OOAD AND UML

UNIT -5

TWO MARKS

1. Responsibilities:

* A responsibility is a contract or an obligation of a class. When you create a class, you are making a statement that all objects of that class have the same kind of state and same kind of behavior.

Responsibilities
-determine the risk of a customer order -handle customer specific criteria for fraud.

2. Relationships:

*When you model a system, not only must you identify the things that form the vocabulary of your system, you must also model how these things stand in relation to one another.

*There are three kinds of relationships: They are

1. Dependencies: In which represent using relationships among classes.

2. Generalization: In which link generalized classes to their specializations.

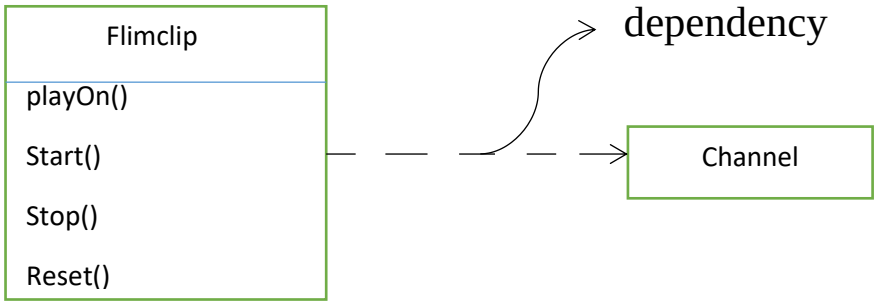
3. Association: In which represent structural relationships among objects.

3. Dependency:

*A dependency is a connection among things.

*A dependency is a using relationship that states that a change in specification of one thing may affect another thing that uses it the reverse.

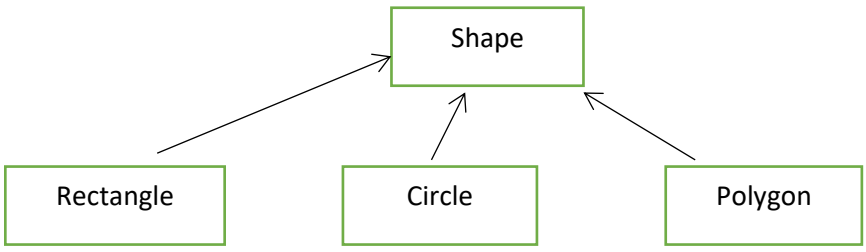
*A dependency is rendered as a dashed directed line, directed to the thing being depended on. Use dependencies when you want to show one thing using another.



4. Generalization:

*A generalization is a relationship between a general thing and a more specific kind of that thing.

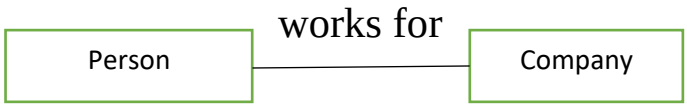
*A generalization means that objects of the child may be used anywhere the parent may appear, but not the reverse.



5. Association:

*An association is structural relationship that specifies that objects of one thing are connected to objects of another.

*Given an association connecting two classes, you can navigate from an object of one class to an object of the other class.



6. Notes:

*A note may contain any combination of text or graphics.

*A note is a graphical symbol for rendering constraints or comments attached to an element or a collection of elements.

*Graphically, a note is rendered as a rectangle with a dog-eared corner, together with a textual or graphical comment.

7. Stereotypes:

*A Stereotype is an extension of the vocabulary of the UML, allowing you to create new kinds of building blocks similar to existing ones but specific to your problem.

*Graphically, a stereotype is rendered as a name enclosed by guillemets and place above the name of another element.

*The stereotype element may be rendered by using a new icon associated with that stereotype.

8. Tagged values:

*A Tagged values is an extension of the properties of a UML element, allowing you to create new information in that element's specification.

*Graphically a tagged value is rendered as a string enclosed by brackets and placed below the name of another element.

9. Constraint:

*A constraint is an extension of the semantics of a UML element, allowing you to add new rules or to modify existing ones.

*Graphically, a constraint is rendered as a string enclosed by brackets and placed near the associated element or connected to that element or elements by dependency relationships.

10. System:

* A system is a collection of subsystems organized to accomplish a purpose and described by a set of models, possibly from different viewpoint.

11. Subsystem:

*A subsystem is a grouping of elements of which some constitute a specification of the behavior offered by the other contained elements.

12. Model:

*A model is a semantically closed abstraction of a system, meaning that it represents a complete and self- consistent simplification of reality, created in order to better understand the system.

13. View:

*A view is a projection into the organization and structure of a system's model, focused on once aspect of that system.

14. Diagram:

*A diagram is the graphical presentation of a set of elements, most often rendered as a connected graph of vertices and arcs.

15. Class diagram:

*A class diagram shows a set of classes, interfaces and collaborations and their relationships. Class diagrams are most common diagrams found in modeling object oriented systems.

*This class diagrams illustrate the static design view of a system.

16. Object diagram:

*An object diagram shows set of objects and their relationships.

*Object diagrams address the static design view or static process view of a system but from the perspective of real or prototypical cases.

17. Component diagram:

*A component diagram shows a set of components and their relationships.

*Component diagrams are related to class diagrams in that a component typically maps to one or more classes, interfaces or collaborations.

18. Deployment diagrams:

*A deployment diagram shows a set of nodes and their relationships.

*Deployment diagrams are related to components diagrams in that a node typically encloses one or more components.

19. Use case diagram:

*A use case diagram shows a set of use cases and actors and their relationships.

*Use case diagrams are especially important in organizing and modeling the behaviors of a system.

20. Interaction diagrams:

*Interaction diagram is the collective name given to sequence diagrams and collaboration diagrams.

*All sequence diagrams and collaborations are interaction diagrams and it is either a sequence diagrams and collaboration diagram.

21. Sequence diagram:

*A sequence diagram is an interaction diagram that emphasizes the time ordering of messages.

*It shows a set of objects and the messages sent and received by those objects.

22. Collaboration diagram:

*A collaboration diagram is an interaction diagram that emphasizes the structural organization of the objects that send and receive messages.

*It shows a set of objects, links among those objects, and messages sent and received by those objects.

23. Statechart diagram:

*A statechart diagram shows a state machine, consisting of states, transitions, events and activities.

*Statechart diagrams emphasize the event ordered behavior of an object, which is especially useful in modeling reactive systems.

24. Activity diagram:

*An activity diagram shows the flow from activity to activity within a system. An activity shows a set of activities, the sequential or branching flow from activity to activity, and objects that act and are acted upon.

*Activity diagram emphasize the flow of control among objects.

25. Forward engineering:

*Forward engineering is the process of transforming a model into code through a mapping to an implementation language.

*It results in a loss of information, because models in the UML semantically richer than any current object oriented programming languages.

26. Reverse engineering:

*Reverse engineering is the process of transforming a code into a model through mapping from a specific implementation language.

*It results in a flood of information some of which is at lower level of detail than you will need to build useful models.

27. Objects and roles:

*The objects that participate in an interaction are either concrete things or prototypical things.

*As a concrete thing, an object represents something in the real world.

28. Links:

*A link is a semantic connection among objects. A link is an instance of an association.

*A class has an association to another class, there may be a link between the instances of the two classes, wherever there is a link between two objects, one object can send a message to the other objects.

29. Messages:

*A message is the specification of a communication among object that conveys information with the expectation that activity will ensue.

*The receipt of a message instance may be considered an instance of an event.

30. Sequencing:

*When an object passes a message to another object, the receiving object might in turn send a message to another object, which might send a message to yet a different object, and so on.

*This stream of messages forms a sequence. Any sequence must have a beginning; the start of every sequence is rooted in some process or thread.

31. Use case:

* A use case describes a set of sequencies, in which each sequence represents the interaction of the things outside the system with the system itself.

*These behaviors are in effect system level functions use to visualize, specify, construct, and document the intended behaviors of your system during requirements capture and analysis.

32. Actors:

*An Actor represents a coherent set of roles that users of use cases play when interacting with these use cases.

*An actor represents a role that a human, a hardware device, or even another system plays with a system.

33. Action states:

*Action states cannot be decomposed.

*These states are atomic meaning that events may occur, but the work of the action state is not interrupted,

*The work of an action state is generally considered to take insignificant execution time.

34. Activity states:

*Activity states can be decomposed.

*These states are not atomic meaning that they may be interrupted and general are considered to take some duration to complete.

35. Transitions:

*When the action or activity of a state completes, flow of control passes immediately to the next action or activity state.

*You specify this flow by using transitions to show the path from one action or activity state to the next action or activity state.

36. Branching:

*A branch may have one incoming transition and two or more outgoing ones.

*On each outgoing transition, you place a Boolean expression, which is evaluated only should not overlap, but should cover all.

37. Forking:

*A Forking represent the splitting of a single flow of control into two or more concurrent flows of control.

*A Fork may have one incoming transition and two or more outgoing transitions, each of which represents an independent flow of control.

38. Joining:

*A Joining represents the synchronization of two or more concurrent flows of control.

*A Join may have two or more incoming transitions and one outgoing transitions. Above the join, the activities associated with each of these paths continues in parallel.

39. Swimlanes:

*When you modeling workflows of business processes, to partition the activity states on an activity diagram into groups, each representing the business organization responsible for those activities.

*In UML, each group is called a swimlane because, visually, each group is divided from its neighbor by a vertical solid line.

40. Components and classes:

*Classes represent logical abstractions; components represent physical things that live in the world of bits.

*Components represent the physical packaging of otherwise logical components and are at different level of abstraction.

*Classes may have attributes and operation directly; components only have operations that are reachable only through their interfaces.

41. Components and interfaces:

*The component that realizes the interface is connected to the interface using a full realization relationship.

*An interface that a component realizes is called an export interface, meaning interface that the component provides as a service to other components.

*The interface that a components uses is called an import interfaces, meaning interface that component conforms to and so builds on.

42. Types of components:

*Three kinds of components:

1. Deployment components: These are the components necessary and sufficient to form an executable system, such as dynamic libraries and executable.

2. Work product components: These components are essentially the residue of development process, consisting of things such as source code files and data files from which deployment components are created.

3. Execution components: These components are created as a consequence of an executing system, such as a COM+ object.

43. UML standard elements:

The UML defines five standard stereotypes that apply to components:

1. executable- specifies a component that may be executed on a node.

2. library- specifies a static or dynamic object library.

3. table- specifies a component that represents a database table.

4. file- specifies a component that represents a document contain source code.

5. document- specifies a component that represents a document.

44. Nodes and components:

*Components are things that participate in the execution of a system; nodes are things that execute components.

*Components represent the physical packaging of otherwise logical elements; nodes represent the physical deployment of components.

45. Pattern:

*A pattern is a common solution to a common problem in a given context.

*Patterns are part of the UML because patterns are important parts of developer's vocabulary.

*Patterns helps to visualize, specify, construct, and document the artifacts of a software-intensive system.

46. Mechanisms or design patterns:

*A mechanism is another name for a design pattern that applies to a society of classes.

*A mechanism simply names a set of abstractions that work together to carry out some common and interesting behavior.

*A mechanism names a template for a set of abstractions that work together to carry out some common and interesting behavior.

47. Frameworks or architectural pattern:

*A framework is an architectural pattern that provides an extensible template for applications within a domain.

*A framework is bigger than a mechanism. When you specify a framework, you specify the skeleton of an architecture, together with the slots, tabs, knobs and dials that you expose to users who want to adapt that framework to their own context.

48. CRUD:

*For simple CRUD is Create, Read, Update, Delete Operations, implement them with standard SQL or ODBC calls.

*It is the process of operations used in component diagrams for modeling the physical databases.

49. Trace relationships:

*A trace is represented as a stereotyped dependency.

*You can model the conceptual relationship among elements that live in different models by using a trace relationship; a trace may not be applied among elements in the same model.

50. Adornments:

*Adornments are textual or graphical items that are added to an element’s basic notation and are used to visualize details from the element’s specification.

*Most adornments are rendered by placing text near the elements of interest or by adding a graphic symbol to the basic notation.