MET's Institute of Engineering

Bhujbal Knowledge City, Adgaon, Nashik.

Department of Computer Engineering

# "RUBY AND RAILS, EJB"

Prepared By

Prof. Anand N. Gharu

(Assistant Professor)

Computer Dept.

CLASS : TE COMPUTER 2019

SUBJECT : WT (SEM-II)

UNIT : VI

25 April 2023

Note: The material to prepare this presentation has been taken from internet and are generated only for students reference and not for commercial use.

# SYLLABUS

**Introduction to Ruby:** Origins & uses of Ruby, scalar types and their operations, simple input and output, control statements, fundamentals of arrays, hashes, methods, classes, code blocks and iterators, pattern matching.

**Introduction to Rails:** Overview of Rails, Document Requests, Processing Forms, Rails Applications and Databases, Layouts, Rails with Ajax.

**Introduction to EJB.**

# Syllabus

**Ruby :**

- Introduction to Ruby:
- Origins & uses of Ruby,
- Scalar types and their operations,
- Simple input and output,
- control statements,
- fundamentals of arrays,
- hashes, methods, classes,
- code blocks and iterators,
- Pattern matching

**Rail**

- Introduction to Rails:
- Overview of Rails,
- Document Requests,
- Processing Forms,
- Rails Applications and Databases,
- Layouts,
- Rails with Ajax.

**EJB**

- Introduction to EJB.

# INTRODUCTION TO RUBY

## Ruby- Origins and Uses of Ruby

- -Ruby is "A Programmer's Best Friend".
- -Designed by Yukihiro Matsumoto; released in 1996
- - Use spread rapidly in Japan
- - Use is now growing in part because of its use in Rails
- - A pure object-oriented purely interpreted scripting language
- - Related to Perl and JavaScript, but not closely

## Features of ruby

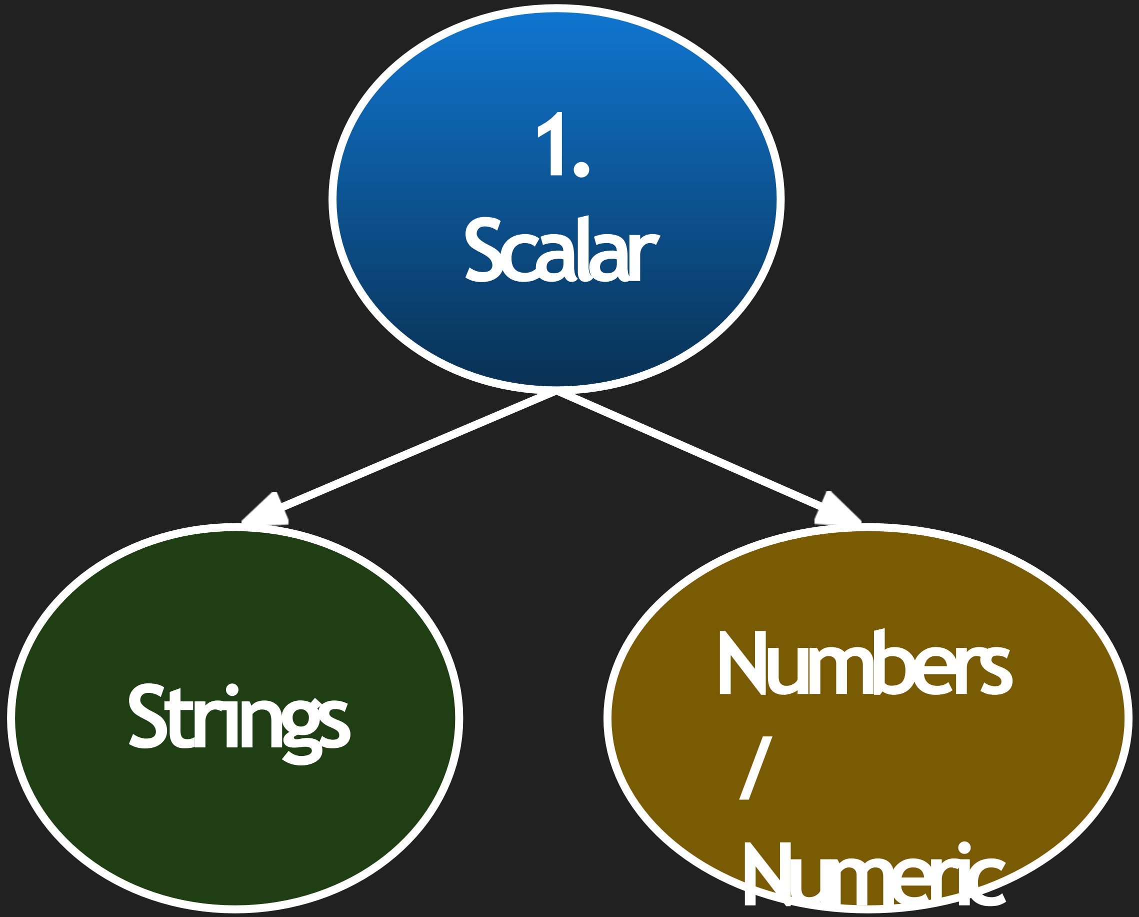
- Ruby is an open-source and is freely available on the Web, but it is subject to a license.
- Ruby is a general-purpose, interpreted programming language.
- Ruby is a true object-oriented programming language.
- Ruby is a server-side scripting language similar to Python and PERL.
- Ruby can be embedded into Hypertext Markup Language (HTML).
- Ruby has a clean and easy syntax that allows a new developer to learn very quickly and easily.
- Ruby is very much scalable and big programs written in Ruby are easily maintainable.
- Ruby can be used for developing Internet and intranet applications.
- Ruby can be installed in Windows and POSIX environments.
- Ruby support many GUI tools such as Tcl/Tk, GTK, and OpenGL.
- Ruby can easily be connected to DB2, MySQL, Oracle, and Sybase.
- Ruby has a rich set of built-in functions, which can be used directly into Ruby scripts.

## RUBY – Scalar Data Type

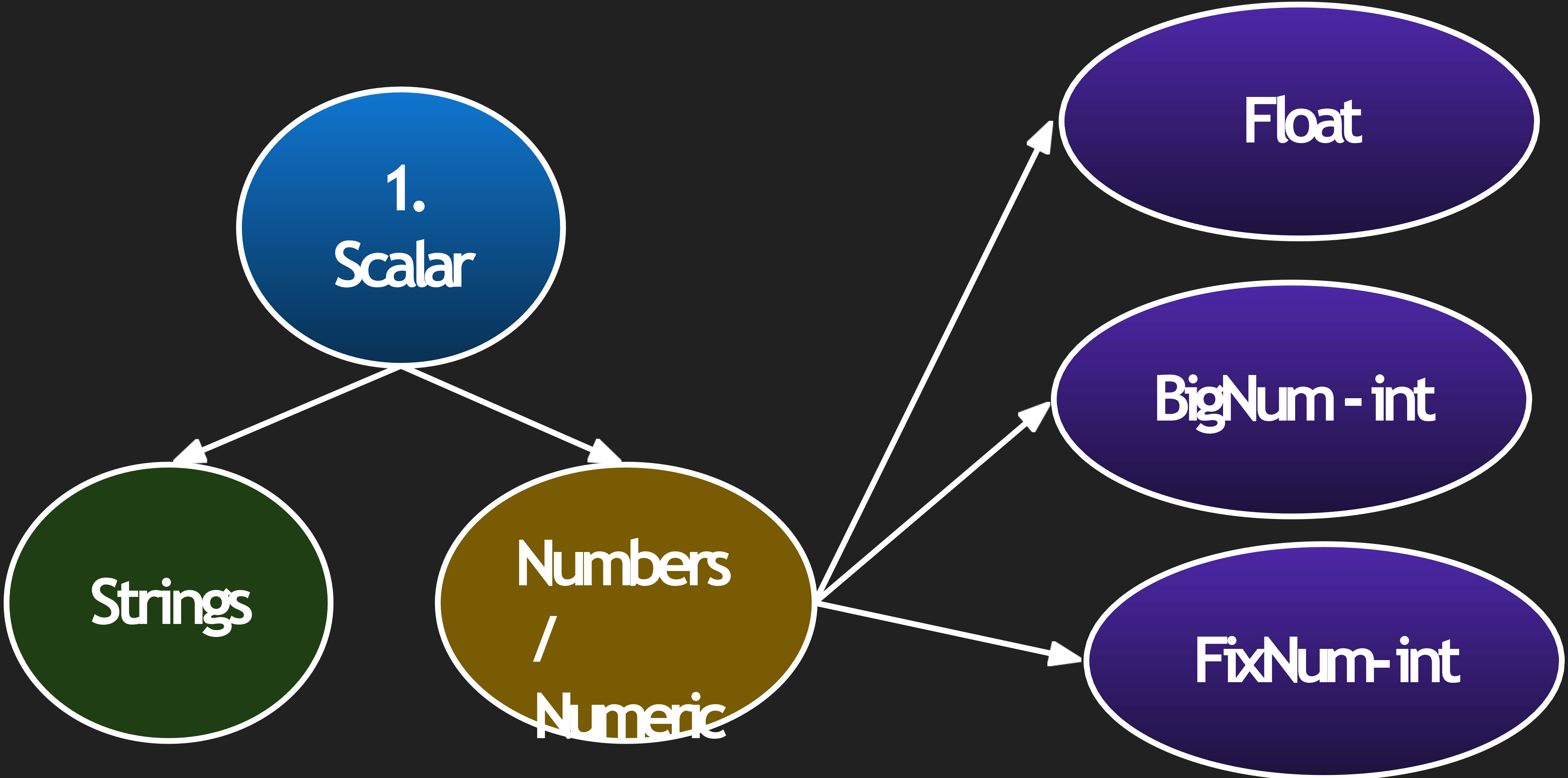
## Ruby- Data Types

1. Scalar
2. Arrays
3. Hashes
4. Boolean
5. Symbols

# Ruby- Data Types



# Ruby- Scalar Data Types

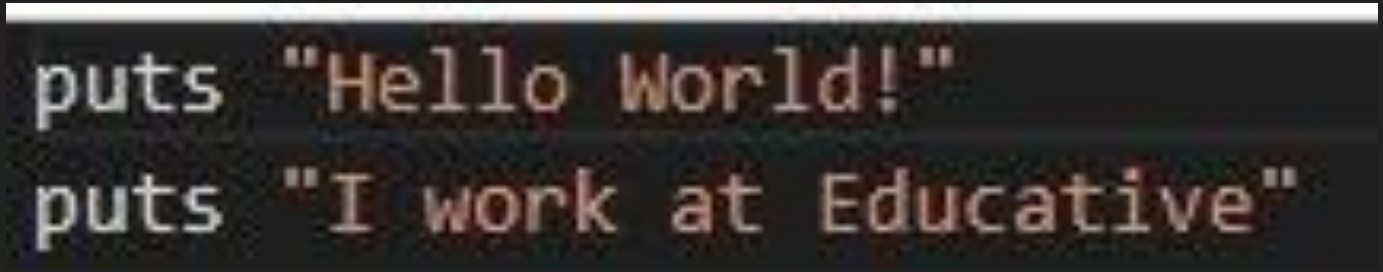


# Ruby- Data Types- Strings

**Strings:**

A string is made up of multiple characters.

They are defined by enclosing a set of characters within single (‘x’) or double (“x”) quotes.

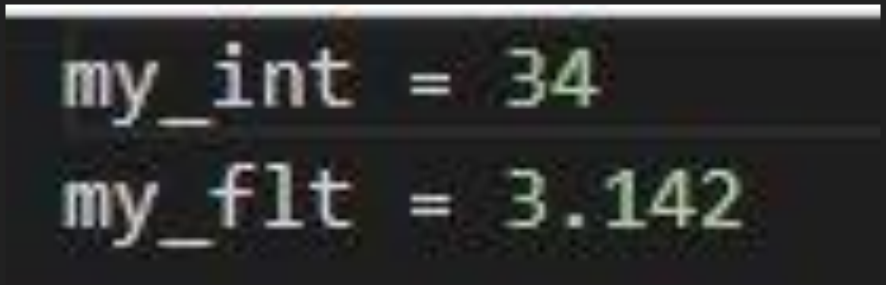


```
puts "Hello World!"
puts "I work at Educative"
```

## Ruby- Data Types- Numbers

**Numbers :**

- A number is a series of digits that use a dot as a decimal mark (where one is required).
- Integers and floats are the two main kinds of numbers; Ruby can handle them both.



```
my_int = 34
my_flt = 3.142
```

# Ruby- Data Types- Boolean

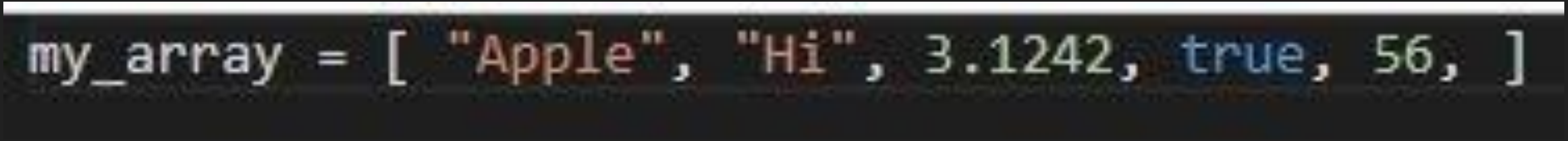
**Booleans :**

- The Boolean data type represents only one bit of information that says whether the value is  true or false.
- A value of this data type is returned when two values are compared.
- In Ruby there are three main boolean operators:
  - ! which represents “NOT”,
  - && represents “AND”
  - || represents “OR”.
  - == for compare two values

# Ruby- Data Types- Arrays

Arrays

- An array can store multiple data items of all types.
- Items in an array are separated by a comma in-between them and enclosed within square brackets.
- The first item of the array has an index of 0

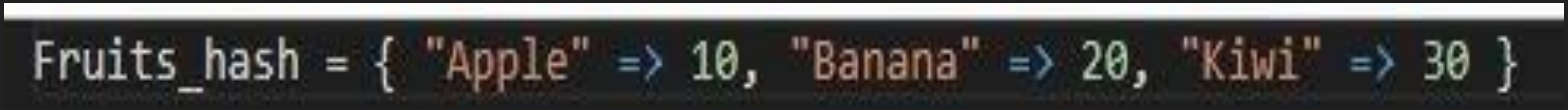


```
my_array = [ "Apple", "Hi", 3.1242, true, 56, ]
```

# Ruby- Data Types- Hashes

Hashes

- A hash stores key-value pairs.
- Assigning a value to a key is done by using the => sign.
- Key-value pairs are separated by commas, and all the pairs are enclosed within  curly braces.

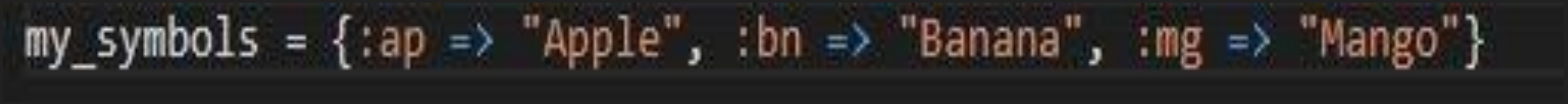


```
Fruits_hash = { "Apple" => 10, "Banana" => 20, "Kiwi" => 30 }
```

# Ruby- Data Types- Symbols

Symbols

- Symbols are a lighter form of strings.
- They are preceded by a colon (:), and used instead of strings because they take up less memory space and have a better performance.



```
my_symbols = {:ap => "Apple", :bn => "Banana", :mg => "Mango"}
```

# RUBY – Operation on Scalar Data Type

# Ruby- Operations/Methods on Numbers

| Method | Use | Example |
|---|---|---|
| .even? | to check whether or not an integer is even. Returns a true or false boolean. | 15.even? #=> false 4.even? #=> true |
| .odd? | to check whether or not an integer is odd. Returns a true or false boolean. | 15.odd? #=> true |
| .ceil | rounds floats up to the nearest integer number | 6.7.ceil #=> 7<br>8.3.ceil #=> 9 |
| .floor | rounds floats down to the nearest integer number | 8.3.floor #=> 8<br>6.7.floor #=> 6 |
| .pred | return the previous consecutive integer | 15.pred #=> 14 |
| .to_s | returns a string of that number. | 15.to_s #=> "15" |

# Ruby- Operations/Methods on String

| Method | Use | Example |
|---|---|---|
| str.chars | Convert a String to a character array | char_array = "abcdeABCDE".chars |
| str.size<br>str.length | Get the length of a String | "HELLO<br>World".length<br>"HELLO World".size |
| str.reverse | Reverse a String | str = "Anna"<br>str.reverse |
| str.include? | Search for one or more characters of a String | "Hello world".include?("w") |
| str.downcase | will convert each character of a string into lowercase | "HELLO<br>World".downcase #<br>"hello world" |
| str.upcase | will convert each character of a string into uppercase. | "hello worlD".upcase<br># "HELLO WORLD" |

## RUBY Simple Input and Output

## Simple Input and Output

**Screen Output**

- puts method is used to display the output on the screen
- Operands for the puts method is String
- To display variable value use #{....}
- Example:
  - Rno=1
  - Puts “My Roll No is #{Rno}”

## Simple Input and Output

### Screen Input

- gets method is used to take input from keyboard
- This method reads line of input
- gets.to_i to take integer value & gets.to_f for float value
- Example:
  - puts “Enter your age”
  - age=gets.to_i

# RUBY – Control Statements

## Ruby if...else Statement

**Syntax**

```
if conditional [then]
   code...
[elsif conditional [then]
   code...]...
[else
   code...]
end
```

**Example**

```
x = 1
if x > 2
   puts "x is greater than 2"
elsif x <= 2 and x!=0
   puts "x is 1"
else
   puts "I can't guess the number"
end
```

## Ruby unless Statement

**Executes code if conditional is false. If the conditional is true, code specified in the else clause is executed.**

**Syntax**

```
unless conditional
  [then]  code
[else
  code
]  end
```

**Example**

```
x = 1
unless x>=2
  puts "x is less than 2"
  else
  puts "x is greater than 2"
  end
```

## Ruby case Statement

**Syntax**

```
case expression
[when expression [, expression ...]  [then]
   code ]...  [else
   code ]  end
```

**Example**

```
$age = 5
case        $age
when 0 .. 2      puts "baby"
when 3 .. 6      puts "little child"
when 7 .. 12      puts "child"
when 13 .. 18  puts "youth"
else  puts "adult"
end
```

# RUBY – Array

## Ruby- Arrays

- Ruby arrays are ordered, integer-indexed collections of any object. Each element in an  array is associated with and referred to by an index.
- an index of -1 indicates the last element of the array, -2 is the next to last element in the  array, and so on.
- Ruby arrays can hold objects such as String, Integer, Fixnum, Hash, Symbol, even other  Array objects.

# Ruby- Creating Arrays

Method 1

```
cars = Array.new
```

Creating array using new method

Method 2

```
cars = Array.new (20)
```

You can set the size of an array at the time of creating array

Method 3

```
CARS= Array.new(3, "Oddi",
```

This will Produce O/P ["Oddi", "Oddi", "Oddi ]

# Ruby- Array Built-in Methods

| Method | Example | Use of Method |
|---|---|---|
| .length | array.length | The .length method tallies the number of elements in your array and returns the count: |
| .first | array.first | The .first method accesses the first element of the array, the element at index 0: |
| .last | array.last | The .last method accesses the last element of the array: |
| .take | array.take(3) | The .take method returns the first n elements of the array: |
| .drop | array.drop(3) | The .drop method returns the elements after n elements of the array: |
| .pop | array.pop | The .pop method will permanently remove the last element of an array |
| .push | array.push(98) | The .push method will allow you to add an element to the end of an array |

## RUBY – Hashes

## Ruby- Hashes

- A Hash is a collection of key-value pairs like this: "employee" = > "salary". It is similar to an Array, except that indexing is done via arbitrary keys of any object type, not an integer index.
- If you attempt to access a hash with a key that does not exist, the method will return nil.

## Ruby- Creating Hashes

Method 1

```
months = Hash.new
```

Creating Hash using new method

Method 2

```
months = Hash.new( "month")
```

hash with a default value

Method 3

```
H = Hash["a" => 100,"b" => 200]
```

puts "#{H['a']}"

This will produce o/p–

100

# Ruby- Hashes Vs Array

| Hashes | Arrays |
|---|---|
| whereas hashes support any object as a key | With arrays, the key is an integer |
| Elements in Hash are not ordered | Elements in Array are in ordered |
| Example: months = Hash.new | Example: cars = Array.new |

# RUBY – Methods, Classes, Code Block & Iteration

## Methods

- Ruby methods prevent us from writing the same code in a program again and again.
- It is a set of expression that returns a value.

- Defining Method
  - Ruby method is defined with the def keyword followed by method name.
  - At the end use end keyword.
  - Methods name should always start with a lowercase letter

## Methods

Syntax:

```
def
    methodName
    code...
end
```

**Example**

**:**

```
def welcome
 puts “Welcome
dear  Students”
end
```

## Methods

**Local variables**

The Local variables either are formal parameters or are variables created in a method. A variable is created in a method by assigning an object to it. The scope of a local variable is from the header of the method to the end of the method.

**Parameters**

The parameter values that appear in a call to a method are called actual parameters. The parameter names used in the method, which correspond to the actual parameters, are called formal

parameters

## Classes

- Each Ruby class is an instance of class Class. Classes in Ruby are first-class objects.
- Ruby class always starts with the keyword class followed by the class name. Conventionally, for class name we use  CamelCase. The class name should always start with a capital letter. Defining class is finished with end keyword

Syntax:

```
class
   ClassName
   codes...
end
```

**Example**

```
Class Vehicle {
  Number no_of_wheels
  String Name
  Function speeding {
                    }
   Function driving {
  }
}
```

## Classes

- **Inheritance**
  - Subclasses are defined in Ruby with the left angle bracket (<):
  - class My_Subclass < Base_class
- Ruby modules provide a naming encapsulation that is often used to define libraries of methods

## Code Blocks

- Ruby code blocks are called closures in other programming languages. It consist of a group of codes which is  always enclosed with braces or written between do..end.

- A block is written in two ways,  1.Multi-line between do .. end

  Example:

  ```
  do
  club.enroll(person
  )  person.socialize
  end
  ```

  2.Inline between braces { }

  Example: { puts
  "Hello" }

# Iterators- Examples

Each iterator

Example:

```
(0..9) . each do | i |
          puts i
     end
```

Times iterator

Example:

```
7.times do |i|
  puts
i  end
```

Upto iterator

Example:

```
7.upto(4) do |i|
puts i
end
```

downto iterator

Example:

```
7.downto(4) do |i|
puts i
end
```

Each iterator

Example O/P:

```
0 1 2 …….9
```

Times iterator

Example O/P:

```
0 1 2 …….6
```

Upto iterator

Example O/P:

```
4 5 6 7
```

downto iterator

Example O/P:

```
7 6 5 4
```

## Iterators

- The word iterate means doing one thing multiple times.
- “Iterators” is the object-oriented concept in Ruby.
- Iterators are the methods which are supported by collections(Arrays, Hashes etc.).
- There are many iterators in Ruby as follows:

1. Each Iterator
2. Collect Iterator
3. Times Iterator
4. Upto Iterator
5. Downto Iterator
6. Step Iterator
7. Each_Line Iterator

## RUBY – Pattern Matching

## Ruby- Pattern Matching

- Pattern matching is a feature that is commonly found in functional programming languages.
- A regular expression is a sequence of characters that define a search pattern, mainly for use in pattern  matching with strings.
- ~ / / This is used for Pattern matching or else can use .match() method

- **Example:-**

```
line1 = "Cats are smarter than dogs";
if ( line1 = ~ / Cats( . *) / )
  puts "Line 1 contains Cats"
end
```

# Pattern Matching- Regular Expression

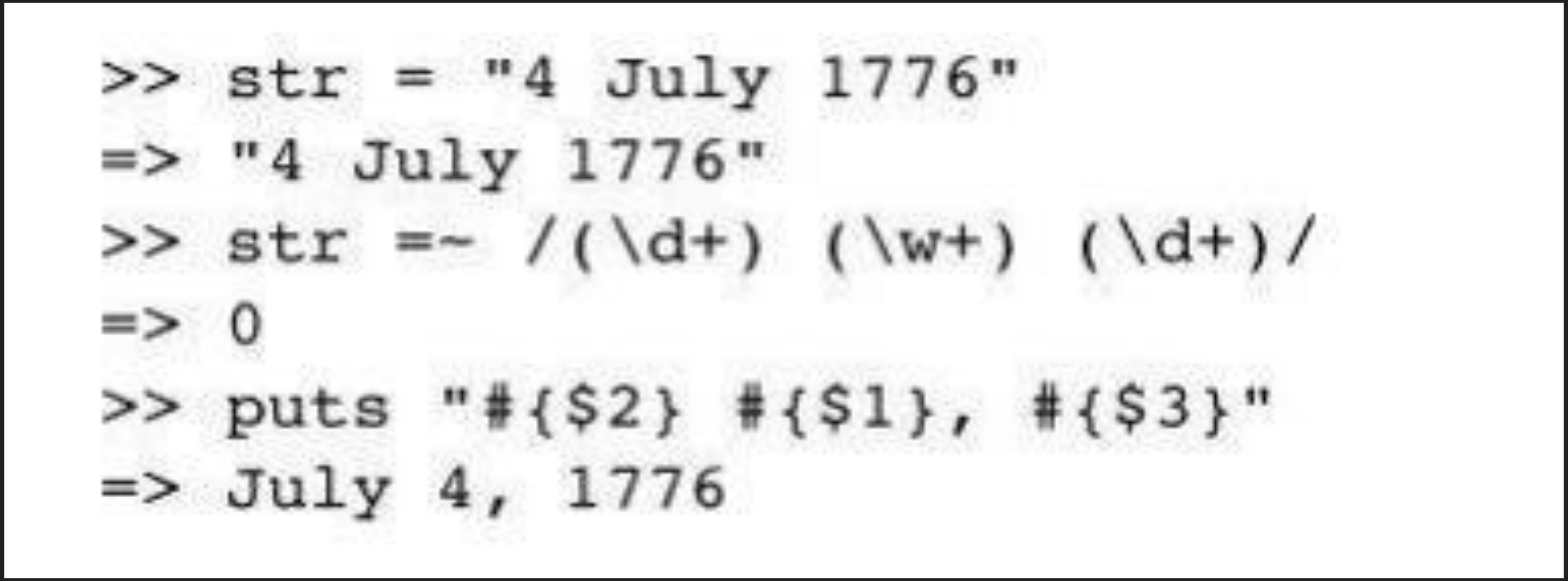
short expressions for specifying character ranges

- \w is equivalent to [0-9a-zA-Z_]
- \d is the same as [0-9]
- \s matches white space
- \W anything that’s not in [0-9a-zA-Z_]
- \D anything that’s not a number
- \S anything that’s not a space

Modifiers

- + is for 1 or more characters
- * is for 0 or more characters
- ? is for 0 or 1 character
- i is for ignores case when matching text.
- m is for matches multiple lines
- ^ Beginning of Line
- $ End of Line

## Pattern Matching- Example



```
>> str = "4 July 1776"
=> "4 July 1776"
>> str =~ /(\d+) (\w+) (\d+)/
=> 0
>> puts "#{$2} #{$1}, #{$3}"
=> July 4, 1776
```

**Remembering Matches** The part of the string that matched a part of the pattern can be saved in an implicit variable for later use. The part of the pattern whose match you want to save is placed in parentheses. The **substring that matched the first parenthesized part of the pattern is saved in $1, the second in $2, and so forth.**

# Rails

## Overview of Rails

## Overview of Rail

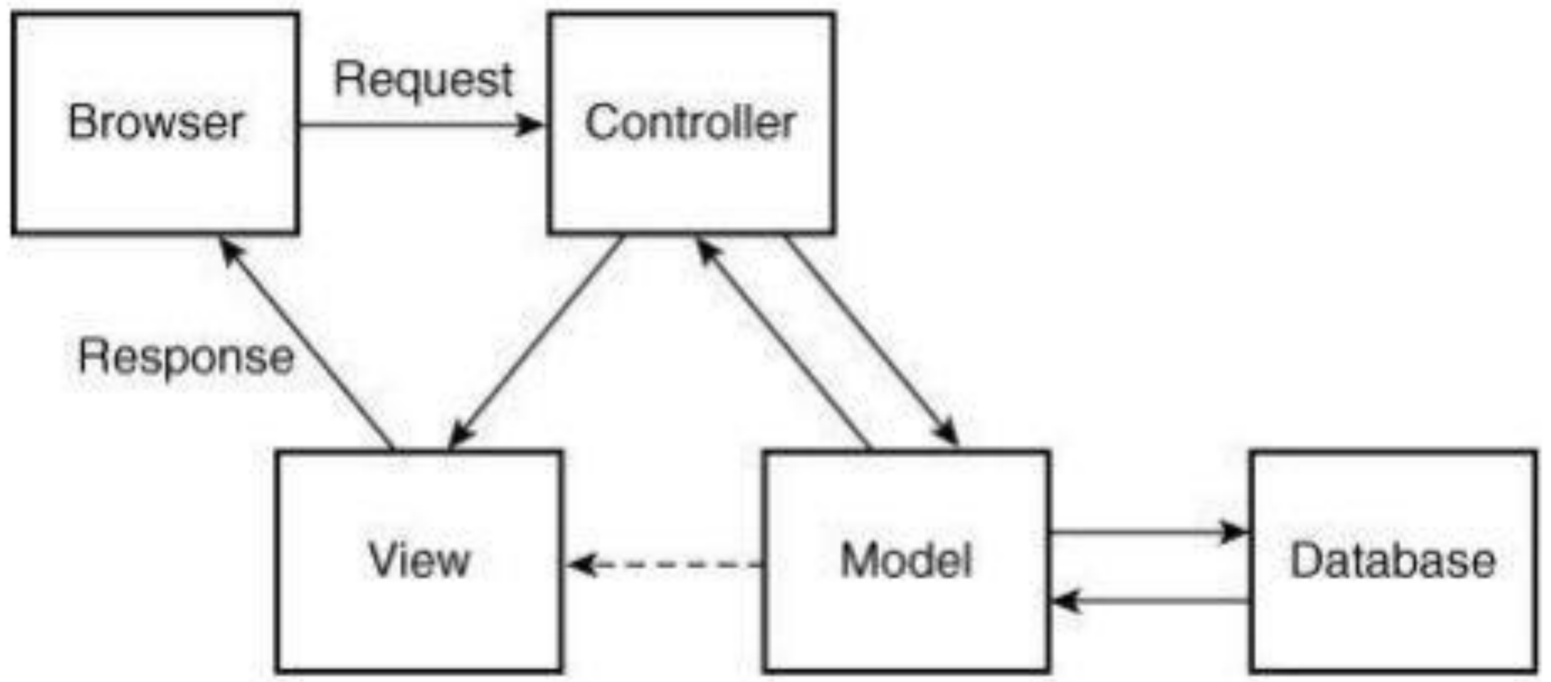
- Ruby on Rails is a server-side web application development framework .
- Written in Ruby by David Heinemeier Hansson.
- It is based on MVC pattern.
- You could develop a web application at least ten times faster with Rails than you could with a typical Java framework.
- An open source Ruby framework for developing database-backed web applications.
- Configure your code with Database Schema.
- No compilation phase required.

## Overview of Rail

### Why Ruby on Rails?

1. Easy to learn
2. Open source (very liberal license)
3. Rich libraries
4. Very easy to extend
5. Truly object-oriented
6. Less coding with fewer bugs
7. Helpful communication

## Overview of Rail



A request and response in a Rails application

https://oseven.in/files/58ce5021925bd.pdf

# Overview of Rail

## Where to use Ruby on Rails?

You can use Ruby on Rails application in various area of web development like in a

- long term project which needs large transformation,
- in the project that has heavy traffic,
- to develop a short prototype,
- in a project that requires wide range of complex functions, etc.

# Overview of Rail

## Advantages of Ruby on Rails

1. **Tooling:** Rails provides tooling that helps us to deliver more features in less time.
2. **Libraries:** There's a 3rd party module(gem) for just about anything we can think of.
3. **Code Quality**: Ruby code quality significantly higher than PHP or NodeJS equivalents.
4. **Test Automation:** The Ruby community is big into and test automation and testing.
5. **Large Community:** Ruby is large in the community.
6. **Productivity:** Ruby is incredibly fast from another language. Its productivity is high.

## Overview of Rail

### Disadvantages of Ruby on Rails

1. **Runtime Speed:** The run time speed of Ruby on Rails is slow as compare to Node.Js
2. Lack of Flexibility
3. Boot Speed
4. Documentation
5. **Multithreading:** Ruby on Rails supports multithreading, but some IO libraries do not  support multithreading

## Rails Application and Database

## Rails Applications and Databases

- Rails uses an **object-relational mapping (ORM)** approach to relate the parts of a relational database to object-oriented constructs.
- Each relational database table is implicitly mapped to a class.
- For example, if the database has a table named employees, the Rails application program that uses employees will have a class named Employee.
- Rows of the employees table will have corresponding objects of the Employee class, which will have methods to get and set the various state variables, which are Ruby attributes of objects of the class.

## Rails Applications and Databases

- In sum, an ORM maps
  - tables to classes,
  - rows to objects, and
  - columns to the fields of the objects.
- Furthermore, the Employee class will have methods for performing table-level operations such as finding an object with a certain attribute value.
- The key aspect of the ORM in Rails is that it is implicit: The classes, objects, methods, an attributes that represent a database in Ruby are automatically built by Rails

## Rails Applications and Databases- Example

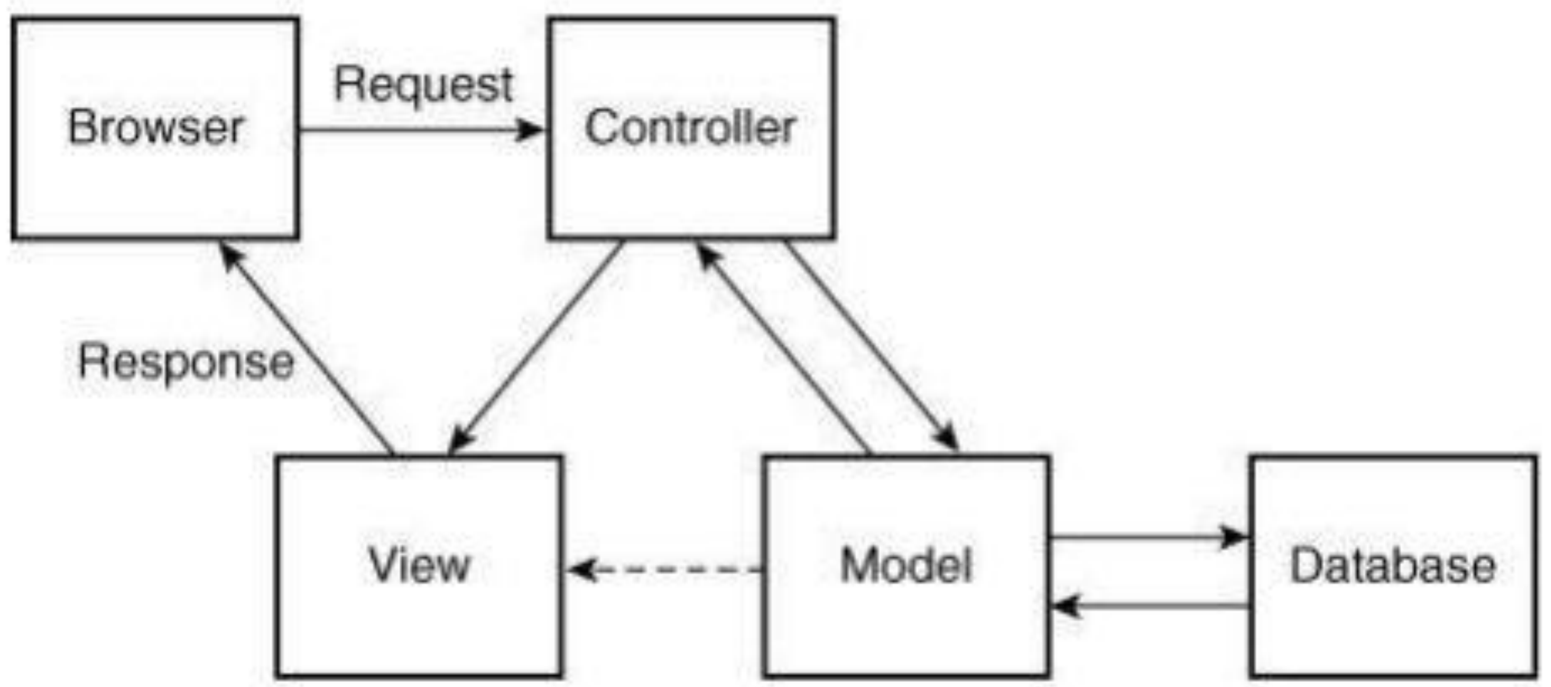
For this example, we create a new application named cars in the examples directory with the following command:

```
>rails -d mysql cars
```

The d flag followed by mysql appears in the rails command in order to tell Rails that this application will use a MySQL database.

## Rails Document Request

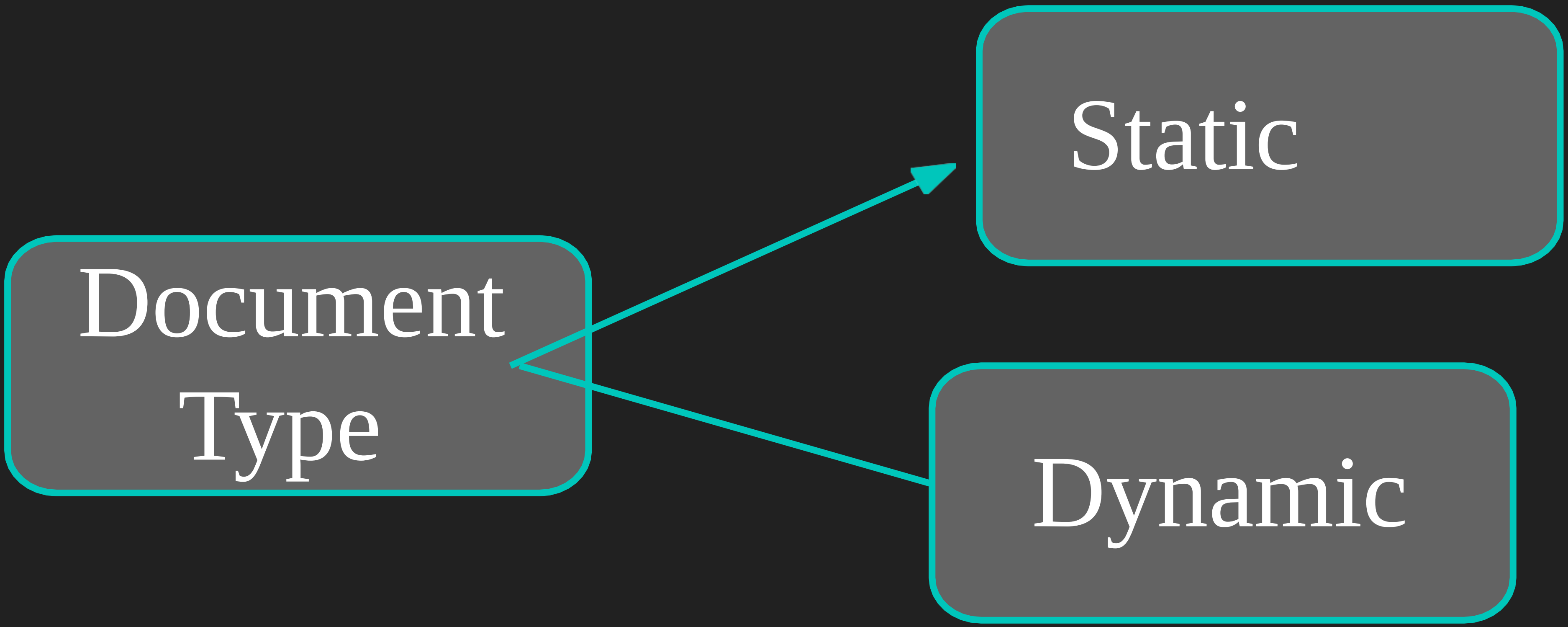
## Document Request



A request and response in a Rails application

https://oseven.in/files/58ce5021925bd.pdf

## Document Request



## Document Request - Static

**Static Documents: Hello, World in Rails:**

- This section describes how to build a Hello, World application in Rails.
- The purpose of such an exercise is to demonstrate the directory structure of the simplest possible Rails application,
- showing what files must be created and where they must reside in the directory structure.

# Document Request - Static

1. Create a new subdirectory.

   Example- We created a subdirectory named examples for the example applications.

2. Next, move to the examples directory and create a new Rails application named greet with the following command:

   ```
   > rails greet
   ```

3. Rails responds by creating more than 45 files in more than 30 directories.

   Most interesting of which at this point is app. The app directory has four subdirectories

# Document Request - Static

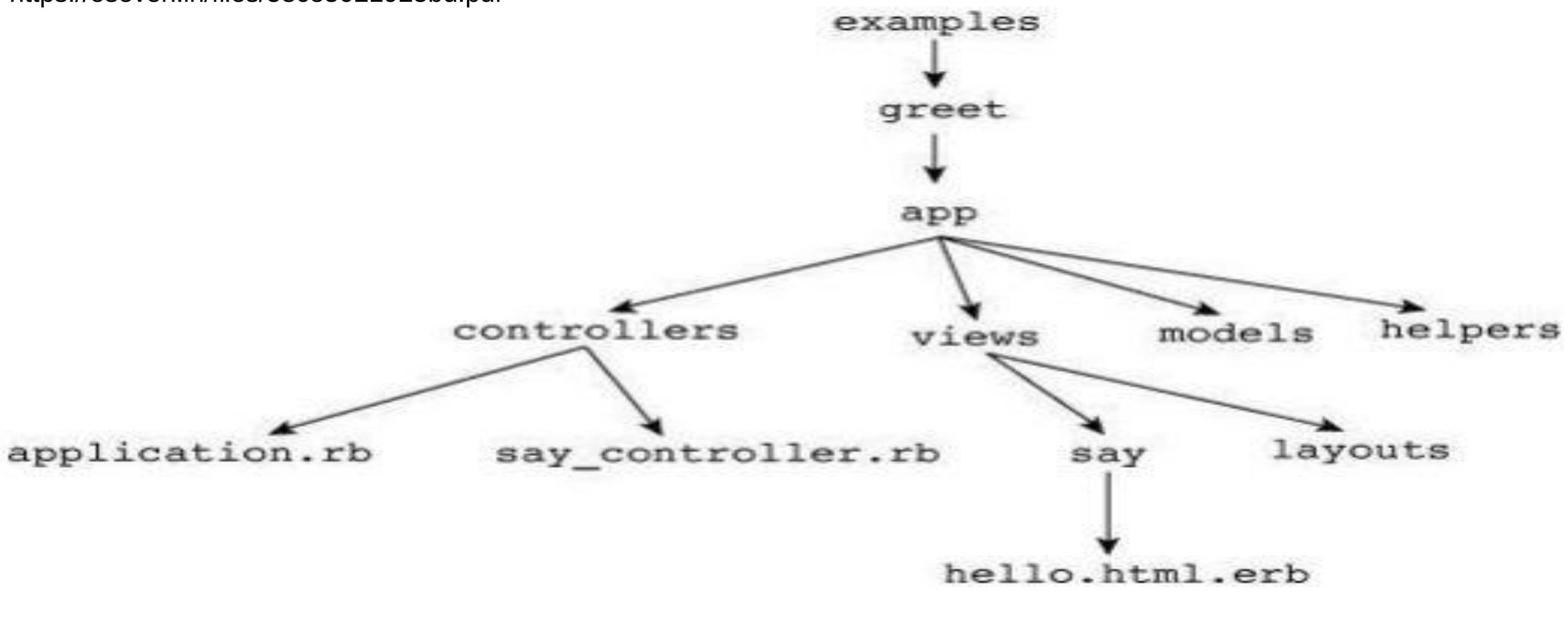
The **app directory** has four subdirectories:

1. models,
2. views,
3. controllers
4. helpers.

- The helpers subdirectory contains Rails-provided methods that aid in constructing applications.
- Most of the user code to support an application will reside in models, views, or controllers
- One of the directories created by the rails command is script, which has several important Ruby scripts that perform services.

## Directory Structure Created

Source: https://oseven.in/files/58ce5021925bd.pdf



Directory structure for the greet application

## Document Request - Static

- The class SayController inherits from ApplicationController, which

  is in the file application.rb.
- The class ApplicationController is a subclass of ActionController, which

  defines the basic functionality of a controller.
- Note that SayController produces the response to requests, so a method must be added to it.

## Document Request - Static

- Next, build the response document, or view file, which will be a simple XHTML file to produce the greeting.
- The following is the template for the greet application:

```
<html>
<body>
<h1> Welcome, to Rail </h1>
</body>
</html>
```

- The extension on this file's name is .html.erb because the file stores an HTML document, but it may include  embedded Ruby code

Document Request - Static

- Next, build the response document, or view file, which will be a simple XHTML file to produce the greeting.
- The following is the template for the greet application:

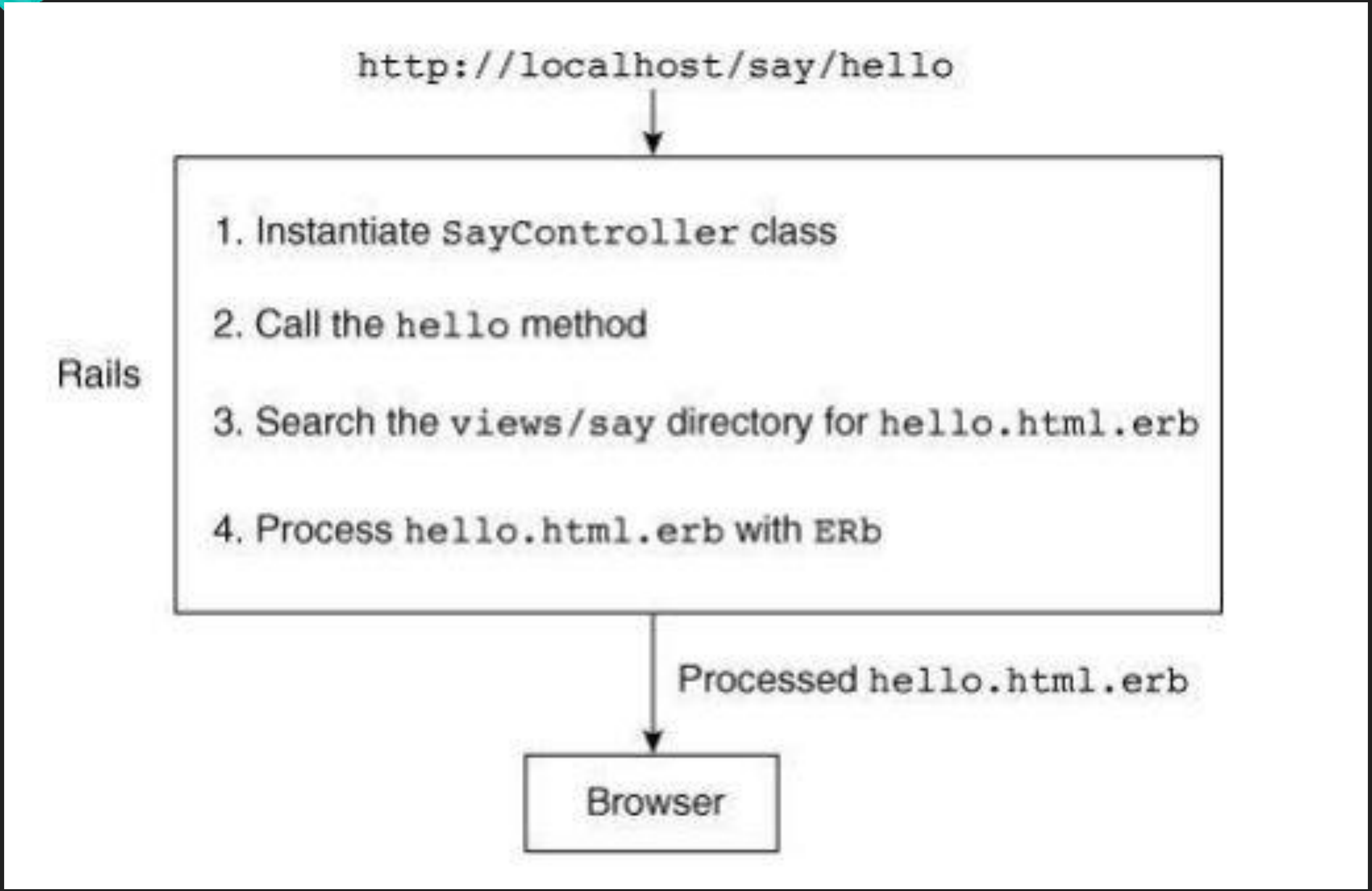
<html>
<body>
<h1> Welcome, to Rail </ha>
</body>
</html>

- The extension on this file's name is .html.erb because the file stores an HTML document, but it may include embedded Ruby code

## How Rails reacts to a request for a static document

1. First, the name of the controller is extracted from the URL.
2. Next, an instance of the controller class —in our example, SayController—is created.
3. The name of the action is then extracted from the URL—in our example, hello.
4. Then Rails searches for a template with the same name as the action method in the subdirectory with the same name as the controller in the app/views directory.
5. Next, the template file is given to ERb to interpret any Ruby code that is embedded in the template.
6. Finally, the template file is returned to the requesting browser, which displays it.

## how Rails reacts to a request for a static document



http://localhost/say/hello
Rails
1. Instantiate SayController class
2. Call the hello method
3. Search the views/say directory for hello.html.erb
4. Process hello.html.erb with ERb
Processed hello.html.erb
Browser

## Document Request - Dynamic document

- Dynamic documents can be constructed in Rails by embedding Ruby code in a template file.
- As an example of a dynamic document, to display the current date and time on the server
- Ruby code is embedded in a template file by placing it between the <% and %> markers.

# Document Request - Dynamic document

- The date can be obtained by calling Ruby’s Time.now method,
- Which returns the current day of the week, month, day of the month, time, time zone, and year, as a string.
- So, we can put the date in the response template with
- `<p> It is now <%= Time.now %> </p>`

- The date can be obtained by calling Ruby's Time.now method,
- Which returns the current day of the week, month, day of the month, time, time zone, and year, as a string.
- So, we can put the date in the response template with
- <p> It is now <%= Time.now %> </p>

## Rails Layout

## Rails Layouts

- In Rails, layouts are pieces that fit together (for example header, footer, menus, etc) to make a complete view.
- An application may have as many layouts as you want.
- Rails automatically pair up layouts with respective controllers having same name.
- Rails layouts basically work on Don't Repeat Yourself principle (DRY).
- In Rails, layouts are enabled by default.
- Whenever you generate a new Rails application, a layout is automatically generated for you in app/views/layouts.
- First we need to define a layout template and then define the path for controller to know that layout exists.

## Rails Layouts

### Creating Responses

There are three ways to create an HTTP response from the controller's point of view:

1. **Call render** to create a full response to send back to the browser
2. **Call redirect_to** to send an HTTP redirect status code to the browser
3. **Call head** to create a response to end back to the browser

## Rails Layouts

- Importance of yield statement
- The yield statement in Rails decides where to render the content for the action in layout.
- If there is no yield statement in the layout, the layout file itself will be rendered but  additional content into the action templates will not be correctly placed within the layout.
- Hence, a yield statement is necessary to add in a layout file.
- <%= yield %>

## Rails Layouts

### Finding correct layout

- Rails searches for the layout with same name in the app/layouts directory as the controllers name.
- For example, if you have a controller called abController, then rails

  will search for  layouts/ab.html.erb layout.
- If no layout with the same name is present, then it will use the default layout

# Rails Layouts

## Relation between Rails Layouts and Templates

When a request is made in an application, following process occur:

1. Rails find a template for corresponding action to render method in your controllers action.
2. Then finds correct layout to use.
3. It uses action template to generate a content specific to the action.
4. Finally it looks for the layout's yield statement and insert action's template here

Relation between Rails Layouts and Templates

When a request is made in an application, following process occur:

1. Rails find a template for corresponding action to render method in your controllers action.
2. Then finds correct layout to use.
3. It uses action template to generate a content specific to the action.
4. Finally it looks for the layout's yield statement and insert action's template here

## Rails with Ajax

## Rails with Ajax

Ajax stands for Asynchronous JavaScript and XML.

Ajax is not a single technology; it is a suite of several technologies as following –

- XHTML for the markup of web pages
- CSS for the styling
- Dynamic display and interaction using the DOM
- Data manipulation and interchange using XML
- Data retrieval using XMLHttpRequest
- JavaScript as the glue that meshes all this together

## Rails with Ajax

- Ajax enables you to retrieve data for a web page without having to refresh the contents of the entire page.
- In the basic web architecture, the user clicks a link or submits a form.
- The form is submitted to the server, which then sends back a response.
- The response is then displayed for the user on a new page.

# Rails with Ajax

## How Rails Implements Ajax

Once the browser has rendered and displayed the initial web page, different user actions  cause it to display a new web page or trigger an Ajax operation –

1. Some trigger fires
2. The web client calls the server
3. The server does processing
4. The client receives the response

## Rails with Ajax

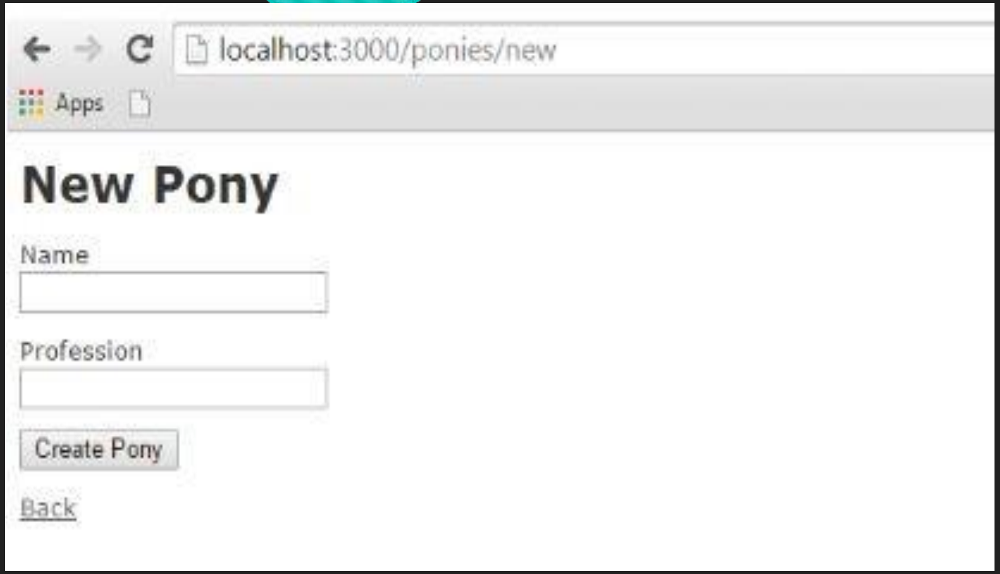
AJAX Example:

In this example, we will provide, list, show and create operations on ponies table.

1. Creating An Application using command:- rails new ponies
2. Call the app directory using with cd command.
3. Migrate the database as follows command: rake db:migrate
4. Now Run the Rails application as follows command: rails s
5. Now open the web browser and call a url as http://localhost:3000/ponies/new
6. Creating an Ajax-Now open app/views/ponies/index.html.erb with suitable text editors.

   Update your destroy line with :remote => true, :class => 'delete_pony'

## Rails with Ajax- Output After running Ajax in Web Browser


localhost:3000/ponies/new

Apps

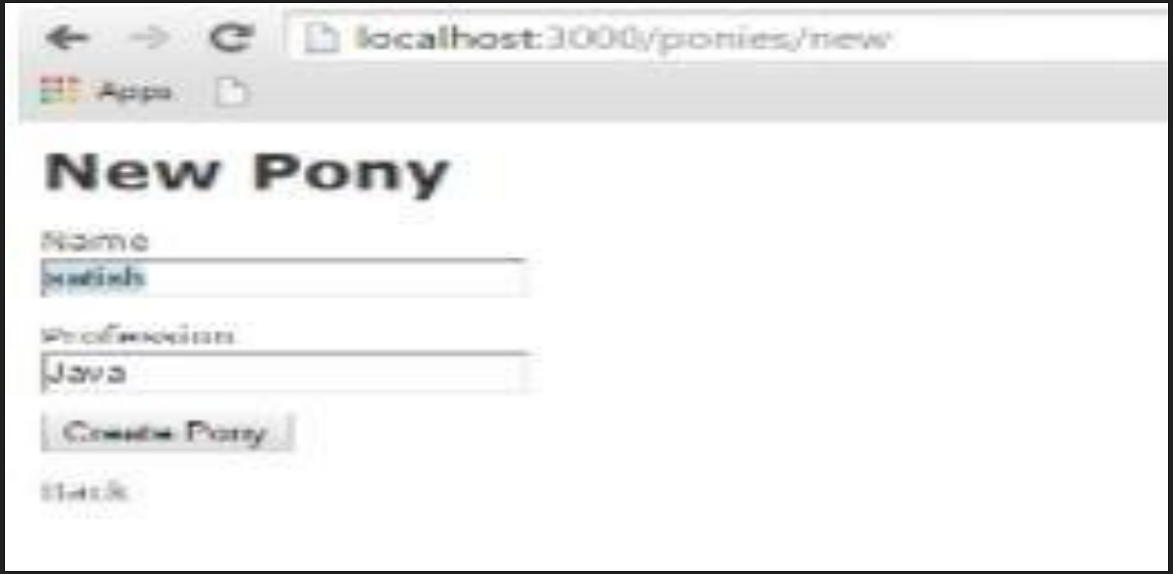
New Pony

Name

Profession

Create Pony

Back


localhost:3000/ponies/new

Apps

New Pony

Name

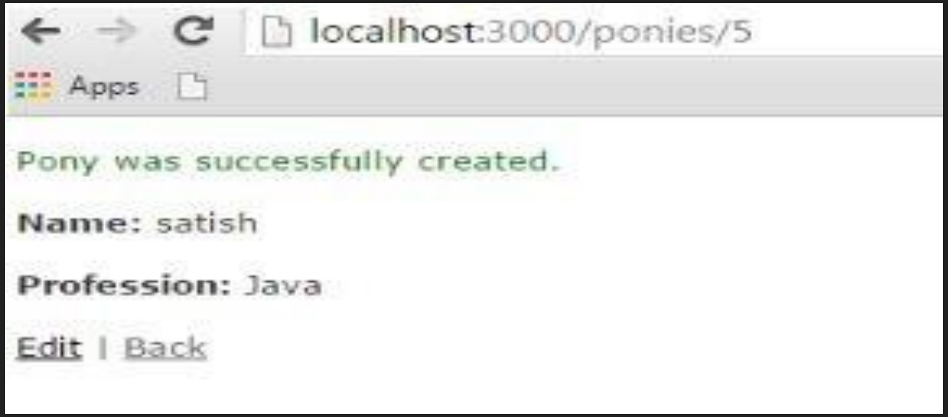
satish

Profession

Java

Create Pony

Back


localhost:3000/ponies/5

Apps

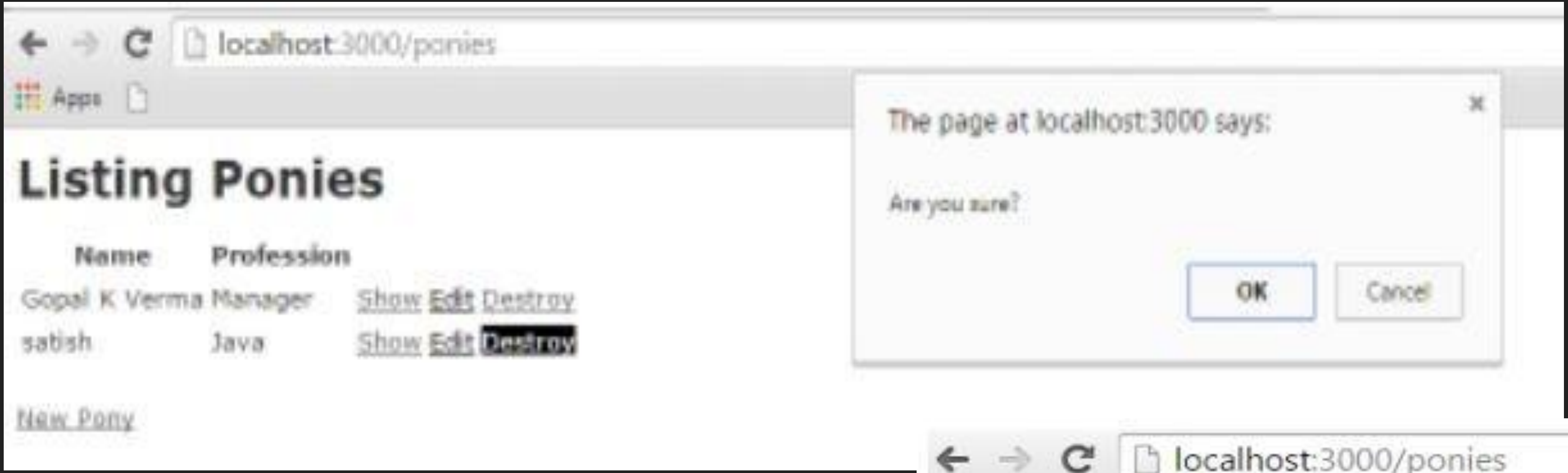
Pony was successfully created.

**Name:** satish

**Profession:** Java

Edit | Back

## Rails with Ajax- Output After running Ajax in Web Browser



localhost:3000/ponies

Apps

# Listing Ponies

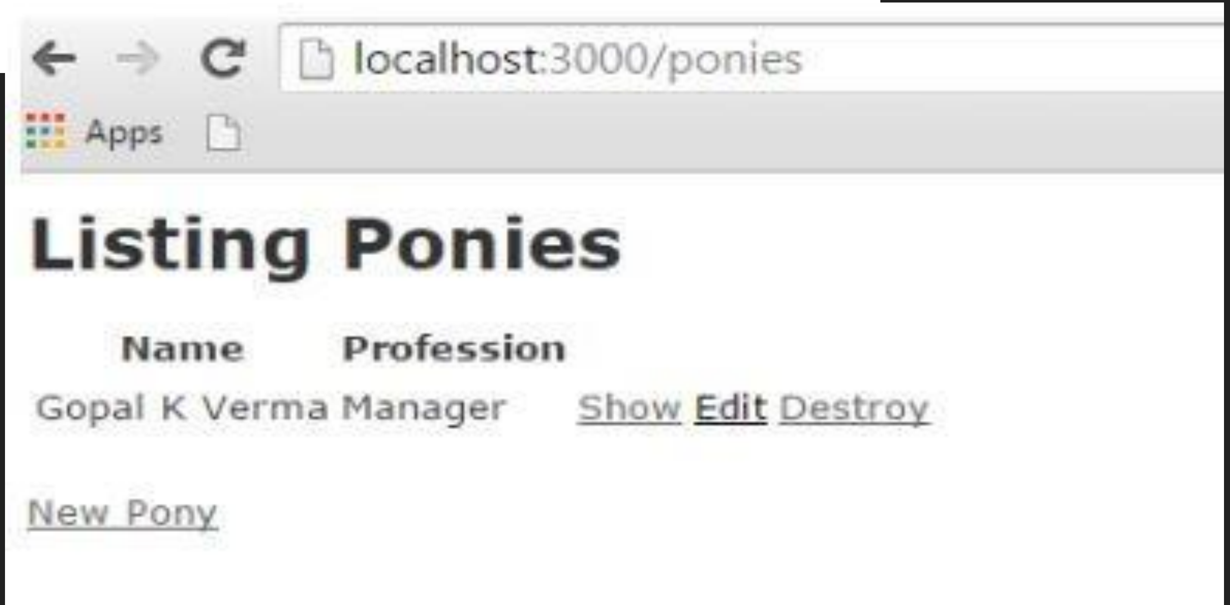
| Name | Profession | |
|---|---|---|
| Gopal K Verma | Manager | Show Edit Destroy |
| satish | Java | Show Edit Destroy |

New Pony

The page at localhost:3000 says:

Are you sure?

OK Cancel



localhost:3000/ponies

Apps

# Listing Ponies

| Name | Profession | |
|---|---|---|
| Gopal K Verma | Manager | Show Edit Destroy |

New Pony

## EJB- Enterprise JavaBeans

## Outline of EJB

EJB: Overview

types of EJB,

Architecture

## EJB: Overview

- EJB stands for **Enterprise Java Beans**.
- EJB is an essential part of a J2EE platform.
- EJB provides an architecture to develop and deploy component based enterprise applications considering  robustness, high scalability, and high performance.

- **When use Enterprise Java Bean?**
  - **Application needs Remote Access**. In other words, it is distributed.
  - **Application needs to be scalable**. EJB applications supports load balancing, clustering and fail-over.
  - **Application needs encapsulated business logic**. EJB application is separated from presentation and persistent layer.

## EJB: Overview

- EJB is an acronym for Enterprise Java Bean. It is a specification provided by Sun Microsystems to develop secured, robust and scalable distributed applications.

- To run EJB application, you need an application server (EJB Container) such as Jboss, Glassfish, Weblogic, Websphere etc. It performs**: (Function of EJB)**

1. Life cycle management
2. Security
3. Transaction management
4. Object pooling

- EJB application is deployed on the server, so it is called server side component also.
- EJB is like COM (Component Object Model) provided by Microsoft. But, it is different from Java Bean, RMI and Web Services.

## TYPES OF EJB

There are 3 types of enterprise bean in java.

**1. Session Bean**

Session bean contains business logic that can be invoked by local, remote or webservice client.

**2. Message Driven Bean**

Like Session Bean, it contains the business logic but it is invoked by passing message.

**3. Entity Bean**

It encapsulates the state that can be persisted in the database. It is deprecated. Now, it is replaced with JPA (Java Persistent API).

## Advantages of EJB

**Advantages of EJB :**

1. EJB is an API, hence the application's build on EJB can run on Java EE web application server.
2. The EJB developer focuses on solving business problems and business logic.
3. Java beans are portable components that help the JAVA application assembler to formulate new applications for the already existing JavaBeans.
4. EJB container helps in providing system-level services to enterprise Java beans.
5. EJB contains business logic hence the front end developer can focus on the presentation of the client interface.
6. It provides simplified development of large scale enterprise level application.

## EJB: Overview

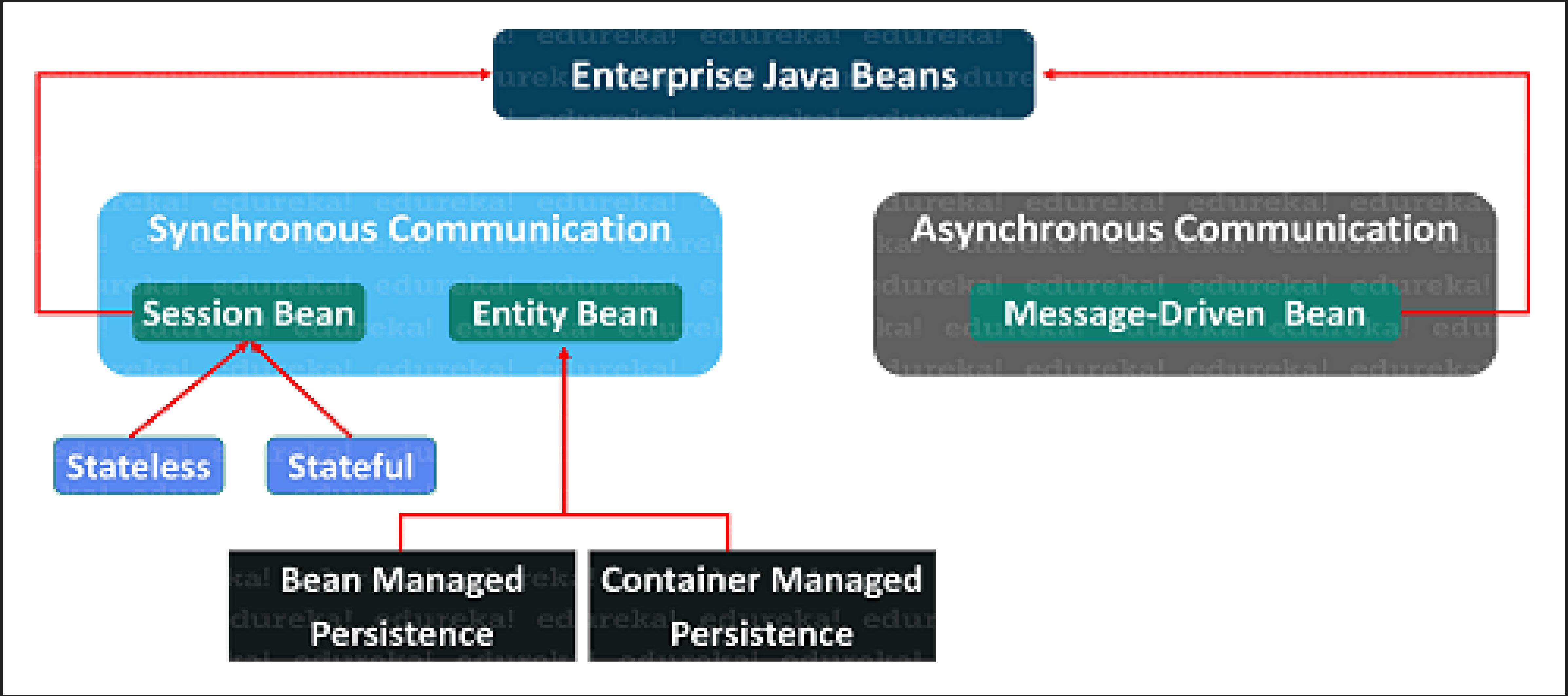
**Disadvantages of EJB**

- Requires application server
- Requires only java client. For other language client, you need to go for web service.
- Complex to understand and develop ejb applications.
- It takes time for development.

## Outline of EJB

- EJB: Overview
- **types of EJB,**
- Architecture

# Types of EJB



Enterprise Java Beans
Synchronous Communication
Session Bean
Entity Bean
Asynchronous Communication
Message-Driven Bean
Stateless
Stateful
Bean Managed Persistence
Container Managed Persistence

## Types of EJB

| S. No | Type & Description |
|---|---|
| 1 | **Session Bean**<br>Session bean stores data of a particular user for a single session. It can be **stateful** or **stateless**.<br>It is less resource intensive as compared to entity bean.<br>Session bean gets destroyed as soon as user session terminates. |
| 2 | **Entity Bean**<br>**Entity beans** represent persistent data storage.<br>User data can be saved to database via entity beans and later on can be retrieved from the database in the entity bean. |
| 3 | **Message Driven Bean**<br>**Message driven beans** are used in context of JMS (Java Messaging Service).<br>Message Driven Beans can consumes JMS messages from external entities and act accordingly. |

## Session beans

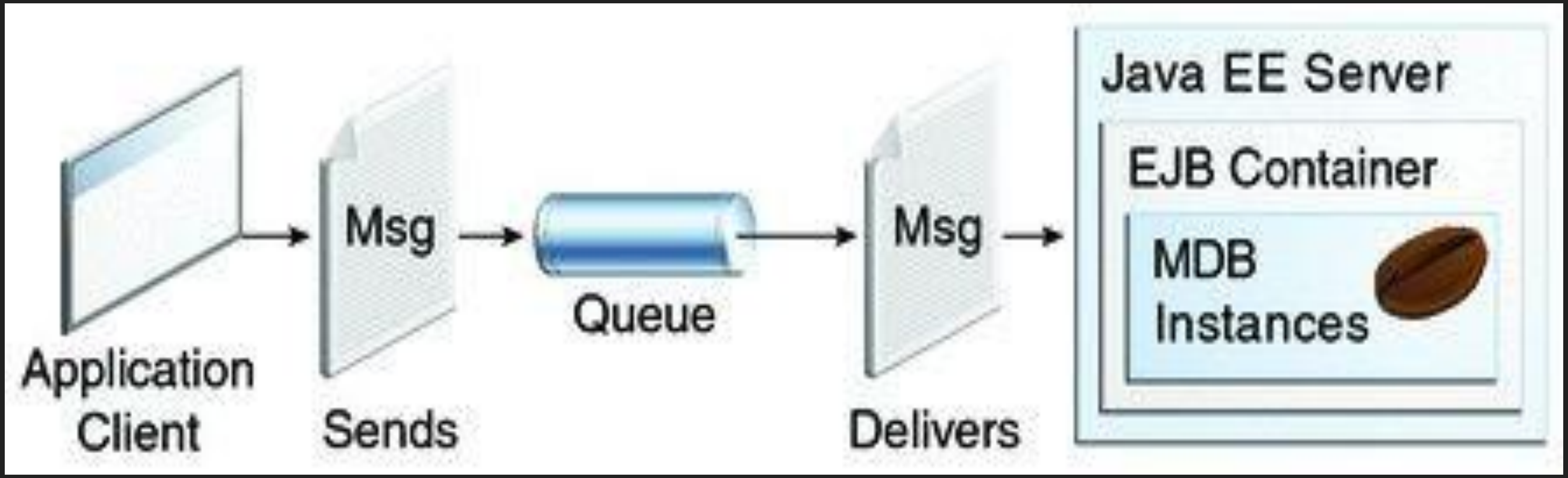
- **Stateless Session bean** *is a business object that represents business logic* only. It doesn't have state (data).
- In other words, *conversational state* between multiple method calls is not maintained by the container in case of stateless session bean.
- The stateless bean objects are pooled by the EJB container to service the request on demand.
- It can be accessed by one client at a time. In case of concurrent access, EJB container routes each request to different instance.
- **Annotations used in Stateless Session Bean**
- @Stateless

# Session beans

- **Stateful Session Bean**
- **Stateful Session bean** *is a business object that represents business logic* like stateless session bean. But, it maintains state (data).
- In other words, *conversational state* between multiple method calls is maintained by the container in stateful session bean.
- **Annotations used in Stateful Session Bean**
- @Stateful

## Message Driven Bean

- **Message Driven Bean**
- A message driven bean (MDB) is a bean that contains business logic. But, it is invoked by passing the message. So, it is like  JMS Receiver.
- MDB asynchronously receives the message and processes it.
- A message driven bean receives message from queue.
- A message driven bean is like stateless session bean that encapsulates the business logic and doesn't maintain state.



## Entity beans

- **Container-managed persistence—**
  - The EJB container manages data by saving it to a designated resource, which is normally a database.
  - For this to occur, you must define the data that the container is to manage within the deployment  descriptors.
  - The container manages the data by saving it to the database.

- **Bean-managed persistence—**
  - The bean implementation manages the data within callback methods.
  - All the logic for storing data to your persistent storage must be included in the ejbStore method and  reloaded from your storage in the ejbLoad method.
  - The container invokes these methods when necessary.

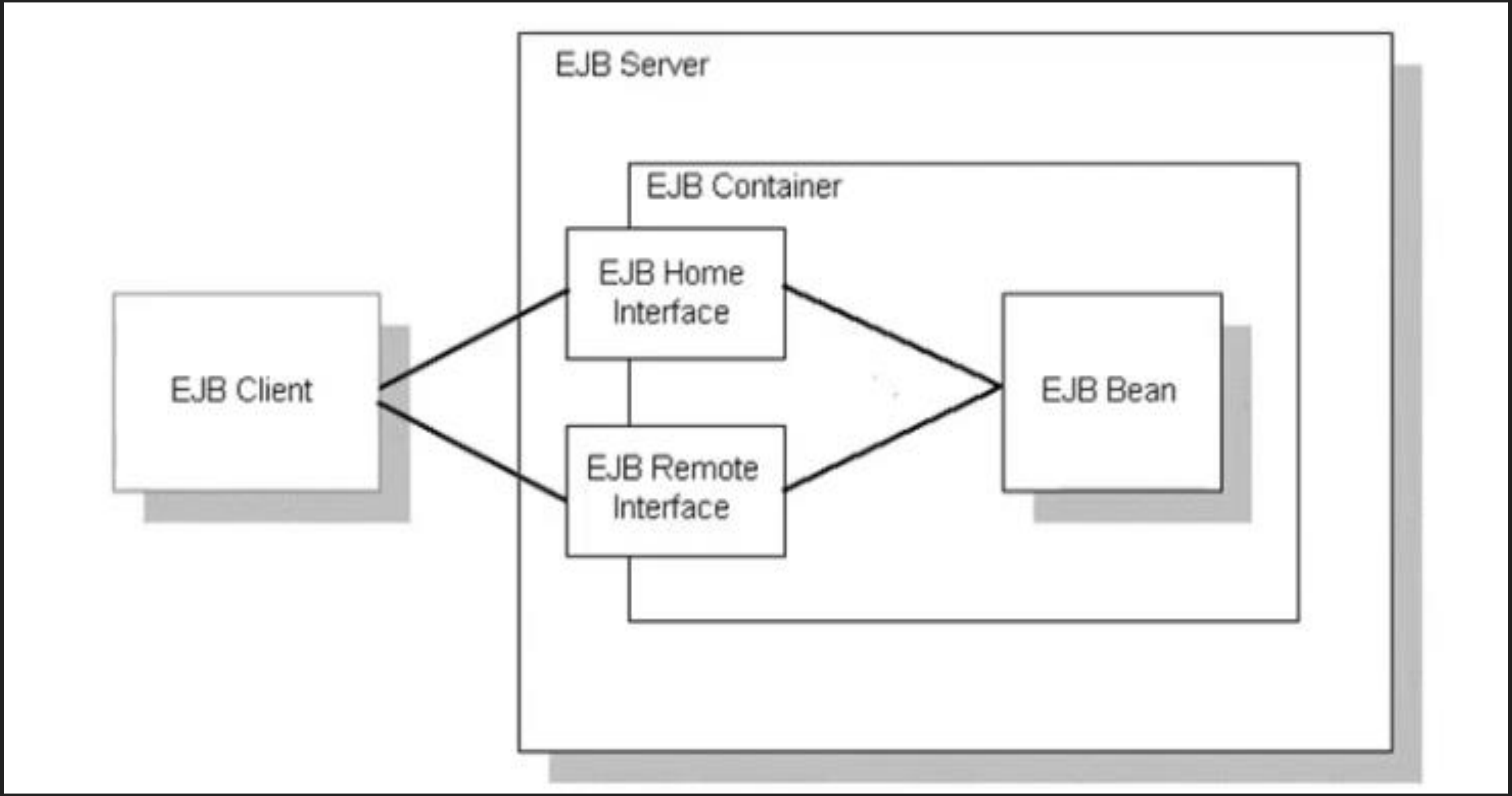
## Outline of EJB

EJB: Overview

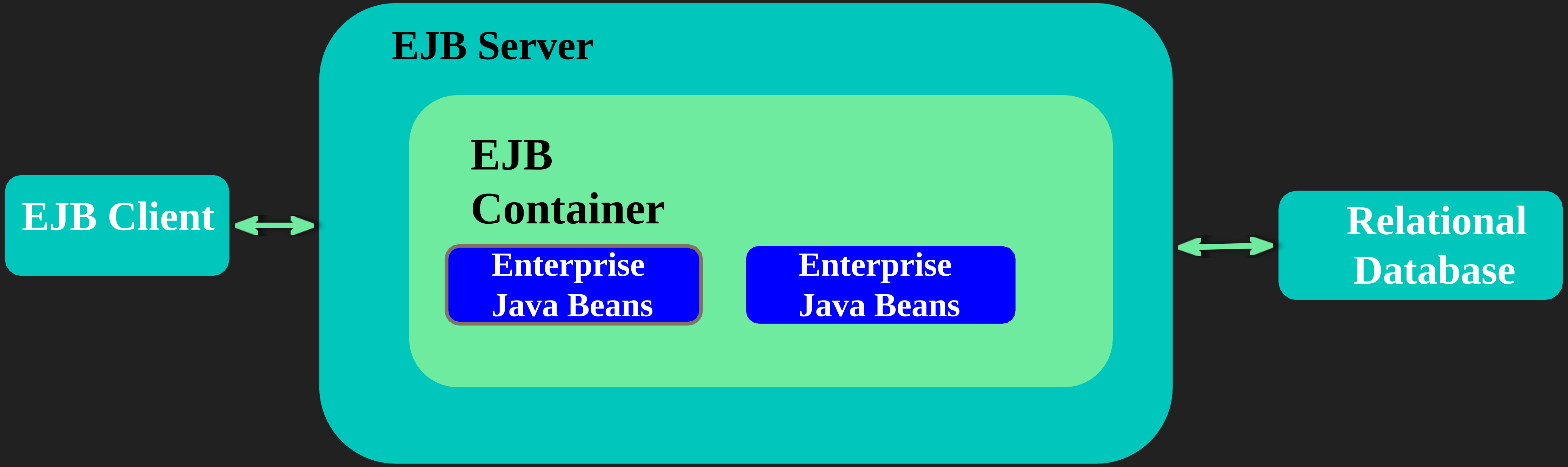
types of EJB,

Architecture

## EJB Architecture



## EJB Architecture



## EJB Architecture- Components

- EJB server(s)
- EJB container(s)
- Enterprise Beans
- EJB clients
- other auxillary systems (e.g. Java Naming and Directory Interface (JNDI) server, Java Transaction  Service (JTS), ...).

## EJB Architecture- Components

**1. Application Server**

This server is the outermost layer in the architecture and it holds the container. It provides the necessary environment to execute the applications developed using the beans. Web-logic, Web-sphere, JBoss, Tomcat, Wildfly, and Glass-finish are some of the application servers popular in the market.

**The main functionality of Application servers is**

a. Manage Interfaces,

b. Execution of the processes,

c. Connecting to the database,

d. Manage other resources.

# EJB Architecture- Components

**2. Container**

Container is the second outermost layer in EJB structure and it provides the following supporting services to the enterprise beans housed in it.

- Transactional service like Registering the objects, assign remote interface, purge the instances
- Monitoring the activities of the objects and coordinating distributed components
- Security service and Pooling of resources
- Manages Life-cycle of beans and its concurrency
- Provide a platform for the developer to concentrate on business logic

## EJB Architecture- Components

**3. Beans**

Enterprise java bean installed in the container is something like Plain old java object (POJO) and they are registered to the container. Beans provide business logic for developing robust, secured and large scale business applications.

- Interacts with EJB container
- EJB Client is a local program which can call & operate remote beans.
- Client locates an Enterprise Java Beans through JNDI(Java Naming Directory  Services)

# Java beans Vs Enterprise Java beans

| Sr. No | Java Beans | Enterprise Java beans |
|---|---|---|
| 1 | Visible | Not visible, Runs as remote |
| 2 | Local, Single process and runs in the Client machine<br>Server-side beans are possible not preferred | Executed in the server-side |
| 3 | Applets and applications are built using generic components created by Java Beans. | It uses component technology but it is not built or extended over Beans. |
| 4 | It has an external interface to interpret bean’s functionality | Works seamlessly with IDE or external builder tool. |
| 5 | Have Property editors, Customizers and Beaninfo classes | No such concepts other than what is provided by deployment descriptor. |
| 6 | No types, No support for transactions | Have three types and transactions are supported |

## References

- https://www.javatpoint.com/what-is-ruby
- https://www.tutorialspoint.com/ruby/ruby_overview.htm#:~:text=Ruby%20is%20a%20server%2Dside,learn%20very%20quickly%20and%20easily.
- https://www.javatpoint.com/ruby-data-types
- https://www.geeksforgeeks.org/ruby-decision-making-if-if-else-if-else-if-ternary-set-1/
- https://www.javatpoint.com/ruby-blocks
- https://www.geeksforgeeks.org/ruby-types-of-iterators/
- https://www.tutorialspoint.com/ruby-on-rails/rails-introduction.htm
- https://www.geeksforgeeks.org/ruby-on-rails-introduction/
- https://www.javatpoint.com/ruby-on-rails-layout#:~:text=In%20Rails%2C%20layouts%20are%20pieces,respective%20controllers%20having%20same%20name.
- https://www.javatpoint.com/what-is-ejb
- https://www.edureka.co/blog/ejb-in-java/

# Thank You

gharu.anand@gmail.com
Blog : anandgharu.wordpress.com

**Prof. Gharu Anand N.**