Tutorial Sheet – Week 3: UML

Part A

1. **What is a primary purpose of a *Use Case Diagram* in UML?**
   - □ To detail the internal design of classes and interfaces
   - □ To model actors, their goals, and how they interact with the system
   - □ To represent hardware deployment architecture
   - □ To specify exact database schemas
2. **Which statement about *UML Sequence Diagrams* is true?**
   - □ They model static relationships between classes and interfaces
   - □ They illustrate object interactions in a time-sequenced manner
   - □ They cannot represent asynchronous messages
   - □ They have no way to show object creation or destruction
3. *_When describing multiplicity in UML, “1..”_ signifies**
   - □ At least one, possibly many
   - □ Optional, maximum one
   - □ Zero or more
   - □ Exactly 1
4. **If we have a “Customer” class in a UML diagram with an arrow pointing to “Order,” labeled “places,” which statement is correct?**
   - □ It’s an example of a dependency; the arrow must be dashed
   - □ It’s an association, meaning Customer places 0..* Orders
   - □ It’s a generalization from Order to Customer
   - □ It’s an “include” relationship typical of Use Case diagrams
5. **Which statement about UML Object Diagrams is accurate?**
   - □ They model final deployment nodes in a network environment
   - □ They show instances of classes at a particular moment in time with specific attribute values
   - □ They strictly represent concurrency patterns in the

system

□ They only show abstract classes, not concrete instances

Part B

1. Suppose you're building a small domain model for a "Movie Rental" system. Which classes, attributes, and relationships might you include in a class diagram? Justify key associations or multiplicities.
2. Outline the *key elements* of a UML sequence diagram (lifelines, messages, activation bars, etc.) and provide a scenario example. Why is time-ordering helpful?
3. In a UML class diagram, how do you decide on multiplicities (e.g., `1..*` vs `0..1`)? Provide an example class relationship from a library system or e-commerce domain, explaining your multiplicity choice.
4. How would you illustrate polymorphism in a UML class diagram for different "Document" types (e.g., PDFDoc, WordDoc)? Show how a sequence diagram might also highlight the polymorphic call if the system calls `doc.print()` on various doc types.

Part C

1. You have a class diagram with classes `Client`, `Account`, and `Transaction`, where `Client` *has-a* `Account`, and `Account` *has-many* `Transaction`. Show how you'd translate that into Java classes with fields and relationships.
2. You receive Java classes for `Book`, `Author`, and `Publisher`. "Book" references "Author" in a one-to-one, and "Book" can have multiple "Publisher" references for translations. Sketch a UML class diagram showing these relations with multiplicities.

3. Suppose you have an interface `Shape` with classes `Rectangle` and `Circle` implementing it. Show a short snippet of code in Java plus the UML Class Diagram depicting polymorphism. Include a method `draw()` in `Shape` and override it in each shape.
4. A use case “User checks order status.” Actor: “Customer.” Basic flow: Customer enters order ID, system retrieves order info, displays status. Show how you might create a `CheckOrderStatusController` class, a domain `Order` class, and a snippet reflecting the use case steps. Optionally, produce a short UML Sequence Diagram to match it.
5. If you have a code snippet:

   ```
   class Course {
      String name;
      Professor prof;
   }
   class Professor {
      String profName;
   }
   public class Demo {
      public static void main(String[] args) {
          Professor p = new Professor("Dr. Sai");
          Course c1 = new Course("OOP", p);
          Course c2 = new Course("Algorithms", p);
      }
   }
   ```

   Construct a UML Object Diagram for the runtime instance state after `main()` executes.

Ideal Answers (Model Solutions)

Below are the **model solutions** for **all** the questions above. Use them to compare and refine your own answers. If you have any doubts, please bring them to the next lab or discussion forum!

Part A.

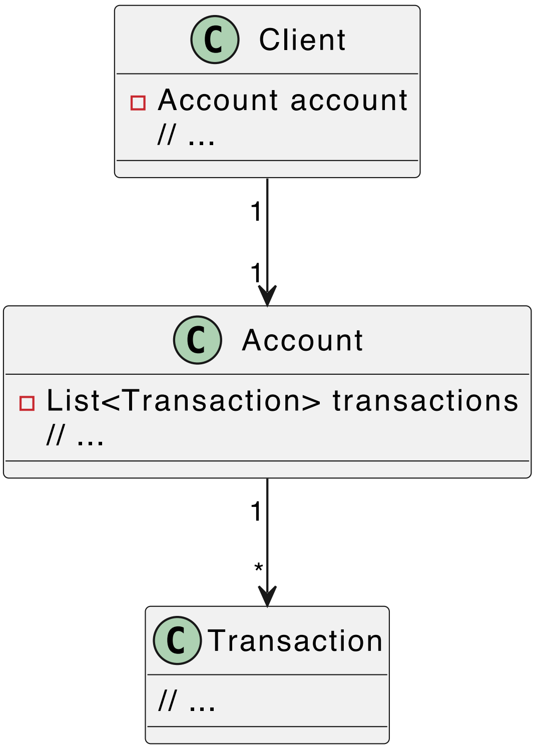
1. **Answer:** B. Use Case Diagrams show actors and their goals with the system.
2. **Answer:** B. Sequence diagrams illustrate object interactions in time order.
3. **Answer:** A. “1..*” means at least one, possibly more.
4. **Answer:** B. An association from Customer to Order means “places” is the label, often 1..* or 0..* Orders.
5. **Answer:** B. Object diagrams show instances and their links at a given time.

Part B

1. **Class Diagrams & Domain Modelling**
    - In “Movie Rental,” classes might be: `Movie`, `Customer`, `Rental`, `Payment`, possibly `Store` or `Inventory`.
    - Key relationships: `Customer` can have many `Rentals`. `Rental` references 1 `Movie`. Possibly, a `Payment` for each rental. Multiplicities: e.g., “A Customer has 0..* Rentals.”
2. **Sequence Diagram Fundamentals**
    - Key elements: *lifelines*, *messages*, *activation bars*. E.g. “User calls `placeOrder()`, which calls `PaymentService.charge()`, which calls `BankAPI.process()`. The time ordering helps clarify how responsibilities pass among objects.
3. **Associations and Multiplicity**
    - Decide multiplicities by real domain logic. E.g. a `Customer` can place many `Orders`, so `1..*`. If optional, maybe `0..*`. *Example:* `Library => Book (1..*)`, i.e. library has many books.
4. **Polymorphism in UML**
    - Class diagram: `Document` is abstract or interface, `PDFDoc` and `WordDoc` implement or extend it. A sequence diagram might show `printManager.printDocument(doc)` calling `doc.print()` at runtime, doc can be PDF or Word.

Part C.

1. **Class Diagram to Java**
    - UML:
        - **Client** -(1..1)--> **Account**
        - **Account** -(1..*)--> **Transaction**



~~~
   - **Java**:

```java
class Client {
    private Account account; // 1..1
    // ...
}

class Account {
    private List<Transaction> transactions; // 1..*
    // ...
}

class Transaction {
~~~

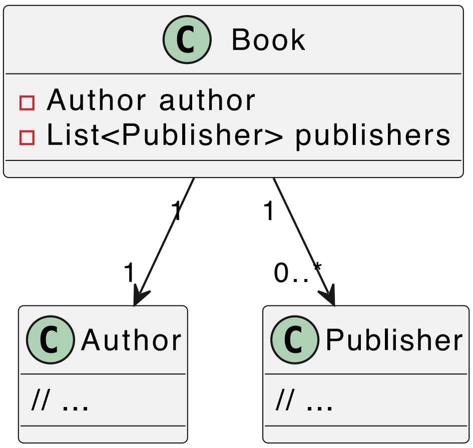
~~~~
      // ...
  }
  ```


    Explanation: *Client* has exactly one `Account`;
 `Account` has many `Transaction`.
~~~~

1. **Java Code ➔ Class Diagram**
   - **Given**:

   ```
    class Book {
        private Author author;
        private List<Publisher> publishers; // multiple for
   translations
    }
    class Author { /* one-to-one with Book? or many? */ }
    class Publisher { /* can appear multiple times, or many
   books. */ }
   ```

   - **UML**:



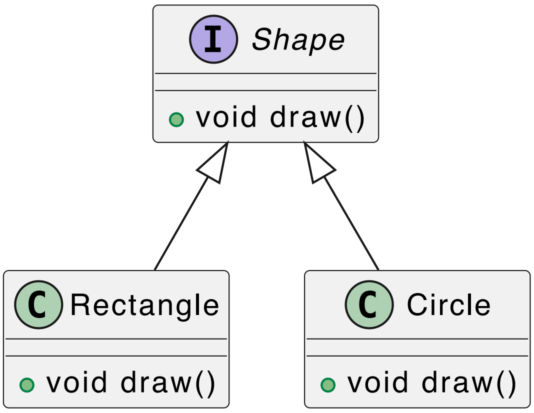
   Explanation: Possibly each Book references a single Author, but many Publishers.

2. **UML + Polymorphism**
   - **Code**:

   ```
    interface Shape { void draw(); }
    class Rectangle implements Shape {
      public void draw() { System.out.println("Drawing
   Rectangle"); }
    }
   ```

```
 class Circle implements Shape {
   public void draw() { System.out.println("Drawing
Circle"); }
 }
```

- **UML**:



Explanation: They implement `Shape`, showing polymorphism.

3. **Use Case → Java Skeleton**
   - Use Case: “User checks order status.”

```
 class CheckOrderStatusController {
     OrderRepository repo;
     public void checkStatus(String orderId) {
         Order o = repo.findOrder(orderId);
         if (o != null) {
             System.out.println("Order Status: " +
o.getStatus());
         } else {
             System.out.println("Order not found.");
         }
     }
 }
 class Order {
     private String status;
     public String getStatus() { return status; }
     // ...
 }
```

   - Potential Sequence Diagram: `User -> CheckOrderStatusController -> OrderRepository -> CheckOrderStatusController -> User`.

4. **Object Diagram**

- **Code**:

```
Professor p = new Professor("Dr. Sai");
Course c1 = new Course("OOP", p);
Course c2 = new Course("Algorithms", p);
```

- **Object Diagram** (runtime):

```
p:Professor
   profName = "Dr. Sai"

c1:Course
   name = "OOP"
   prof -> p

c2:Course
   name = "Algorithms"
   prof -> p
```

Explanation: We see real instances with attribute values.