COMP1003 - CW03

Team name: Segfault

Test plans:

ID	Class	Method	Test	Input	Expected Output	Pass / Fail	Notes
1.1.1	Transaction	Default constructor	Ensure correct category is set for unknown transaction	*nothing*	get method for transaction category returns 0		No parameters
1.1.2			Ensure correct name is set for unknown transaction	*nothing*	get method for name returns "[Pending transaction]"		
1.1.3			Ensure correct value is set for	nothing	get method for transaction value returns **null**		
1.1.4			Ensure correct time is set for unknown transaction	nothing	get method for transaction time returns time of creation		
1.2.1		Main Constructor	Checks for valid input (String,BigDecimal, Int)	"hello", 35.99, 4	Returns Transaction Time (Type: Date)		
1.2.2			Checks that Transaction name is not too long	"abcdefghijklmnopqrstuvwxyz", 2.0, 3	Throws exception due to invalid Transaction name (above 25 characters)		
1.2.3			Checks that Transaction name has been entered	"", 2.0, 3	Throws exception due to invalid Transaction name (no name entered)		
1.2.4			Checks that Transaction value is not less than 0	"hello", -1, 3.0	Throws exception due to negative value		
1.2.5			Checks that Transaction value is greater than 0	"hello", 0, 3.0	Throws exception due to zero value entered		
1.2.6			Checks that Transaction Category is not a String	"hello", 3, "4"	Throws exception due to incorrect type - should be int		
1.2.7			Checks that Transaction Category is not a float	"hello", 3, 4.0	Throws exception due to incorrect type - should be int		
1.2.8			Checks that Transaction Name can be in unicode	"σφαλμα", 3.0, 3	Returns Transaction Time (Type: Date)		
1.2.9			Check for very large Transaction values entered	2543874350845237052	Exception thrown if number is out of range		
1.3.1		transactionName()	Ensure transaction name handles normal strings	Object with "test123" as name	"test123"		
1.3.2			Ensure transaction name handles unicode	Object with "σφαλμα" as name	"σφαλμα"		
1.4.1		transactionValue()	Ensure that the value can handle very small numbers	Object with 0.01 as value	0.01		
1.4.2			Check that NULL is handled properly	Object with NULL as value	NULL		
1.4.3			Ensure that the value can handle huge numbers	Object with 1000000000 as value	1000000000		
1.4.4			Check that the value can handle normal numbers	Object with 100 as value	100		
1.5.1		transactionCategory()	Check that the value can handle zero	Object with 0 as value	0		
1.5.2			Check that the value can handle negative numbers	Object with -10 as value	-10		
1.5.3			Check that the value can handle positive numbes	Object with 10 as value	10		
1.6.1		transactionTime()	Ensure the date is handled before EPOCH	Object with date as 18:00 31st October 1958	18:00 31st October 1958		
1.6.2			Ensure the date is handled after EPOCH	Object with date as 13:44 8th November 2014	13:44 8th November 2014		
1.6.3			Ensure leap years are handled properly	Object with date as 20:00 29th February 2000	20:00 29th February 2000		

ID	Class	Method	Test	Input	Expected Output	Pass / Fail	Notes
1.6.4			Check for the 2038 problem (EPOCH overflow)	Object with date as 23:59 31st December 2999	23:59 31st December 2999		
1.7.1		setTransactionName()	Checks for valid string parameter	"hello"	Sets Transaction name variable to "hello"		
1.7.2			Ensures integers are not accepted	3	Throws exception due to invalid type		
1.7.3			Ensures floats are not accepted	4.3	Throws exception due to invalid type		
1.7.4			Checks that method will truncate long strings	"abcdefghijklmnopqrstuvwxyz"	Sets Transaction name to "abcdefghijklmnopqrstuvwxy", having truncated the string to 25 characters		
1.7.5			Checks that Transaction Name hasn't already been set	"hello2" with (Transaction name = "hello")	If Transaction name is already set to someting other than "[Pending transaction]", exception will be thrown		
1.7.6			Checks that reserved name is not used	"[Pending transaction]"	Exception thrown due to reserved word		
1.8.1		setTransactionValue()	Checks for a valid parameter of type BigDecimal	3.99	Transaction value is set to 3.99		
1.8.2			Checks that negative parameter values are not allowed	-4.3	Exception thrown due to negative value		
1.8.3			Checks that parameter value is greater than zero	0	Exception thrown due to zero value		
1.8.4			Checks that Transaction value has not already been set	2 with (Transaction value = 3)	If Transaction name is already set, then exception will be thrown		
1.8.5			Check for very large Transaction values entered	2543874350845237052	Exception thrown if number is out of range		
1.9.1		setTransactionCategory()	Checks for valid Transaction category parameter value entered	7	Sets Transaction category to 7		
1.9.2			Checks that value entered corresponds to an avaliable category (non-negative)	-1	Throws exception if not included in avaliable categories		
1.9.3			Checks that value entered corresponds to an avaliable category (non-zero)	0	Throws exception if not included in avaliable categories		
1.9.4			Checks that value entered corresponds to an avaliable category (not greater than number of available categories)	1.00E+101	Throws exception if not included in avaliable categories		
1.10.1		isComplete()	Checks that the current transaction as a name and value [Name & Value]	"Good" 10	TRUE		
1.10.2				"Good" NULL	FALSE		
1.10.3				NULL 10	FALSE		
1.10.4				NULL NULL	FALSE		
1.11.1		toString()	Checks for a clear representation (valid transaction date, name and value)	18:00 31st October 1958, "S	Converts class to a String		
1.11.2			Checks for all 3 parameters entered	18:00 31st October 1958, "S	Throws exception due to missing transaction value parameter		
1.11.3				NULL, "Spending", 39.99	Throws exception due to missing transaction date parameter		
1.11.4				18:00 31st October 1958, N	Throws exception due to missing transaction name parameter		
1.11.5				18:00 31st October 1958, N	Throws exception due to 2 missing parameters		

ID	Class	Method	Test	Input	Expected Output	Pass / Fail	Notes
1.11.6				NULL, "Spending", NULL	Throws exception due to 2 missing parameters		
1.11.7				NULL, NULL, 39.99	Throws exception due to 2 missing parameters		

ID	Class	Method	Test	Input	Expected Output	Pass / Fail	Notes
2.1.1	Category	WhatNestCategory()	Check that both spend and budget are set to 0	nothing	Category object with budget and spend to 0		
2.1.2			Check that each name is unique by creating multiple classes (1000+) and make sure they all have unique names	nothing	1000+ Category objects should be created all with unique names		
2.2.1		WhatNestCategory(String newTitle)	Check that meets specification	"Example"	Category object with name "Example"		
2.2.2			Create 2 classes with same name make sure it gives error	"Example"	Error - dublicate name		
2.2.3			Make sure that class has spend and budget to 0 and not an arbitrary value	"Example"	Category with name "Example" and spend and budget values set to 0.		
2.2.4			Check Unicode characters	"σφαλμα"	Category with name "σφαλμα" and budget and spend set to 0		
			Check that default constructor handles long names	"AVeryVeryVeryLongLongName"	Should give error as name length is 15 characters		
			Check that it handles empty names	""	Should give error		
2.3.1		CategoryName()	Create Category object with name "Example" and make sure it returns the correct category name	"Example"	Call to the getCategoryName() function of the Category object create should return "Example" as a string		
2.4.1		CategoryBudget()	Create Category object with name "Example" and budget to 12 and make sure that getCategoryBudget returns 12	"Example" 12	Call to the getCategoryBudget() funciton of the Category object created should return 12 as a BigDecimal		
2.5.1		CategorySpend()	Create Category object with name "Example" and spend to 300 and make sure that getCategorySpend returns 300	"Example" 300	Call to the getCategorySpend() funciton of the Category object created should return 300 as a BigDecimal		
2.6.1		setCategoryName(String newName)	Make sure that the name changes when you call function with a string	"Example" "AnotherName"	Should change the CategoryName from "Example" to "AnotherName"		
2.6.2			Check that it handles long names	"AVeryVeryLongLongName"	Should give error as name length is 15 characters		
2.6.3			Check that it handles empty names	""	Should give error		
2.6.4			Check that meets specification	"Example"	Category object with name "Example"		
2.6.5			Create 2 classes with same name make sure it gives error	"Example"	Error - dublicate name		
2.6.6			Make sure that class has spend and budget to 0 and not an arbitrary value	"Example"	Category with name "Example" and spend and budget values set to 0.		
2.6.7			Check Unicode characters	"σφαλμα"	Category with name "σφαλμα" and budget and spend set to 0		
2.7.1		setCategoryBudget(BigDecimal newValue)	Test that it accepts floats	2500.01	Sets category budget to 2500.01		
2.7.2			Test that it does not accept negatives	-2.5	Error negative input not allowed		
			Check very large numbers	999999.99999999900	Sets category budget to 999999.999999999		
2.8.1		addExpense(BigDecimal valueToAdd)	Test that it accepts BigDecimal type	1.34589349	Check that spend increased by 1.34589349		
2.8.2			Test with integer	1	total spend for category is (x+1)		
2.8.3			Test with negative numbers	-10000.12	Error - cannot have negative expense		
2.8.4			Test with large positive number	99999999999	total spend should be set to (x+99999999999)		

ID	Class	Method	Test	Input	Expected Output	Pass / Fail	Notes
2.8.5			Test with expense larger than budget. Category created with name "Example", Budget 400 and spend 200, when given an expense bigger than the budget (600) sets the budget to negative number.	"Example" 400 200 600	-200		
2.9.1		removeExpense(BigDecimal valueToRemove)	Test that it accepts BigDecimal type	1.34589349	Total spend for the category should be (x-1.34589349)		
2.9.2			Test with 0. Should give error. Expenses have a cost.	0	Error - Expense has cost 0		
2.9.3			Test with negative numbers	-10000.12	Error - cannot have negative expense. Function is already subtracting and subtracting a negative number does not make sense.		
2.9.4			Test with large positive number	99999999999	total spend should be set to (x-99999999999)		
2.9.5			Test with expense larger than spend. Category created with name "Example", Budget 400 and spend 200, when given an expense bigger than the spend (400) gives an error	"Example" 400 200 400	Error - Spend smaller than expense		
2.10.1		resetBudgetSpend()	Test that it sets Spend in a category to zero.	nothing	Spend should be set to zero after resetSpend is called		
2.11.1		getRemainingBudget()	For Category created with name "Example", Budget 400 and spend 200 check that it returns 400-200 = 200	"Example" 400 200	200		
2.12.1		toString() Override	For Category created with name "Example", Budget 0 and spend 0. Check everything is 0	"Example"	"[Example] (£0) - Est. £0 (£0)"		
2.12.2			For Category created with name "Example", Budget 400 and spend 200. Check base Case	"Example" 400 200	"[Example] (£400) - Est. £200 (£200)"		
2.12.3			For Category created with name "Example", Budget 400 and spend 500. Check overspent	"Example" 400 500	"[Example] (£400) - Est. £500 (£100 overspent)"		

ID	Class	Method	Test	Input	Expected Output	Pass / Fail	Notes
3.1.1	Main	Main Method	Test input that isn't allowed - something not offered in menu	"potet"	should re-offer the options list (menu)		
3.1.2			Test overview overview with no categories existing	none	Will not display anything, no categories exist.		
3.1.3			Test overview with categories existing	none	Displays an overview of categories.		
3.1.4			Test list tranactions option	none	Displays transaction history (if it exists)		
3.1.5			Test change transaction category	1	Displays a list of options that you can change the current transaction to.		
3.1.6			Test add transaction function	"Transaction1" 25	Adds a transaction called Transaction1 of value £25 with unknown category. Adds this to the arraylist of WhatNestTransactions		
3.1.7			Test add category function	"Category1" 2500	Should create a category Category1 with a budget of £2500. Adds it to the arraylist of WhatNestCategories.		
3.1.8			Test Exiting the system	Exit	Should print out "Goodbye!" to the user and close the program.		
3.2.1		listTransactions()	Test with no transaction history	none	none		
3.2.2			Test with transaction history	none	should print out the transactions as: "Store (category) - £xx.xx" in order.		
3.3.1		categoryOverview()	Test with no categories	none	none		
3.3.2			Test with categories but no budgets/spending	none	1. CategoryName (budget: £0.00) -£0.00 (£0.00 remaining) For each of the categories that exist, initial number increases with each category.		
3.3.3			Test with no overspending	none	1. CategoryName (budget: £20.00) -£10.00 (£10.00 remaining) For each of the categories that exist, initial number increases with each category. Set budget to 20, spend to 10 before calling.		
3.3.4			Test with overspending	none	1. CategoryName (budget: £10.00) -£20.00 (£10.00 overspent) For each of the categories that exist, initial number increases with each category. Set budget to 10 and spend to 20 before calling.		
3.4.1		listTransactionCategory()	Test with valid category ID	1	prints out all transactions from the category with ID 1 in "Store (category) - £xx.xx" format in order for the user.		
3.4.2			test with non-valid (not existing) category id	999	Error, category does not exist.		
3.5.1		changeTransactionCategory()	Test with valid category ID	1	Gives user options of other valid category IDs to change it to. It displays the categories as a list		
3.5.2			test with non-valid (not existing) category id	999	Error, current category does not exist.		

ID	Class	Method	Test	Input	Expected Output	Pass / Fail	Notes
3.6.1		addTransaction()	Test with valid transaction name, value and category	"Transaction1" 3.99 3	Adds transaction to ArrayList of transactions. Gives confirmation to the user in the form: "[transaction name] (£value) was added to [category name]"		
3.6.2			Test with valid transaction name and value (category excluded - optional)	"Transaction1" 3.99	Adds transaction to ArrayList of transactions. Gives confirmation to the user in the form: "[transaction name] (£value) was added to Unknown"		
3.6.3			Test with invalid transaction name	"", 3.99, 3	Gives user an error message and prompts them to enter a valid name. Nothing added to ArrayList of transactions		
3.6.4			Test with invalid transaction value	"Transaction1", NULL, 3	Gives user an error message and prompts them to enter a valid value. Nothing added to ArrayList of transactions		
3.7.1		addCategory()	Test with valid name and budget for category	"CategoryName", 350	Adds category to ArrayList of categories. User receives confirmation that process was successful, stating which category and its budget		
3.7.2			Test with invalid name	"", 350	Gives user an error message and prompts them to enter a valid name. Nothing added to ArrayList of categories		
3.7.3			Test with invalid budget	"Category", NULL	Gives user an error message and prompts them to enter a valid budget value. Nothing added to ArrayList of categories		

Bug reports

ID	Class	Method name	Problem description	Proposed fix	Priority
1	WhatNestTransaction	WhatNestTransaction()	The transaction time is not set	Set transactionTime to new Date()	HIGH
2		WhatNestTransaction(String,BigDecimal,int)	The transaction value is not checked to be above 0	Use if statement and inequality	MEDIUM
3			The transaction category is not checked to be a valid index	Use if statement and inequality, checking it is a valid element in WhatNestApp.UserTransactions	MEDUM
4		transactionName()	If transaction name is NULL, should return "Pending Transaction"	Use if statement to check for NULL	MEDIUM
5		setTransactionName()	Checking wrong parameter	Change tName to transactionName	MEDIUM
6		setTransactionCategory	Zero should also be valid	tCat > 0 should be tcat >= 0	MEDIUM
7			tCat might be above the number of categories	Check against count of categories in WhatsNestApp	Low
8		?? setTransactionTime()	Time cannot be set, this method is no valid	Delete this entire method	HIGH
9		isComplete()	Function is not present in the program	Add isComplete() function	HIGH
10		itoString	No check for NULL is included	Use if statement to check for NULL	MEDIUM
11			Date is not included in the returned string	Add date to the return value	MEDIUM
12	WhatNestCategory	WhatNestCategory()	Category name should always be unique	Create a random name generator (possibly using random numbers and a hashing function)	HIGH
13		setCategoryName(String newName)	Need to check that name is less than 15 characters long	if statement with check	MEDIUM
14		addExpense(BigDecimal valueToAdd)	Need to check that valueToAdd is positive	if statement with check using compareTo(...)	HIGH
15		removeExpense(BigDecimal valueToRemove)	Need to check that value is positive and smaller than spend	if statement with check using compareTo(...) to check for both positive value and smaller than spend	HIGH
16		toString()	Missing case where getRemainingBudget() is negative. Need to print overspent	if/else statement with the getRemainingBudget as a condition	MEDIUM
17	WhatNestApp	main(String[] args)	Line 68, should check for lower case x instead of X	Change string checked	HIGH
18		AddTransaction(Scanner in)	Output isn't according to spec	last line change output to this format: “[transactionname] (£value)was addedto[categoryname]	MEDIUM
19		AddCategory(Scanner in)	User confirmation ouput is invalid	should give category name and it's budget	MEDIUM