Ruby on Rails
Migrations, Models, and Unit Tests

CPS353 Internet Programming

Simon Miner

Gordon College

Last Modified: 10/16/2013

Agenda

- Scripture (Philippians 1) and Prayer
- Check-in
- Ruby
- Ruby on Rails
- Milestone 5

Check-in

- (Brief) Discussions
  - Pat Gelsinger convocation
  - GC CSC Meeting on 10/24 with Win Mattina
- Updates
  - Syllabus
  - Programming Environment
- Homework 3
- Homework 4
- Milestone 4

Ruby

Continued from last week
(starting on slide 57)

Rails Strengths

- Uses Ruby -- a flexible OO scripting language
- Model-View-Controller pattern (MVC)
- Support for unit and functional testing (TDD)
- Good deployment support
- Steep learning curve
  - Use the docs
  - http://api.rubyonrails.org
  - http://guides.rubyonrails.org

Rails Philosophy

- Don't Repeat Yourself (DRY)
- Convention over configuration
- Individuals and interactions over processes and tools
- Working software over comprehensive documentation
- Customer collaboration over contract negotiation
- Responding to change over following a plan

Rails Development Environment

- Command Line
- Version Control (Git, Subversion)
- Continuous Integration (Jenkins)
- Editor (vim, Emacs, Notepad++?)

What to Look for in a Rails Editor

- Syntax highlighting support for Ruby, HTML, and .erb files
- Automatic (re)indentation of Ruby source code
- Shortcuts for common Ruby and Rails constructs
- Good file navigation -- lots of directories and files in Rails that you make small changes to
- Name completion

The steps of a Rails request (see p. 24)

- User enters URL (/say/hello)
- Rails matches the URI to a route pattern and sends the request to a controller and action method
  - say (controller) -> SayController in app/controllers/say_controler.rb
  - hello (action -> hello method executes, creating Time object, and assigning current time to @time
- Rails finds a view template for the result
  - Look in app/views for a directory with the same name as controller (say)
  - Finds file named after action (hello.html.erb)
- Parses ERB (embedded Ruby code) file with appropriate value substitution for (@time)
- Returns result to browser

The link_to method

- Dynamically creates an anchor/link tag to a URL or Rails controller / action pair
- Parameters
  - display text
  - URL (or expression to generate one)

```
<%= link_to "Hello", say_hello_path %>
```

Model/View/Controller (MVC) Pattern

- Splits application logic into 3 different pieces ("separation of concerns")
  - Model - maintains the application's state
  - View - generates the application's user interface
  - Controller - orchestrates the application, coordinate the actions of the model and view
- Rails imposes MVC on your application
  - Can be good as it handles interactions between components with sensible defaults so you don't have to

MVC Architecture

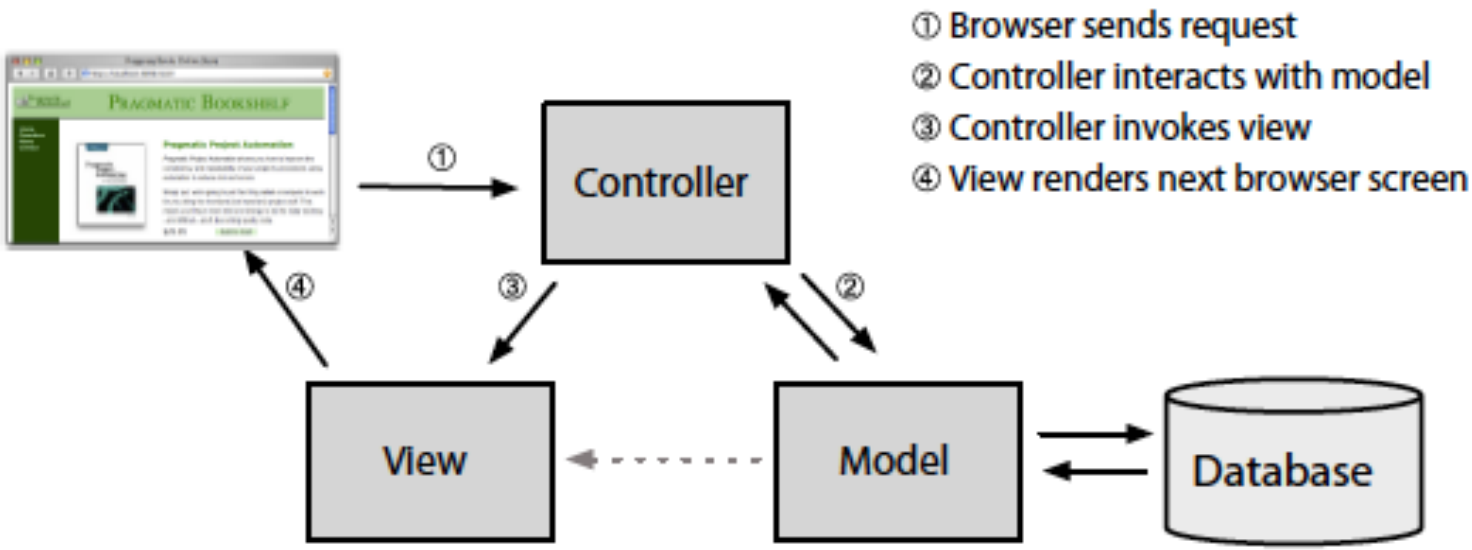


Figure 4—The Model-View-Controller architecture

Rails MVC Flow

- Request sent to a router
  - Maps it to a controller and action method
- Action method interacts with model to validate, retrieve, and store data
- Action method prepares data for view and invokes view code

Rails MVC Example

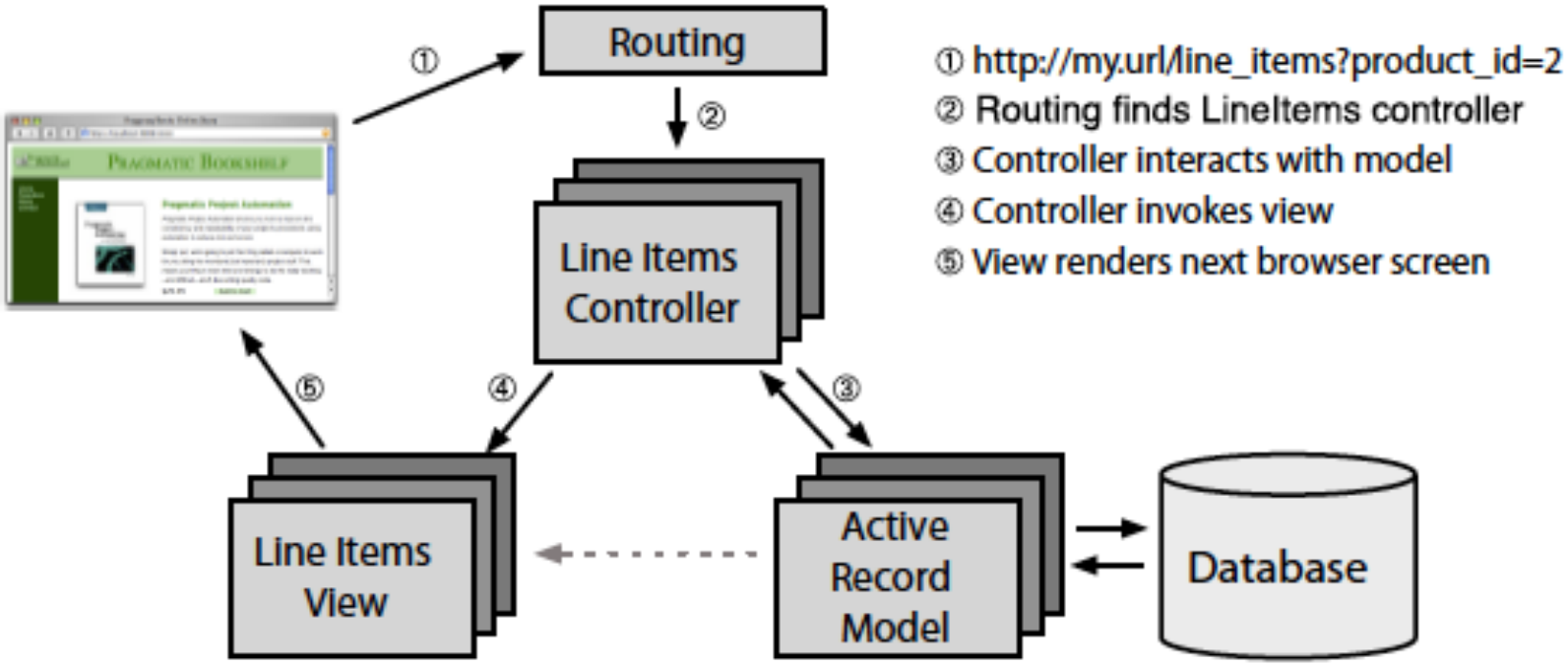


Figure 5—Rails and MVC

Object-Relational Mapping (ORM)

- Maps OO to relational data model
  - Addresses and minimizes "impedance mismatch" between these
- Classes => tables (Order class to orders table)
- Class method => table-level operation (Order.find(1) to "select from order where id = 1")
- Object => individual table row (order = Order.find is first row returned by select)
- Object attributes => column values (order.payment_type to payment_type column in orders table)
- Object methods => row operations (order.save to insert or update statement)
  - Model objects have other class and instance methods and attributes that do not map to the data store

Rails Active Record

- Base class for all ORM classes in Rails
- Follows ORM model using conventions
  - singular noun names for classes
  - plural names for tables
  - primary key columns and fields named id
  - date column and fields names end with "_on" ("created_on")
  - date/time column and field names end with "_at" ("modified_at")

rails Command

```
Usage: rails COMMAND [ARGS]

The most common rails commands are:
 generate    Generate new code (short-cut alias: "g")
 console     Start the Rails console (short-cut alias: "c")
 server      Start the Rails server (short-cut alias: "s")
 dbconsole   Start a console for the database specified in
             config/database.yml
             (short-cut alias: "db")
 new         Create a new Rails application.
             "rails new my_app" creates a
             new application called MyApp in "./my_app"
…
All commands can be run with -h (or --help) for more
information.
```

Specifying a Different Database

- -d/--database Tells rails to use a different database for the application (besides SQLite3)
  - Provided you've installed the necessary libraries (gems)
  - And that you have a database already setup

rails new my_app –d mysql

Rails Application Layout

- “rails new my_app” yields a directory tree structure like the following:
- app/ - Contains the controllers, models, views, helpers, mailers and assets for your application.
- bin/ - Contains the rails script that starts your app and can contain other scripts you use to deploy or run your application.
- config/ - Configure your application's runtime rules, routes, database, and more.
- config.ru - Rack configuration for Rack based servers used to start the application.
- db/ - Contains your current database schema, as well as the database migrations.
- Gemfile, Gemfile.lock - These files allow you to specify what gem dependencies are needed for your Rails application. These files are used by the Bundler gem

Rails Application Layout (continued)

- lib/ - Extended modules for your application.
- log/ - Application log files.
- public/ - The only folder seen to the world as-is. Contains the static files and compiled assets.
- Rakefile - This file locates and loads tasks that can be run from the command line. The task definitions are defined throughout the components of Rails. Rather than changing Rakefile, you should add your own tasks by adding files to the lib/tasks directory of your application
- README.rdoc - This is a brief instruction manual for your application. You should edit this file to tell others what your application does, how to set it up, and so on.
- test/ - Unit tests, fixtures, and other test apparatus.
- tmp/ - Temporary files (like cache, pid and session files)
- vendor/ - A place for all third-party code. In a typical Rails application, this includes Ruby Gems and the Rails source code (if you optionally install it into your project).

Setting up MySQL database access

rails new my_app –d mysql

- Edit my_app/config/database.yml
  - Each database (development, test, and production) has a block like this
  - Need to update credentials

```
development:
  adapter: mysql2
  database: my_app_development
  encoding: utf8
  username: root
  password:
  socket: /tmp/mysql.sock
  pool: 5
```

rake Command

- Ruby's version of make – executes a set of tasks on your application

```
rake --tasks
rake about                  # List versions of all Rails
frameworks and the environment
rake assets:clean           # Remove compiled assets
rake assets:precompile      # Compile all the assets
named in config.assets.precompile
rake db:create              # Create the database from
config/database.yml for the current Rails.env
…
rake log:clear              # Truncates all *.log files
in log/ to zero bytes (specify which logs with
LOGS=test,development)
…
```

Migrations

- Represent a change we want to make to the database
- Expressed in a database-agnostic form (in Ruby code)
- Can modify both the schema of and data within the database
- Migrations can be applied and rolled back
- Generated along with models or on their own
  - Reside in db/migrate

Migration File Names

- Example: 20131010180137.create_restaurant.rb
- Sortable date/time stamp containing
  - 4-digit year
  - 2-digit month
  - 2-digit day
  - 2-digit hour (military time)
  - 2-digit minute
  - 2-digit second
- Description of migration (assigned by generator for models or specified directly).
  - The description in the file name can actually provide adequate information for Rails to generate its code
- .rb extension since it's Ruby code

rake db:migrate

- By default, brings the schema up to current version
- Force schema to a specific version with version parameter
  - Will try to rollback changes as possible
- Redo migrations by one or more steps
- Every Rails database has a schema_migrations table to track which migrations have been applied
  - If there are migration files whose timestamps are not in this table, “rake db:migrate” will apply them
  - “rake db:migrate:status” to check the status of migrations

Migration Class

- Inherits from ActiveRecord::Migration
- Methods
  - up – apply a migration
  - down – roll back a migration
  - change - apply or rollback a migration
    - Rails decides
    - Provided the migration is reversible

Support Column Data Types

- :binary
- :boolean
- :date
- :datetime
- :decimal
- :float
- :integer
- :string
- :text
- :time
- :timestamp

Methods to Manage Tables

- create_table table, options/block
  - Block contains column and constraint definitions
  - Columns/constraints can also be defined by calls to separate methods
  - Options include force, temporary, and database-specific commands
  - Reverse is drop_table table
- rename_table table_name, new_name
  - Sometimes reversible (if model classes are not also renamed)

Table Defaults and Shortcuts

- create_table automatically includes an integer id in the table
- -- Can be customized
- t.timestamps adds created and last modified date/ timestamps
- -- created_at
- -- updated_at

create_table Example

```
class CreateProducts < ActiveRecord::Migration
  def change
    create_table :products do |t|
      t.string :title
      t.text :description
      t.string :image_url
      t.decimal :price, precision: 8, scale: 2
      t.timestamps
    end
  end
end
```

Methods to Manage Columns

- add_columns table, column, type, options
  - Options include null, limit, and default
  - Reverse is remove_column table, column
- rename_column table, column_name, new_column_name
  - Reversible
- change_column table, column, type, options
  - May not be reversible (i.e. redefining to a less restrictive type, removing a constraint)

Native SQL

- execute method allows for custom database commands
  - SQL is specific to database
  - Not reversible
- Example: module to define foreign keys
  - See

Undoing the Database

- Rollback the previous migration(s)

```
rake db:rollback
```

- Reset the database entirely and the rebuild it

```
rake db:reset
```

The WEBrick server

- Comes with Rails, good for testing in development
  - rails server -p/--port <port>
  - Default port is 3000
- Each member of the class will be assigned a unique port to use on ips.cs.gordon.edu
  - http://ips.cs.gordon.edu:<port>/...
- Ctrl/Command-C to stop the server
- -d flag makes server run in the background as a daemon

rails generate

- Scaffold
  - Sets up migration, model, controller, views, tests, and more for a given resource
  - rails generate scaffold resource field:type …
- Controller
  - Sets up a controller with specified actions
  - rails generate controller ControllerName action …
- Model
  - rails generate model ModelName field:type …
- And more…

Model Classes

```
class Product < ActiveRecord::Base
end
```

- Subclasses of Active Record
  - Located in app/models
- Do not contain list of the model's attributes
  - Can get them via ModelClass.column_name
  - Can get individual attribute definitions via ModelClass.columns_hash['attribute_name']
- Setting an attribute does not write its value to the database
  - Need to call object.save (object.save! throws exception if there's an error)

SQL Type to Ruby Class Map

SQL Type	Ruby Class
Int, integer	Fixnum
Float, double	Float
decimal, numeric	BigDecimal
char, varchar, string	String
clob, blob, text	String
interval, date	Date
datetime, time	Time
boolean	object.attribute? method

Columns/Attributes Added by Rails

- id – primary key attribute
- created_at, created_on, updated_at, updated_on
  - Date/timestamps that are automatically modified when a record is created or updated
  - "_on" for date stamps
  - "_at" for timestamps
- <table>_id – foreign key column to another table
- <table>_count – count of related rows in table referenced by foreign key

Database CRUD

Operation	Sample code

Create
`new` constructors can:
• Return an object to use
• Take a block
• Take a hash of data
`create` builds one or more records
	`Order.new do |o|`
	`  o.name = "Aardvark"`
	`  # . . .`
	`  o.save`
	`end`
	`Order.create( ... )`

Read – supports find, select, joins, where, order, limit, offset, group, find_by_sql (native SQL)	`Order.find`
	`Order.where(name: params[:name])`
	`Order.where(name: "cat").order(:age)`

Update	• `= Order.find(123)`
	`o.name("aardvark")`
	`o.save`
	`o.update(name: "aardvark")`

Delete	`Order.delete(123)` `o.destroy`

Active Record Associations

- Define relationships between model classes
- One-to-one relationship
  - belongs_to – connotes subordination to an instance of another model
  - has_one – connotes ownership of a single instance of another model
  - has_one :through – connotes ownership of a single instance of a model through another model
- One-to-many relationship
  - has_many – connotes ownership of (potentially) multiple instances of another model
- Many-to-many relationship
  - has_many :through – connotes ownership of (potentially) many instances of a model through another model
  - has_and_belongs_to_many – sets up many-to-many relationship without an intervening model

belongs_to

```
class Order < ActiveRecord::Base
  belongs_to :customer
end

class CreateOrders < ActiveRecord::Migration
  def change
    create_table :orders do |t|
      t.belongs_to :customer
      t.datetime :order_date
      t.timestamps
    end
  end
end
```

has_many

```
class Customer < ActiveRecord::Base
  has_many :orders
end

class CreateCustomers < ActiveRecord::Migration
  def change
    create_table :customers do |t|
      t.string :name
      t.timestamps
    end
  end
end
```

has_many :through

```
class Animal < ActiveRecord::Base
  has_many :pursuits,
    :foreign_key => 'predator_id',
    :class_name => 'Hunt',
    :dependent => :destroy
  has_many :preys, :through => :pursuits
  has_many :escapes,
    :foreign_key => 'prey_id',
    :class_name => 'Hunt',
    :dependent => :destroy
  has_many :predators, :through => :escapes
end
class Hunt < ActiveRecord::Base
  belongs_to :predator, :class_name => "Animal"
  belongs_to :prey, :class_name => "Animal"
end
```

Generating Associations

- Relationships can be set up when scaffolds or models are being generated

rails generate scaffold LineItem product:references cart:belongs_to

- Generates this model class

```
class LineItem < ActiveRecord::Base
  belongs_to :product
  belongs_to :cart
end
```

- Need to manually define inverse (has_many) relationships in Product and Cart models

Building Relationships

- The build method constructs a relationship between two associated entities
  - Rails can do this automatically from either entity

```
product = Product.find(params[:product_id])
@line_item = @cart.line_items.build(product: product)
```

Seed Data

- Rails lets you initialize your database models with data
  - Add code to create records (via model classes) to db/seeds.rb
- Run rake db:seed

```
cities = City.create!(
  [{ name: 'Chicago' }, { name: 'Copenhagen' }]
)
Mayor.create!(name: 'Emanuel', city: cities.first)
```

- Using create! throws an exception if there's a problem
  - Easier to debug than create (without the !)

Validation

- All validation in Rails is done in the model classes
  - Model is the “gatekeeper” to the database
  - All data goes through the model before being stored
    - Through a web form
    - Programmatic interface in the application
    - CLI program
  - More reliable than client-side validation
  - Less storage-engine dependent than in-database validation
- Validation occurs as part of these model methods
  - create, create!
  - save, save!
  - update
  - update_attributes, update_attributes!
  - Skipped for some model methods (update_all, update_attribute, etc.)
  - Can be disabled by passing :validate => false to a method

validates() method

- Checks one or more model fields against one or more conditions
  - validates attributes, helpers/options
- Examples
  - validates :title, :description, :image_url, presence: true
  - validates :price, numericality: {greater_than_or_equal_to: 0.01}
  - validates :title, uniqueness: true
  - validates :image_url, allow_blank: true, format: {
    with: %r{\.(gif|jpg|png)\Z}i,
    message: 'must be a URL for GIF, JPG or PNG image.'
    }
  - validates :terms, acceptance: true
  - validates :password, confirmation: true
  - validates :username, exclusion: { in: %w(admin superuser) }
  - validates :age, inclusion: { in: 0..9 }
  - validates :first_name, length: { maximum: 30 }

Validation Helpers

- acceptance
- validate_associated
- confirmation
- exclusion
- format
- Inclusion
- length
- numericalilty
- presence
- uniqueness
- validates_with
- validates_each

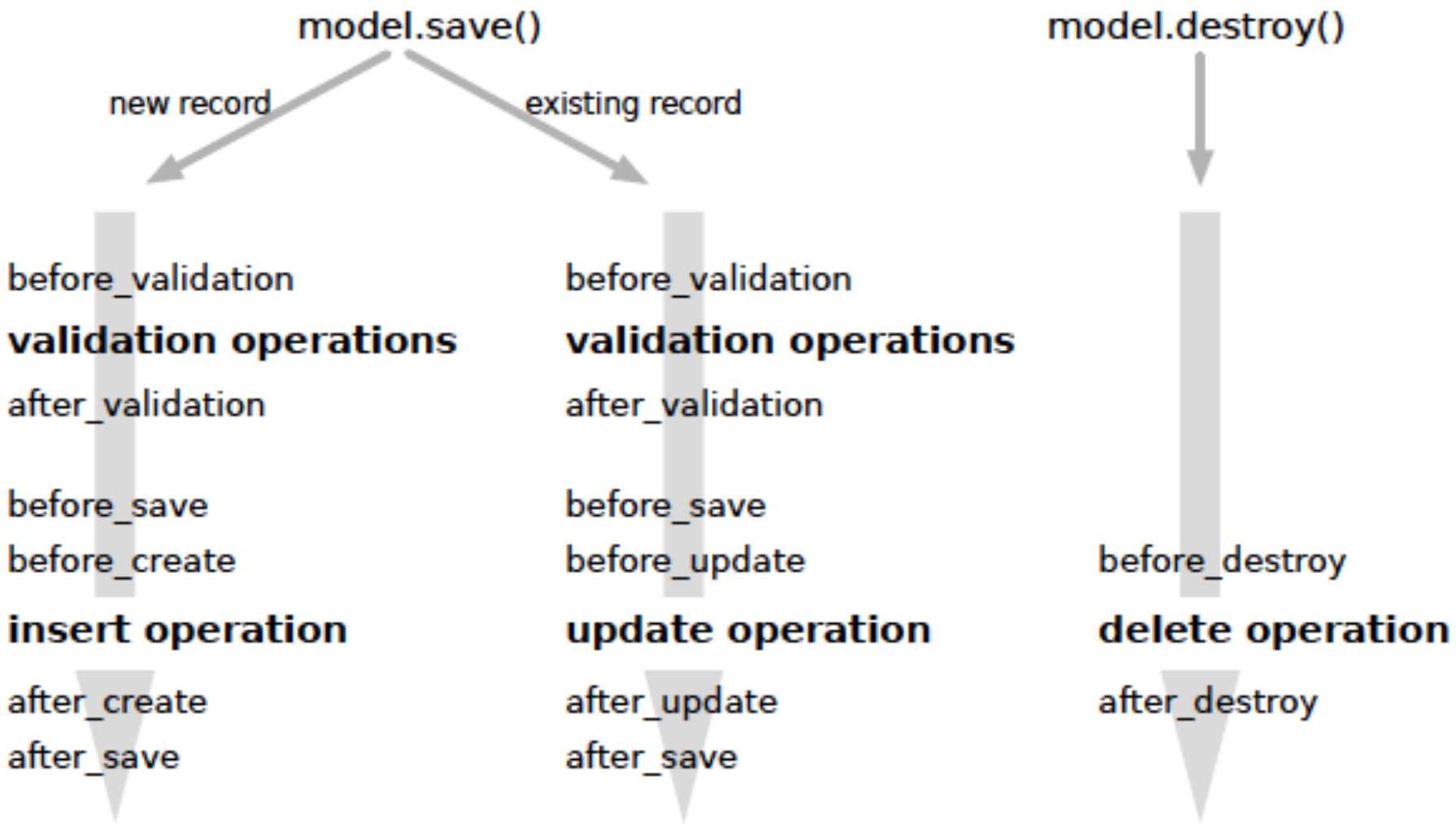
Common Validator Options

- :allow_nil – allow nil values
- :allow_blank – allow blank values
- :message – custom error message
- :on – when to run the validation
  - Defaults to :save (both create and update)
  - Can be set to :create or :update

Model Validation Example

```
class Product < ActiveRecord::Base
  validates :title, :description, :image_url,
    presence: true
  validates :price, numericality:
    {greater_than_or_equal_to: 0.01}
  validates :title, uniqueness: true
  validates :image_url, allow_blank: true, format: {
    with: %r{\.(gif|jpg|png)\Z}i,
    message: 'must be a URL for GIF, JPG or PNG image.'
  }
end
```

Active Record Callbacks



Example Callback and Hook Method

```
class Product < ActiveRecord::Base
  has_many :line_items

  before_destroy :ensure_not_referenced_by_any_line_item
  #...
  private
  # ensure that no line items reference this product
  def ensure_not_referenced_by_any_line_item
    if line_items.empty?
      return true
    else
      errors.add(:base, 'Line Items present')
      return false
    end
  end
end
```

Unit Tests

- Programs to test functionality of packages
  - Usually for stand-alone classes that get integrated into other code
    - i.e. models, business logic, or other libraries
- Rails automatically creates a unit test class for each generated model
  - Subclass of ActiveSupport:TestCase, which inherits from MiniTest framework that comes with Ruby
  - Located in test/models/*model*_test.rb
- Unit test classes use the test method to run tests
  - test message block
    - Usually a do/end block

Testing Methods and Attributes

- obj.valid? and obj.invalid? – whether a (model) object is valid or not
- obj.errors – list of errors generated by validation tests
  - obj.errors.any? – Are there any errors?
- assert – ensure that a test is true
  - assert test, message
- refute – ensure that a test is false
  - refute test, message

Available Assertions

- assert/refute
- assert_equal
- assert_same
- assert_nil
- assert_match
- assert_in_delta
- assert_throws
- assert_raises
- assert_instance_of
- assert_respond_to
- assert_operator
- assert_send
- flunk

Example Unit Test

```
test "product price must be positive" do
  product = Product.new(title: "My Book Title",
                        description: "yyy",
                        image_url: "zzz.jpg")
  product.price = -1
  assert product.invalid?
  assert_equal ["must be greater than or equal to 0.01"],
    product.errors[:price]
  product.price = 0
  assert product.invalid?
  assert_equal ["must be greater than or equal to 0.01"],
    product.errors[:price]
  product.price = 1
  assert product.valid?
end
```

Fixtures

- Sometimes unit tests need to work with data stored in the database
- The unit test class could create this itself
- A Rails fixture is a specification of the initial contents of a model under the test environment
  - Reside in test/fixtures/*table*.yml
  - Created when model is generated
  - Contains an entry for each row to be inserted into the database for testing
    - Named with a label we can use in our test code
      - Should be concise, self-explanatory, and natural – like a variable name
    - Includes a set of name/value pairs for the database table columns

Defining and Loading Fixtures

- In products.yml fixture file

```
ruby:
  title: Programming Ruby 1.9
  description:
    Ruby is the fastest growing and most exciting dynamic
    language out there. If you need to get working
    programs delivered fast, you should add Ruby to your
    toolbox.
  price: 49.50
  image_url: ruby.png
```

- In product_test.rb unit test class

```
class ProductTest < ActiveSupport::TestCase
  fixtures :products
  @product = products(:ruby)
  #...
end
```

  - Default behavior is to load all fixtures for each unit test

Test Database

- Fixtures get applied to the test environment database when a test begins
  - All records are deleted from the test database table
  - Then fixture records are created in the test database table
- Test database defined in /config/database.yml

```
test:
  adapter: mysql2
  database: my_app_test
  encoding: utf8
  username: root
  password:
  socket: /tmp/mysql.sock
  pool: 5
```

Using Fixtures

- Rails defines a method for each fixture name to access the record defined by that fixture

```
test "product is not valid without a unique title" do
  product = Product.new(
    title: products(:ruby).title,
    description: "yyy",
    price: 1,
    image_url: "fred.gif”
  )
  assert product.invalid?
  assert_equal ["has already been taken"],
    product.errors[:title]
end
```

Writing Effective Unit Tests

- Each test method exercises a single validation constraint
- Asserts that invalid values generate expected errors
- Asserts that valid values cause the object to validate
- Options to “rake test”
  - Verbose mode

```
rake test TESTOPTS=-v
```

  - Run tests from a single file

```
rake test TEST test/model/product_test.rb
```

Milestone 5