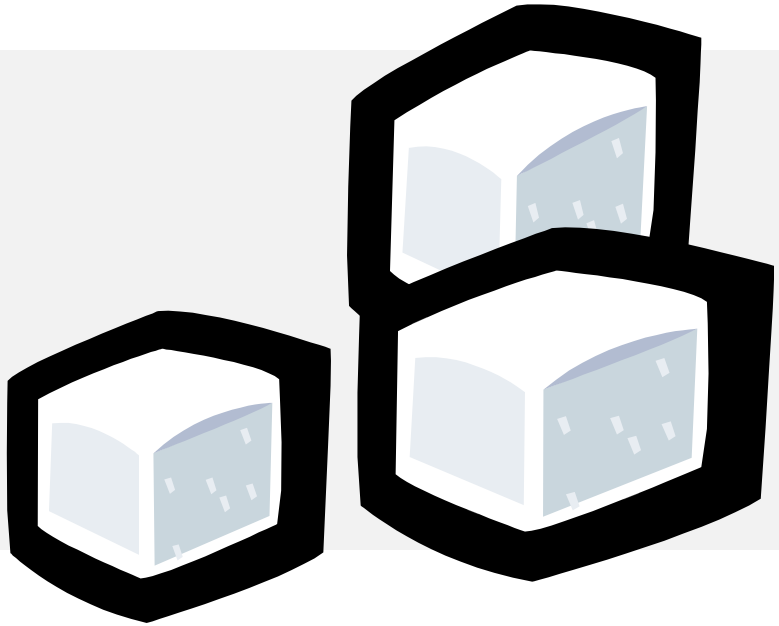
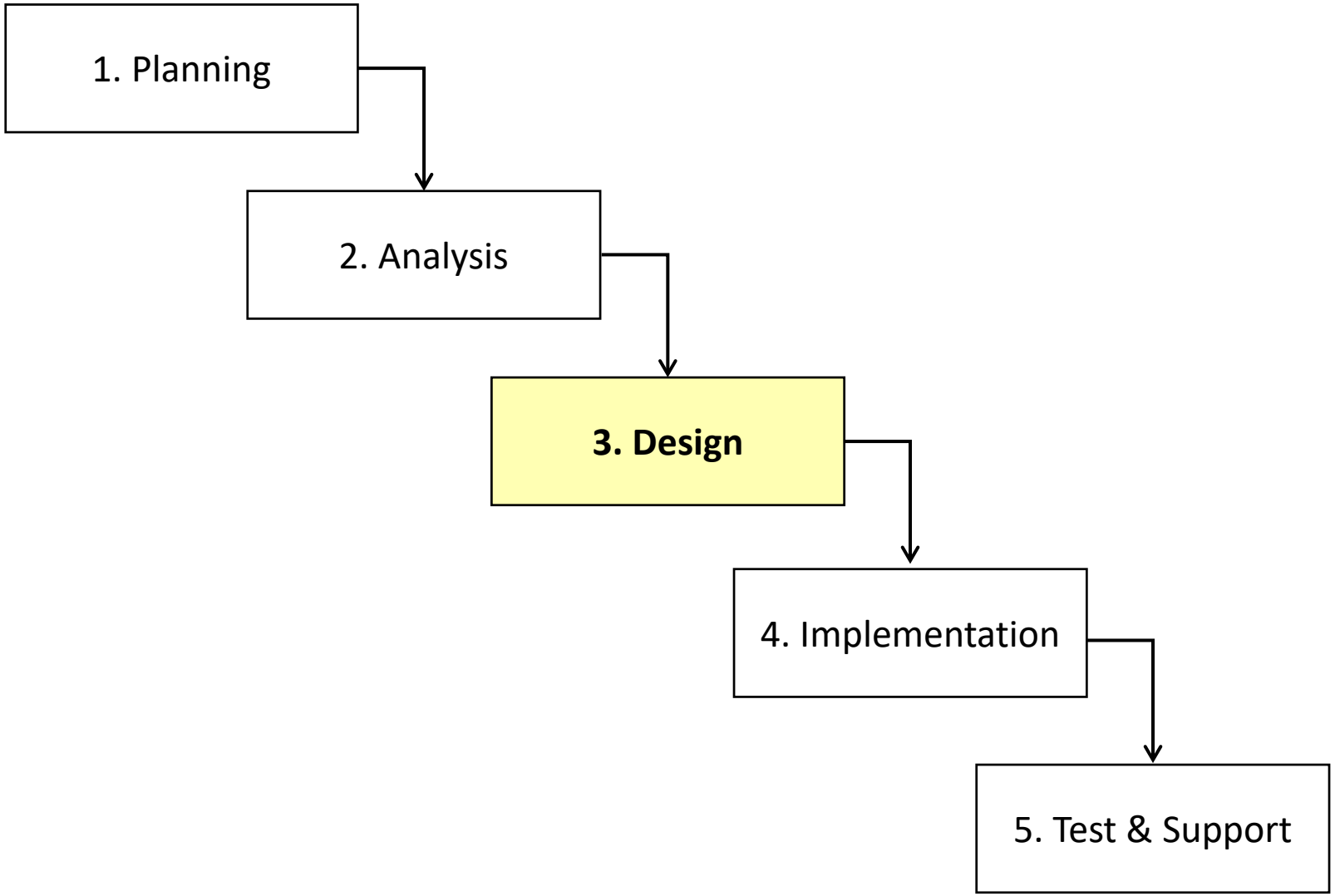


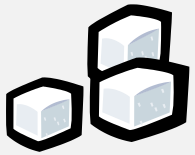
UML Class Diagrams & Class Implementation

Author: Mohammad Amin Kuhail

Reviewed By: Sujith Mathew, Afshan parker, Nishara Nizamuddin

Software Life Cycle





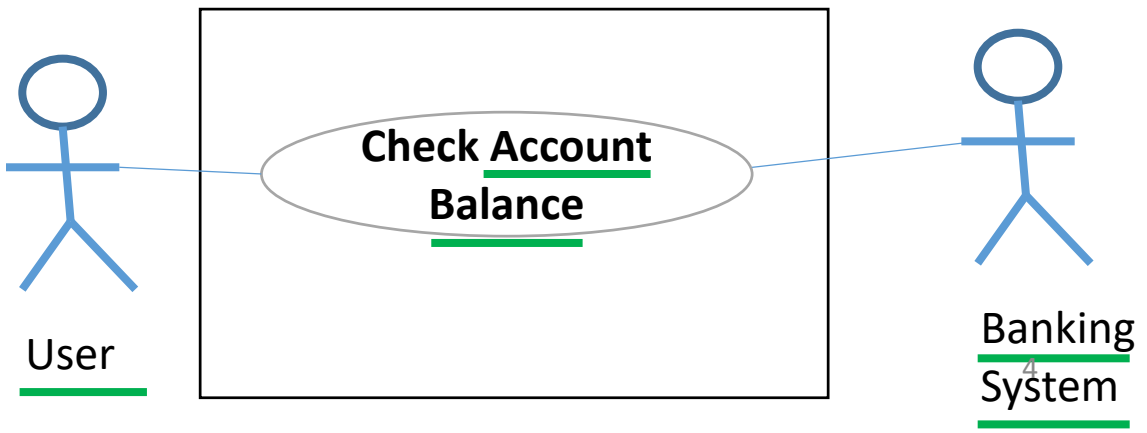
Design Phase

- In the design phase of software development, developers start thinking about how to model the solution.
- The design phase consists of diagraming the concepts of the software that have been specified in the use cases and other problem descriptions.
- UML stands for Unified Modeling Language, a general-purpose language that developers use to visualize the design of the system.
- One diagram that is commonly used in the design phase is a UML class diagram, which shows the main concepts of the system as well as the relationships between them.

Identifying the Classes (1)

- The first step in designing a UML class diagram is identifying the classes.
- To identify a class, we need to read the description in a use case or the problem statement and look for major concepts (nouns).
- For example, in the example below, there are several concepts (nouns): Account, Balance, User, and System (all underlined in green)

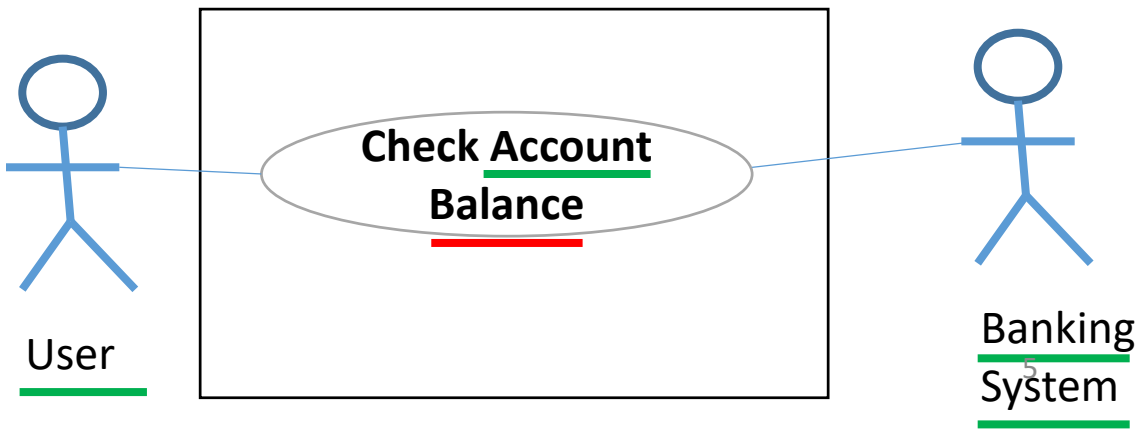
Use Case:	**Check Account Balance**
Main scenario:	
1.	Authenticate the user
2.	The user signals to the system that they want to check their account balance.
3.	The system shows the balance to the user.



Identifying the Classes (2)

- However, classes have to be a substantial concept that can be described by attributes and functions.
- For example, a user qualifies as a class because it may have various attributes such as first name, last name, gender, etc.
- Balance, on the other hand, does not qualify as a class because it is a simple attribute of an account. Balance does not have attributes to describe it.

Use Case:	**Check Account Balance**
Main scenario:	
1.	Authenticate the user
2.	The user signals to the system that they want to check their account balance.
3.	The system shows the balance to the user.



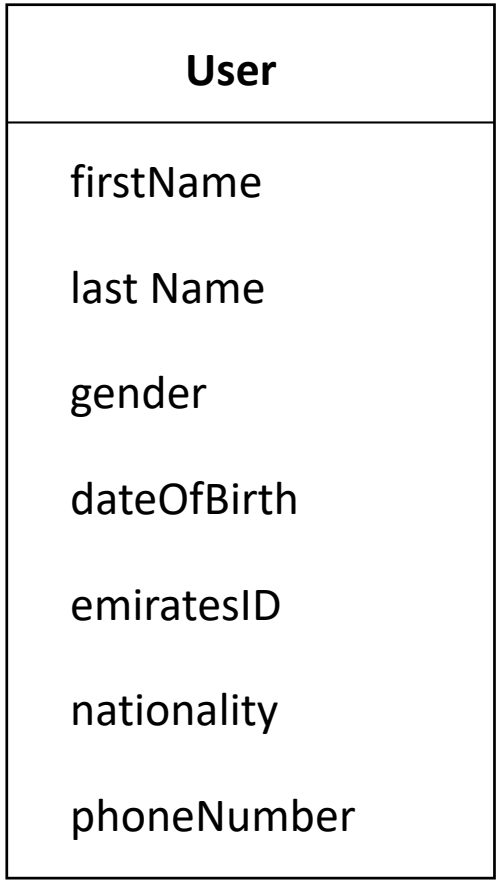
Identifying Attributes of a Class (1)

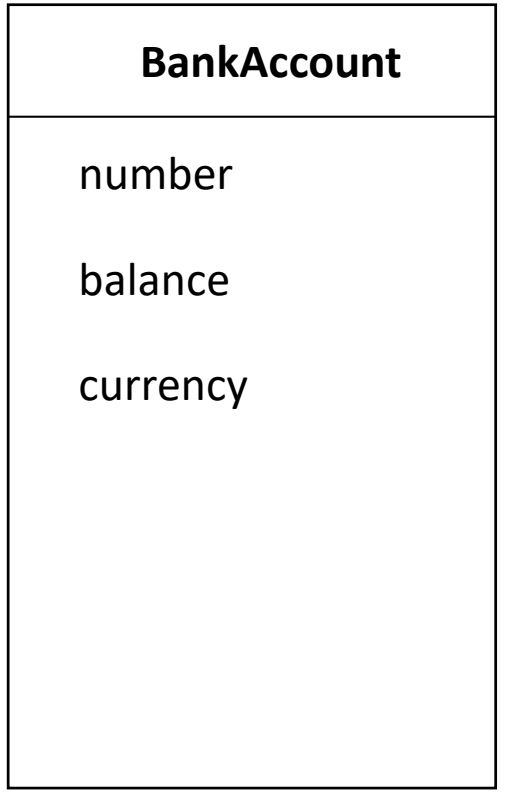
- We read carefully the attributes that describe a certain concept (a class).
- For instance, in the use case “Open Bank Account”, we can see in step 1 that the user provides his/her details (i.e. attributes) such as first name, last name, nationality, date of birth, phone number, gender, and EID.
- In steps 2 and 3, we can see that the bank account has a currency, balance, and number.
- Now we are ready to list the attributes under each class.

Use Case:	**Open Bank Account**
Main scenario:	
1.	The user provide his/her details: first name, last name, nationality, date of birth, phone number, gender, and EID.
2.	The user provides bank account currency and the initial balance.
3.	The system opens the bank account for the user with a specific number.

Identifying Attributes of a Class (2)

- Classes are visually represented using boxes. This notation is defined by a specific language: UML (Unified Modelling language)
- The first rectangle in the box is used for the class name, while the second rectangle is for the attributes.
- Each box represents a concept and the attributes that describe it. The name of the class should be capitalized, while the names of the attributes start with a small letter.
- In UML, When we write an attribute, it is customary to use camel case to distinguish the two words (e.,g. word1Word2)



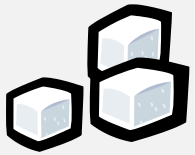


Identifying Types of Attributes (1)

- Next, we identify the types of attributes.
- For the User class, both first name and last name are considered strings as they contain alphabetical letters.
- Gender is a value of two choices (male/female). As such, it is considered an enumerator type.
- The date of birth is a date type.
- EID is an ID that contains digits and letters. As such, it is a String type.
- Like Gender, Nationality is a value of multiple choices (i.e. countries of the world), making it an enumerator type.

User
firstName: String
lastName: String
gender: ENUM
dateOfBirth: Date
emiratedID: String
nationality: ENUM
phoneNumber: String

BankAccount
number: String
balance: Float
currency: ENUM



Identifying Types of Attributes (2)

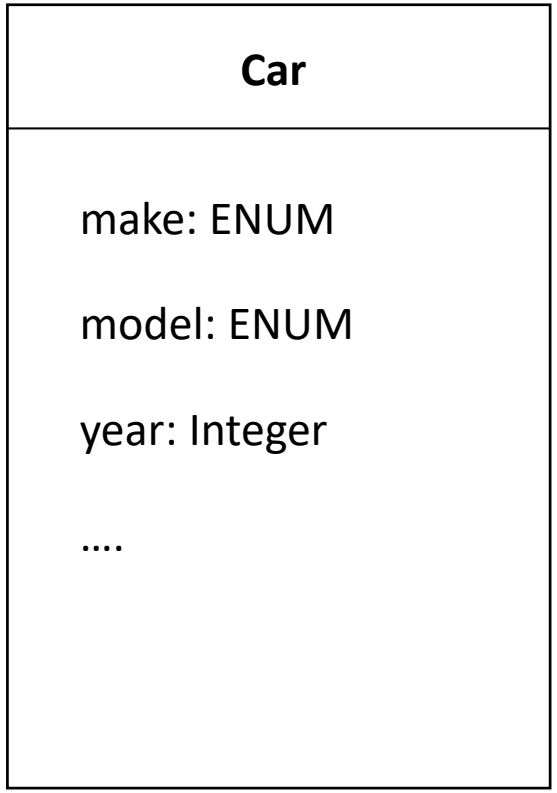
- For Bank Account, the number is a String type as it contains digits, and possibly letters.
- The balance is the amount of money containing the decimal value. As such, Float would be an appropriate type.
- Currency is a value of multiple choices. Hence, it is an enumerator type.

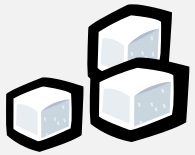
User
firstName: String
lastName: String
gender: ENUM
dateOfBirth: Date
emiratesID: String
nationality: ENUM
phoneNumber: String

BankAccount
number: String
balance: Float
currency: ENUM

Task 1

- Using your experience, in the following figure, identify attributes and types for the class Car, in addition to the ones already identified in the figure.





Identifying Behavior of a Class (1)

- The description below is part of a system analysis report of a banking system.
- We can use it to extract some behavior of the classes that we identified previously.
- In assigning behavior (functions) to classes, we must make sure that each class manages its own information. For instance, the Account class can manage the balance (i.e. funds), while the user can manage his/her own information (e.g. address, phone).
- As such, from the description below, we can see that a user can update his/her contact information, while a bank account allows for depositing, withdrawing, and transferring funds.

Banking System

A user can update some contact information, such as an address, phone number, etc.
Moreover, he/she can withdraw and deposit funds into his/her bank account.
Moreover, he/she can transfer funds from one account to another.

Identifying Behavior of a Class (2)

User
firstName: String lastName: String gender: ENUM dateOfBirth: Date emiratesID : String nationality: ENUM phoneNumber: String
setPhoneNumber(phone Number: String)

BankAccount
number: String balance: Float currency: ENUM
withdraw(funds: float) deposit(funds:float) transfer(funds:float,accou nt: BankAccount)

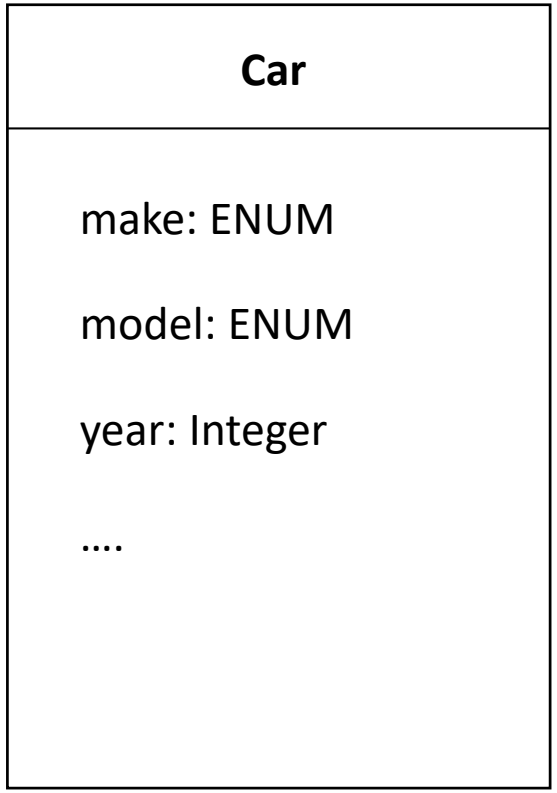
- When we write function names, by convention, the names start with a small letter. We should also specify the input name type and name if available, and the type of the output, if available

Banking System

A user can update some contact information, such phone number, etc. Moreover, he/she can withdraw and deposit funds into his/her bank account. Moreover, he/she can transfer funds from one account to another.

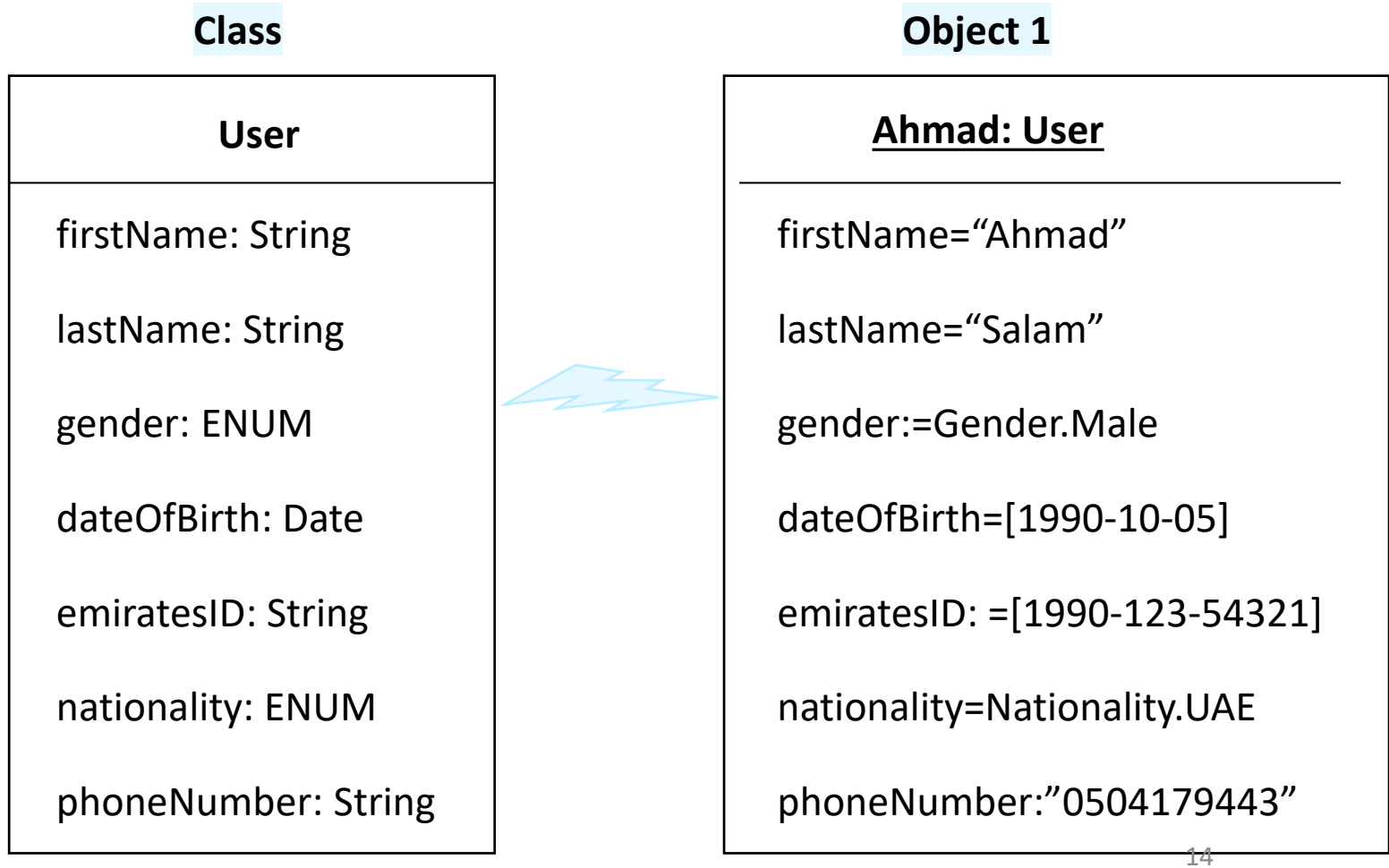
Task 2

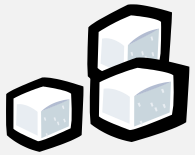
- Using your experience, identify functions that can be listed in the Car class.



Classes vs Objects

- A class is a blueprint that lists the attributes and their types as well as the functions representing a concept.
- An object, on the other hand, is an instance of a class, which shows actual values of the attributes of the class.





Classes vs Objects (More Examples)

Class

User
firstName: String
lastName: String
gender: ENUM
dateOfBirth: Date
emiratesID: String
nationality: ENUM
phoneNumber: String

Object 1

Ahmad: User
firstName="Ahmad"
lastName="Salam"
gender=Gender.Male
dateOfBirth=[1990-10-05]
emiratesID: =[1990-123-54321]
nationality=Nationality.UAE
phoneNumber:"0504179443"

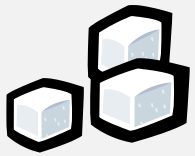
Object 2

Maha: User
firstName="Mahad"
lastName="Razak"
gender=Gender.Male
dateOfBirth=[1992-08-12]
emiratesID: =[1992-153-39215]
nationality=Nationality.EGY
phoneNumber:"0504179323"

Task 3

- Given the Car class, show three created objects representing the Car class.

Car
make: ENUM
model: ENUM
year: Integer
color: ENUM
speed: Float
numberOfDoors: Integer

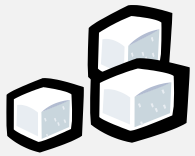


Defining the User Class and the Constructor

- To define a class in Python, we use the syntax: class User, where “User” is the name of the class.
- A constructor is a special function called when an object of the class is created.
- The purpose of the constructor is to initialize the class’s attributes. The attributes are initialized based on the given parameters in the function.
- The **`self`** keyword refers to an instantiated object of the class. It is used as the first parameter of all functions of that class.
- To refer to the attributes (data members) of the class, we use the self keyword. For instance, **`self.firstName`** refers to the attribute “firstName” of the class “User”.

```
class User:
    #constructor to initalize the three main attributes
    def __init__(self, firstName, lastName, gender):
        self.firstName = firstName
        self.lastName = lastName
        self.gender = gender
```

User
-firstName: String -lastName: String -gender: ENUM
+getFirstName():String +setFirstName(firstName:String) +getLastName():String +setLastName(lastName:String) +setGender(gender:Gender) +getGender():ENUM



Defining setters and getters

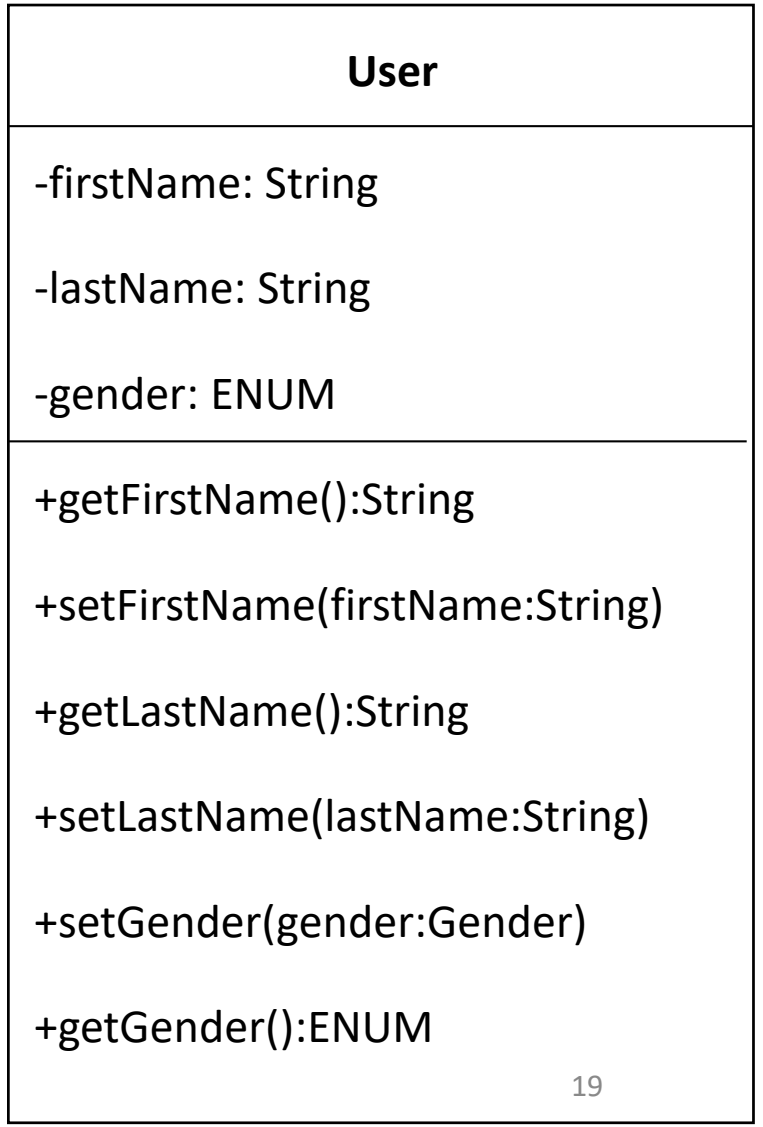
- A setter function allows the changing of an attribute.
- A getter function allows the reading of an attribute.
- Below is an example for a setter and getter functions for the firstName attribute:

```
#a function allowing the changing of the first name
    def setFirstName(firstName):
        self.firstName = firstName
#a function allowing the reading of the first name
    def getFirstName():
        return self.firstName
```

User
-firstName: String -lastName: String -gender: ENUM
+getFirstName():String +setFirstName(firstName:String) +getLastName():String +setLastName(lastName:String) +setGender(gender:Gender) +getGender():ENUM

Implementing The User Class (Full Code)

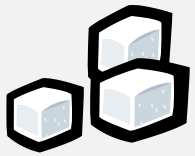
```
class User:
    #constructor to initalize the three main attributes
    def __init__(self, firstName, lastName, gender):
        self.firstName = firstName
        self.lastName = lastName
        self.gender = gender
    #a function allowing the changing of the first name
    def setFirstName(firstName):
        self.firstName = firstName
    #a function allowing the reading of the first name
    def getFirstName():
        return self.firstName
    #a function allowing the changing of the last name
    def setLastName(lastName):
        self.lastName = lastName
    #a function allowing the reading of the last name
    def getLastName(self):
        return self.lastName
    #a function allowing the changing of the gender
    def setGender(gender):
        self.gender = gender
    #a function allowing the reading of the gender
    def getGender():
        return self.gender
```



Task 4

- Write Python code to implement the Car class.
- Please define a constructor.
- Please define setters and gets for the attributes.

Car
make: ENUM
model: ENUM
year: Integer
color: ENUM
speed: Float
numberOfDoors: Integer



Creating ENUM types

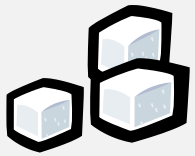
- Enumerator types are useful for allowing you to define values from a finite list of choices.
- For instance, for Gender, the values are male and female.
- Each value in the list is given a number, for instance, male=1, while female=2
- We use enumerator types because it limits the user to enter one of the possible values, making the code more robust.
- The code below shows how to implement the Gender enumerator

```
from enum import Enum

class Gender(Enum):
    male=1
    female=2
```

Gender
male=1
female=2

Nationality
UAE=1
UK=2
EGY=3
KSA=4
USA=5
…



Using ENUM types

- To use the ENUM type, simply refer to the ENUM class that you created and the option you want, for instance, see this example:

```
gender=Gender.female
```

- To print the name of the gender (“male” or “female”), use the name attribute.
- To print the value of the gender (1 for “male”, 2 for “female), use the value attribute.
- See this example

```
print("The gender is:",gender.name)
print("The gender is:",gender.value)
```

Gender
male=1
female=2

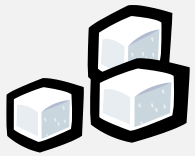
Nationality
UAE=1
UK=2
EGY=3
KSA=4
USA=5
…

Task 5

- Create enumerator types for make, model, and color for the Car class.

- Test the enumerator types you created. Show the names and the values.

Car
make: ENUM
model: ENUM
year: Integer
color: ENUM
speed: Float
numberOfDoors: Integer



Creating Objects

- To create an object, we use the constructor, and provide the parameter values defined in __init__
- There are two ways to do it. The first is to supply the values without specifying the parameter names, but we need to ensure that the order of the values matches the order of the parameter names in the constructor.
- The second way is to specify the parameter names, and in this way, the order of the values does not matter.

```
#an example of creating an object..make sure you match the values
# with the parameter order in init
ahmad_1=User("Ahmad","Salam",Gender.male)
#another example of creating an object..no need to match the order
of the values
#with the parameter orders because we are specifying the parameter
names
ahmad_2=User(lastName="Salam",firstName="Ahmad",gender=Gender.male)
```

Class

User
firstName: String
lastName: String
gender: ENUM

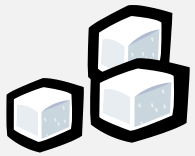
Object 1

Ahmad: User
firstName="Ahmad"
lastName="Salam"
gender:=Gender.Male

Task 6

- Create two objects for two cars of your choice.

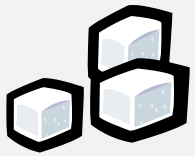
Car
make: ENUM
model: ENUM
year: Integer
color: ENUM
speed: Float
numberOfDoors: Integer



Private vs Public Attributes and Functions

- Both attributes and functions can be set as private or public.
- A private attribute or function is represented by a “-” sign in the UML class, whereas a public attribute or function is represented by a “+” sign.
- A private attribute or function can only be accessed from inside the class.
- A public attribute or function can be accessed from outside the class.
- Typically, attributes are private to protect them from being directly accessed, and potentially corrupted by outside classes.
- Functions, on the other hand, are typically made public as we want outside classes to access them.

User
-firstName: String -lastName: String -gender: ENUM
+getFirstName():String +setFirstName(firstName:String) +getLastName():String +setLastName(lastName:String) +setGender(gender:Gender) +getGender():ENUM



Implementing Private/Public Attributes/Functions

- By default, the functions and attributes are implemented to be public. However, to implement private attributes in Python, we use double underscores as prefixes to the attribute name. See this example.

```
class User:
    #constructor to initalize the three main attributes
    def __init__(self, firstName, lastName, gender):
        self.__firstName = firstName
        self.__lastName = lastName
        self.__gender = gender
```

- Make sure you implement the changes (making the attributes private) everywhere in the code. See this example.

```
    #a function allowing the changing of the first name
    def setFirstName(firstName):
        self.__firstName = firstName
    #a function allowing the reading of the first name
    def getFirstName():
        return self.__firstName
```

User
-firstName: String -lastName: String -gender: ENUM
+getFirstName():String +setFirstName(firstName:String) +getLastName():String +setLastName(lastName:String) +setGender(gender:Gender) +getGender():ENUM

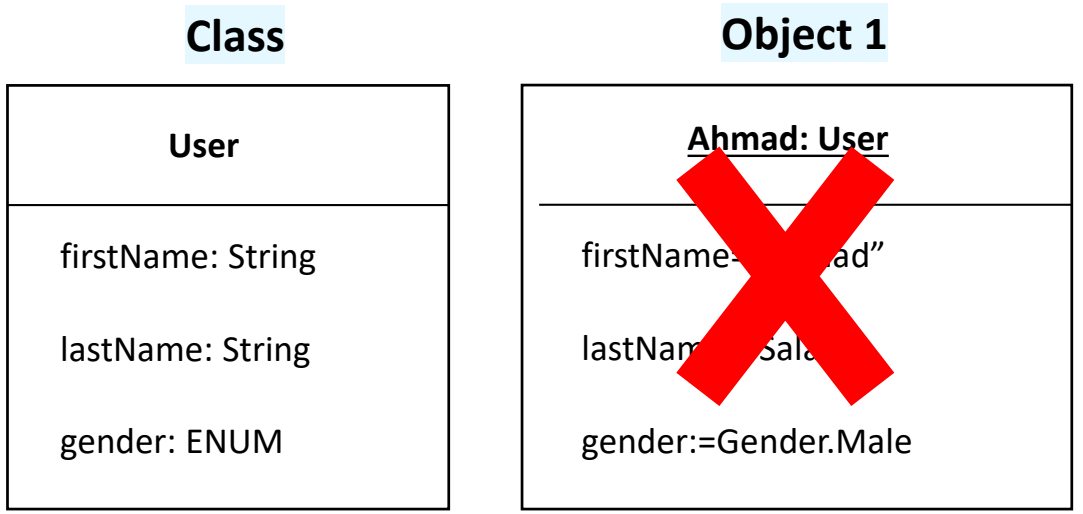
Destroying Objects

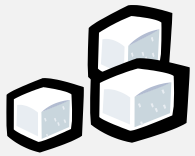
- Python has a garbage collector that automatically deletes objects when they are no longer in use, for example, when the program exits.
- We can specifically write code to be called upon destroying an object. See this example.

```
def __del__(self):
    print("The object was destroyed")
```

- We can also explicitly delete an object by using the del command. See this example.

```
ahmad_1=User("Ahmad","Salam",Gender.male)
del ahmad_1 #destroy the object that we just created
```





Defining other functions

- Not all functions are setters and getters.
- Some functions allow the user to implement certain logic specific to the class.
- For instance, for the BankAccount class, the withdraw function allows the withdrawal of funds from the account, causing the balance to be reduced by the withdrawn amount. See this example.

```
class BankAccount:
    def __init__(self,number,balance,currency):
        self.__number=number
        self.__balance=balance
        self.__currency=currency

    def withdraw(self,funds):
        if funds<=self.__balance:
            self.__balance-=funds
            return True
        return False #insufficient funds
```

Bank Account
-number: String -balance: Float -currency: ENUM
withdraw(funds: float) deposit(funds:float)

Task 7

- Implement the deposit function

Bank Account
-number: String -balance: Float -currency: ENUM
withdraw(funds: float) deposit(funds:float)