\5 semestre\POOB\Lab02\CC1\LAB03\DeBlueJConsola\schelling>java -jar schelling.jar  
C:\Users\julia\OneDrive\Escritorio\TRABAJOS U\5 semestre\POOB\Lab02\CC1\LAB03\DeBlueJConsola\schelling>java -jar schelling.jar
```

```
stre\POOB\Lab02\CC1\LAB03\DeBlueJConsola\schelling>java -jar schelling.jar  
stre\POOB\Lab02\CC1\LAB03\DeBlueJConsola\schelling>jar cfe schelling.jar presentation.CityGUI -C bin .  
stre\POOB\Lab02\CC1\LAB03\DeBlueJConsola\schelling>jar tf schelling.jar
```


RETROSPECTIVA

1. ¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes? (Horas/Hombre)

Alrededor de 16 horas por cada uno

2. ¿Cuál es el estado actual del laboratorio? ¿Por qué? (Para cada método incluya su estado)

El estado actual del laboratorio está terminado. todos los métodos funcionalidades requeridas fueron implementados y probados correctamente, por lo cual cumple los objetivos del laboratorio.

3. Considerando las prácticas XP del laboratorio de hoy ¿por qué consideran que son importe?

Code must written to agreed standard: es importante mantener los estándares propios de java porque es bueno tener un código legible y limpio, lo hacemos mediante el nombre de las variables y demás.

Code The unit test first: es importante codificar la prueba de unidad primero para tener claro la funcionalidad de lo que se va a escribir y que el código fuera legible.

Consideramos que

4. ¿Cuál considera fue su mayor logro? ¿Por qué? ¿Cuál consideran que fue su mayor problema? ¿Qué hicieron para resolverlo?

Lograr desarrollar los puntos en consola con los comandos y demás. Nuestro mayor problema fue la ejecución de pruebas puesto que era necesario descargar múltiples archivos para probar el mismo, igualmente la implementación de las diferentes clases que se pedían.

Mayor problema: ejecutar las pruebas por consola lo resolvimos mediante investigación y detallar bien lo que estábamos haciendo (Nos fallaba por una t que estaba en minúscula)

5. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?

Como equipo, siempre tratamos de dividirnos el trabajo de manera equitativa donde podamos aprender los dos de manera igualada, también solemos trabajar en pair programming lo que nos ayuda a tener un mejor rendimiento a la hora de hacer el código, nos comprometemos a seguir mejorando la calidad de los laboratorios para que cada vez sea mejor la entrega de estos.

6. ¿Qué referencias usaron? ¿Cuál fue la más útil? Incluyan citas con estándares adecuados.

Documentación oficial de Java:

[Java Platform Standard Edition 8 Documentation](#)

Documentacion de JUnit

[JUnit 5 User Guide](#)

Tutoriales en línea:

[Guide to Creating and Running a Jar File in Java | Baeldung](#)

Referencia más útil la documentación oficial de java porque proporciona información precisa y confiable sobre herramientas y los comandos que usamos a lo largo del laboratorio.