From Java programming to JEE

christophe.cerin@univ-paris13.fr

(adapted from a Lecture of Heithem Abbess)

Back to Java programming

Class, inheritance, interface

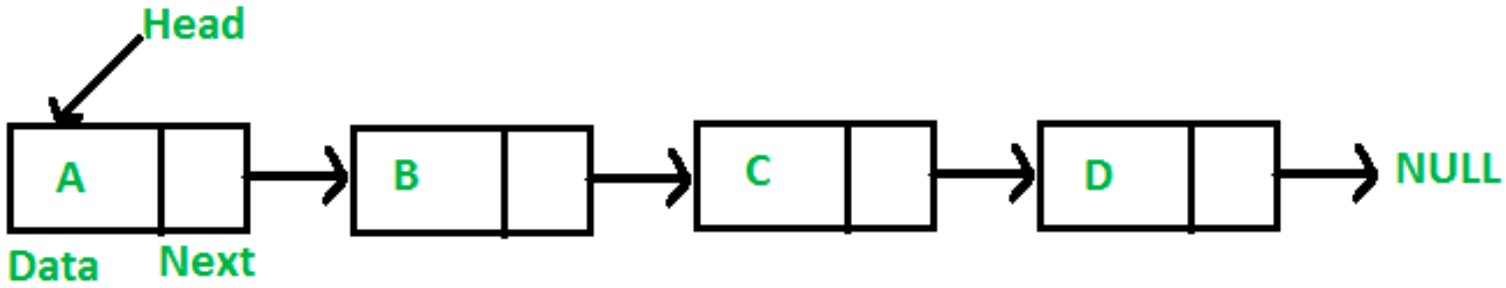
- Software objects are often used to model the real-world objects that you find in everyday life.
- A class is prototype from which objects are created. A class models the state (internal variables, constants) and behavior (methods) of a real-world object.
- Classes inherit state and behavior from their superclasses, and we may derive one class from another using the simple syntax provided by the Java programming language.

Class, inheritance, interface

- An interface is a contract between a class and the outside world. When a class implements an interface, it promises to provide the behavior published by that interface.
- An interface is a specification, not an implementation. A class is an implementation. An interface defines the name, the parameter types, the type of the result of a method. That's all folks!

Some advantages for OOP

- Bundling code into individual software objects provides a number of benefits, including:
  - **Modularity:** The source code for an object can be written and maintained independently of the source code for other objects. Once created, an object can be easily passed around inside the system.
  - **Information-hiding:** By interacting only with an object's methods, the details of its internal implementation remain hidden from the outside world.
  - **Code re-use:** If an object already exists (perhaps written by another software developer), you can use that object in your program. This allows specialists to implement/test/debug complex, task-specific objects, which you can then trust to run in your own code.
  - **Pluggability and debugging ease:** If a particular object turns out to be problematic, you can simply remove it from your application and plug in a different object as its replacement. This is analogous to fixing mechanical problems in the real world. If a bolt breaks, you replace *it*, not the entire machine.



Cell A='Monday + 11°C + 6mm'

Cell B='Tuesday + 10°C + 5mm'

etc, etc

```
private String lieu;// Town
private String numSemaine;// number of the week
```

} Weather station object

6 Weather Station in Java Swing

We want to implement a weather station that stores, for a town and a week, the day, the temperature and the precipitation

ValMeteoJour class (implements one cell)


```
public class ValMeteoJour {
     private String jour;
     private float temp;
     private float prec;
     public ValMeteoJour(){

     }
     public ValMeteoJour(String jour, float temp, float prec) {
        this.jour = jour;
        this.temp = temp;
        this.prec = prec;
     }
     public String getJour() {
        return jour;
     }
     public void setJour(String jour) {
        this.jour = jour;
     }
…
     @Override
     public String toString() {
        return "[jour=" + jour + ", prec=" + prec + ", temp=" + temp + "]\n";
     }
}
```

First Implementation


```
public class EnsValMeteo extends java.util.ArrayList < ValMeteoJour > {

        private String lieu;
        private String numSemaine;

        public EnsValMeteo() {

        }
…
…
…
        public static void main(String[] args) {
                EnsValMeteo evmm = new EnsValMeteo();
                evmm.charge("testEnsValMeteo.txt");
                evmm.afficheValeurs();
                EnsValMeteo evm = new EnsValMeteo();
                evm.LireClavier();
                evm.afficheValeurs();
        }

}
```

Second implementation


```
public class EnsValMeteoYanis extends JFrame{

  private String lieu;
  private String numSemaine;
  private ArrayList<ValMeteoJour> MyList;

  public EnsValMeteoYanis(){
    super("Station Meteo");
    this.lieu = "";
    this.numSemaine = "";
    this.MyList = new ArrayList<ValMeteoJour>();
    …
  }
…
…
  public static void main(String[] args){

    EnsValMeteoYanis evmm = new EnsValMeteoYanis();
    evmm.charge("testEnsValMeteo.txt");
    evmm.afficheValeur();
    evmm.lireClavier();
    evmm.afficheValeur();

  }
}
```

What are the differences and the consequences?

- The access of the internal states may be different;
- i.e sharing of information may change (visibility?);

```
public void afficheValeur(){
    String chaine = "";
    chaine += this.lieu + "\n";
    chaine += this.numSemaine;
    // Iterator
    for (ValMeteoJour val : this.MyList){
      chaine += "\n" + val.toString();
    }
    System.out.println(chaine);
  }
```

```
public void afficheValeur() {
    System.out.println("Lieu : " + this.getLieu() + " ; Semaine : " + this.getSemaine());
    // Iterator
    for (ValMeteoJour n: this)
         System.out.println(n.getJour() + "\t" + n.getTemp() + "\t" + n.getPrec());
}
```

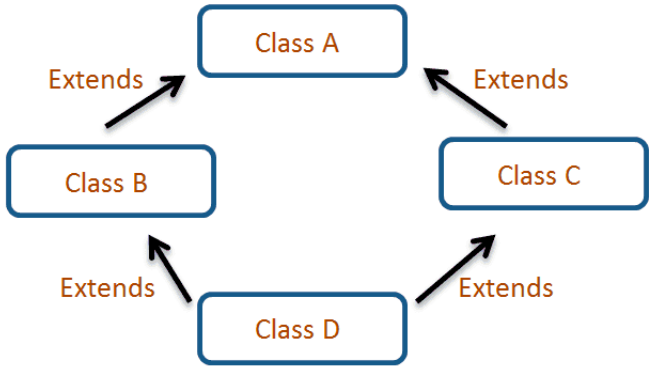
What are the differences and the consequences?


```
while (line != null) {
       StringTokenizer st = new StringTokenizer(line, "\t");
       ValMeteoJour aVal = new ValMeteoJour();
       aVal.setJour(st.nextToken());
       aVal.setTemp(Float.parseFloat(st.nextToken()));
       aVal.setPrec(Float.parseFloat(st.nextToken()));
       this.add(aVal);
       line = br.readLine();
}
```

```
while(line != null){
      StringTokenizer st = new StringTokenizer(line, "\t");
      ValMeteoJour aVal = new ValMeteoJour();
      aVal.setJour(st.nextToken());
      aVal.setTemp(Float.parseFloat(st.nextToken()));
      aVal.setPrec(Float.parseFloat(st.nextToken()));
      this.MyList.add(aVal);
      line = br.readLine();
}
```

Back to inheritance in Java

- In the Java programming language, each class is allowed to have **one** direct superclass, and each superclass has the potential for an unlimited number of *subclasses*.
- The diamond problem: the compiler is confused as to which method to use.
- When we instantiate an object of class D, any calls to method definitions in class A will be ambiguous – because it's not sure whether to call the version of the method derived from class B or class C.
- Java has no diamond problem;
- No multiple inheritance in Java



In Java, multiple inheritance is not allowed for classes, only for interfaces

Given the declarations

```
interface A {
    default void m() { System.out.println("hello from A"); }
}
interface B extends A {
    default void m() { System.out.println("hello from B"); }
}
interface C extends A {}
class D implements B, C {}
```

the code

```
C c = new D();
c.m();
```

will print `hello from B`. The static type of `c` is unimportant; what counts is that it is an instance of `D`, whose most specific version of `m` is inherited from `B`.

Other writing (with abstract method instead of default method)


```
public interface A {

    void m();
}

public class ImplementationOfA implements A {

    void m() {
         System.out.println("hello from A");
    }
}
```

Introduction to JEE

Introduction

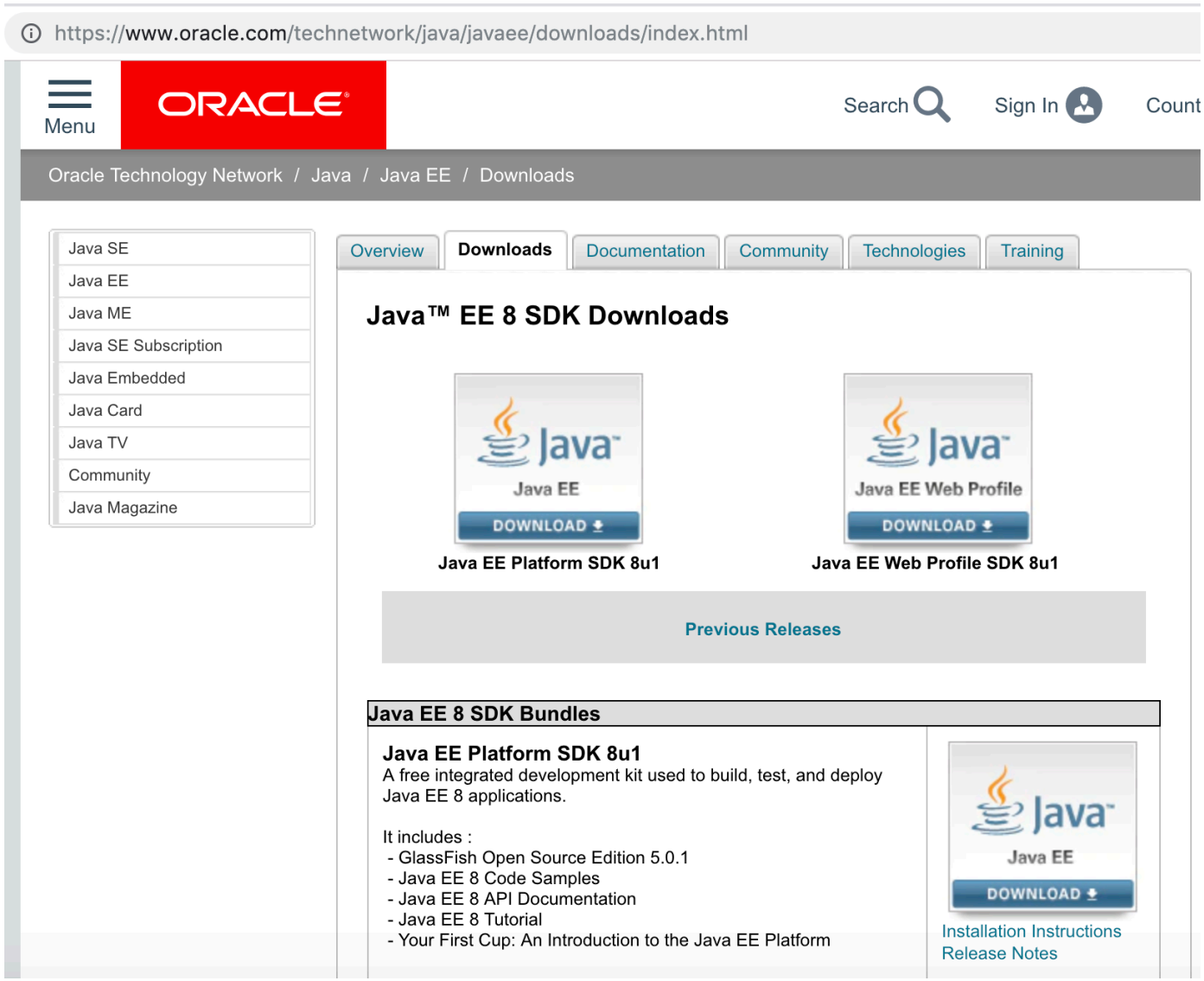
- **Java**
  - JSE : Java Standard Edition (PCs)
  - JME: Java Micro Edition (mobile phones)
  - JEE: Java Enterprise Edition (servers)
- **Objectives of JEE**
  - Facilitating the development of new component-based applications
  - Support for critical applications of the company:
    - Availability, fault tolerance, pic of activities / overloading, security ...

Java EE

- Previous name: J2EE (Java2 Enterprise Edition)
- Based on an old project, proposed by SUN aiming at the definition of a standard for developing community-driven enterprise software, and based on components.

http://www.oracle.com/technetwork/java/javaee/overview/index.html

JEE: download and installation



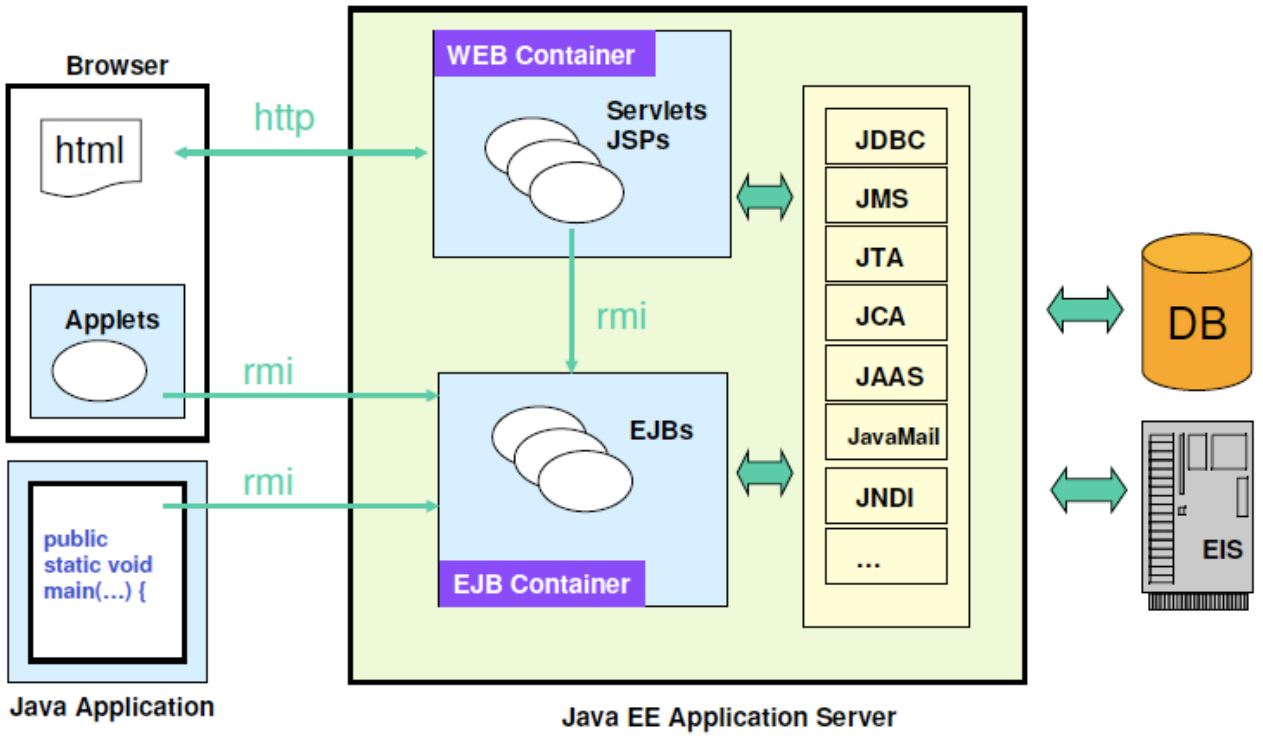
JEE: download and installation

- You also need Apache Maven and Glassfish:
  - **Maven** is a software project management and comprehension tool. Based on the concept of a project object model (POM), **Maven** can manage a project's build, reporting and documentation from a central piece of information
  - Glassfish is an open source application server (GlassFish is the Open Source Java EE Reference Implementation)
- The sample applications coming with Glassfish depend on the following software:
  - Java SE Development Kit (JDK) version 1.8.0_144.
  - Apache Maven version 3.3+.
  - The Java DB database included in the Java EE SDK

JEE application servers

- Commercial offers:
  - BEA WebLogic (haut de gamme)
  - IBM Websphere (no 1)
  - Sun Java System App Server
  - Borland Enterprise Server
  - Oracle Application Server
  - Macromedia jRun
  - SAP Web application server
  - Iona Orbix E2A
  - …
- Open source offers:
  - JBoss ()
  - OW2 JOnAS
  - Sun Glassfish (*Platform edition* de l'offre *Sun Java System App Server*)
  - Apache Geronimo (*Community edition* de IBM Websphere)
  - openEjb
  - …

Java EE - Architecture



- **Clients**
  - Thin (Web, browser) and heavy (Java Application, Applet…)
- **Application Server**
  - Container EJB + container web + business logic
  - non functional services (JEE services)
- **EIS** (Entreprise Information System) or **Database**

JEE Services

- **JDBC** (*Java DataBase Connectivity) is an API to access relational databases*
- **JNDI** (*Java Naming and Directory Interface) is an API to access name-services and directory-services such as* DNS, NIS, LDAP
- **JTA/JTS** (*Java Transaction API/Java Transaction Services) is an API which defines standard interfaces for the management of transactions;*
- **JPA** *(Java Persistance API)* is an API for the mapping between the Object model and the Relational model (ORM, Object-Relational Mapping).
- **JCA** (*JEE Connector Architecture) is an API to connect to the company's information system, such as to the* ERP (Entreprise Resource Planning).

Services JEE

- **JMX** (*Java Management Extension) provides with extensions allowing the development of web applications for the supervision of applications;*
- **JAAS** (*Java Authentication and Authorization Service) is an API for the management of authentification and access rights;*
- **JavaMail** is an API which allows the management of emails;
- **JMS** (*Java Message Service) provides with asynchronous communication* (called *MOM for Middleware Object Message) between* applications;
- **RMI-IIOP** is an API that allows communication between distant objects;

EJB components

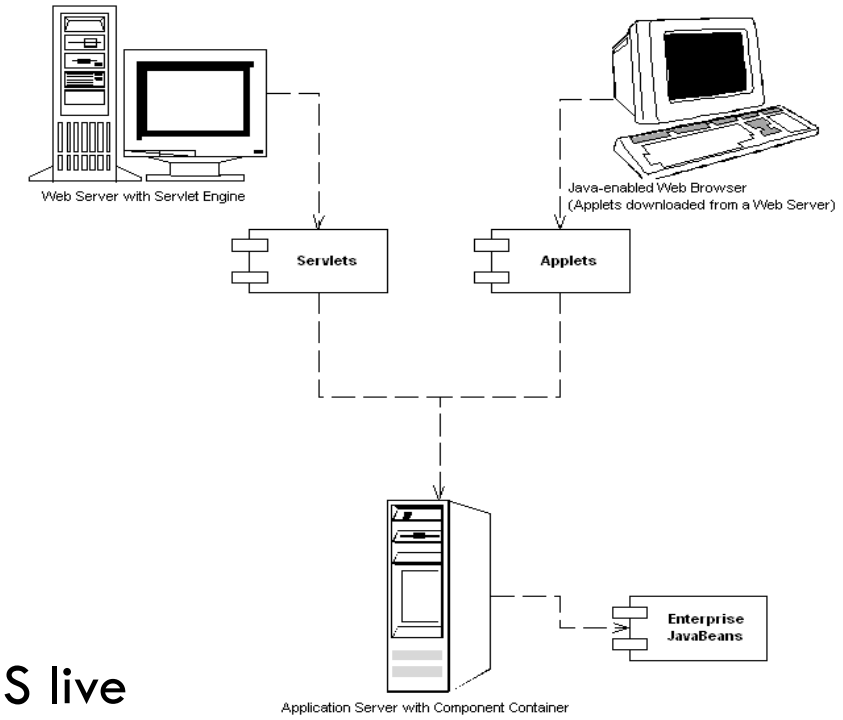
- « **A server-side component that encapsulates the business logic of an application** »
- Server side / client side (Web) programming;
- See an example of client side web programming with Brython on https://lipn.univ-paris13.fr/~cerin/CIG-S1/multiens.html In this code:
  - Load the Python interpret written in Javascript + standard library also written in Python;
  - Define the Model (a Python Class)
  - Define the View (buttons and layouts…)
  - Define the Controller (map events to buttons)

EJB Components

- We put now a focus on the application and business logics:
  - The developer only takes into account the business logic **of the** EJB.
  - The **system services** are provided by the container
    - Persistence
    - Transactions
    - Security (authentication, confidentiality, etc.)
    - Pool of objects, load balancing
- **Vocabulary**: ***Bean = EJB = component***
- Lot of versions: EJB 3.2 (**JEE 8**)

EJB components

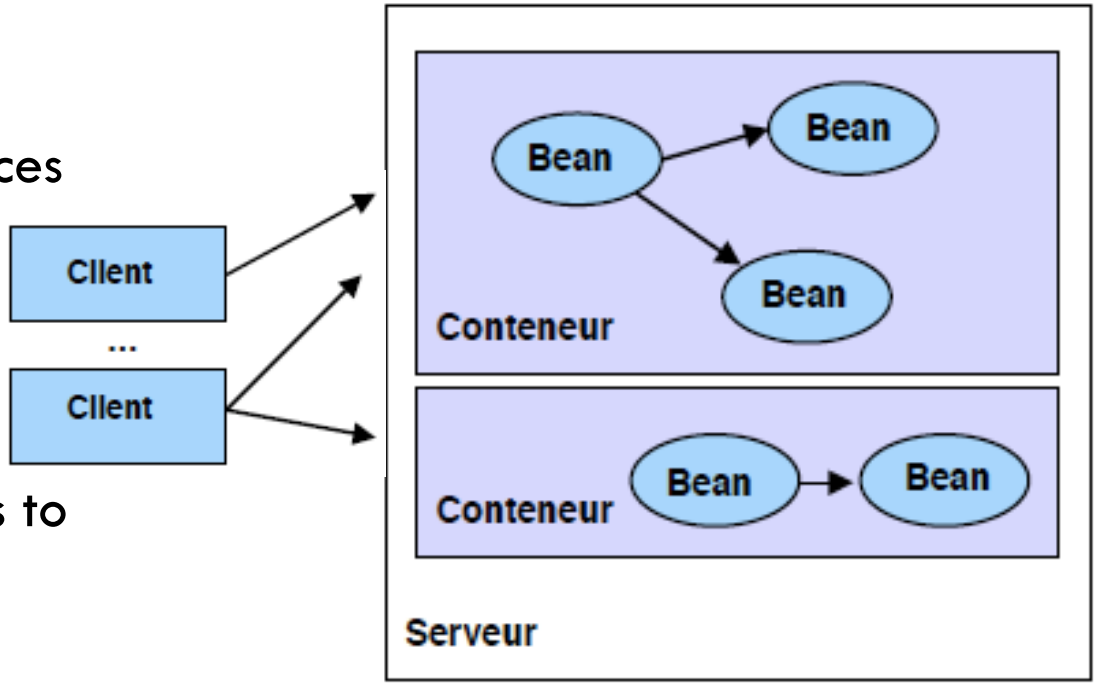
- **EJB does not provide with GUI**
  - GUI (Graphical User Interface) = client side
    - Classical applications
    - Java Beans
    - Servlets, JSP



- **EJB container**
  - environment within the server in which EJBS live
  - provide with **necessary services** to manage EJBs
  - In General, the container and application server are confused.

Execution schema for EJBs

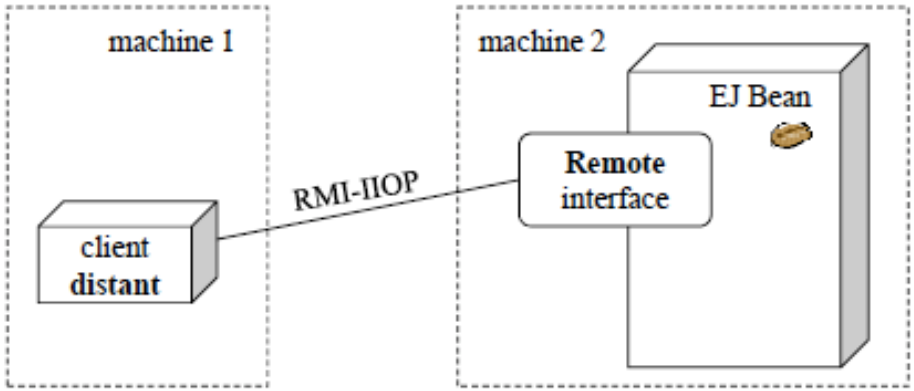
- **beans** provide with a set of services by realizing some dedicated treatments on one application ("*business logic*").
- **clients** call services.
- The **container** **intercepts** the calls to beans and it realizes diverses communs functions (persistence, transactions, security).



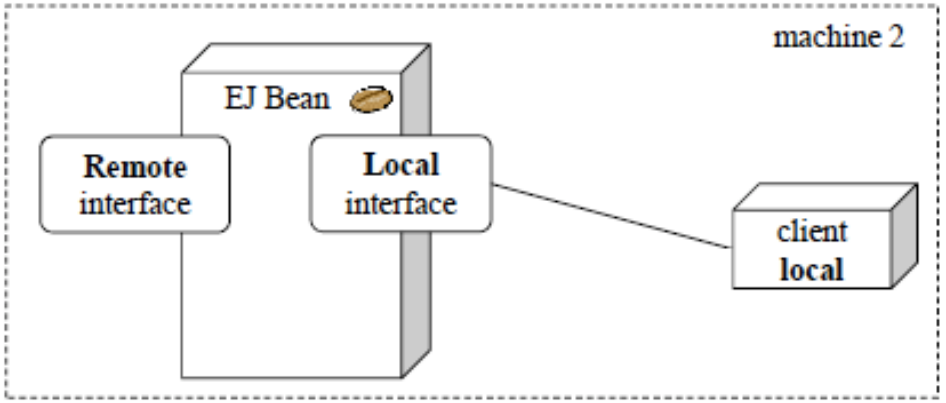
- Containers **isolate** beans from clients and from a specific server implementation

Access to EJB components

- Every EJB provides with 1 remote access **interface**
  - services (methods) exposed to clients
- The EJB client can be: servlet, applet, classical application, other EJB



- + possibly **1 local access interface**
  - services exposed by the bean to local clients
  - the same (or others) as those offered remotely



Types of EJB

- 3 types of beans:
  - Session Beans
    - Represents a treatment (services provided to a client)
  - Entity Beans
    - Represents a business object which exists in the permanent storage system (ex: client account)
  - Message-Driven Bean
    - ensures the treatment of assynchronous messages asynchrones

Session Bean

- Modeling of treatment (or session)
  - *business process*: services provided to clients
    - Example: bank account management, product catalog display, Bank data checker..
    - Life time = the **session**
  - The time a customer remains connected to the bean
- Lifecycle
  - The container creates an instance when a client logs on to the bean session.
  - It can destroy it when the client disconnects.
- Session Beans do not tolerate the fault/crash of the serveur
  - Some objects are in memory, **non persistent**
  - Unlike Entity Beans

Types de Session Beans

- Each bean session maintains a conversation with a client.
  - **Conversation** = call methods list.
- There exists two type of Beans session
  - ***Stateful Session Beans***
  - ***Stateless Session Beans***
- Each model a particular type of conversation

Stateless Session Beans

- **Stateless**: does not keep any information between two successive calls
- Some conversations can be summarized as a method call, without needing to know the current state of the bean
- The client passes all the data necessary for processing during the method callUn tel EJB est partageable consécutivement par de multiples clients
- Examples
  - bank account number validation, a currency converter…

One instance of a Stateless Session Bean is not dedicated to only one user, it can be shared

Stateful Session Beans

- Some conversations take place in the form of successive requests.
- From one request to another, there must be a way to maintain a State
- Example: a client is navigating on an e-commerce site, selects some goods, fills his basket…
- A **Stateful Session Bean** **maintains the state** during the « life » of the client
- During calls of successive methods. If a method call changes the State of the bean, during another method call the State will be available.
- Such **EJB is** dedicated to a **client** during all the life of the Session

We have one instance of *Stateful Session Bean* per client

Session Bean - Development

- **1 interface** (possibly 2 : **Local** + **Remote**) + **1 classe**
  - **Interface**
    - annotations **@javax.ejb.Local** or **@javax.ejb.Remote**

```
import javax.ejb.Remote;

@Remote
public interface CalculatriceItf {
   public double add(double v1,double v2);
   public double sub(double v1,double v2);
   public double mul(double v1,double v2);
   public double div(double v1,double v2);
}
```

Session Bean - Development

- **Class**
  - annotations **@Stateless** ou **@Stateful**

```
import javax.ejb.Stateless;

@Stateless
public class CalculatriceBean implements CalculatriceItf {
  public double add(double v1,double v2) {return v1+v2;}
  public double sub(double v1,double v2) {return v1-v2;}
  public double mul(double v1,double v2) {return v1*v2;}
  public double div(double v1,double v2) {return v1/v2;}
}
```

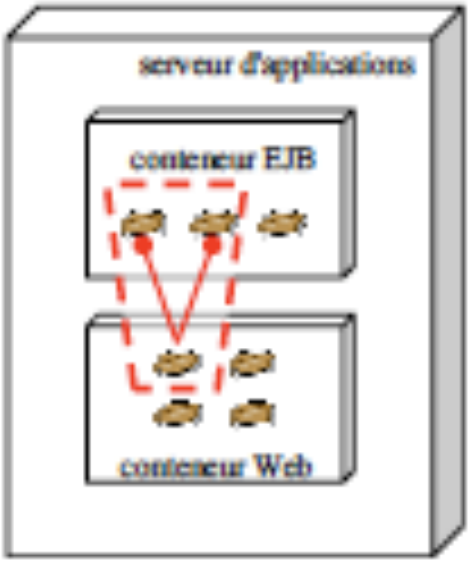
Possibility to name the *beans* : **@Stateless(name="foobar")**

By default, the name of the class

Session Bean - Development

Local Client

- Typically a servlet or a JSP
  - co-located on the same server as the bean
- Mechanism by « **dependency injection** »
  - annotation: **@EJB**
    - possibly **@EJB(name="foobar")**




```
public class ClientServlet extends HttpServlet {
  @EJB(name="foobar")
  private CalculatriceItf myBean;
  public void service( HttpServletRequest req, HttpServletResponse resp ) {
     resp.setContentType("text/html");
     PrintWriter out = resp.getWriter();
     double result = myBean.add(12,4.75);
     out.println("<html><body>"+result+"</body></html>");
} }
```

Session Bean - Development

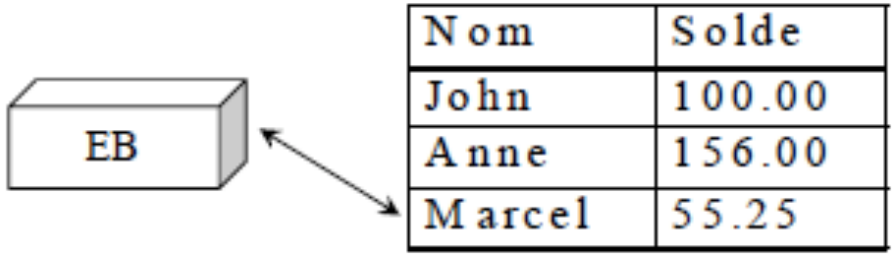
Remote Client

1. Search of the *bean in directory* **JNDI**
2. Retrieving the directory reference **JNDI**
3. Call the bean methods

```
public class Client {
  public static void main(String args[]) throws Exception {
    javax.naming.Context ic = new javax.naming.InitialContext();
    CalculatriceItf bean = (CalculatriceItf) ic.lookup("foobar");
    double res = bean.add(3,6);
} }
```

Entity Bean (EB)

- Representation of a data, manipulated by the application, typically stored in a DBMS
- The state of an Entity Bean is **persistent**.
- **Persistence** means that the status of the entity bean exists (persists) even after the duration of the application's operation (session)

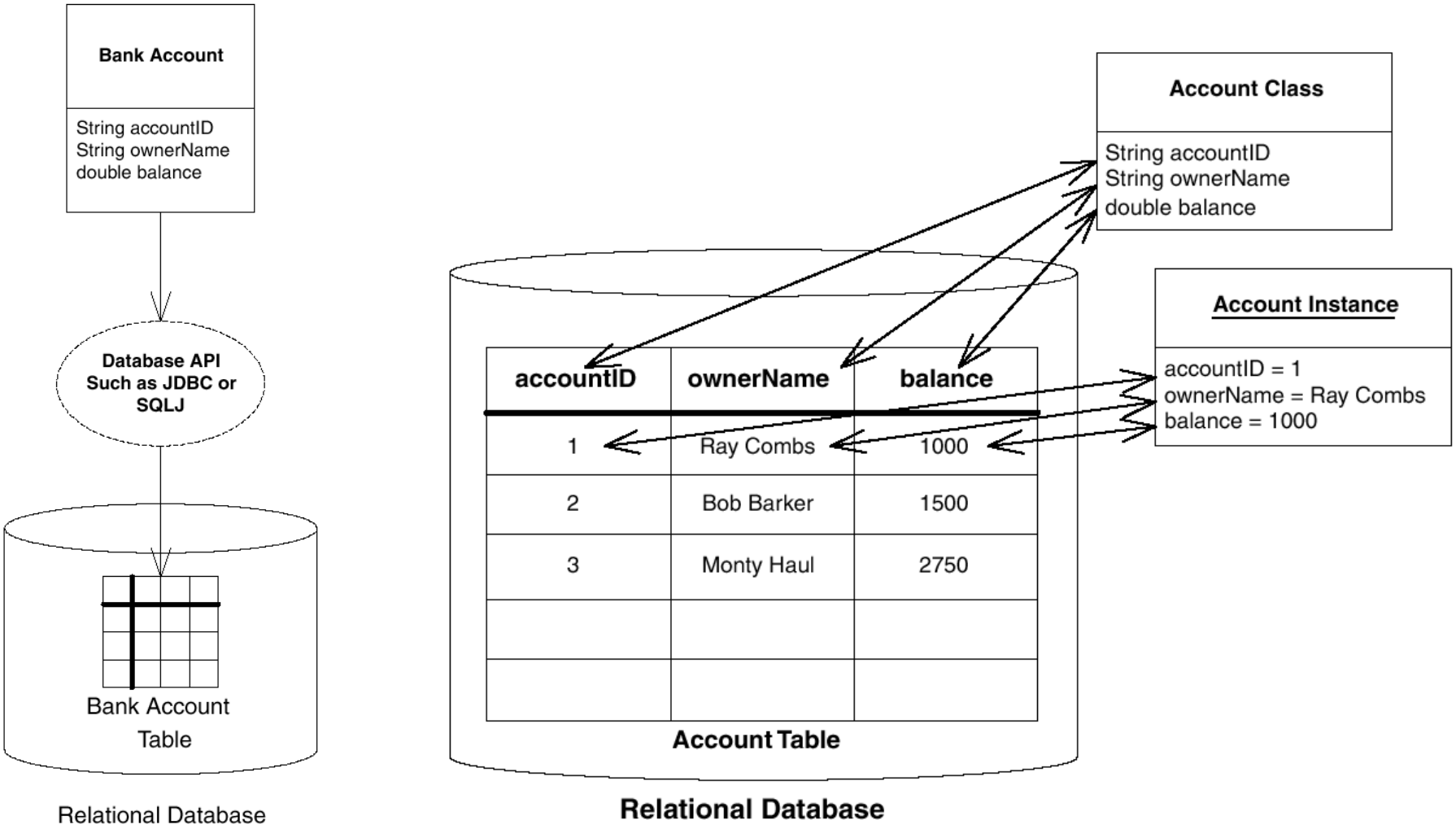


Session Bean	Entity Bean
Gestion de compte	Compte bancaire
Vérificateur de CB	Carte de crédit
Système d'entrée gestion de commandes	Commande, ligne de commande
Gestion de catalogue	Produits
Gestionnaire d'enchères	Enchère, Produit
Gestion d'achats	Commande, Produit, ligne de commande

Entity Bean (EB)

- *Mapping Object/Relational*
  - Attributes of an object are saved on a persistent media
  - Avantage: manipulating Java objects rather than SQL queries
  - Annotations Java 5 (@)
  - API JPA (**Java Persistence API**) starting from EJB 3.0
  - Persistence frameworks (Hibernate, TopLink, EclipseLink…)
- **Main interest of JPA**: allows to be independent of the framework managing the persistence

Mapping objet/BD relational



Properties of Entity Beans

- Multiple clients can use EB that "point" to the same data
- An EB instance contains a copy of the storage system data
- When many clients share one EB, they
  - receive their own instances of EB
  - share the underlying data
  - do not have to manage the **synchronization** on data

Persistence of an Entity Bean

- There exists two type of persistence for the Entity Beans:
  - Bean-Managed Persistence (BMP): the entity bean code contains the calls that access the database
  - Container-Managed Persistence (CMP): the EJB container automatically generates access calls to the database, using the bean deployment descriptor (XML file or annotations)
    - **Avantage**: redeploying the same entity bean to different JEE servers using different databases will not require any changes to the bean.
- In both cases the container is responsible for the consistency between the State of the bean and the State of the DB

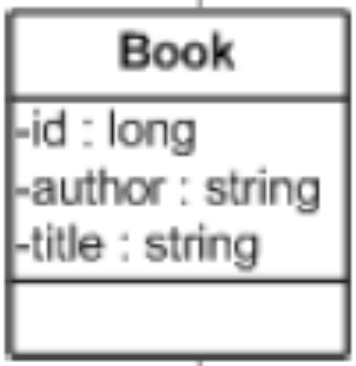
Entity Bean - Development

- annotation **@Entity** :
  - declare a class corresponding to an Entity Bean
- each class of EB is mapped to a table
  - by default, table name = name of the classa
  - except if annotation **@Table(name="...")**

- annotation **@Id**: define a primary key
- annotation **@Column**: define a column

Entity Bean - Development


```
@Entity
public class Book {

    private long id;
    private String author;
    private String title;

    public Book() {}
    public Book(String author, String title) {
        this.author = author;
        this.title = title; }

    @Id
    public long getId() { return id; }
    public void setId(long id) { this.id = id; }

    public String getAuthor() { return author; }
    public void setAuthor(String author) { this.author = author; }

    public String getTitle() { return title; }
    public void setTitle(String title) { this.title = title; } }
```



Entity Bean – Entity Manager

- ***Entity Manager***
  - provides correspondence between Java objects and relational tables
  - main entry point in persistence service
  - allows you to add records
  - allows you to execute queries
  - accessible via a dependency **injection**
    - attribut of type **javax.persistence.EntityManager**
    - annotation with **@PersistenceContext** or **@PersistenceUnit**

Entity Bean – Entity Manager

- Example
  - creation of 3 records in the book table

```
@Stateless
public class MyBean implements MyBeanItf {
    @PersistenceContext
    private EntityManager em;
    public void init() {
        Book b1 = new Book("Honore de Balzac","Le Pere Goriot");
        Book b2 = new Book("Honore de Balzac","Les Chouans");
        Book b3 = new Book("Victor Hugo","Les Miserables");
        em.persist(b1);
        em.persist(b2);
        em.persist(b3);
}   }
```

in a similar way **em.remove(b2)** removes the record from the table

Entity Bean – Entity Manager

Search by primary key

- **find method** from the entity manager

```
Book myBook = em.find(Book.class,12);
```

  - return null if the key does not exist in the table
  - IllegalArgumentException
    - if 1st parameter is not an EB class
    - if 2nd parameter does not correspond to the type of the primary key

Entity Bean – Entity Manager

Search by query

- SELECT query with EJB-QL syntax
- OBJECT keyword to designate a result to return in the form of an object
- named parameters (prefixed by:) to configure the query

```
Query q =
  em.createQuery("select OBJECT(b) from Book b where b.author = :au");
String nom = "Honore de Balzac";
q.setParameter("au",nom);
List<Book> list = (List<Book>) q.getResultList();
```

- getSingleResult() method to get a unique results
  - NonUniqueResultException in case of non-uniqueness

Entity Bean – Entity Manager

Search by pre-compiled queries

- creation of a named query **attached to** EB

```
@Entity
@NamedQuery(name="allbooks",query="select OBJECT(b) from Book b")
public class Book { ... }

Query q = em.createNamedQuery("allbooks");
List<Book> list = (List<Book>) q.getResultList();
```