**NATIONAL UNIVERSITY OF COMPUTER AND**

**EMERGING SCIENCES**

**SL-2002 – Software Design & Architecture Lab**

**Lab Instructor: Zarnain Maryam Awan**

**Lab 10**

**Component Diagram:**

A component diagram is a type of structural diagram in the Unified Modeling Language (UML) that illustrates the components of a system and their relationships. It provides a high-level view of the architecture of a software system, focusing on the modular structure and the interactions between different components.

In a component diagram:

1. **Components**: Components are represented as boxes or rectangles, each representing a modular part of the system. These can be software modules, classes, packages, subsystems, or even larger system elements.
2. **Relationships**: Relationships between components are depicted using lines or arrows connecting them. These relationships may include dependencies, associations, aggregations, or compositions.
3. **Interfaces**: Components may expose interfaces that define the contracts or services they provide. Interfaces specify the methods or operations that other components can invoke, facilitating communication and interaction between components.
4. **Ports**: Ports represent points of interaction between components. They define the interfaces through which components communicate with each other. Ports are often depicted as small circles or squares on the edges of component boxes, connected to other components through lines representing communication paths.
5. **Grouping**: Components can be grouped into packages or subsystems, representing logical groupings of related functionality within the system. Packages help organize the components and clarify the overall system architecture.

Component diagrams are valuable for visualizing the structure of a system, understanding how components interact, and identifying opportunities for modularization, reuse, and scalability. They are commonly used during the design and architecture phases of software development to communicate system structure to stakeholders and development teams.

**Steps To Create A Component Diagram:**

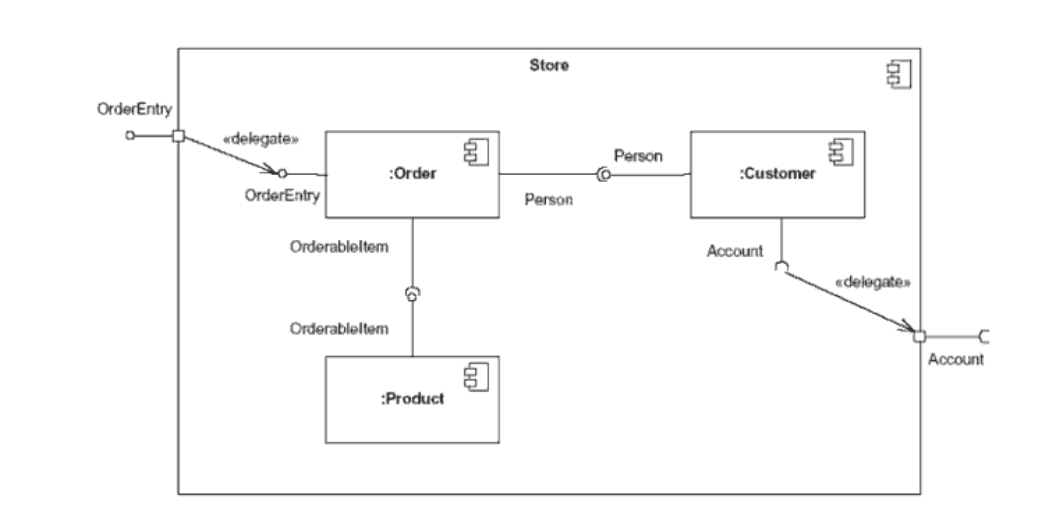
Drawing a component diagram involves several steps to effectively model the structure and relationships between components within a system. Here's a step-by-step guide:

1. **Identify Components**: Begin by identifying the major components of your system. These could be modules, classes, packages, or larger system elements.
2. **Add Components**: Start by adding the identified components to your diagram. Represent each component as a box or rectangle, and label it with its name and/or type.
3. **Group Components**: Group related components together using subsystem components or packages. This helps organize the diagram and clarify the structure of the system.
4. **Add Details**: Include additional details such as lines, component interfaces, ports, relevant to your system.
5. **Review and Refine**: Review the diagram to ensure it accurately represents the structure and relationships of your system. Make any necessary refinements or adjustments to improve clarity and completeness.
6. **Document**: Finally, document the component diagram to provide context and explanation for readers who may not be familiar with the system. Describe the purpose of each component and its relationships with others.
7. **Iterate**: As your system evolves, revisit the component diagram to reflect any changes or updates. Iteratively updating the diagram ensures it remains an accurate representation of the system architecture over time.

By following these steps, you can create a comprehensive component diagram that effectively communicates the structure and relationships within your system.

**Example:**

 The Store component provides the interface of OrderEntry and requires the interface of Account. The Store component is made up of three components: Order, Customer, and Product components. The OrderEntry port delegates to the Order component's OrderEntry interface for processing. Also, the internal Customer component's required Account interface is delegated to the Store component's required Account interface port. By connecting to the Account port, the internals of the Store component (e.g. the Customer component) can have a local representative of some unknown external entity which implements the port's interface. The required Account interface will be implemented by a component outside of the Store component.



Top of Form