Constructors

• Writing a new class is simple!
• Example:
  
  ```ruby
  class Point
  end
  ```

• But we may want to initialize state (constructor)
  – initialize()
  – Example:
    
    ```ruby
    class Point
      def initialize(x, y)
        @x = x # the convention for instance variables
        @y = y # is @parameter_name
      end
    end
    ```
Instantiating New Objects

• We instantiate a new object by calling the new() method on the class we want to instantiate

• Example
  
  p = Point.new(2,3

• How do we get the @x of p?
  
  p.@x?
  
  p.x?
Accessing State

• Instance variables are private by default
• The instance variables for our Point class are 
  \( @x, @y \)
• To access them, we must write methods that return their value
  – Remember "encapsulation" from CSE 142/143
class Point
  def initialize(x, y)
    @x = x
    @y = y
  end

  def get_x
    @x
  end
end

p = Point.new(2, 3)
puts p.get_x  # get value of x by calling a method
class Point
  def initialize(x, y)
    @x = x
    @y = y
  end

  def x
    @x
  end
end

p = Point.new(2, 3)
p.puts p.x  # get value of instance variable by calling a method
Accessing State

• We do not need to write these methods by hand
• Example:
  ```ruby
  class Point
    attr_reader :x, :y
    def initialize(x, y)
      @x = x
      @y = y
    end
  end
  ```
• What if we want to assign values?
Accessing State

• To assign a value to \(@x\), we can write a method

• Example:
  ```ruby
  def set_x(x)
    @x = x
  end
  p.set_x(7)
  ```

• Similarly we can use attr_writer
  ```ruby
  attr_writer :x, :y
  ```
Accessing State

• If we want to read and write all of our instance variables, we can combine attr_reader and attr_writer to simplify our class, replacing them with attr_accessor

class Point
  attr_accessor :x, :y
  def initialize(x, y)
    @x = x
    @y = y
  end
end
Objects in erb

- Objects work as expected in erb
- We can include the class directly in the erb file within the code tags `<% ... %>`
- We can also save an external `.rb` file (Point.rb) and then require the class file in our `.erb` file (plot_points.erb)
  ```ruby
  require 'Point.rb'
  p = Point.new(3,5)
  ```
- The files should be in the same folder, or specify the path to the class file
Inheritance

• Ruby supports single inheritance

• This is similar to Java where one class can inherit the state and behavior of exactly one other class

• The parent class is known as the superclass, the child class is known as the subclass
Inheritance

```
Foo
  method1
  method2
  toString

Bar  Baz
  (method1)
  method2
  (toString)

  Mumble
    (method1)
    method2
    (toString)
```
Inheritance

```
Foo
- method1
- method2
- toString

Bar
- (method1)
- method2
- (toString)

Baz
- method1
- (method2)
- toString

Mumble
- (method1)
- method2
- (toString)
```
Public and Private Methods

• Methods are public by default
• Private methods are declared the same way as public methods (no keyword at the beginning of method like Java)
• Private methods are designated by an "area" of private methods
• They keyword "private" designates this area
• Any methods after "private" are private methods
Public and Private Methods

• Public – any class can use the methods
• Private – only this particular object can use these methods

• There is a middle ground... methods can be "protected"
• Protected – only objects of this class or its subclasses can use these methods
Modifying Class Behavior

• Ruby allows us to add or modify functionality to ANY class
• This includes built-in classes like Fixnum and String
• Lets allow Strings to add any object to it without having to say to_s

"hello" + 3  # instead of "hello" + 3.to_s