Lab Tutorial 5A

Binary Search Tree

Nama berkas kode sumber	:	SDA<npm>L5A.java
Batas waktu eksekusi program	:	3 detik / kasus uji
Batas memori program	:	256 MB / kasus uji

Binary Search Tree merupakan sebuah struktur data yang menyimpan objek yang comparable secara dinamis, yaitu dapat mengubah posisi objek sedemikian hingga pada sebuah tree, semua elemen pada subtree kiri bernilai lebih kecil dari elemen pada root dan semua elemen pada subtree kanan bernilai lebih besar dari elemen pada root.

Terdapat beberapa operasi dasar pada binary search tree, yaitu add, remove, min, max, dan contains.

Operasi add

Operasi add adalah operasi untuk menambahkan suatu objek ke dalam tree. Objek dimasukkan ke dalam tree sedemikian hingga menjaga struktur dari tree.

Misalnya, terdapat sejumlah string sebagai berikut :

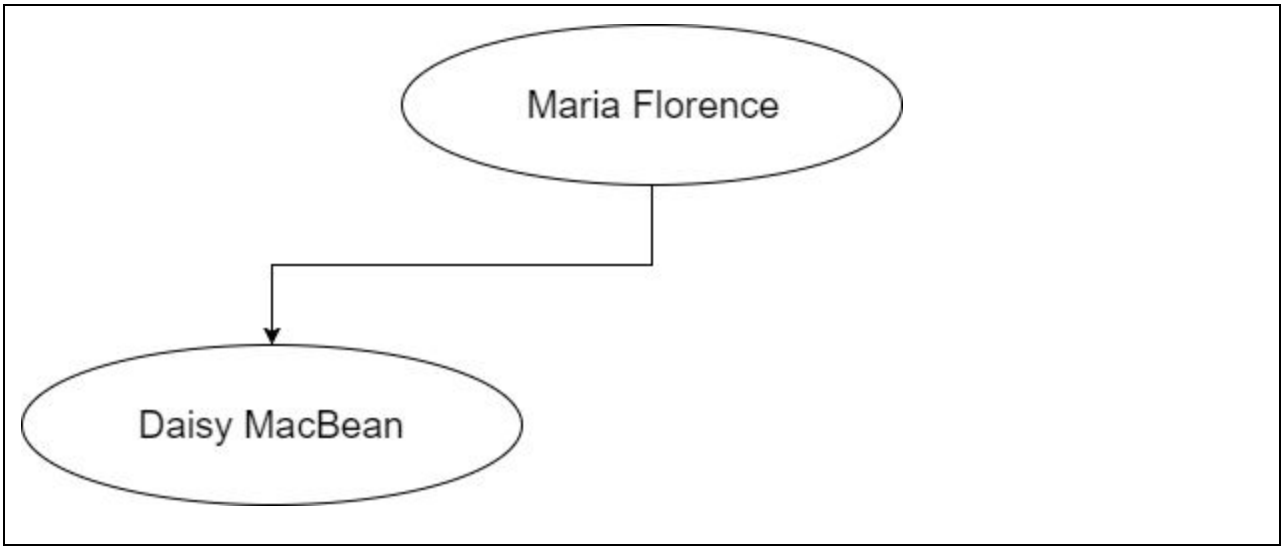
```
Maria Florence
Daisy MacBean
Sarah Clay
Ange le Carre
Charlotte
Elizabeth Cassandra Austin
Chise Todo
Shirley Collins
Jane McLean
Beatrice
Meifan Zhu
```

Problem setter : **JCA**

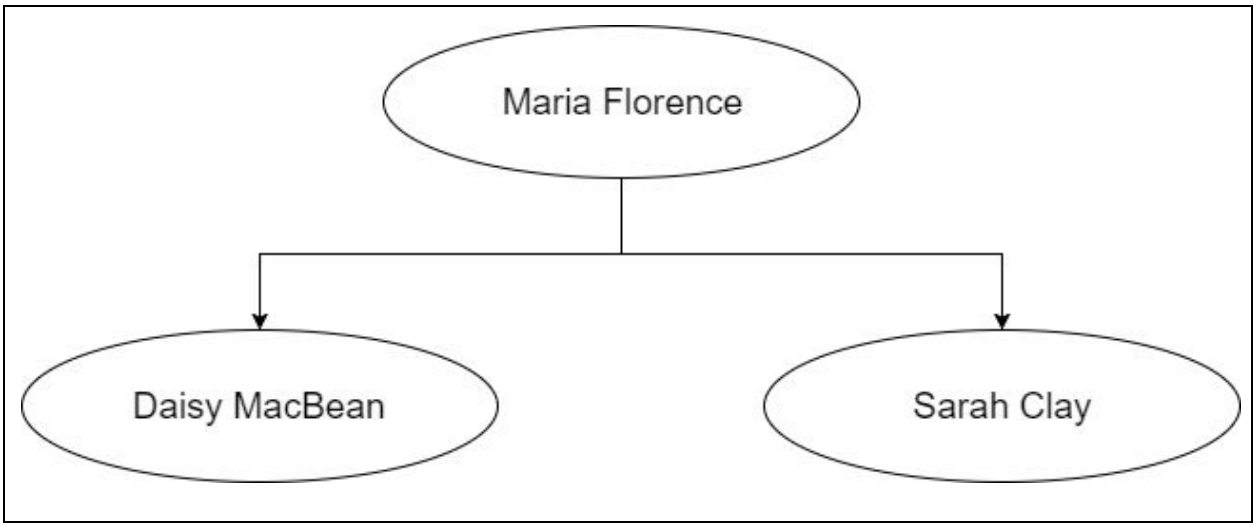
```
Elena Clay
Alicia
Ange le Carre
Chise Todo
Beatrice
Daisy MacBean
Charlotte
```

String-string tersebut akan dimasukkan ke dalam tree. Setelah memasukkan string “Maria Florence”, kondisi tree akan menjadi sebagai berikut.

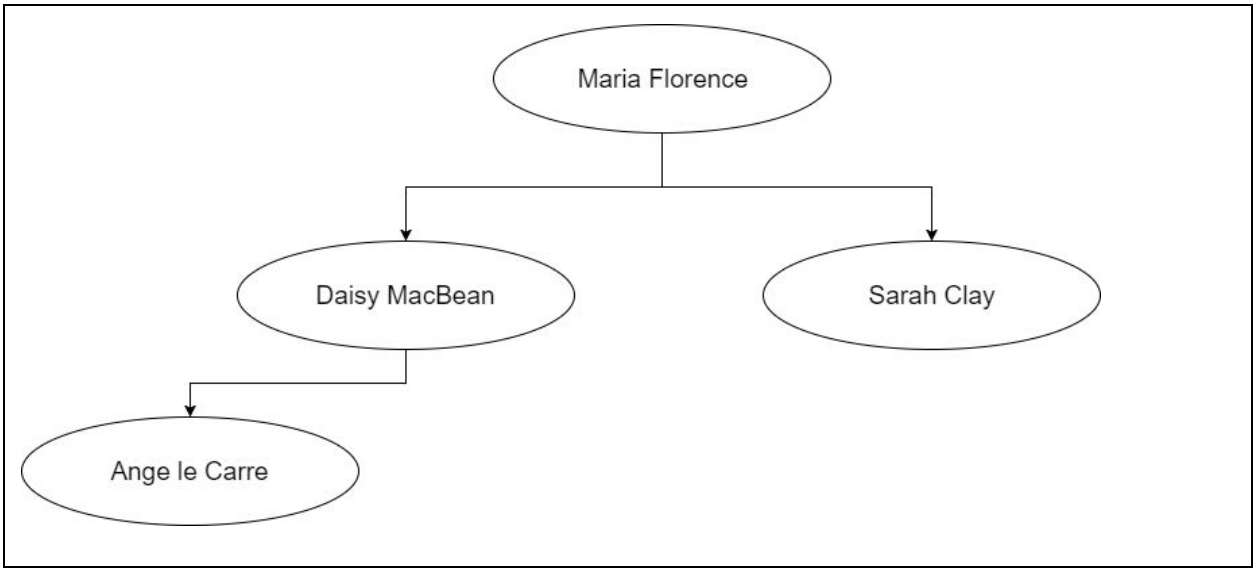
Setelah dimasukkan “Daisy MacBean”



Setelah dimasukkan “Sarah Clay”

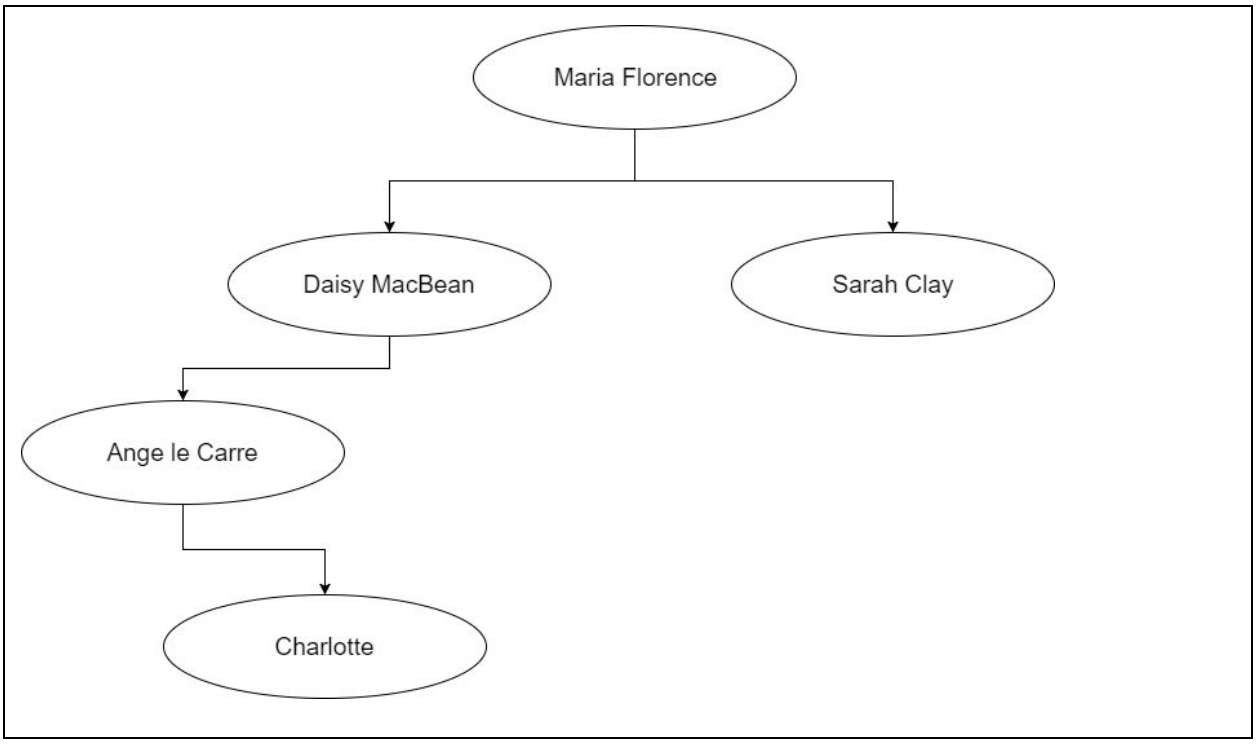


Setelah dimasukkan “Ange le Carre”



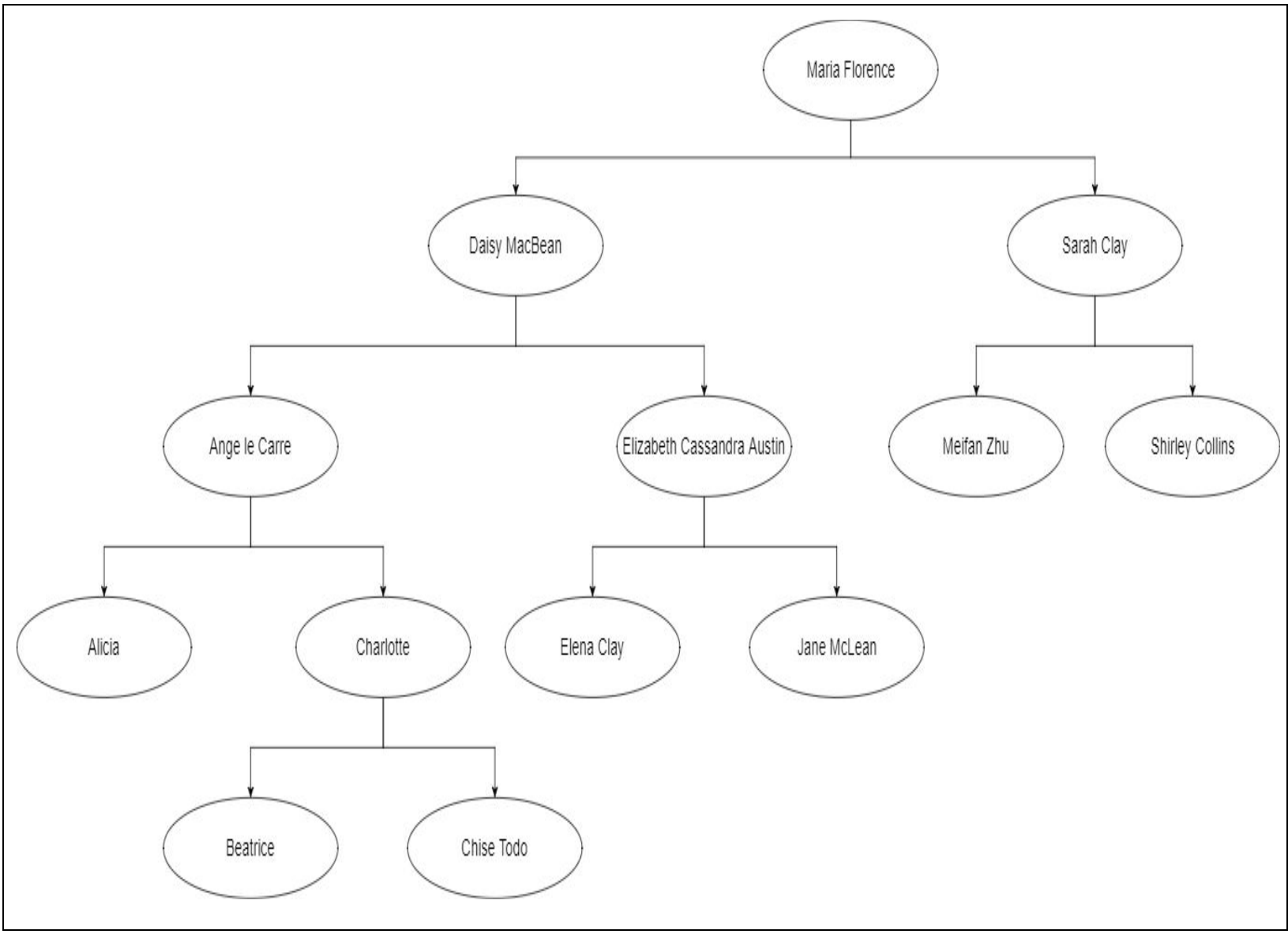
Setelah dimasukkan “Charlotte”

Problem setter : **JCA**



Dan seterusnya sehingga setelah penambahan semua string tersebut, kondisi tree menjadi

Problem setter : **JCA**



Perhatikan bahwa 5 string terakhir “Ange le Carre”, “Chise Todo”, “Beatrice”, “Daisy MacBean”, dan “Charlotte” tidak dimasukkan ke dalam tree karena string-string berikut sudah ada sebelumnya pada tree.

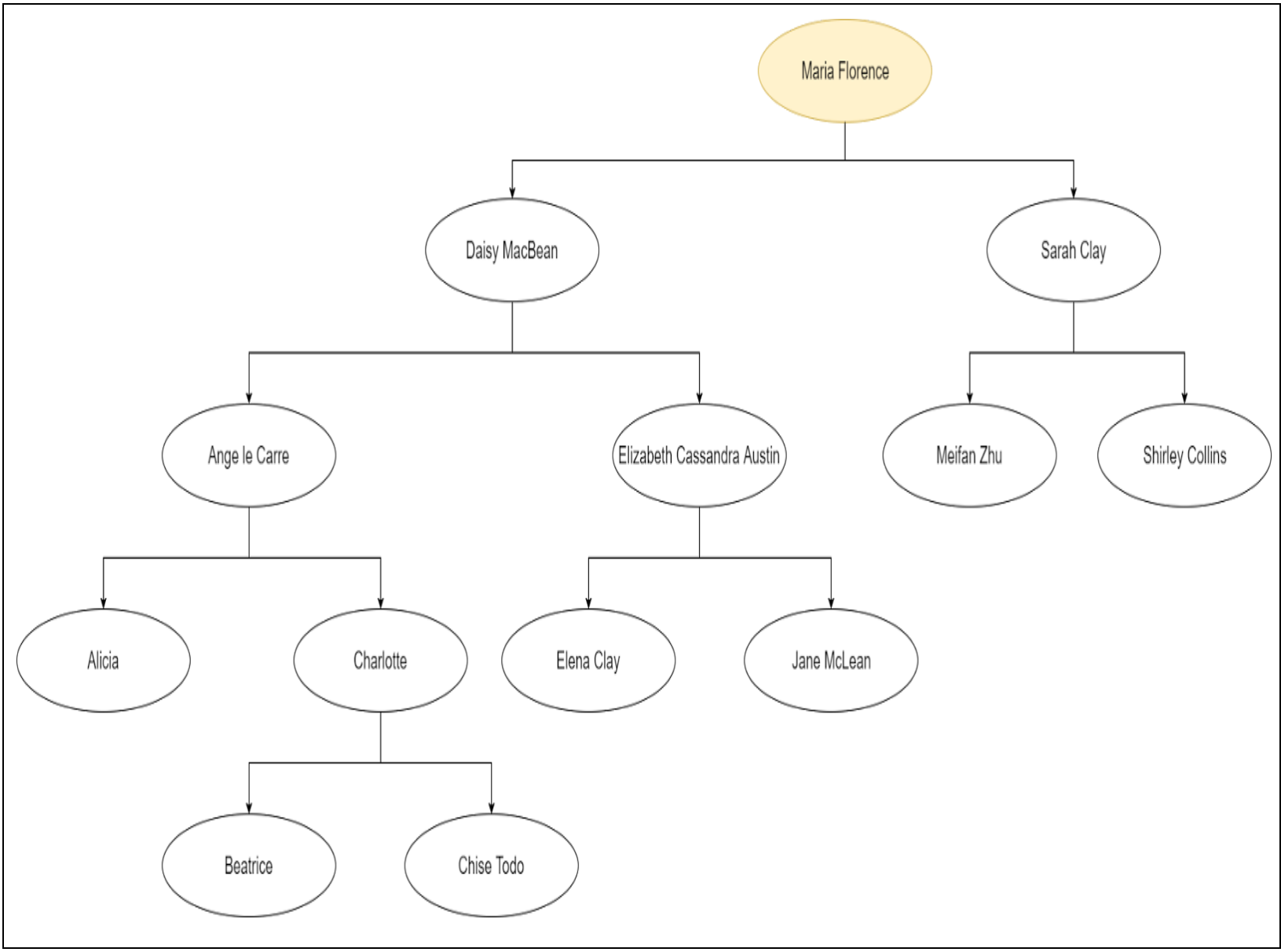
Operasi contains

Untuk mengetahui apakah suatu objek terdapat pada, dapat menggunakan operasi contains. Operasi contains mengembalikan nilai true jika elemen yang dicari ditemukan dan false jika sebaliknya.

Problem setter : **JCA**

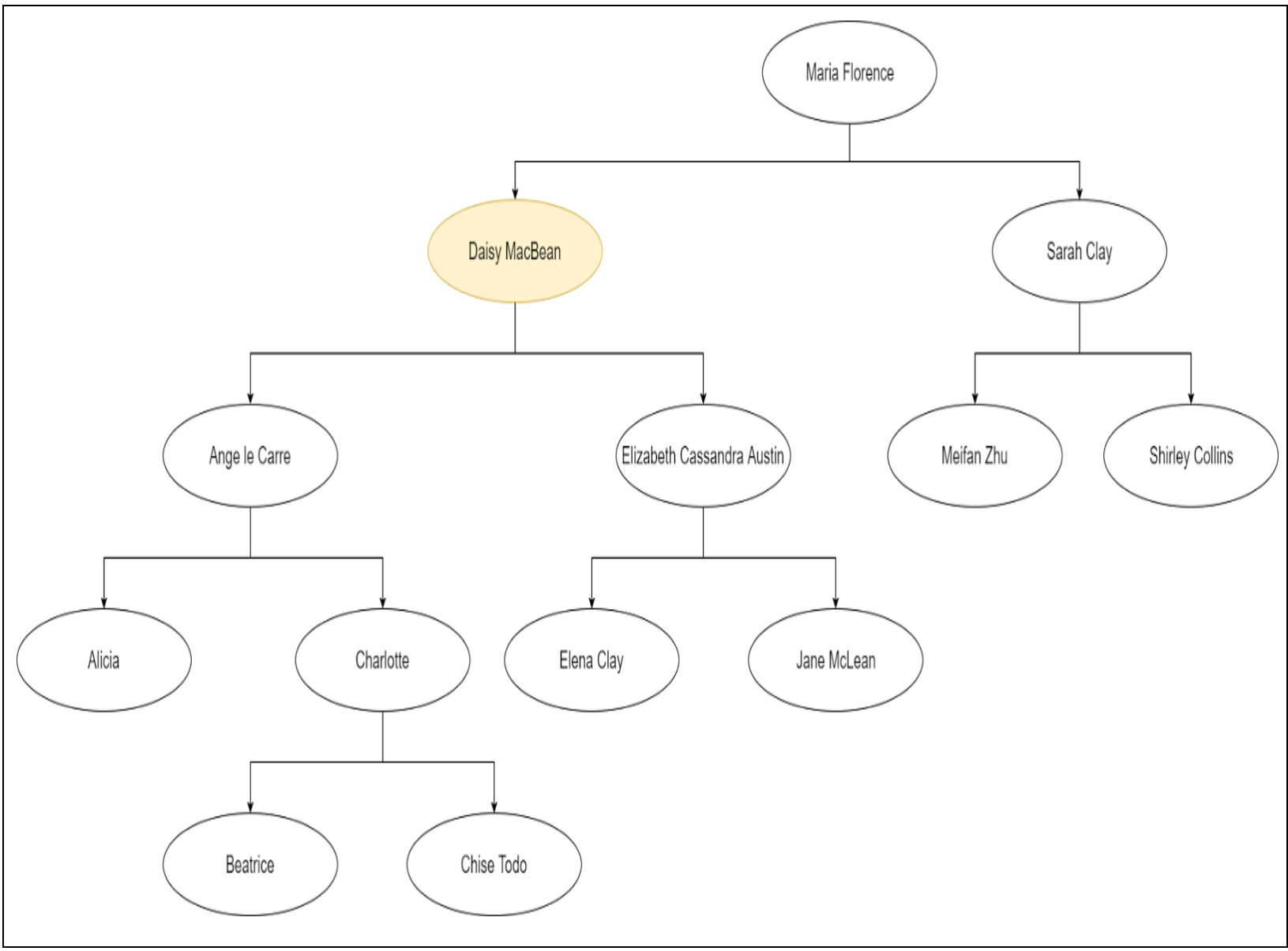
Berikut merupakan proses pencarian string “Charlotte”.

Dimulai dengan membandingkan string “Charlotte” dengan string “Maria Florence”



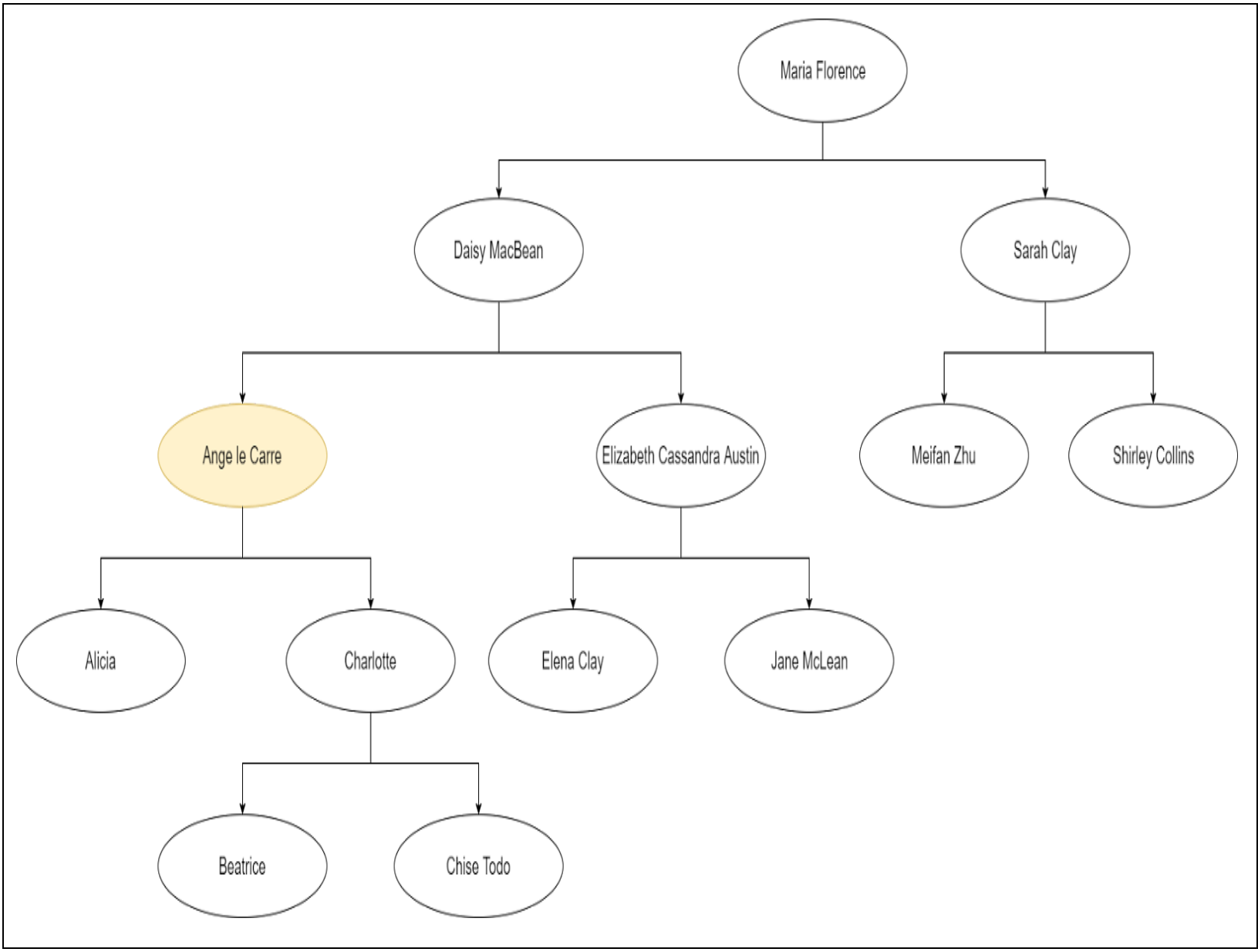
Problem setter : **JCA**

Karena “Charlotte” < “Maria Florence”, mengambil node kiri dari “Maria Florence”, yaitu “Daisy MacBean” untuk dibandingkan dengan “Charlotte”



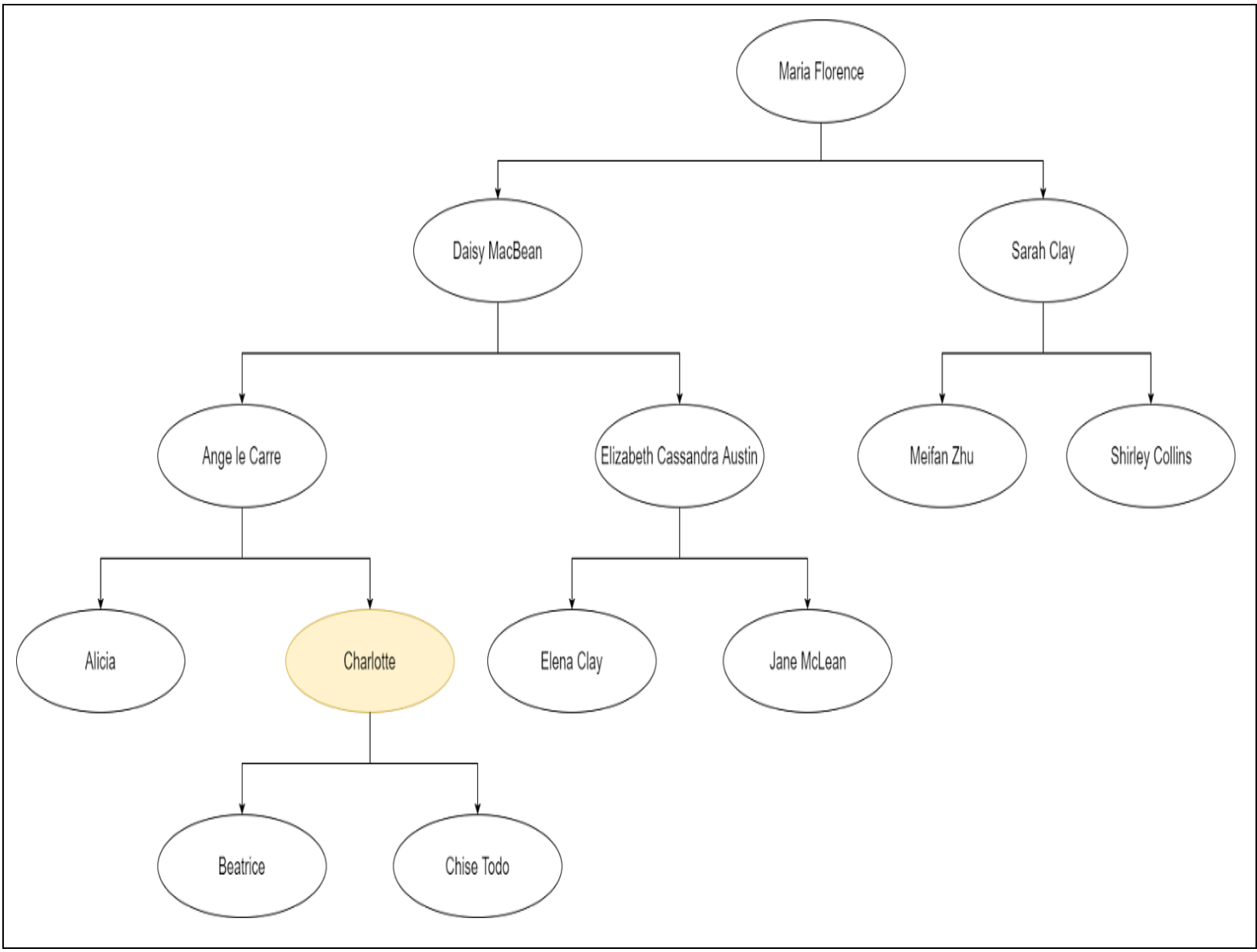
Problem setter : **JCA**

Karena “Charlotte” < “Daisy MacBean”, maka mengambil node kiri dari “Daisy MacBean” yaitu “Ange le Carre” untuk dibandingkan dengan “Charlotte”

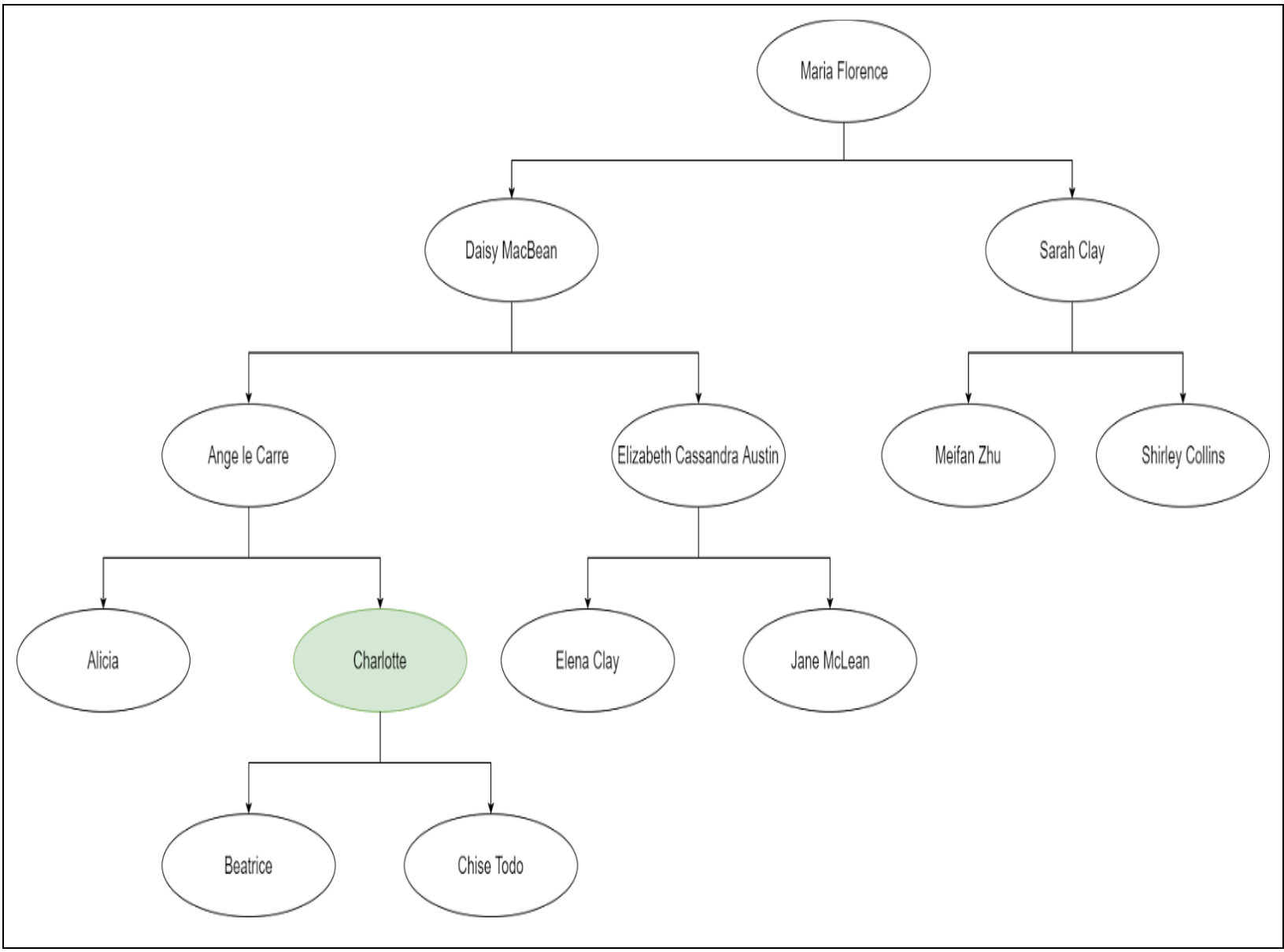


Karena “Charlotte” > “Ange le Carre”, maka mengambil node kanan dari “Ange le Carre” yaitu “Charlotte” untuk dibandingkan dengan “Charlotte”.

Problem setter : **JCA**



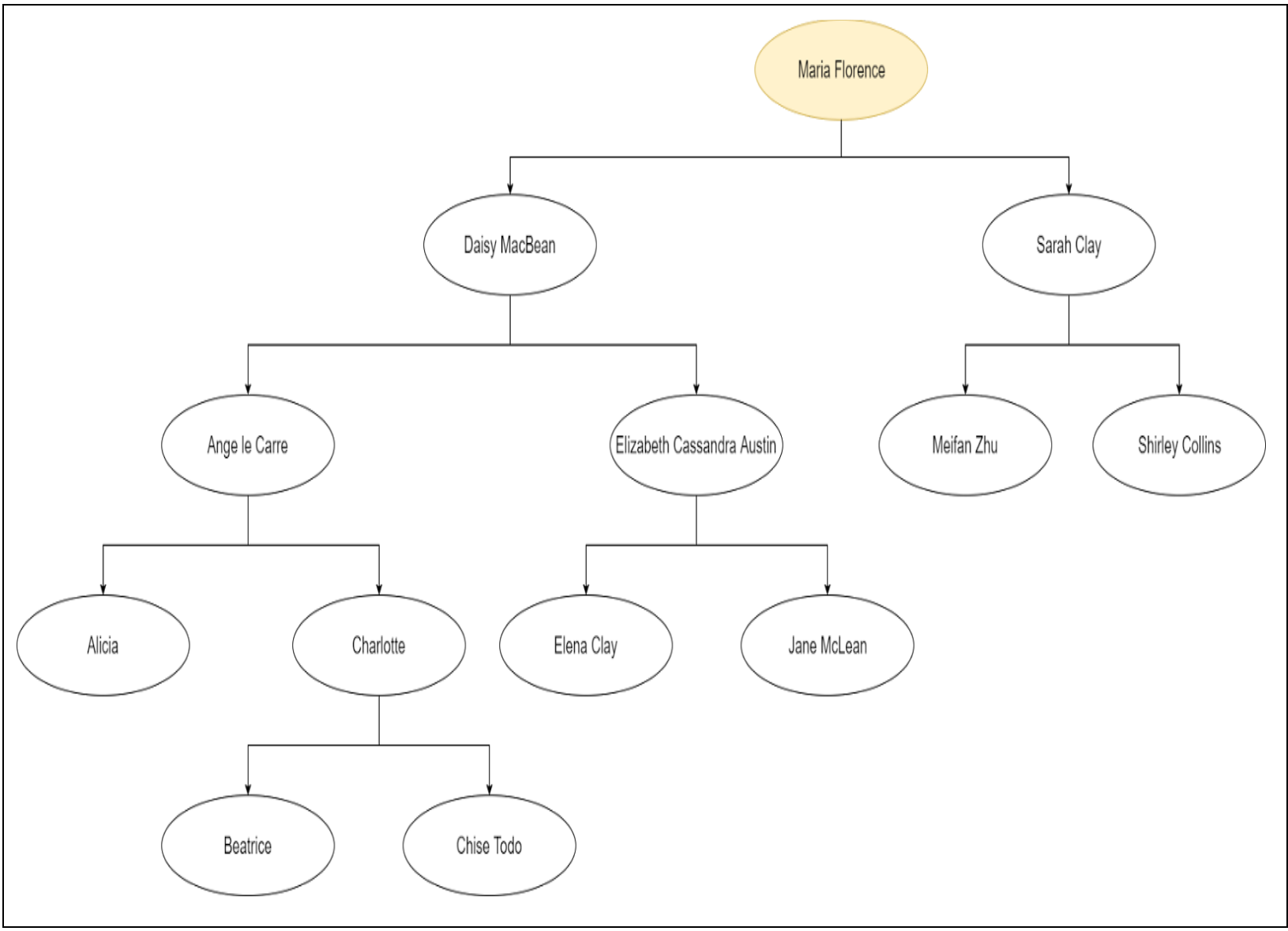
Problem setter : **JCA**



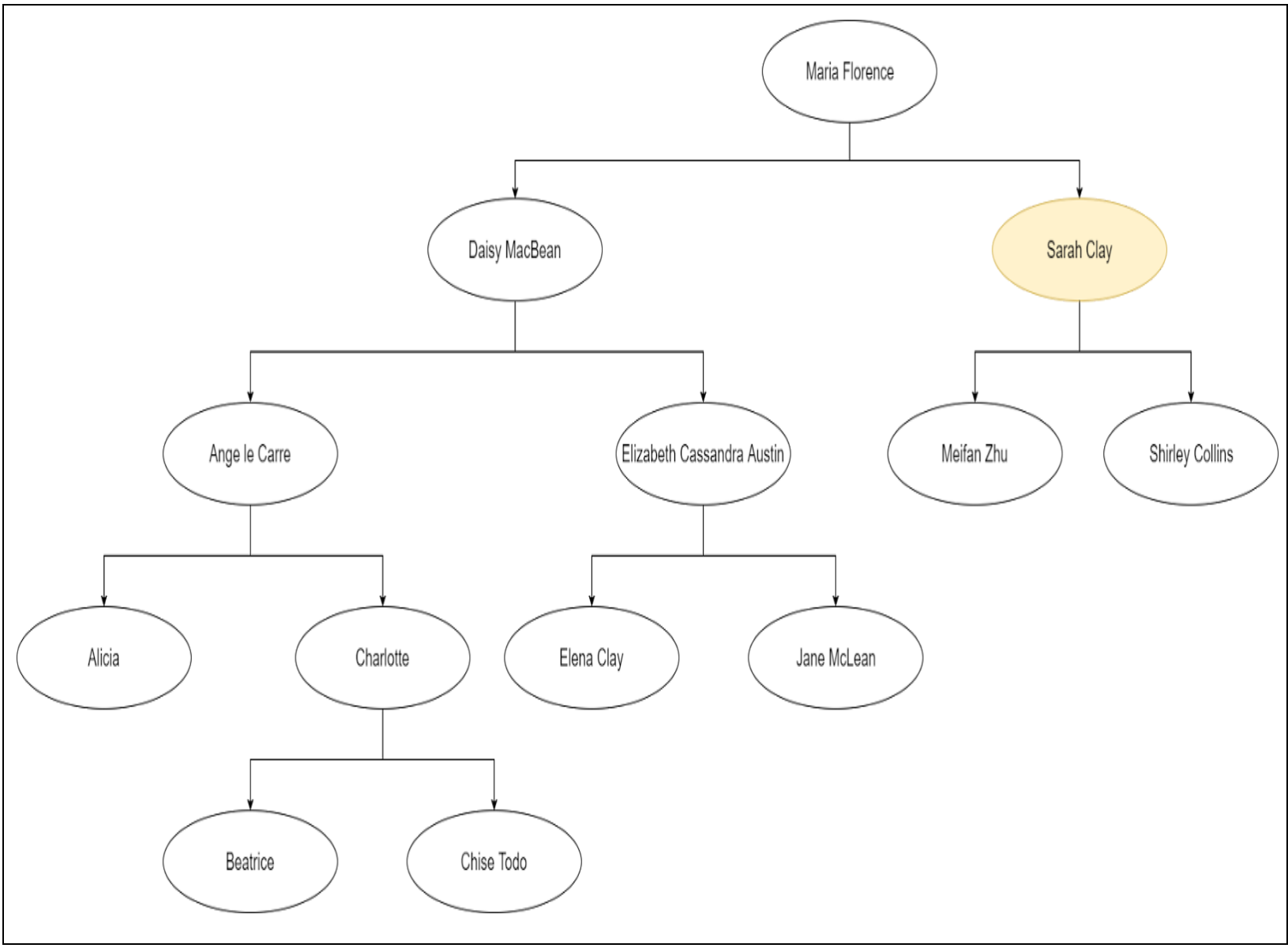
Perhatikan bahwa elemen “Charlotte” sudah ditemukan sehingga operasi contains mengembalikan nilai true.

Berikut proses pencarian string “Sophie Mackenzie” pada tree yang telah digunakan sebelumnya.

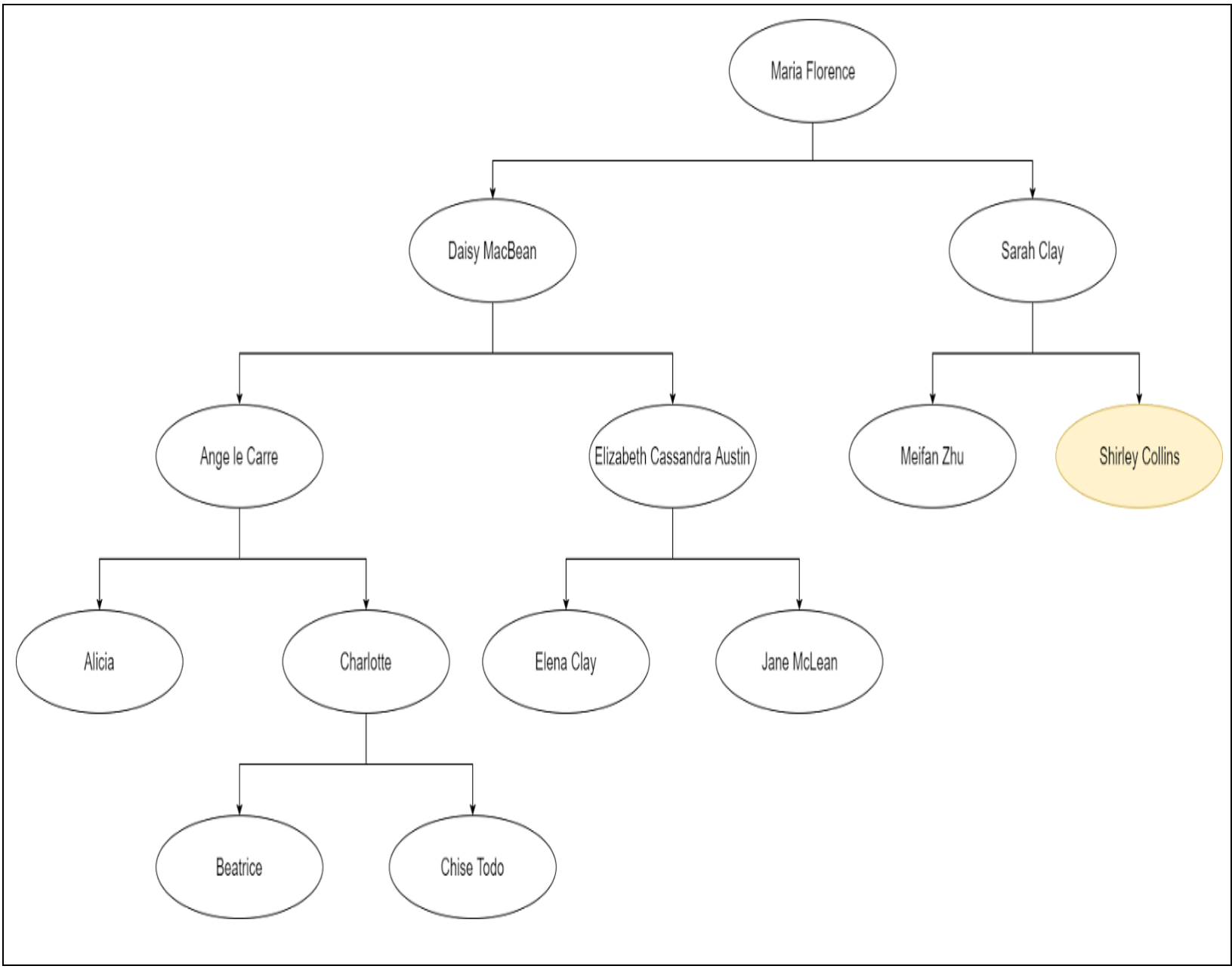
Problem setter : **JCA**



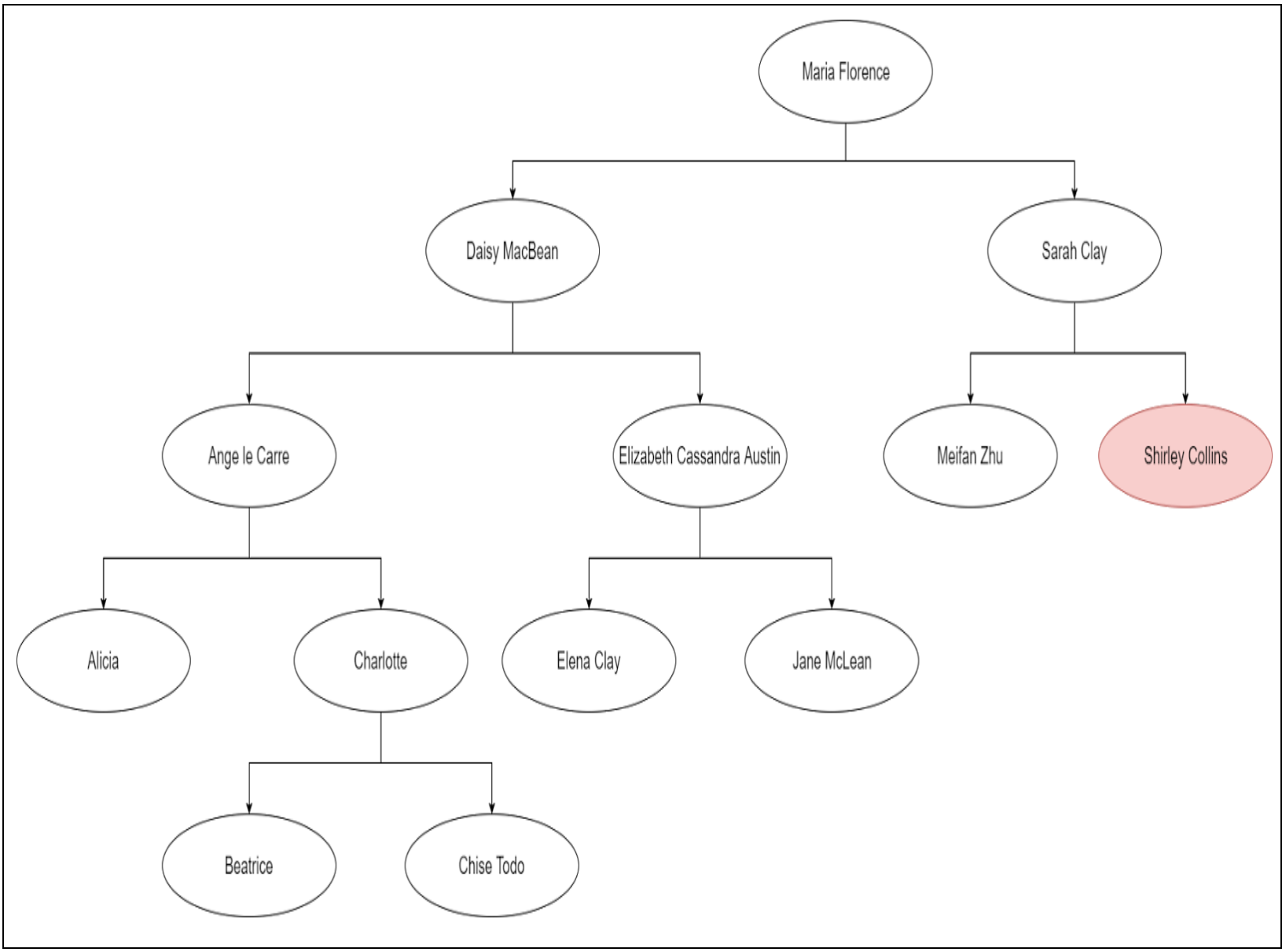
Problem setter : **JCA**



Problem setter : **JCA**



Problem setter : **JCA**



Perhatikan bahwa karena “Sophie Mackenzie” > “Shirley Collins”, maka node kanan dari “Shirley Collins” akan dibandingkan dengan “Sophie Mackenzie”. Namun, karena “Shirley Mackenzie” tidak memiliki node kanan, maka pencarian berhenti dan mengembalikan nilai false.

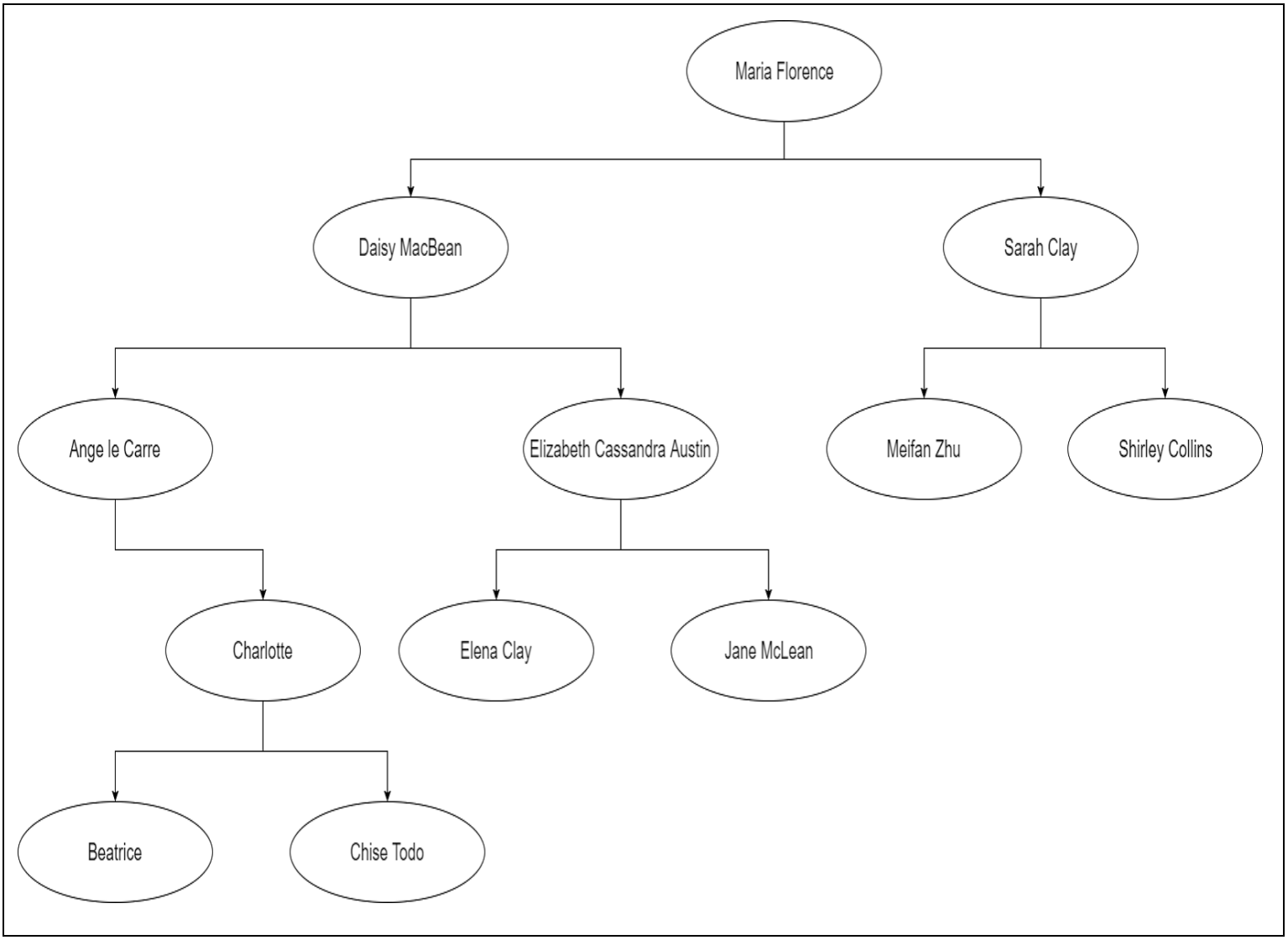
Operasi remove

Operasi remove merupakan operasi untuk menghapus elemen tertentu pada tree. Terdapat 3 kemungkinan kondisi saat melakukan penghapusan.

Problem setter : **JCA**

Pertama, jika elemen yang ingin dihapus merupakan leaf node, maka node tersebut akan langsung dihapus.

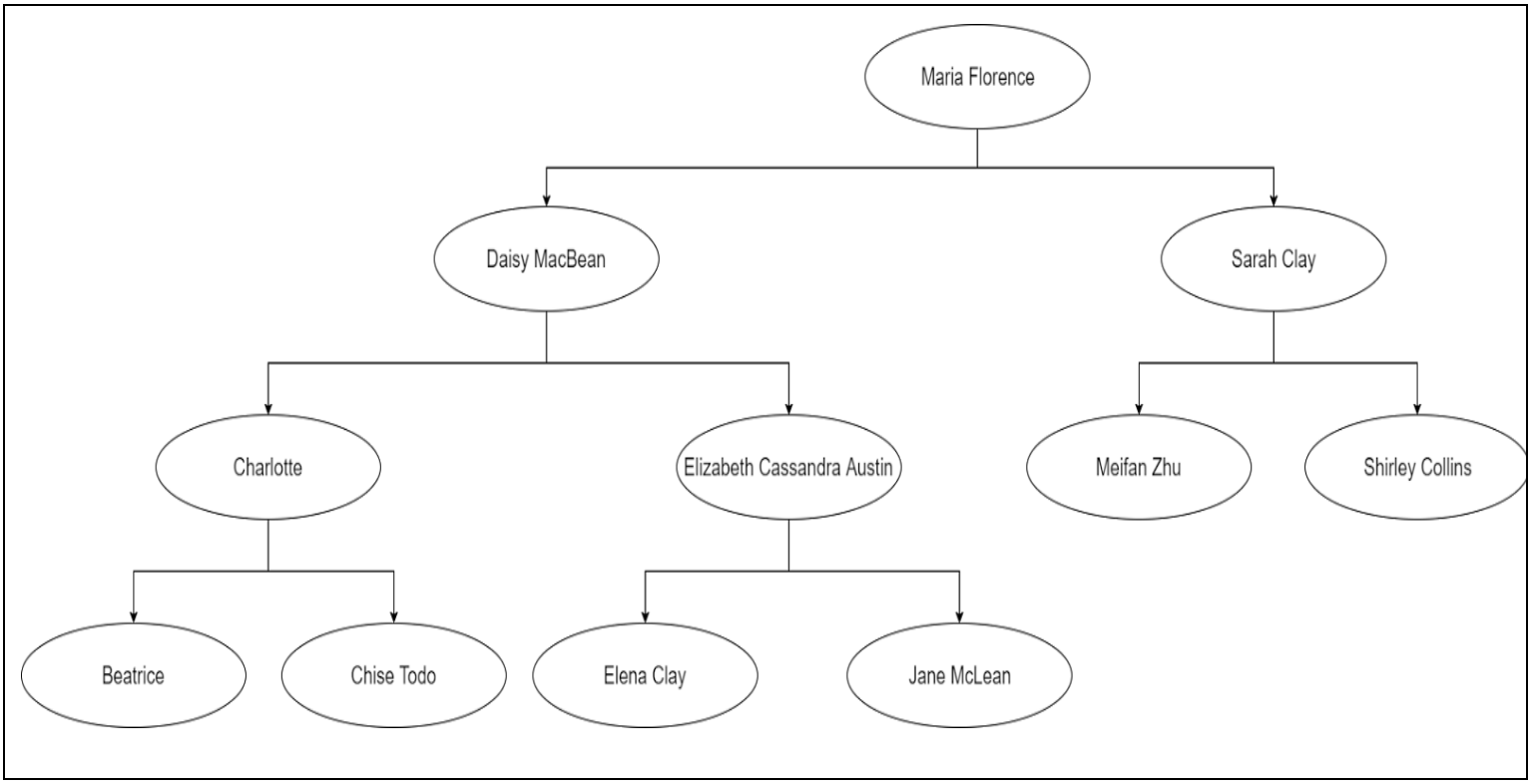
Berikut contoh penghapusan dengan kondisi pertama, yaitu dengan menghapus string “Alicia” dengan menggunakan tree pada kondisi sebelumnya.



Kedua, jika elemen yang ingin dihapus memiliki satu child note (left atau right), maka node yang ingin dihapus tersebut akan di-bypass (child node dari node yang ingin dihapus akan menjadi child node dari parent node dari node yang ingin dihapus tersebut).

Problem setter : **JCA**

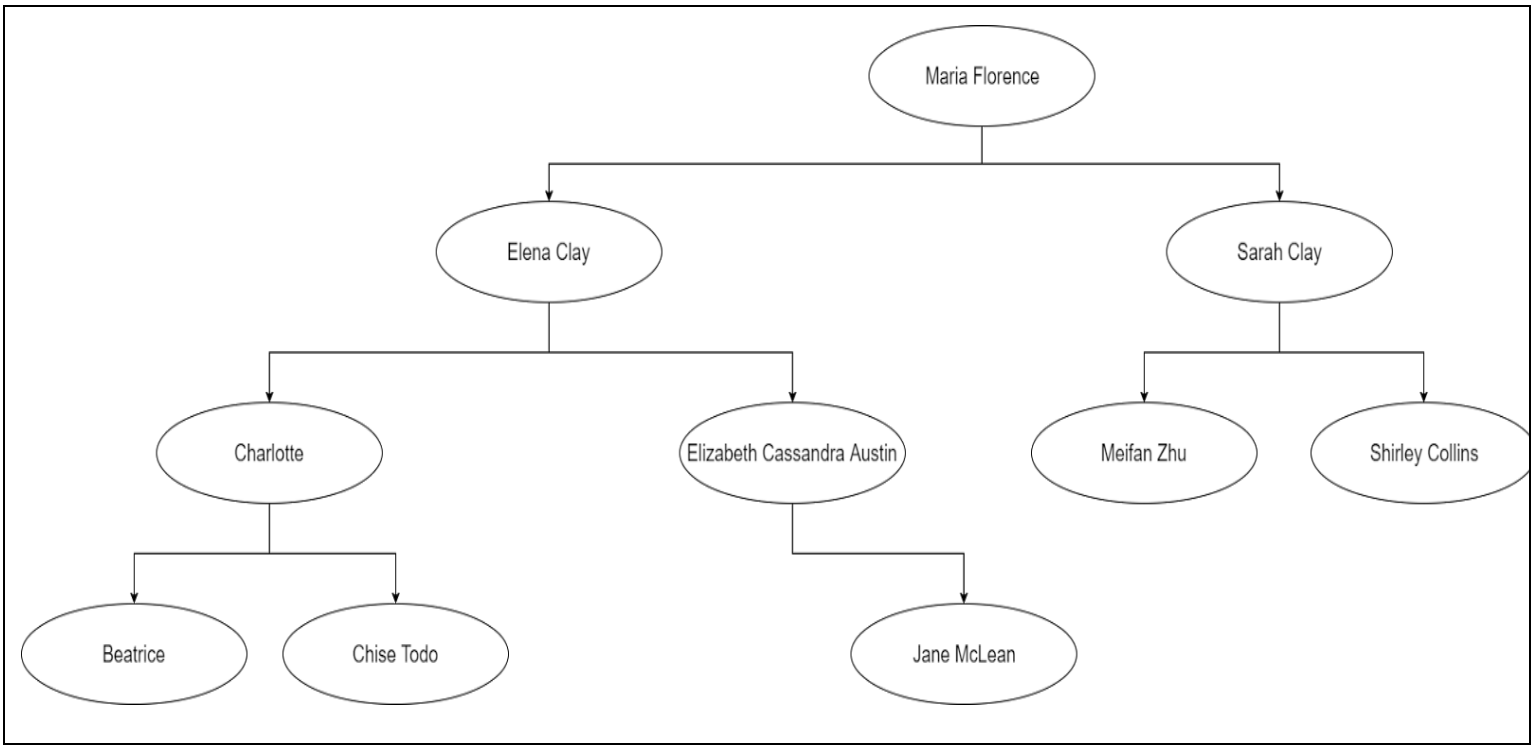
Berikut contoh dari penghapusan elemen dengan kondisi kedua dengan menghapus string “Ange le Carre” dengan menggunakan tree setelah penghapusan pertama



Ketiga, jika elemen yang dihapus memiliki dua child, terdapat dua cara, yaitu mencari predecessor inorder (elemen terbesar dari subtree kiri dari node yang ingin dihapus) atau successor inorder (elemen terkecil dari subtree kanan dari node yang ingin dihapus) lalu mengganti nilai node yang ingin dihapus dengan nilai pada node predecessor/successor inorder lalu node tersebut (node predecessor/successor inorder) dihapus dan child dari node tersebut (jika ada) menjadi child dari parent dari node tersebut (node predecessor/successor inorder di-bypass). Pada soal ini, akan digunakan pencarian successor inorder untuk penghapusan.

Berikut contoh penghapussan dengan kondisi ketiga, yaitu dengan menghapus string “Daisy MacBean”

Problem setter : **JCA**



Perhatikan bahwa “Elena Clay” merupakan node successor inorder dari node “Daisy MacBean” sehingga nilai pada node “Daisy MacBean” diganti dengan “Elena Clay” dan node yang sebelumnya bernilai “Elena Clay” dihapus dan karena node tersebut (node successor inorder) tidak memiliki child node, maka node tersebut akan langsung dihapus.

Operasi max dan operasi min

Operasi max berfungsi untuk mencari elemen dengan nilai terbesar pada tree sedangkan operasi min berfungsi mencari elemen dengan nilai terkecil pada tree. Kedua operasi tersebut dapat dengan mudah diimplementasi dengan memanfaatkan struktur dari binary search tree (semua elemen pada subtree kiri memiliki nilai lebih kecil dari nilai pada elemen root dan semua elemen pada subtree kanan memiliki nilai lebih besar dari nilai pada elemen root). Pada tree setelah dilakukan penghapusan ketiga, operasi min mengembalikan nilai “Beatrice” dan operasi max mengembalikan nilai “Shirley Collins”.

Traversing order

Selain operasi diatas, terdapat operasi khusus untuk mendapatkan semua elemen pada binary search tree dengan urutan tertentu.

Problem setter : **JCA**

Pre-order traversal

Urutan pre-order traversal didapatkan secara rekursif dengan mendapatkan node root terlebih dahulu, lalu mendapatkan urutan pre-order traversal subtree kiri, lalu mendapatkan urutan pre-order traversal subtree kanan. Pada tree setelah penghapusan ketiga, urutan pre-order traversalnya adalah *"Maria Florence", "Elena Clay", "Charlotte", "Beatrice", "Chise Todo", "Elizabeth Cassandra Austin", "Jane McLean", "Sarah Clay", "Meifan Zhu", "Shirley Collins"*.

Post-order traversal

Urutan post-order traversal didapatkan secara rekursif dengan mendapatkan urutan post-order traversal subtree kiri terlebih dahulu, lalu mendapatkan urutan post-order traversal subtree kanan, lalu mendapatkan node root. Pada tree setelah penghapusan ketiga, urutan post-order traversalnya adalah : *"Beatrice", "Chise Todo", "Charlotte", "Jane McLean", "Elizabeth Cassandra Austin", "Elena Clay", "Meifan Zhu", "Shirley Collins", "Sarah Clay", "Maria Florence".*

In-order traversal ascending

Urutan in-order traversal ascending didapatkan secara rekursif dengan cara mendapatkan urutan in-order traversal ascending subtree kiri, mendapatkan node root, lalu mendapatkan urutan in-order traversal ascending subtree kanan. Pada tree setelah penghapusan ketiga, urutan in-order traversal ascending adalah : *"Beatrice", "Charlotte", "Chise Todo", "Elena Clay", "Elizabeth Cassandra Austin", "Jane McLean", "Maria Florence", "Meifan Zhu", "Sarah Clay", "Shirley Collins"*. Perhatikan bahwa urutan in-order traversal ascending selalu mendapatkan semua elemen pada tree yang sudah terurut ascending

In-order traversal descending

Urutan in-order traversal descending didapatkan secara rekursif dengan cara mendapatkan urutan in-order traversal descending subtree kanan, mendapatkan node root, lalu mendapatkan urutan in-order traversal descending subtree kiri. Pada tree setelah penghapusan ketiga, urutan in-order traversal ascending adalah : *"Shirley Collins", "Sarah Clay", "Meifan Zhu", "Maria Florence", "Jane McLean", "Elizabeth Cassandra Austin", "Elena Clay", "Chise Todo", "Charlotte", "Beatrice"*. Perhatikan bahwa urutan in-order traversal descending selalu mendapatkan semua elemen pada tree yang sudah terurut descending.

Problem setter : **JCA**

Format Masukan dan Keluaran

Program membaca masukan sampai end of file. Perintah pada setiap baris beserta outputnya adalah sebagai berikut :

- Input : ADD;<string>
  Menambahkan <string> ke dalam tree
  Output : “<string> berhasil ditambahkan ke dalam tree” (jika berhasil) (<string> akan ditambahkan ke dalam tree)
  “<string> sudah dimasukkan sebelumnya” (jika <string> sudah terdapat pada tree) (<string> tidak akan ditambahkan ke dalam tree)
- Input : REMOVE;<string>
  Menghapus <string> dari tree
  Output : “<string> berhasil dihapus dari tree” (jika berhasil)
  “<string> tidak ditemukan” (jika <string> tidak ditemukan)
- Input : CONTAINS;<string>
  Mengetahui apakah <string> terdapat pada tree
  Output : “<string> terdapat pada tree” (jika <string> ditemukan)
  “<string> tidak terdapat pada tree” (jika <string> tidak ditemukan)
- Input : PREORDER
  Mendapatkan urutan pre-order dari tree
  Output : “<string_1>;<string_2>;...;<string_n>” (jika terdapat elemen pada tree)
  “Tidak ada elemen pada tree” (jika tidak terdapat elemen pada tree)
- Input : POSTORDER
  Mendapatkan urutan post-order dari tree
  Output : “<string_1>;<string_2>;...;<string_n>”
  “Tidak ada elemen pada tree” (jika tidak terdapat elemen pada tree)
- Input : ASCENDING
  Mendapatkan urutan in-order ascending dari tree
  Output : “<string_1>;<string_2>;...;<string_n>”
  “Tidak ada elemen pada tree” (jika tidak terdapat elemen pada tree)
- Input : DESCENDING
  Mendapatkan urutan in-order ascending dari tree

Problem setter : **JCA**

Output : “<string_1>;<string_2>;...;<string_n>”

“Tidak ada elemen pada tree” (jika tidak terdapat elemen pada tree)

- Input : MAX

  Mendapatkan elemen dengan nilai paling besar pada tree

  Output : “<max> merupakan elemen dengan nilai tertinggi” (jika terdapat elemen pada tree) (<max> merupakan elemen dengan nilai terbesar pada tree)

  “Tidak ada elemen pada tree” (jika tidak terdapat elemen pada tree)

- Input : MIN

  Mendapatkan elemen dengan nilai paling kecil pada tree

  Output : “<min> merupakan elemen dengan nilai terendah” (jika terdapat elemen pada tree) (<min> merupakan elemen dengan nilai terkecil pada tree)

  “Tidak ada elemen pada tree” (jika tidak terdapat elemen pada tree)

Contoh Masukan

```
ADD;Maria Florence
ADD;Daisy MacBean
ADD;Sarah Clay
ADD;Ange le Carre
ADD;Charlotte
ADD;Elizabeth Cassandra Austin
ADD;Chise Todo
ADD;Shirley Collins
ADD;Jane McLean
ADD;Beatrice
ADD;Meifan Zhu
ADD;Elena Clay
ADD;Alicia
ADD;Ange le Carre
ADD;Chise Todo
ADD;Beatrice
ADD;Daisy MacBean
```

Problem setter : **JCA**

```
ADD;Charlotte
CONTAINS;Charlotte
CONTAINS;Sophie Mackenzie
REMOVE;Alicia
REMOVE;Ange le Carre
REMOVE;Daisy MacBean
MAX
MIN
PREORDER
POSTORDER
ASCENDING
DESCENDING
```

Contoh Keluaran

```
Maria Florence berhasil ditambahkan ke dalam tree
Daisy MacBean berhasil ditambahkan ke dalam tree
Sarah Clay berhasil ditambahkan ke dalam tree
Ange le Carre berhasil ditambahkan ke dalam tree
Charlotte berhasil ditambahkan ke dalam tree
Elizabeth Cassandra Austin berhasil ditambahkan ke dalam tree
Chise Todo berhasil ditambahkan ke dalam tree
Shirley Collins berhasil ditambahkan ke dalam tree
Jane McLean berhasil ditambahkan ke dalam tree
Beatrice berhasil ditambahkan ke dalam tree
Meifan Zhu berhasil ditambahkan ke dalam tree
Elena Clay berhasil ditambahkan ke dalam tree
Alicia berhasil ditambahkan ke dalam tree
Ange le Carre sudah dimasukkan sebelumnya
Chise Todo sudah dimasukkan sebelumnya
Beatrice sudah dimasukkan sebelumnya
Daisy MacBean sudah dimasukkan sebelumnya
```

Problem setter : **JCA**

```
Charlotte sudah dimasukkan sebelumnya
Charlotte terdapat pada tree
Sophie Mackenzie tidak terdapat pada tree
Alicia berhasil dihapus dari tree
Ange le Carre berhasil dihapus dari tree
Daisy MacBean berhasil dihapus dari tree
Shirley Collins merupakan elemen dengan nilai tertinggi
Beatrice merupakan elemen dengan nilai terendah
Maria Florence;Elena Clay;Charlotte;Beatrice;Chise Todo;Elizabeth Cassandra
Austin;Jane McLean;Sarah Clay;Meifan Zhu;Shirley Collins
Beatrice;Chise Todo;Charlotte;Jane McLean;Elizabeth Cassandra Austin;Elena
Clay;Meifan Zhu;Shirley Collins;Sarah Clay;Maria Florence
Beatrice;Charlotte;Chise Todo;Elena Clay;Elizabeth Cassandra Austin;Jane
McLean;Maria Florence;Meifan Zhu;Sarah Clay;Shirley Collins
Shirley Collins;Sarah Clay;Meifan Zhu;Maria Florence;Jane McLean;Elizabeth
Cassandra Austin;Elena Clay;Chise Todo;Charlotte;Beatrice
```

Note : Teks dengan warna selain hitam pada contoh keluaran merupakan keluaran satu baris untuk masing-masing warna

Catatan

- **Anda harus mengimplementasikan Binary Search Tree sendiri**
- **Program yang tidak mengimplementasi Binary Search Tree tidak akan dinilai**
- **Penggunaan Arrays.sort dan Collections.sort pada soal ini tidak diperbolehkan**
- **Anda tidak diperbolehkan menggunakan kelas TreeSet, TreeMap, dan PriorityQueue bawaan Java**
- **Sudah disediakan template untuk soal ini**

Problem setter : **JCA**