Design Patterns in Ruby: Factory and Factory Method Pattern

Abram Hindle
abram.hindle@ualberta.ca
Department of Computing Science
University of Alberta
http://softwareprocess.es/

Design Patterns

- *Just because you have duck-typing doesn't mean you can ignore common OO idioms!*
- Design patterns communicate intent, so it is best if we have a similar understanding.
- OO is hard :(

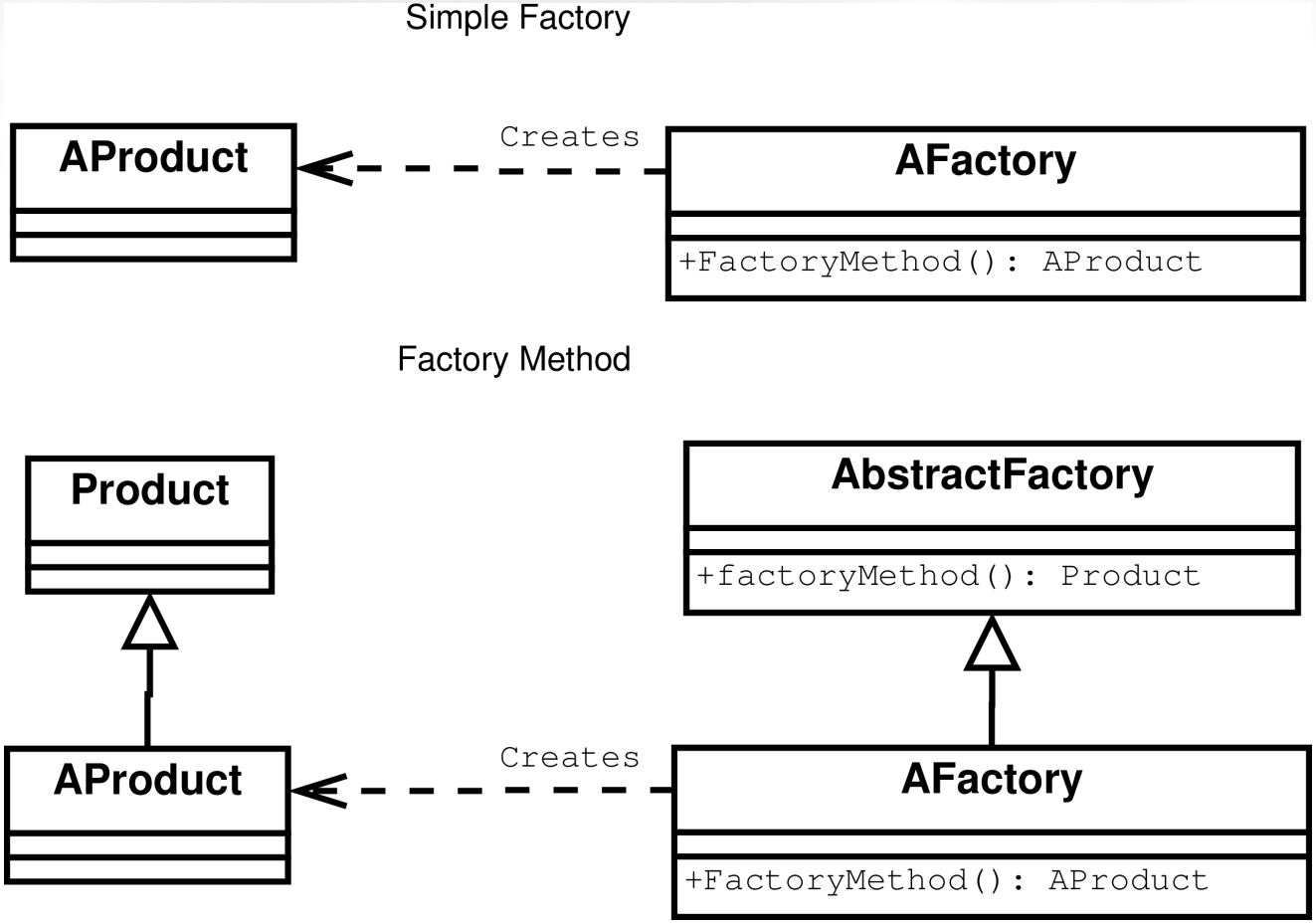
Construction

- For OO languages object construction and instantiation is often complicated.
- Multiple dependencies often make creating an object difficult.
- Constructor parameters are not good enough documentation about the assumptions or requirements of an object.
- Dependency injection makes constructors more complicated but aids testability.

Factory Pattern

- An object that builds other objects.
- Build an a variety of objects of the same type, based on some input.
- Construct objects based on an ID, string or context (not always the case).
- Should be an object, rather than a static class.
  - What if a lot of context needs to be maintained and referenced? Objects tend to be better than static methods.
- Does creation need to be centralized?

Factory Pattern



Example Factory

```
class CommandFactory
 def initialize(context)
  @context = context
 end
 def create_command(name)
  if (name == "Paste")
   return PasteCommand.new(@context)
  elsif (name == "Cut")
   return CutCommand.new(@context)
  else
   throw ("Could Not Construct "+name)
  end
 end
end
```

```
class PasteCommand
 def initialize(context)
 end
end
class CutCommand
 def initialize(context)
 end
end
```

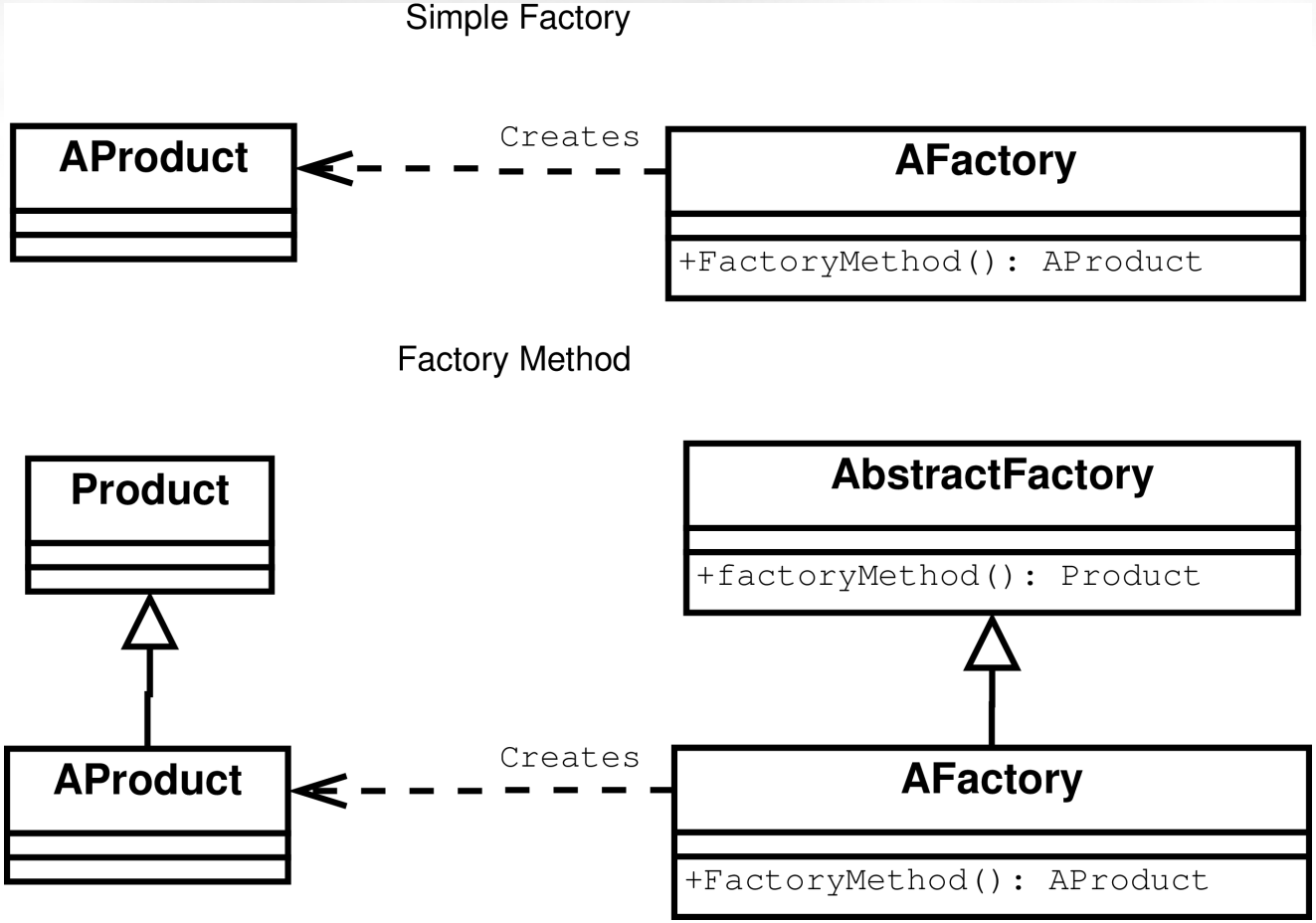
Factory and Ruby

- Hashes, eval, reflection can make Factories very fun and possibly really insecure in Ruby.
- Sometimes a set of objects might share a creational interface to be used by a factory.
- Some factories can be configured.
- In Java Abstract factories are often used to hide concrete implementations of a class.

Factory Method

- A method in a class that creates concrete objects but is meant to be overloaded.
- Allows for themed implementations.
- Used when the algorithm is constant but the theme of objects is not.
- Sometimes a set of objects are complicated to create and share the same “invocation”.

Factory Pattern



Factory Method Example

```
class Report
  def get_printer(type)
    throw "Abstract: Please Overload"
  end
  def print_report(type)
    printer = get_printer(type)
    printer.print_header(@header)
    printer.print_body(@body)
    printer.print_footer(@footer)
  end
end
class DraftReport < Report
  def get_printer(type)
    if (type == "ascii")
      return AsciiDraftPrinter.new()
    elsif (type == "postscript")
      return PostScriptDraftPrinter.new()
    end
  end
end
class BasicReport < Report
  def get_printer(type)
    if (type == "ascii")
      return AsciiPrinter.new()
    elsif (type == "postscript")
      return PostScriptPrinter.new()
    end
  end
end
```

```
class Printer
  def print_header(header)
  end
  def print_body(body)
  end
  def print_footer(footer)
  end
end
class AsciiPrinter < Printer
end
class AsciiDraftPrinter < AsciiPrinter
end
class PostScriptPrinter < Printer
end
class PostScriptDraftPrinter < PostScriptPrinter
end
```

Factory Method

- You have an algorithm that can vary based on the objects created.
- Objects might be difficult to create, or require long invocations, but share a common invocation.

Factory Resources

- Creational Design Patterns by Gregory Brown
  http://ur1.ca/ciohf
  http://blog.rubybestpractices.com/posts/gregory/0
- C2 on Factory Pattern
  http://c2.com/cgi/wiki?FactoryPattern
- Wikipedia on Factory Method Pattern

  https://en.wikipedia.org/wiki/Factory_method_pattern