Generics

Simon Robinson
Software Developer

@TechieSimon www.SimonRobinson.com

Overview

Re-using logic for different types

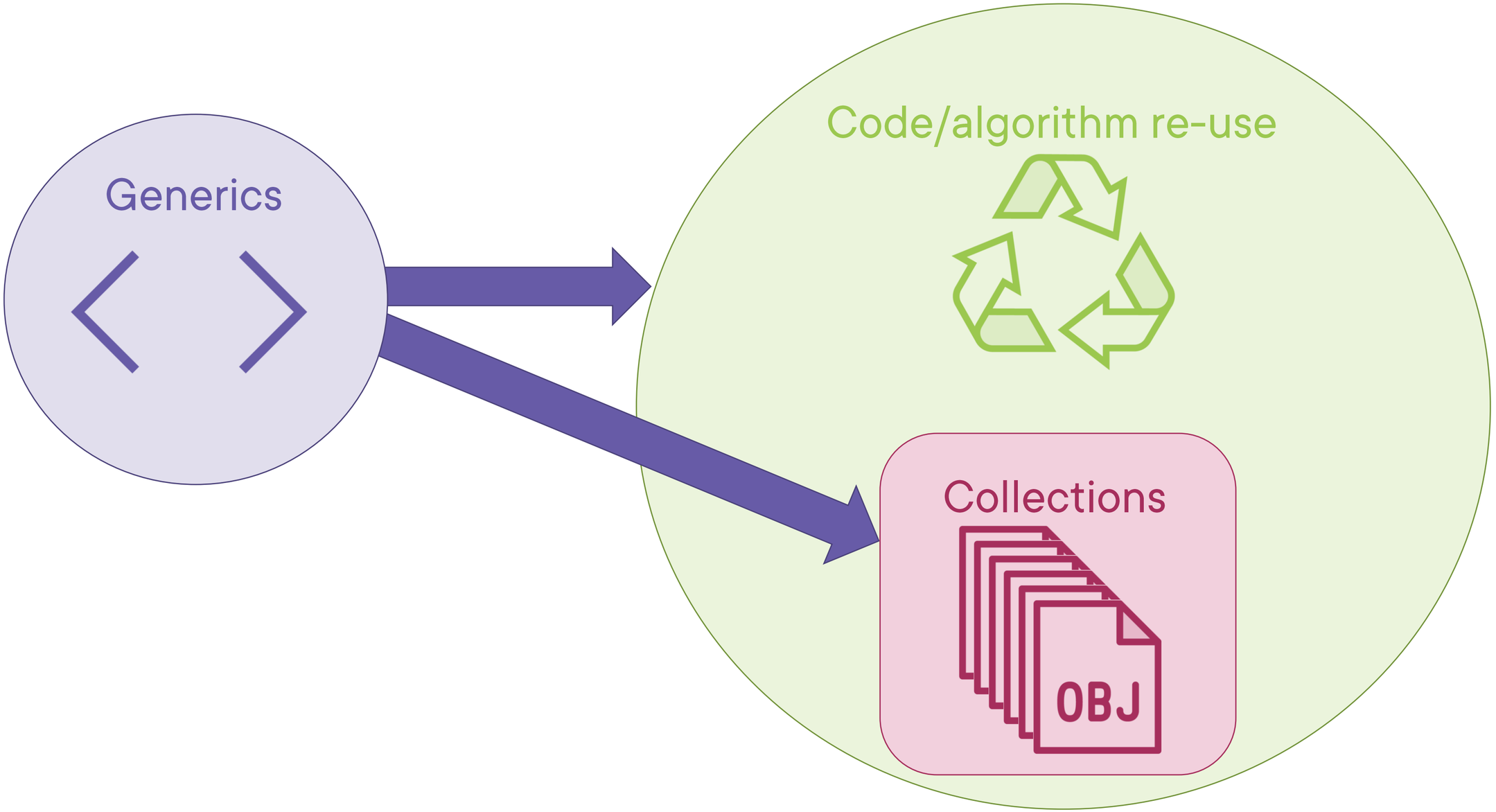
- This is what generics are for

Comparing generic values

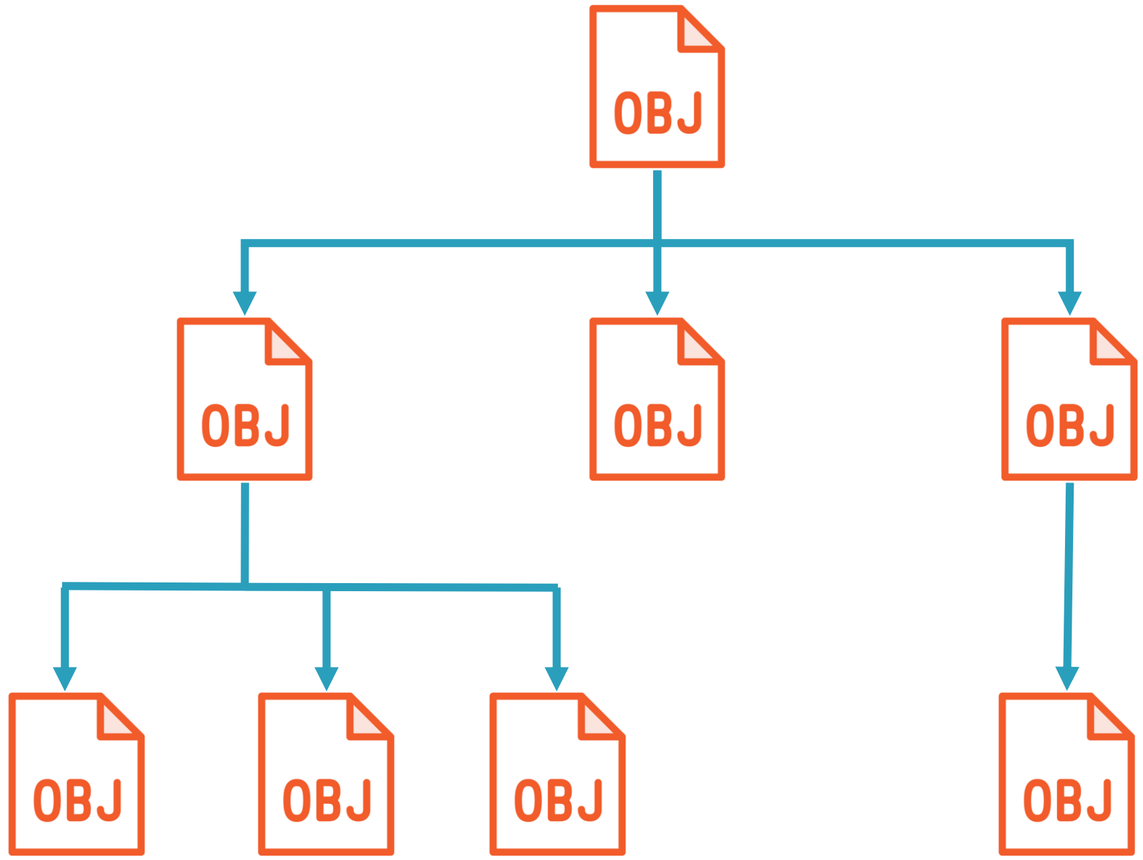
Generic base classes

- Can improve type safety
- Useful for business objects

What Are Generics For?

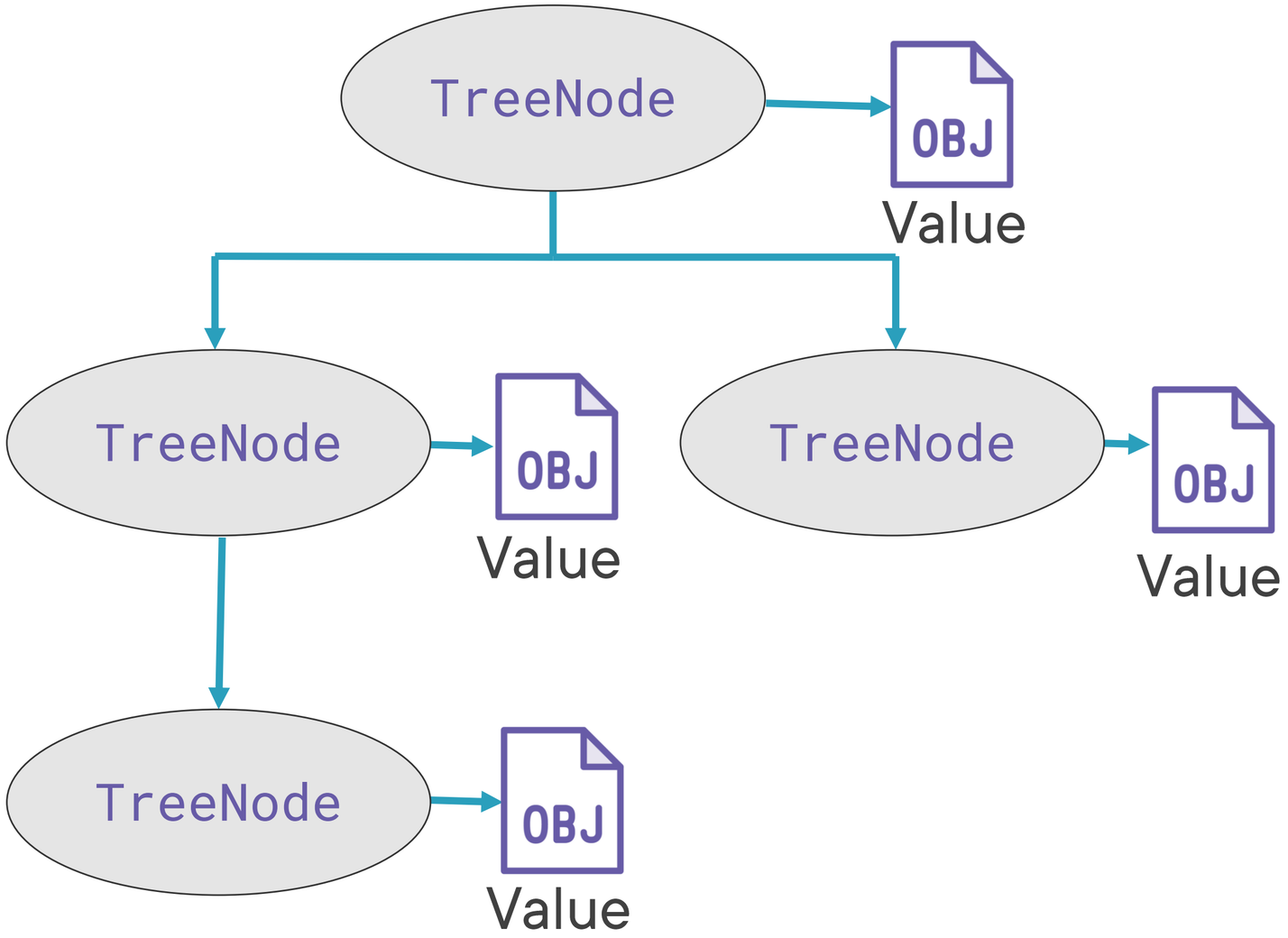


Hierarchy of Data



Examples:
Manager-employee
Parts of a product
Company ownership

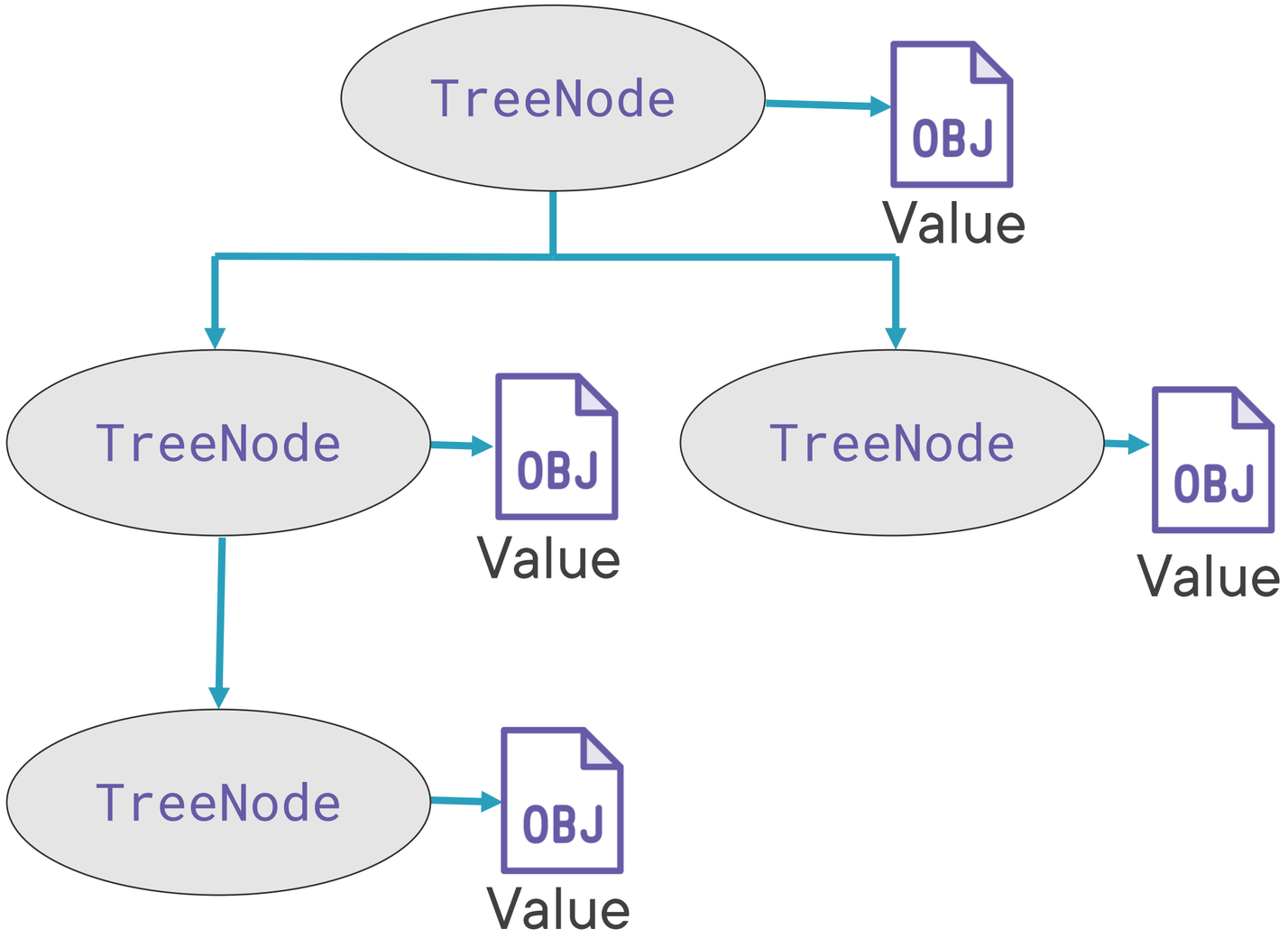
Hierarchy of Data



`TreeNode` captures the logic of placing an object in the hierarchy

No other types required

Hierarchy of Data

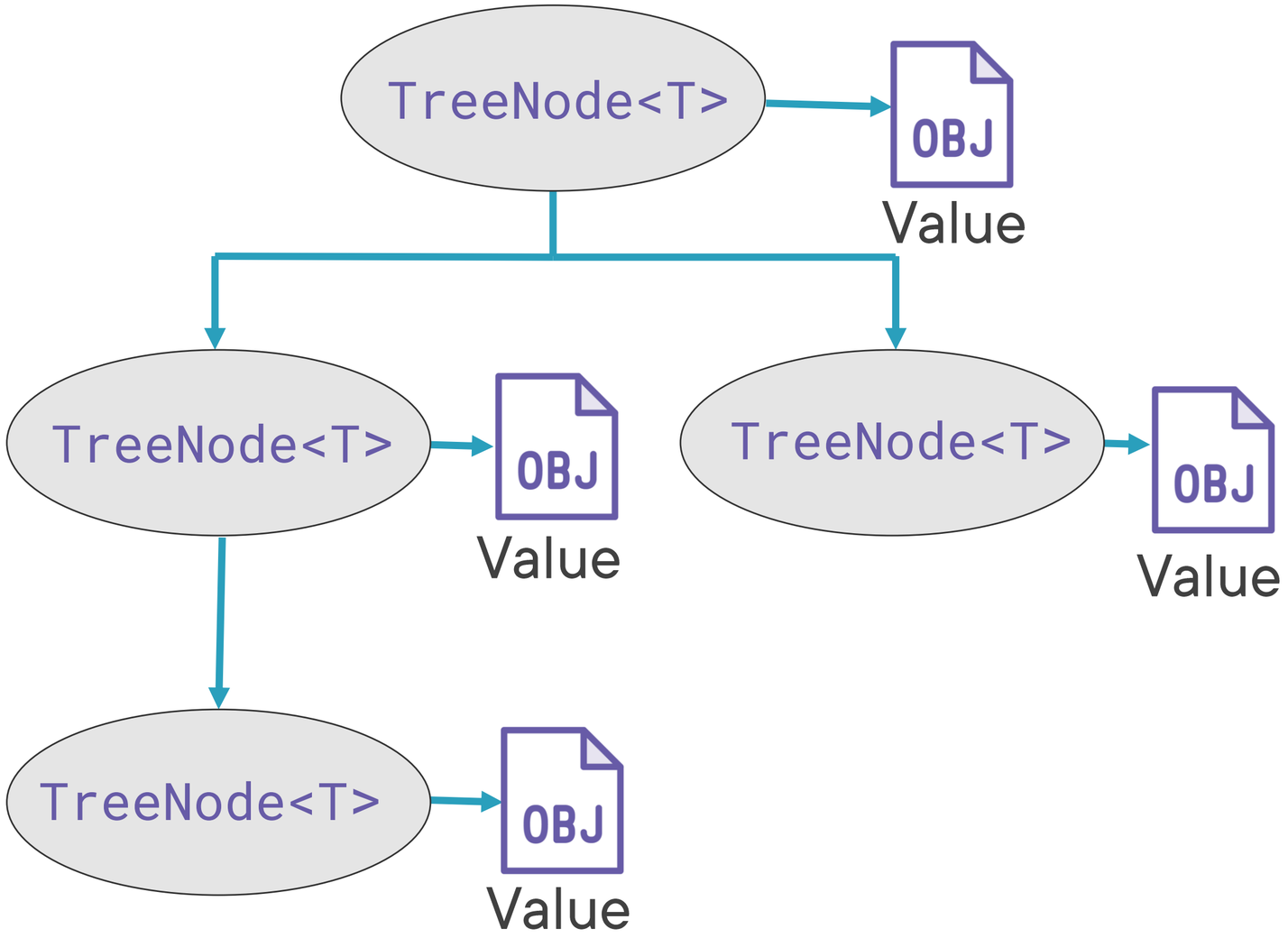


Need a hierarchy of any type!

`TreeNode` must supply the logic, but not specify the type

So need to use generics

Hierarchy of Data



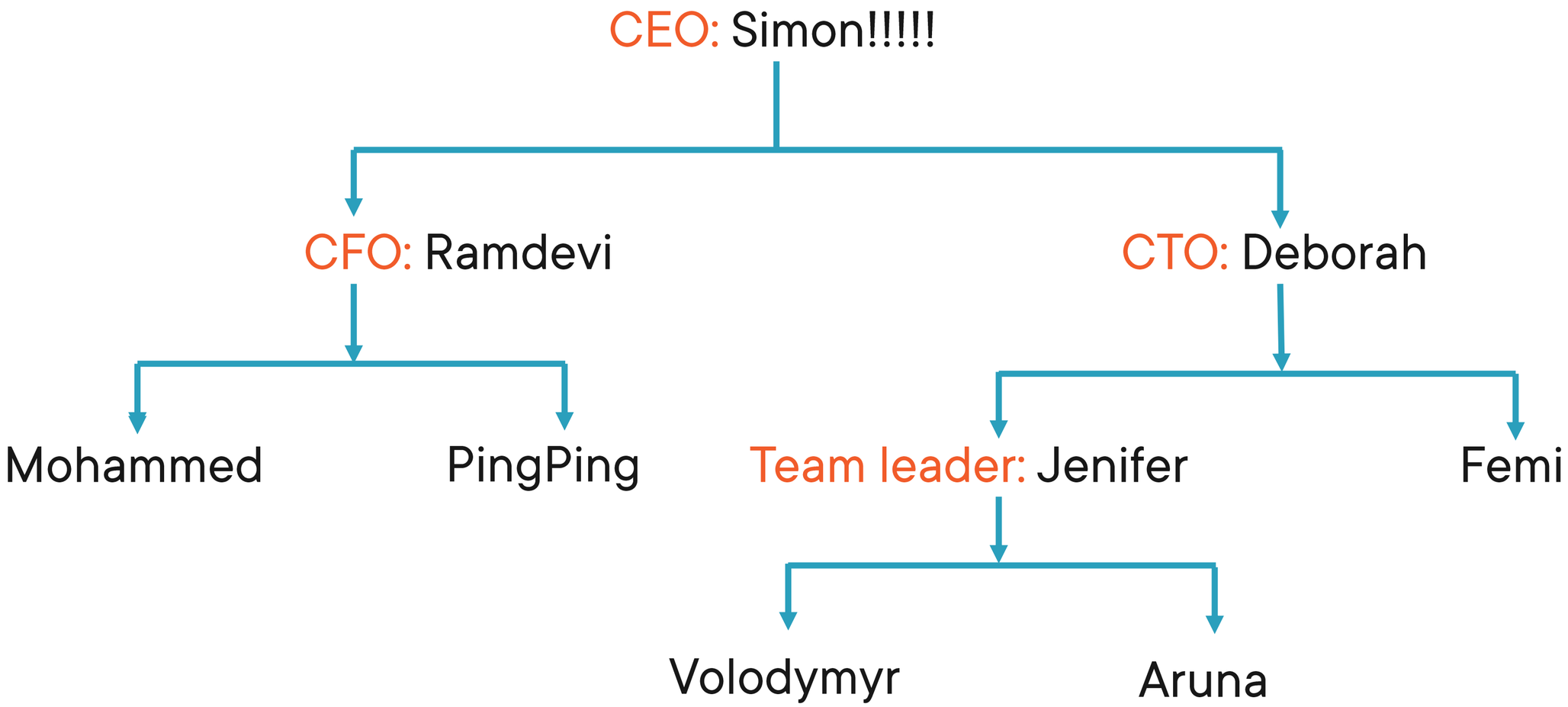
T is a placeholder
for the actual data type

Demo

Write `TreeNode<T>` **class**

- Apply it to create strongly-typed trees

A Demo Employee Hierarchy



Strong Typing with Generics – Avoiding Bugs

Demo

Build hierarchy of a different type

- Show strong typing prevents adding value to the ‘wrong’ tree

Demo

New requirement

- Sort items in the tree
- Requires sorting the generic type T

Generic Base Classes

Scenario: Tech Products

Business object representing tech product

Related products

Course would have related courses

Software library would have related software libraries

You can solve this with a generic base class

Summary

Generics

- Allow re-using logic for different types
- Maintains strong typing

To compare instances

- Use interface constraint specifying `IComparable<T>`
- This is example of interface constraint to indicate required functionality

Generic base type

- Particularly useful for business objects