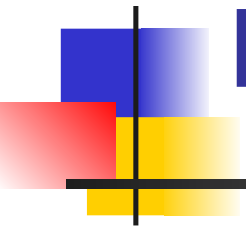
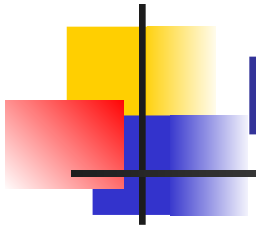


Práctica 3

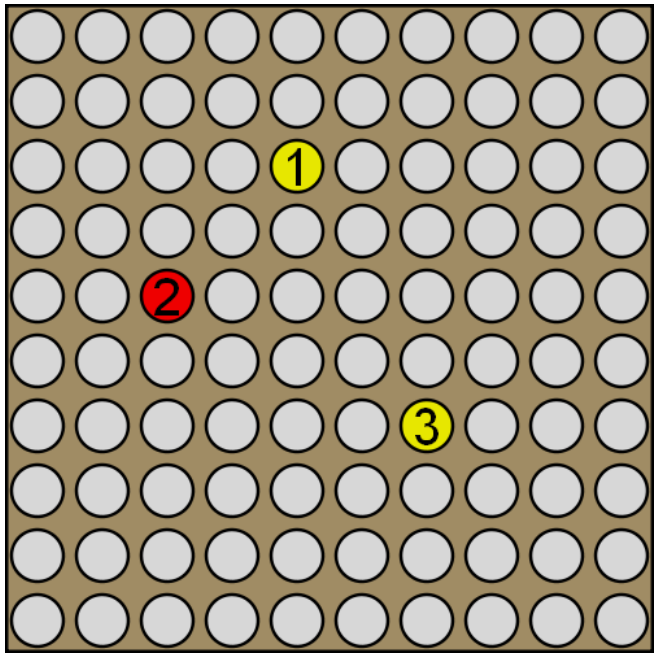
Puri Arenas
DSIC-UCM

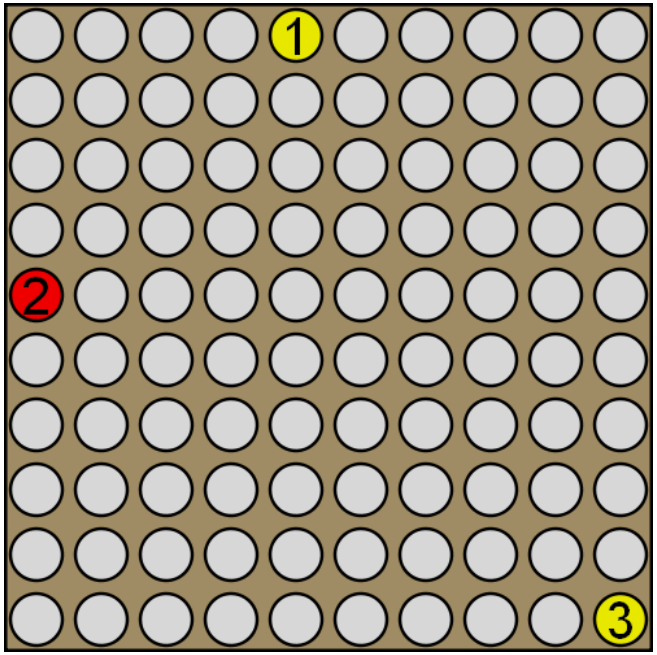


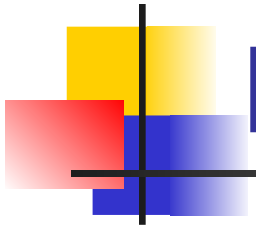
Nuevo Juego: Gravity

- El *tamaño del tablero no es fijo*, sino que se establece al iniciar la partida.
- Por defecto se juega en tableros de **`10x10`**, pero se pueden utilizar tamaños de tablero distintos (y no necesariamente cuadrados).
- Los jugadores *colocan la ficha en cualquier casilla vacía* del tablero.
- Una vez colocada, la ficha es atraída por los bordes de forma que ésta se desplaza en línea recta hacia el borde (o bordes) más cercanos.

Gravity







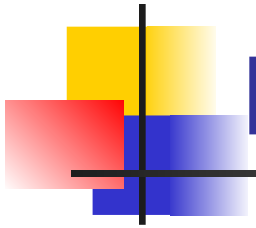
Nuevos comandos

- Nuevo comando que permite cambiar el tipo de jugador:

  `jugador <color> <tipo>`

  - donde **`color`** podrá ser **`blancas`** o negras y el tipo de jugador podrá ser **`humano`** o **`aleatorio`**.

- El jugador aleatorio elige una posición correcta: en el **Conecta 4** elegirá una columna que no esté llena, y en **Gravity** una posición que no esté ocupada.



Nuevos Comandos

- El comando **Jugar** se extiende añadiendo la posibilidad de jugar a **Gravity**. En **Gravity** el tamaño del tablero puede configurarse:

```
gr 9 10
```

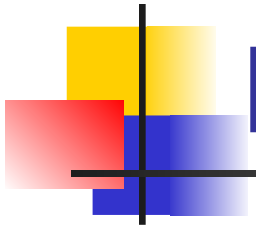
- **Reiniciar** un juego supone poner ambos jugadores humanos.

- Nuevo comando **Ayuda**, que muestra por pantalla todos los comandos disponibles y una breve explicación de lo que hace cada uno.

Parámetros de la Aplicación

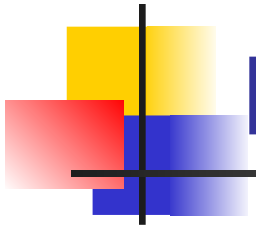
- **`-g`** o **`--game`** seguido del tipo de juego: **`c4`**, **`co`** y **`gr`**.
  - Por defecto, **`-g`** sin nada juega a **Conecta 4**.
  - En **Gravity**, se utilizará el tamaño del tablero por defecto de **`10x10`**, a no ser que se especifiquen los parámetros siguientes:
    - **`-x`** o **`--tamX`** seguido de un número.
    - **`-y`** o **`–tamY`** seguido de un número.

- **`-h`** o **`--help`**: solicita a la aplicación que muestre la ayuda sobre cómo utilizar los parámetros.



Exepciones

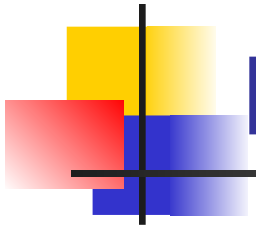
- En **Conecta 4** y **Complica**, si el usuario indica una columna que no existe se mostrará **“Columna incorrecta. Debe estar entre 1 y K.”** donde **K** será sustituido por el valor que corresponda.
- En **Conecta 4** si se intenta colocar en una columna llena se escribirá **“Columna llena.”**
- En **Gravity**, si se intenta colocar en una posición incorrecta (fuera de los límites del tablero), se pondrá **“Posición incorrecta.”**
- En **Gravity**, si se intenta colocar en una posición que ya tiene ficha, se especificará **“Casilla ocupada.”**



Factorías

- Interfaz que aglutina los métodos de construcción de los distintos objetos involucrados en un juego concreto.

```
public interface FactoriaJuego {
    public ReglasJuego creaReglas();
    public Movimiento creaMovimiento(int fila, int col, FICHA color);
    public Jugador creaJugadorAleatorio();
    public Jugador creaJugadorHumano(Scanner sc);
}
```



Factoría Conecta4

```
public class FactoriaJuegoConecta4 implements FactoriaJuego {

  public Jugador creaJugadorAleatorio() {
    return new JugadorAleatorioConecta4();
  }
  public Jugador creaJugadorHumano(Scanner sc) {
    return new JugadorHumanoConecta4(sc);
  }
  public ReglasJuego creaReglas() {
    return new Conecta4();
  }
  public Movimiento creaMovimiento(int f, int c, FICHA color) {
    return new MovConecta4(c, color);
  }
}
```

Clase Abstracta Jugador

- Devuelve el siguiente movimiento a efectuar por el jugador.

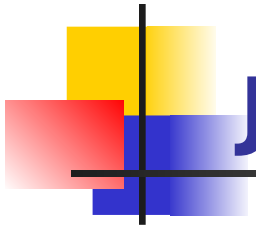
```
abstract public class Jugador {
  protected int fila; protected int columna;
  protected abstract void obtenFilaColumna(Tablero tab, FICHA color);
  public Movimiento getMovimiento(FactoriaJuego factoria,
       Tablero tab, FICHA color) throws DatosIncorrectos {
     try { this.obtenFilaColumna(tab,color);
          return factoria.creaMovimiento(this.fila,this.columna,color);
     }
     catch (InputMismatchException e){
       throw new DatosIncorrectos("Los datos introducidos no son
     numericos");
     }
  }
}
```

Jugador Humano Gravity

Heredan de Jugador las clases:

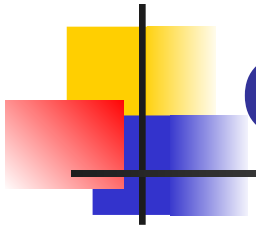
- **JugadorHumanoConecta4, JugadorAleatorioConecta4**
- **JugadorHumanoComplica, JugadorAleatorioComplica**
- **JugadorHumanoGravity, JugadorAleatorioGravity**

```
public class JugadorHumanoGravity extends Jugador {
   private Scanner sc;
   public JugadorHumanoGravity(Scanner sc) { this.sc = sc;}
   public void obtenFilaColumna(Tablero tab, FICHA color) {
     System.out.print("Introduce la fila: ");
     int fila = sc.nextInt();
     System.out.print("Introduce la columna: ");
     int col = sc.nextInt();
     sc.nextLine(); // Quitamos el INTRO
     this.fila = fila; this.columna=col;
  }
}
```



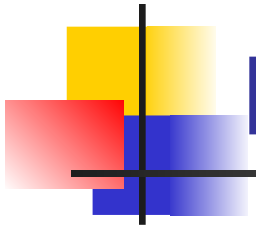
Jugador Aleatorio Conecta4

```
public class JugadorAleatorioConecta4 extends Jugador {

  @Override
  public void obtenFilaColumna(Tablero tab, FICHA color) {
   boolean fin=false;
   int columna=0;
   while (!fin) {
    columna = (int)(tab.getNumColumnas() * Math.random());
    if (tab.getFicha(tab.getNumFilas()-1,columna) == FICHA.VACIA)
         fin = true;
   }
   this.columna = columna;
  }
}
```



Clase Movimiento

```
public abstract class Movimiento {
  protected int columna;
  protected FICHA color;
  protected int fila;

  public Movimiento(int columna, FICHA color) {...}
  public FICHA getJugador() {...}
  public int getColumna(){...}
  public int getFila(){...}
  public abstract void ejecutaMovimiento(Tablero tab)
           throws MovimientoInvalido;
  public abstract void undo(Tablero tab);
}
```



Movimiento Conecta 4

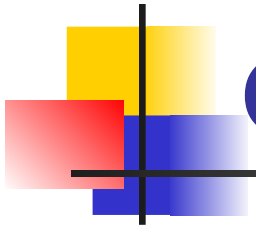
```
public class MovConecta4 extends Movimiento {

  public MovConecta4(int columna,FICHA ficha) { super(columna,ficha);}

public void ejecutaMovimiento(Tablero tab) throws MovimientoInvalido {
    if (!this.columnaCorrecta(tab))
              throw new ColumnaIncorrecta(tab.getNumColumnas());
    else if (!this.columnaLlena(tab))
              throw new ColumnaLlena(this.columna);
    else {
      int f = // busca la fila en el tablero;
      tab.ponFicha(f, this.columna, this.color);
      this.fila = f;
    }
  }
  public void undo(Tablero tab) {
   tab.ponFicha(this.fila,this.columna, FICHA.VACIA);
  }
}
```

Clase Partida

```
public class Partida {
    private ReglasJuego reglas;
    private Tablero tablero;
    private FICHA turno;
    private Stack<Movimiento> undoStack;
    // resto de atributos

    public Partida(ReglasJuego reglas){ reset(reglas); }
    public void reset(ReglasJuego reglas) {...}
    public void ejecutaMovimiento(Movimiento mv)
                throws MovimientoInvalido {...}
    public void undo() throws ErrorDeEjecucion {...}
    public Movimiento getMovimiento(FactoriaJuego factoria, Jugador jugador)
                throws DatosIncorrectos {...}
    // resto de métodos públicos
}
```

Clase Partida

```
public Movimiento getMovimiento(FactoriaJuego factoria, Jugador jugador)
                    throws DatosIncorrectos {
    try {
        return jugador.getMovimiento(factoria, this.tablero, this.turno);
    }
    catch (DatosIncorrectos e){
        throw e;
    }
}
```

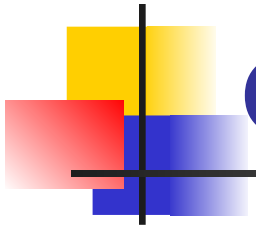


Clase Partida

```
public void ejecutaMovimiento(Movimiento mv)
                                    throws MovimientoInvalido {
     if (terminada)
            throw new MovimientoInvalido("Partida ya terminada.");
     FICHA color = mv.getJugador();
     if (color != this.getTurno())
            throw new MovimientoInvalido("Turno incorrecto.");
     mv.ejecutaMovimiento(this.tablero);
     undoStack.push(mv);
     // código asociado al control de ganador y cambio de turno
  }
}
```

Clase Controlador

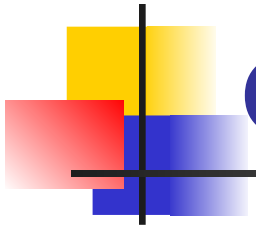
```
public class Controlador {

  private Partida partida;
  private Scanner in;
  private Jugador jugador1;
  private Jugador jugador2;
  private FactoriaJuego factoria;
  // pueden necesitarse más atributos

  public Controlador(FactoriaJuego f, Partida p, Scanner in) {...}
  // al resetear ambos jugadores pasan a ser humanos
  public void reset(FactoriaJuego factoria) {...}
  // leer comando y procesar comando hasta que acabe el juego
  // o se ejecute salir
  public void run(){...}
  public void undo() throws ErrorDeEjecucion {...}
  // resto de metodos públicos
}
```



Comandos

Los comandos ahora los implementamos utilizando una interfaz que deben implementar.

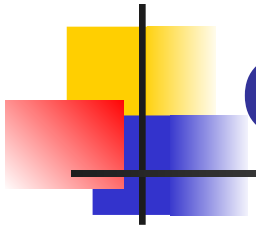
```
interface Comando {
    void execute(Controlador control) throws ErrorDeEjecucion;
    Comando parsea(String[] cadena);
    String textoAyuda();
}
```



Comandos

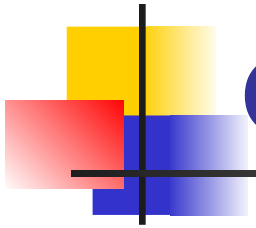
La interfaz **Comando** es implementada por las clases:

- **`Ayuda`**
- **`Deshacer`**
- **`Jugar`**
- **`Poner`**
- **`PonJugador`**
- **`Salir`**



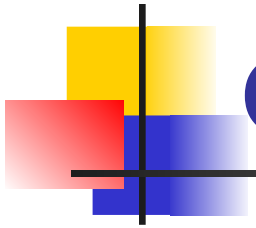
Comandos

```
public class Ayuda implements Comando {

 public void execute(Controlador c) throws ErrorDeEjecucion {
      System.out.println(ParserAyudaComandos.AyudaComandos());
 }
 public Comando parsea(String[] cadena) {
    if (cadena.length!=1) return null;
    else if (cadena[0].equalsIgnoreCase("AYUDA"))
         return new Ayuda();
    else return null;
 }

 public String textoAyuda() {
    return "AYUDA: Muestra la ayuda";
 }
}
```



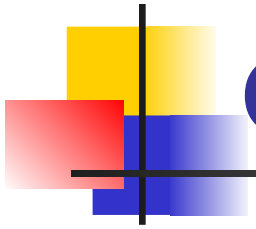
Comandos

```
public class Jugar implements Comando {
  private FactoriaJuego factoria;

  public Jugar(){}
  public Jugar(FactoriaJuego factoria) {
     this.factoria = factoria;
  }
  // resto de la implementación
}

 public class Poner implements Comando {

 // el método execute le pide al controlador que ejecute
 // el movimiento

 }
```



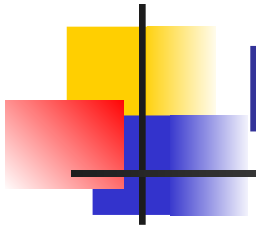
Comandos

```
public class PonJugador implements Comando {

  private FICHA color;
  private String tipoJugador;

  public PonJugador(FICHA color, String tipoJugador) {...}
  public PonJugador() {...}
  public void execute(Controlador control) {
    // manda al controlador que ponga el tipo de jugador y la ficha
  }
  // resto de código

}
```



Comandos

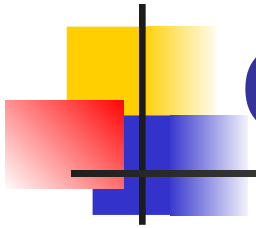
```
public class Salir implements Comando {


  public void execute(Controlador control) {
      // le indica al controlador que termine el juego
  }
  // resto de código
}
```



Parser Comandos

- Añade además una clase **ParserAyudaComandos** encargada de parsear un **`String`** y generar el comando asociado.
- Esta clase la usarás desde el comando **`Ayuda`** y desde el **`Controlador`**.

```
public class ParserAyudaComandos {

 private static Comando[] comandos = {
    new Salir(), new Deshacer(), new Jugar(),new Poner(),
    new PonJugador(), new Ayuda()
};
static public String AyudaComandos(){
    // devuelve el texto asociado al comando Ayuda
}
static public Comando parsea(String[] cadenas){
    // parsea el string y devuelve el comando asociado o null en caso
    // de error. Puedes usar tratamiento de excepciones en lugar de null
 }
}
```



Opcional

Utiliza la librería “org.apache.commons.cli.Options” para parsear los parámetros de la aplicación. Busca la librería y aprende a usarla.

```
Options cmdLineOptions;
cmdLineOptions = new Options();
cmdLineOptions.addOption(
       OptionBuilder.withArgName("game")
      .hasArg()
      .withDescription("Tipo de juego (c4, co, gr). Por defecto, c4.")
      .withLongOpt("game")
      .create('g')
    );
…
```