

Laboratorio 0: Uso del Debugger en Eclipse

Introducción

Cuando hay errores en las aplicaciones hay que depurarlos o eliminarlos, para lo cual se requiere detectar dónde y por qué ocurren, antes de corregir la implementación que evite que ocurran de nuevo. Existen herramientas imprescindibles para los desarrolladores, que ayudan en la tarea de detección de errores en el software: son los programas depuradores o debuggers. Un programa depurador sirve para ejecutar paso a paso el código fuente con el fin de buscar el punto exacto donde se ha cometido un error de programación (también conocido como bug).

En este laboratorio utilizaremos el código inicial del proyecto Bets. No vamos a intentar arreglar un error en concreto, sino que veremos cómo utilizar el debugger: definición de puntos de parada, ejecución paso a paso y visualización de los valores de las variables.

Objetivos

El objetivo de este laboratorio es saber utilizar el debugger incluido en la herramienta de desarrollo Eclipse.

Pasos a seguir para utilizar el debugger

1. Definición de puntos de parada (breakpoints)

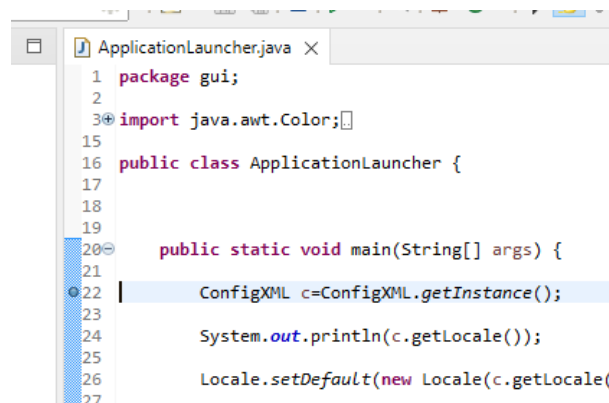
En el programa se pueden definir unos puntos de parada, para que, a partir de ellos, se pueda realizar esa ejecución paso a paso. Para definir un punto de parada (breakpoint) en el código:

Posicionarse en una línea concreta del código fuente (sobre el número de línea)

=> Hacer click derecho => Toggle breakpoint

[=> O si no, tras posicionarse en la línea => Hacer doble Click izquierdo]

Definir el siguiente breakpoint en la línea 22 de la clase gui.ApplicationLauncher:





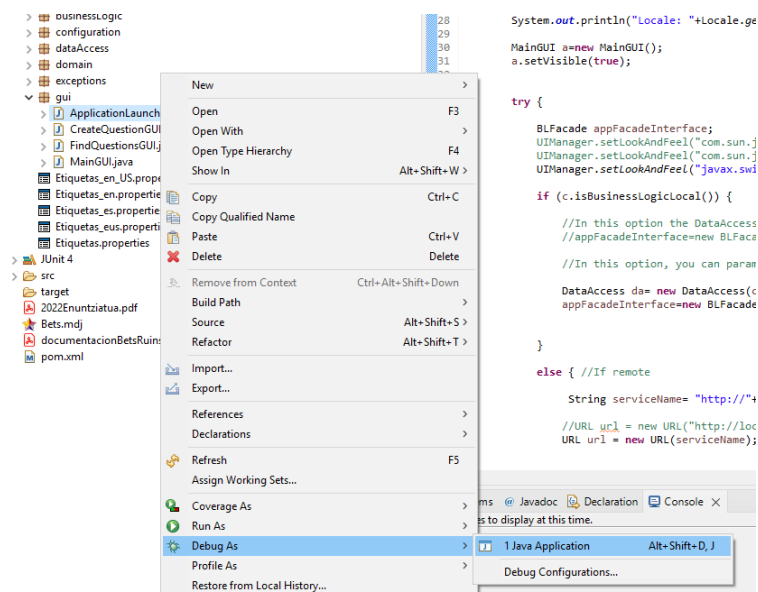
Y otro breakpoint en la línea 90 de la clase businessLogic.BLFacadeImplementation:

```
ApplicationLauncher.java | BLFacadeImplementation.java X
61  * @throws QuestionAlreadyExist if the same question a
62  */
63  @WebMethod
64  public Question createQuestion(Event event, String ques
65
66  //The minimum bed must be greater than 0
67  dbManager.open(false);
68  Question qry=null;
69
70
71  if(new Date().compareTo(event.getEventDate())>0)
72  throw new EventFinished(ResourceBundle.getBund
73
74
75  qry=dbManager.createQuestion(event,question,betMi
76
77  dbManager.close();
78
79  return qry;
80
81
82
83  /**
84  * This method invokes the data access to retrieve the
85  * @param date in which events are retrieved
86  * @return collection of events
87  */
88  @WebMethod
89  public Vector<Event> getEvents(Date date) {
90  dbManager.open(false);
91  Vector<Event> events=dbManager.getEvents(date);
92  dbManager.close();
93  return events;
94
95
--
```

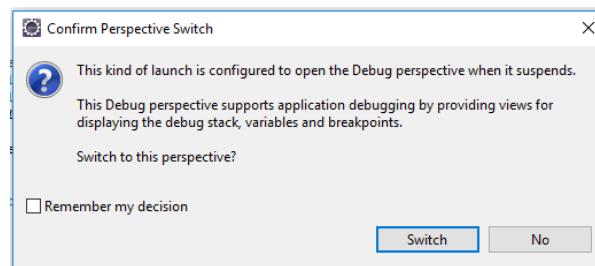
2. Ejecutar la aplicación en modo debugger

Ejecutaremos la aplicación en modo debugger, la cual se ejecutará normalmente hasta que se alcance alguno de los breakpoints definidos. Como la clase principal es ApplicationLauncher, esa es la que se ejecutará en modo debugger:

Seleccionar la clase ApplicationLauncher => Click dcho. => Debug As => Java Application

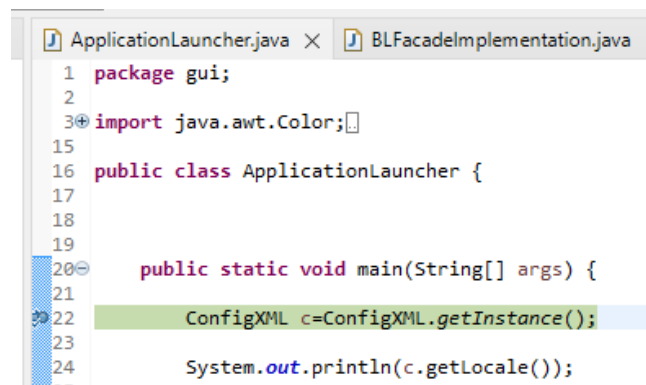


Cuando vaya a ejecutarse la instrucción de un breakpoint entonces Eclipse solicitará cambiar la perspectiva actual a la de Debug => Pulsar Switch



3. Ejecutar las instrucciones del código fuente paso a paso

En este punto, se puede ver en la perspectiva de Debug que la ejecución se ha detenido en el primer breakpoint encontrado:



Y también se puede comenzar a controlar la ejecución paso a paso con las siguientes opciones:



Haciendo click en la opción "Step Over" o pulsando F6, indicaremos que se ejecute la siguiente instrucción de una vez, esto es, sin entrar dentro de la llamada al método `getInstance()`

`ConfigXML c=ConfigXML.getInstance();`



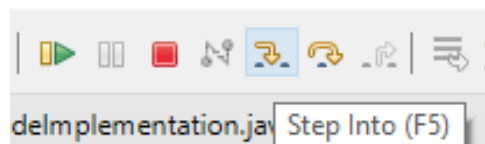
A continuación puede verse que el debugger se ha vuelto a detener en la siguiente instrucción: `System.out.println(c.getLocale());`;

```
18
19
20 public static void main(String[] args) {
21
22     ConfigXML c=ConfigXML.getInstance();
23
24     System.out.println(c.getLocale());
25
26     Locale.setDefault(new Locale(c.getLocale()));
27
28     System.out.println("Locale: "+Locale.getDefault());
29
30     MainGUI a=new MainGUI();
31     a.setVisible(true);
32 }
```

Después de pulsar 3 veces la opción “Step Over (F6)” el debugger se detendrá en la siguiente instrucción: `MainGUI a=new MainGUI();`;

```
18
19
20 public static void main(String[] args) {
21
22     ConfigXML c=ConfigXML.getInstance();
23
24     System.out.println(c.getLocale());
25
26     Locale.setDefault(new Locale(c.getLocale()));
27
28     System.out.println("Locale: "+Locale.getDefault());
29
30     MainGUI a=new MainGUI();
31     a.setVisible(true);
32
33
34     try {
```

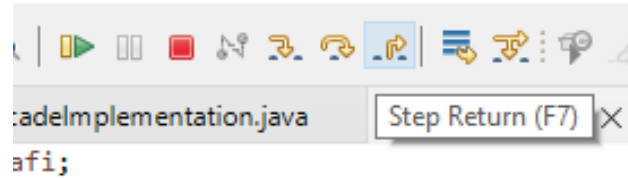
Haciendo click ahora en la opción “Step Into” o pulsando F5



se indica al debugger que se ejecute la siguiente instrucción entrando dentro de la llamada al método constructor `MainGUI()`, tal y como se muestra en la siguiente figura:

```
50 /**
51  * This is the default constructor
52  */
53 public MainGUI() {
54     super();
55
56     addWindowListener(new WindowAdapter() {
57         @Override
58         public void windowClosing(WindowEvent e) {
59             try {
60                 //if (ConfigXML.getInstance().isBusinessLogicLocal
61             } catch (Exception e1) {
62                 // TODO Auto-generated catch block
63                 System.out.println("Error: "+e1.toString()+" , prc
64             }
65             System.exit(1);
66         }
67     });
68
69     initialize();
70     //this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
71 }
72
73 }
```

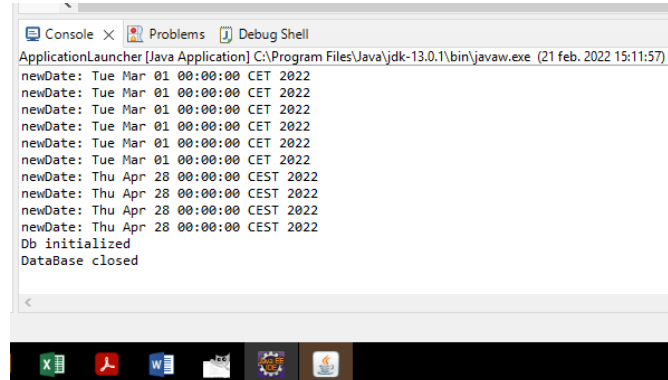
Podríamos continuar con el debugger dentro de este método, o podríamos regresar al punto desde donde se llamó al método para continuar desde ese punto anterior. Para ello tenemos la opción “Step Return (F7)”:



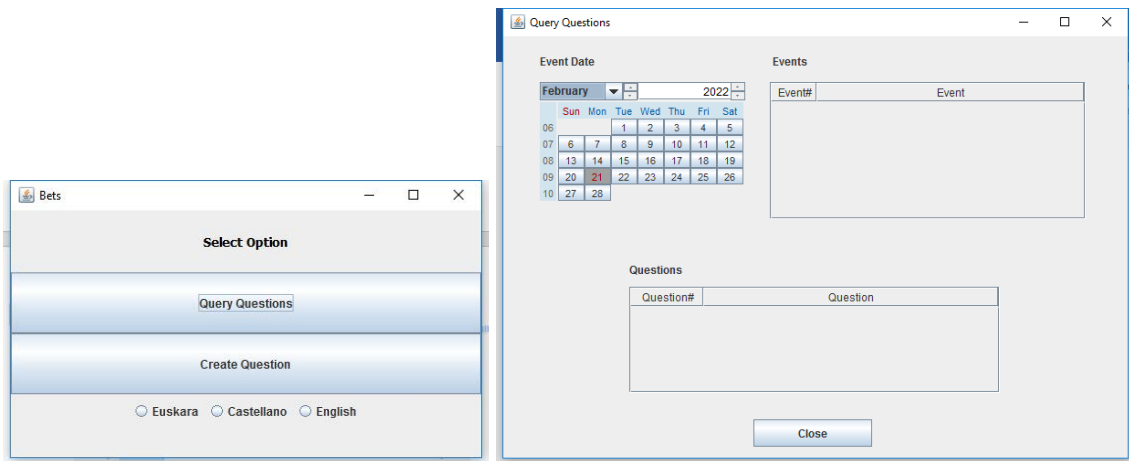
También podríamos dejar que la aplicación continúe su ejecución hasta el siguiente breakpoint. Eso se hace con la opción “Resume (F8)”



Tras ejecutar “Resume (F8)”, veremos que la aplicación sigue su ejecución, y ha creado la ventana principal, que se debe activar (pulsando el icono de Java en la barra de herramientas):



Para llegar al siguiente breakpoint (definido en el método `getEvents` de la lógica del negocio) deberemos provocar su ejecución.
Para ello: pulsar botón “Query Questions” => Seleccionar fecha en el calendario



Ahora el debugger muestra la siguiente instrucción a ejecutar: `dbManager.open(false);`

```

70
71
72     if(new Date().compareTo(event.getEventDate())>0)
73         throw new EventFinished(ResourceBundle.getBundle("Etiquetas").getString(
74
75     qry=dbManager.createQuery(event,question,betMinimum);
76
77     dbManager.close();
78
79     return qry;
80
81
82
83     /**
84     * This method invokes the data access to retrieve the events of a given date
85     * @param date in which events are retrieved
86     * @return collection of events
87     */
88     @WebMethod
89     public Vector<Event> getEvents(Date date) {
90         dbManager.open(false);
91         Vector<Event> events=dbManager.getEvents(date);
92         dbManager.close();
93         return events;
94     }
95

```

Pulsando F6 => F5, se entrará dentro de la llamada al método `getEvents(date)` de `DataAccess`:

```

175     db.getTransaction().begin();
176     Question q = ev.addQuestion(question, betMinimum);
177     //db.persist(q);
178     db.persist(ev); // db.persist(q) not required when CascadeType.PERSIST is added in questions property
179     // @OneToMany(fetch=FetchType.EAGER, cascade=CascadeType.PERSIST)
180     db.getTransaction().commit();
181     return q;
182
183
184
185     /**
186     * This method retrieves from the database the events of a given date
187     * @param date in which events are retrieved
188     * @return collection of events
189     */
190
191     public Vector<Event> getEvents(Date date) {
192         System.out.println(">>> DataAccess: getEvents");
193         Vector<Event> res = new Vector<Event>();
194         TypedQuery<Event> query = db.createQuery("SELECT ev FROM Event ev WHERE ev.eventDate=?1",Event.class);
195         query.setParameter(1, date);
196         List<Event> events = query.getResultList();
197         for (Event ev:events){
198             System.out.println(ev.toString());
199             res.add(ev);
200         }
201         return res;
202

```

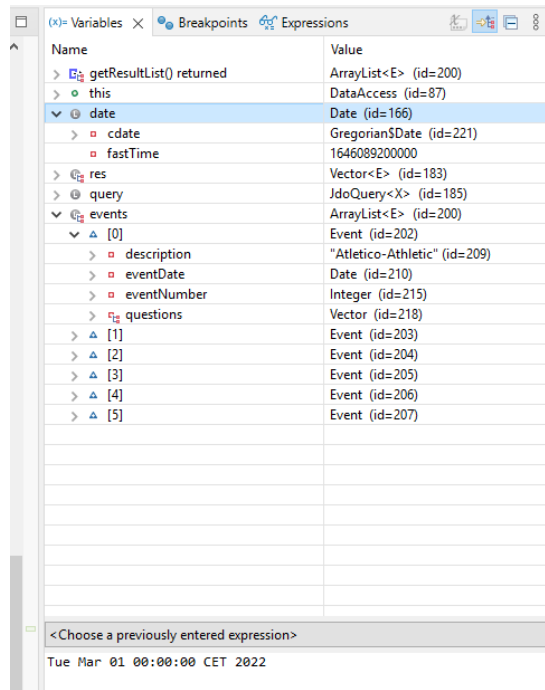
Si se pulsa 6 veces F6 hasta llegar a la instrucción `query.getResultList();`

```

198
199
200     public Vector<Event> getEvents(Date date) {
201         System.out.println(">>> DataAccess: getEvents");
202         Vector<Event> res = new Vector<Event>();
203         TypedQuery<Event> query = db.createQuery("SELECT ev FROM Event ev WHERE ev.eventDate=?1",Event.class);
204         query.setParameter(1, date);
205         List<Event> events = query.getResultList();
206         for (Event ev:events){
207             System.out.println(ev.toString());
208             res.add(ev);
209         }
210         return res;
211

```

En este punto puede verse el contenido de las variables (events, query, date,...), que podría ser el siguiente, si se hubiera seleccionado una fecha con eventos (1/03/2022, en este caso).



Si no aparece la pestaña con las variables: Window => Show View => Variables

Si no hay eventos en la fecha actual, se puede hacer "Resume (F8)" en el debugger y escoger una fecha con eventos en el calendario.

Por último, para terminar la ejecución del debugger está la opción "Terminate (Ctrl + F2)"

