CHAPTER 8:
USING OBJECTS

Introduction to Computer Science Using Ruby

Ruby: Philosophy & Implementation

- Ruby is the latest in the family of **Object Oriented Programming Languages**
- As such, its designer studied the problems and promises of past languages
- Ruby is an **extreme implementation** of such a language, containing large complexity on one hand and the ability to ignore any such complexity on the other hand

Ruby: Philosophy & Implementation

Eliminate **ANY** unnecessary statements, declarations and complexity

Startup has to be very **simple**

Complexity increase has to be **unlimited**

Development is supported by **interactive capability**

Portability is assured via an **interpretive implementation**

Classes

- Programs can be millions of lines of code
  - Eventually, they become very difficult to debug and maintain
- **Classes** are created to organize programs and data based on functionality

Objects

- Classes define the **characteristics** and **behaviors** of *objects* belonging to them
- Classes provide an **abstraction** of possible objects
- **Objects** are the instantiation of classes
  - They have a name and possess all the properties of the class
    - Example: Simple variables and their methods

Classes & Objects

- **Classes** are designed to separate key activities in a program
- **Objects** instantiated from classes provide the implementation of the program
  - Activities are isolated
  - Communicate information without knowing how it is produced
- Classes enable programs to be **compartmentalized**
  - Programmers can work at the same time on different classes without running into each other

Methods

- Classes have their own private chunks of data and actions
  - Actions that an object instantiated from a class may perform are referred to as **Methods** that belong to that Class

Built-in Classes & their Objects

- **Everything** in Ruby is an Object, even a simple variable
- As such, it has to be **instantiated** from a Class
- In Ruby, instantiation is many times done **automatically**, using "hidden" Class definitions
- This is one of the ways to eliminate declarations and various auxiliary and obscure statements

Built-in Classes

- A **Class** defines the characteristics and behaviors of an object
  - Contains the variables and the code necessary to implement the operations (Methods) of the object
- Examples of Built-in Classes:
  - Array
  - Fixnum
  - Float
  - String

Built-in Objects – Classes

- There are many more classes than these
- It is not required that you know HOW they do what they do, but it is required that you know WHAT they do, how to find them, and how to deploy them

Built-in Classes & their Objects

- In Ruby, instantiation can be done automatically using "hidden" Class definitions, or can be done explicitly, using the proper method

```
Example:   # Automatic creation – no Class name....
           arr = [1,2,3,"Wow"]
```

- This is an automatic creation of an Object (in this case: arr)
- No class name is used → Class is hidden → no method

Built-in Classes & their Objects

- In Ruby, instantiation is done automatically many times, using "hidden" Class definitions, or can be done explicitly.

```
Example:   # instantiate an object from the class Array
           arr = Array.new
```

- This is an explicit creation of an Object
  - In this case arr: instantiating it from the class Array using the built-in method "new"

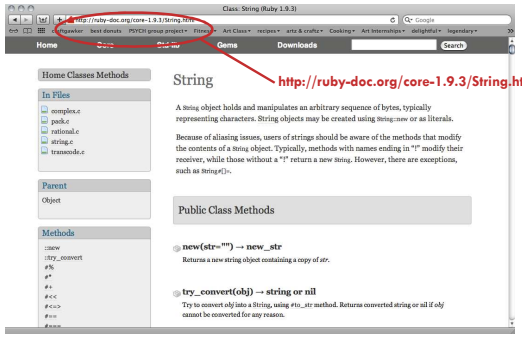
Built-in Classes & their Methods

- You can understand Class functionality by looking at the Ruby **API**, or **A**pplication **P**rogram(ming) **I**nterface
  - API allows the use of **built-in functionality** (that is, the built-in classes and their methods) without knowing the specifics of the implementation

Figure 8.1: String API Documentation

- The next slides show the API documentation for the String class
- It is taken straight out of the book – which is taken straight from the Ruby public information (with proper attribution)

Figure 8.1: String API Documentation



Built-in Methods

- The length method descriptor is "`str.length => integer`"
  - Means that the method will return an **integer**

```
irb(main):001:0> "hello".length => 5
```

Built-in Methods

- Not all methods are this simple:

str.index(substring [, offset]) => fixnum or nil
str.index(fixnum [, offset]) => fixnum or nil
str.index(regexp [, offset]) => fixnum or nil

Returns the index of the first occurrence of the given *substring*, character (*fixnum*), or pattern (*regexp*) in *str*. Returns `nil` if not found. If the second parameter is present, it specifies the position in the string to begin the search.

```
"hello".index('e')             #=> 1
"hello".index('lo')            #=> 3
"hello".index('a')             #=> nil
"hello".index(101)             #=> 1
"hello".index(/[aeiou]/, -3)   #=> 4
```


Parameter Passing

- Parameters are **data** supplied to a method (or to a Class – see later)
- See the API for the description of the built-in methods that require parameter(s) (variable(s) in parenthesis
- Methods with parameters send the value of the variable to the implementing code

Example 8.1: Parameter Passing

- Look at the made up *multiplier* method:

```
1 x = 3
2 y = x.multiplier(4)
3 puts "The number is: " + y.to_s
```

- It multiplies the value of the parameter by *x*
- Output: `The number is 12.`

Parameter Passing

- The method works like a black box
  - The program inside is not known and doesn't matter
  - Only the **function,** thus the **output,** is important

Figure 8.4: Black Box for Multiplier Method

Example 8.2: Parameter Passing

- Example of **Split** (an actual Ruby built-in method):
  - This method splits strings into **array elements** based on the parameter passed

```
1 my_string = "Good;day;sir!"
2 arr = my_string.split(";")
3 puts arr
4
5 # The following array is created:
6 # arr[0]: "Good"
7 # arr[1]: "day"
8 # arr[2]: "sir!"
```


Example 8.3: Split Example #2

- Change the parameter to "a":

```
1 my_string = "Good;day;sir!"
2 arr = my_string.split("a")
3 puts arr
4
5 # The following array is created:
6 # arr[0]: "Good;d"
7 # arr[1]: "y;sir!"
```

Output:

```
Good;d
y;sir!
```


Example 8.4: Split Example #3

- A parameter not found in the string will result in an array containing a string that isn't split

```
1 my_string = "Good;day;sir!"
2 arr = my_string.split("z")
3 puts arr
4
5 # The following array is created:
6 # arr[0]: "Good;day;sir!"
```

Output:

```
Good;day;sir!
```


Summary

- **Classes** define the characteristics and behaviors of objects belonging to the class
- **Objects** are instantiations of a class: they have a name and possess all the properties of the class, namely the variables and the methods
- The **application user interface,** or **API,** is an interface used to communicate with some underlying functionality
- **Parameter passing** is used to transfer information to an object