Introduction to .NET

Florin Olariu
“Alexandru Ioan Cuza”, University of Iași
Department of Computer Science

Agenda

- Unit testing – shortcut to expert level
- Entity Framework Core – part1

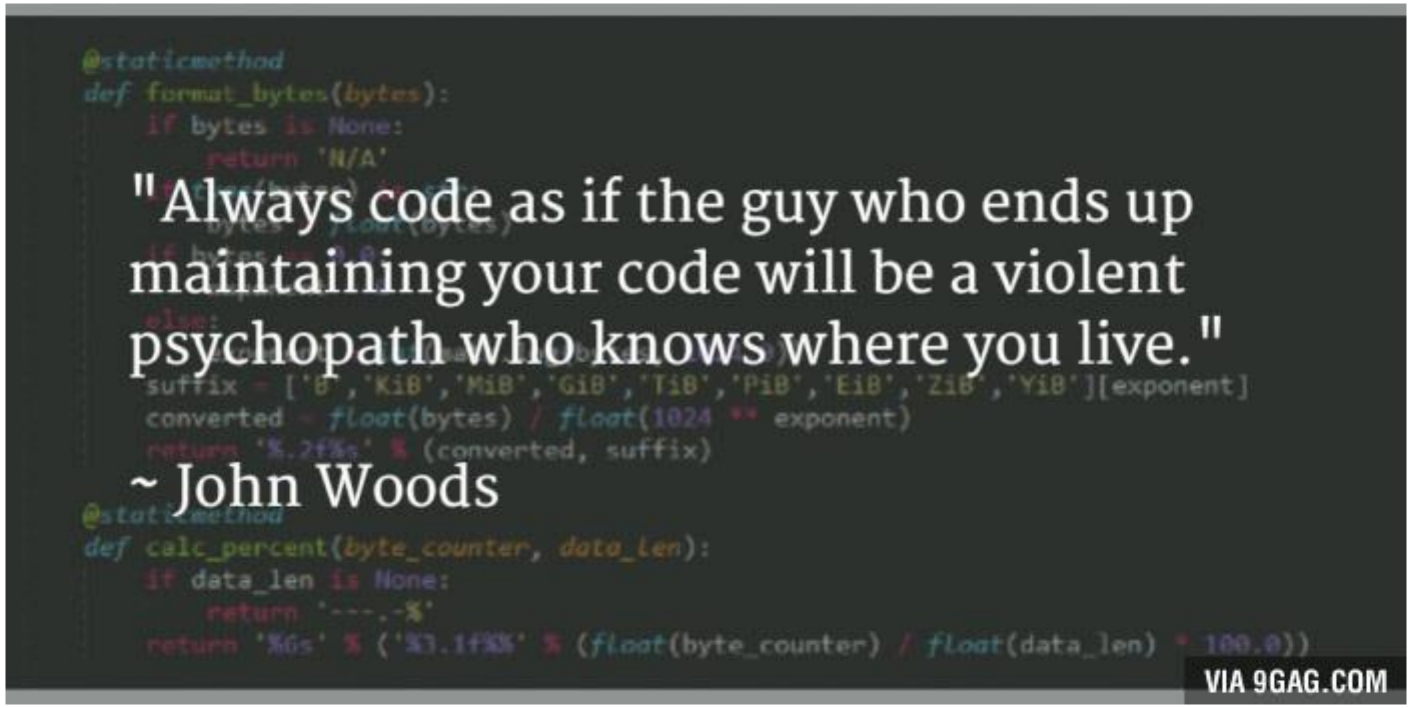
Unit testing – shortcut to expert level

Unit testing – shortcut to expert level

- Why unit testing?
    - Benefits of unit tests
- What is a unit test?
- Unit test life cycle
- What makes a good unit test?
- Best practices

Why unit testing?

Why unit testing?

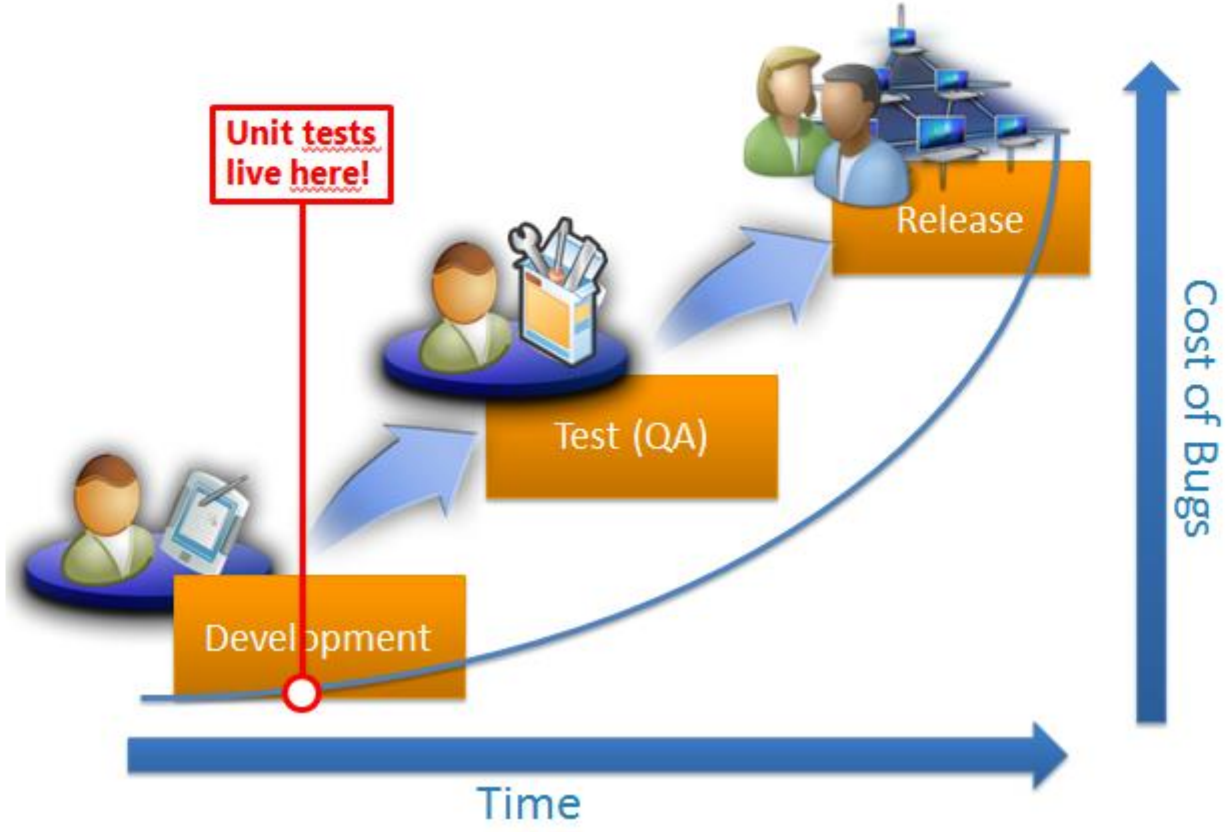


"Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live."

~ John Woods

VIA 9GAG.COM

Why unit testing?



Why unit testing?

- Benefits of unit testing
  - Find defects early
  - Prevent regressions
  - Provide living documentation (source code)
  - Automate testing efforts

Why unit testing?

- Benefits
  - Find defects early
  - Prevent regressions
  - Provide living documentation (source code)
  - Automate testing efforts

Requires tests to be as complete as possible and to be run as early and often as possible

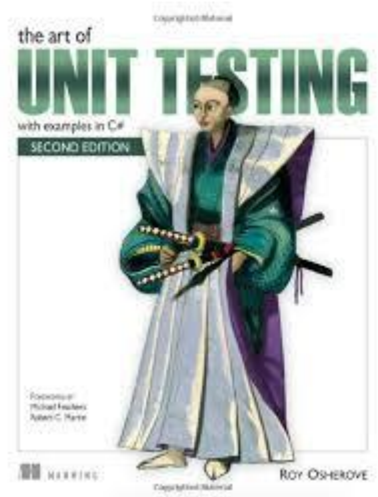
What is a Unit Test?

What is a Unit Test?

- Definition 1

What is a Unit Test?

- Definition 1
- "A unit test is a piece of code (usually a method) that invoke another piece of code and checks the correctness of some assumptions afterwards."
  - "The art of unit testing"



What is a Unit Test?

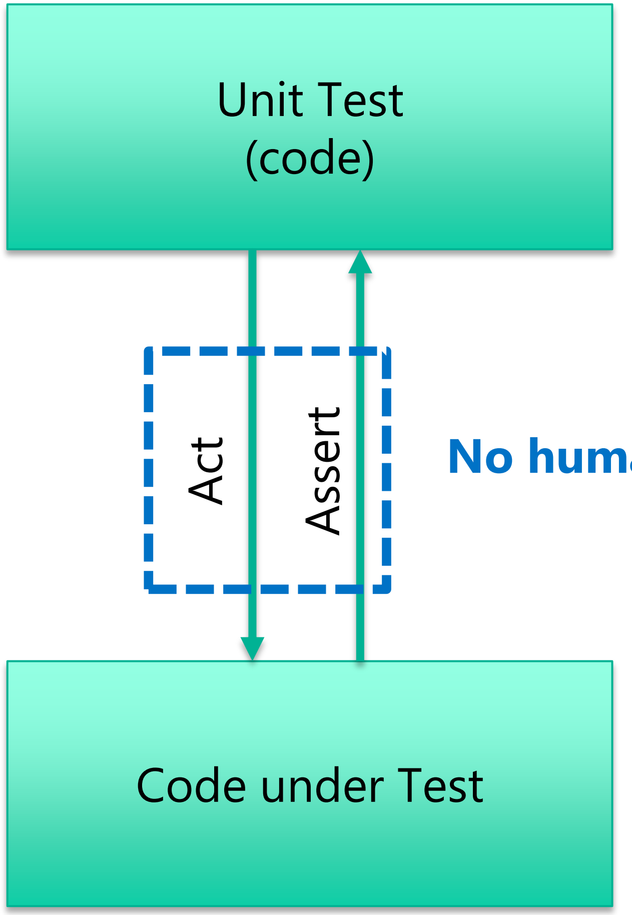
- Definition 2

What is a Unit Test?

- Definition 2

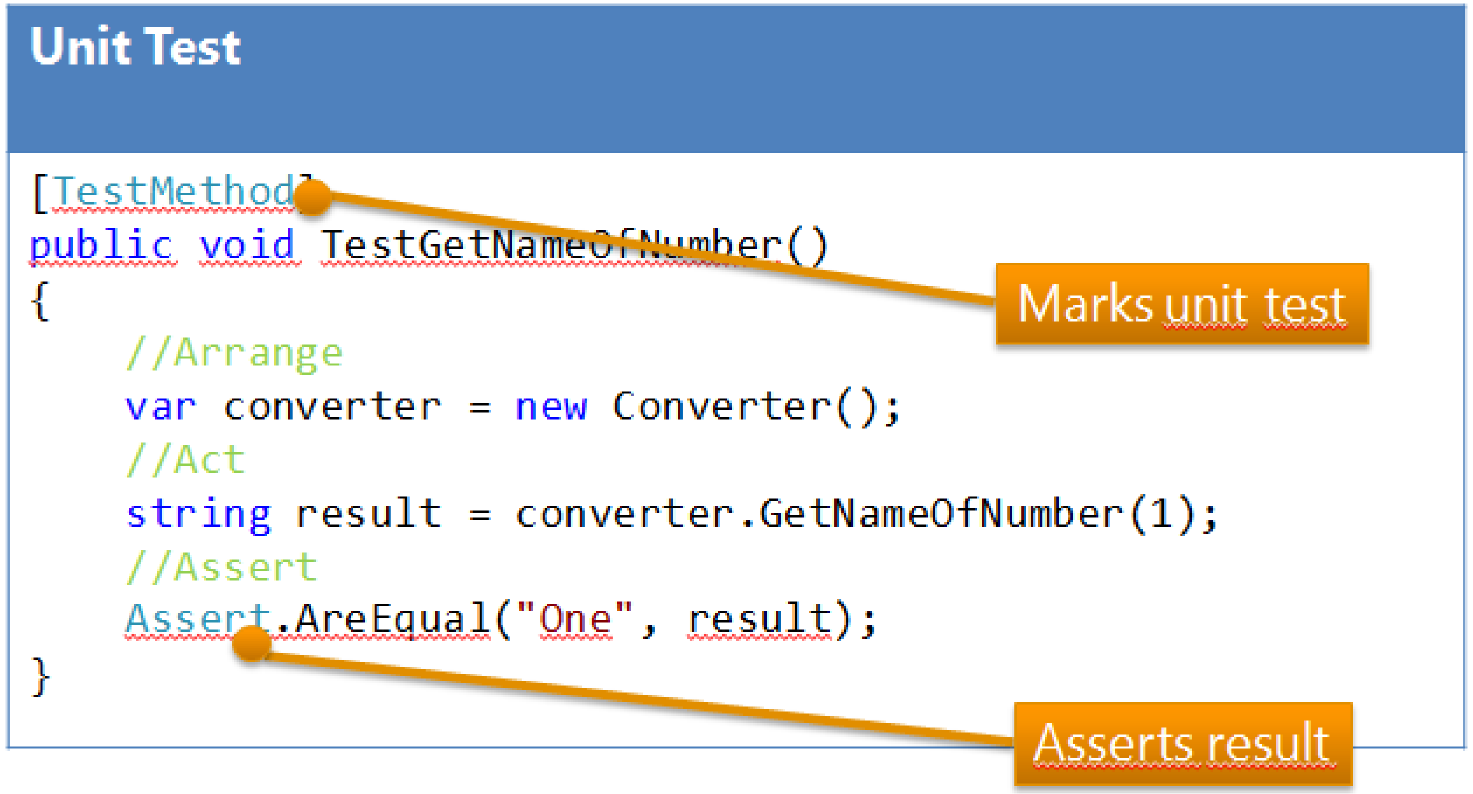
- **Essentially**, a unit test is a method that instantiates a small portion of our application and verifies its behavior **independently from other parts.**

Unit test representation



No human interaction!

Unit test example



Unit Test

```
[TestMethod]
public void TestGetNameOfNumber()
{
    //Arrange
    var converter = new Converter();
    //Act
    string result = converter.GetNameOfNumber(1);
    //Assert
    Assert.AreEqual("One", result);
}
```

What Makes a Good Unit Test?

What Makes a Good Unit Test?

- Easy to write

What Makes a Good Unit Test?

- Easy to write
- Readable

What Makes a Good Unit Test?

- Easy to write
- Readable
- Reliable

What Makes a Good Unit Test?

- Easy to write
- Readable
- Reliable
- Fast

Best practices

- A test should be:

Best practices

- A test should be:
    - Isolated

Best practices

- A test should be:
  - Isolated
  - Test Only One Condition at a Time

Best practices

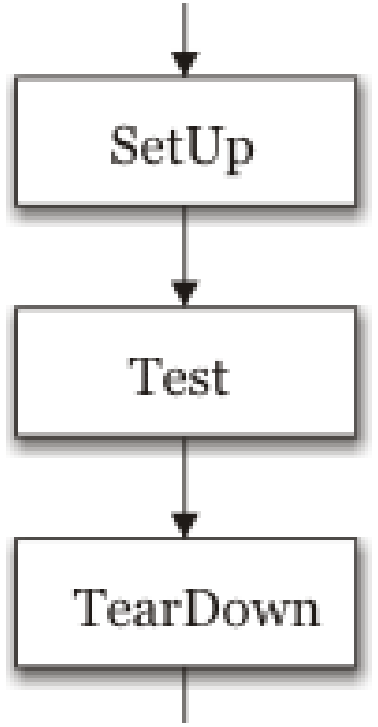
- A test should be:
    - Isolated
    - Test Only One Condition at a Time
    - Repeatable

Best practices

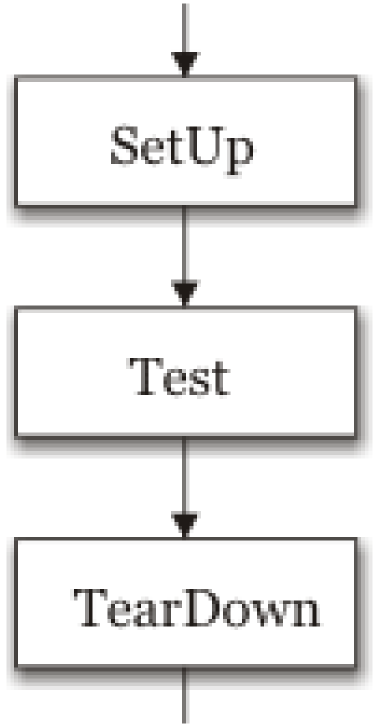
- A test should be:
  - Isolated
  - Test Only One Condition at a Time
  - Repeatable
  - Thorough
  - Mock external references

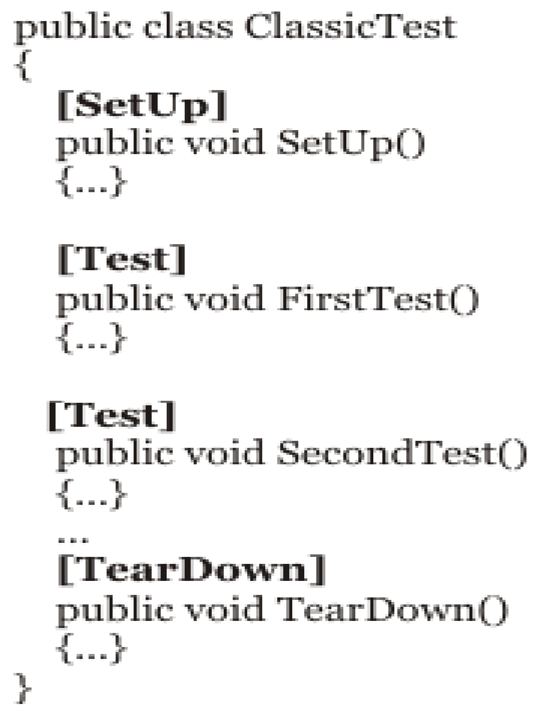
Unit test life cycle

Unit test life cycle



Unit test life cycle





```
public class ClassicTest
{
   [SetUp]
   public void SetUp()
   {...}

   [Test]
   public void FirstTest()
   {...}

  [Test]
   public void SecondTest()
   {...}
   ...
   [TearDown]
   public void TearDown()
   {...}
}
```

Entity Framework Core - part1

Entity Framework Core – part1

- Introduction in Entity Framework
- Entity Framework code first
  - Conventions
  - Attributes
  - Fluent API
  - Creating models with Entity Framework code first/Creating Context class with Entity Framework – from scratch sample
  - Transactions
  - Manipulating data(Inserting entities, Updating entities, Deleting entities, querying entities)

Introduction in Entity Framework

Introduction in Entity Framework

- What is Entity Framework?
- Short history
- Pros/Cons in using EF Core

Introduction in Entity Framework

- What is Entity Framework?
- Short history
- Pros/Cons in using EF Core

Introduction in Entity Framework

- What is Entity Framework?
- Short history
- Pros/Cons in using EF Core

It is an ORM => object relational mapping tool from Microsoft.

Introduction in Entity Framework

- What is Entity Framework?
- Short history
- Pros/Cons in using EF Core

Introduction in Entity Framework

- What is Entity Framework
- Short history
- Pros/Cons in using EF Code

Was first released as part of .NET Framework 3.5 with Service Pack 1 back in late 2008.

Introduction in Entity Framework

- What is Entity Framework?
- Short history
- Pros/Cons in using EF Core

The version included with Visual Studio 2015 is Entity Framework 6.1.3 (EF6).

Introduction in Entity Framework

- What is Entity Framework?
- Short history
- Pros/Cons in using EF Core

It is mature, stable, and supports the "old" EDMX design-time way

Introduction in Entity Framework

- What is Entity Framework?
- Short history
- Pros/Cons in using EF Core

Introduction in Entity Framework

- What is Entity Framework?
- Short history
- Pros/Cons in using EF Core

Pros	Cons

Introduction in Entity Framework

- What is Entity Framework?
- Short history
- Pros/Cons in using EF Core

Pros	Cons
Is available for .NET Core => can be used for Mac, Linux or Windows	

Introduction in Entity Framework

- What is Entity Framework?
- Short history
- Pros/Cons in using EF Core

Pros	Cons
Is available for .NET Core => can be used for Mac, Linux or Windows	
Supports modern cloud-based, non-relational databases(Azure Table Storage, Redis)	

Introduction in Entity Framework

- What is Entity Framework?
- Short history
- Pros/Cons in using EF Core

Pros	Cons
Is available for .NET Core => can be used for Mac, Linux or Windows	Does not support the old EDMX design time
Supports modern cloud-based, non-relational databases(Azure Table Storage, Redis)	

Introduction in Entity Framework

- What is Entity Framework?
- Short history
- Pros/Cons in using EF Core

Pros	Cons
Is available for .NET Core => can be used for Mac, Linux or Windows	Does not support the old EDMX design time
Supports modern cloud-based, non-relational databases(Azure Table Storage, Redis)	Does not support yet complex inheritance models

Introduction in Entity Framework

- What is Entity Framework?
- Short history
- Pros/Cons in using EF Core

Pros	Cons
Is available for .NET Core => can be used for Mac, Linux or Windows	Does not support the old EDMX design time
Supports modern cloud-based, non-relational databases(Azure Table Storage, Redis)	Does not support yet complex inheritance models
Tip: Use EF6 for Windows platform applications until EF Core becomes more stable and implements more features. Use EF Core for cross-platform development.	

Entity Framework code first

Entity Framework code first - conventions

Entity Framework code first - conventions

- If a connection string exists with the same name as the class derived from DbContext, then it is loaded and used to connect to the database automatically.

Entity Framework code first - conventions

- If a connection string exists with the same name as the class derived from DbContext, then it is loaded and used to connect to the database automatically.
- The name of a table is assumed to match the name of a DbSet <T> property in the DbContext class, for example, Customers.

Entity Framework code first - conventions

- If a connection string exists with the same name as the class derived from DbContext, then it is loaded and used to connect to the database automatically.
- The name of a table is assumed to match the name of a DbSet <T> property in the DbContext class, for example, Customers.
- The names of the columns are assumed to match the names of properties in the class, for example, CustomerID.

Entity Framework code first - conventions

- If a connection string exists with the same name as the class derived from DbContext, then it is loaded and used to connect to the database automatically.
- The name of a table is assumed to match the name of a DbSet <T> property in the DbContext class, for example, Customers.
- The names of the columns are assumed to match the names of properties in the class, for example, CustomerID.
- The string .NET type is assumed to be an nvarchar type in the database.

Entity Framework code first - conventions

- If a connection string exists with the same name as the class derived from DbContext, then it is loaded and used to connect to the database automatically.
- The name of a table is assumed to match the name of a DbSet <T> property in the DbContext class, for example, Customers.
- The names of the columns are assumed to match the names of properties in the class, for example, CustomerID.
- The string .NET type is assumed to be an nvarchar type in the database.
- The int .NET type is assumed to be an int type in the database.

Entity Framework code first - conventions

- If a connection string exists with the same name as the class derived from DbContext, then it is loaded and used to connect to the database automatically.
- The name of a table is assumed to match the name of a DbSet <T> property in the DbContext class, for example, Customers.
- The names of the columns are assumed to match the names of properties in the class, for example, CustomerID.
- The string .NET type is assumed to be an nvarchar type in the database.
- The int .NET type is assumed to be an int type in the database.
- A property that is named ID or the name of the class has ID as the suffix, it is assumed to be a primary key. If this property is any integer type or the Guid type, then it is also assumed to be an IDENTITY

Entity Framework code first attributes

Entity Framework code first attributes

- With attributes we can specify different aspects database oriented: Not Null field, dimensions for nvarchar, custom types

Entity Framework code first attributes

- With attributes we can specify different aspects database oriented: Not Null field, dimensions for nvarchar, custom types

```
[Required]
[StringLength(40)]
public string CustomerName { get; set; }
```

or:

Entity Framework code first attributes

- With attributes we can specify different aspects database oriented: Not Null field, dimensions for nvarchar, custom types

```
[Required]
[StringLength(40)]
public string CustomerName { get; set; }
```

or:

```
[Column( TypeName = "money")]
public decimal? UnitPrice { get; set; }
```

Entity Framework code first Fluent API

Entity Framework code first Fluent API

- It can be used:
  - Combined with attributes

Entity Framework code first Fluent API

- It can be used:
  - Combined with attributes
  - As a replacement for attributes

Entity Framework code first Fluent API

- It can be used:
  - Combined with attributes
  - As a replacement for attributes

```
[Required]
[StringLength( 40)]
public string CompanyName { get; set; }
```

They could be deleted and replaced with this Fluent API statement in the Context class OnModelBuilding method:

```
modelBuilder.Entity<Customer>()
.Property(customer=>customer.CompanyName)
.IsRequired()
.HasMaxLength(40);
```

Entity Framework code first – models and context class

- Creating models with Entity Framework code first/Creating Context class with Entity Framework – from scratch sample - Demo

Entity Framework code first transactions

Entity Framework code first transactions

- Transactions maintain the integrity of your database by applying locks to prevent reads and writes while a sequence of operations is occurring.

Entity Framework code first transactions

- Transactions maintain the integrity of your database by applying locks to prevent reads and writes while a sequence of operations is occurring.
- Transactions should be ACID.

Entity Framework code first transactions

- Transactions maintain the integrity of your database by applying locks to prevent reads and writes while a sequence of operations is occurring.
- Transactions should be ACID.
  - A : is for atomic

Entity Framework code first transactions

- Transactions maintain the integrity of your database by applying locks to prevent reads and writes while a sequence of operations is occurring.
- Transactions should be ACID.
  - A : is for atomic
  - C: is for consistent

Entity Framework code first transactions

- Transactions maintain the integrity of your database by applying locks to prevent reads and writes while a sequence of operations is occurring.
- Transactions should be ACID.
  - A : is for atomic
  - C: is for consistent
  - I: is for isolated

Entity Framework code first transactions

- Transactions maintain the integrity of your database by applying locks to prevent reads and writes while a sequence of operations is occurring.
- Transactions should be ACID.
  - A : is for atomic
  - C: is for consistent
  - I: is for isolated
  - D: is for durable

Entity Framework code first manipulating data

Entity Framework code first manipulating data

- Inserting entities
- Updating entities
- Deleting entities
- Querying entities

Entity Framework code first manipulating data

- Inserting entities
- Updating entities
- Deleting entities
- Querying entities

Entity Framework code first manipulating data

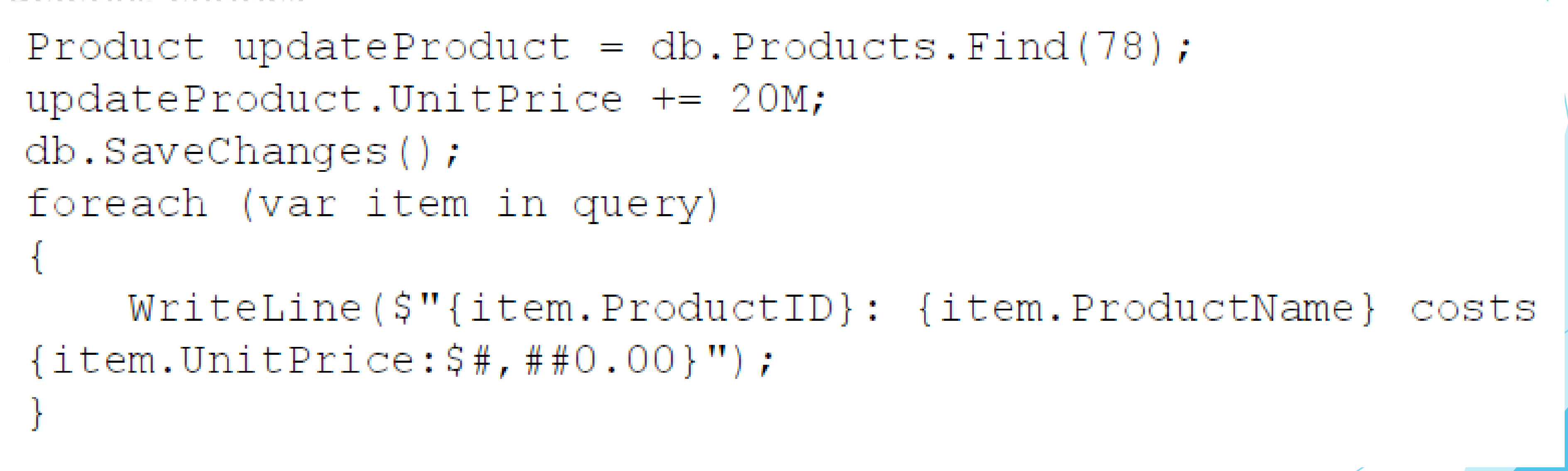
- Inserting entities
- Up
- Del
- Qu

```
var newProduct = new Product
{
    ProductName = "Bob's Burger",
    UnitPrice = 500M
};
// mark product as added in change tracking
db.Products.Add(newProduct);
// save tracked changes to database
db.SaveChanges();
foreach (var item in query)
{
    WriteLine($"{item.ProductID}: {item.ProductName} costs
{item.UnitPrice:$#,##0.00}");
}
```

Entity Framework code first manipulating data

- Inserting entities
- Updating entities
- Deleting entities
- Querying entities

Entity Framework code first manipulating data

- Inserting entities
- Updating entities
- Deleting entities
- 

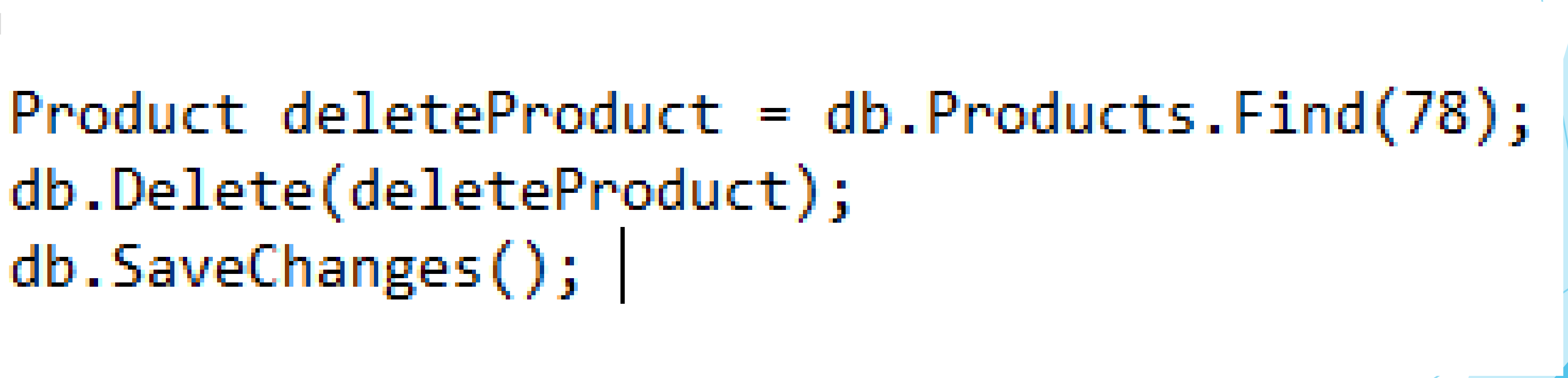
```
Product updateProduct = db.Products.Find(78);
updateProduct.UnitPrice += 20M;
db.SaveChanges();
foreach (var item in query)
{
    WriteLine($"{item.ProductID}: {item.ProductName} costs
{item.UnitPrice:$#,##0.00}");
}
```

Entity Framework code first manipulating data

- Inserting entities
- Updating entities
- Deleting entities
- Querying entities

Entity Framework code first manipulating data

- Inserting entities
- Updating entities
- Deleting entities
- Qu



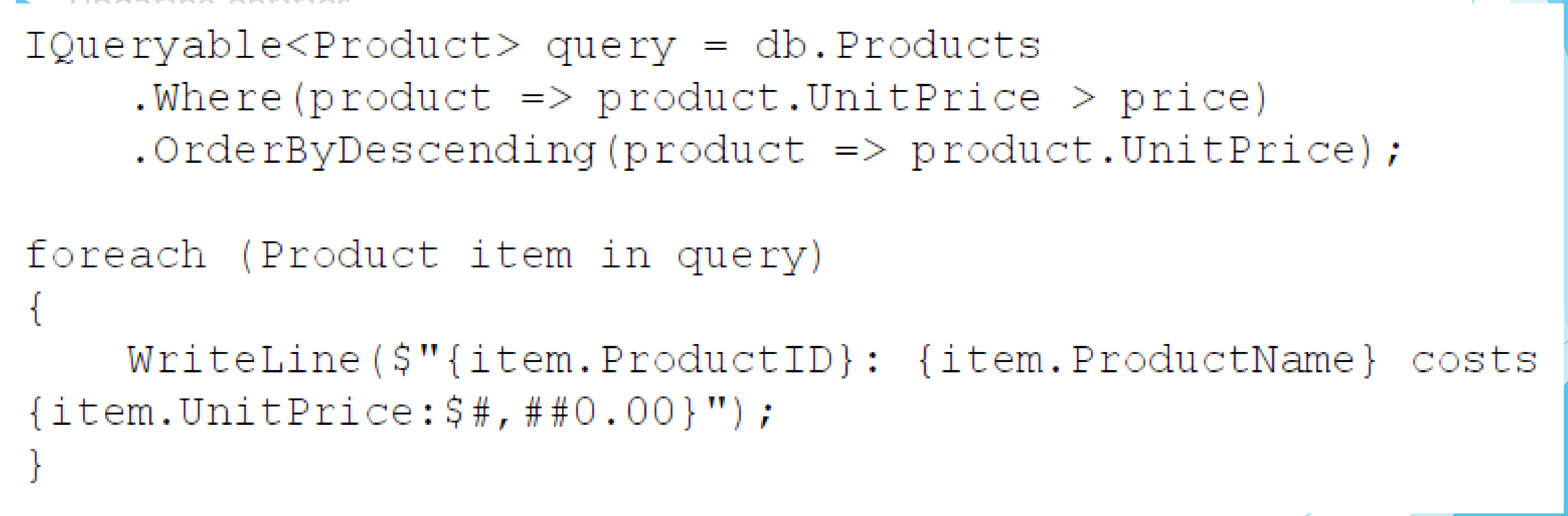
```
Product deleteProduct = db.Products.Find(78);
db.Delete(deleteProduct);
db.SaveChanges();
```

Entity Framework code first manipulating data

- Inserting entities
- Updating entities
- Deleting entities
- Querying entities

Entity Framework code first manipulating data

- Inserting entities



```
IQueryable<Product> query = db.Products
    .Where(product => product.UnitPrice > price)
    .OrderByDescending(product => product.UnitPrice);

foreach (Product item in query)
{
    WriteLine($"{item.ProductID}: {item.ProductName} costs
{item.UnitPrice:$#,##0.00}");
}
```

What’s next …

- Entity Framework Core – part 2

One more thing…

- *Read the story from notes!!!!!*

One more thing…

- *Read the story from notes!!!!!*

Early one morning, a programmer asked the great master:

“I am ready to write some unit tests. What code coverage should I aim for?”

The great master replied:

“Don’t worry about coverage, just write some good tests.”

The programmer smiled, bowed, and left.

…

Questions

- Do you have any other questions?

Thanks!

See you next time! ☺