UNIVERSIDAD ADOLFO IBÁÑEZ

# Object Oriented Programming
# TICS-201 (Stgo)

Professor: Ricardo Seguel, Ph.D.

13-Abr-2018

UNIVERSIDAD ADOLFO IBÁÑEZ

# Test 3 (30 min)

Professor: Ricardo Seguel, Ph.D.

13-Abr-2018

# Control 3 - (30 min)

- Descargue el archive Gato_control.java desde Webcursos
- El archivo contiene el código del famoso juego del “GATO”
- Lea y entienda lo que hace el código con las funciones definidas en la clase Java
- Complete las líneas de código que faltan en la función ***main()*** para ejecutar apropiadamente el juego que simula 2 jugadores virtuales.
- Ejecute el juego y pruebe que funciona bien ingresando las movidas de 2 jugadores virtuales en la línea de comandos.
- Suba su archivo Java al link habilitado en Webcursos.

```
Player 'X', enter your move (row[1-3] column[1-3]): 2 2
   |   |
-----------
   | X |
-----------
   |   |

Player 'O', enter your move (row[1-3] column[1-3]): 1 1
 O |   |
-----------
   | X |
-----------
   |   |

Player 'X', enter your move (row[1-3] column[1-3]): 1 3
 O |   | X
-----------
   | X |
-----------
   |   |

Player 'O', enter your move (row[1-3] column[1-3]): 3 1
 O |   | X
-----------
   | X |
-----------
 O |   |

Player 'X', enter your move (row[1-3] column[1-3]): 2 2
This move at (2,2) is not valid. Try again...
Player 'X', enter your move (row[1-3] column[1-3]): 2 3
 O |   | X
-----------
   | X | X
-----------
 O |   |

Player 'O', enter your move (row[1-3] column[1-3]): 2 1
 O |   | X
-----------
 O | X | X
-----------
 O |   |

Player 'O' won!
```

UNIVERSIDAD ADOLFO IBÁÑEZ

# Answers

Professor: Ricardo Seguel, Ph.D.

13-Abr-2018

# Solution

```
/** The entry main method (the program starts here) */
public static void main(String[] args) {

 // Initialize the game-board and current status
      initGame();
      // Play the game once
      do {
         playerMove(currentPlayer); // update currentRow and currentCol
         updateGame(currentPlayer, currntRow, currentCol); // update currentState
         printBoard();
         // Print message if game-over
         if (currentState == CROSS_WON) {
            System.out.println("'X' won! Bye!");
         } else if (currentState == NOUGHT_WON) {
            System.out.println("'O' won! Bye!");
         } else if (currentState == DRAW) {
            System.out.println("It's a Draw! Bye!");
         }
         // Switch player
         currentPlayer = (currentPlayer == CROSS) ? NOUGHT : CROSS;
      } while (currentState == PLAYING); // repeat if not game-over
}
```

**Check this out!**

# Ok, let’s continue…

“Java is not magical, it's a tool for people to build stuff. It's the hammer and nail. It's turns out to be easy to do things like AI and other fairly intelligent systems.”

“**It makes my head explode when there are people who think you can do everything in HTML**”

“An API that isn't comprehensible isn't usable.”

James Gosling

# What we have studied until now

- Diagnostic of student programming skills
- Idea and coding of the assistant (course project)
- Key basic concepts of Programming
- Why Java
- Key concepts of Java
  - Data types
  - Operators
  - Strings
  - Input/Output
  - Loops
  - Conditions
  - Functions, Libraries and Scope
  - Recursion
  - Arrays
  - Object and Classes in Java and UML
- Chapters 1-8 of the Java book

# What we'll study today

- More about Objects
- Hands-on of the exercises
- Optional homework exercises of Chapters 7, 8, 9 and 10 of the guide Java Book of the course
  - **Optional means** that if you didn't have time to complete the exercise in the week, anyways you will study them after for preparing for P1.
  - **Optional does not mean** the exercise's contents will not be part of P1.
  - Students that missed the submission of one or more assignments ***must*** do the homework exercises and upload them to GitHub.

# Let's continue with "El Gato"

```
// Named-constants to represent the various states of the game
public static final int PLAYING = 0;
public static final int DRAW = 1;
public static final int CROSS_WON = 2;
public static final int NOUGHT_WON = 3;

// The current state of the game
public static int currentState = PLAYING;  // Assigned to a name, which is easier to read and understand,
                                           // instead of an int number 0
```

This approach of using int named-constants is better than using number in the programming statements, but it is not ideal. This is because you may inadvertently assign an int value outside the valid range to the variable ***currentState***. For example:

```
currentState = 99;   // A logical error but can compile
```

# Enumerations

- JDK 1.5 introduces a new feature called enumeration, which is a special class for storing an enumeration (list) of items. In our case, we can define an enumeration called GameState as follows:

```
/**
 *  Enumerations for the various states of the game
 */
public enum GameState {  // to save as "GameState.java"
   PLAYING, DRAW, CROSS_WON, NOUGHT_WON
}
```

To reference an item in an enum, use enumName.itemName (e.g., GameState.PLAYING and GameState.DRAW), just like referencing static variables of a class (e.g., Math.PI).
You can create an instance for an enum (just like creating an instance of a class) and assign a value into it. We shall now declare the variable currentState as an instance of GameState, which can take the value of GameState.PLAYING, GameState.DRAW, GameState.CROSS_WON, and GameState.NOUGHT_WON.

# Reference to enumerations

- Take note that you can only assign a value defined in the enumeration (such as GameState.PLAYING, GameState.DRAW), and NOT an arbitrary int value in the earlier example. Enum is SAFE!

```
private GameState currentState;    // declare variable currentState as an instance of enum GameState
currentState = GameState.PLAYING;  // assign a value (an enum item) to the variable currentState
```

- We shall also create an enum called ***Seed*** for the various seeds and cell contents.

```
1  /**
2   * Enumerations for the seeds and cell contents
3   */
4  public enum Seed { // to save as "Seed.java"
5     EMPTY, CROSS, NOUGHT
6  }
```

# Enumerating the seed

- Again, you need to use Seed.EMPTY, Seed.CROSS, Seed.NOUGHT to refer to these values, just like any public static variable.

```
private Seed currentPlayer;  // declare variable currentPlayer as an instance of Seed
currentPlayer = Seed.CROSS;  // assign a value (an enum item) to the variable currentPlayer

private Seed content;   // cell's content
content = Seed.EMPTY;
```

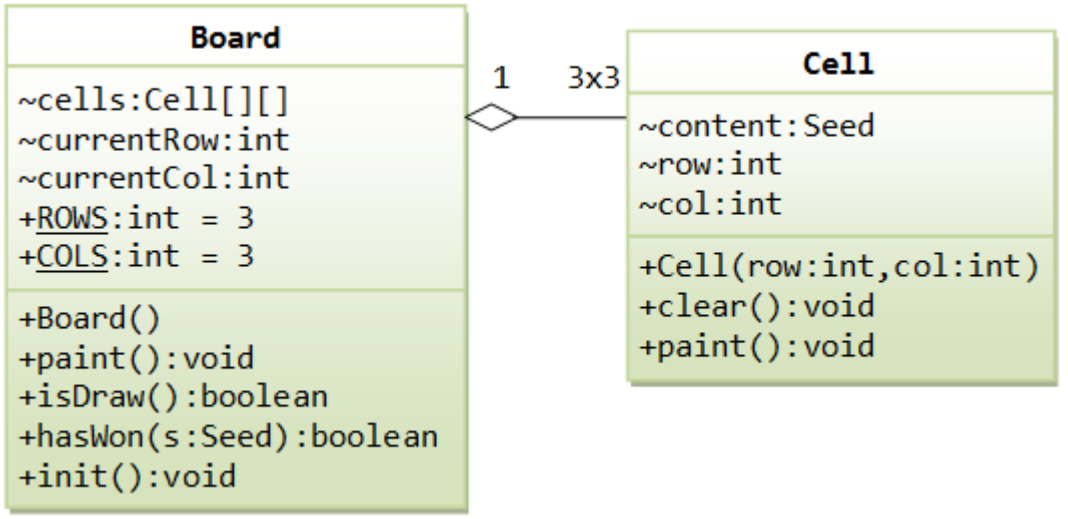
- We shall declare the variables currentPlayer and content as instances of enum Seed.
- In brief, **an enum is just a special class with a list of named-constants**.
- But **enum is safe, compared with name-constants**.

# Classes Board and Cell

- We begin with two classes, a class Cell for each individual cell of the game board, and a class Board for the 3x3 game board.

- The Cell class has an instance variable called content (with package access), of the type enum Seed. You can only assign a value from the enum's constants, such as Seed.EMPTY, Seed.CROSS, and Seed.NOUGHT, into content. A Cell can paint() itself and has its own operations such as clear().

- The Board class composes of nine Cell instances, arranged in an 3×3 array called cells (with package access), of the type Cell[][]. A Board can paint() itself, and supports its own operations such as checking the status of the current board (isDraw(), hasWon()).



# Cell.java

```java
/**
 * The Cell class models each individual cell of the game board.
 */
public class Cell {  // save as Cell.java
   // package access
   Seed content; // content of this cell of type Seed.
                 // take a value of Seed.EMPTY, Seed.CROSS, or Seed.NOUGHT
   int row, col; // row and column of this cell, not used in this program

   /** Constructor to initialize this cell */
   public Cell(int row, int col) {
      this.row = row;
      this.col = col;
      clear();  // clear content
   }

   /** Clear the cell content to EMPTY */
   public void clear() {
      content = Seed.EMPTY;
   }

   /** Paint itself */
   public void paint() {
      switch (content) {
         case CROSS:  System.out.print(" X "); break;
         case NOUGHT: System.out.print(" O "); break;
         case EMPTY:  System.out.print("   "); break;
      }
   }
}
```

UNIVERSIDAD ADOLFO IBÁÑEZ

# Board.java

```java
/**
 * The Board class models the game-board.
 */
public class Board {  // save as Board.java
   // Named-constants for the dimensions
   public static final int ROWS = 3;
   public static final int COLS = 3;

   // package access
   Cell[][] cells;  // a board composes of ROWS-by-COLS Cell instances
   int currentRow, currentCol;  // the current seed's row and column

   /** Constructor to initialize the game board */
   public Board() {
      cells = new Cell[ROWS][COLS];  // allocate the array
      for (int row = 0; row < ROWS; ++row) {
         for (int col = 0; col < COLS; ++col) {
            cells[row][col] = new Cell(row, col); // allocate element of the array
         }
      }
   }

   /** Initialize (or re-initialize) the contents of the game board */
   public void init() {
      for (int row = 0; row < ROWS; ++row) {
         for (int col = 0; col < COLS; ++col) {
            cells[row][col].clear();  // clear the cell content
         }
      }
   }

   /** Return true if it is a draw (i.e., no more EMPTY cell) */
   public boolean isDraw() {
      for (int row = 0; row < ROWS; ++row) {
         for (int col = 0; col < COLS; ++col) {
```

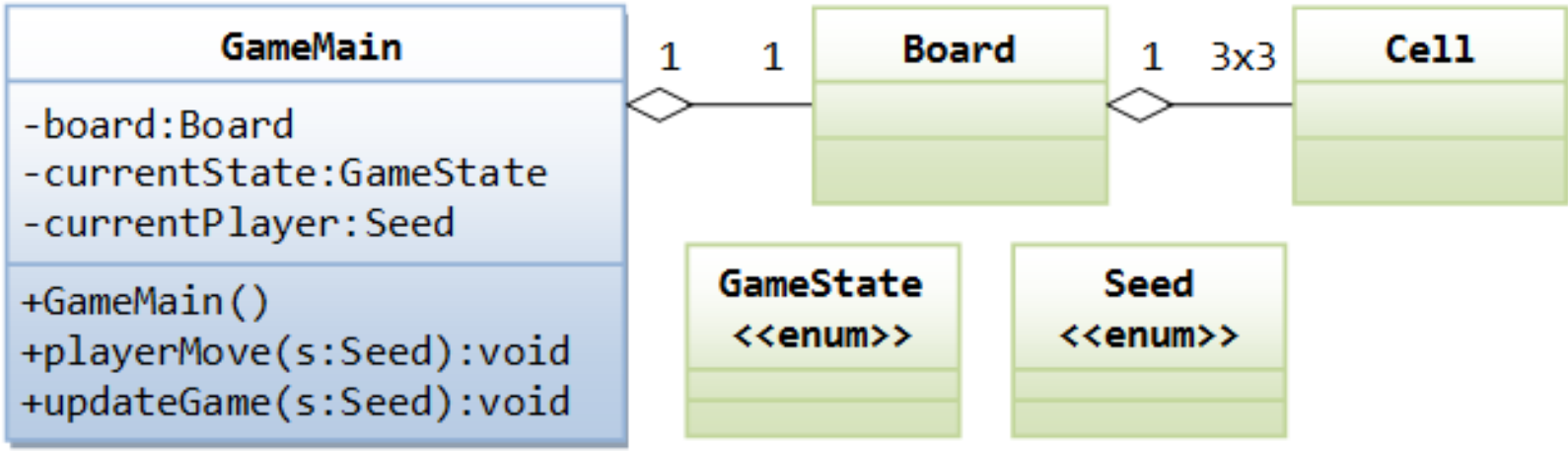
UNIVERSIDAD ADOLFO IBÁÑEZ

# Board.java

```
36              if (cells[row][col].content == Seed.EMPTY) {
37                 return false; // an empty seed found, not a draw, exit
38              }
39           }
40        }
41        return true; // no empty cell, it's a draw
42     }
43
44     /** Return true if the player with "theSeed" has won after placing at
45         (currentRow, currentCol) */
46     public boolean hasWon(Seed theSeed) {
47        return (cells[currentRow][0].content == theSeed         // 3-in-the-row
48                    && cells[currentRow][1].content == theSeed
49                    && cells[currentRow][2].content == theSeed
50                || cells[0][currentCol].content == theSeed      // 3-in-the-column
51                    && cells[1][currentCol].content == theSeed
52                    && cells[2][currentCol].content == theSeed
53                || currentRow == currentCol            // 3-in-the-diagonal
54                    && cells[0][0].content == theSeed
55                    && cells[1][1].content == theSeed
56                    && cells[2][2].content == theSeed
57                || currentRow + currentCol == 2    // 3-in-the-opposite-diagonal
58                    && cells[0][2].content == theSeed
59                    && cells[1][1].content == theSeed
60                    && cells[2][0].content == theSeed);
61     }
62
63     /** Paint itself */
64     public void paint() {
65        for (int row = 0; row < ROWS; ++row) {
66           for (int col = 0; col < COLS; ++col) {
67              cells[row][col].paint();   // each cell paints itself
68              if (col < COLS - 1) System.out.print("|");
69           }
70           System.out.println();
71           if (row < ROWS - 1) {
72              System.out.println("-----------");
73           }
74        }
75     }
76  }
```

# Class GameMain

- Finally, let's write a main class called GameMain to pull all the pieces together.
- GameMain acts as the overall controller for the game.

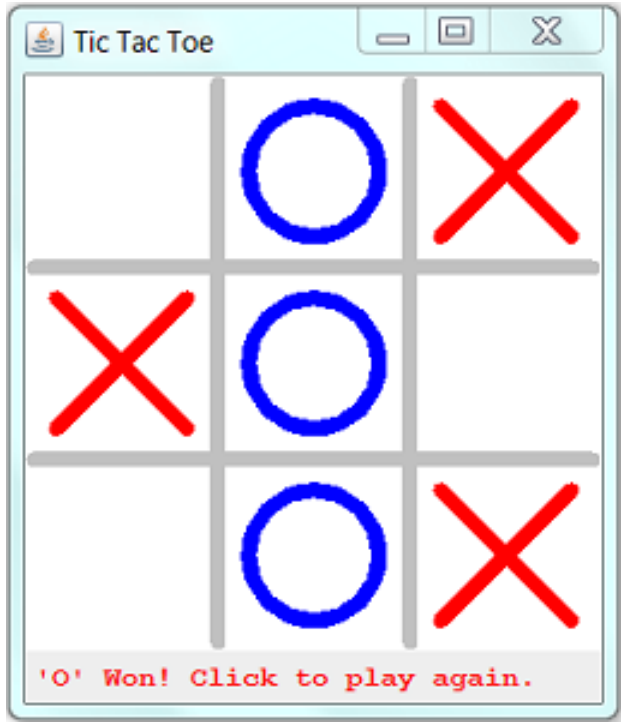



**Now we have our Gato game programmed in Java using Object Oriented Programming!!**

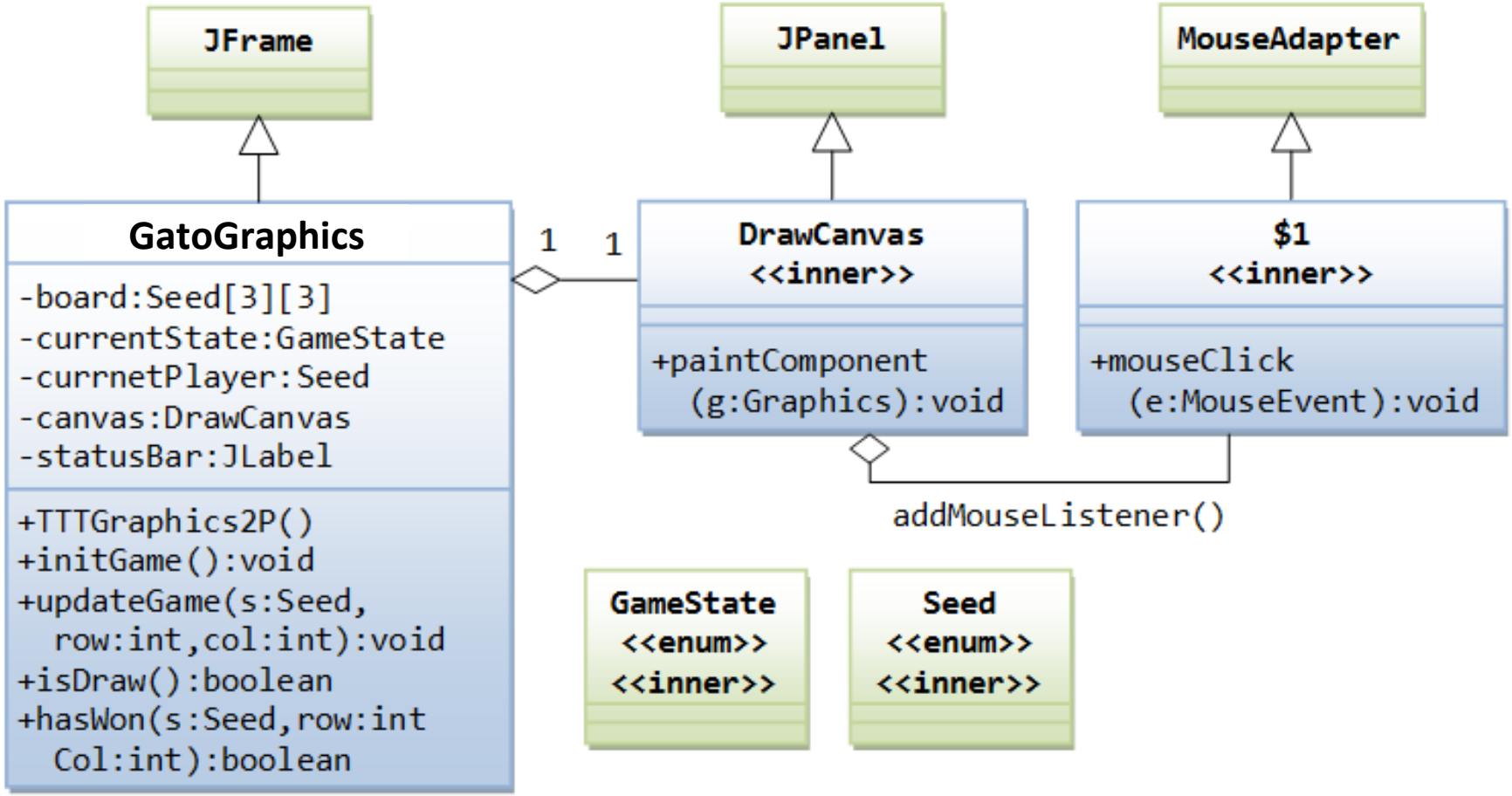
**Download the source code of this example from Webcursos and make it run!**

# Upgrading our Gato game using Java 2D Graphics with SWING

- A **Java Swing application** is illustrated.
- In this initial design, we do not separate the cell and board into dedicated classes, but include them in the main class.
- We used an inner class DrawCanvas (that extends JPanel) to do the custom drawing, and an anonymous inner class for MouseListener.
- The content-pane (of the top-level container JFrame) is set to BorderLayout.
- The DrawCanvas (JPanel) is placed at the CENTER; while a status-bar (a JLabel) is placed at the SOUTH (PAGE_END).



# The class diagram using graphics with Swing



# Let's play using the graphics Gato version with Swing

- **Download the source code from Webcursos and make it run!**

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
```

- Change the colors of the X and O
- Is it posible to change the board background color?

- Check the documentation in the Java API
- https://docs.oracle.com/javase/8/docs/api/

# Let's add sound to the Gato game!

- **Download the Java class SoundTest.java from Webcursos**
- Prepare a sound file called "gameover.wav" (try googling) and place it under under your project root (or "bin") directory.
- If you receive error message "Couldn't find file: xxx". Try moving your file around your project (such as under the "bin" sub-directory).
- **JDK supports only a sampled audio format**, ".wav", ".au", and ".aiff".
- It does not support ".mp3" (You will get an error message "Audio Format not supported: xxx". There used to be a "Java Media Framework (JMF)" that supports MP3.
- You may try to download the pronunciations for the words "game" and "over", and join them into a "wav" file.
- We need to test the sound effect under a Swing application, instead of placing all the codes under the main(). This is because main() exits before the sound gets a chance to play.

# Adding the sound effect to the graphics version of the Gato game

- Two sound clips were used in the demo: one for the move ("move.wav") and the other for game-over ("gameover.wav"). (Google to find some interesting "wav" files.)

- Include the following codes at the appropriate locations:

```
// Add the following import statements
import java.io.IOException;
import java.net.URL;
import javax.sound.sampled.*;

......

// Declare the following variables for the sound clips in the main class
String fileMove = "sounds/move.wav";         // audio filename for move effect
String fileGameOver = "sounds/gameover.wav"; // audio filename for game-over effect
Clip soundClipMove;      // Sound clip for move effect
Clip soundClipGameOver;  // Sound clip for game-over effect

......
```

# Preparing the sound clips

```
// In the main class's Constructor, prepare the sound clips
try {
   URL url = this.getClass().getClassLoader().getResource(fileGameOver);
   if (url == null) {
      System.err.println("Couldn't find file: " + fileGameOver);
   } else {
      AudioInputStream audioIn = AudioSystem.getAudioInputStream(url);
      soundClipGameOver = AudioSystem.getClip();
      soundClipGameOver.open(audioIn);
   }

   url = this.getClass().getClassLoader().getResource(fileMove);
   if (url == null) {
      System.err.println("Couldn't find file: " + fileMove);
   } else {
      AudioInputStream audioIn = AudioSystem.getAudioInputStream(url);
      soundClipMove = AudioSystem.getClip();
      soundClipMove.open(audioIn);
   }
} catch (UnsupportedAudioFileException e) {
   System.err.println("Audio Format not supported!");
} catch (Exception e) {
   e.printStackTrace();
}

......
```

# Sound effect in the event handler

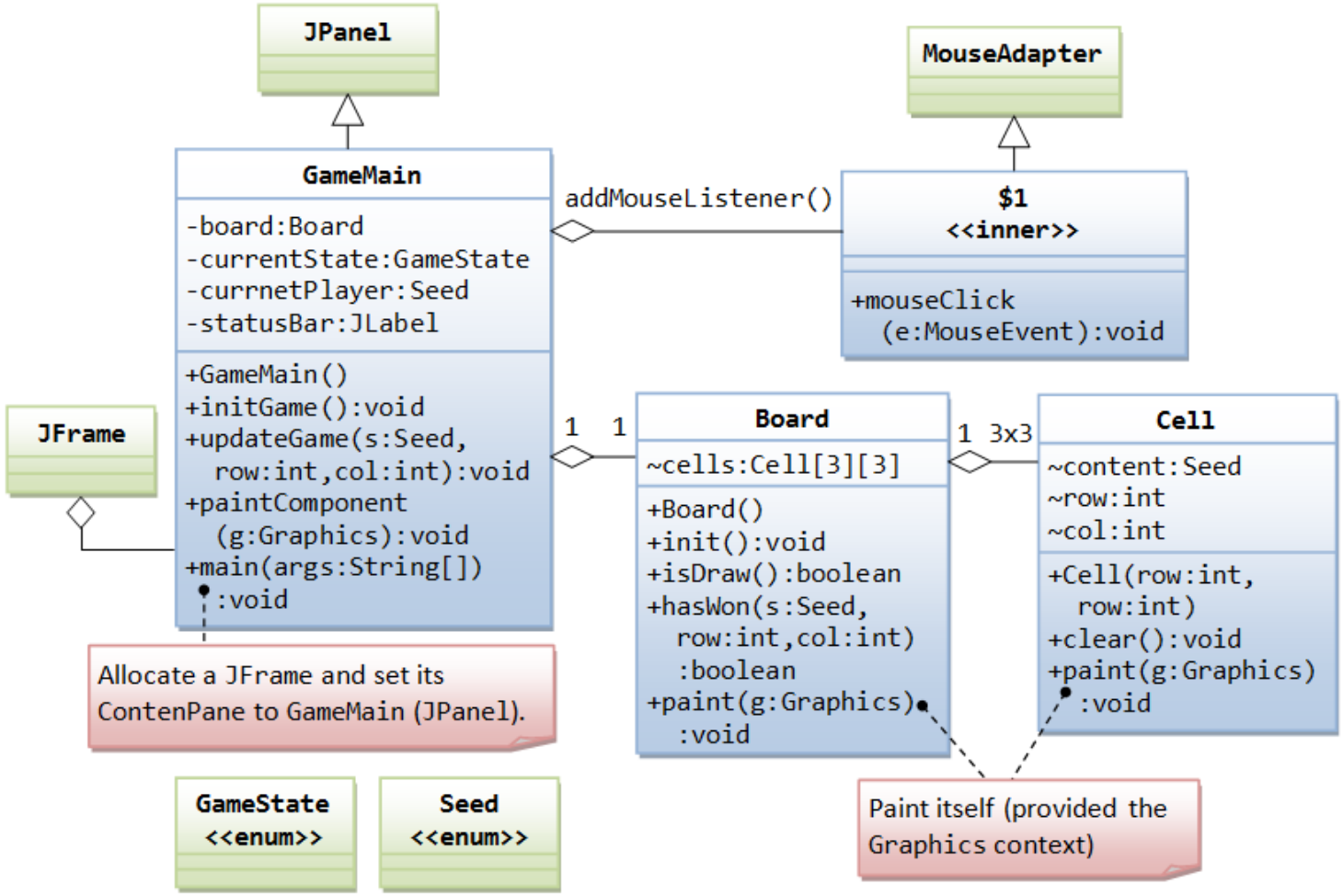
```
......

// In mouseClicked() event-handler - play sound effect upon mouse-click
if (currentState == GameState.PLAYING) {
   if (soundClipMove.isRunning()) soundClipMove.stop();
   soundClipMove.setFramePosition(0); // rewind to the beginning
   soundClipMove.start();             // Start playing
} else {
   if (soundClipGameOver.isRunning()) soundClipGameOver.stop();
   soundClipGameOver.setFramePosition(0); // rewind to the beginning
   soundClipGameOver.start();             // Start playing
}
```

# Challenge: transforming the code to OOP + Swing

- Using the code of the Swing graphics version of our Gato game, understand the UML class diagram and start transforming the code writing the proper classes in Java and make them run!

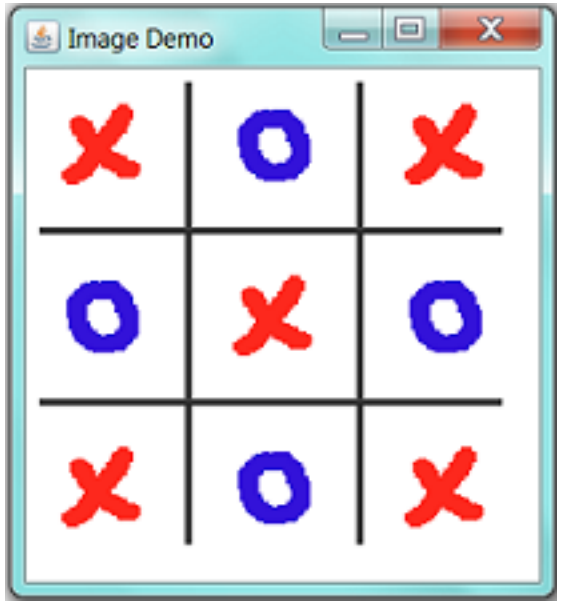
- **Check the examples in Webcursos**



# Adding images to the graphics in Swing

- Find out how to add an image to the X and O instead of drawing them.

Images:




- What are the required classes and libraries that you have to use?

- Check the complementary reading and examples in Webcursos

# Explore other alternatives to handle graphics in Java

- Find out what is **WindowBuilder** and how can you use it?
- Find out what is **JavaFX** and how can you use it?
- Which of these alternatives would you use for your course project?
- Next session after P1 we will continue exploring these alternatives with some examples.


Showtime!

UNIVERSIDAD ADOLFO IBÁÑEZ

# Show your project advance for this week

# Course Project

- Every week assignment. Small chunks -> Agile!
- We will have weekly advances as assignments for demonstrations in the class
- General Requirement:
  - Code an automated assistant to keep you updated with your To-Do list and give you alerts for every task and inform you about important news and calls you received during your busy time (performing a task).
- Specific requirements for the next week to continue coding with Java
  - Use console input/output (not a file yet, if so it's a plus)
  - Use conditions, loops, and void or value methods, arrays and recursion if needed and classes.
  - Design the class diagram of the program in UML as it is showed in the lesson, using the UML Designer in Eclipse (Menu->Help->Eclipse Marketplace->Search "UML Designer"-> Install **UML Lab**)
  - A good brief reference for UML here: http://edn.embarcadero.com/article/31863
  - Remember that the user should be able to add tasks to the To-Do list and the assistant has to show an alert to the user about the next event (task to perform) according to the time it was scheduled in the list.
  - Follow the instructions on how to write clear code published in Webcursos

# Hands-on

Coding Java in Eclipse

(Follow the instructions on how to write clear code published in Webcursos)

# Exercise 11 (Chapter 7)

- Resolve exercise 7.1,  7.3 and 7.4 of chapter 7 of the Java Book

# Exercise 12 (Chapter 8)

- Resolve exercise 8.2, 8.5 and 8.8 of chapter 8 of the Java Book

# Exercise 13 (Chapter 9)

- Resolve exercise 9.3 and 9.4 of chapter 9 of the Java Book

# Exercise 14 (Chapter 10)

- Resolve exercise 10.1, 10.2, 10.3 and 10.4 of chapter 10 of the Java Book

# Next Week…

# **Compulsory** Self study for next week

- **Compulsory** reading of **Chapter 9 and 10** of the Book “Think Java: How to Think Like a Computer Scientist” for preparing your self for the next lecture and the test of the week after.

- **P1 on April 20 2018 at 15.00**
  - Selection of alternatives
  - Java Programming with Eclipse
  - All contents until now in Webcursos:
    - All lessons
    - All complementary readings (not links to news or reference sites)
    - All test (1-3)
    - Chapters **1-10** of the Java guide book

# Homework (Optional)

- Practice and finish the exercises 11, 12, 13 and 14.
- **This will help you to master the concepts and be prepared for P1.**
- **Optional means** that if you didn't have time to complete the exercise in the week, anyways you will study them for preparing for P1.
- **Optional does not mean** the exercise's contents will not be part of P1.
- Students that missed the submission of one or more assignments ***must*** do this homework and upload them to GitHub.

# Keep your GitHub updated every week

- Upload all your lesson exercises, test solutions, homework exercises, optional exercises and weekly advances of the course project in separated folders to keep them ordered.

- Be sure your GitHub is updated every week before the class since we will check the files to review if you are working and self studying.

- **Keeping your GitHub updated is compulsory:**
  1) Otherwise you might be graded with 1.0
     when we check it weekly.
  2) Otherwise at the end of the course you will not have excuse
     if you need to recover a grade of a weekly test or homework exercise

# Course Project

- Next week there will not be a review of the weekly advance since we have P1
- **BUT we will have the first milestone (Delivery 1) on April 26**
- Be sure that:
  - your code is programmed using the Object Oriented paradigm
  - the UML diagram is well specified and represent the classes you have coded, otherwise you should indicate what is missing or under development
  - your code is working with the main functionalities that we specified in the beginning of the course and the weekly advances
  - your code compiles with no errors
  - your code runs with no issues
  - your code executes some use cases properly
  - The minimal requirement for this delivery is that your code runs using the console to show the execution
  - If your code runs using a graphics interface (Swing or JavaFX) you will get a bonus of 1 pt (not cumulative)
- Together with the code of your project uploaded and updated in **Github**, you will have to record a video using **Jing** to show a demo of the execution of the program and the use cases. You will have to provide the URL were your video will be stored by Jing to play it afterwards for the revision. The video must be consistent with the code we will review from your GitHub.

UNIVERSIDAD ADOLFO IBÁÑEZ

# Object Oriented Programming
# TICS-201 (Stgo)

Professor: Ricardo Seguel, Ph.D.

13-Apr-2018