Access control(scope) to various methods, data members, initialize methods.

Comparison of access controls of Java against Ruby:

If the method is declared private in Java, it can only be accessed by other methods within the same class. If a method is declared protected it can be accessed by other classes which exist within the same package as well as by subclasses of the class in a different package. When a method is public it is visible to everyone. In Java, access control visibility concept depends on where these classes lies in the inheritance/package hierarchy.

Whereas in Ruby, the inheritance hierarchy or the package/module don't fit. It's all about which object is the receiver of a method.

For a **private method in Ruby**, it can never be called with an explicit receiver. We can (only) call the private method with an implicit receiver.

This also means we can call a private method from within a class it is declared in as well as all subclasses of this class.

```
class Test1
  def main_method
    method_private
  end

  private
  def method_private
    puts "Inside methodPrivate for #{self.class}"
  end
end

class Test2 < Test1
  def main_method
    method_private
  end
end

Test1.new.main_method
Test2.new.main_method
```

```
Inside methodPrivate for Test1
Inside methodPrivate for Test2

class Test3 < Test1
  def main_method
    self.method_private #We were trying to call a private method with
an explicit receiver and if called in the same class with self would
fail.
  end
end

Test1.new.main_method
This will throw NoMethodError
```

You can never call the private method from outside the class hierarchy where it was defined.
Protected method can be called with an implicit receiver, as like private. In addition, **protected method can also be called by an explicit receiver (only) if the receiver is "self" or "an object of the same class".**

```
class Test1
  def main_method
    method_protected
  end

  protected
  def method_protected
    puts "InSide method_protected for #{self.class}"
  end
end

class Test2 < Test1
  def main_method
    method_protected # called by implicit receiver
  end
end
```

```
class Test3 < Test1
  def main_method
    self.method_protected # called by explicit receiver "an object of
the same class"
  end
end


InSide method_protected for Test1
InSide method_protected for Test2
InSide method_protected for Test3


class Test4 < Test1
  def main_method
    Test2.new.method_protected # "Test2.new is the same type of object
as self"
  end
end

Test4.new.main_method

class Test5
  def main_method
    Test2.new.method_protected
  end
end

Test5.new.main_method
This would fail as object Test5 is not subclass of Test1
Consider Public methods with maximum visibility
```

Summary

- Public: Public methods have maximum visibility
- Protected: Protected method can be called with an implicit receiver, as like private. In addition, protected method can also be called by an explicit receiver (only) if the receiver is "self" or "an object of the same class".

- Private: For a private method in Ruby, it can never be called with an explicit receiver. We can (only) call the private method with an implicit receiver. This also means we can call a private method from within a class it is declared in as well as all subclasses of this class.