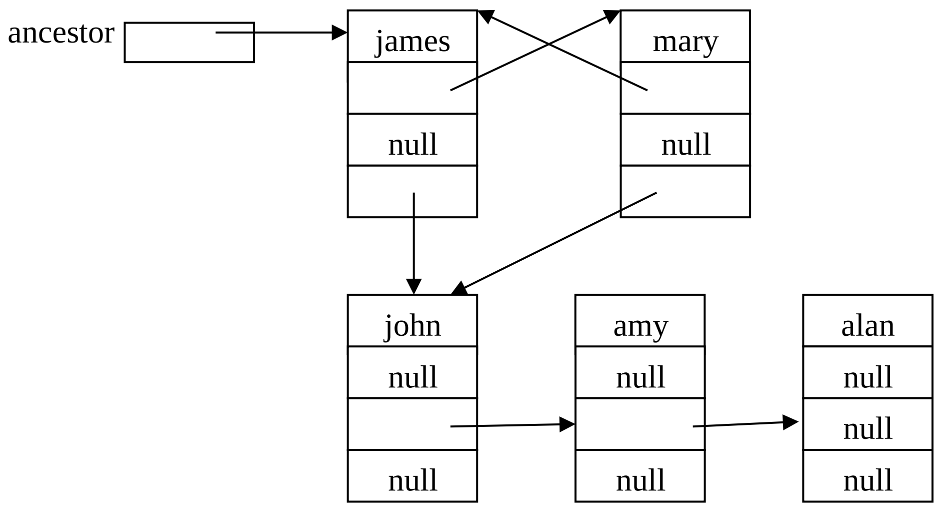
COURSEWORK

This coursework can be carried out either individually or in pairs. Submission should be via Aula. You are to implement a dynamic data structure to maintain a Family Tree. You are advised to develop the program one step at a time and keep a copy of the version for each step, which should then be included in your submission. In addition to the marks at each step, there are 20 marks for documentation, programming style, additional features, and an **individual** report. If you are working as a pair then you should both submit a separate report which includes details of the contribution that you each made to the submission. Your report should cover how you tested your system and discuss any difficulties you encountered, including any parts of your solution that are not working correctly (see marking scheme). There are **80** marks for this coursework, and it is worth **40%** of the marks for the module.

Note: at each step, you can assume that the user provides valid inputs but only in terms of the type of input required e.g., when asked to select an identifier, a whole number value will be input.

Step1: Functionality is worth 25 marks

A typical example structure:



Write a *FamilyTree* class that includes:

- a definition of the *FamilyTreeNode* class
    - the *FamilyTreeNode* class contains
        - a String representing the name of the family member
        - a *FamilyTreeNode* reference to the partner
        - a *FamilyTreeNode* reference to the next sibling
        - a *FamilyTreeNode* reference to the first child

e.g., a typical node:

object info	name
reference to partner	null
reference to sibling	null
reference to child	null

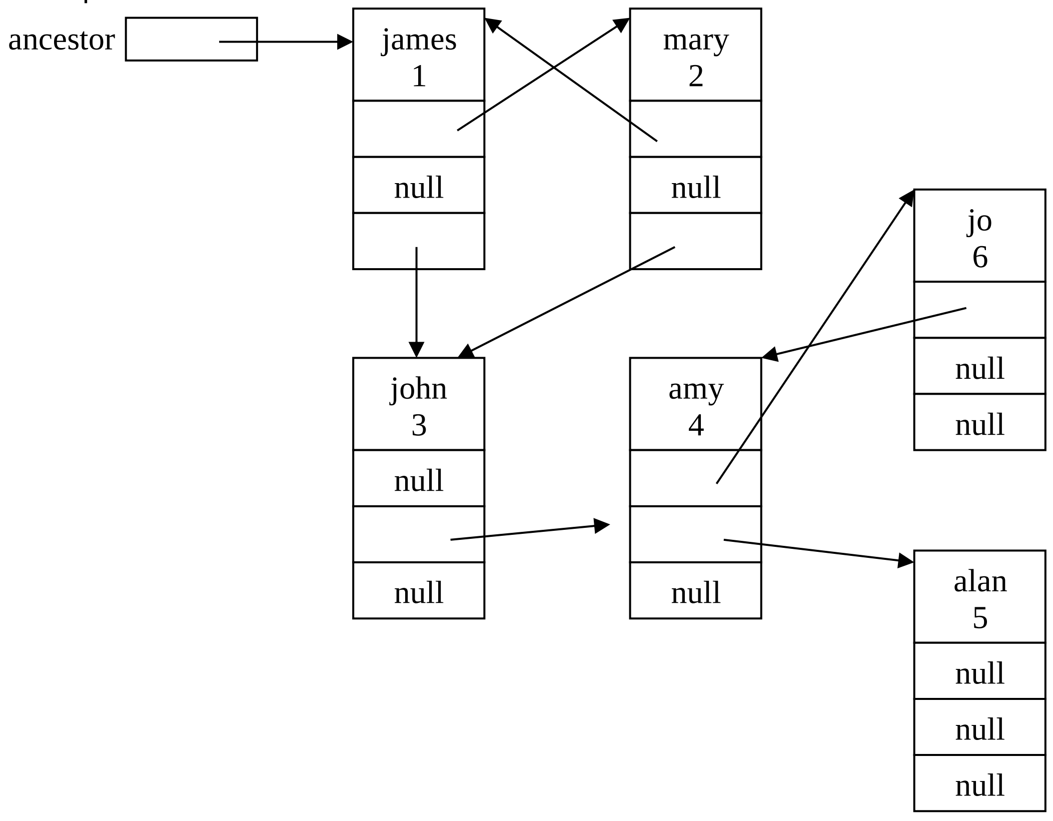
- a *constructor* that sets up the ancestor and their partner
- an *addChild* method
    - at this stage only the addition of children to the ancestor and their partner is required.
    - sibling names should be unique although not case sensitive
        - use exception handling and have *main* handle the exception
    - there is no sort order to how the child list is organised
- a *toString* method to allow the structure to be displayed e.g.

```
james partner mary
   john
   amy
   alan
mary partner james
   john
   amy
   alan
```

Write a *FamilyTreeTest* class that contains *main* and offers a simple menu to add a child or to display the family tree.

Step2: Functionality is worth 35 marks

A typical example structure:



Update the *FamilyTreeTest* class

- to include an option to add a partner
  - should display the complete structure and then get the identifier of the family member to whom a partner is to be added
- replace the display option with options to display the whole family or to display a specific family member

Ensure that when add a partner:

- only ask for the partner name if the identifier matches an existing family member and the family member does not already have a partner (a family member can have only 1 partner, which for the purposes of this exercise, cannot be changed)

Ensure that when add a child

- only ask for the child name if the ancestor has a partner

Update the FamilyTreeNode class

- add an identifier attribute that can be used to identify a particular family member
  - this should be setup automatically without the need for interaction with the user

Update the *FamilyTree* class

- the *constructor* should now only set up the ancestor node
- add appropriate methods to allow a partner to be added based on the requirements outlined above under *FamilyTreeTest*
- add appropriate methods to allow a child to be added based on the requirements outlined above under *FamilyTreeTest*
- update the *toString* method to display the complete structure
- add a *getFamilyMember* method which returns the details for a particular family member (via the identifier which is passed as a parameter)

Use exception handling and have *main* handle the exceptions for the following erroneous conditions

- no match found for an identifier (when adding a partner or displaying a family member)
- try to add a partner to a family member who already has a partner
- try to add a child to the ancestor who does not have a partner
- child name not unique

An example of displaying the whole family based on the example structure

```
james(identifier 1) partner mary(identifier 2)
```

```
        john(identifier 3) has no partner
        amy(identifier 4) partner jo(identifier 6)
          no children
        alan(identifier 5) has no partner
```

An example of displaying a family member based on selecting identifier 4

```
      amy(identifier 4) partner jo(identifier 6)
        no children
```

An example of displaying a family member based on selecting identifier 2

```
      mary(identifier 2) partner james(identifier 1)
        john(identifier 3) has no partner
        amy(identifier 4) partner jo(identifier 6)
          no children
        alan(identifier 5) has no partner
```

Submission

Hand in date is Sunday 24th April at 11:59pm

Upload your work via Aula

- create a folder based on your banner id e.g. *B00123456.* If the coursework is done in pairs, then include both Banner IDs.
- copy ALL the projects for the different versions of the program into this folder as well as your report(s).
- create a compressed zip version of this folder
  - in Windows Explorer, select the folder with the right button then select Send To then Compressed (zip) Folder
- select the Coursework Submission link in the Assignments tab on Aula for your campus and follow the instructions to upload the zip file

Marking Scheme

		Max	Actual			
Step 1						
	set up partner and ancestor	2				
	add each child	3				
	sibling name unique	3				
	not case sensitive	1				
	exception handling	2				
	display via the ancestor	1				

	display partner	1				
	display children	3				
	no children message if appropriate	1				
	display via the partner	1				
	display partner	1				
	display children	3				
	no children message if appropriate	1				
	quit message when choose quit option	1				
	invalid message if invalid menu option	1				
					/	**25**
Step 2						
	constructor only sets up ancestor	1				
	identifier is assigned automatically	1				
	before add, display no partner for ancestor	1				
	no “no children” message	1				
	detects attempt to add child when no partner	1				
	exception handling	1				
	suitable message displayed	1				
	only asks for child name if has partner	1				
	detects no match for identifier when adding partner	1				
	exception handling	1				
	suitable message displayed	1				
	only asks for partner name if identifier found	1				
	detects if add when already has a partner	1				
	exception handling	1				
	suitable message displayed	1				
	only asks for partner name if no partner already	1				
	displays structure when add partner	1				
	displays identifier for each family member	1				
	can add partner to ancestor	2				
	can add partner to child	2				
	identifier assigned is unique each time	1				
	displays child’s partner or	1				
	no partner message where appropriate	1				
	displays no children for child	1				
	only if has partner	1				
	display family member					
	family member correctly identified	1				
	detects no match for identifier	1				
	exception handling	1				
	suitable message displayed	1				
	including e.g. child’s partner	1				
	limit to branch of family member	1				

	display complete branch	2				
					/	**35**
Style & Report						
	code layout/indentation/structured programming	1				
	meaningful names	1				
	appropriate use of public/private/static	1				
	appropriate override of Object methods	1				
	use of *this* to identify object components	1				
	Additional marks can be gained for:	7				
	• Junit Tests					
	• Evidence of use of the GitHub repository					
	Report including testing	8				
					/	**20**
					/	**80**
					%	
			Grade			

NOTE - No marks will be awarded if the source code does not compile.