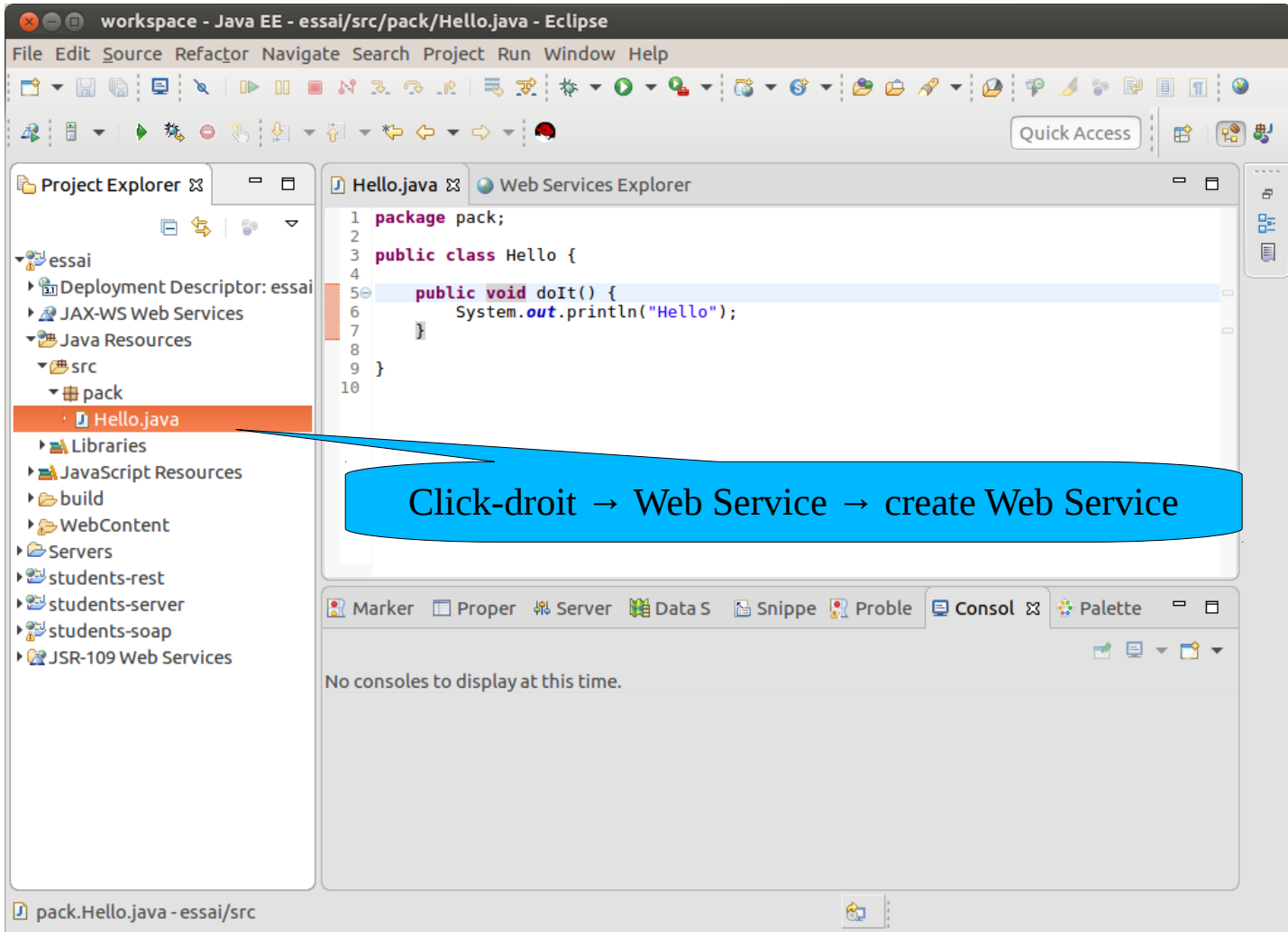


# Création d'un WS SOAP



The screenshot shows the Eclipse IDE interface. The Project Explorer on the left shows a project named 'essai' with a package 'pack' containing a file 'Hello.java'. The main editor displays the code for 'Hello.java':

```
1 package pack;
2
3 public class Hello {
4
5     public void doIt() {
6         System.out.println("Hello");
7     }
8 }
9
10
```

A blue callout bubble points to the 'Hello.java' file in the Project Explorer with the text: "Click-droit → Web Service → create Web Service".

The bottom of the IDE shows a toolbar with icons for Marker, Proper, Server, Data S, Snippe, Proble, Consol, and Palette. The console area below is empty, displaying "No consoles to display at this time."

# Test d'un WS SOAP

The screenshot shows the Eclipse IDE interface. The Project Explorer on the left displays a project structure with a file named `Hello.wsdl` circled in red. A blue callout bubble points to this file with the text: **Click-droit → Web Service → Test with Web Services Explorer**. The main editor shows the `Hello.java` file with the following code:

```
1 package pack;
2
3 public class Hello {
4
5     public void doIt() {
6         System.out.println("Hello");
7     }
8 }
9
10
```

The console at the bottom displays the following logs:

```
Tomcat v8.0 Server at localhost [Apache Tomcat] /home/hagimont/install/jdk1.8.0_45/bin/java (26 nov. 2018 9:22:59 PM)
INFO: At least one JAR was scanned for TLDs yet contained no TLDs. Enable debug logging for more details
nov. 26, 2018 9:22:59 PM org.apache.axis.utils.JavaUtils isAttachmentSupported
AVERTISSEMENT: Unable to find required classes (javax.activation.DataHandler and javax.activation.DataSource)
nov. 26, 2018 9:22:59 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["http-nio-8080"]
nov. 26, 2018 9:22:59 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-nio-8009"]
nov. 26, 2018 9:22:59 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 2521 ms
```

# Test d'un WS SOAP

The screenshot displays the Eclipse IDE interface. The top menu bar includes File, Edit, Navigate, Search, Project, Run, Window, and Help. The toolbar contains various icons for file operations and development tools. The Project Explorer on the left shows a project named 'essai' with a 'wsdl' folder containing 'Hello.wsdl'. The Web Services Explorer in the center shows a tree view of the WSDL file, with 'dolt' selected under the 'Operations' section. The Actions panel on the right displays 'WSDL Binding Details' and 'Operations' with a table:

Name	Documentation
<a href="#">dolt</a>	

The 'dolt' link is circled in red. Below the Actions panel is the Status panel. The Console at the bottom shows the following output:

```
Tomcat v8.0 Server at localhost [Apache Tomcat] /home/hagimont/install/jdk1.8.0_45/bin/java (26 nov. 2018
nov. 26, 2018 9:22:59 PM org.apache.coyote.AbstractProtocol start
INFO: Starting ProtocolHandler ["ajp-nio-8009"]
nov. 26, 2018 9:22:59 PM org.apache.catalina.startup.Catalina start
INFO: Server startup in 2521 ms
Hello
```

The address bar at the bottom shows the URL: <http://127.0.0.1:34293/wse/wsd/actions/SelectWSDLNavigatorNodeActionJSP.jsp?nodeId=11>

# WS REST avec Resteasy

```
@Path("/rest")
public class Facade {

    // bad
    static Hashtable<String, Person> ht = new Hashtable<String, Person>();

    @POST
    @Path("/addperson")
    @Consumes({ "application/json" })
    public Response addPerson(Person p) {
        ht.put(p.getId(), p);
        return Response.status(201).entity("person added").build();
    }

    @GET
    @Path("/getperson")
    @Produces({ "application/json" })
    public Person getPerson(@QueryParam("id") String id) {
        return ht.get(id);
    }

    @GET
    @Path("/listpersons")
    @Produces({ "application/json" })
    public Collection<Person> listPersons() {
        return ht.values();
    }
}
```

Reçoit un Json

Retourne 201 + une String

Renvoie un Json

Reçoit une String (id)

Person est un simple POJO

# Client REST avec Resteasy

Interface du service pour générer un stub

```
@Path("/rest")
public interface FacadeInterface {

    @POST
    @Path("/addperson")
    @Consumes({ "application/json" })
    public Response addPerson(Person p);

    @GET
    @Path("/getperson")
    @Produces({ "application/json" })
    public Person getPerson(@QueryParam("id") String id);

    @GET
    @Path("/listpersons")
    @Produces({ "application/json" })
    public Collection<Person> listPersons();

}
```

# Client REST avec Resteasy

URL du service

```
public class Client {  
    public static void main(String args[]) {  
        final String path = "http://localhost:8080/rs-server-person/rest";  
  
        ResteasyClient client = new ResteasyClientBuilder().build();  
        ResteasyWebTarget target = client.target(UriBuilder.fromPath(path));  
        FacadeInterface proxy = target.proxy(FacadeInterface.class);  
  
        Response resp;  
        resp = proxy.addPerson(new Person("007", "James Bond"));  
        System.out.println("HTTP code: " + resp.getStatus()  
            + " message: "+resp.readEntity(String.class));  
  
        resp.close();  
  
        Person p = proxy.getPerson("006");  
        System.out.println("get Person: "+p.getId()+"/"+p.getName());  
    }  
}
```

stub