Lecture 11
CSE/ISE 337 Scripting Languages
Ruby 3 : Lecture 11

Instr: Prof Abid M. Malik
2/28/2023

Last Time

- Array
- Hash
- Methods
  - Proc, Block, and Lambda
- Iterators
- More on Range Operators
- I/O File operations

Today's Agenda

- **Exceptions**
- **Classes**
  - **Inheritance**
  - **Polymorphism**
  - **Singleton**
  - **Modules**
  - **Mixin**
- **Maths Library**
- **Symbol**

Some problems based on the last lecture

Exception in Ruby

- The execution and the exception always go together. If you are opening a file, which does not exist, then if you did not handle this situation properly, then your program is considered to be of bad quality.
- The program stops if an exception occurs. So exceptions are used to handle various type of errors, which may occur during a program execution and take appropriate action instead of halting program completely.
- Ruby provide a nice mechanism to handle exceptions. We enclose the code that could raise an exception in a begin/end block and use rescue clauses to tell Ruby the types of exceptions we want to handle.

Exceptions in Ruby

```
begin
# -
rescue OneTypeOfException
# -
rescue AnotherTypeOfException
# -
else
# Other exceptions
ensure
# Always will be executed
end
```

- Everything from begin to rescue is protected. If an exception occurs during the execution of this block of code, control is passed to the block between rescue and end.
- For each rescue clause in the begin block, Ruby compares the raised Exception against each of the parameters in turn. The match will succeed if the exception named in the rescue clause is the same as the type of the currently thrown exception, or is a superclass of that exception.
- In an event that an exception does not match any of the error types specified, we are allowed to use an else clause after all the rescue clauses.

Catch and Throw

- The raise-clause is good for abandoning execution; but catch-throw is used to jump out of deeply nested constructs

```
r = catch (:done)
do  i = 0
while i < 10
   throw :done, "Quitting now ..." if i == 5
   puts i
  i = i + 1
end
puts "still in catch"
end
puts r
```

When Ruby encounters a **throw**, it zips back up the call stack looking for a **catch** block with a matching symbol.

When it finds it, Ruby unwinds the stack to that point and terminates the block.

If the **throw** is called with the optional second parameter, that value is returned as the value of the **catch**

Exceptions in Ruby

- **Why we need them!**
- **What is the syntax!**
- **See code on Jupyter notebook**
  - **Ruby2 (jupyter notebook file)**

Object Oriented Programming in Ruby

Check the jupyter notebook notes (Ruby3)

1. **Basis of OOP**
2. **Instance Variables**
3. **Classes are always open for work!**
4. **Class Initialization (constructor)**
5. **Concept of Accessors**
6. **Class Variables**
7. **Class Methods**

8 Singleton Methods

In Ruby, a singleton method is a method that is defined on a specific instance of a class rather than on the class itself.

This means that the method is only available for that particular instance and not for other instances of the same class.

9 Inheritance

Inheritance is when a class inherits behavior from another class.

The class that is inheriting behavior is called the subclass and the class it inherits from is called the superclass.

Polymorphism

- Polymorphism is the ability of objects of different classes to be treated as objects of a common superclass.
- In the context of method overriding, polymorphism allows a subclass to provide a specific implementation of a method that is already defined in its superclass, thereby changing or extending the behavior of that method.

super keyword

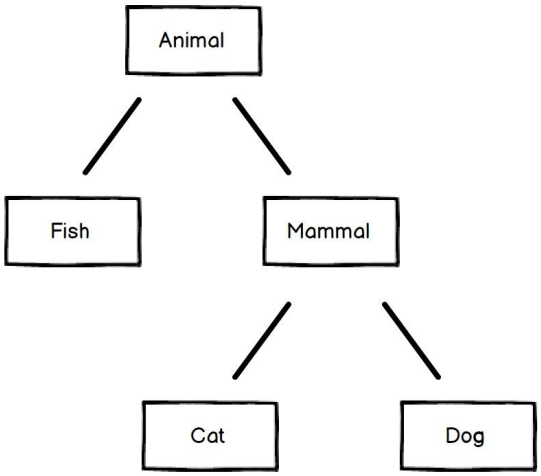
Ruby provides us with the `super` keyword to call methods earlier in the method lookup path.

When you call `super` from within a method, it searches the method lookup path for a method with the same name, then invokes it.

10 Modules

- Module is a container for code that provides a way to organize and encapsulate methods, constants, and other module definitions. Modules are similar to classes in that they serve as a way to group related code together, but they have some important differences:
  - No Instances:
  - No Inheritance Chain:
  - Namespacing:
  - Code Reusability:

10 Modules and concept of Mixin



Maths Module

Math module consists of the module methods for basic trigonometric and transcendental functions.

11 private, public, or protected

`private`, `public`, and `protected` are access control keywords used to control the visibility of methods within a class. They determine which methods are accessible from outside the class and which are not.

13 Symbols

Symbols were made to be used as *identifiers* (i.e., variables, method names, constant names).

In contrast, strings were made to be used as *data*.

Symbols vs strings

- **A string, in Ruby, is a *mutable* series of characters or bytes.**
- **Symbols, on the other hand, are *immutable* values. Just like the integer 2 is a value.**

Symbols are faster

Since, symbols are immutable values, working with symbols requires less memory, and it's just easier for computers to work with literal values than it is to work with complex objects. So, lookups for symbols are faster as well.

Questions!

Next time

- **Next week is the last week for Ruby Programming**
- **Regular Expressions**
  - **Ruby and Python support for regular expressions**
- **More practice examples**
- **HW2 (Ruby Programming) will be out next week**