

Tercera práctica

Prueba de las aplicaciones desarrolladas

Tercera sesión: Ahora que todas las clases del juego están implementadas, ha llegado el momento de probar la aplicación de Napakalaki, tanto en Java como en Ruby. Para ello, es necesario jugar unas partidas a dicho juego. De este modo, todos los métodos desarrollados acabarán por ejecutarse y podréis comprobar que funcionan correctamente. Una interfaz de usuario es requerida para permitir que los jugadores tomen sus decisiones, es decir que jueguen sus cartas. Usando esta interfaz, podréis probar en profundidad el juego y detectar errores en tiempo de ejecución que deberéis depurar.

NOTAS:

- Para facilitar la tarea os proporcionamos dos interfaces textuales, una en Java y otra en Ruby, que deberéis usar como interfaz de usuario de vuestras aplicaciones Napakalaki.
- El código de las interfaces de usuario está disponible en swad.

En Java y Ruby realizar las siguientes tareas:

1. Ojear las interfaces de usuario proporcionadas.

Para comprender, a grandes rasgos, el funcionamiento de las interfaces podéis tratar de responderos las siguientes cuestiones:

- ¿Para qué sirve y cómo funciona el método *manageMenu*?
- ¿Qué ventajas proporciona usar el enumerado *Command*?
- ¿Cuál es el método que conecta, principalmente, la interfaz con el modelo (conjunto de clases que implementan la lógica de negocio de Napakalaki)?
- ¿Cuál es la instrucción que permite leer de teclado en Java? ¿En Ruby?
- ¿Cómo se “construye” la frase que debe aparecer en la interfaz de usuario con el número entero de la opción elegida por el usuario?
- ¿Qué pasaría si se introduce algo que no sea un entero desde teclado? ¿Cómo se resuelve en Java? ¿Cómo se resuelve en Ruby?

No obstante, entender la interfaz que os facilitamos, en java y ruby, no es requisito imprescindible para realizar correctamente esta práctica. Siguiendo una especificación común se pueden conectar módulos de software de modo que el resultado funcione sin problemas, sin tener que estudiar los entresijos del software que no hemos desarrollado.

2. Integrar las clases de la interfaz con vuestro modelo.

Aseguraos de que los nombres de paquetes en Java (o módulos en Ruby), clases y métodos usados en vuestro modelo coinciden exactamente con los utilizados en la interfaz proporcionada. Deberían coincidir si se han seguido fielmente los diagramas UML, pero si no es así deberéis realizar las correcciones oportunas en el modelo.

3. Ejecutar la aplicación.

Introducir por la interfaz de usuario textual los datos que se requieren para jugar a Napakalaki: equiparse, mostrar el monstruo, combatir, descartarse, comprar niveles, etc.

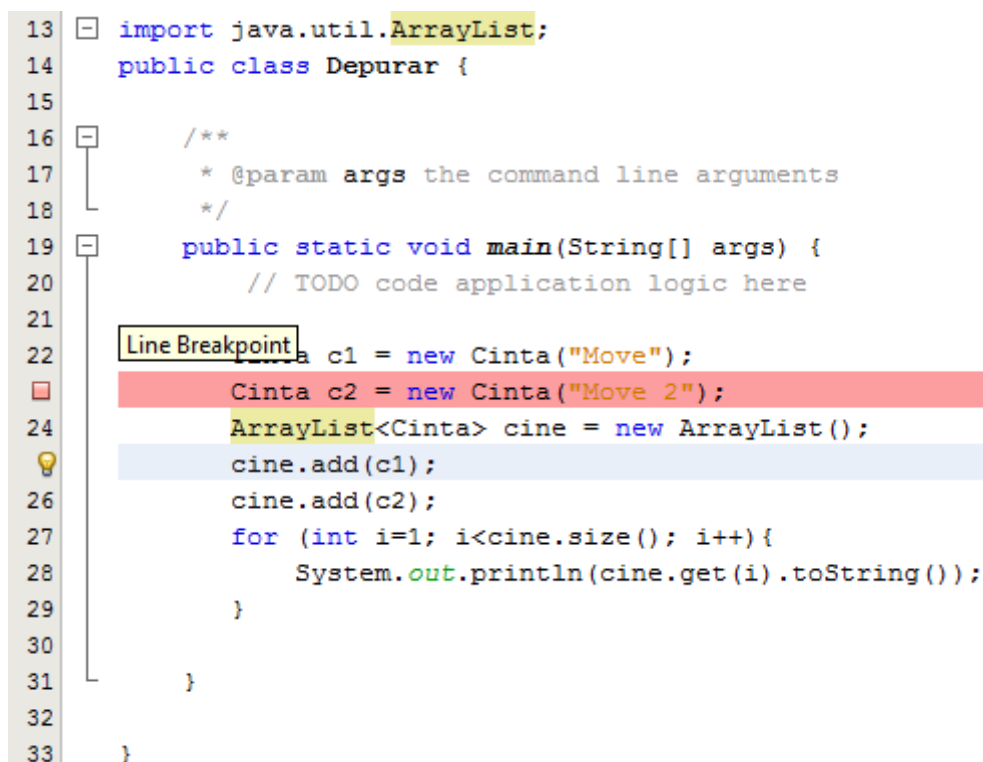
Tratad de ser sistemáticos para utilizar todas las opciones que posibilita la interfaz, así evitaréis que surjan errores inesperados más adelante.

4. Depurar el código.

Probablemente, aparecerán errores durante el paso 3. Algunos errores de programación pueden ser evidentes y fáciles de solucionar, pero es posible que haya otros que no sepáis de dónde vienen. Para rastrear estos últimos, os recomendamos que utilicéis el depurador.

Depurar un programa consiste en analizar el código en busca de errores de programación (*bugs*). Los entornos de desarrollo suelen proporcionar facilidades para realizar dicha tarea. En el caso de **Netbeans** el **procedimiento de depuración** es muy sencillo:

- ✓ Lo primero que debéis hacer es establecer un punto de control (*breakpoint*) en la sentencia del programa donde se desea que la ejecución se detenga para comenzar a depurar. Para ello, únicamente hay que hacer *clic* en el número de línea donde se encuentra dicha instrucción (*line breakpoint*). La línea, en el ejemplo la número 23 (figura 1), se resaltará en rosa.



```
13 import java.util.ArrayList;
14 public class Depurar {
15
16     /**
17      * @param args the command line arguments
18      */
19     public static void main(String[] args) {
20         // TODO code application logic here
21
22         Cinta c1 = new Cinta("Move");
23         Cinta c2 = new Cinta("Move 2");
24         ArrayList<Cinta> cine = new ArrayList();
25         cine.add(c1);
26         cine.add(c2);
27         for (int i=1; i<cine.size(); i++){
28             System.out.println(cine.get(i).toString());
29         }
30
31     }
32
33 }
```

The image shows a code editor with line numbers 13 to 33. Line 23, which contains the code `Cinta c2 = new Cinta("Move 2");`, is highlighted in pink. A yellow box labeled "Line Breakpoint" is positioned over the line number 23. A light blue highlight is visible on line 25.

Figura 1: Line Breakpoint.

- ✓ Después, pulsar **Control+F5** o la correspondiente opción del menú **Debug** para comenzar la depuración (figura 2).

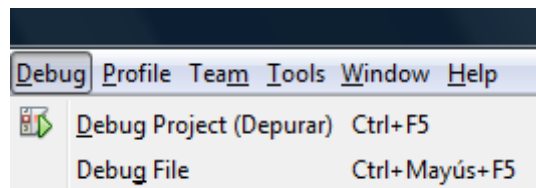


Figura 2: Menú *Debug*.

- ✓ Una vez que el flujo de control del programa llegue a un *breakpoint*, la ejecución se pausará para que podáis seguirla y la línea de código correspondiente se coloreará en verde. Además, aparecerá una barra de herramientas de depuración (arriba en la figura 3, después del *play*) y dos paneles de depuración (abajo en la figura 3): variables y *breakpoints*.

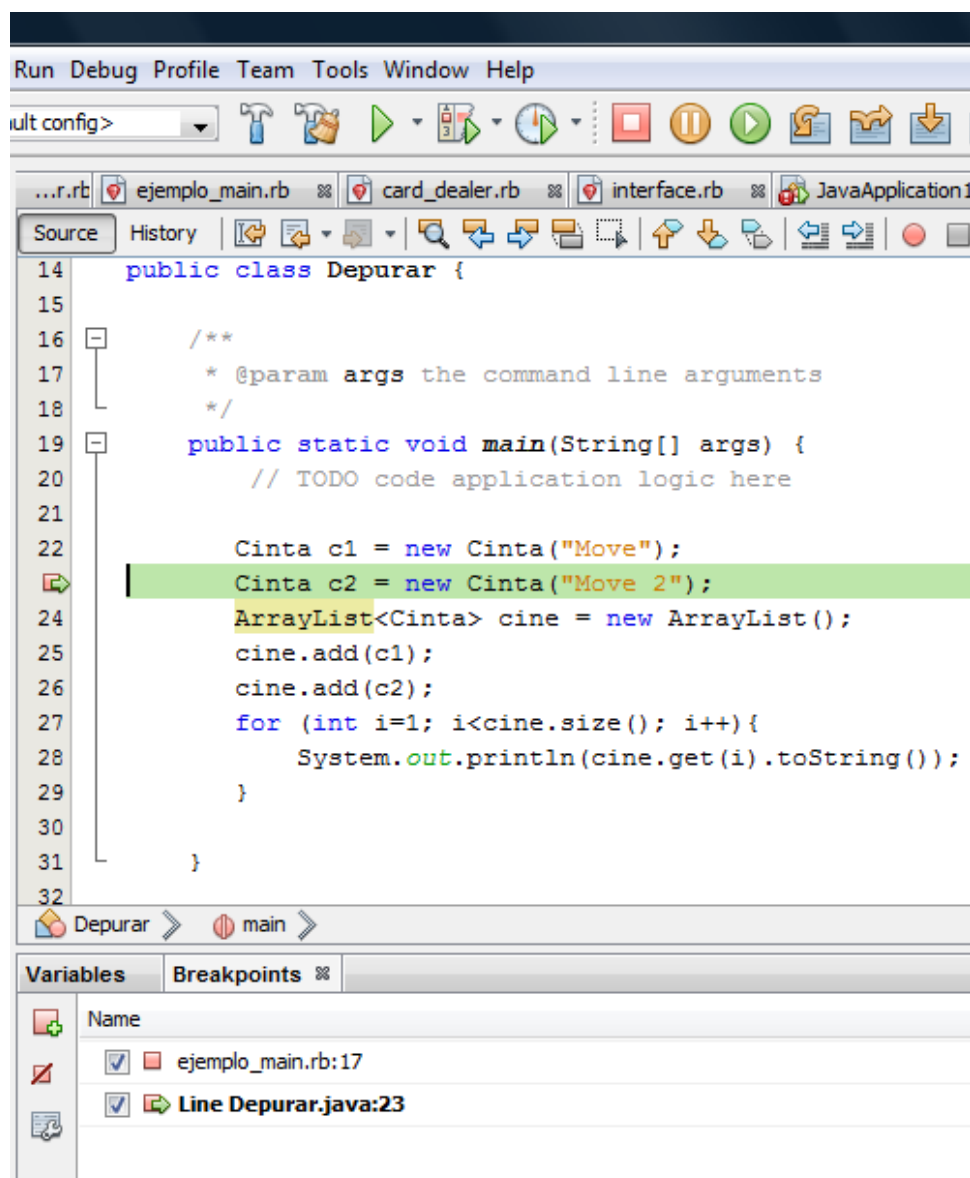


Figura 3: Herramientas de depuración.

- ✓ Ahora se puede trabajar de dos modos diferentes: pulsando **F7** o **F8**.



F7 Si se pulsa F7, el control entrará en el método invocado en la instrucción actual (en verde). En nuestro ejemplo se ejecutaría el constructor de la clase Cinta, parándose en la primera línea de dicho método.



F8 Si se pulsa F8, el control salta a la siguiente instrucción del programa. En nuestro ejemplo se ejecutaría el constructor (línea 23 del código) y se pararía en la línea 24. Usaréis, por tanto, esta opción cuando estéis seguros de que el *bug* no está ni se deriva del método invocado.

Si se han definido varios *breakpoints* en el fichero o proyecto, podéis usar la opción **F5** que continuará ejecutando instrucciones hasta el siguiente *breakpoint*, donde se pausará de nuevo la ejecución.



F5 Si se pulsa F5 continuará la ejecución hasta el siguiente punto de control. No tiene sentido en nuestro ejemplo, pues solo hemos definido un *breakpoint*.

- ✓ En cualquier momento podéis situar el cursor sobre una variable del programa y, si la variable está activa (en ámbito), obtendréis su valor. En el ejemplo (figura 4), la variable *i* tiene valor 1 en ese preciso instante de la ejecución.

```

    for (int i=1; i<cine.size(); i++) {
        System.out.println(cine.get(i).toString());
    }

```

Figura 4: Ver el valor de una variable.

- ✓ Adicionalmente, en el panel informativo de Variables (abajo en figura 3), podéis consultar el valor de cualquier variable activa en el contexto de ejecución actual. En el ejemplo (figura 5) es posible examinar que *i* tiene valor 1, y también el tipo de las variables, por ejemplo que *c1* es un objeto de la clase Cinta.

Variables		Breakpoints	
Name	Type	Value	
<input checked="" type="checkbox"/> m		>"m" is not a known variable in the current context.<	
<input checked="" type="checkbox"/> x		>"x" is not a known variable in the current context.<	
<Enter new watch>			
<input checked="" type="checkbox"/> Static			
<input checked="" type="checkbox"/> args	String[]	#74(length=0)	
<input checked="" type="checkbox"/> c1	Cinta	#72	
<input checked="" type="checkbox"/> c2	Cinta	#75	
<input checked="" type="checkbox"/> cine	ArrayList<Cinta>	"size = 2"	
<input checked="" type="checkbox"/> i	int	1	

Figura 5: Panel de Variables.

Si pulsamos el símbolo **+** que aparece junto a cada variable, aparecen más detalles sobre esta (figura 6). Por ejemplo, para el *ArrayList* **cine** podemos conocer su tamaño (2) y cada uno de sus elementos (dos objetos de la clase Cinta). Y para un objeto de la clase Cinta podemos inspeccionar sus atributos (en este caso, nombre).

cine	ArrayList<Cinta>	"size = 2"
[0]	Cinta	#72
nombre	String	"Move"
[1]	Cinta	#75
nombre	String	"Move 2"

Figura 6: Detalles sobre la variable cine.

- ✓ Si lo deseáis, situaros sobre una variable y pulsando *New Watch* en el menú contextual que se despliega, podéis definir un centinela (*watch*) que permitirá consultar el valor de una expresión evaluable en el contexto actual con dicha variable. Por ejemplo, en la figura 7, una operación aritmética definida sobre el valor de *i*, o la llamada a un método del objeto compuesto **cine**. Los *watches* aparecen en el panel de variables, tal y como se observa en la figura.

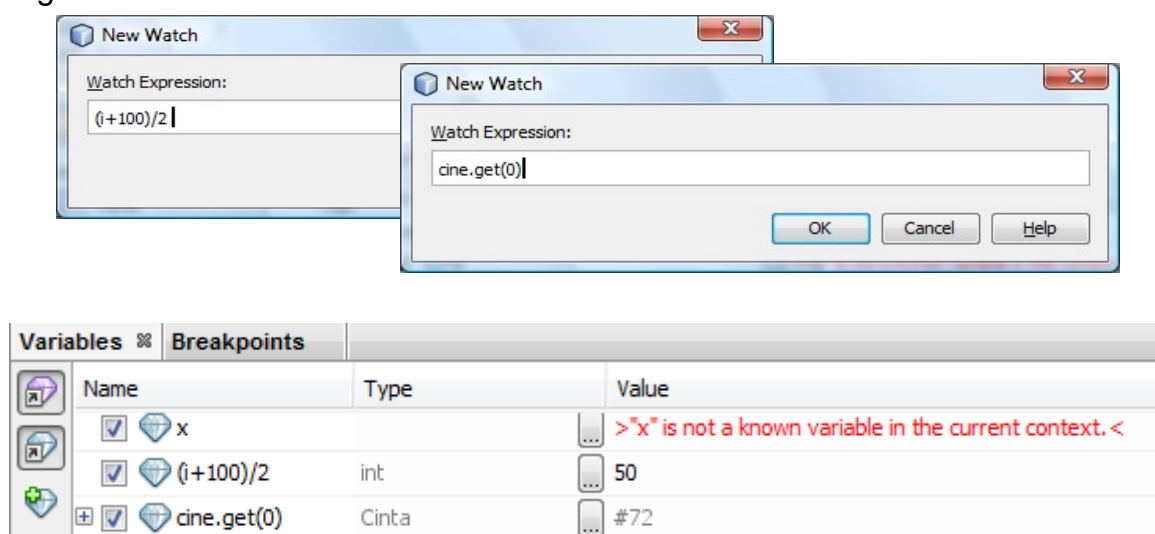


Figura 7: New Watch.

- ✓ Finalmente, para detener la depuración y continuar con la ejecución normal usaremos **Mayúscula+F5** o el botón correspondiente:

 **May+F5** Para finalizar

Recordad

Entrega del código: antes de que comience la sesión en la que se realiza el examen.

Examen: en la semana del 1 al 5 de diciembre.

Lugar: Aula en la que se desarrolla la correspondiente sesión de prácticas.

Día: El correspondiente a la sesión de cada uno de la semana indicada.

Duración: 20 minutos al comienzo de la sesión.

Tipo examen: Realizar pequeñas modificaciones sobre el código.

Requisitos:

- Disponer del código desarrollado para su modificación durante el examen.
- Disponer de los medios para su entrega en swad en el plazo de tiempo indicado.