

closer to 100%. This is advocated also in MVC architecture(see below).

#### EJB code

(Figure 5) In the 1990s, with the client server computing, a trend started, that is to move away from Mainframe computing. That resulted in many small separate applications in a Company/Enterprise. Many times the same data was used in different applications. A new philosophy, "Enterprise Computing", was created to address these issues. The idea was to create components that can be reused throughout the Enterprise. The Enterprise Java Beans (EJBs) were supposed to address this.

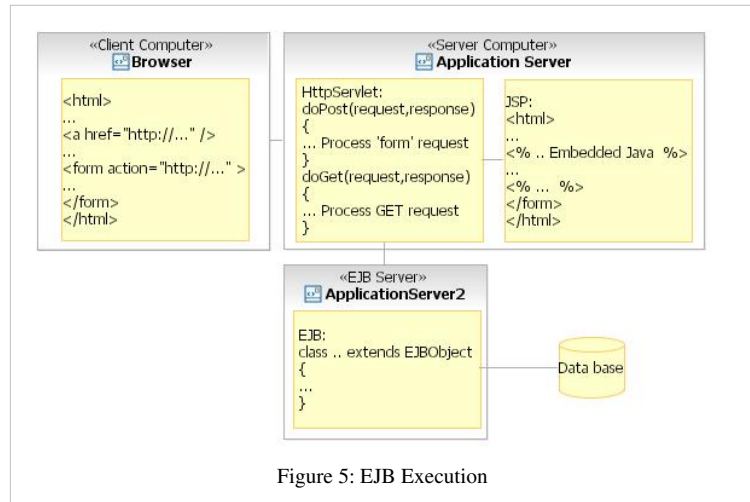


Figure 5: EJB Execution

An **EJB** is an application component that runs in an EJB container. The client accesses the EJB modules through the container, never directly. The container manages the life cycle of the EJB modules, and handles all the issues that arise from network/enterprise computing. Some of those are security/access control, object pooling, transaction management, ... .

EJBs have the same problems as any reusable code: they need to be generic enough to be able to be reused and the changes or maintenance of EJBs can affect existing clients. Many times EJBs are used unnecessarily when they are not really needed. An EJB should be designed as a separate application in the enterprise, fulfilling one function.

Combine J2EE components to create an MVC architecture

This leads us to the three layers/tiers as shown in (Figure 6).

In modern web applications, with lots of static data and nice graphics, how the data is presented to the user became very important and usually needs the help of a graphic artist.

To help programmers and graphic artists to work together, the separation between data, code, and how it is presented became crucial.

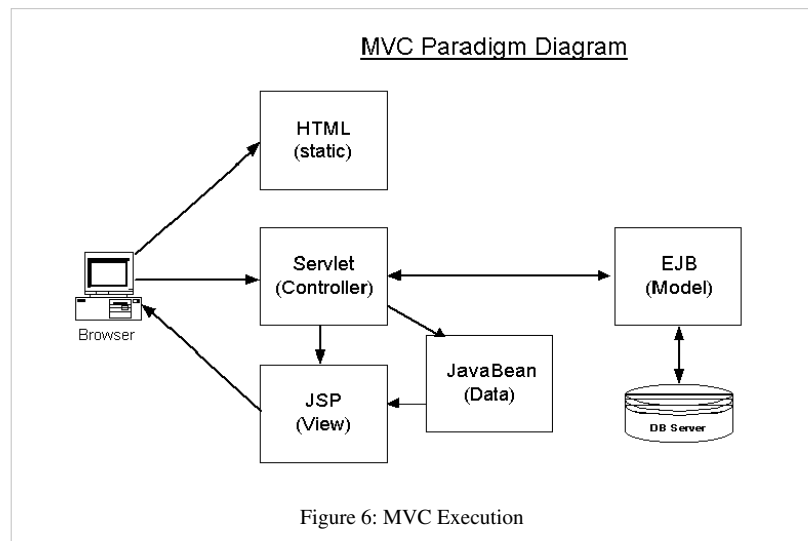


Figure 6: MVC Execution

- The **view** (User Interface Logic) contains the logic that is necessary to construct the presentation. This could be handled by JSP technology.
- The servlet acts as the **controller** and contains the logic that is necessary to process user events and to select an appropriate response.
- The business logic (**model**) actually accomplishes the goal of the interaction. This might be a query or an update to a database. This could be handled by EJB technology.

For more information about MVC, please see MVC.

## JINI

After J2EE Sun had a vision about the next step of network computing. That is JINI. The main idea is that in a network environment, there would be many independent services and consumers. JINI would allow these services/consumers to interact dynamically with each other in a robust way. The basic features of JINI are:

- **No user intervention** is needed when services are brought on or offline. (In contrast to EJBs where the client program has to know the server and port number where the EJB is deployed, in JINI the client is *supposed to find*, to discover, the service in the network.)
- **Self healing** by adapting when services(consumers of services) come and go. (Services periodically need to renew a lease to indicate that they are still available.)
- Consumers of JINI services do not need prior knowledge of the service's implementation. The **implementation is downloaded dynamically** and run on the consumer JVM, without configuration and user intervention. (For example, the end user may be presented with a slightly different user interface depending upon which service is being used at the time. The implementation of the user interface code would be provided by the service being used.)

A minimal JINI network environment consists of:

- One or more **services**
- A **lookup-service** keeping a list of registered services
- One or more **consumers**

JINI is not widely used at the current writing (2006). There are two possible reasons for it. One is JINI a bit complicated to understand and to set it up. The other reason is that Microsoft pulled out from Java, which caused the industry to turn to the use of proprietary solutions.

## First Java Program

Navigate **Getting Started** topic:

- Installing Java on Your Computer
- Compiling programs
- Running Java programs
- Developing the first Java program
- Understanding a Java program

## Hello World

Generally when you first start programming in any language, you'll start with the traditional *Hello World* example. That said, let's start building your first Java program. You guessed it, it's *Hello World*! Before starting this exercise, make sure you know how to compile and run Java programs.

Open your IDE and write the following text. Pay close attention to capitalization, as Java is case sensitive.

```
public class HelloWorld {  
    public static void main(String[] args) {  
        System.out.println("Hello, world!");  
    }  
}
```

Save it as `HelloWorld.java`. Again, make sure that the filename is the same case as the class name. Compile and run it:

```
javac HelloWorld.java
java HelloWorld
```

Your computer should display

```
Hello, world!
```

## Line-by-line Analysis

The first line of the class,

```
public class HelloWorld {
```

declares a Java class named HelloWorld. This class is declared **public** - it is available to any other class. The next line,

```
public static void main(String[] args) {
```

begins a Java method named main. The main method is where Java will start executing the program. **args** is a method parameter which will contain the command line arguments used to run the program. The method must be both **public** and **static** for the program to run correctly. For more information on modifiers such as **public** and **static**, see Access Modifiers, though at this level, you don't need to know a whole lot about them.

The

```
System.out.println("Hello, world!");
```

statement sends the text Hello, world! to the console (with a line terminator). The final two braces mark the end of the main method and the end of the class.

## Modifying the Program

Now, we will modify this program to print the first command line argument, if there is one, along with the greeting. For example, if you invoke the program as

```
java HelloWorld wikibooks
```

it will print

```
Hello, wikibooks!
```

Go back to the program, and modify it to read

```
public class HelloWorld {
    public static void main(String[] args) {
        String who;
        if (args.length > 0) {
            who = args[0];
        } else {
            who = "World";
        }
        System.out.println("Hello, " + who + "!");
    }
}
```

Run it again. It should display