```
isNormal

    self customerType = 'Normal' ifTrue: [^true] ifFalse: [^false]
```

```
availableBalanceIn: aDuration

    | timePeriod aLine aDate firstStream secondStream nextLine nextDate |
    firstStream := ReadStream on: availableBalanceStream contents.
    secondStream := ReadStream on: availableBalanceStream contents.
    secondStream nextLine.
    timePeriod := Timestamp now + aDuration.
    aDate := firstStream upTo: Character tab.

    aDate toDate > timePeriod ifTrue: [^0.0 asCurrency].
    firstStream reset.

[secondStream peek ~= nil] whileTrue:
        [aLine := ReadStream on: firstStream nextLine.
        aDate := aLine upTo: Character tab.
        nextLine := ReadStream on: secondStream nextLine.
        nextDate := nextLine upTo: Character tab.
        aDate toDate = timePeriod
            | (aDate toDate < timePeriod & (nextDate toDate > timePeriod))
                ifTrue:
                    [aLine upTo: $$.
                    ^aLine upToEnd asCurrency]].
```

```
balanceIn: aDuration

    | timePeriod aLine aDate firstStream secondStream nextLine nextDate |
    firstStream := ReadStream on: historyStream contents.
    secondStream := ReadStream on: historyStream contents.
    secondStream nextLine.
    timePeriod := Timestamp now + aDuration.
    aDate := firstStream upTo: Character tab.

    aDate toDate > timePeriod ifTrue: [^0.0 asCurrency].
    firstStream reset.

[secondStream peek ~= nil] whileTrue:
        [aLine := ReadStream on: firstStream nextLine.
        aDate := aLine upTo: Character tab.
        nextLine := ReadStream on: secondStream nextLine.
        nextDate := nextLine upTo: Character tab.
        aDate toDate = timePeriod
            | (aDate toDate < timePeriod & (nextDate toDate > timePeriod))
                ifTrue:
                    [aLine upTo: $$.
                    ^aLine upToEnd asCurrency]].
```

```
availableBalanceIn: aDuration

    ^availableBalance valueFromNow: aDuration
```

```
balanceIn: aDuration

    ^balance valueFromNow: aDuration
```

```
transactionFrom: filePath

    | transactionFile fileReader listOfLines |
    transactionFile := filePath asFilename.
    fileReader := transactionFile readStream.
    listOfLines := fileReader lines.
    self makeCollectionOfData: listOfLines.
    self parseFileBasedOnLine.
    fileReader close
```

Names

```
transactionFrom: aFilename
    "This method with read the file given by the user and return the balance"
```

Names

```
orderedDates
    "Sorting all the dates present in the file."

    | aLine aDate |
    orderedDates := SortedCollection new.
    [fileRead peek ~= nil] whileTrue:
        [aLine := ReadStream on: fileRead nextLine.
        aLine upTo: Character tab.
        aDate := aLine upTo: Character tab.
        orderedDates add: aDate toDate].
    ^orderedDates
```

Names

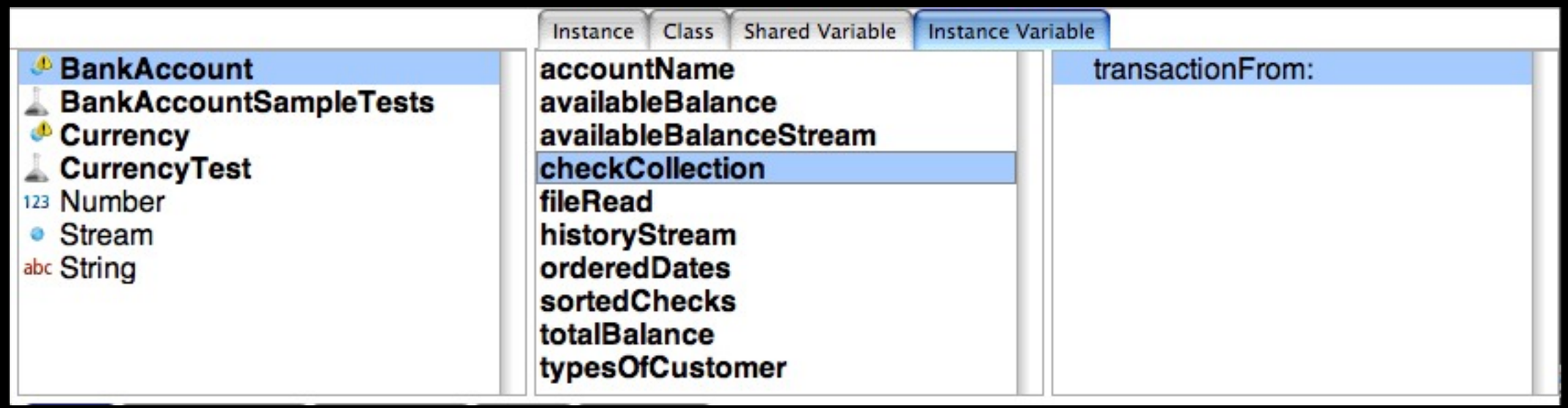
```
orderedDates
    "Sorting all the dates present in the file."

    | transactionStream transactionDateString |
    orderedDates := SortedCollection new.
    [fileRead peek ~= nil] whileTrue:
            [transactionStream := ReadStream on: fileRead nextLine.
            transactionStream upTo: Character tab.
            transactionDateString := transactionStream upTo: Character tab.
            orderedDates add: transactionDateString toDate].
    ^orderedDates
```

```
transactionFrom: aFilename

    |  aString type anAmount aDuration transactionID aDate oldDate tempAmount |
    sortedChecks := SortedCollection new.
    fileRead := file asFilename readStream.
    tempAmount := 0.0 asCurrency.
    self orderedDates.

    1 to: orderedDates size
      do:
        [:each |
        aDate := orderedDates at: each.
        fileRead reset.
```


Instance
Class
Shared Variable
Instance Variable
BankAccount
BankAccountSampleTests
Currency
CurrencyTest
Number
Stream
String
accountName
availableBalance
availableBalanceStream
checkCollection
fileRead
historyStream
orderedDates
sortedChecks
totalBalance
typesOfCustomer
transactionFrom:

```
availableBalanceIn: aDuration
    "returns the current available balance in the account in aDuration amount of time from now"

    | timestamp amount |
    timestamp := Timestamp now + aDuration.
    availableBalanceInTimestamp do:
            [:item |
            | timestampElement |
            timestampElement := Timestamp
                        readFromString: ((item tokensBasedOn: Character tab) at: 1).
            amount := (item tokensBasedOn: Character tab) at: 2.
            timestamp < timestampElement
                ifTrue: [^(amount copyFrom: 2 to: amount size) asCurrency]].
    ^(amount copyFrom: 2 to: amount size) asCurrency
```