Contents

1. Vend 2

1.1. Create GitHub vend-kata repository 2

1.2. Create Blank Solution 2

1.3. Add Projects 3

1.4. Accept Counts Kata 3

1.4.1. TDD Flow 4

1.4.1.1. Should\_Detect\_A\_Nickel 4

1.4.1.2. AcceptAccountsKataTest.cs 1 5

1.4.1.3. AcceptAccountsKataTest.cs 2 5

1.5. Select Product Kata 6

1.6. Return Coins Kata 7

1.7. Sold Out Kata 7

1.8. Exact Change Only Kata 7

# Vend

## Create GitHub vend-kata repository

1. In Chrome Browser, navigate to https://github.com/brotherbill, my Github account.

2. Click on Repositories (7)

3. Click on New Button, near top right

4. Fill in Repository name as: vend-kata

Check [x] Initialize this repository with a README

5. 

6. Click: Create repository button

Resulting URL is: https://github.com/brotherbill/vend-kata

## Create Blank Solution

Open Visual Studio 2017, but don't create or open and Solutions or Projects yet.

**Clone from Github**

a. Team Explorer | Connect | Local Git Repositories

Click Clone

Textbox 1: https://github.com/brotherbill/vend-kata

Textbox 2: F:\dev\tutorials\VendKata

[x] Recursively Clone Submodules

Click Clone

**Add Blank Solution VendKata**

Name: VendKata

Location: F:\dev\tutorials

Solution: Create new solution

Solution name: VendKata

[x] Create directory for solution

[ ] Create new Git repository

Click OK.

**Commit local chnages**

Team Explorer | Changes

Commit Message: Add Blank Solution VendKata.

Click Commit All

**Push changes to Github**

Team Explorer | Sync

In Outgoing Commits (1)

Click Push

## Add Projects

Add New Project | Console App (.NET Core)

Name: VendApp

Add New Project | Test | xUnit Test Project (.NET Core)

Name VendAppTest

Add Reference VendApp to VendAppTest

Commit changes locally

Team Explorer | Changes

Commit Message: Added projects

Click Commit All

To push to Github,

Team Explorer | Sync | Outgoing Commits (1)

Click Push

## Accept Counts Kata

We will accept valid coins (5, 10, 25) and reject other coins (1, 50, 100, etc.)

Coins must be American, not Canadian, or other foreign country.

A real mint coin has composition, such as Cupro-Nickel, Weight, Diameter, Thickness, Edge and No. of Reeds

Our fictitious vending machine uses two factors to determine coin validity, ignoring Composition, Thickness and Edge.

A real machine would likely examine those factors.

Our ficticious vending machine will ignore malicious activity, such as tilting machine, putting bubble gum in coin slot,

hitting with a sledge hammer, etc.A real machine might sound an alarm.

A mint nickel

1. weighs 5.000g  
 2. diameter 21.21 mm

However coins are circulated, get dirt on them, and gradually lose material due to scratched, scuffs, etc.  
So our machine uses this grid to detect whether a coin is valid

Nickel Dime Quarter

--------------- ----------------- -----------------------

Weight Min 4.990g 2,250g 5.660g  
Weight Max 5.020g 2.280g 5.680g

Diameter Min 21.10mm 17.70mm 24,10mm  
Diameter Max 21.30mm 18.00mm 25.40mm

The vending machine has sensors for coin weight and coin diameter  
To be valid, a nickel, dime or quarter coin must:

1. be within a low weight and a high weight, including endpoints.

2. have diameter between a minimum and maximum diameter, including endpoints

For example,

Happy Path: dime: weight = 2,26178 diameter = 17.7531  
Happy Path: other: weight = 4.990g diameter = 18.00mm (slug: weight of nickel, diameter of quarter)

Happy Path: other: weight = 10g, diameter = 10mm (slug)

Test cases:  
 1. Coin detected

a. Test each of five cases for each valid coin:  
 less than Min, Min, between Min and Max, Max, greater than max. (between excludes min and max)

b. For each valid coin, must pass both weight test and diameter test.

### TDD Flow

1. Write first case, without writing any code.

[Should\_Detect\_A\_Nickel](#Should_Detect_Nickel)

2. Create Test case [Should\_Detect\_A\_Nickel](#Should_Detect_Nickel)

Fails to compile, due to actual function not written yet.

3. Next step is to make Should\_Detect\_Nickel pass.

4. Write CoinDetector.CoinValue, returning hardcoded 5 to make test pass

Whoopie, we have our first passing test.

Next test with [AcceptAccountsKataTest.cs 2](#CoinDetector_2_cs) is to detect a dime.   
 The nickel test still passes, but the dime test fails, as the result is always a nickel.  
 This one will require real code.

3. Create StandardCoinMetrics class

Doing Test After.

Do commit: Add Coin Detection

4. Next test it. Will test some cases, not every possible case.

a. Returning to test Should\_Detect\_A\_Dime(). Success.

b. Test Should\_Detect\_A\_Quarter(). Success.

c. Test Should\_Detect\_A\_Slug()

#### Should\_Detect\_A\_Nickel

This was written, but will not compile, as CoinDetector.CoinValue has not been written yet.

using System;

using Xunit;

namespace Test

{

/\* weight is in grams

\* diameter is in millimeters

\*/

public class CoinTests

{

[Fact]

public void Should\_Detect\_A\_Nickel()

{

// Arrange

float weight = 5.02f;

float diameter = 21.1f;

// Act

int actualCoin = CoinDetector.CoinValue(weight, diameter);

// Assert

Assert.Equal(5, actualCoin);

}

}

}

#### AcceptAccountsKataTest.cs 1

using VendApp;

using Xunit;

namespace VendAppTest

{

public class UnitTest1

{

[Fact]

public void Should\_Detect\_A\_Nickel()

{

// Arrange

float weight = 5.02f;

float diameter = 21.1f;

// Act

int actualCoin = CoinDetector.CoinValue(weight, diameter);

// Assert

Assert.Equal(5, actualCoin);

}

}

}

#### AcceptAccountsKataTest.cs 2

using VendApp;

using Xunit;

namespace VendAppTest

{

public class AcceptAccountsKataTest

{

[Fact]

public void Should\_Detect\_A\_Nickel()

{

// Arrange

float weight = 5.02f;

float diameter = 21.1f;

// Act

int actualCoin = CoinDetector.CoinValue(weight, diameter);

// Assert

Assert.Equal(5, actualCoin);

}

[Fact]

public void Should\_Detect\_A\_Dime()

{

// Arrange

float weight = 2.25f;

float diameter = 18.0f;

// Act

int actualCoin = CoinDetector.CoinValue(weight, diameter);

// Assert

Assert.Equal(10, actualCoin);

}

}

}

## Select Product Kata

There are three products

$1.00 cola

$0.50 chips

$0.65 candy

Case 1. Buy chips  
 User first select product, then enters coins to pay for it.  
 At Start, Display will display "SELECT PRODUCT"

After selecting product,

Display will toggle "PRICE <balance due>" for two seconds,

then "PAID <amount paid>" for two seconds,

then "INSERT COIN" for two seconds.

Once paid for, Display will display "THANK YOU" for two seconds, then display "SELECT PRODUCT". Machine keeps the change.

Any coins added while SELECT PRODUCT is displayed are kept by vending machine.  
 Any unrecognized coins are also kept by vending machine.

After selecting a product and possibly adding coins, if the user pushes button for another product,

If paid for, goes to step (e) and dispenses chips, thanks, and resets.

If not paid for, then Display toggles every two seconds with "PRICE $#.##", then "PAID $#.##", then "INSERT COIN"  
 The coins already entered are used to pay for new product.

a. Display starts with "SELECT PRODUCT".   
 SelectedProduct is null. Price is 0, AmountPaid is 0.  
 BalanceDue = Price - AmountPaid = 0.

b. User presses Chips button.   
 Sets Start Time with mock.

Display toggles every two seconds with "PRICE $0.50" first, then "INSERT COIN"  
 SelectedProduct is Chips. Price is 0.50, AmountPaid is 0.  
 BalanceDue = Price - AmountPaid = 0.

c. User enter 25 cents. Display toggles every two seconds with "PRICE $0.50" then "PAID $0.25" then "INSERT COIN"

d. User enter 10 cents. Display toggles every two seconds with "PRICE $0.50" then "PAID $0.35" then "INSERT COIN"

e. User enter 25 cents. Machine dispenses Chips  
 Display displays "THANK YOU" for two seconds, then resets to step (a). Machine keeps the change.

## Return Coins Kata

This adds to our vending machine the ability to return money paid before paying for selected product.

Pressing Refund button also resets the machine to No Product Selected.

Case 1: Return coins added before selecting product.

Machine tracks coins added before selecting product, and returns them when Refund button pressed.

Case 2: Return coins added after selecting product, but before fully paying for it.

Machine keeps coins added before selecting product, but tracks coins added after selecting product.  
 Return those coins when Refund button pressed.

## Sold Out Kata

The machine tracks the loaded count of each product.

After each sale, it decrements in-stock count of that product.

Pressing a button for out-of-stock product displays

"SOLD OUT" for two seconds, then "PAID $#.##", then "SELECT PRODUCT"

If all products are sold out, then displays "SOLD OUT" and refunds money.

This way, user doesn't have to press every button.

## Exact Change Only Kata

The machine will be loaded with change (nickels and dimes) when re-stocking.  
But if enough customers need change, it could run out of nickels or dimes.

The machine will display "EXACT CHANGE ONLY" instead of "INSERT COIN" when out of sufficient change.

To make change, the machine needs at least these possibilities to return 5, 10, 15 or 20 cents.

4 nickels or (1 dime and 2 nickels) or (2 dimes and 1 nickel)

When machine is unable to make exact change, overpaying will result in refund.