Defining Fields Appropriately

Deborah Kurata

@deborahkurata | blogs.msmvps.com/deborahk/

Fields

```
private DateTime? availablityDate;
private decimal? cost;
private string description;
private int productId;
private string productName;
private Vendor productVendor;

public const int Red = 0xFF0000;
public const double InchesPerMeter = 39.37;

public readonly decimal MinimumPrice;
```

Module Overview

Backing Fields

Nullable Types

Constants

Read-Only Fields

Constant vs Read-Only

FAQ

Backing Fields

```
private string description;
private int productId;
private string productName;
private Vendor productVendor;
```

- A variable in a class
- Holds data for each object

Data Encapculation/Information Hiding

- Object's data is only accessible to that object
- Fields are private
- Accessible outside of the class through property getters and setters

Backing Fields

```
private string description;
private int productId;
private string productName;
private Vendor productVendor;
```

- Optional accessibility modifier
- Data type
- Name
- Optional initialization

Backing Fields

```
private string description = "coming soon";
private int productId;
private string productName;
private Vendor productVendor;
```

- Optional accessibility modifier
- Data type
- Name
- Optional initialization

Backing Field Best Practices

Do:

Naming

Define a meaningful name

Use camelCasing

Keep fields private

Use properties to provide access to the fields

Avoid:

Naming

Single character name

Abbreviations

Initializing to the field's default value

Nullable Types

```
private decimal? cost;
private DateTime? availablityDate;
```

- Allows definition of a value OR null
- Specified with a "?" suffix on the type
- Distinguishes "not set" from the default value

Nullable Type Best Practices

Do:

Use on simple types to distinguish "not set" and "default value"

Use properties of the type such as `HasValue` and `Value` as needed

```
if (AvailablityDate.HasValue)
{
   DateTime aDate = AvailabilityDate.Value;
}
```

Avoid:

Using them if not necessary

Constants

```
public const double Pi = 3.14;
public const int Red = 0xFF0000;
public const double InchesPerMeter = 39.37;
```

- Defined in a class
- Holds a hard-coded value that does not change
- Must be assigned to an expression that can be fully evaluated at compile time
  - Think of a constant as a "compile-time" constant value
- Compiled into every location that references it
- Are static

Constants

```
public const double Pi = 3.14;
public const int Red = 0xFF0000;
public const double InchesPerMeter = 39.37;
```

- Optional accessibility modifier
- const keyword
- Data type
- Name
- Assigned value

Constant Best Practices

Do:

Class naming

Define a meaningful name

Use PascalCasing

Use for compile-time values that will never change

Avoid:

Class naming

Single character name

Abbreviations

All upper case

For fields that could change over time

Read-Only Fields

```
public static readonly decimal MinimumPrice;
public readonly string DefaultMeasure = GetDefaultMeasure();
public Product()
{
   MinimumPrice = RetrieveMinimumPrice();
}
```

- A variable in a class
- Holds a value that is initialized and then not changed
- Must be initialized
  - In the declaration
  - Or in a constructor
- Think of a read-only field as a "runtime" constant value

Read-Only Fields

```
public static readonly decimal MinimumPrice;
public readonly string DefaultMeasure = GetDefaultMeasure();
public Product()
{
    MinimumPrice = RetrieveMinimumPrice();
}
```

- Optional accessibility modifier
- Optional `static` keyword
- `readonly` keyword
- Data type
- Name
- Assigned value

Read-Only Field Best Practices

Do:

Class naming

Define a meaningful name

Use PascalCasing

Use for runtime constants

Use static if the constant value is valid for all instances

Avoid:

Class naming

Use abbreviations

Constant vs. Read-Only

Constant Field	Read-only Field
Compile-time constant	Runtime constant
Assigned to an expression evaluated at compile time	Assigned to any valid expression at runtime
Assigned on declaration	Assigned on declaration or constructor
Only number, Boolean, or string	Any data type
Always static	Optionally static

Frequently Asked Questions

- Explain the data encapsulation principle
  - An object's data should be accessible only to the object
  - Backing fields containing the object data should be marked as private
- What is a backing field?
  - A variable in a class used to retain each object's data
- When should you use a backing field?
  - For every data field retained for an object

Frequently Asked Questions (cont)

- When should you use a constant?
  - When defining a field with a simple data type that will never change
- When should you use a read-only field?
  - When defining a field that is initialized from a file, table, or code but should not then be changed anywhere else in the application

Frequently Asked Questions (cont)

- What is the difference between a constant and a read-only field?
  - A constant
    - Is static
    - Assigned on the declaration
    - Assigned to an expression that is fully evaluated at compile time
  - A read-only field
    - Can be static or non-static
    - Assigned in the declaration or in a constructor
    - Assigned to any valid expression

This Module Covered

Backing Fields

Nullable Types

Constants

Read-Only Fields

Constant vs Read-Only