CS 535 Object-Oriented Programming & Design
Fall Semester, 2013
Doc 15 Assignment 4 Comments
Oct 31 2013

```
testProductSeparatedBy: aCharacter
	self assert:  ('2-3-4' productSeparatedBy: $-) = 24.

"I don't know why the test doesn't work, it works in the work space."
```

```
testProductSeparatedBy
	self assert:  ('2-3-4' productSeparatedBy: $-) = 24.
```

Test methods can not have arguments

```
deposit: amount
	"will perform a deposit into the account and increment the balance"

	[balance := balance + amount asNumber] on: Exception
		do:
			[:exception |
			Transcript
				show: exception description;
				cr]
```

```
withdraw: amount

	| writeToFile |

	[(amount isKindOf: Number)
		ifTrue:
			[(amount negative)
				ifFalse:
					[(amount > balance)
						ifFalse:
							[self balance: self balance - amount.
							writeToFile := file appendStream.
							writeToFile
								nextPutAll: 'Withdraw';
								tab;
								nextPutAll: amount printString;
								cr;
								close.
							^balance asFloat]
						ifTrue: [self error: 'transaction not possible']]
			ifTrue: [self error: 'negative withdrawal not allowed']]
			ifFalse: [self error: 'enter valid amount']]
on: Error
do:
	[:ex |
	Transcript
		show: 'in handler, terminated successfully';
		cr.
	^'invalid']
```

```
deposit: anAmount

        total := total + anAmount.
        ^'Balance deposited is = ' , total printString
```

```
readFile
	"read the content of the file and specify that which amount is related to deposit or withdrawal .   readFile: aFilename When we use a specific file.  "

	| name file fileRead content dAmount wAmount |
	name := 'C:\Users\Home\Documents\BankAccountTr.txt'.
	file := name asFilename.
	fileRead := file readStream.
	[fileRead atEnd] whileFalse:
			[content :=fileRead upTo: Character tab.
			content = 'deposit'
				ifTrue:
					[dAmount := fileRead upTo: Character cr.
					self deposit: dAmount]
				ifFalse:
					[content = 'withdrawal'
						ifTrue:
							[wAmount := fileRead upTo: Character cr.
							self withdrawal: wAmount]]].
	fileRead close
```

```
| token |
	token:= OrderedCollection new.
^token addLast: (self upTo: Character cr). "Adds everything in ordered collection upto
carraige return (Character cr) and position shifts to next ine"
```

```
deposit: aAmount
"Amount is deposited only if the value is Positive or zero and has decimal places upto
two"

	(self isPositiveAmount:aAmount) & (self isValidDecimal:aAmount)
"first condition : Amount should be greater than equal to zero"
"second condition: Amount should be upto two decimal points
 Ex: 100.3213 is checked by --	100.3213 multiplied by 100 = 10032.13--	Stored
as int = 10032--	divided by 100 = 100.32-- Compared with original amount
(100.3213)-- If not equal then the amount is invalid because more than two digits after
decimal"
		ifTrue:
			[accountBalance := accountBalance + (aAmount * 100).
			^'Amount Deposited Successfully']
		ifFalse: [Error raiseSignal:'Amount Invalid. Transaction Failed']
```

Stream>>getACollectionOfLinesInTheStream

Stream>>getACollectionOfLines

Stream>>collectionOfLines

Stream>>lines

checkIfAmountIsValid:

Some Solution & Issues

Checking For valid Numbers

Java Literal Numbers	Smalltalk Literal Numbers
29	29
035	035
0x1D	18.0
0x1d	1.58e3
18.	1.58e-3
18.3	158e4
1.8e1	158d2
18.2f	16rFF

```
self
        assert: ('10,020,00,31.1,2e1' sumSeparatedBy: $,) = (10 + 20 + 31.1 + 2e1)
```

Fun With floats

```
sum := 0.
100000 timesRepeat: [ sum := sum + 0.01].
^sum
```

1000.67

```
sum := 0.
1000000 timesRepeat: [ sum := sum + 0.01].
^sum
```

9865.22

Fun With floats

```
sum := 0.
100000 timesRepeat: [ sum := ((sum*100) + (0.01*100))/100].
^sum
```

1000.25

```
sum := 0.
1000000 timesRepeat: [ sum := ((sum*100) + (0.01*100))/100].
^sum
```

9982.31

Don't Use Floats

```
sumInCents := 0.
100000 timesRepeat: [ sumInCents := sumInCents + (0.01*100) asInteger].
^sumInCents
```

```
sum := 0s2.
100000 timesRepeat: [ sum := sum + (0.01 asFixedPoint:2)].
^sum
```

```
sumSeparatedBy: separatorCharacter
	"Multiplies all numbers within the string separated by the given separator"

	| sum |
	sum :=0.
	(self tokensBasedOn:separatorCharacter) do: [:each | sum := sum + each asNumb
	^sum
```

```
productSeparatedBy: separatorCharacter

	| product selfStream stringPiece |
	product := 1.
	selfStream := self readStream.
	[selfStream atEnd] whileFalse: [
		stringPiece := selfStream upTo: separatorCharacter.
		product := product * stringPiece asNumber.
	].
	^product
```

```
Stream>>nextLine

	^self upTo:Character cr.
```

```
lines

	| linesCollection |
	linesCollection := OrderedCollection new.
	[self atEnd] whileFalse: [linesCollection add: self nextLine].
	^linesCollection
```

```
BankAccountTest>>testFileImport

    | testAccount testFileName |
    testFileName := 'testTransactionFile'.

    [self createTestFile: testFileName contents: self balance100Transactions.
    testAccount := BankAccount withBalance: 0 andName: 'testAccount'.
    testAccount applyTransactionsInFile: testFileName.
    self assert: testAccount balance = 100]
            ensure: [testFileName asFilename delete]
```

```
BankAccountTest>>createTestFile: aFilename contents: aString

	| writeStream |

	[writeStream := aFilename asFilename writeStream.
	writeStream nextPutAll: aString]
		ensure: [writeStream close]
```

```
BankAccountTest>>balance100Transactions

	^'deposit	100
withdrawal	100
deposit	100'
```