# 8.14. Creating an Abstract Method

## Problem

You want to define a method of a class, but leave it for subclasses to fill in the actual implementations.

## Solution

Define the method normally, but have it do nothing except raise a `NotImplementedError`:

```
class Shape2D
  def area
    raise NotImplementedError.
    new("#{self.class.name}#area is an
abstract method.")
  end
end

Shape2D.new.area
# NotImplementedError: Shape2D#area is an
abstract method.
```

A subclass can redefine the method with a concrete implementation:

```
class Square < Shape2D
  def initialize(length)
    @length = length
  end

  def area
    @length ** 2
  end
end

Square.new(10).area                          # => 100
```

## Discussion

Ruby doesn't have a built-in notion of an abstract method or class, and though it has many built-in classes that might be considered "abstract," it doesn't enforce this abstractness the way C++ and Java

do. For instance, you can instantiate an instance of `Object` or `Numeric`, even though those classes don't do anything by themselves.

In general, this is in the spirit of Ruby. But it's sometimes useful to define a superclass method that every subclass is expected to implement. The `NotImplementedError` error is the standard way of conveying that a method is not there, whether it's abstract or just an unimplemented stub.