Module Title

Principles of Programming

Assessment Weightage & Type

50% of Coursework & Regular Year

2025

Student Name: NIRVIK K.C.

UWE ID: 25024649

Assignment Submission Date: January 18, 2026

Submission details

Module title and code: UFCFHS-30-1 Principles of Programming Individual Assignment

Assessment type: Individual Assignment – Software Development Portfolio

Assessment title: Application for Car Parts and Accessories Shop

Assessment weighting: 50% of total module mark

Step 1

Imagine a Car Parts and Accessories shop, which requires a software system to keep track of stock items and prices. The shop will sell different kinds of stock items. However, to start with, you have been tasked with designing and implementing a class called StockItem with the following properties.

▪ An instance (object) of the StockItem class represents a particular item which the shop sells, with a string representing fixed stock code, an integer representing variable quantity in stock and a double representing variable price of the stocked item. All these variables should be declared as private variables. The StockItem class also contains a class variable (shared by all instances) of type string representing stock category which you can initialise as 'Car accessories'.

▪ A constructor that creates a Stock Item with the specified quantity, price and the stock code.

▪ All the appropriate ‘setters’ and ‘getters’ methods, including a getStockName() method which returns the string "Unknown Stock Name" and a getStockDescription() method which returns the string "Unknown Stock Description".

▪ An increaseStock() method that increases the stock level by the given amount. If the value is less than 1 or the stock exceeds 100, a suitable error message should be printed.

▪ A sellStock() method that attempts to reduce the stock level by the given amount. If it is less than 1, a suitable error message should be printed. If the amount is otherwise less than or equal to the stock level, then the reduction is successful and true is returned. Else there is no effect, but false is returned.

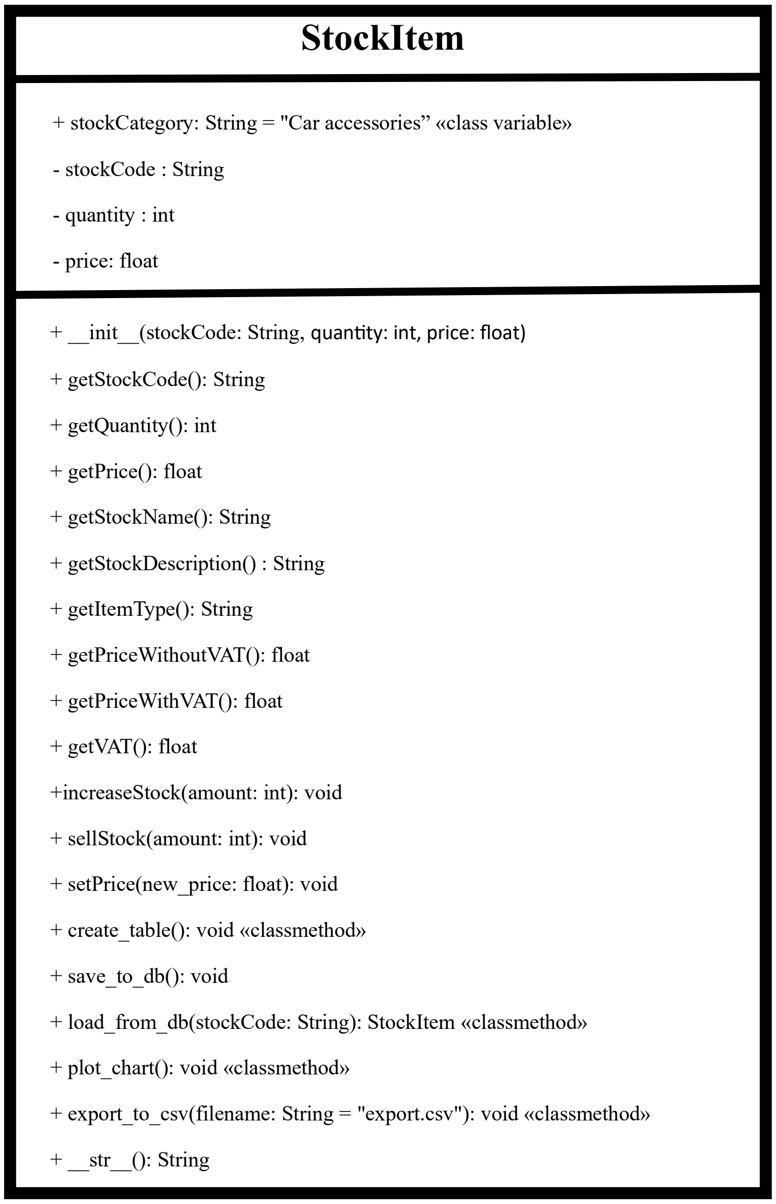
▪ A getVAT() method that returns the standard percentage VAT rate, e.g., you can use 17.5

▪ Appropriate ‘setters’ method for price (without VAT) and ‘getters’ methods for price with and without VAT ▪ A method named __str__ () that returns a string giving the stock code, the stock name, the description, the quantity in stock, the price before VAT and the price after VAT. It must use the appropriate methods above to obtain the stock name, description, quantity and prices.

Task 1.1 Design and draw the corresponding UML class diagram of the StockItem class.

Answer:

UML Diagram for Step 1:



StockItem

+ stockCategory: String = "Car accessories" «class variable»
- stockCode : String
- quantity : int
- price: float

+ __init__(stockCode: String, quantity: int, price: float)
+ getStockCode(): String
+ getQuantity(): int
+ getPrice(): float
+ getStockName(): String
+ getStockDescription() : String
+ getItemType(): String
+ getPriceWithoutVAT(): float
+ getPriceWithVAT(): float
+ getVAT(): float
+increaseStock(amount: int): void
+ sellStock(amount: int): void
+ setPrice(new_price: float): void
+ create_table(): void «classmethod»
+ save_to_db(): void
+ load_from_db(stockCode: String): StockItem «classmethod»
+ plot_chart(): void «classmethod»
+ export_to_csv(filename: String = "export.csv"): void «classmethod»
+ __str__(): String

Note:

stockCategory: default value “Car accessories” (class variable)

stockCode: immutable – cannot be changed after initialization

quantity: modified only by increaseStock() and sellStock() methods

price: must be non-negative, set via setPrice()

increaseStock(): amount constraints – min1, max 100 per operation

sellStock(): amount must be ≤ current quantity

getStockName(): default “Unknown Stock Name”

getStockDescription(): default “Unknown Stock Description”

getItemType(): default “General Car Accessory”

VAT rate: default value is 17.5%

All instance attributes private (-)

All methods public (+)

Exception: stockCategory is public (+) as it is a class variable shared by all instances.

Task 1.2. (Implementation and testing) Implement the above class and test it. You should create some instances of StockItem class, increase stock, sell some stock and change the price, whilst printing out the items in between. Testing is typically a part of the program development – you should use a test strategy to test your program thoroughly. You may look at your practical exercises, identify suitable test cases for the StockItem class, write and document them in the form of a table along with the UML class design file.

Test Cases	Purposes	Expected result	Outcome
Test Case 1	Create StockItem object with valid parameters	Object created successfully, saved to database	Pass and no error
Test Case 2	Test getStockCode() method	Returns 'W101"	Pass - No errors
Test Case 3	Test getQuantity() method	Returns 10	Pass - No errors
Test Case 4	Test getPriceWithoutVAT() method	Returns 99.99	Pass - No errors
Test Case 5	Test getPriceWithVAT() method	Returns 117.49 (with 17.5% VAT)	Pass - No errors

Continue table:

Test Cases	Purposes	Expected result	Outcome
Test Case 6	Test getStockName() method	Returns "Unknown Stock Name"	Pass - No errors
Test Case 7	Test getStockDescription() method	Returns "Unknown Stock Description	Pass - No errors
Test Case 8	Test getItemType() method	Returns "General Car Accessory"	Pass - No errors
Test Case 9	Test increaseStock() with valid amount (10)	Quantity increases from 10 to 20, success message displayed	Pass - No errors
Test Case 10	Test increaseStock() with zero (0)	Error message: "Increased item must be greater than or equal to one"	Pass - No errors
Test Case 11	Test increaseStock() with negative value (-5)	Error message: "Increased item must be greater than or equal to one	Pass - No errors

Continue table:

Test Cases	Purposes	Expected result	Outcome
Test Case 12	Test increaseStock() exceeding maximum (101)	Error message: "Increased item must be less than or equal to 100"	Pass - No errors
Test Case 13	Test increaseStock() with invalid string ("ten")	Error message: "Amount must be a whole number (integer)"	Pass - No errors
Test Case 14	Test sellStock() with valid amount (2)	Quantity decreases to 18, message: "Sold 2 and Remaining: 18"	Pass - No errors
Test Case 15	Test sellStock() with zero (0)	Error message: "Cannot sell less than 1 item"	Pass - No errors
Test Case 16	Test sellStock() exceeding stock (1000)	Error message: "Only 12 items in stock - can't sell 1000"	Pass - No errors

Continue table:

Test Cases	Purposes	Expected result	Outcome
Test Case 17	Test sellStock() triggering low stock alert (≤5)	Alert message: "Restock your items is recommended. Low stock alert!"	Pass - No errors
Test Case 18	Test sellStock() triggering critical alert (≤2)	Alert message: "Stock items are critically low. Only a few units left!"	Pass - No errors
Test Case 19	Test sellStock() reaching zero stock	Alert message: "Items are unavailable. Out of stock!"	Pass - No errors
Test Case 20	Test str() method output	Returns formatted string with all stock information	Pass - No errors

Step 2

The Car Parts and Accessories shop has got plenty of GeoVision Sat Nav navigation system at very competitive prices, which are going to be the first item on sale. You need to design and implement a class NavSys which is a sub-class of StockItem.

▪ You also need to create one new private instance variable of your NavSys class to represent navsys brand.

▪ A parameterised constructor of the NavSys class must call the StockItem's constructor using super to initialise the super class's instance variables.

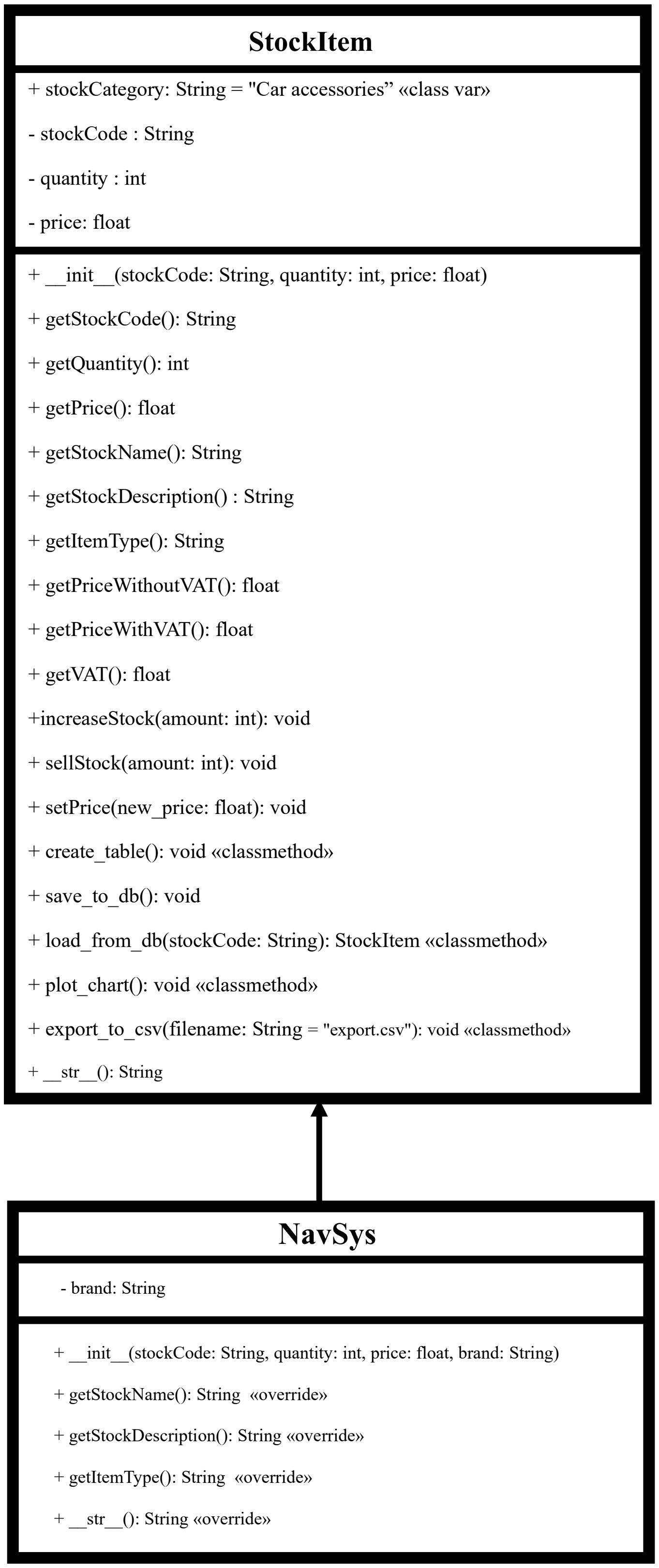
▪ The NavSys class will override the instance methods getStockName() and getStockDescription() with ones that return "Navigation system" and " GeoVision Sat Nav" respectively.

▪ NavSys class will also override the __str__ () method using the concept of super in Python.

Task 2.1 (Revised design). Revise your UML class diagram in Task 1.1, to incorporate the NavSys class and show their relationship using appropriate UML notations.

Answer:

UML Diagram for Step 2:


StockItem
+ stockCategory: String = "Car accessories" «class var»
- stockCode : String
- quantity : int
- price: float
+ __init__(stockCode: String, quantity: int, price: float)
+ getStockCode(): String
+ getQuantity(): int
+ getPrice(): float
+ getStockName(): String
+ getStockDescription() : String
+ getItemType(): String
+ getPriceWithoutVAT(): float
+ getPriceWithVAT(): float
+ getVAT(): float
+increaseStock(amount: int): void
+ sellStock(amount: int): void
+ setPrice(new_price: float): void
+ create_table(): void «classmethod»
+ save_to_db(): void
+ load_from_db(stockCode: String): StockItem «classmethod»
+ plot_chart(): void «classmethod»
+ export_to_csv(filename: String = "export.csv"): void «classmethod»
+ __str__(): String
NavSys
- brand: String
+ __init__(stockCode: String, quantity: int, price: float, brand: String)
+ getStockName(): String «override»
+ getStockDescription(): String «override»
+ getItemType(): String «override»
+ __str__(): String «override»

Notes:

For StockItem class:

stockCategory: default "Car accessories" (class variable, shared by all instances)

stockCode: immutable – cannot be changed after initialization

price: must be non-negative, validated in setPrice()

increaseStock(): amount constraints – min 1, max 100 per operation

sellStock(): amount must be ≤ current quantity

getStockName(): default “Unknown Stock Name”

getStockDescription(): default "Unknown Stock Description"

getItemType(): default "General Car Accessory"

VAT rate: default “value fixed at 17.5%”

For NavSys Class:

All the attributes and methods are inherited from StockItem

Add new private attributes brand (required, no default value)

Overrides methods: getStockName(), getStockDescription, getItemType(), __str__()

getStockName(): default “Navigation system” getStockDescription: default “GeoVision Sat Nav”

getItemType(): default “Navigation System”

__str__(): calls super().__str__ and appends the brand information

All instance attributes private (-)

All methods public (+)

Exception: stockCategory is public (+) as it is a class variable shared by all instances.

Clear picture of UML diagram:

Continue to next page.

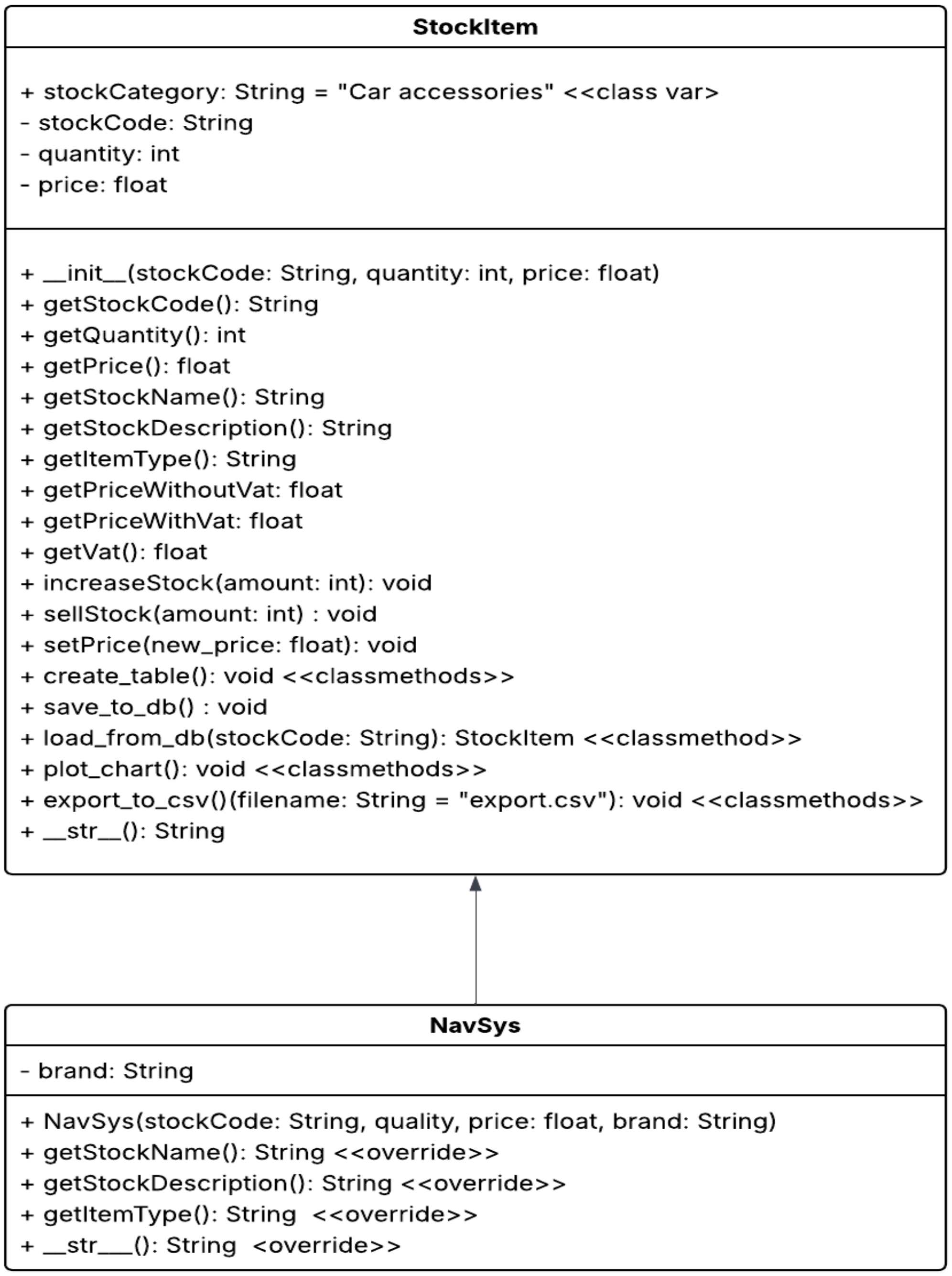

StockItem
+ stockCategory: String = "Car accessories" <<class var>>
- stockCode: String
- quantity: int
- price: float
+ __init__(stockCode: String, quantity: int, price: float)
+ getStockCode(): String
+ getQuantity(): int
+ getPrice(): float
+ getStockName(): String
+ getStockDescription(): String
+ getItemType(): String
+ getPriceWithoutVat: float
+ getPriceWithVat: float
+ getVat(): float
+ increaseStock(amount: int): void
+ sellStock(amount: int) : void
+ setPrice(new_price: float): void
+ create_table(): void <<classmethods>>
+ save_to_db() : void
+ load_from_db(stockCode: String): StockItem <<classmethod>>
+ plot_chart(): void <<classmethods>>
+ export_to_csv()(filename: String = "export.csv"): void <<classmethods>>
+ __str__(): String
NavSys
- brand: String
+ NavSys(stockCode: String, quality, price: float, brand: String)
+ getStockName(): String <<override>>
+ getStockDescription(): String <<override>>
+ getItemType(): String <<override>>
+ __str___(): String <override>>

Figure 1 : Image View of UML Diagram for Application for Car Parts and Accessories Shop (Lucidchart)

Task 2.2 (Revised implementation and testing). Implement the NavSys class and test it by creating an instance of NavSys, adding and then selling some navigation system stock and changing the price, whilst printing out the item in between.

Test Cases	Purposes	Expected result	Outcome
Test Case 21	Create NavSys object with valid parameters including brand	Object created with brand "TomTom" and saved to database	Pass – No Errors
Test Case 22	Test getStockCode() inherited method	Returns "NS401"	Pass – No Errors
Test Case 23	Test getQuantity() inherited method	Returns 10	Pass – No Errors
Test Case 24	Test getPrice() inherited method	Returns 99.99	Pass – No Errors
Test Case 25	Test getPriceWithoutVAT() inherited method	Returns 99.99	Pass – No Errors
Test Case 26	Test getPriceWithVAT() inherited method	Returns 117.49 (with 17.5% VAT)	Pass – No Errors

Continue table:

Test Cases	Purposes	Expected result	Outcome
Test Case 27	Test getStockName() overridden method	Returns "Navigation system" (not "Unknown Stock Name" for NavSys)	Pass – No Errors
Test Case 28	Test getItemType() overridden method	Returns "Navigation system" (not "General Car Accessory")	Pass – No Errors
Test Case 29	Test increaseStock() inherited with validation	Quantity increases to 20 and validation rules apply	Pass – No Errors
Test Case 30	Test str() overridden method with super()	Displays parent information plus "Brand: TomTom"	Pass – No Errors
Test Case 31	Test sellStock() inherited alert system	Low stock alerts work correctly for NavSys	Pass – No Errors

Continue table:

Test Cases	Purposes	Expected result	Outcome
Test Case 32	Test create_table() class method	Database table created successfully and message shown: "Database table ready."	Pass – No Errors
Test Case 33	Test load_from_db() with existing NavSys	Loads NavSys object with brand from the database	Pass – No Errors
Test Case 34	Test export_to_csv() with default filename	Creates "export.csv" file with all the records	Pass – No Errors
Test Case 35	Test plot_chart() visualization	Displays bar chart comparing Navigation vs Other stock levels using Python library	Pass – No Errors