Modeling Complex Domain Objects

Zoran Horvat
CEO at Coding Helmet

@zoranh75 https://codinghelmet.com

EXPLORER

DEMO
- Application
- ConsoleDemo
  - ConsoleDemo.csproj
  - Program.cs
- Models
  - Common
  - Media
  - Time
  - Types
    - Common
    - Components
    - Media
    - Products
      - AssemblySpecification.cs
  - Models.csproj
- TestPersistence
  - Inventory.cs
  - PartsReadRepository.cs
  - SpecsRepository.cs
  - TestPersistence.csproj
- Web
- Demo.sln

TIMELINE

AssemblySpecification.cs | Program.cs

Models > Types > Products > AssemblySpecification.cs > { } Models.Types.Products > Models.Types.Products.AssemblySpecification

```csharp
using Models.Types.Common;
using Models.Types.Components;

namespace Models.Types.Products;

public class AssemblySpecification
{
    public Guid Id { get; init; } = Guid.NewGuid();
    public string Name { get; init; } = string.Empty;
    public string Description { get; init; } = string.Empty;
    public IEnumerable<(Part part, DiscreteMeasure quantity)> Components { get; init; }
        = Enumerable.Empty<(Part, DiscreteMeasure)>();
    public IEnumerable<(InventoryItem item, Measure quantity)> Consumables { get; init; }
        = Enumerable.Empty<(InventoryItem, Measure)>();
}
```

0 0 Demo.sln Ln 6, Col 1 Spaces: 4 UTF-8 CRLF C#

DDD Entity

Object's identity will remain unchanged

```
public class AssemblySpecification
{
    public Guid Id
    public string Name
    public string Description
    public IEnumerable<(Part part, DiscreteMeasure quantity)> Components
    public IEnumerable<(InventoryItem item, Measure quantity)> Consumables
}
```

Object's attributes will change over time

Entity equivalence tests:

- **Test 1:** Compare identities
  Equal Ids mean the same "thing" in two moments in time
- **Test 2:** Compare all attributes
  Equal Ids and all other attributes mean an identical state at any time

Sample Specification

The home hobby electronic project, building a circuit for the traffic light using BC547 transistors.

1. Take all components as shown in the list
2. Connect 2×red LEDs to BC547 transistor
3. Connect 2×green LEDs to BC547 transistor
4. Connect 2×yellow LEDs to BC547 transistor
5. Connect emitter of 3×BC547 transistors
6. Connect 3×1kΩ resistors
7. Connect 2×100kΩ resistors
8. Connect 33kΩ resistor
9. Connect out wires of all resistors
10. Connect 100μF capacitor
11. Connect 2×470μF capacitors
12. Connect +ve pin of 2nd 470μF capacitor
13. Connect battery clipper wire
14. Connect 9V battery

Sample Specification

The home hobby electronic project, building a circuit for the traffic light using BC547 transistors.

1. Take all components as shown in the list
2. Connect 2×red LEDs to BC547 transistor
3. Connect 2×green LEDs to BC547 transistor
4. Connect 2×yellow LEDs to BC547 transistor
5. Connect emitter of 3×BC547 transistors
6. Connect 3×1kΩ resistors
7. Connect 2×100kΩ resistors
8. Connect 33kΩ resistor
9. Connect out wires of all resistors
10. Connect 100µF capacitor
11. Connect 2×470µF capacitors
12. Connect +ve pin of 2^{nd} 470µF capacitor
13. Connect battery clipper wire
14. Connect 9V battery

Components:

1. red LED ×2
2. green LED ×2
3. yellow LED ×2
4. BC547 transistor ×3
5. 1kΩ resistor ×3
6. 100kΩ resistor ×2
7. 33kΩ resistor ×1
8. 100µF capacitor ×1
9. 470µF capacitor ×2
10. battery clipper
11. 9V battery

Sample Specification

The home hobby electronic project, building a circuit for the traffic light using BC547 transistors.

1. Take all components as shown in the list
2. Connect 2×red LEDs to BC547 transistor
3. Connect 2×green LEDs to BC547 transistor
4. Connect 2×yellow LEDs to BC547 transistor
5. Connect emitter of 3×BC547 transistors
6. Connect 3×1kΩ resistors
7. Connect 2×100kΩ resistors
8. Connect 33kΩ resistor
9. Connect out wires of all resistors
10. Connect 100µF capacitor
11. Connect 2×470µF capacitors
12. Connect +ve pin of 2nd 470µF capacitor
13. Connect battery clipper wire
14. Connect 9V battery

Components:

1. red LED ×2
2. green LED ×2
3. yellow LED ×2
4. BC547 transistor ×3
5. 1kΩ resistor ×3
6. 100kΩ resistor ×2
7. 33kΩ resistor ×1
8. 100µF capacitor ×1
9. 470µF capacitor ×2
10. battery clipper
11. 9V battery

Sample Specification

The home hobby electronic project, building a circuit for the traffic light using BC547 transistors.

1. Take all components as shown in the list
2. Connect 2×red LEDs to BC547 transistor
3. Connect 2×green LEDs to BC547 transistor
4. Connect 2×yellow LEDs to BC547 transistor
5. Connect emitter of 3×BC547 transistors
6. Connect 3×1kΩ resistors
7. Connect 2×100kΩ resistors
8. Connect 33kΩ resistor
9. Connect out wires of all resistors
10. Connect 100μF capacitor
11. Connect 2×470μF capacitors
12. Connect +ve pin of 2nd 470μF capacitor
13. Connect battery clipper wire
14. Connect 9V battery

Components:

1. red LED ×2
2. green LED ×2
3. yellow LED ×2
4. BC547 transistor ×3
5. 1kΩ resistor ×3
6. 100kΩ resistor ×2
7. 33kΩ resistor ×1
8. 100μF capacitor ×1
9. 470μF capacitor ×2
10. battery clipper
11. 9V battery

Sample Specification

The home hobby electronic project, building a circuit for the traffic light using BC547 transistors.

1. Take all components as shown in the list
2. Connect 2×red LEDs to BC547 transistor
3. Connect 2×green LEDs to BC547 transistor
4. Connect 2×yellow LEDs to BC547 transistor
5. Connect emitter of 3×BC547 transistors
6. Connect 3×1kΩ resistors
7. Connect 2×100kΩ resistors
8. Connect 33kΩ resistor
9. Connect out wires of all resistors
10. Connect 100µF capacitor
11. Connect 2×470µF capacitors
12. Connect +ve pin of 2nd 470µF capacitor
13. Connect battery clipper wire
14. Connect 9V battery

Requirements

Specification:

A list of instructions

Instruction:

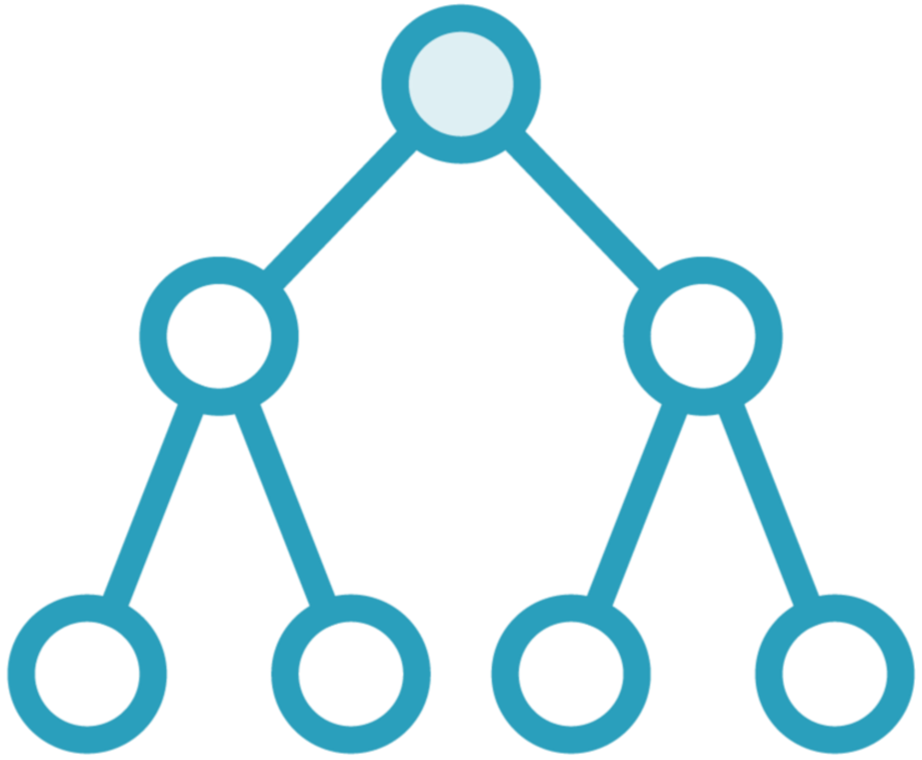
A block of text consisting of

- plaintext
- new parts
- part mentions

Components:

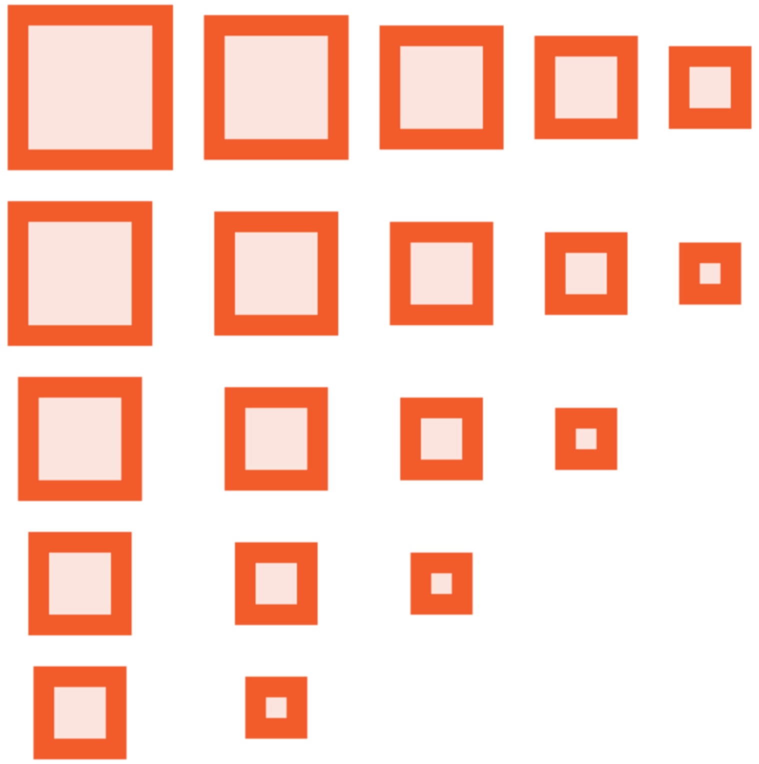
1. red LED ×2
2. green LED ×2
3. yellow LED ×2
4. BC547 transistor ×3
5. 1kΩ resistor ×3
6. 100kΩ resistor ×2
7. 33kΩ resistor ×1
8. 100µF capacitor ×1
9. 470µF capacitor ×2
10. battery clipper
11. 9V battery

Immutable List Performance



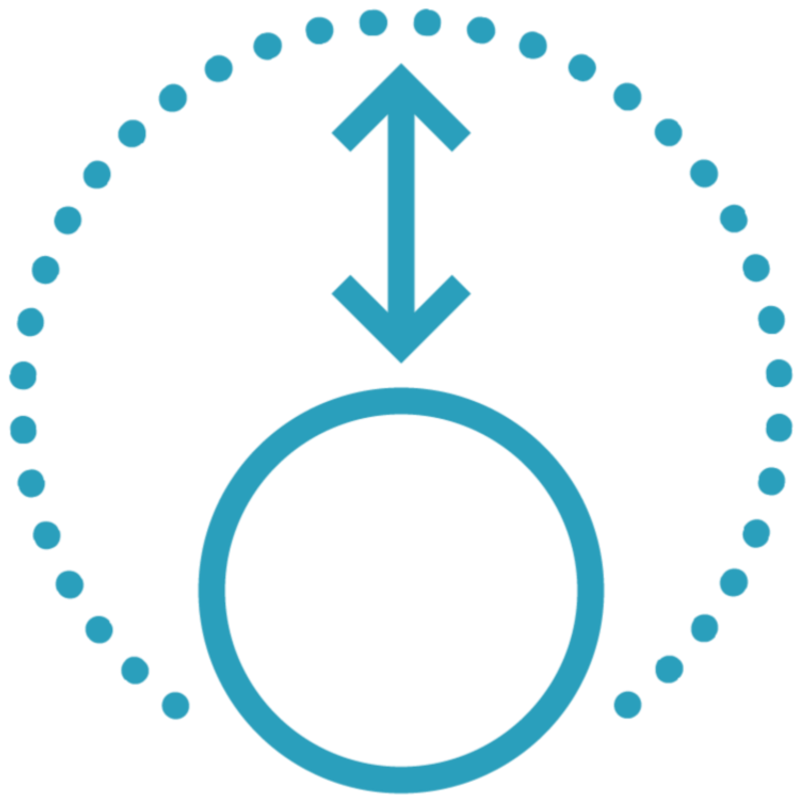
Uses balanced tree
Each node's left and right subtree of similar heights (recursively)

Insertion cost
Proportional to $\log n$, where n is number of all nodes

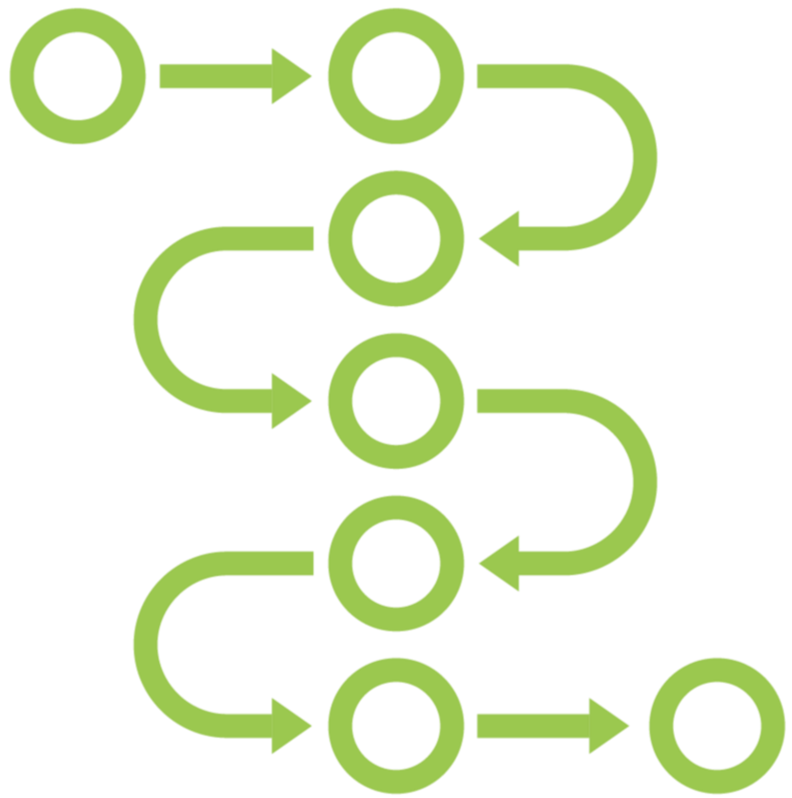


Rebalancing
Every change to the tree may require rebalancing

Immutable List Performance

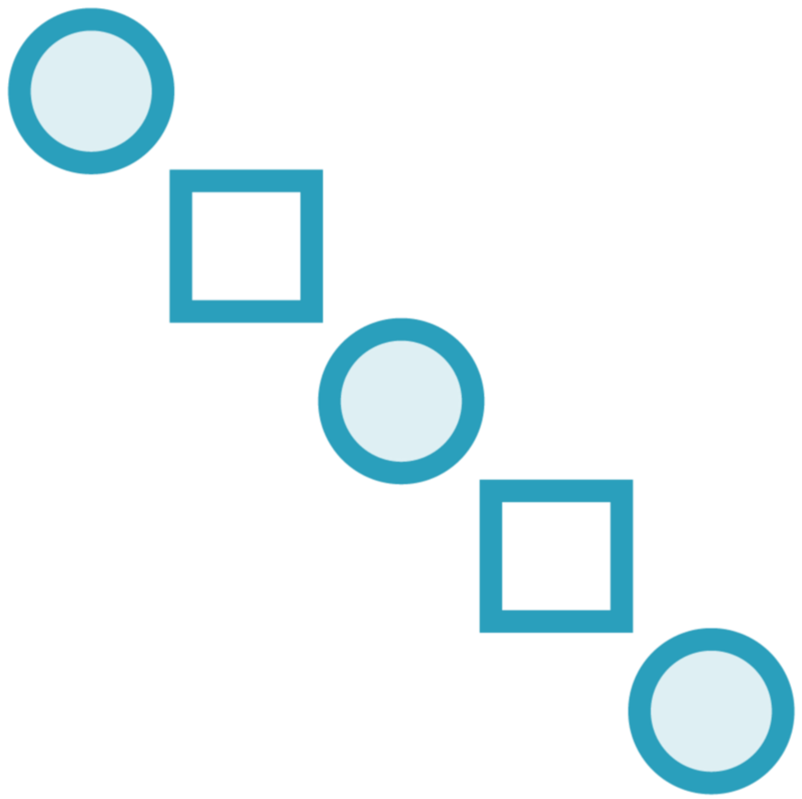


Changing the list in a loop
Takes $O(n \log n)$ time
for an immutable list
Takes $O(\mathrm{n})$ time for a mutable list



Iterating the list
Takes $O(\mathrm{n})$ time both in immutable
and in common (mutable) list
Immutable list is much slower

Immutable List Performance



Creating a non-empty list
Use `ToImmutableList()` extension method
Use `AddRange()` on an empty list

Bulk adding to a list
Use `AddRange()` method to add multiple items
Followed by a bulk rebalance

EXPLORER ···

Program.cs | AssemblySpecification.cs | InstructionSegment.cs | AssemblyInstruction.cs ●

Models > Types > Products > AssemblyInstruction.cs > {} Models.Types.Products > Models.Types.Products.AssemblyInstruction > Append(params InstructionSegment[] segments)

- DEMO
  - Application
  - ConsoleDemo
    - ConsoleDemo.csproj
    - Program.cs
  - Models
    - Common
    - Media
    - Time
    - Types
      - Common
      - Components
      - Media
      - Products
        - AssemblyInstruction.cs
        - AssemblySpecification.cs
        - InstructionSegment.cs
    - Models.csproj
  - TestPersistence
  - Web
  - Demo.sln

```csharp
public class AssemblyInstruction
{
    private ImmutableList<InstructionSegment> Segments { get; }

    public AssemblyInstruction() : this(ImmutableList<InstructionSegment>.Empty) { }

    public AssemblyInstruction(IEnumerable<InstructionSegment> segments)
        : this(segments.ToImmutableList()) { }

    private AssemblyInstruction(ImmutableList<InstructionSegment> segments) =>
        Segments = segments;

    public AssemblyInstruction Append(IEnumerable<InstructionSegment> segments) =>
        new(this.Segments.AddRange(segments));

    public AssemblyInstruction Append(params InstructionSegment[] segments) =>
        segments.Length == 0 ? this
        : segments.Length == 1 ? new(this.Segments.Add(segments[0]))
        : new(this.Segments.AddRange(segments));
}
```

TIMELINE

⊗ 0 ⚠ 0 | Demo.sln | Ln 23, Col 49 | Spaces: 4 | UTF-8 | CRLF | C#

EXPLORER

DEMO
- Application
- ConsoleDemo
  - ConsoleDemo.csproj
  - Program.cs
- Models
  - Common
  - Media
  - Time
  - Types
    - Common
    - Components
    - Media
    - Products
      - AssemblyInstruction.cs
      - AssemblySpecification.cs
      - InstructionSegment.cs
  - Models.csproj
- TestPersistence
- Web
- Demo.sln

TIMELINE

Program.cs | AssemblySpecification.cs | InstructionSegment.cs | AssemblyInstruction.cs

Models > Types > Products > AssemblyInstruction.cs > {} Models.Types.Products > Models.Types.Products.AssemblyInstruction > Append(params InstructionSegment[] segments)

```csharp
public class AssemblyInstruction
{
}
```

Object freezing

```csharp
public AssemblyInstruction Append(IEnumerable<InstructionSegment> segments) =>
    new(this.Segments.AddRange(segments));
```

Allow wild mutations during initialization to improve performance

Then *freeze* the object and return it as an immutable instance

Freezing is a common tactic in immutable collections

⊗ 0 ⚠ 0 Demo.sln Ln 23, Col 49 Spaces: 4 UTF-8 CRLF C#

Summary

Managing complex immutable objects

- Some classes cannot implement equivalence
- Such classes cannot be implemented as records

Custom nondestructive mutation

- Usually implemented with a copy constructor
- Backed by C# 11 `required` properties

Summary

Immutable collections

- Offer the same services as common, mutable collections
- Follow the principles of immutable design
- A mutating method returns a new instance

Mitigating performance penalties

- Immutable collections are slower than their mutable counterparts
- There are bulk mutators that help keep operations performant

Summary

Encapsulating immutable collections

- Keep the collection private
- Expose public mutators that guarantee integrity
- Expose members that expose collection's content
- Use LINQ and `IEnumerable<T>`

Course Summary

Practical functional programming with C#

- Left most of the theory out
- Outlined C# syntax and coding style that support functional programming

Course Summary

Fundamental principles

- Types (often implemented as records)
- Type composition and function composition
- Separation of functions from types
- Design of pure functions
- Function decomposition and composition

Course Summary

Advanced functional programming

- Partial function application, resembling dependency injection in OOP
- Discriminated unions
- Optional objects
- Immutable collections

Course Summary

Functional vs. object-oriented modeling

- Largely equivalent in modeling capabilities
- Functional offers better extensibility and (usually) lower bug count
- Functional code is often self-documenting and more readable

Learn more theory

Learn F#

Michael Heydt

F# 6 Fundamentals

August 17th, 2022

Apply functional programming concepts

The best object-oriented code is the functional code