Equality, Immutability, and Record Types

Mel Grubb
Developer

@melgrubb | www.melgrubb.com

Side Effect

Changes made to a system's state other than the stated purpose of the module that made them.

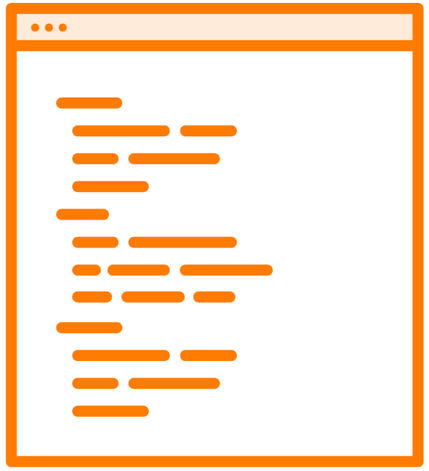
Addressed using immutable objects in functional languages.

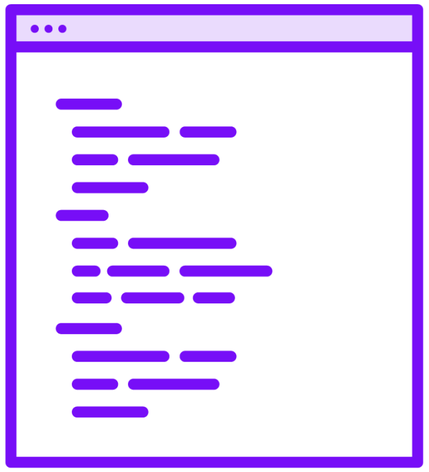
Overview

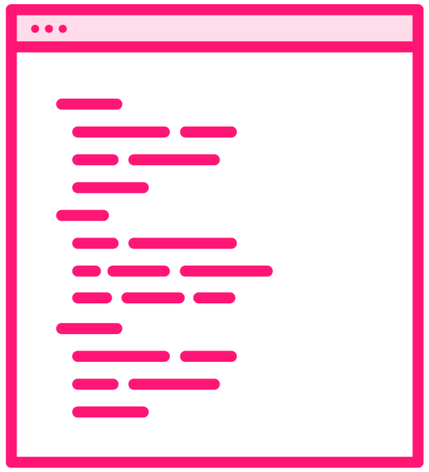
- Immutability
- Equality
- Records
- Record structs

Immutability

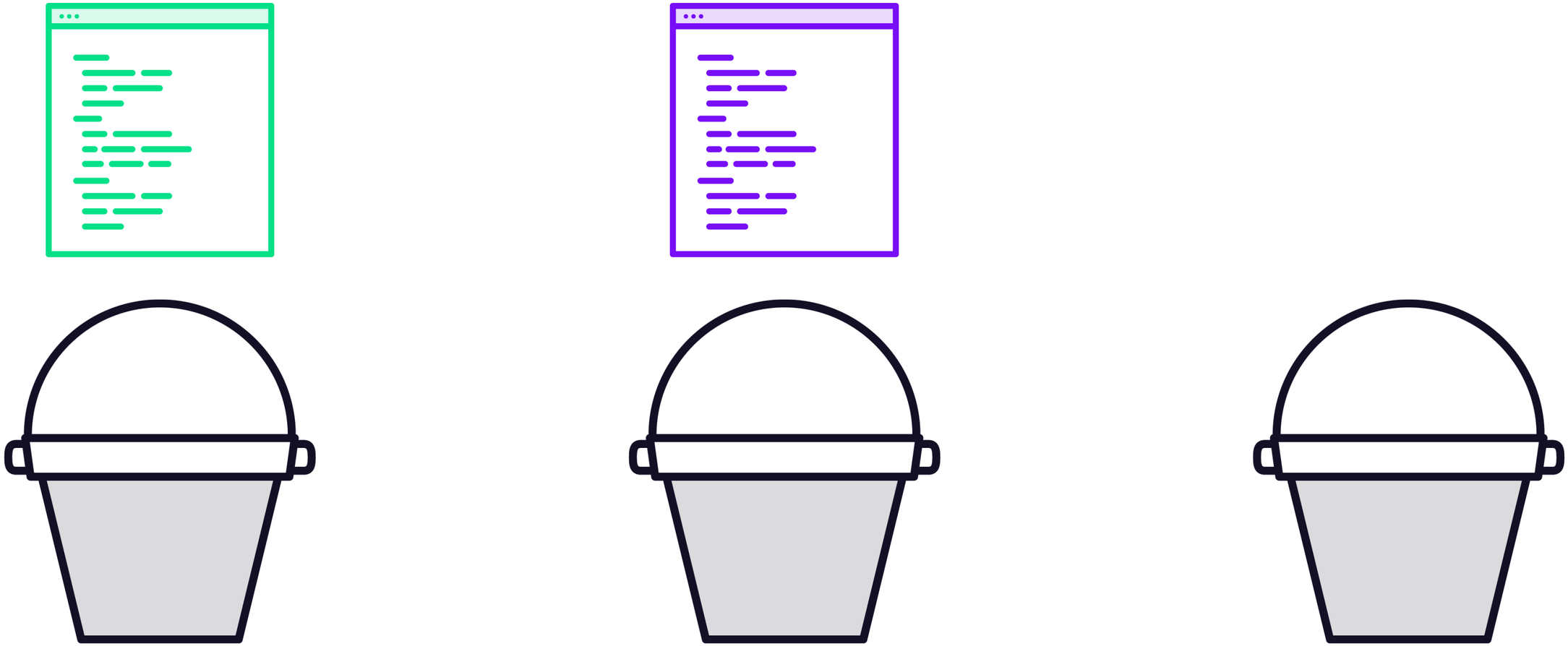
Immutability







Immutability



Immutable Types

String

DateTime

Thread safe

Validated once

Design Tips For Immutable Types

- **Communicate expectations**
  - Include only what is required
  - Set expectations for consuming code
- **Validate data up-front**
- **Repeat for sub-objects**

Equality

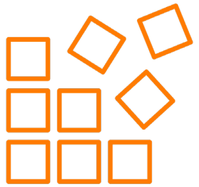
Design Tips For Equality

- **Don’t partially implement IEquatable**
  - Equals
  - Equals(object)
  - Operators (== and !=)
  - GetHashCode
- **Hash code should be consistent**
  - Use the same fields as Equals

Records

Additional Record Abilities

PrintMembers



Deconstruction

For More Information

Working with C# Records

Roland Guijt

Record Structs

Record Classes vs. Record Structs

Record Classes (Records)	Record Structs
Reference type	Value type
Value equality	Value equality
	Lighter weight

Design Tips For Records

- Favor records for light duty types
- Use positional parameters for convenience
- Keep them immutable if possible
- Mutable records are unexpected
- Favor record structs for DTOs or parameter objects

Summary

Immutability in C#

Equality

- Reference vs. value
- IEquatable
- ==, !=, and GetHashCode

Record classes

Record structs

Object-oriented Solutions to Common Problems