**PROGRAMACIÓN ORIENTADA A OBJETOS**

**Herencia e interfaces**

**ADEMÁS Java desde consola**

**2024-1**

**Laboratorio 3/6**

**OBJETIVOS**

Desarrollar competencias básicas para:

1. Aprovechar los mecanismos de la herencia y el uso de interfaces.
2. Organizar las fuentes en paquetes.
3. Usar la utilidad jar de java para entregar una aplicación.
4. Extender una aplicación cumpliendo especificaciones de diseño, estándares y verificando su corrección.
5. Vivenciar las prácticas XP : Code must be written to agreed [standards](http://www.extremeprogramming.org/rules/standards.html). Code the [unit test first](http://www.extremeprogramming.org/rules/testfirst.html).



1. Utilizar los programas básicos de java (javac, java, javadoc, jar), desde la consola.

**ENTREGA**

 Incluyan en un archivo .zip los archivos correspondientes al laboratorio. El nombre debe ser los dos apellidos de los miembros del equipo ordenados alfabéticamente.

 Deben publicar el avance al final de la sesión y la versión definitiva en la fecha indicada en los espacios preparados para tal fin.

# DESARROLLO

## Contexto

El modelo de Schelling permite entender cómo a través de decisiones individuales de los agentes se pueden formar grupos segregados solo con la preferencia de vivir con un cierto porcentaje del grupo al que pertenece. Schelling usa una ciudad bidimensional de ubicaciones. En las ubicaciones se pueden colocar diferentes ítems. Uno de ellos son las personas que pueden estar en tres estados: feliz, triste e indiferente.

En una secuencia de pasos discretos, cada ítem primero decide lo que quiere hacer y luego cambia de acuerdo a esa decisión.

**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? b) ¿Cuál es el propósito del paquete presentación? c) ¿Cuál es el propósito del paquete dominio?

* a) En total tiene 2 paquetes: Presentation, domain.
* b) Creemos que el propósito del paquete de presentación es mostrar el GUI de la ciudad en sí, como se vería la ciudad de este laboratorio. Como una pequeña blue print.
* c) En el paquete dominio se puede apreciar todo el funcionamiento en sí del modelo de Schelling, la ciudad, personas, agentes e items. Es decir, todo lo que soporta en sí el programa.

1. Revisen el paquete de dominio, a)¿Cuáles son los diferentes tipos de componentes de este paquete? b) ¿Qué implica cada uno de estos tipos de componentes?

* a) Tiene dos clases **City, Person**, una interfaz **Item** y para finalizar una clase abstracta **Agent.**
* b) Respuestas:
  + **Clases (City, Person)**: Representan objetos concretos que pueden instanciarse en el programa.
  + **Interfaz (Item)**: Define un contrato que otras clases deben seguir si implementan esta interfaz.
  + **Clase abstracta (Agent)**: No puede instanciarse directamente y sirve como base para otras clases. Proporciona una estructura común y puede incluir métodos abstractos que deben ser implementados por sus subclases.

1. Revisen el paquete de presentación, a) ¿Cuántos componentes tiene? b) ¿Cuántos métodos públicos propios (no heredados) ofrece?

* a) Tiene solo 1 componente que es **CityGUI.**
* b) Se tiene 2 métodos públicos propios que son: gettheCity() y main(String[] args).

1. Para ejecutar un programa en java, Qué método se debe ejecutar? ¿En qué clase se encuentra?

* Se debe ejecutar el método de main, se encuentra en la clase de CityGUI como:

**public static void main(String[] args) {**

**CityGUI cg=new CityGUI();**

**cg.setVisible(true);**

**}**

1. Ejecuten el programa. ¿Qué funcionalidades ofrece? ¿Qué hace actualmente? ¿Por qué?

**(Deben ejecutar la aplicación java, no crear un objeto como lo veníamos haciendo)**

* El programa proporciona una interfaz gráfica (CityGUI) que representa una ciudad y permite actualizar su estado mediante un botón llamado "Tic-tac". Actualmente lo que hace es crear una instancia de City, que representa el modelo de la ciudad. Se dibujan líneas de cuadrícula según el tamaño de City. Se representan los elementos dentro de la ciudad (Item y Agent) con formas y colores específicos. Si el elemento es un agente (Agent), se muestra su estado emocional con un símbolo (u, \_, ~).
* Investigando un poco más a detalle el código sigue la estructura del **Modelo-Vista-Controlador (MVC)** porque separa claramente la lógica de la aplicación. El modelo (City) maneja los datos y la evolución de la ciudad, la vista (PhotoCity) se encarga de la representación gráfica y el controlador (CityGUI) gestiona la interacción del usuario, actualizando el modelo y redibujando la vista. Esta estructura permite modularidad y facilita el mantenimiento del código.

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

1. Consulte el significado de las palabras package e import de java. ¿Qué es un paquete? ¿Para qué sirve? ¿Para qué se importa? Explique su uso en este programa.

* Un **paquete** en Java es un conjunto de clases y otros elementos organizados en una estructura de directorios. Sirve para organizar el código, evitar conflictos de nombres y mejorar la modularidad del programa.
* La palabra clave **import** se usa para traer clases o paquetes externos al archivo actual, facilitando su uso sin necesidad de escribir su ruta completa.
* En este programa, package presentation; indica que CityGUI pertenece al paquete presentation, mientras que import domain.\*; importa todas las clases del paquete domain, lo que permite acceder a City y otros elementos sin especificar el paquete en cada uso.

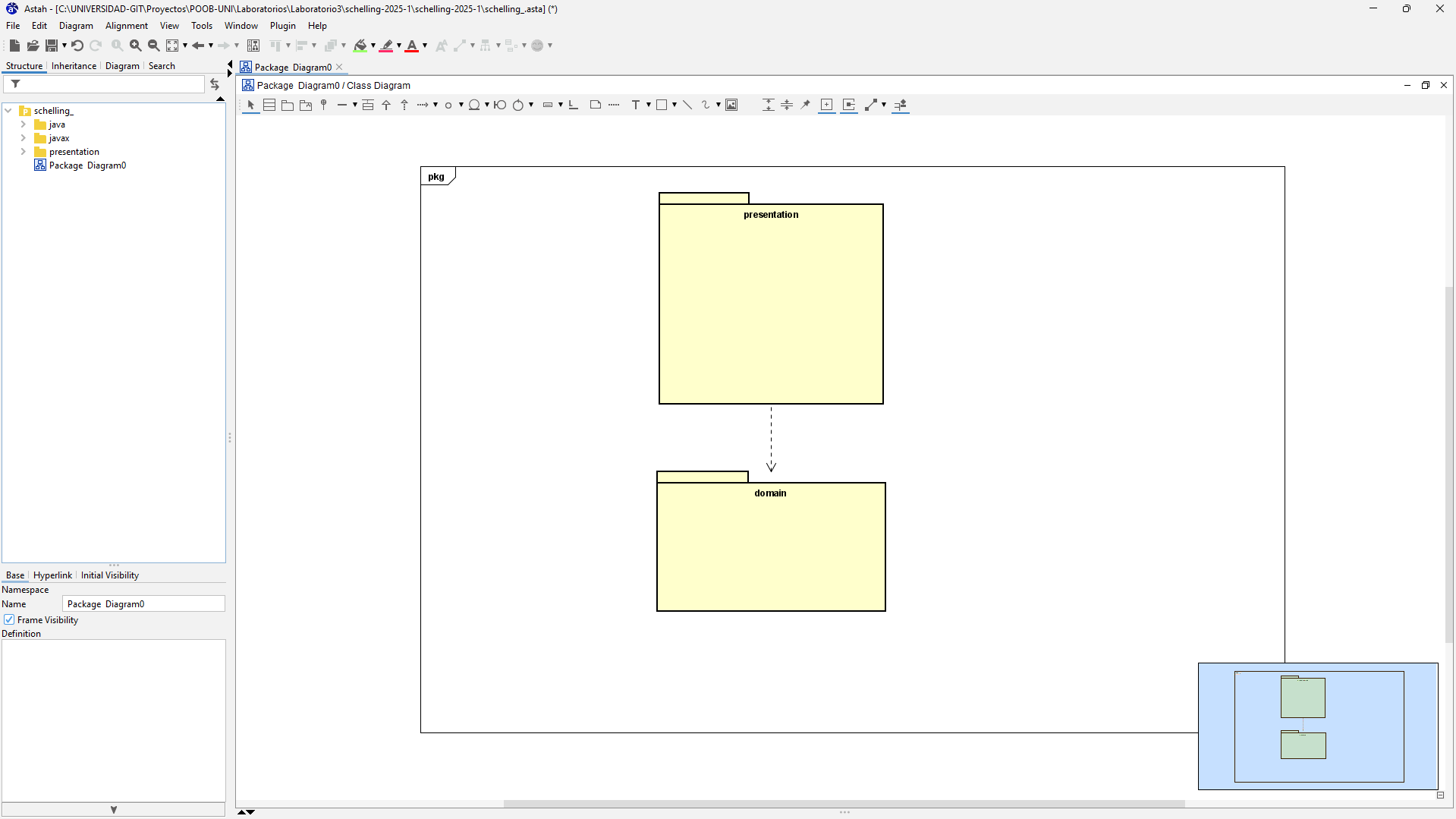
1. Revise el contenido del directorio de trabajo y sus subdirectorios. Describa su contenido. ¿Qué coincidencia hay entre paquetes y directorios?

* Al revisar el directorio de trabajo, se encuentran carpetas que representan los paquetes (presentation, domain, etc.), cada una conteniendo clases relacionadas.
* La coincidencia entre paquetes y directorios es directa: cada paquete en Java corresponde a un directorio en el sistema de archivos, asegurando una estructura ordenada y facilitando la gestión del código.

1. 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**

**Diagram0)**



**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?
2. Completen el diseño detallado del paquete de presentación. Adicionen el diagrama de clases al paquete correspondiente. a) ¿Por qué hay dos clases y un archivo .java?
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? ¿Por qué? ¿Asociado a qué clase? ¿Por qué?

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

## (NO OLVIDE BDD – MDD)

1. Estudie la clase City ¿Qué tipo de colección usa para albergar cosas? ¿Puede recibir personas? ¿Por qué?
2. Estudie el código asociado a la clase Person, ¿en qué estado se crea? ¿qué forma usa para pintarse? ¿cuándo aumenta su tiempo? ¿Qué clases definen la clase Person ? Justifique sus respuestas.
3. Person por ser un Agent, ¿qué atributos tiene? ¿qué puede hacer (métodos)? ¿qué decide hacer distinto? ¿qué no puede hacer distinto a todos los agentes? ¿qué debe aprender a hacer? Justifique sus respuestas.
4. Por comportarse como un Ítem, ¿qué sabe hacer? ¿qué decide hacer distinto? ¿qué no puede hacer distinto? ¿qué debe aprender a hacer? Justifique sus respuestas.
5. De acuerdo a lo anterior una Person, ¿Cómo actúa (decide+cambia)?
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? Capturen una pantalla significativa.
7. Diseñen, construyan y prueben el método llamado ticTac() de la clase City.
8. ¿Cómo quedarían adan y eva después de uno, dos, cuatro y seis **Tic-tac**? Ejecuten el programa. Capturen pantallas significativas en momentos correspondientes. ¿Es correcto?

**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 rectangulos de color verde; (ii) inician indiferentes; (ii) se mueven hacia el norte[[1]](#footnote-1), (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)?
2. Diseñen, construyan y prueben esta nueva clase. (Mínimo dos pruebas de unidad)
3. Adicione una pareja de caminantes, llámelas messner y kukuczka, (a) ¿Cómo quedarían después de tres **Tic-tac**? Ejecuten el programa y hagan tres clics en el botón. Capturen una pantalla significativa. (b) ¿Es correcto?

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

El objetivo de este punto es incluir semaforos (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?
2. Diseñen , construyan y prueben esta nueva clase. (Mínimo dos pruebas de unidad)
3. Adicionen dos semáforos en las esquinas superiores de la ciudad, llámenlos alarm y alert, (a) ¿Cómo quedarían después de cuatro **Tic-tac**? Ejecuten el programa y hagan cuatro clics en el botón. Capturen una pantalla significativa. (b) ¿Es correcto?

**Ciclo 4. Nueva persona: Proponiendo y diseñando**

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

## (NO OLVIDE BDD – MDD)

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

(Mínimo dos pruebas de unidad)

1. Considerando una pareja de ellas con el apellido de ustedes. (a) Piensen en otra prueba significativa y expliquen la intención. (b) Codifiquen la prueba de unidad correspondiente y capturen la pantalla de resultados de ejecución de la prueba. (c) Ejecuten el programa con esa prueba como prueba de aceptación y capturen las pantallas correspondientes.

### Ciclo 5. Nuevo item: Proponiendo y diseñando

El objetivo de este punto es permitir recibir un nuevo item (no persona) en la ciudad **(NO OLVIDE BDD – MDD)**

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

(Mínimo dos pruebas de unidad)

1. Considerando un par de ellos con el nombre de ustedes. (a) Piensen en otra prueba significativa y expliquen la intención. (b) Codifiquen la prueba de unidad correspondiente y capturen la pantalla de resultados de ejecución de la prueba. (c) Ejecuten el programa con esa prueba como prueba de aceptación y capturen las pantallas correspondientes.

**Ciclo 6. BONO. Persona Schelling** [**[http://ncase.me/polygons-es/]**](http://ncase.me/polygons-es/) La persona Schelling se rigue por las siguientes reglas:

* Las personas Schelling sólo se mueven si están insatisfechos con su vecindad.
* Cuando están bien donde están, las personas Schelling no se mueven.
* Están indiferentes si todos los vecinos son como ellos o no tienen vecinos.
* Están insatisfechos si menos de ⅓ de los vecinos no son como ellos.
* Están satisfechos si más de ⅓ de los vecinos son como ellos y no todos son como ellos.

Primero todas las personas Schelling toman la decisión de lo que pasará en el tiempo siguiente y luego la realizan.

**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. ejecutenlo ¿qué pasa?
3. ¿Qué ventajas tiene esta forma de entregar los proyectos? Explique claramente.

**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] es 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.
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

1. En el directorio copien únicamente los archivos \*.java del paquete de aplicación . Consulte y capture el contenido de src/domain

**Estructura de proyectos java** [En lab03.doc]

En java los proyectos se estructuran considerando tres directorios básicos.

schelling src bin docs

1. Investiguen los archivos que deben quedar en cada una de esas carpetas y la organización interna de cada una de ellas.
2. ¿Qué archivos debería copiar del proyecto original al directorio bin? ¿Por qué?

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

**Comandos de java** [En lab03.doc]

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

javac java javadoc jar

1. Cree una sesión de consola y consulte en línea las opciones de los comandos java y javac. Capture las pantallas.
2. Busque la opción que sirve para conocer la versión a qué corresponden estos dos comandos. Documente el resultado.

**Compilando** [En lab03.doc]

1. Utilizando el comando javac, **desde el directorio raíz (desde schelling con una sóla 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.
2. Revise de nuevo el contenido del directorio de trabajo y sus subdirectorios. ¿Cuáles nuevos archivos aparecen ahora y dónde se ubican?

**Documentando** [En lab03.doc]

1. Utilizando el comando javadoc, desde el directorio raíz, genere la documentación (API) en formato html, en este directorio. ¿Cuál es el comando completo para generar esta documentación?
2. ¿Cuál archivo hay qué abrir para empezar a navegar por la documentación? Ábralo y capture la pantalla.

**Ejecutando**  [En lab03.doc]

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

**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?
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](https://stackoverflow.com/questions/2235276/how-to-run-junit-test-cases-from-the-command-line) 3. Pegue en su documento el resultado de las pruebas

**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 empaquetó jar ?
2. ¿Cómo se ejecuta el proyecto empaquetado?

**RETROSPECTIVA**

1. ¿Cuál fue el tiempo total invertido en el laboratorio por cada uno de ustedes? (Horas/ Hombre)
2. ¿Cuál es el estado actual del laboratorio? ¿Por qué? (Para cada método incluya su estado)
3. Considerando las prácticas XP del laboratorio de hoy ¿por qué consideran que son importe?
4. ¿Cuál consideran fue su mayor logro? ¿Por qué? ¿Cuál consideran que fue su mayor problema? ¿Qué hicieron para resolverlo?
5. ¿Qué hicieron bien como equipo? ¿Qué se comprometen a hacer para mejorar los resultados?
6. ¿Qué referencias usaron? ¿Cuál fue la más útil? Incluyan citas con estándares adecuados.

1. Primero, todos los caminantes deciden hacia dónde se van a mover y luego todos los caminantes se mueven.

   Dos caminantes no se pueden mover hacia la misma posición. [↑](#footnote-ref-1)