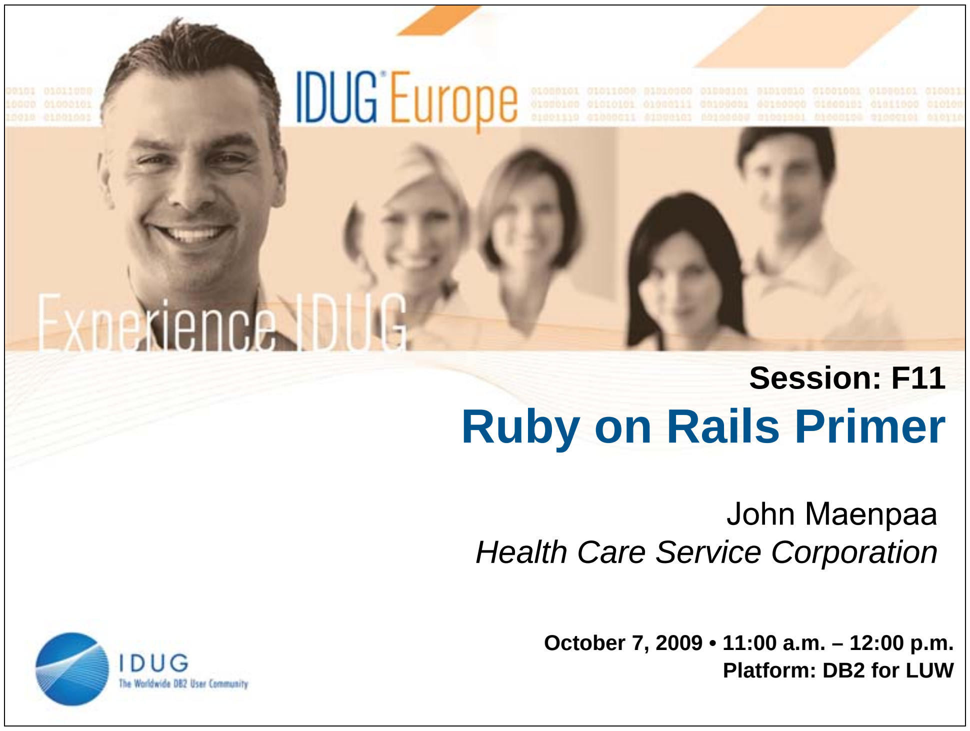


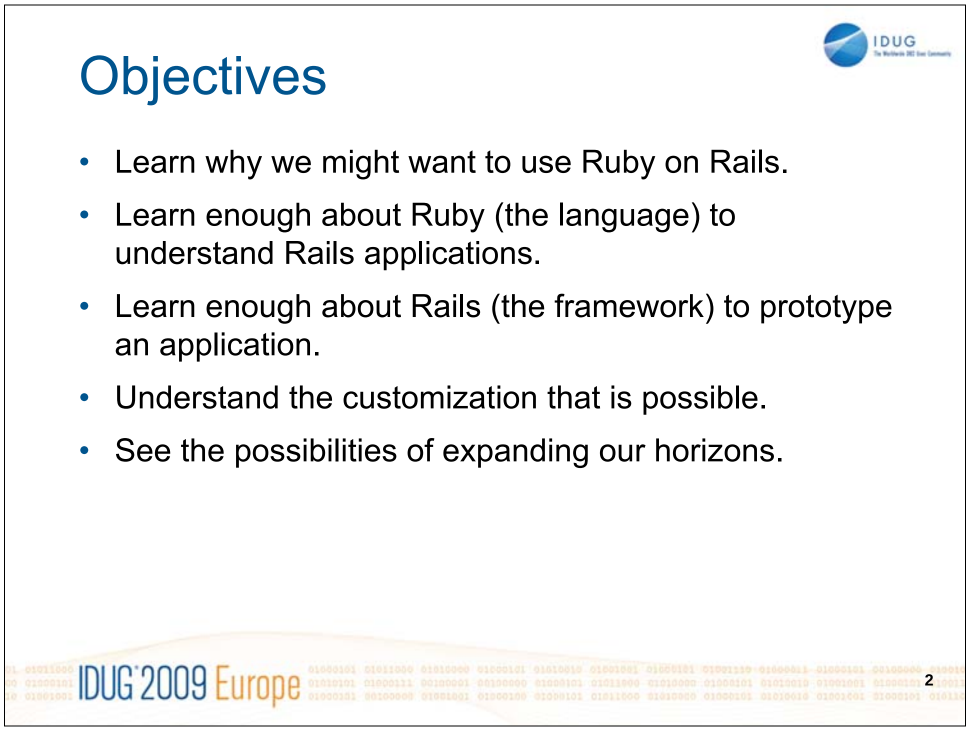
Session: F11

Ruby on Rails Primer

John Maenpaa
Health Care Service Corporation

October 7, 2009 • 11:00 a.m. – 12:00 p.m.
Platform: DB2 for LUW

Abstract:
Ruby on Rails is an up and coming development framework for web-based applications. It excels in producing applications where standard database CRUD activities are the norm. This presentation will provide a primer on Ruby and Rails, including an interactive demonstration on building a prototype application. The Rails community has embraced REST, impacting design possibilities, so we'll need to cover that too.



Objectives

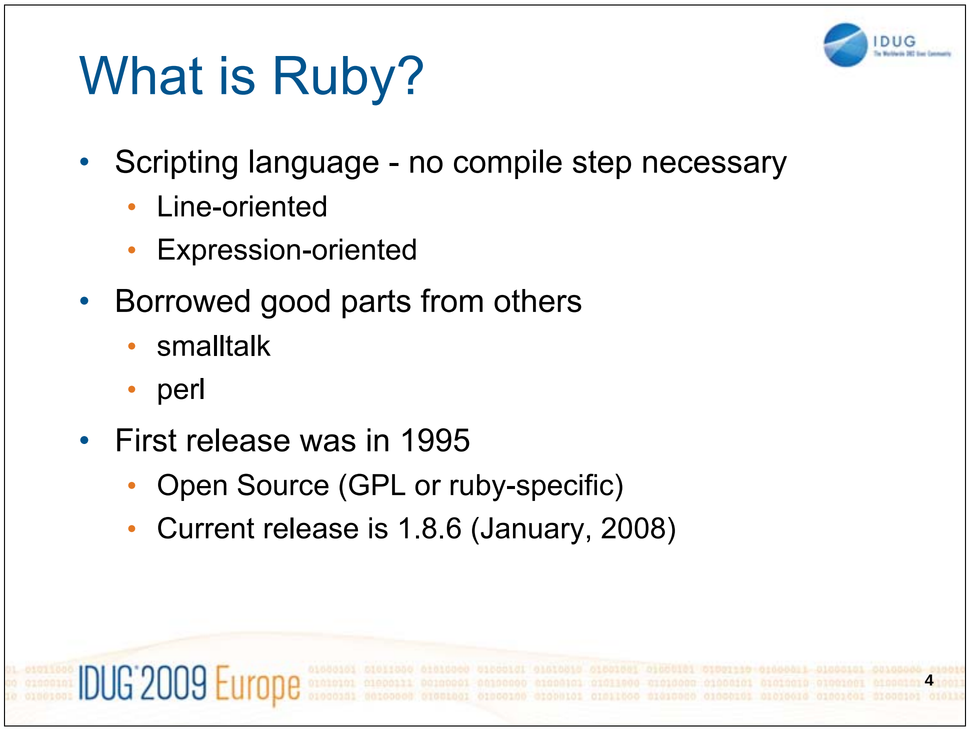
- Learn why we might want to use Ruby on Rails.
- Learn enough about Ruby (the language) to understand Rails applications.
- Learn enough about Rails (the framework) to prototype an application.
- Understand the customization that is possible.
- See the possibilities of expanding our horizons.

Abstract:
Ruby on Rails is an up and coming development framework for web-based applications. It excels in producing applications where standard database CRUD activities are the norm. This presentation will provide a primer on Ruby and Rails, including an interactive demonstration on building a prototype application. The Rails community has embraced REST, impacting design possibilities, so we'll need to cover that too.

Agenda

- Overview
  - What is Ruby?
  - What is Rails?
- Ruby: the Language
  - Enough Ruby to code a Rails application
- Rails: the Framework
  - Enough Rails to get you going

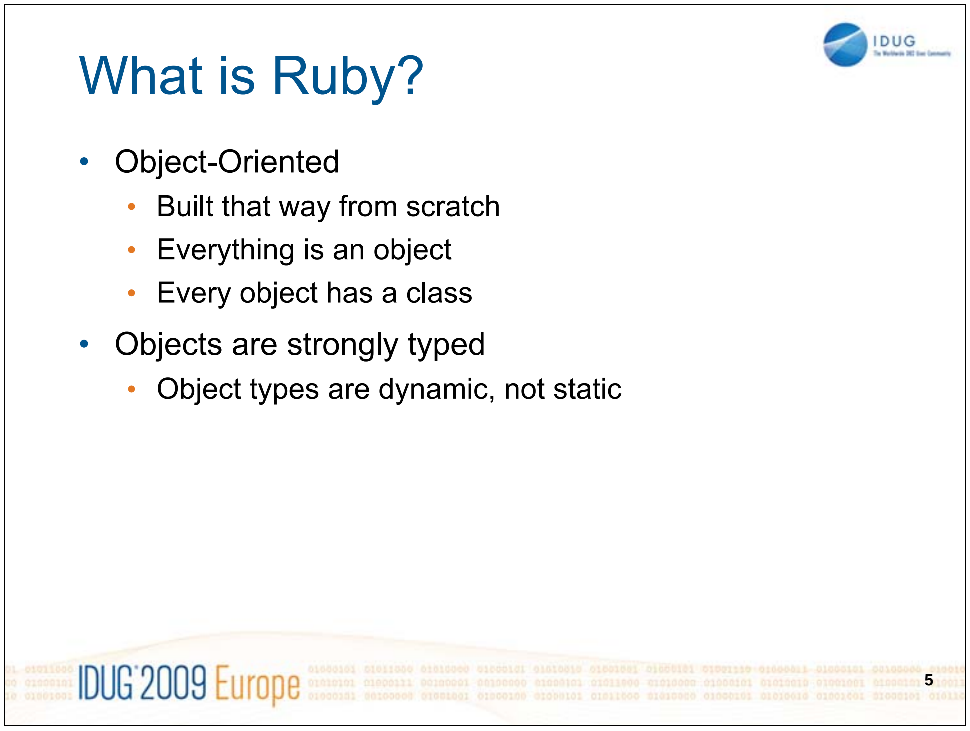
Updated for Rails 2.2



What is Ruby?

- Scripting language - no compile step necessary
  - Line-oriented
  - Expression-oriented
- Borrowed good parts from others
  - smalltalk
  - perl
- First release was in 1995
  - Open Source (GPL or ruby-specific)
  - Current release is 1.8.6 (January, 2008)

Ruby was created by Yukihiro “matz” Matsumoto. He wanted a language that was object-oriented yet easy to use. He took some of the best features from perl and smalltalk and combined them to create ruby.



What is Ruby?

- Object-Oriented
  - Built that way from scratch
  - Everything is an object
  - Every object has a class
- Objects are strongly typed
  - Object types are dynamic, not static

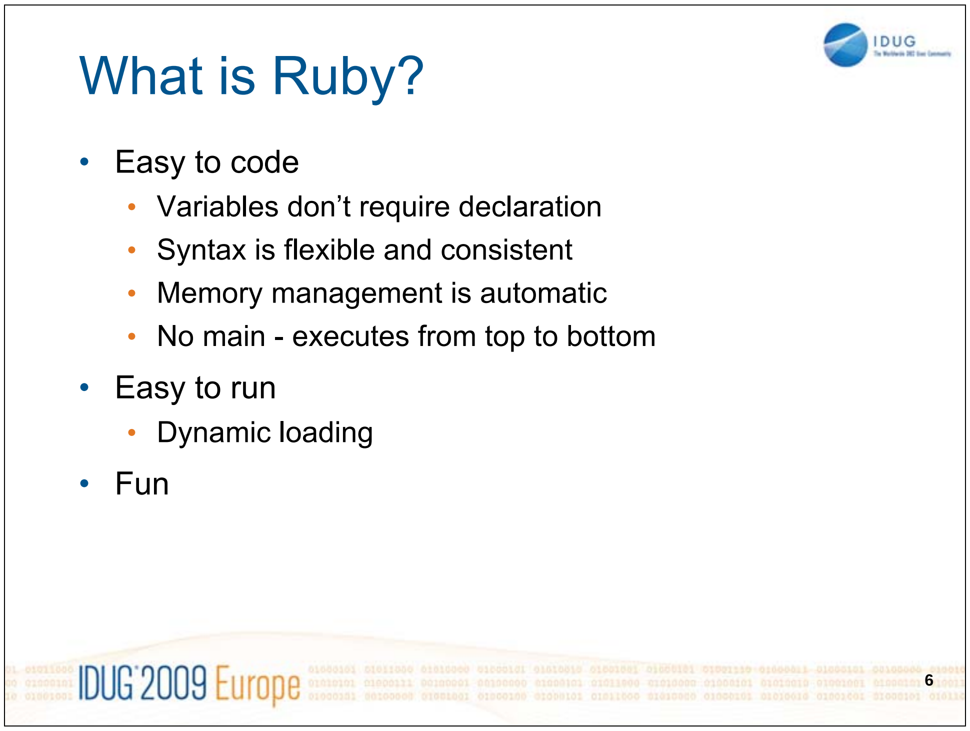
IDUG 2009 Europe

5

Matz developed ruby to be object-oriented from scratch. Everything in ruby is an object and every object has a class. This has distinct advantages in that methods for base objects are available for all of their descendants.

Even integers are objects in ruby, which makes for some convenient syntax where we can invoke methods that are defined in the integer class.

Every object is strongly typed, but the types are assigned dynamically at run time. This means that you can write routines that are intended to act on one class of object and work on other object types as well.



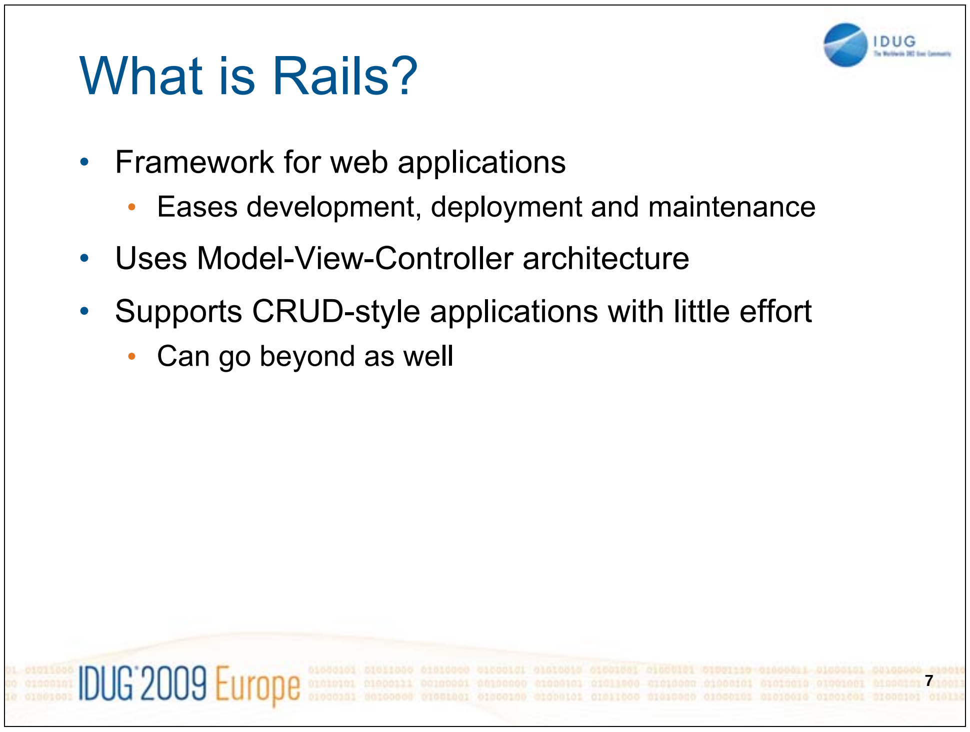
What is Ruby?

- Easy to code
  - Variables don't require declaration
  - Syntax is flexible and consistent
  - Memory management is automatic
  - No main - executes from top to bottom
- Easy to run
  - Dynamic loading
- Fun

IDUG 2009 Europe 6

Because the language borrowed elements from other languages, the syntax is both familiar and distinct. Like other scripting languages, there is no "main" routing like there is in C. Execution proceeds from top to bottom naturally with appropriate flow of control, of course. Ruby manages memory automatically, so there is no need to clean up after yourself.

Ruby is easy to run because it truly is a scripting language. There is no need for separate compile and link steps. You just point the ruby program at your script and it will run it. Classes and modules are dynamically loaded as they are required or used.

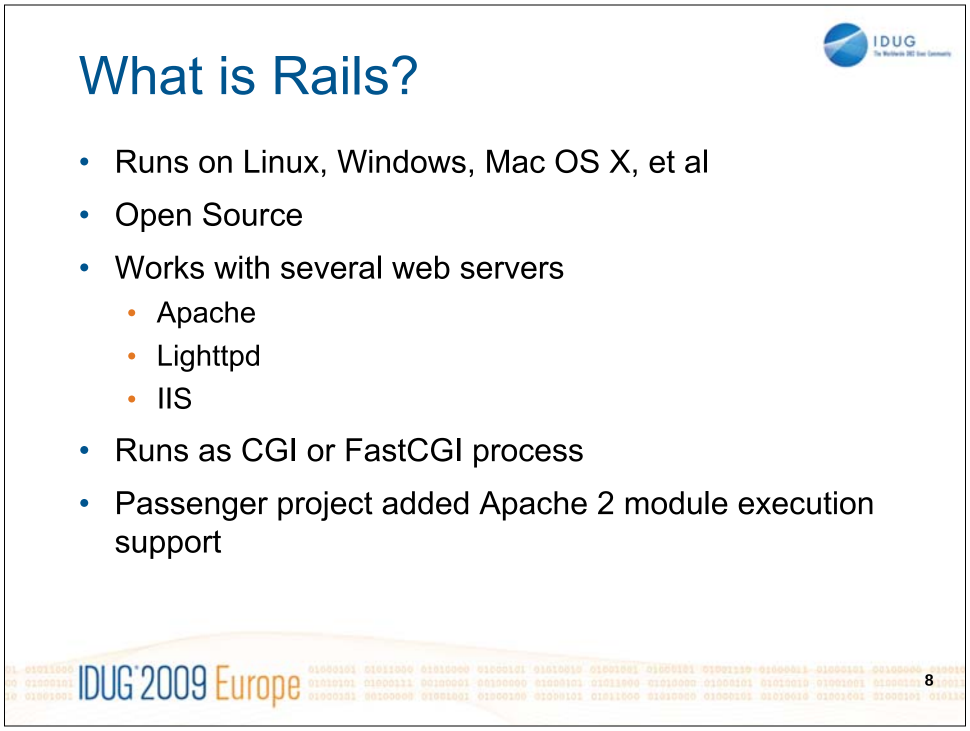


What is Rails?

- Framework for web applications
  - Eases development, deployment and maintenance
- Uses Model-View-Controller architecture
- Supports CRUD-style applications with little effort
  - Can go beyond as well

Rails is a framework built on top of ruby. It was designed by David Heinemeier Hansson and released in July, 2004. Unlike a lot of the frameworks for Java that force you to code lots of XML files that define the parts of your application, rails was developed to make setting up your web application easy. To avoid the need to develop support structures full of XML files, most of the configuration files for rails are implemented using ruby. This eases development and allows flexible deployment using a single source tree.

The rails framework implements a model-view-controller architecture where the data elements are separated from the controlling logic, which is also separated from the view. We'll cover this a little bit more when we dig into the details.



What is Rails?

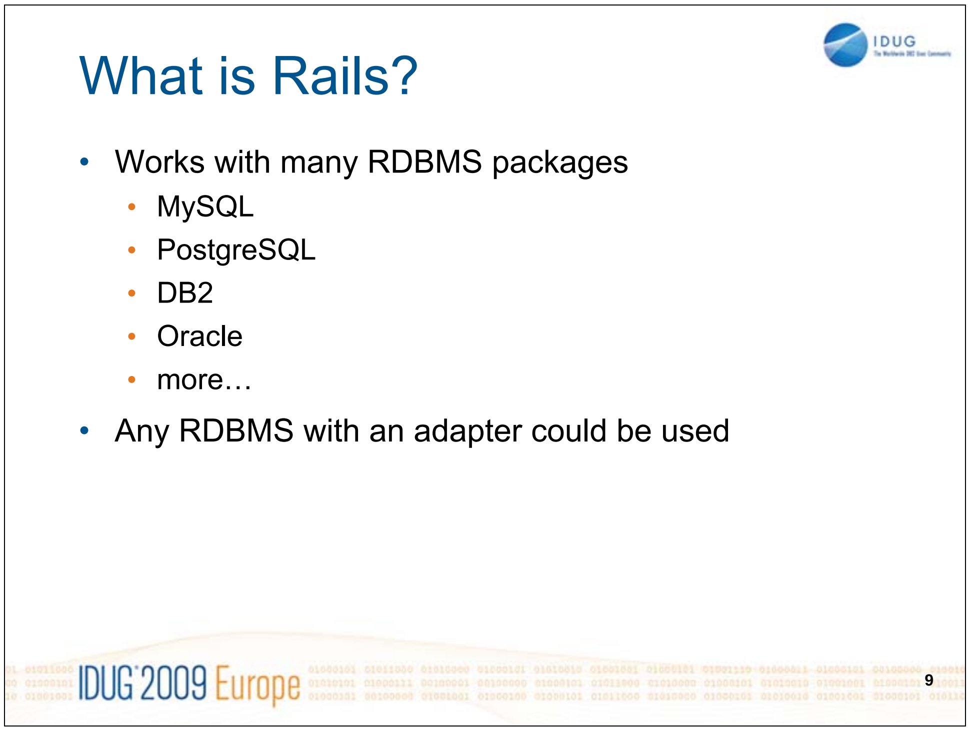
- Runs on Linux, Windows, Mac OS X, et al
- Open Source
- Works with several web servers
  - Apache
  - Lighttpd
  - IIS
- Runs as CGI or FastCGI process
- Passenger project added Apache 2 module execution support

IDUG 2009 Europe 8

Rails can execute on any platform where ruby has been ported. Because both ruby and rails are open source there is tremendous support for running them on most of the common open source platforms as well as Windows and Mac OS X.

Because it is a web framework, rails needs to work with a web server. In the agile programming tradition, rails works with many web servers. There are many to choose from and any web server that implements standards should work. Microsoft IIS has the most problems with rails but there are reports of some that have been able to make it work.

Rails normally runs as a CGI (or FastCGI) process executed by the web server. This primarily defines the interface between the web server and the running rails processes. Rails applications can also be run as back-end processes controlled outside of the web server.

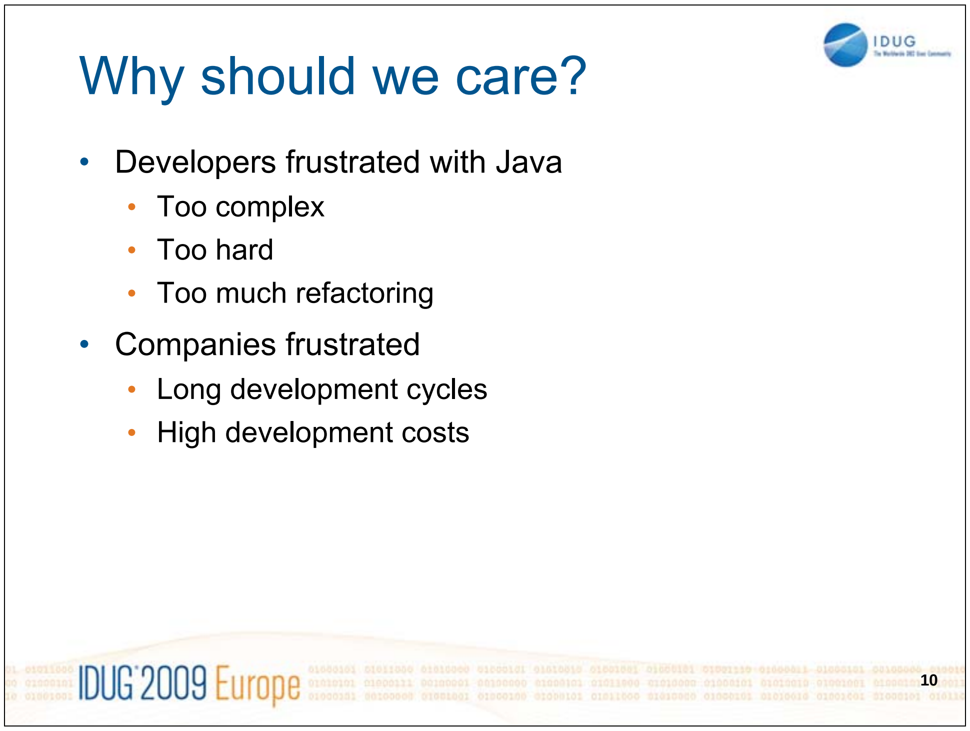


What is Rails?

- Works with many RDBMS packages
  - MySQL
  - PostgreSQL
  - DB2
  - Oracle
  - more…
- Any RDBMS with an adapter could be used

IDUG 2009 Europe

Just as it supports many operating systems and web servers, rails also works with a variety of relational DBMS packages. The most common are those that are available on the largest variety of platforms. MySQL and PostgreSQL are both quite popular in the rails community. The primary requirement for any given RDBMS is the existence and support of an interface module that supports the rails "model" methods and translates them into the RDBMS' specific CLI.



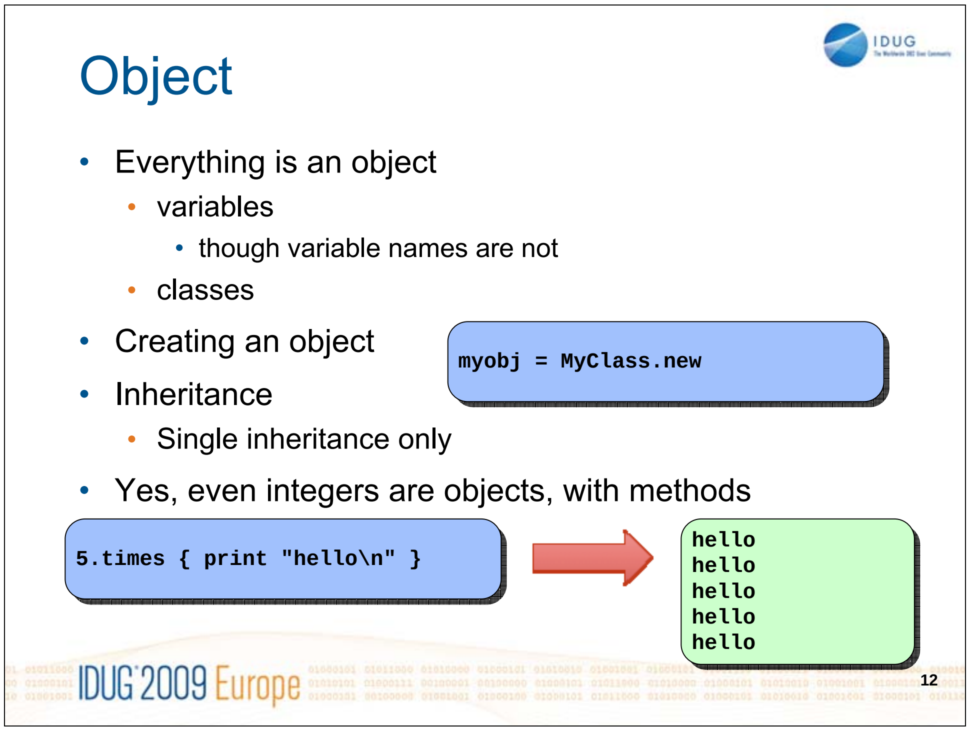
Why should we care?

- Developers frustrated with Java
  - Too complex
  - Too hard
  - Too much refactoring
- Companies frustrated
  - Long development cycles
  - High development costs

IDUG 2009 Europe

And finally, why should we care about ruby and rails? Have you noticed how the migration to Java has slowed development? It used to take a matter of days to develop CICS applications in COBOL. Now it takes months in Java to develop the equivalent web-based application. And that is with experienced Java programmers. Java itself isn't a complex language, but all of the classes that you need to support a web application are over engineered. It just takes way too much time to get the work done, and too long to learn the details of each of the classes. Have you heard "refactor" used as a dirty word by your business customers yet?

Ok, back to ruby.



Object

- Everything is an object
  - variables
    - though variable names are not
  - classes
- Creating an object
- Inheritance
  - Single inheritance only
- Yes, even integers are objects, with methods

```
myobj = MyClass.new
```

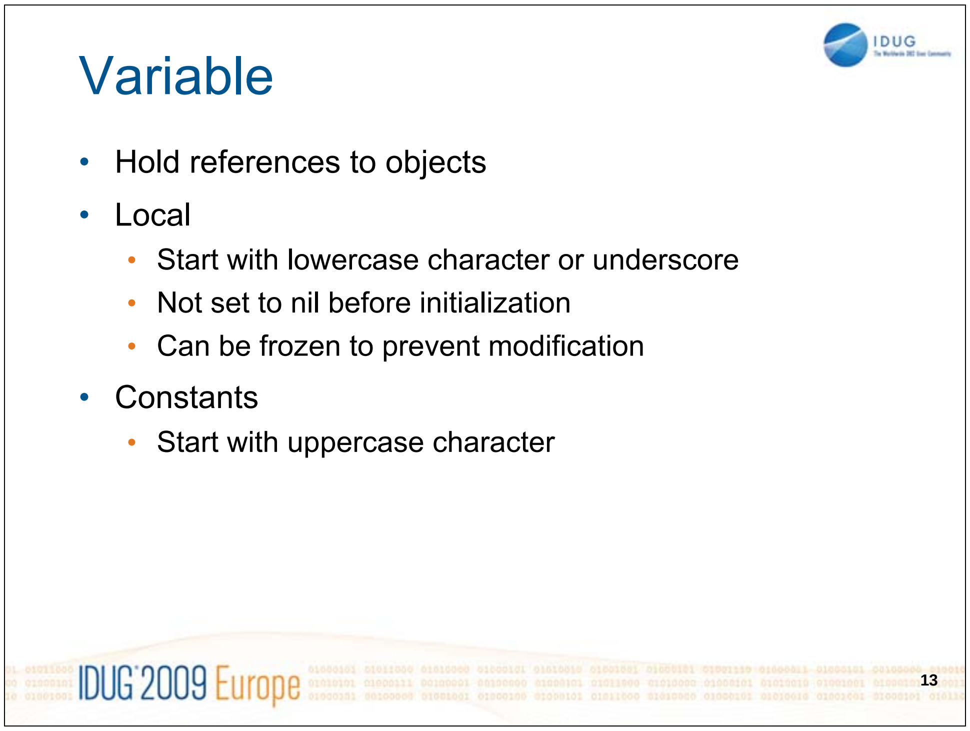
```
5.times { print "hello\n" }
```

```
hello
hello
hello
hello
hello
```

IDUG 2009 Europe

Remember everything in ruby is an object. Every variable is an object, and so is every literal. The variable names themselves are not objects. You can create a new object and assign it to a variable by invoking the "new" class method.

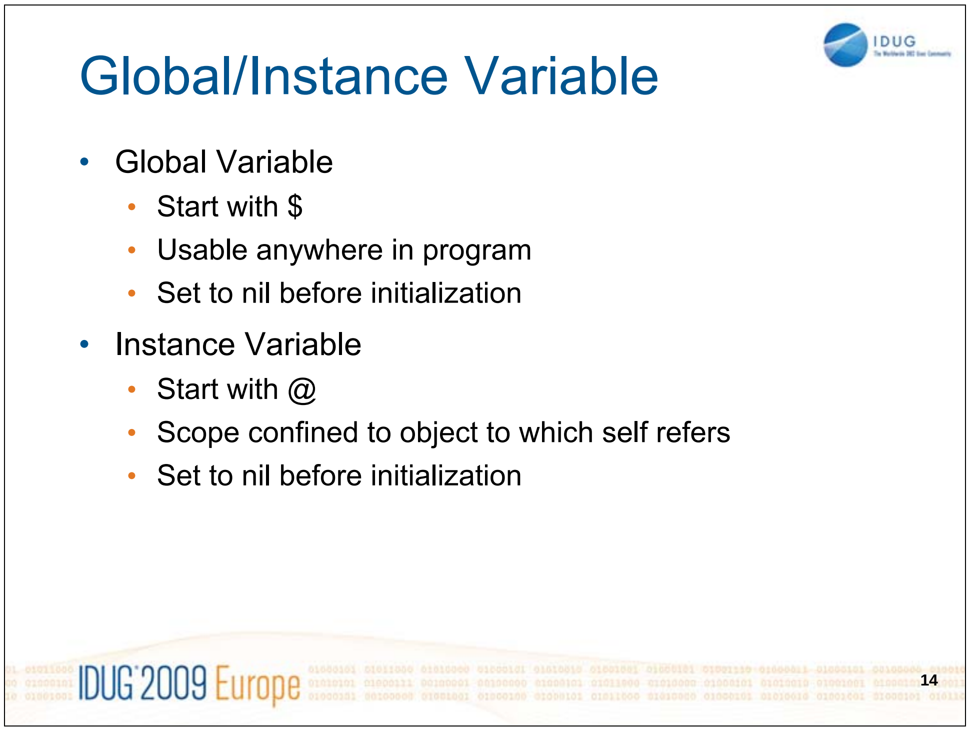
Because integers are objects, they have methods too. Ruby has quite a few handy built-in methods, like the one shown here that can be used as a looping control.



Variable

- Hold references to objects
- Local
  - Start with lowercase character or underscore
  - Not set to nil before initialization
  - Can be frozen to prevent modification
- Constants
  - Start with uppercase character

Like most languages, we have variables. These hold references to the objects they contain. Variables generally start with a lowercase character or an underscore. Constants start with an uppercase character. Both of these are limited to the scope in which they are defined. Class names start with an uppercase character too, but then they aren't variables.

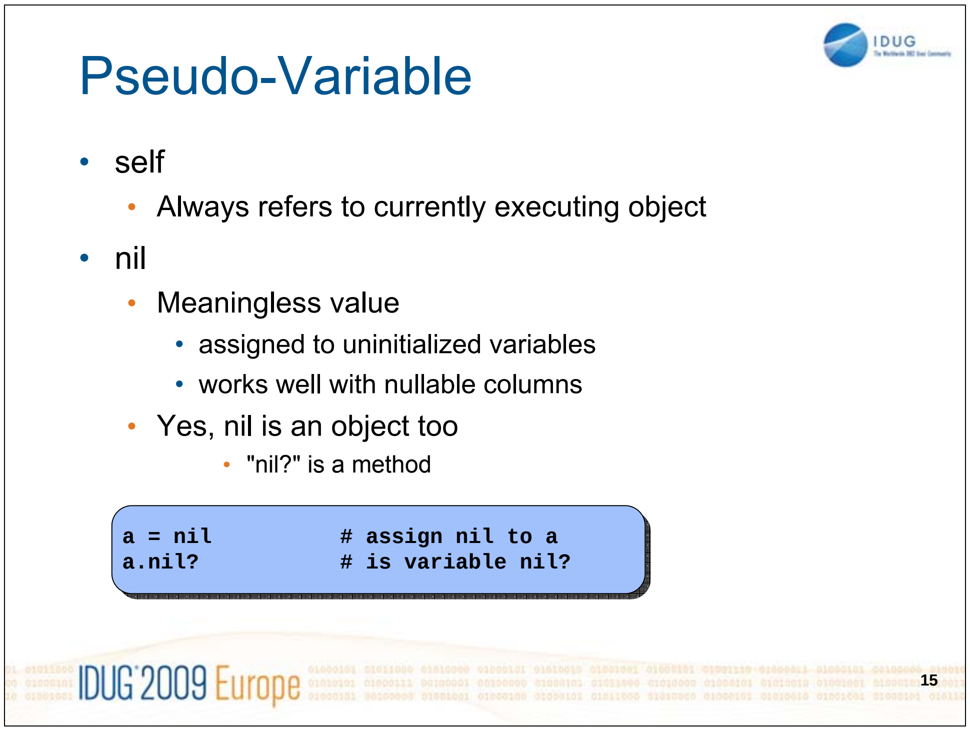


Global/Instance Variable

- Global Variable
  - Start with $
  - Usable anywhere in program
  - Set to nil before initialization
- Instance Variable
  - Start with @
  - Scope confined to object to which self refers
  - Set to nil before initialization

IDUG 2009 Europe

Global variables start with a dollar-sign and are usable anywhere in your program. Instance variables start with an at-sign and can only be used within the object class that defined them.



Pseudo-Variable

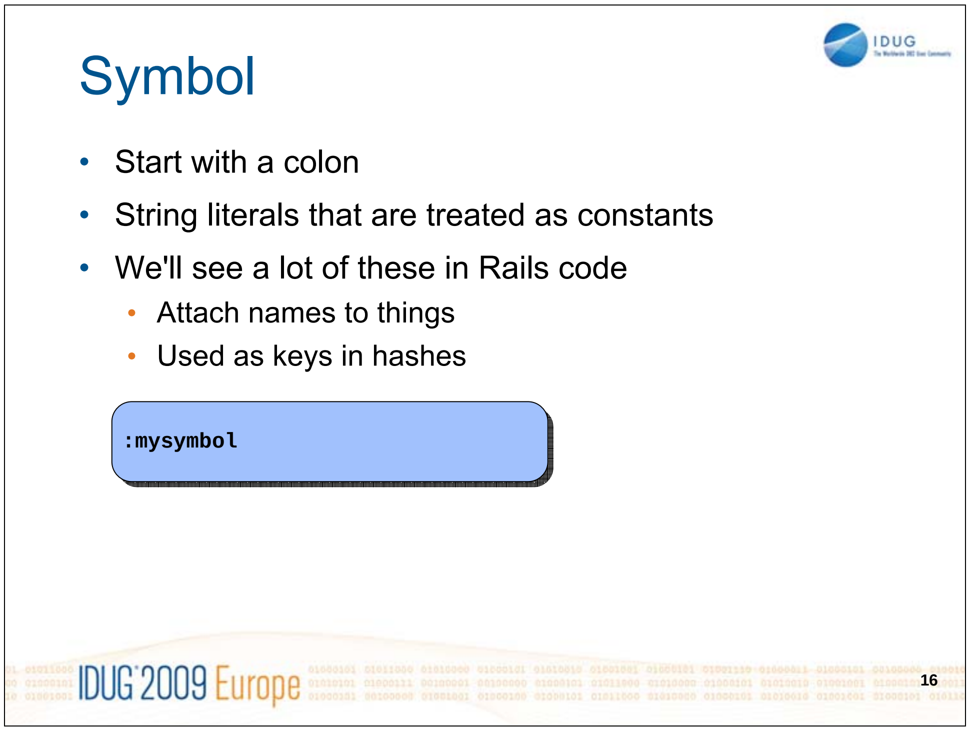
- self
  - Always refers to currently executing object
- nil
  - Meaningless value
    - assigned to uninitialized variables
    - works well with nullable columns
  - Yes, nil is an object too
    - "nil?" is a method

```
a = nil          # assign nil to a
a.nil?           # is variable nil?
```

IDUG 2009 Europe

Ruby also supports a few pseudo-variables. The "self" variable is used within a class to refer to itself.

Then the "nil" object is the equivalent of a null. Ruby actually implements a nil object with its own methods.



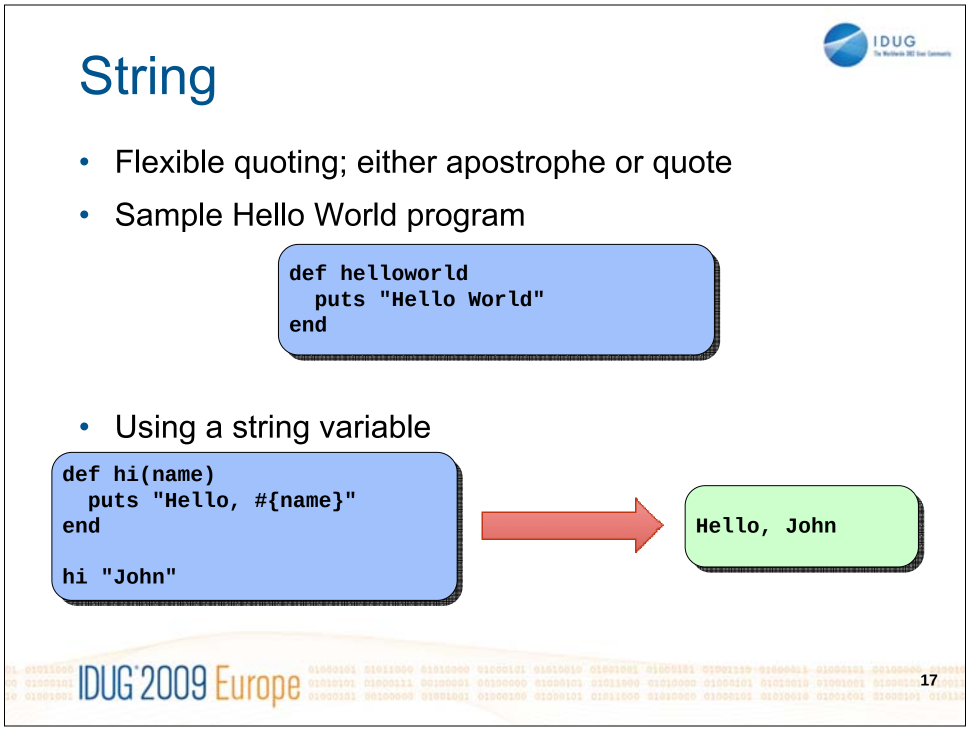
Symbol

- Start with a colon
- String literals that are treated as constants
- We'll see a lot of these in Rails code
  - Attach names to things
  - Used as keys in hashes

```
:mysymbol
```

IDUG 2009 Europe

Symbols are similar to literals and are implemented internally as identified integer values. Objects can be assigned to symbols similar to variables by using them as the keys in a hash. More on this in a bit. Rails makes extensive use of hashes and symbols for parameters passed into many methods.



String

- Flexible quoting; either apostrophe or quote
- Sample Hello World program

```
def helloworld
  puts "Hello World"
end
```

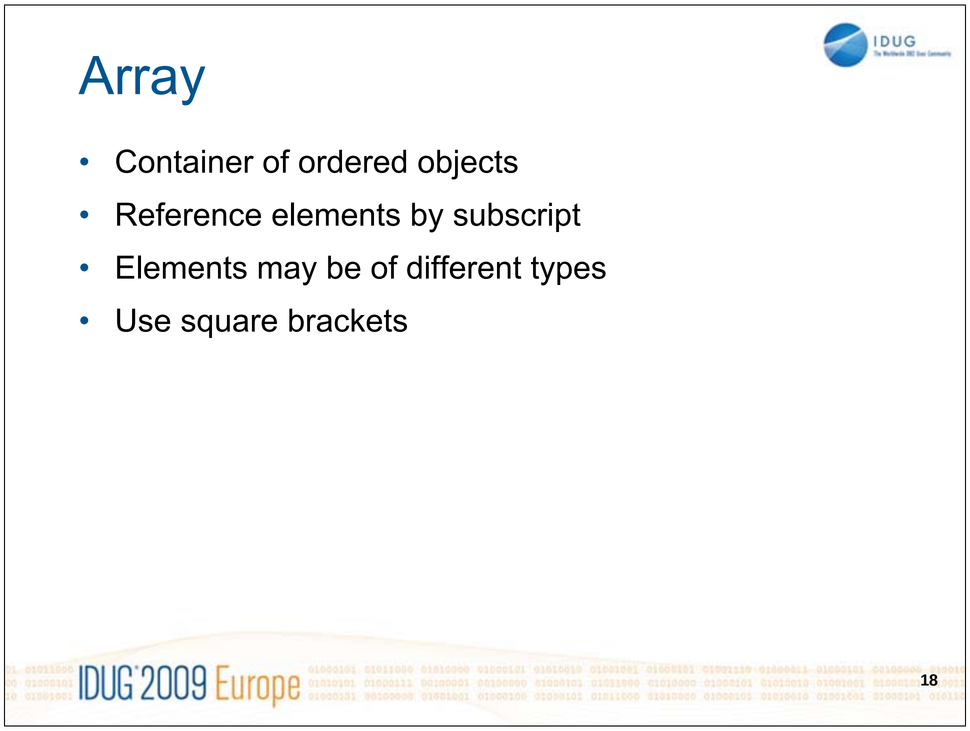
- Using a string variable

```
def hi(name)
  puts "Hello, #{name}"
end

hi "John"
```

```
Hello, John
```

Ruby supports string objects and variable substitution within string objects.



Array

- Container of ordered objects
- Reference elements by subscript
- Elements may be of different types
- Use square brackets

Like most useful languages, ruby has arrays as well. Square brackets with a numeric index are used to reference a particular array element. The first array element has a subscript of zero. Unlike many languages, the objects stored in a ruby array do not need to have the same type.

Array Examples

- Create an array

```
arr1 = Array.new
```

```
[]
```

- Add "string1" to the end

```
arr1 << "string1"
```

```
["string1"]
```

- Set 2nd element

```
arr1[1] = 13
```

```
["string1", 13]
```

- Replace the contents of the array

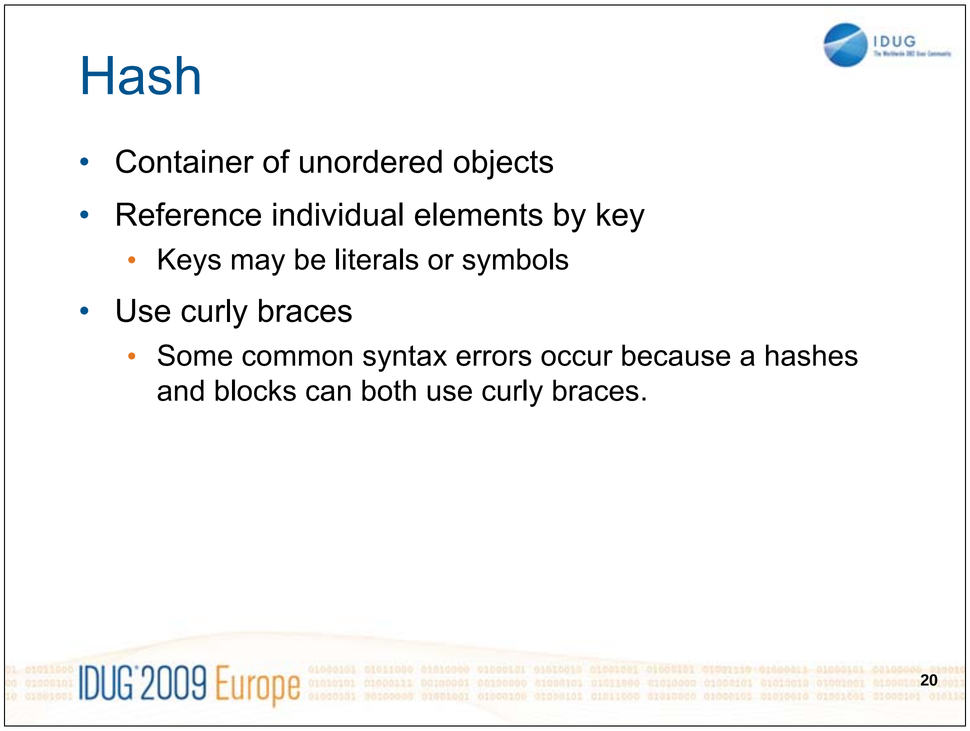
```
arr1 = [ 1, 2, 3, 4, 5]
```

```
[1, 2, 3, 4, 5]
```

- Display the 5th element

```
puts arr1[4]
```

```
5
```



Hash

- Container of unordered objects
- Reference individual elements by key
  - Keys may be literals or symbols
- Use curly braces
  - Some common syntax errors occur because a hashes and blocks can both use curly braces.

IDUG 2009 Europe

20

Ruby also implements hashes. Hashes are similar to arrays, but use a key to reference elements rather than a numeric index.

Hash Examples

- Creation and assignment

```
h1 = Hash.new
h1 = { 'k1' => 1, 'k2' => 2 }
```

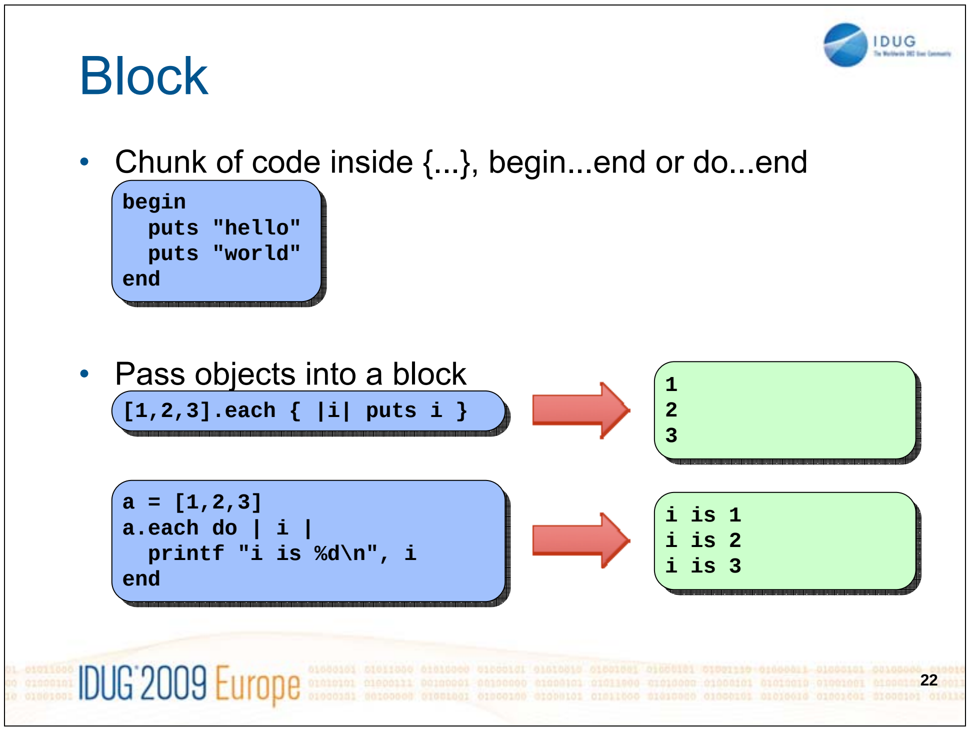
- Symbols as keys

```
h1 = { :sym1 => true, :sym2 => "Hi" }
```

- Retrieval

```
puts h1[:sym2]
```

```
Hi
```



Block

- Chunk of code inside {...}, begin...end or do...end

```
begin
  puts "hello"
  puts "world"
end
```

- Pass objects into a block

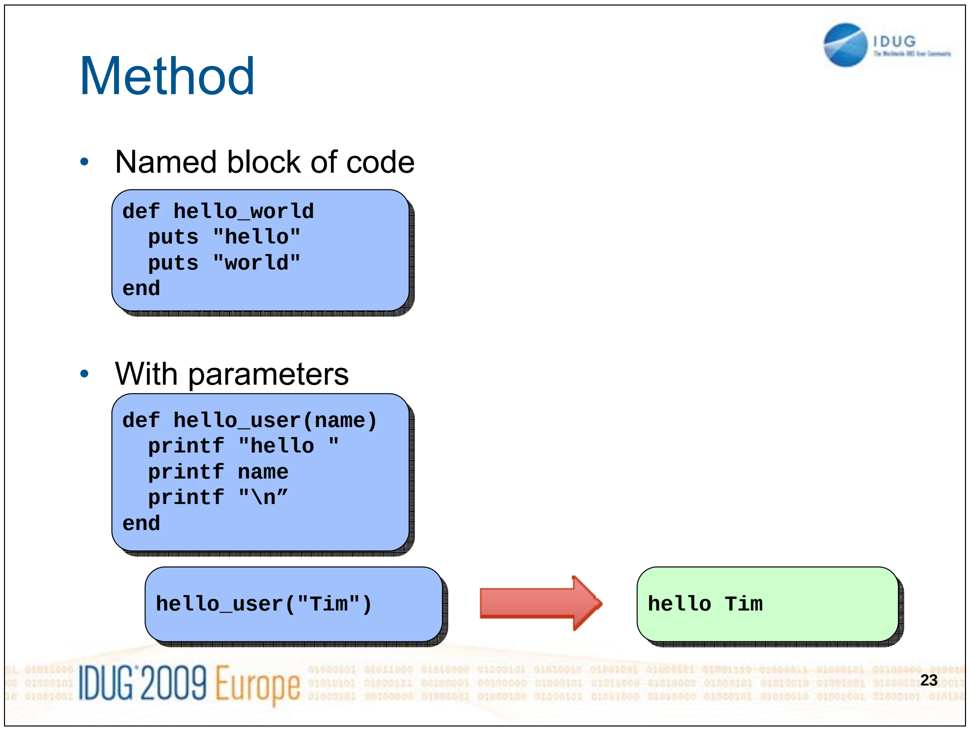
```
[1,2,3].each { |i| puts i }
```

```
1
2
3
```

```
a = [1,2,3]
a.each do | i |
  printf "i is %d\n", i
end
```

```
i is 1
i is 2
i is 3
```

Program code in ruby can be grouped into blocks. Blocks can be delimited by either curly braces or "do" and "end". Blocks of code can accept parameters passed into them from the caller. This is one of the key fun things about ruby and takes advantage of the dynamic typing. As long as the code within a block only uses methods available to the type of object passed in it can act on different input object types.



Method

- Named block of code

```
def hello_world
  puts "hello"
  puts "world"
end
```

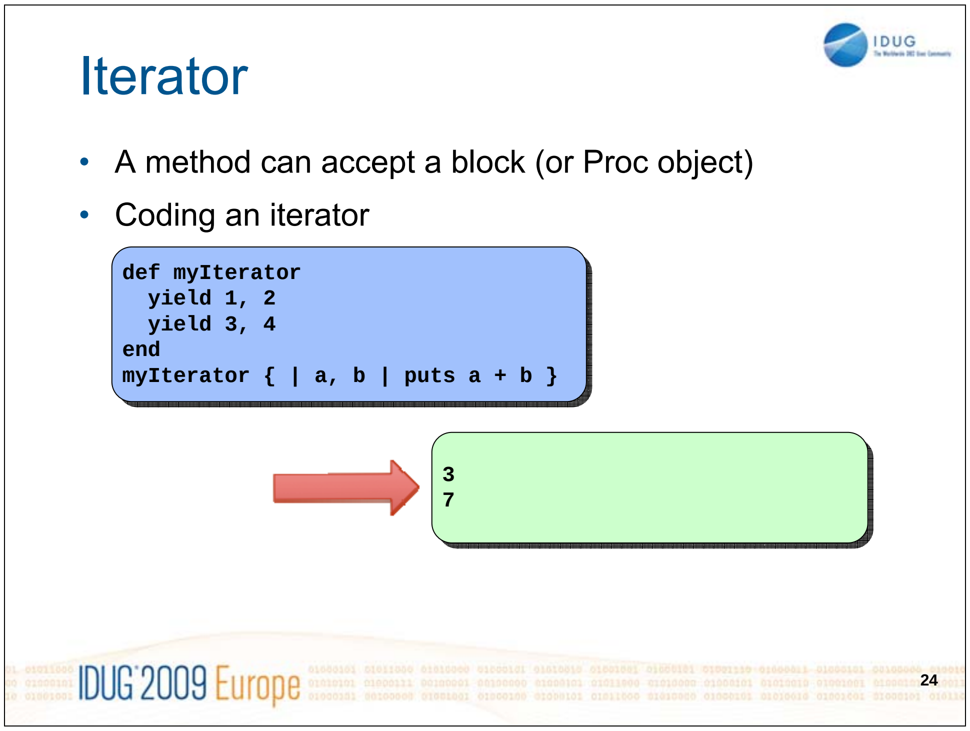
- With parameters

```
def hello_user(name)
  printf "hello "
  printf name
  printf "\n”
end
```

```
hello_user("Tim")
```

```
hello Tim
```

Program code in ruby can be grouped into blocks. Blocks can be delimited by either curly braces or "do" and "end". Blocks of code can accept parameters passed into them from the caller. This is one of the key fun things about ruby and takes advantage of the dynamic typing. As long as the code within a block only uses methods available to the type of object passed in it can act on different input object types.

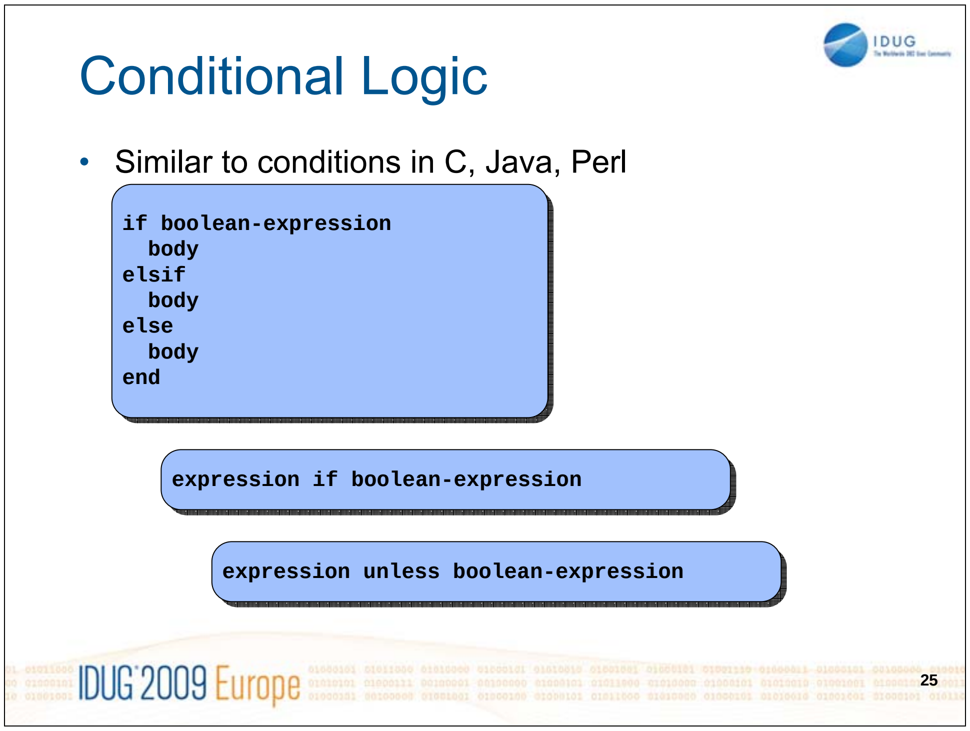

Iterator

- A method can accept a block (or Proc object)
- Coding an iterator

```
def myIterator
  yield 1, 2
  yield 3, 4
end
myIterator { | a, b | puts a + b }
```

```
3
7
```

Iterators are methods that accept blocks of code. The iterator takes the block and then invokes it using the "yield" method. The example here isn't typical as iterators would more frequently loop through some kind of structure, input file, or query result.



Conditional Logic

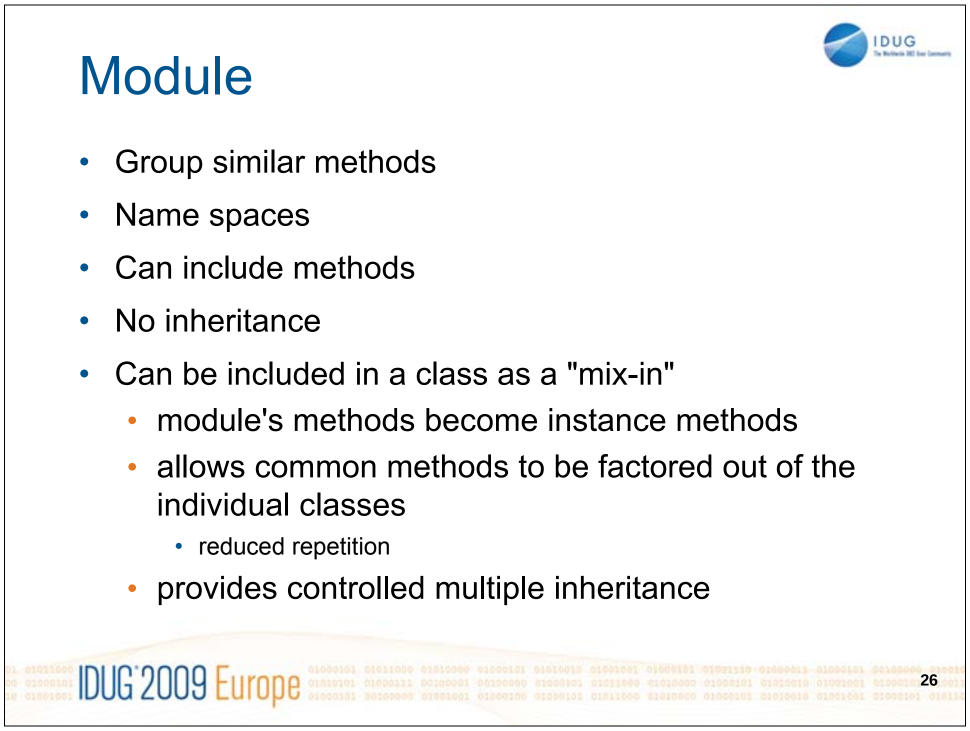
- Similar to conditions in C, Java, Perl

```
if boolean-expression
  body
elsif
  body
else
  body
end
```

```
expression if boolean-expression
```

```
expression unless boolean-expression
```

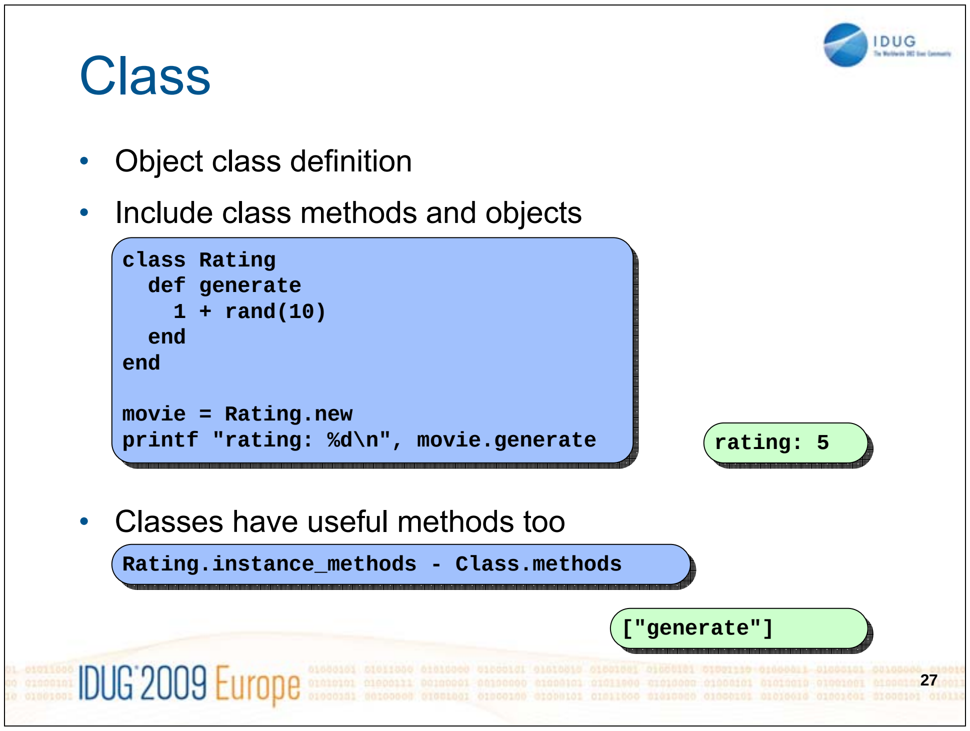
Imaging this. Conditional logic that looks like a variety of other languages. And like perl, you can place the conditional after its resulting expression.



Module

- Group similar methods
- Name spaces
- Can include methods
- No inheritance
- Can be included in a class as a "mix-in"
  - module's methods become instance methods
  - allows common methods to be factored out of the individual classes
    - reduced repetition
  - provides controlled multiple inheritance

Modules in ruby are a method of grouping similar methods together. Methods do not inherit from classes, but can be used within a class to implement interfaces (which act similar to multiple inheritance in some languages).



Class

- Object class definition
- Include class methods and objects

```
class Rating
  def generate
    1 + rand(10)
  end
end

movie = Rating.new
printf "rating: %d\n", movie.generate
```

```
rating: 5
```

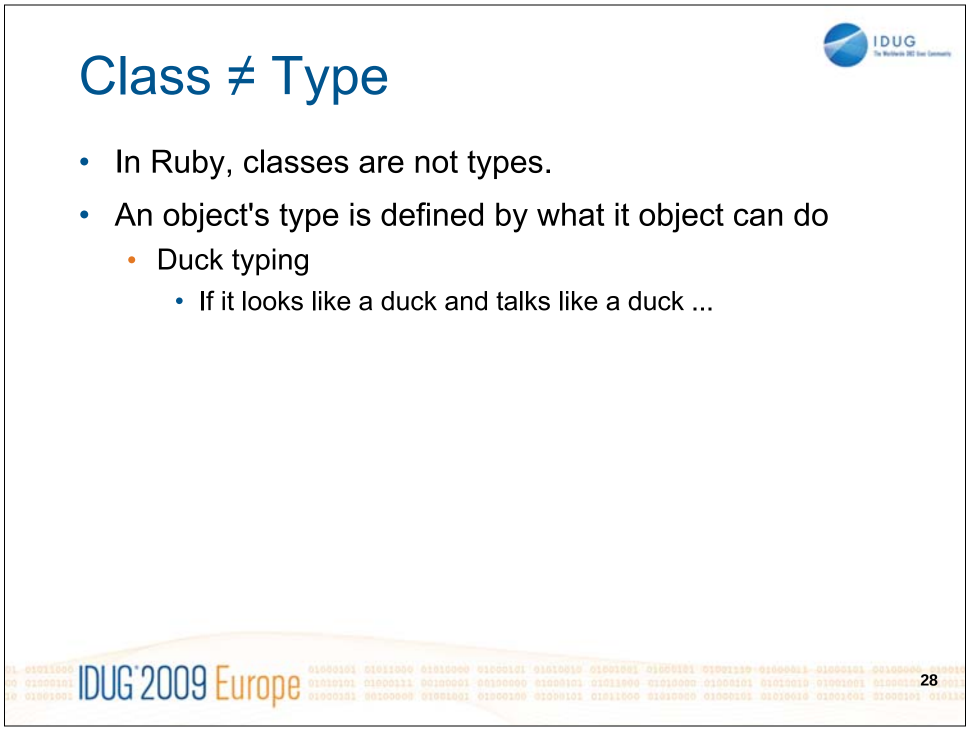
- Classes have useful methods too

```
Rating.instance_methods - Class.methods
```

```
["generate"]
```

And then there is the class. This is, after all, an object-oriented language.

And a class is itself an object with appropriate methods for introspection.

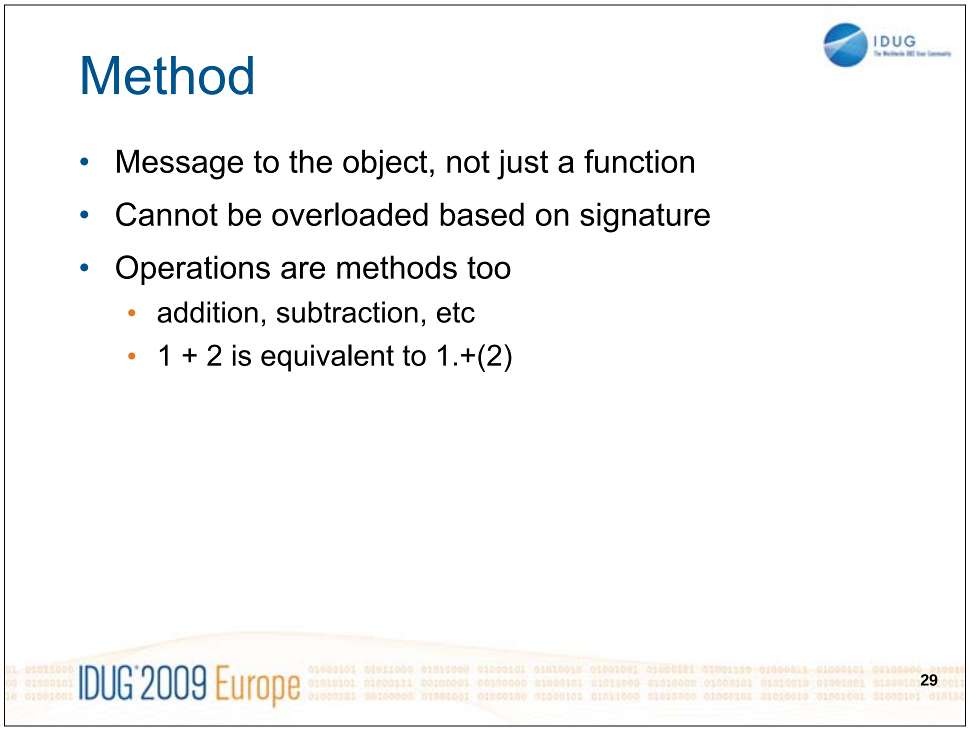


Class ≠ Type

- In Ruby, classes are not types.
- An object's type is defined by what it object can do
  - Duck typing
    - If it looks like a duck and talks like a duck ...

IDUG 2009 Europe

But here is where we move away from the familiar. In ruby, a type is defined by the methods that an object supports. This means that an object's type is more than its class. The class does influence the type since it defines the methods and the methods invoked are those of the class (or mix-in modules).



Method

- Message to the object, not just a function
- Cannot be overloaded based on signature
- Operations are methods too
  - addition, subtraction, etc
  - 1 + 2 is equivalent to 1.+(2)

Methods in ruby are implemented as messages to objects rather than functions. This makes it more like Java than C++. Unlike Java and C++, ruby does not support overloading of methods. But then it doesn't have to because the same method can accept any "type" rather than requiring static types.

Inheritance

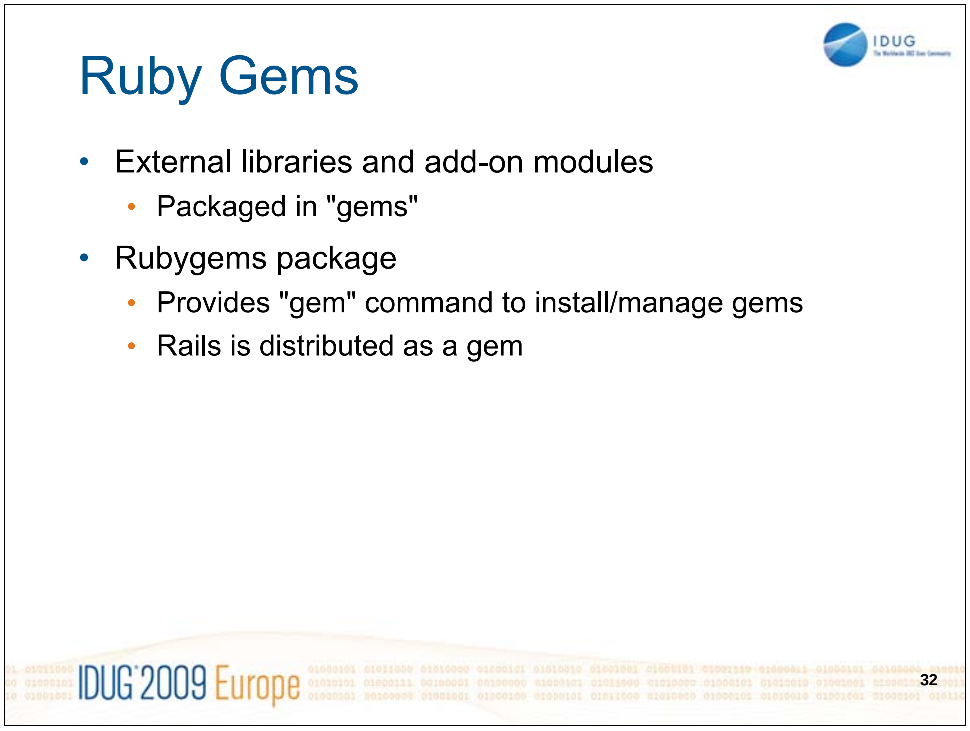
- Classes may have only one parent class

```
class MyClass < MySuperClass
  # class content goes here
end
```

Mix-in

- Solution for classes that may benefit from "multiple" inheritance.
- Similar to Java interfaces

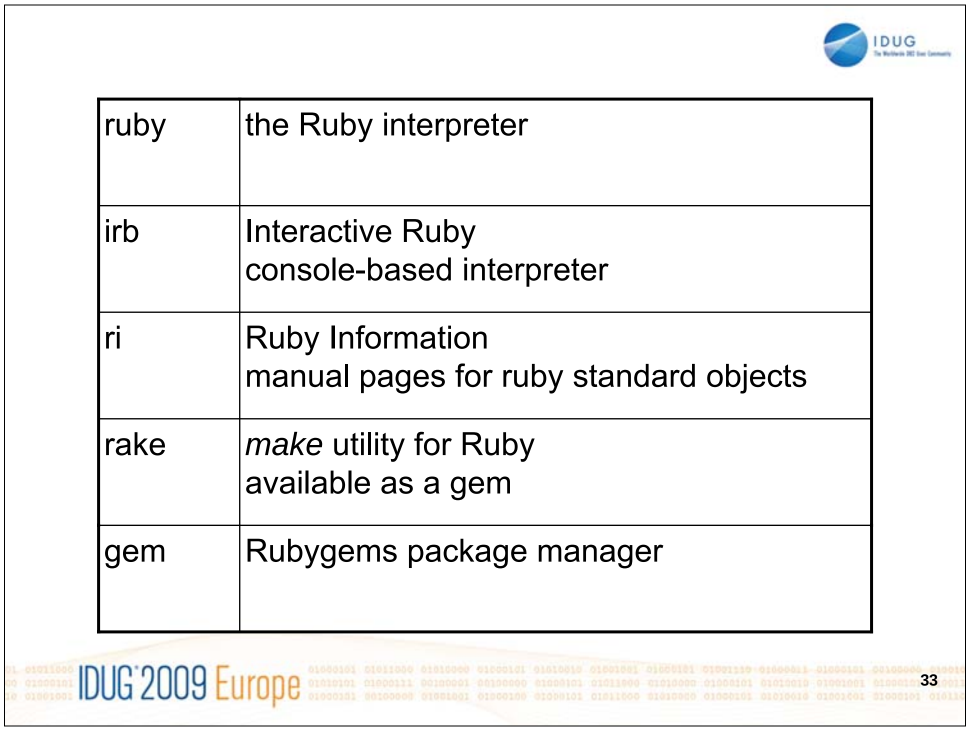
```
class MyClass < MySuperClass
  include MyInterface
  # MyClass specific stuff goes here
end
```



Ruby Gems

- External libraries and add-on modules
  - Packaged in "gems"
- Rubygems package
  - Provides "gem" command to install/manage gems
  - Rails is distributed as a gem

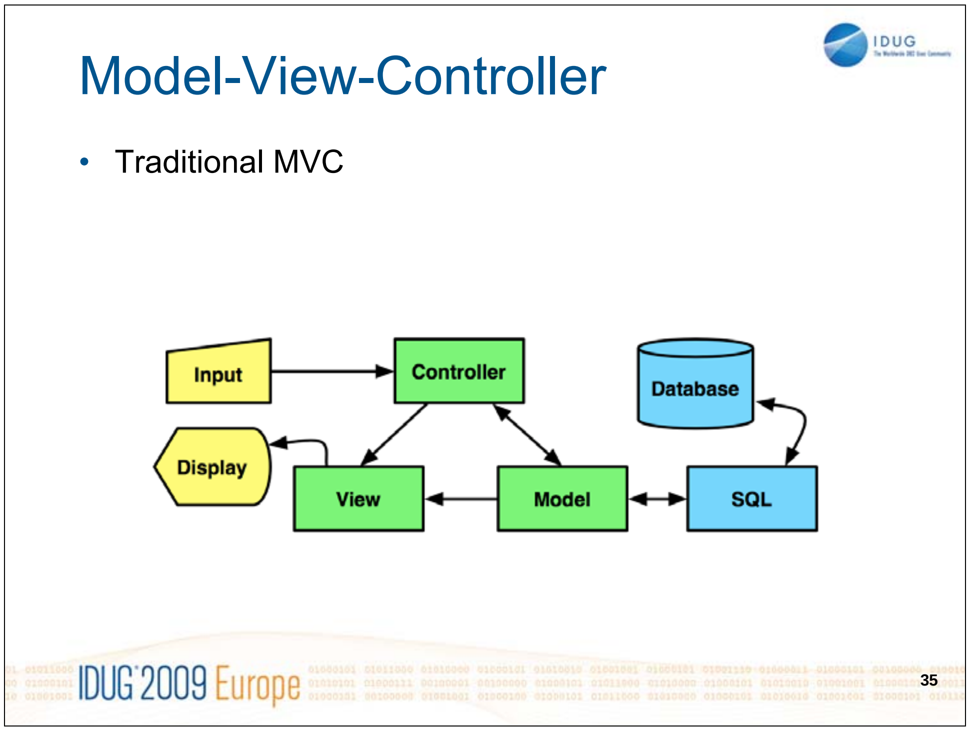
Many developers share their ruby routines, modules, and classes with others. Some have developed an easy method to deliver these shared modules over the Internet. The Rubygems package, once installed, provides the gem command that allows you to find install and upgrade shared pieces of code. Rails itself is packages as a gem.



ruby	the Ruby interpreter
irb	Interactive Ruby console-based interpreter
ri	Ruby Information manual pages for ruby standard objects
rake	*make* utility for Ruby available as a gem
gem	Rubygems package manager

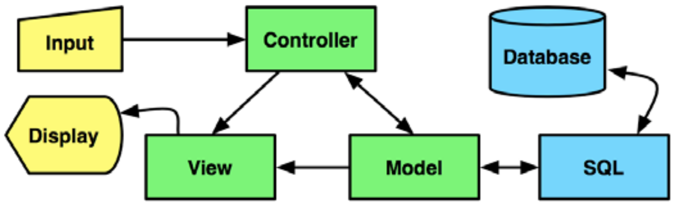
Ruby includes a couple of line commands. The primary of these is "ruby" which invokes the ruby interpreter. Remember that ruby is a scripting language.

Rails: the Framework

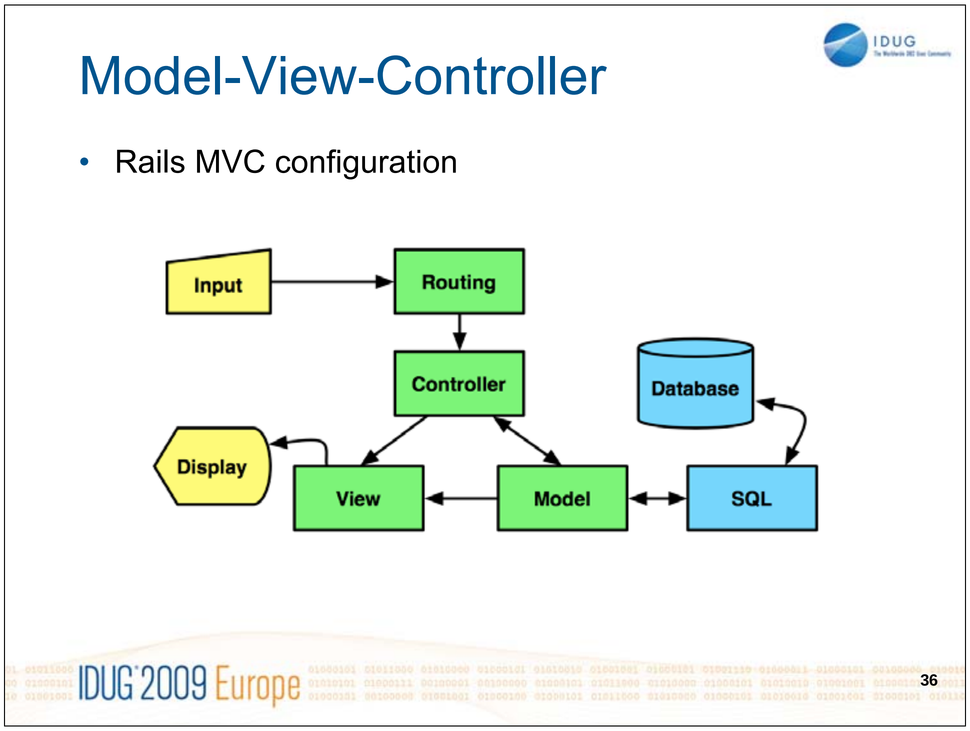


Model-View-Controller

- Traditional MVC

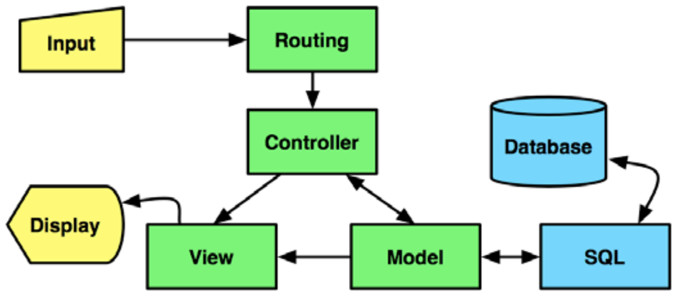


Here is a picture of your standard model-view-controller configuration. The controller receives input messages and passes them along to the model and view. Business logic is usually located in the controller. The model deals with the data and interfaces with the RDBMS. Both the controller and the model feed the view, which is responsible for displaying the information back to the user of the application. In a web-based application, the view is generally implemented in HTML or JSP.

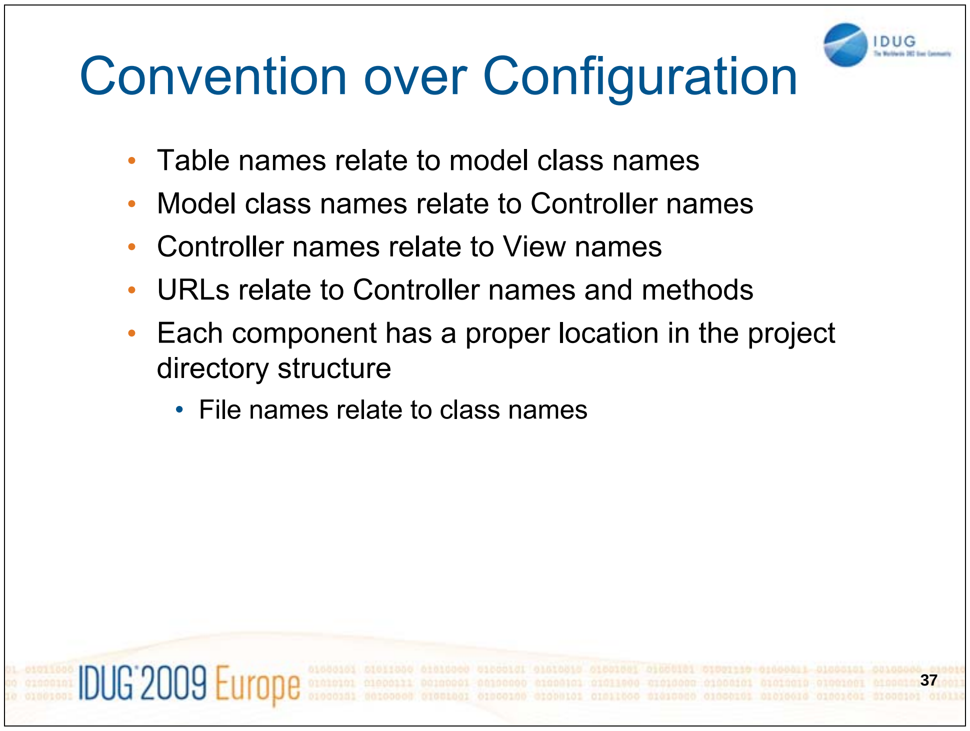


Model-View-Controller

- Rails MVC configuration



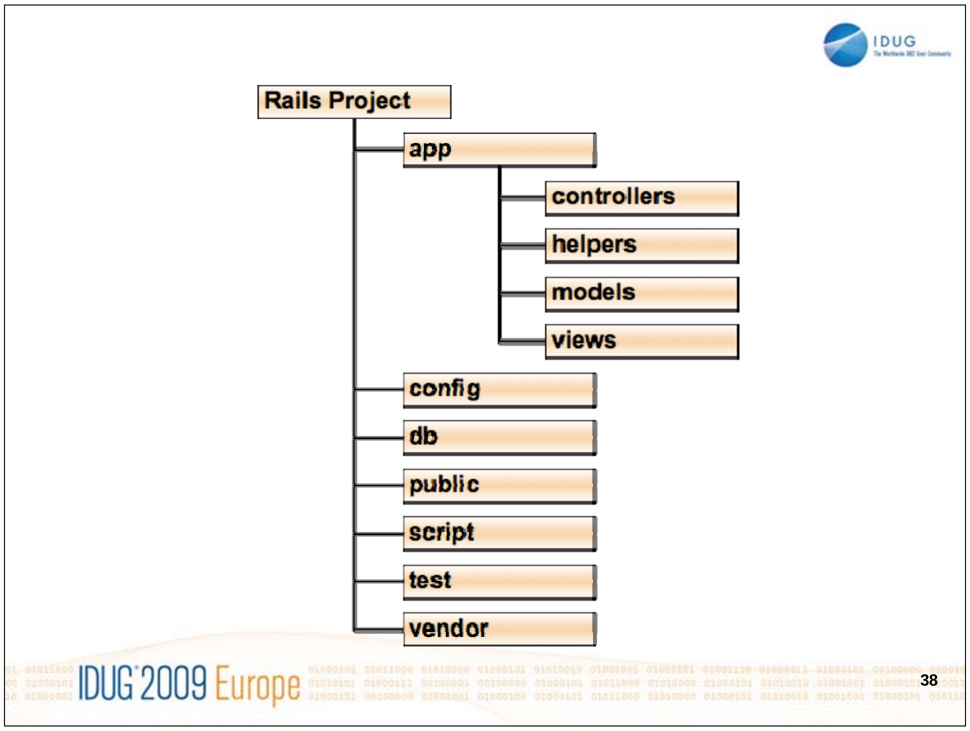
Rails adds one more component to this mix. The routing component takes all input and decides which controller should receive the input messages. This provides the ability to easily add common logic for all controllers.



Convention over Configuration

- Table names relate to model class names
- Model class names relate to Controller names
- Controller names relate to View names
- URLs relate to Controller names and methods
- Each component has a proper location in the project directory structure
  - File names relate to class names

Rather than require lots of configuration files and details in order to link components, rails uses conventions. The names of the controller, model and view elements are closely related. The URLs used to access the controllers use similar conventions. And to keep it all organized, the application hierarchy defines a place for each item.



Here are just some of the directories within a rails project. Most of your actual application code goes under the "app" directory. The database configuration is stored under "config" and non-programmatic web content goes under "public". The “script” directory provides script that help manage your application.

The “test” directory can be used to place components for test-driven development, including unit, functional and integration test cases. Rails provides a rake script that executes all of the defined tests and reports any errors.

Models

- Active Record implements the Model components
  - Implements CRUD operations
  - Allows SQL operations as well
- Model files are stored in:
  - app/models
- One file for each model
  - Name is singular of table name with .rb
    - UserProfile class for user_profiles table is in
    - app/models/user_profile.rb
    - Inherits from ActiveRecord::Base

Model Names

- Table/Model Names
  - Table names
    - are plural
    - use underscores
    - Examples: exercises, user_profiles
  - Model class names are singular
    - singular
    - use multiple capitalized words
    - Examples: Exercise, UserProfile

Models: Keys

- Primary Keys
  - Surrogate keys are used by default
  - Every table has primary key column called ***id***
  - Override-able, but more work
  - Rails does not support composite primary keys
    - But there are plugins that implement support
- Foreign Keys
  - Foreign keys use singular name of parent table plus *_id*

Models: Validation

- Active Record can perform validation on content

```
validates_uniqueness_of :name
validates_numericality_of :duration
validates_length_of :name :in => 2..30
validates_presence_of :start_date
```

Models: Relationships

- Relationships between tables are defined within the models

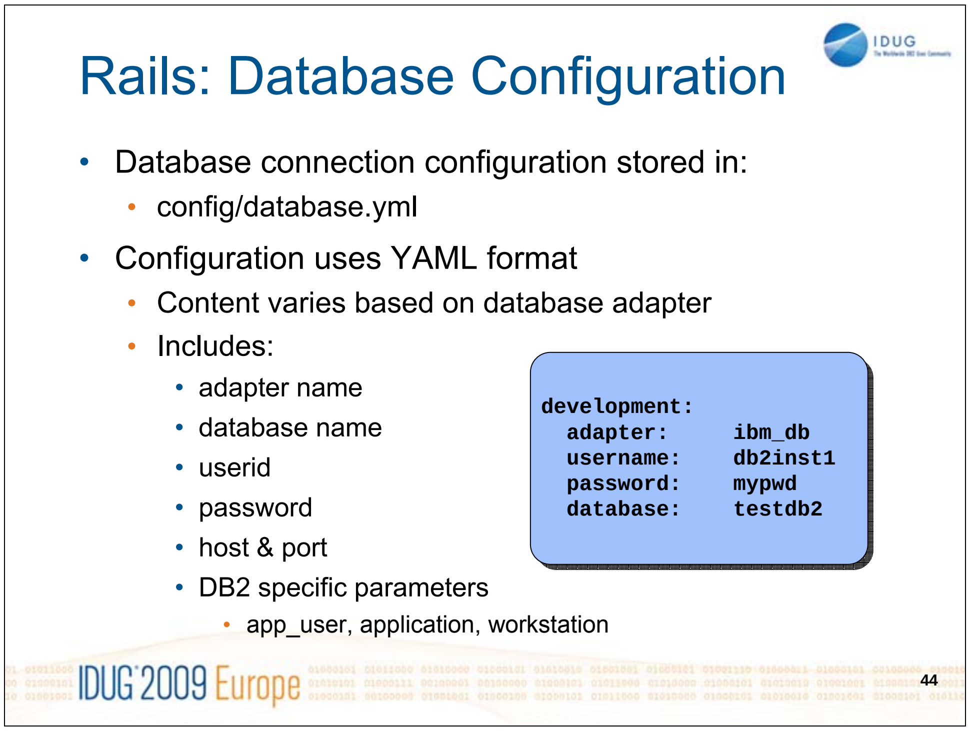
```
belongs_to :parenttable
has_many :childtable
```

- Defined relationships allow pre-loading of data from the related tables

```
w = Workout.find(:all,:include => :exercise)
```

  - Exercise columns are available

```
puts w.exercise.name
```


Rails: Database Configuration

- Database connection configuration stored in:
  - config/database.yml
- Configuration uses YAML format
  - Content varies based on database adapter
  - Includes:
    - adapter name
    - database name
    - userid
    - password
    - host & port
    - DB2 specific parameters
      - app_user, application, workstation

```
development:
  adapter:     ibm_db
  username:    db2inst1
  password:    mypwd
  database:    testdb2
```

Here is one of the few exceptions to providing configuration information using ruby. The database configuration file tells rails which database adapter module to use as well as the parameters necessary to start and control the connection.

Controllers

- Action Pack implements the Controller and View components
  - Action Controller manages the application routing and controller invocation
- Controller files are stored in:
  - app/controllers
- One file for each controller
- Routes are defined in:
  - config/routes.rb

Controller Names

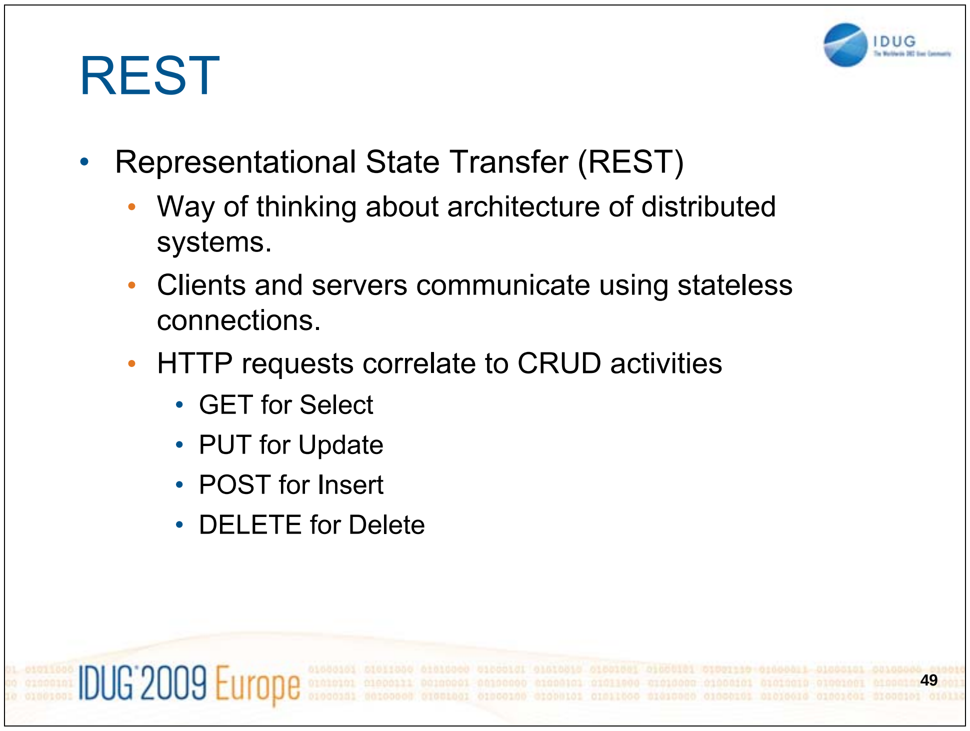
- Controller names relate to URLs
  - Each URL routes to a controller and an action (method/message) within the controller and passes additional parameters
  - URL to controller action mapping controlled by:
    - config/routes.rb
- Controller names often relate to Model names
  - ExerciseController is the controller class for the Exercise model class
    - File name is exercise_controller.rb

Application Controller

- The ApplicationController class
  - in file application.rb
  - Contains code executed every time the application receives an incoming message
  - Useful for managing common tasks:
    - Security controls
    - Session management

Controller Methods

- Basic CRUD methods
  - **index** - returns list of data rows
  - **show** - selects a single row
  - **new** - creates a new model object for the client
  - **create** - inserts a new row (following new display)
  - **edit** - returns a model form so client can update
  - **update** - updates a single row (following edit display)
  - **destroy** - deletes a single row



REST

- Representational State Transfer (REST)
  - Way of thinking about architecture of distributed systems.
  - Clients and servers communicate using stateless connections.
  - HTTP requests correlate to CRUD activities
    - GET for Select
    - PUT for Update
    - POST for Insert
    - DELETE for Delete

IDUG 2009 Europe

REST was formulated by Roy Fielding in his 2000 PhD dissertation.

Originally, rails implemented all of the CRUD functions using the HTTP GET/POST methods. More recently, there has been some thought that a more RESTful approach would use the HTTP PUT and DELETE methods as well. This has a bit of an impact on the URLs, but the resulting application works well as both a web application for users and a web service for other clients. We won't really have time to explore this today, but it is definitely worth pursuing if you want to give rails a try.

REST Routes and URLs

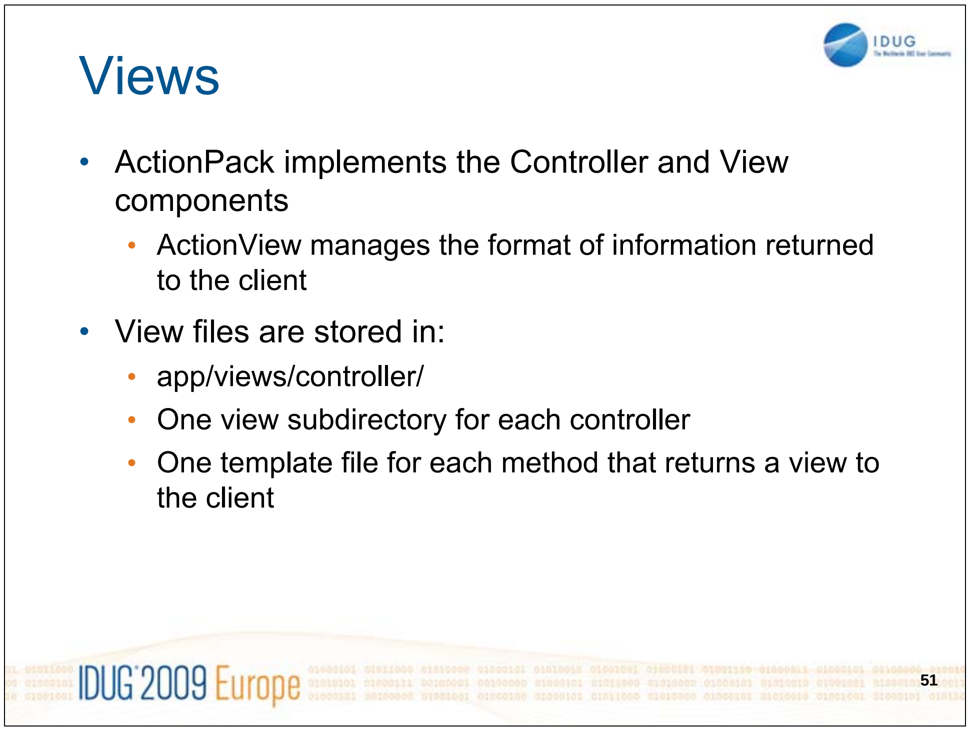
- In config/routes.rb

```
map.resources :exercises
```

HTTP	URL	Controller	Action
GET	/exercises	exercises	index
GET	/exercises/new	exercises	new
POST	/exercises	exercises	create
GET	/exercise/1	exercises	show
GET	/exercise/1/edit	exercises	edit
PUT	/exercise/1	exercises	update
DELETE	/exercise/1	exercises	destroy

- Review routes with

```
rake routes
```



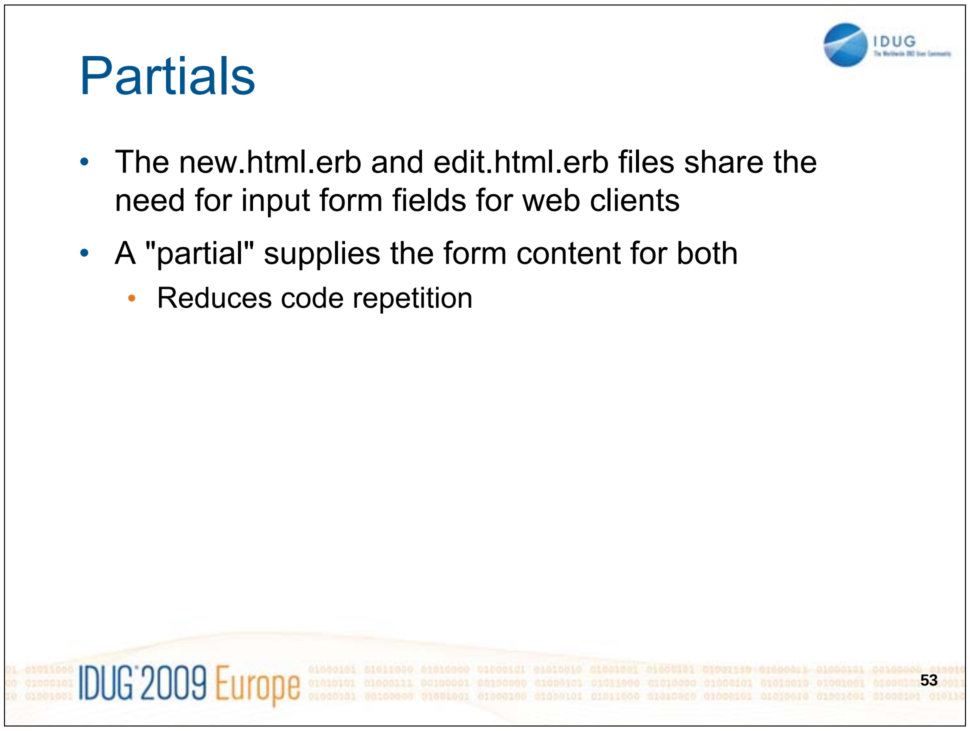
Views

- ActionPack implements the Controller and View components
  - ActionView manages the format of information returned to the client
- View files are stored in:
  - app/views/controller/
  - One view subdirectory for each controller
  - One template file for each method that returns a view to the client

Because rails was originally developed for web-based applications, its first implementation of Views (from MVC) used a template mechanism that allows the mixing of HTML and ruby code in one file. Generally, there is one template file for each page layout that is returned to the user. Each controller can have multiple templates, so the templates are stored in an appropriate subdirectory using the controller name.

View Names

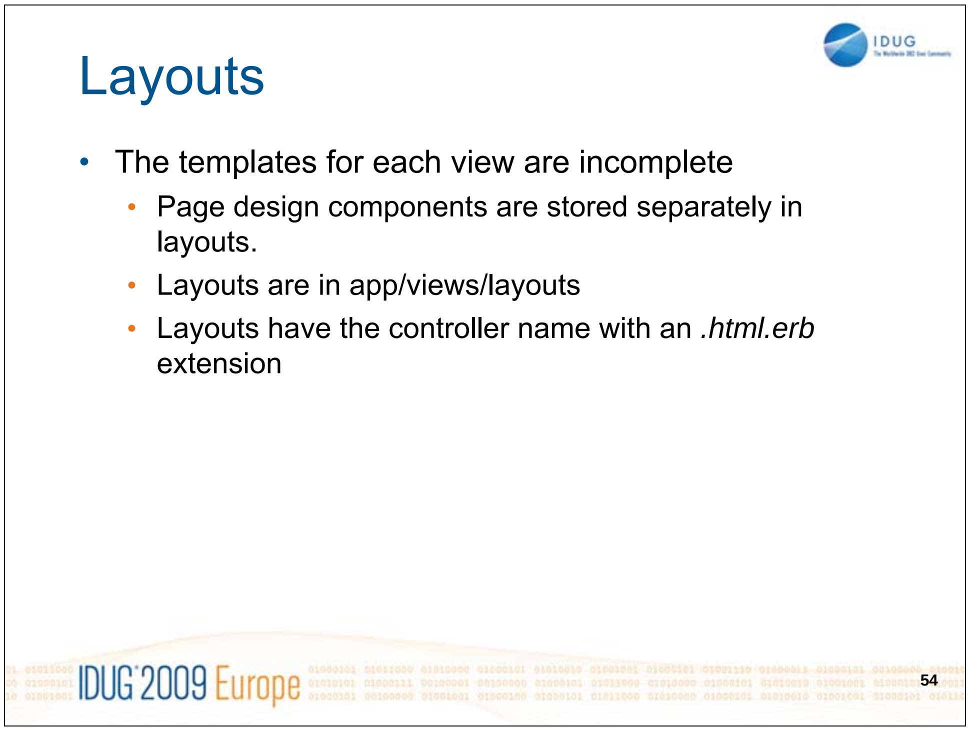
- View subdirectory has same name as the controller
  - without *_controller.rb* at the end
- The view templates for the ExerciseController class are in:
  - app/views/exercise/
- The templates are usually:
  - index.html.erb
  - new.html.erb
  - edit.html.erb
  - show.html.erb



Partials

- The new.html.erb and edit.html.erb files share the need for input form fields for web clients
- A "partial" supplies the form content for both
  - Reduces code repetition

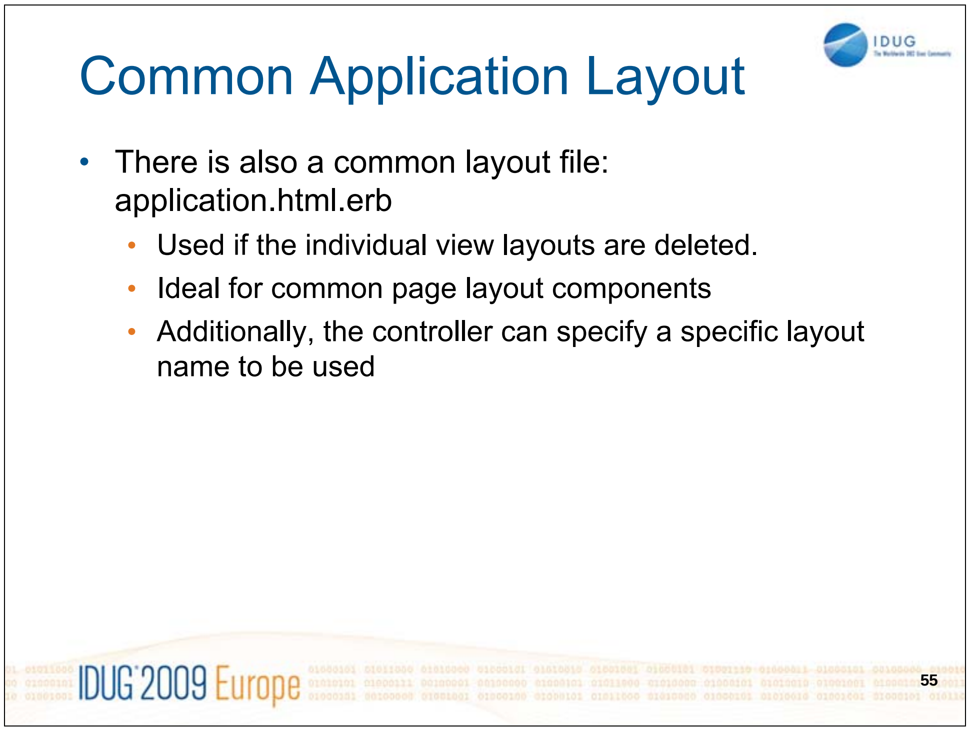
Partials are templates that do not make up a complete web page. They are included from other web pages in order to reduce duplication.



Layouts

- The templates for each view are incomplete
  - Page design components are stored separately in layouts.
  - Layouts are in app/views/layouts
  - Layouts have the controller name with an *.html.erb* extension

Additionally, each view can have a layout file. The layout file contains the header and trailer types of information that will be used to build the page.



Common Application Layout

- There is also a common layout file: application.html.erb
  - Used if the individual view layouts are deleted.
  - Ideal for common page layout components
  - Additionally, the controller can specify a specific layout name to be used

IDUG 2009 Europe

To make it easier to develop consistent websites and reduce duplication of effort, you can delete all of your view-specific layout files and use a single layout for the entire application.

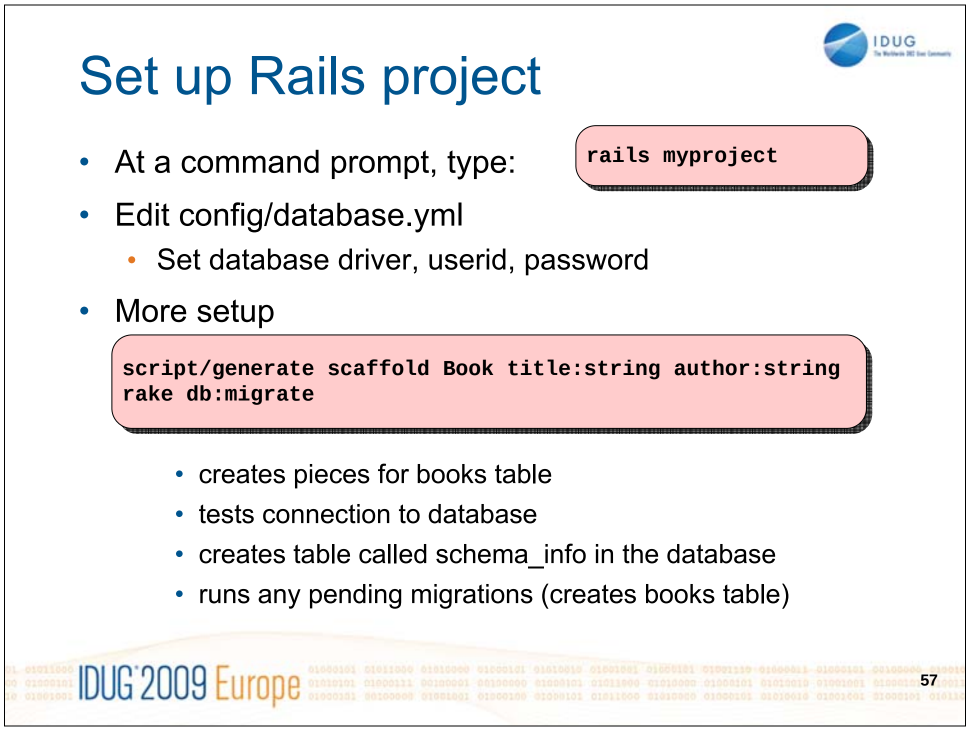
Embedding Ruby Code

- Templates may contain embedded Ruby statements
- Execute code without output

```
<% exset = Exercise.find(:all) %>
```

- Execute code with output

```
<%= puts "this goes into output" %>
```



Set up Rails project

- At a command prompt, type:

```
rails myproject
```

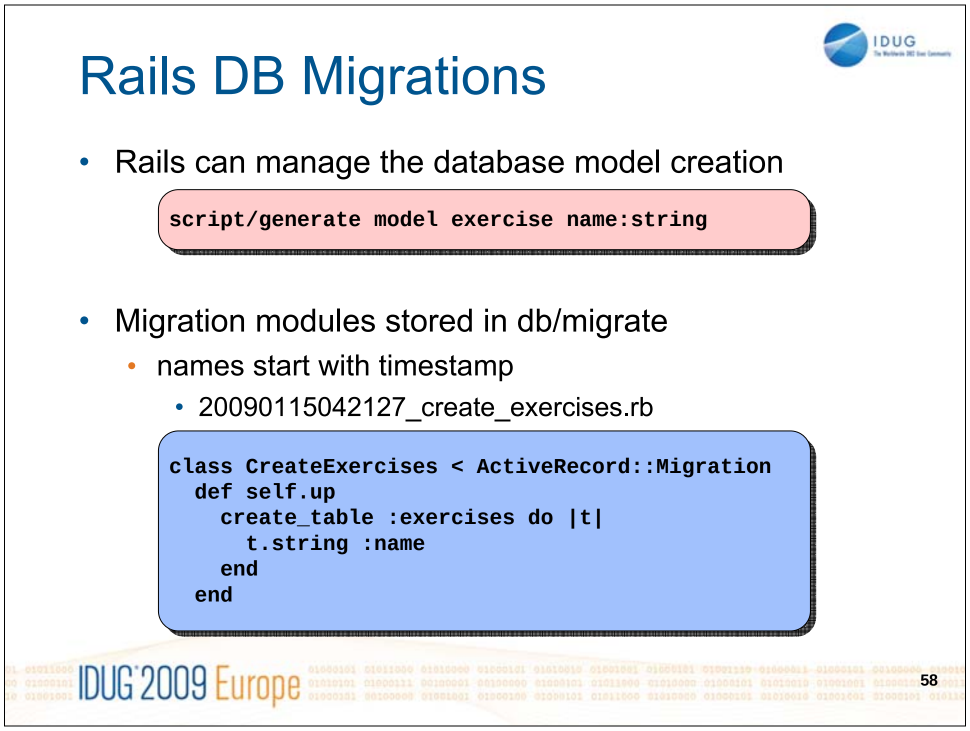
- Edit config/database.yml
  - Set database driver, userid, password
- More setup

```
script/generate scaffold Book title:string author:string
rake db:migrate
```

  - creates pieces for books table
  - tests connection to database
  - creates table called schema_info in the database
  - runs any pending migrations (creates books table)

IDUG 2009 Europe

To set up a rails project, you first have to install ruby, rubygems and rake. When you are ready to create your rails project, you just use a command window (or terminal) to run the rails command. This will create the project directory and the hierarchy of necessary files under it.



Rails DB Migrations

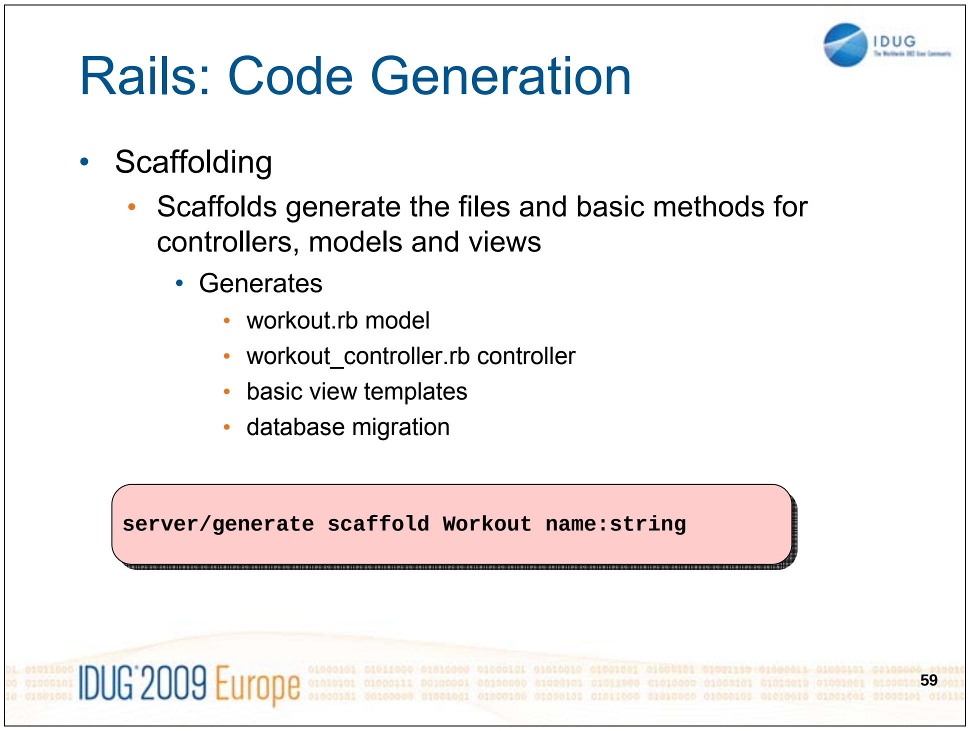
- Rails can manage the database model creation

```
script/generate model exercise name:string
```

- Migration modules stored in db/migrate
  - names start with timestamp
    - 20090115042127_create_exercises.rb

```
class CreateExercises < ActiveRecord::Migration
  def self.up
    create_table :exercises do |t|
      t.string :name
    end
  end
```

In addition to using the database, rails also has the capability to manage migrations of the database schema within the database.



Rails: Code Generation

- Scaffolding
  - Scaffolds generate the files and basic methods for controllers, models and views
    - Generates
      - workout.rb model
      - workout_controller.rb controller
      - basic view templates
      - database migration

```
server/generate scaffold Workout name:string
```

And now for the real power of rails, the generator. Rails includes several very handy scripts. The "generate" script creates basic functional controllers, models and views for you. Given a table named "workouts" in your database, the command here will generate the code you need to create, update, delete, list and select the rows of that table. Actually, with Rails 2.x the command will generate a “migration” that will create the table for you.

Once you've done this, you can configure your web server to point to your rails project and start it up. And your basic "prototype" application is up and running. Which brings us to the demo ...

Appendix

- Ruby Resources
- Rails Resources

Ruby Resources

- Ruby Home Page
  - http://www.ruby-lang.org/en/
- Try Ruby!
  - http://tryruby.hobix.com/
- Ruby Documentation
  - http://www.ruby-doc.org/
- Ruby User's Guide
  - http://www.rubyist.net/~slagell/ruby/
- Ruby Brain - more documentation
  - http://www.rubybrain.com/

Rails Resources

- Ruby on Rails Home Page
  - http://rubyonrails.com/
- Ruby on Rails Guides
  - http://guides.rubyonrails.org/
- The Pragmatic Programmers (books)
  - http://www.pragmaticprogrammer.com/
- DB2 on Rails
  - http://db2onrails.com/

F11

Ruby on Rails Primer

John Maenpaa

Health Care Service Corporation

jmaenpaa@acm.org