オブジェクト指向プログラミング入門（その３）

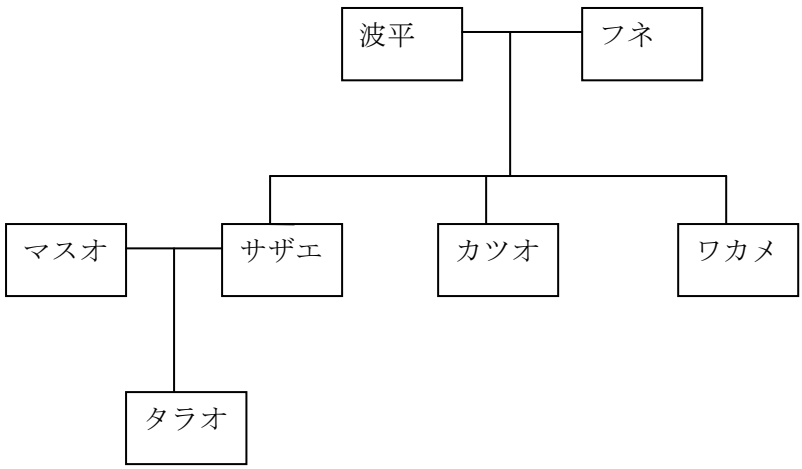
加賀　赳寛

さて、ここでは、オブジェクト指向らしいデータ構造を取上げてみましょう。

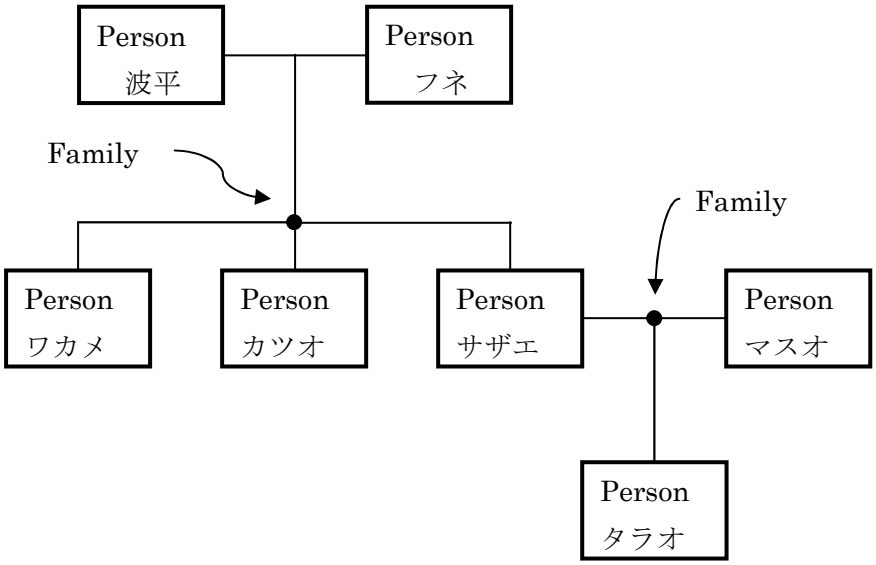
なお、本資料は「 ISBN4-7561-1909-3 オブジェクト指向データベース　石塚圭樹　著 」
http://www-6.ibm.com/jp/software/rational/library/books/bk_04.html　を参照させていただいています。

まずはサザエさんの家系図です。

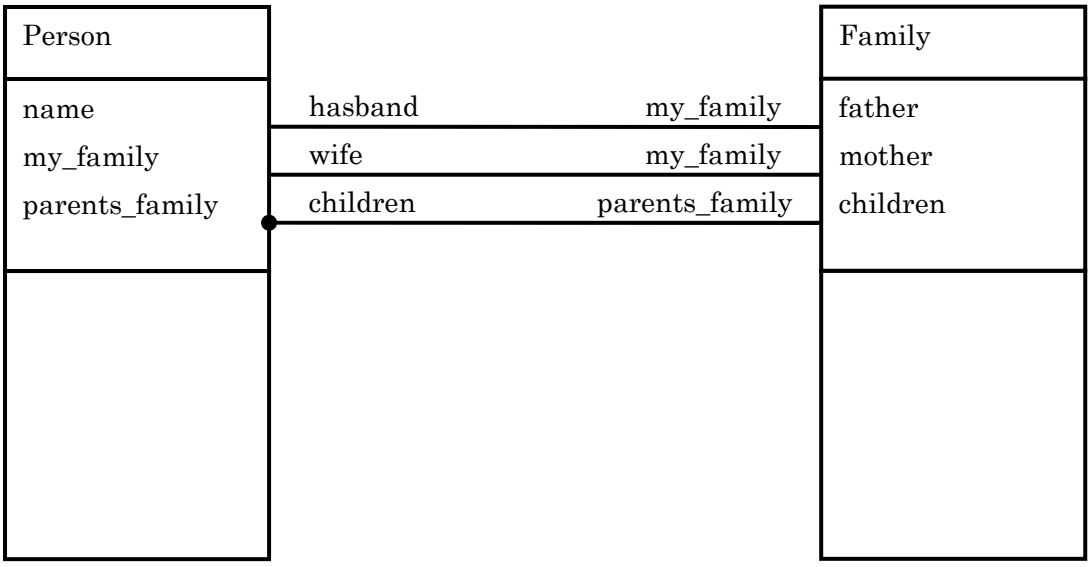
オブジェクト指向の例として、下記ＷＥＢにあるサザエさんの家系図
http://www.asahi-net.or.jp/~TK7M-ARI/sazae_family_tree.jpg
を対象にしてみましょう。下図にはその一部を抜き出しています。



クラスとして、個人を示す Person と家族（父、母、子供たち）を示す Family を用いて
家系図を表すことができます。

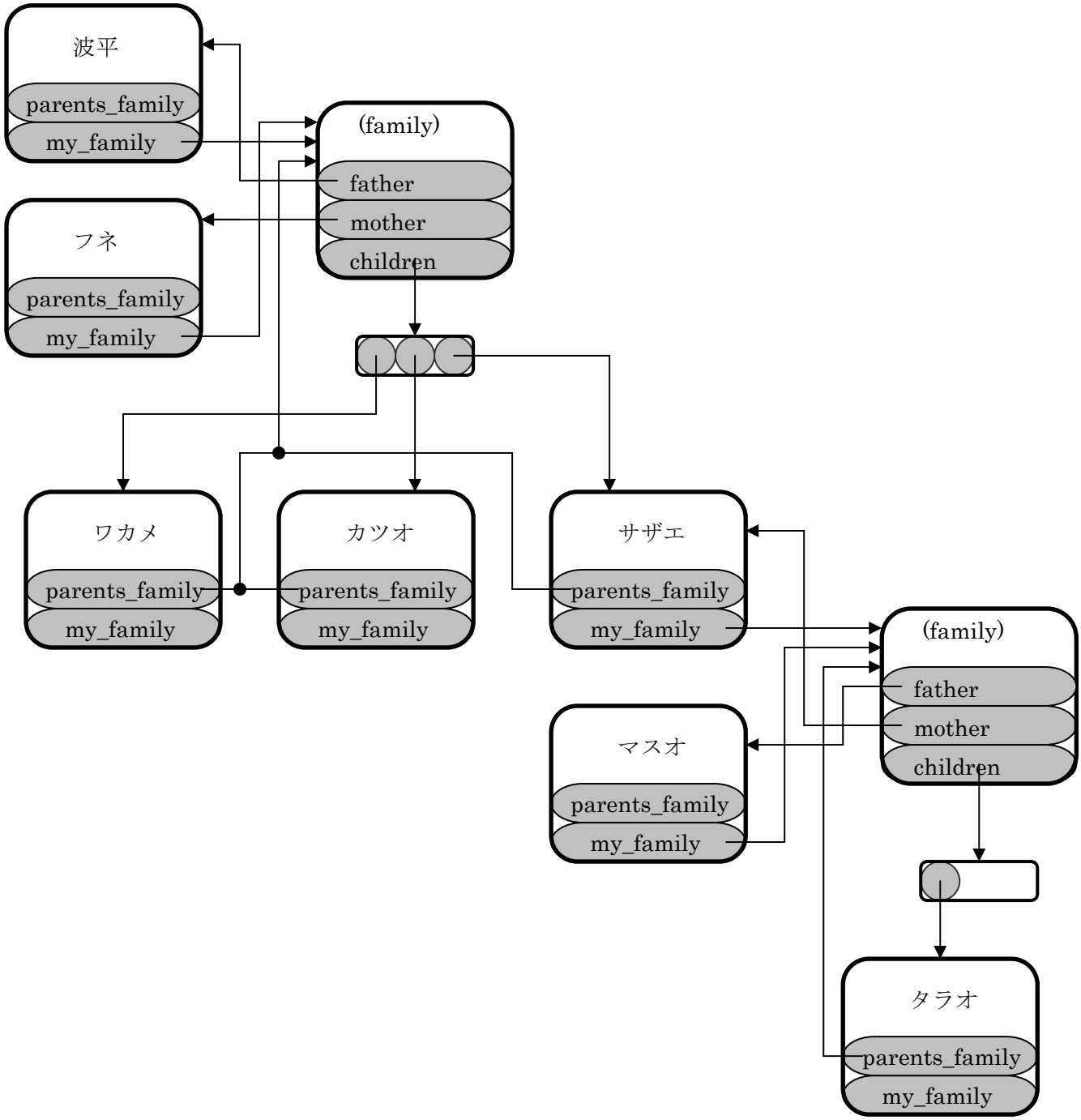


クラス図で書くと下図となります。



Person と Family の関係は Person から見ると自分の Family と両親の Family があります。
自分の Family の場合には Family の持つデータ father は夫であり、mother は妻です。
両親の Family の場合には Family の持つデータ father は自分の父親であり、mother は自分の母親です。また、この場合 children は自分の兄弟達（自分を含む）です。

具体的な家系図のオブジェクトのイメージは次のようになります。



コードは次のようになります。(C:¥rubytest に family.rb として格納)

```
class Family
  def initialize(father, mother)
    @father = father
    @mother = mother
    @children = Array.new
  end
  def insert_children(person)
    @children.push(person)
  end
  def remove_children(person)
    @children.delete(person)
  end
  def father
    @father
  end
  def mother
    @mother
  end
  def children
    @children
  end
end

class Person
  def initialize(name)
    @name = name          #名前
    @my_family = nil      #自分の家族
    @parents_family = nil #両親の家族
  end
  def set_family(family)
    @my_family = family
  end
  def set_parents(family)
    @parents_family = family
  end
```

```
  def name
    @name
  end
  def father                    #父親
    @parents_family.father
  end
  def mother                    #母親
    @parents_family.mother
  end
  def brothers                  #兄弟
    @parents_family.children  # 現状では自分も含んでいる
  end
  def hasband                   #夫
    @my_family.father
  end
  def wife                      #妻
    @my_family.mother
  end
  def children                  #子供
    @my_family.children
  end
end

namiheinoChichi = Person.new("波平の父")

namiheinoHaha = Person.new("波平の母")

namiheinochichinoFamily = Family.new(namiheinoChichi, namiheinoHaha)
namiheinoChichi.set_family(namiheinochichinoFamily)
namiheinoHaha.set_family(namiheinochichinoFamily)

namiheinoAne = Person.new("波平の姉")
namiheinochichinoFamily.insert_children(namiheinoAne)
namiheinoAne.set_parents(namiheinochichinoFamily)

namiheinoAni = Person.new("海平")
namiheinochichinoFamily.insert_children(namiheinoAni)
```

```
namiheinoAni.set_parents(namiheinochichinoFamily)

namihei = Person.new("波平")
namiheinochichinoFamily.insert_children(namihei)
namihei.set_parents(namiheinochichinoFamily)

fune = Person.new("フネ")

isonoFamily = Family.new(namihei, fune)  # 波平　フネ　結婚し磯野家誕生
namihei.set_family(isonoFamily)
fune.set_family(isonoFamily)

sazae = Person.new("サザエ")             # サザエ誕生
sazae.set_parents(isonoFamily)
isonoFamily.insert_children(sazae)

katsuo = Person.new("カツオ")            # カツオ誕生
katsuo.set_parents(isonoFamily)
isonoFamily.insert_children(katsuo)

wakame = Person.new("ワカメ")            # ワカメ誕生
wakame.set_parents(isonoFamily)
isonoFamily.insert_children(wakame)

masuo = Person.new("マスオ")

sazaeFamily = Family.new(masuo, sazae)  # サザエさん結婚、サザエ家誕生
sazae.set_family(sazaeFamily)           #
masuo.set_family(sazaeFamily)           #

puts "サザエさんの父は　" + sazae.father.name
puts "サザエさんの母は　" + sazae.mother.name
puts "サザエさんの夫は　" +sazae.hasband.name
puts "サザエさんの父方のおじいさんは　" + sazae.father.father.name
sazae.children.each {|child| puts "サザエさんの子供は　" + child.name }
sazae.brothers.each {|hito| puts "サザエさんの兄弟は　" + hito.name }
```

実行結果

```
C:¥rubytest>ruby -Ks family.rb
サザエさんの父は　波平
サザエさんの母は　フネ
サザエさんの夫は　マスオ
サザエさんの父方のおじいさんは　波平の父
サザエさんの兄弟は　サザエ
サザエさんの兄弟は　カツオ
サザエさんの兄弟は　ワカメ
```

練習問題

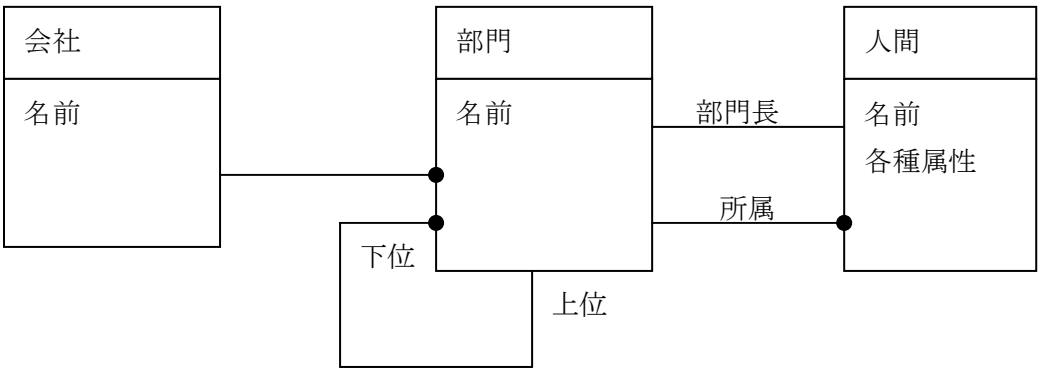
（1） 父がマスオさんで、母がサザエさんの家庭を作り、タラちゃん（名前はタラオ）を子供とするようにコードを追加してください。

（2） マスオさんの両親の家庭を作ってください。
なお、マスオさんの父の名前および母の名前は任意に決めてください。
マスオさんの両親の家庭にマスオさんとマスオさんの兄を子供として追加してください。
なお、マスオさんの兄の名前は任意に決めてください。

（3） タラちゃんのおじいさんを求めてください。勿論、父方と母方があります。

組織データベース

オブジェクト指向の例として、簡単な組織データベースを対象にしてみましょう。
組織データベースと言っても、ここでは３つのクラス、会社（Company)、部門（Division)、人間（People）との関連だけを扱うことのできる簡単な組織データベースを例題とします。この場合の組織データベースのクラス図を簡略表記すると下記となります。



部門は下位の部門を持つことができます。これは、事業部、システム部、課、係といった組織構造を表現できます。
部門には部門長と複数の人員が所属します。
このようなクラス間の関係により、会社の全社員とか、特定部門の所属員とかを必要に応じて取出すことができるようになります。
また、組織変更があった場合にも本質的な構造は変わらないので対応可能でしょう。

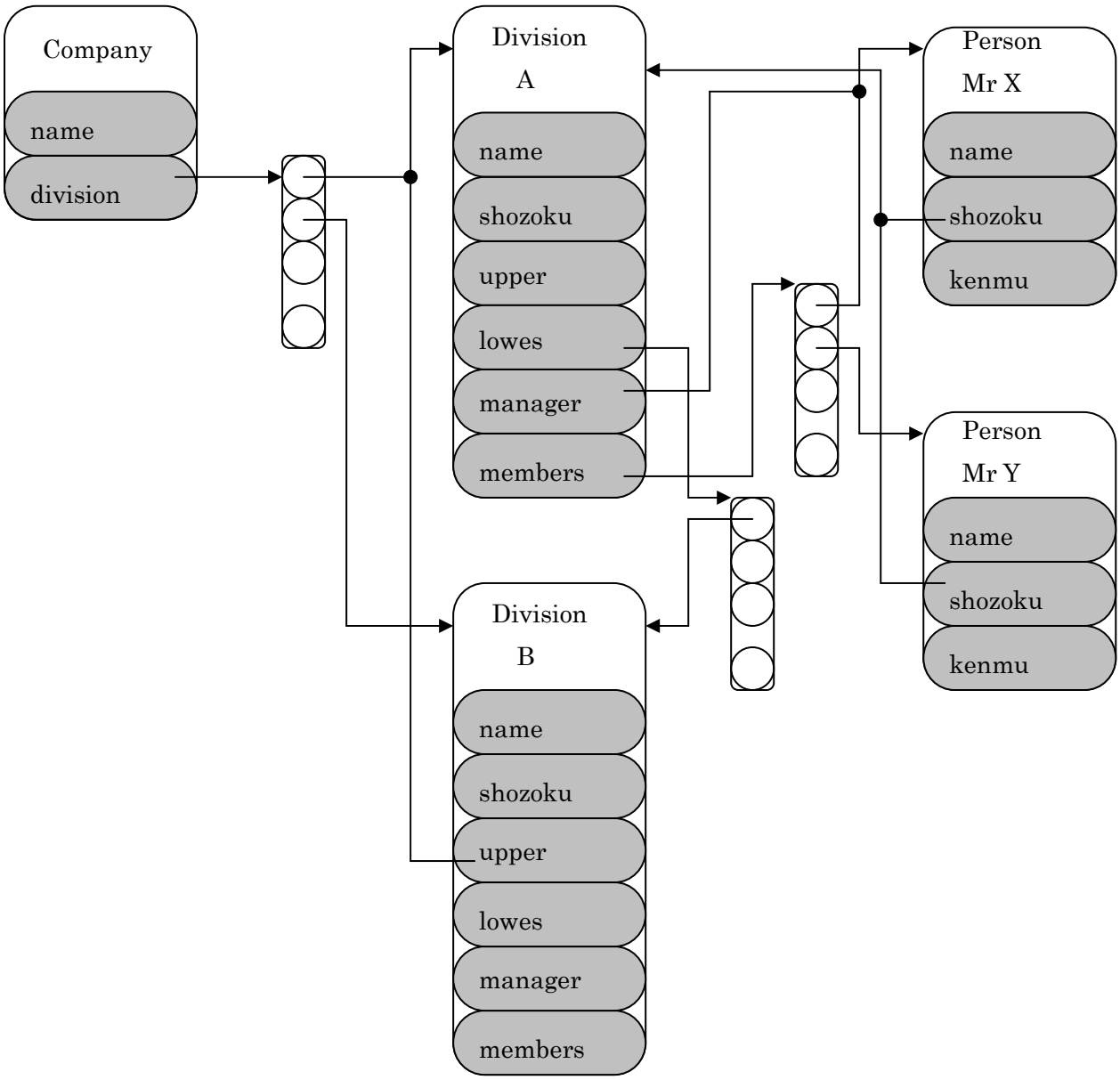
もう少し、具体的なオブジェクトのイメージは次ページに示します。

コーディングは各クラスを記載した後に、ＡＢ社　ソリューション事業部の組織の一部を表現してみました。

練習問題：

このコーディングを参考に各自の所属する部門のデータを作成して実行してみてください。

組織データベースにおけるオブジェクトのイメージ



```
$stdout = File.open("test.txt", 'w')
class Company
  def initialize(name)
    @name = name
    @division = Array.new
  end
  def create_division(division)
    @division.push(division)
  end
  def remove_division(division)
    @division.delete(division)
  end
end

class Division
  def initialize(name, shozoku)
    @name = name
    @shozoku = shozoku　#所属する会社
    @upper = nil
    @lowers = Array.new
    @manager = nil
    @members = Array.new
  end
  def set_name
  end
  def name
    @name
  end
  def set_upper(division)
    @upper = division
  end
  def upper
    @upper
  end
  def insert(division)
    @lowers.push(division)
```

```
  end
  def remove(division)
    @lowers.delete(division)
  end
  def set_manager(person)
    @manager = person
  end
  def manager
    @manager
  end
  def add(person)
    @members.push(person)
  end
  def remove(person)
    @members.delete(person)
  end
  def disp                  # 表示メソッド（基本的部分のみの仮版）
    puts @name
    @members.each {|member| puts member.name }
    @lowers.each {|division| division.disp }          # 注目してください!!!
  end
end


class Person
  def initialize(name)
    @name = name
    @shozoku = nil
    @kenmu_shozoku = nil
  end
  def name
    @name
  end
  def set_shozoku(division)
    @shozoku = division
  end
  def shozoku
```

```
    @shozoku
  end
  def set_kenmu_shozoku(division)
    @kenmu_shozoku = division
  end
  def kenmu_shozoku
    @kenmu_shozoku
  end
end



ab_company = Company.new("AB")
sol_sys = Division.new("ソリューション事業部", ab_company)
ab_company.create_division(sol_sys)
j_san = Person.new("Mr J")
power_sys.set_manager(j_san)
j_san.set_shozoku(sol_sys)
k_san = Person.new("Mr K")
k_san.set_shozoku(sol_sys)
h_san = Person.new("Mr H")
hayashi.set_shozoku(sol_sys)
g_san = Person.new("Mr G")
g_san.set_shozoku(sol_sys)
sol_sys.add(j_sani)
sol_sys.add(k_san)
sol_sys.add(h_san)
sol_sys.add(g_san)

planDept = Division.new("企画室", ab_company )
planDept.set_upper(sol_sys)

systemDept = Division.new("システム部", ab_company)
systemDept.set_upper(sol_sys)

developDept = Division.new("開発センター", ab_company)
developDept.set_upper(sol_sys)
```

```
sol_sys.insert(planDept)
sol_sys.insert(systemDept)
sol_sys.insert(developDept)

developsysG = Division.new("システム G", ab_company)
developsysG.set_upper(developDept)

developQ = Division.new("品質管理課", ab_company)
developQ.set_upper(developDept)

developDept.insert(developsysG)
developDept.insert(developQ)

m_san = Person.new("Mr M")
m_san.set_shozoku(planDept)
planDept.set_manager(m_san)
planDept.add(m_san)

k_san.set_kenmu_shozoku(systemDept)
systemDept.set_manager(k_san)
systemDept.add(k_san)

y_san = Person.new("Mr Y")
y_san.set_shozoku(developDept)
developDept.set_manager(y_san)
developDept.add(y_san)

t_san = Person.new("Mr T")
t_san.set_shozoku(developsysG)
developsysG.set_manager(t_san)
developsysG.add(t_san)

n_san = Person.new("Mr N")
n_san.set_shozoku(developQ)
developQ.set_manager(n_san)
developQ.add(n_san)
```

```
puts "===ソリューション事業部のメンバーを表示する==="
sol_sys.disp

puts "===開発センターのメンバーを表示する==="
developDept.disp

# ----------プログラムはここまで----------
```

実行結果

```
===ソリューション事業部のメンバーを表示する===
系統ソリューション事業部
Mr J
Mr K
Mr H
Mr G
企画室
Mr M
システム部
Mr K
開発センター
Mr Y
システムG
Mr T
品質管理課
Mr N
===開発センターのメンバーを表示する===
開発センター
Mr Y
システムG
Mr T
品質管理課
Mr N
```