Appendix: Introduction to PlantUML

PlantUML is an open-source tool that allows users to create UML diagrams using a simple textual language. It supports various UML diagrams, such as use case diagrams, class diagrams, sequence diagrams, activity diagrams, and more. The main advantage is that diagrams can be created and updated by editing plain text, which is ideal for version control and collaboration. In this LAB you are **advised** to use PlantUML or other tools like Mermaid (https://docs.mermaidchart.com/) due to the limited capabilities of the submission system for uploading images during the **final exam**.

For full reference and syntax of PlantUml, see:

https://plantuml.com/guide/

You can render your diagrams using PlantUML's web service, see:

https://editor.plantuml.com/uml

Note: When copying the code snippets from this document, you might encounter a syntax error caused by the comment indicator character: " ' ". If you delete that line or replace " ' " from your own keyboard, the error will disappear.

Important: The following examples are only to demonstrate the capabilities of the PlantUML language. They are not meant to represent realistic systems.

1. Use Case Diagram

A use case diagram describes the functional requirements of a system and the interactions between actors and use cases. Here is an example of a simple online banking system:

```
@startuml
actor Customer
actor "Bank Employee" as Employee

rectangle "Online Banking System" {
    (Login)
    (Contact Support)
    (Transfer Money)
    (Pay Bills)
    (Approve Transfers)
    (Manage Accounts)

    'Relationships
    Customer --> (Login)
    Customer --> (Transfer Money)
    Customer --> (Pay Bills)
    Customer --> (Contact Support)

    (Transfer Money) .> (Login) : <<include>>
    (Pay Bills) .> (Transfer Money) : <<extend>>

    Employee --> (Approve Transfers)
    Employee --> (Manage Accounts)
}
@enduml
```

Explanation:

- `-->` : standard association.
- `.>` : dashed arrow for `include` and `extend` relationships.
- `<<include>>`: indicates mandatory inclusion of another use case.
- `<<extend>>`: indicates optional/conditional behavior extending a base use case.

2. Class Diagram

Class diagrams show structure, attributes, methods, and relationships. PlantUML supports multiple relationship types: association, aggregation, composition, and inheritance.

```
@startuml
class Customer {
    +name: String
    +email: String
    +login(): void
}

class Account {
    +accountNumber: String
    +balance: double
    +deposit(amount: double): void
    +withdraw(amount: double): void
}

class SavingsAccount
class CheckingAccount

class Transaction {
    +date: Date
    +amount: double
    +type: String
    +execute(): void
}

'Relationships
Customer "1" --> "*" Account : owns
Account "1" --> "*" Transaction : logs
Account <|-- SavingsAccount
Account <|-- CheckingAccount
Account o-- Transaction : composed of
@enduml
```

Explanation:

- `-->` : association
- `o--` : composition (strong ownership)
- `<|--` : inheritance/generalization
- Multiplicity can be added with `"1" --> "*"`

3. Sequence Diagram

Sequence diagrams show dynamic interactions between objects over time.

```
@startuml
actor Customer
participant "Login Service" as LoginService
participant Account
participant Transaction

Customer -> LoginService: login(username, password)
LoginService -> Customer: authenticationResult()
```

```
Customer -> Account: checkBalance()
Account -> Customer: balance

Customer -> Account: transferMoney(amount, targetAccount)
Account -> Transaction: createTransaction(amount, targetAccount)
Transaction -> Account: updateBalance()
Account -> Customer: confirmation()
@enduml
```

Explanation:

- `actor`: external user
- `participant`: system components
- Arrows represent messages; top-down order shows temporal sequence.