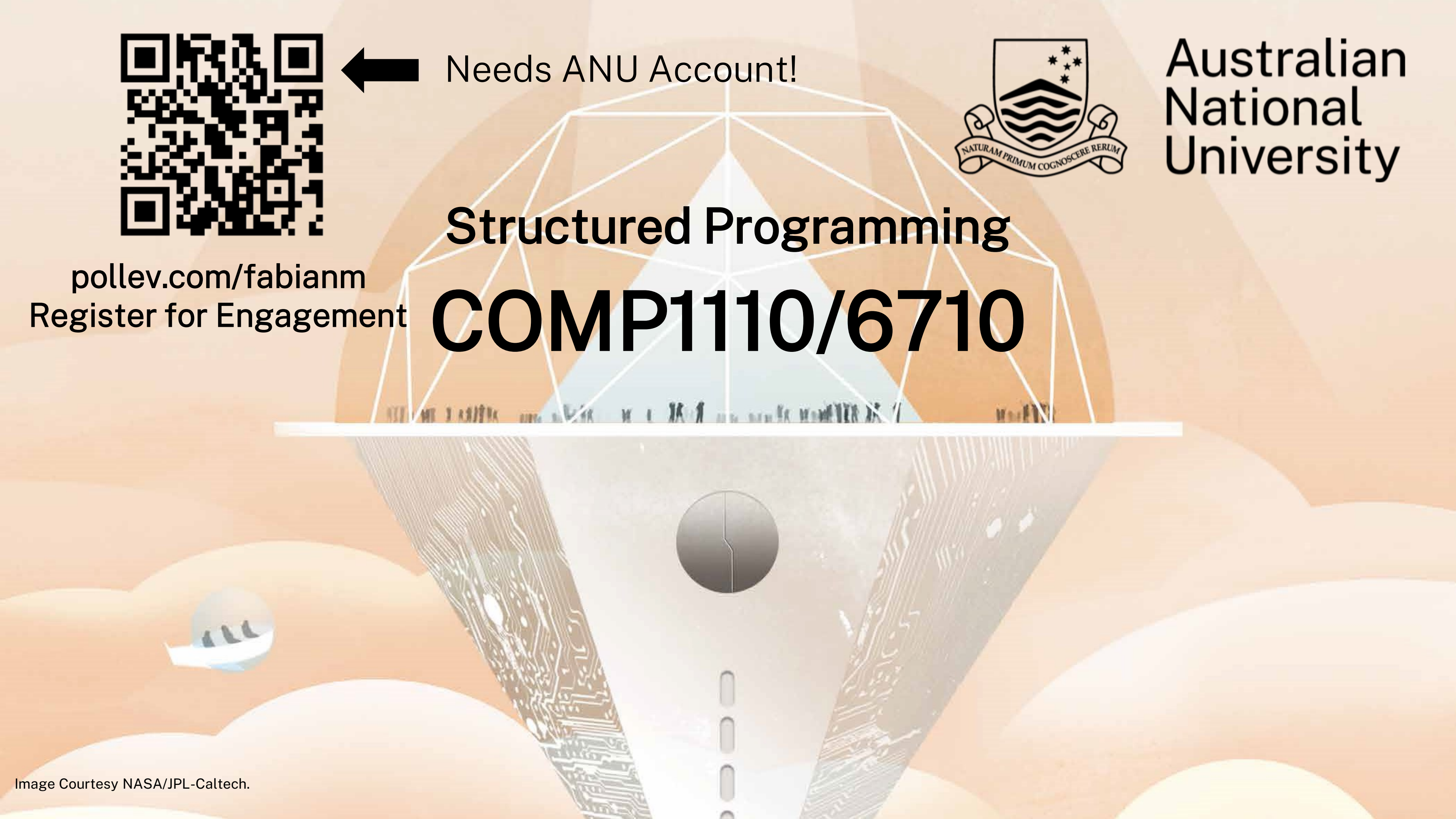


Needs ANU Account!

Australian National University

pollev.com/fabianm
Register for Engagement

Structured Programming

COMP1110/6710

Image Courtesy NASA/JPL-Caltech.

Recap: Assignments

Three steps you must do:

- Push Code on GitLab (can use assignment variables/extensions)
- Register for Code Walk (Base Deadline Day, 18:00, no extensions)
- Attend Code Walk at Scheduled Time

Two optional steps:

- Provide Particular Commit You Want to Submit (otherwise, latest commit before base deadline)
- Provide Scheduling Preferences for Code Walk (otherwise, some time during your registered tutorial time)

Recap: Academic Integrity in Assignments

You are expected to work on the assignments on your own.

- NOT with a friend
- NOT with ChatGPT & Co
- NOT with an outside tutor, cram school, or online tutoring service
  Instead of paying someone, ask us for help!

Code written by different people is not as similar as you might think!

You must sign a Statement of Originality for every assignment.

Code Walks

General Formula:

Tests (20 marks) – in P1, 16 automated, 4 manual

+ Code Walk (40 marks) – 8 Style, 17 Design, 15 Presentation

+ Test * Code Walk / 20 (40 marks)

- Form Deductions (up to 50 marks)

most common: 5 marks per disallowed Java feature

= ___/100 marks

* 1 if signed SOO, 0 otherwise

= ___/100 marks

See Skeleton Rubric on Course Website

Code Walks

Key reasons for low scores:

- Automated tests could not run
- Could not answer questions
- Did not give examples (in data type definitions and/or function definitions)
- Did not write down template where required
- Did not specify Design Strategy
- Did not follow any Design Strategy
- Highly Complex Code
- Code did not adhere to Functional Java Restrictions
- Too few tests (if any)
- No Statement of Originality/not properly signed

$\left.\begin{matrix} \\ \\ \\ \\ \\ \end{matrix}\right\}$ Not following the Design Recipe

Case Distinction vs. Template Application

Yes, switch-expressions/statements may form a case distinction.

However, if you use it to distinguish cases of an enumeration or itemization, it better be a template application.

Data Definitions

with thanks to Indiana Wilson

Explanation

Examples

Signature

Interpretations

Function Definitions

with thanks to Indiana Wilson

Purpose Statement

Examples

Strategy

Signature

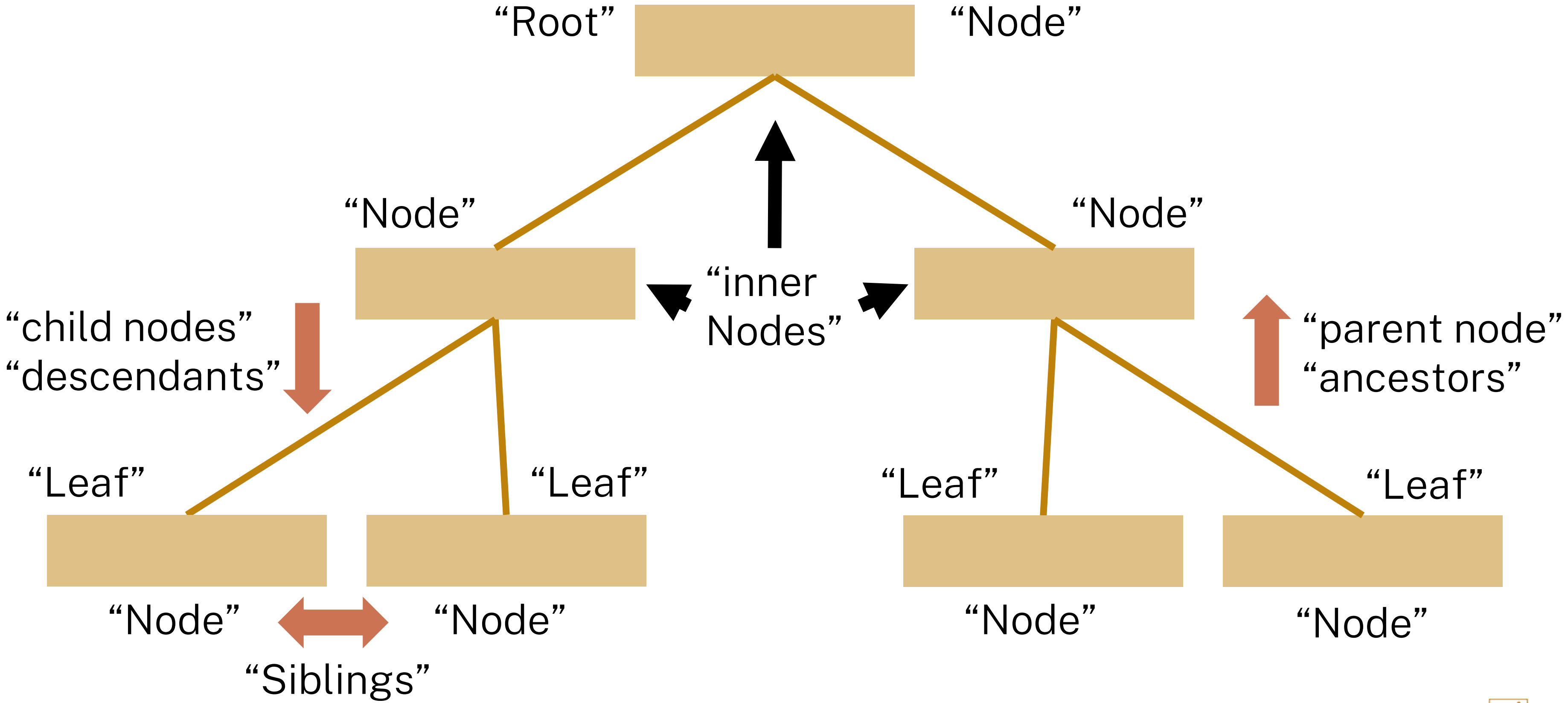
Tests

Trees

As Imagined by Computer Scientists

Australian National University

Trees


“Root”
“Node”
“Node”
“Node”
“inner Nodes”
“child nodes”
“descendants”
“parent node”
“ancestors”
“Leaf”
“Leaf”
“Leaf”
“Leaf”
“Node”
“Node”
“Node”
“Node”
“Siblings”

Binary Trees

```
/** ... */
sealed interface BinaryTree permits Leaf, Node {}
/** ... */
record Leaf() implements BinaryTree {}
/** ... */
record Node(BinaryTree left, BinaryTree right)
            implements BinaryTree {}
```

Recap: The ConsList<T> Template

```
// { ...
//   return ... switch(list) {
//    case Nil<T>() ->... ;
//    case Cons<T>(var element, var rest)->
//          ... element ... [recursiveCall](... rest ... ) ... ;
//   } ...;
// }
```

Key Motto:

The shape of the data determines the shape of the code!

A Tree Template (for binary trees)

```
// { ...
//   return ... switch(tree) {
//    case Leaf() ->... ;
//    case Node(var left, var right)->
//              ... [recursiveCall](... left  ... )  ...
//              ... [recursiveCall](... right ... )  ... ;
//   } ...;
// }
```

Key Motto:

The shape of the data determines the shape of the code!

Trees

- (Inner) Nodes (recursive cases) and Leaves (base cases)
- Topmost node is called “root”
- There may be multiple types of nodes
- Certain nodes may contain data
- Number of children may be fixed per node type or variable

Intertwined Data

Also called “mutual recursion”

Variable Arity Trees

```
/** ... */
sealed interface MixedTree permits BiNode, Node {}
/** ... */
record BiNode(MixedTree left, MixedTree right)
                implements MixedTree {}
/** ... */
record Node(ConsList<MixedTree> children)
              implements MixedTree {}
```

Intertwined data:
MixedTree & ConsList

Intertwined Data - Templates

```
// { …
//   return switch(tree) {
//     case BiNode(var left, var right) -> …
//       [recursiveCall](…left…)…[recursiveCall](…right…)…;
//     case Node(var children)->…[consListFun](…children…)…;
//   };
// }
```

Intertwined Data - Templates

```
// { …
//   return switch(list) {
//     case Nil() -> …;
//     case Cons(var first, var rest) -> …
//       [treeFun](…first…) … [recursiveCall](…rest…)…;
//   };
// }
```

Intertwined Data

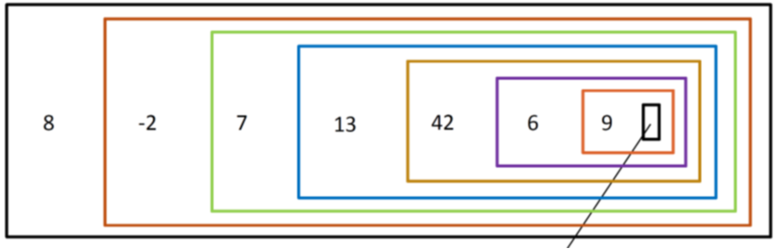
Different Data Definitions that refer to each other

- Need templates that refer to functions for each other
- Need to write functions in corresponding pairs

Recursive Functions

More Generally

Structural Recursion



Key idea:

Values contained within other values are smaller than the outer value.

Eventually, this must reach a base case.

Easy to see for general itemizations, but also true for integers going towards 0 or strings going towards the empty string.

Example: factorial

Maps

Storing and looking up values with keys

ConsList-based Maps

New in the standard library: functions for ConsList-based Maps:

`ConsList<Pair<K,V>>`

`K` are keys

`V` are values

For each K, there is at most one V in the map.

`Get(map, key)` returns a `Maybe<V>` - something if there is a `V` for `key`, otherwise nothing.

`Put(map, key, value)` returns a new map with key mapped to value

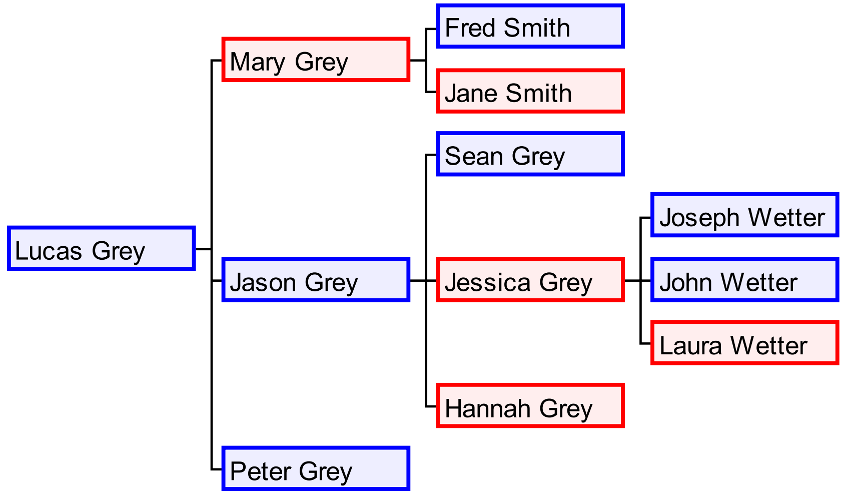
Practice

Fork and clone the comp1110-2025s1-workshops project.
Create a folder “ws4a”, and work in “DescendantsTree.java”.
Commit and push when you are done.

Following the design recipe, design a program able to model a particular person's descendants tree. A person's descendants include the children of that person (immediate descendants), the children of the children of that person (grandchildren) and so on. The program should keep track of the full name (i.e., names + surnames), gender, birth and death date (***if and only if the person died***) of every person in the descendant's tree (including the root of the tree).

You can assume: (1) all dates are in the same time zone, and time of day does not matter; (2) the birth date of the children of any parent in the tree has to be a biologically compatible future date compared to the birth date of the parent; (3) the difference among birth dates of any two siblings is biologically compatible; (4) the death date can never be an earlier date than the birth date; (5) there cannot be two descendants with the same full name.

Person's descendant tree example



Source: Wikipedia

Lucas Grey's descendant tree

Lucas has:

- 3 children (Mary, Jason, Peter)
- 5 grandchildren (Fred, Jane, Sean, Jessica, Hannah); and
- 3 great-grandchildren (Joseph, John, Laura)

Practice

Fork and clone the comp1110-2025s1-workshops project.
Create a folder "ws4a", and work in "DescendantsTree.java".
Commit and push when you are done.

Design the following functions:

1. Given the full name of a person (possible different from the person in the root of the tree), count how many descendants of that person are there in a person's descendant tree. Hint: develop first a function that counts how many descendants are there in a person's descendant tree.
2. Given a person's descendant tree, generate a list with the full names of (1) all descendants which are females; (2) descendants still alive; (3) descendants born after a given date; (4) descendants that have had 2 or more children born the same date. *Note 1*: you can develop a single higher-order function to code (1)-(4). *Note 2:* the order of elements in the output list is not important; the output of the function is correct as far all the requested full names are in the output list.
3. Given a person's descendant tree, return the name of the person that had their first children earliest in their life among all parents in the tree. If more than one person fulfills this criterion, return one of such people arbitrarily.
4. Given a person's descendant tree, returns a new tree where all males and their descendants are removed from the input tree. Generalize the function such that this operation (removal of some people and all their descendants) can be applied to any predicate passed as an argument to the function.