```
@@classes = []
@@subject = ""

def ClassDiagram ( *options, &block )
  @@subject = options[0]
  # the following code defines the graph layout and properties in DOT
language
  puts "digraph G { "
  puts "    fontname=\"Bitstream Vera Sans\""
  puts "    rankdir=LR "
  puts "    layout=\"circo\" "
  puts "    node [fontname=\"Bitstream Vera Sans\", fontsize=10.0] "
  puts "    edge [fontname=\"Bitstream Vera Sans\", fontsize=8.0,
minlen=1, arrowsize=0.8, labeldistance=1.5, labelangle=35.0] "
  block.call

  puts "    // make the classes "
  # we analyze each class in the list of class and transform its
properties into DOT language
  @@classes.each do |c|
    if c.class == Klass or c.class == Interface
      if c.meth == [] and c.attr == [] then
        # we don't have any attributes or methods: show a box
        puts "    #{c.object_id} [shape=box, label=\"#{c.name}\"]"
      else
        # we have attributes or methods: show a record
        print "    #{c.object_id} [shape=record, label=\"#{c.name}| "
        c.attr.each { |a| print "#{a}\\l" }
        print " | "
        c.meth.each { |m| print "#{m}\\l" }
        puts "\"]"
      end
    end

    if c.class == Note then
      puts "    #{c.object_id} [shape=note, fontsize=6.0, label=\"#
{c.name}\", style=\"dashed\"]"
    end
    if c.class == Constraint then
      puts "    #{c.object_id} [shape=plaintext, fontsize=8.0,
label=\"#{c.name}\"]"
    end
  end

  puts "    // make the associations relations "
  # associations are stored in the classes, so we have to iterate
through the classes to access the links
  @@classes.each do |c|
    if c.assoc != nil
      # we go through all the associations stored in a class and look
at each of them
      c.assoc.each do |a|
        label    = a.center
        hlbl     = a.right
        tlbl     = a.left
        directed = a.directed
        lblStr   = "label=\"#{label}\", taillabel=\"#{tlbl} \",
headlabel=\"#{hlbl} \""
        if a.class == ASSOCIATION
          directed ? head = "vee" : head = "none"
          puts "    #{c.object_id} -> #{a.dst.object_id} [#{lblStr},
dir=\"both\", arrowhead=\"#{head}\", arrowtail=\"none\"] "
        end
        if a.class == AGGREGATION
          directed ? head = "vee" : head = "none"
          puts "    #{c.object_id} -> #{a.dst.object_id} [#{lblStr},
dir=\"both\", arrowhead=\"#{head}\", arrowtail=\"odiamond\"] "
        end
        if a.class == COMPOSITION
          directed ? head = "vee" : head = "none"
          puts "    #{c.object_id} -> #{a.dst.object_id} [#{lblStr},
dir=\"both\", arrowhead=\"#{head}\", arrowtail=\"diamond\"] "
        end
        if a.class == EXTENSION
          directed ? head = "vee" : head = "none"
          puts "    #{a.dst.object_id} -> #{c.object_id} [#{lblStr},
dir=\"both\", arrowhead=\"#{head}\", arrowtail=\"empty\"] "
        end
        if a.class == INTERFACE
          directed ? head = "empty" : head = "none"
          puts "    #{c.object_id} -> #{a.dst.object_id} [#{lblStr},
dir=\"both\", arrowhead=\"#{head}\", arrowtail=\"none\",
style=\"dashed\"] "
        end
      end
    end
  end
  puts "}"
end

def Class( name )
  return Klass.new name
end
```

```
def Interface( name )
  return Interface.new name
end

def Note( text )
  return Note.new( text )
end

def Constraint( text )
  return Constraint.new( text )
end

def Association ( klass, *options )
  return ASSOCIATION.new klass, *options
end

def Aggregation ( klass, *options )
  return AGGREGATION.new klass, *options
end

def Composition ( klass, *options )
  return COMPOSITION.new klass, *options
end

def Extension ( klass, *options )
  return EXTENSION.new klass, *options
end

class Klass
  attr_reader :name
  attr_reader :attr
  attr_reader :meth
  attr_reader :assoc
  def initialize ( name )
    if name == @@subject
      @name = "\\<subject\\> \n" << name
    else
      @name = name
    end
    @attr = []
    @meth = []
    @assoc = []
    @@classes << self
  end
  def attributes( *attr )
    attr.each { |a| @attr << a }
  end
  def methods( *meth )
```

```
    meth.each { |m| @meth << m }
  end
  def implements( klass, *options )
    links INTERFACE.new klass, {:dir => true}, *options
  end
  def extends( klass, *options )
    links EXTENSION.new klass, *options
  end
  def has( klass, *options )
    links AGGREGATION.new klass, *options
  end
  def creates( klass, *options )
    links ASSOCIATION.new klass, {:center => "<creates>", :dir =>
true}, *options
  end
  def uses( klass, *options )
    links ASSOCIATION.new klass, {:center => "<uses>", :dir => true},
*options
  end
  def acquires( klass, *options )
    links ASSOCIATION.new klass, {:center => "<acquires>", :dir =>
true}, *options
  end
  def owns( klass, *options )
    links COMPOSITION.new klass, *options
  end
  def connects( klass, *options )
    links ASSOCIATION.new klass, *options
  end
  def attaches( klass, *options )
    links INTERFACE.new klass, *options
  end
  def addNote( text )
    links INTERFACE.new Note text
  end
  def links( association )
    @assoc << association
  end
end

class Interface < Klass
  def initialize( name )
    super( "\\<interface\\> \n#{name}" )
  end
end
class Note < Klass
  def initialize( text )
    super
  end
end
class Constraint < Klass
  def initialize( text )
    super
  end
end

class ASSOCIATION
  attr_reader :dst
  attr_reader :left
  attr_reader :center
  attr_reader :right
  attr_reader :directed
  def initialize( dst, *options )
    @dst      = dst
    @directed = false
    if options != nil
      options.each do |entry|
        entry.each_pair do |k, v|
          case k
          when :center   then @center = v
          when :left     then @left   = v
          when :right    then @right  = v
          when :dir      then @directed = v
          when :directed then @directed = v
          else raise "#{k}: not a valid option"
          end
        end
      end
    end
  end
end

class AGGREGATION < ASSOCIATION
  def initialize( dst, *options )
    super
  end
end

class COMPOSITION < ASSOCIATION
  def initialize( dst, *options )
    super
  end
end

class EXTENSION < ASSOCIATION
  def initialize( dst, *options )
```

```ruby
    super
  end
end

class INTERFACE < ASSOCIATION
  def initialize( dst, *options )
    super
  end
end

def align( *nodes )
  print "    { rank=same "
  nodes.each { |n| print "#{n.object_id} " }
  puts  "}"
end
```