PassQueen
a.k.a dumpsqueen.com

OMG-Certified UML Professional Advanced Exam

OMG OMG-OCUP-300

Version Demo

Total Demo Questions: 13

Total Premium Questions: 133

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, UML Infrastructure	12
Topic 2, UML Superstructure	119
Topic 3, Meta Object Facility (MOF)	2
Total	133

QUESTION NO: 1

Which action retrieves the end objects from an instance of an association class?

A. ReadLinkObjectEndAction

B. ReadStructuralFeatureAction

C. ReadVariableAction

D. ReadSelfAction

E. ReadLinkAction

ANSWER: A

Explanation:

`ReadLinkObjectEndAction` is the UML action specifically defined to retrieve an object at a specified end of a link that is represented by an instance of an AssociationClass. An association class combines the properties of an association, including its ends, with the ability to have instances that hold attributes and participate in behavior. Consequently, when a behavior has an association-class instance (the link object) and needs the object connected at one of that association's ends, it uses this action. The action's input pin accepts the association-class instance, while its end property identifies which association end is to be read; its result output pin supplies the corresponding end object. This provides an explicit executable-model mechanism for navigating from the link object to one of the objects related through that association. The UML specification defines this action in its Actions package as the form of link-end reading applicable when the link itself is modeled as an object through an association class. See the OMG UML 2.5.1 specification, especially the Actions clause: OMG Unified Modeling Language 2.5.1 PDF. The specification's official landing page and normative document links are also available at OMG UML 2.5.1.

QUESTION NO: 2

What is usually true of an ExecutionEnvironment?

A. an implementation of a state machine

B. a standalone type of node

C. part of a more general node

D. an infrastructure in which actions occur

ANSWER: C

Explanation:

An ExecutionEnvironment is usually *part of a more general node*. In UML deployment modeling, an ExecutionEnvironment is a specialized Node that represents the context in which deployed software artifacts or components execute. Typical examples include an operating system, a Java virtual machine, an application server, a database-management system, or a workflow engine. These environments are commonly hosted by, or nested within, a more general Node representing the physical or virtual computational resource on which they run, such as a server, device, virtual machine, or container host.

This relationship allows a deployment diagram to show the relevant hosting structure at an appropriate level of detail. For example, a server node can contain an operating-system execution environment, which can in turn contain an application-server execution environment. Software may then be deployed to the execution environment that actually provides the runtime services it needs. The UML specification describes an ExecutionEnvironment as a Node that represents a particular execution environment for executing behavior, and its notation and semantics support showing execution environments as nested nodes. See the OMG UML 2.5.1 specification and the OMG UML specification overview.

QUESTION NO: 3

What does the protocol conformance between two protocol state machines mean?

A. the specific state machine must abide by the behavior of the general state machine

B. all triggers in the two protocol state machines must be the same

C. the two protocol state machines must be the same

D. the general state machine must abide by the behavior of the specific state machine E. the specific state machine must have the same number of states and transitions as the general machine

ANSWER: A

Explanation:

Protocol conformance means that the specific protocol state machine must abide by the behavior of the general protocol state machine. In UML, a protocol state machine specifies the permitted sequence of operations or events for a classifier, including the legal states in which operations may occur and the conditions under which state changes are valid. A conformance relationship allows a more specific classifier to refine that protocol while preserving compatibility with the protocol promised by its more general classifier.

Accordingly, the specific protocol may introduce detail, such as additional states or more precise behavior, but it must remain substitutable wherever the general protocol is expected. Its observable behavior must therefore satisfy the constraints established by the general protocol. This captures UML's generalization principle: a specialized element can extend or refine an inherited contract, but it cannot violate that contract. Protocol conformance is particularly useful for modeling interfaces, components, and classes whose valid operation sequences are inherited and specialized across a type hierarchy.

The UML specification defines protocol state machines and their conformance relationships in its state-machine package; see the OMG UML 2.5.1 Specification and the OMG UML 2.5.1 specification overview.

QUESTION NO: 4

What is true of composite aggregation? (Choose two)

A. A part may be included in at most one composite at a time.

B. The multiplicity of the composite end is at most one.

C. A part must always be created when its composite is created.

D. A part may simultaneously belong to any number of composites.

E. A composite association has no ownership semantics.

ANSWER: A B

Explanation:

In a composite aggregation, a part may be included in at most one composite at a time. This expresses the strong ownership implied by composition: a part cannot simultaneously be a constituent of multiple composite objects.

Also, the multiplicity of the composite end is limited to at most one. A composite association end therefore identifies no more than one composite owner for each part. UML composition may also imply lifecycle responsibility, but it does not require every part to be created at the same time as its composite. See the Associations clause in the OMG UML 2.5.1 specification.

QUESTION NO: 5

What is true about a parameterized class? (Choose two)

A. A class with parameters that are names of other, imported classes, and the binding tells which actual classes are imported.

B. A class where all methods in the class are parameterized.

C. A class with different variants, and the parameters tell how the class may vary.

D. Names of elements within a parameterized class can be parameters of the class. E. A class with parameters where the actual parameters are provided when creating instances of the class.

E. A class where some of the elements used for its definition are parameters.

ANSWER: C E

Explanation:

A parameterized class is a UML template: it defines a reusable class structure whose definition includes formal parameters. Those parameters identify the aspects of the class definition that may be supplied or substituted when the template is bound. Consequently, the description *"A class where some of the elements used for its definition are parameters"* directly captures the essential UML concept. Such parameters can represent parameterable model elements used by the template's definition, allowing the same general class design to be reused with different actual elements.

The description *"A class with different variants, and the parameters tell how the class may vary"* is also correct. A template defines a family of related model elements rather than a single fixed class. A template binding supplies actual parameters for the formal parameters and produces a bound element representing a particular variant of that general definition. For example, a collection-like class can be defined once and bound using different element types, yielding specialized variants while retaining the same reusable structure. UML specifies templates, template parameters, and template bindings in its Templates package; a parameterized class is a Classifier with a template signature. See the OMG UML 2.5.1 specification and the OMG UML specification landing page.

QUESTION NO: 6

What kind of relationship is an information flow in UML 2.0?

A. connector

B. dependency

C. association

D. transition

ANSWER: B

Explanation:

An information flow is a specialized UML dependency relationship. It represents the conveyance of information—such as data, signals, or control information—from one or more sources to one or more targets. Its purpose is to show that specified information items flow between model elements without requiring the modeler to specify the detailed mechanism by which that transfer occurs. The relationship is normally drawn as a dependency-style dashed arrow, stereotyped «flow», with the arrow directed from the information source to the information target. UML permits information flows to be realized by more concrete elements or relationships, including connectors, associations, and messages; however, those may describe the mechanism that realizes the flow rather than the abstract information-flow relationship itself. This distinction lets an analyst communicate what information is conveyed while deferring implementation or structural details. The UML 2.0 Superstructure specification defines InformationFlow as a kind of PackageableElement and Dependency, and identifies its sources, targets, and conveyed classifiers. See the OMG UML 2.0 Superstructure specification, section 17.3, UML 2.0 Superstructure PDF, and the current UML specification's normative materials at OMG UML specifications.

QUESTION NO: 7

What is true of a PackageMerge? (Choose two)

A. The receiving package is extended by the contents of the merged package.

B. Corresponding elements with the same name and kind may be merged.

C. It makes the merged package a subclass of the receiving package.

D. It is identical in semantics to a PackageImport.

E. It requires every element in the two packages to have a distinct name.

ANSWER: A B

Explanation:

A PackageMerge is a directed relationship between a receiving package and a merged package. Its effect is that the receiving package is extended by the contents of the merged package. It supports incremental construction of a package definition from reusable package fragments.

When corresponding elements in the packages have the same name and are of the same kind, they are merged into a resulting element that combines their definitions. Package merge is therefore different from package import, which makes members available for use in a namespace without combining their definitions. See the Packages clause in the OMG UML 2.5.1 specification.

QUESTION NO: 8

The preconditions of behavioral features of an interface can be specified in what way?

A. the actions of its protocol state machine

B. one or more guard conditions of a protocol state machine

C. comments attached to the corresponding features

D. the set of its features

E. the parameters of its protocol state machine

ANSWER: B

Explanation:

one or more guard conditions of a protocol state machine is correct because a UML protocol state machine defines the permitted order in which an interface's behavioral features, such as operations and receptions, may occur. Its protocol transitions represent valid invocations in particular protocol states. A guard on such a transition expresses the condition that must hold for the transition—and therefore for the associated behavioral feature invocation—to be valid. Consequently, the guard serves as the feature's precondition in that protocol state.

This use is especially valuable where an interface has a lifecycle or usage protocol. For example, an operation may be available only after an object has been initialized, or only while it is in an active state. The protocol state machine makes that contract explicit to clients without prescribing the internal implementation of the classifier that realizes the interface. UML specifies that protocol transitions can have guards and that these guards specify the preconditions under which the associated operation or reception may occur. This makes guarded protocol transitions the standard UML mechanism for expressing interface behavioral-feature preconditions. See the OMG UML specification's protocol state machine and protocol transition definitions: OMG UML 2.5.1 Specification (PDF) and OMG UML specification page.

QUESTION NO: 9

What determines which value an object flow edge chooses to move from the source?

(Choose two)

A. upper bound

B. weight

C. transformation

D. effect

E. ordering

F. selection

ANSWER: C F

Explanation:

An object flow can use a *selection* behavior to determine which token or tokens are selected from those offered at the source before they are conveyed along the edge. This behavior is especially relevant where multiple values are available: it provides the semantic rule for choosing the value or values that the object flow will accept and transfer. The selection behavior is therefore the direct mechanism that determines the choice from the source.

A *transformation* behavior also participates in determining the value moved by an object flow. After a token has been selected, the transformation behavior may produce a replacement value that is offered to the target. Thus, selection determines the source token(s) used, while transformation determines the value representation subsequently conveyed by applying a behavior to the selected token.

These are defined properties of UML `ObjectFlow`. UML specifies that an object flow may have a selection behavior for selecting tokens from those offered by its source, and a transformation behavior for transforming token values before they are offered to the target. See the OMG UML specification's Activity and ObjectFlow semantics in the OMG UML 2.5.1 specification and the official OMG UML specification page.

QUESTION NO: 10

What is NOT true of a transition in a protocol state machine?

A. may have a postcondition

B. may have a precondition

C. may have multiple associated operation

D. may have no associated operation

ANSWER: C

Explanation:

The statement "may have multiple associated operations" is not true for a protocol state-machine transition. A protocol state machine models the permitted sequences of operation invocations and related events on a classifier, and its protocol transitions express the legal change from one protocol state to another. Where a protocol transition is triggered by an operation call, that operation is identified through the CallEvent of its trigger. UML constrains a ProtocolTransition to have at most one trigger, so it cannot represent multiple associated operation calls on the same transition. Distinct operations that lead from the same source state must therefore be represented by separate protocol transitions (or modeled through an appropriate single triggering event). This restriction keeps the protocol precise: each transition states one specific contractual interaction that may move an instance between protocol states. The UML metamodel also provides optional precondition and postcondition constraints for ProtocolTransition. These constraints define the condition required before the protocol interaction and the condition that is established after it, reinforcing the transition's role as a behavioral contract. See the OMG UML specification's ProtocolTransition definition and constraints in the OMG UML 2.5.1 specification, and the official OMG UML specification page.

QUESTION NO: 11

What is true of an InteractionOperand? (Choose two)

A. It is an operand of a CombinedFragment.

B. It may have a guard.

C. It is a kind of Lifeline.

D. It can occur only in a loop CombinedFragment.

E. It owns the MessageEnds of every Message in an Interaction.

ANSWER: A B

Explanation:

An InteractionOperand is an operand of a CombinedFragment. It represents the portion of an interaction that is executed according to the semantics of the combined-fragment operator.

An InteractionOperand can have a guard. Guards are particularly significant for operators such as *alt* and *opt*, where they state the condition under which an operand is selected. An *opt* CombinedFragment has one operand, whereas an *alt* CombinedFragment may have multiple operands. See the Interactions clause in the OMG UML 2.5.1 specification.

QUESTION NO: 12

The InfrastructureLibrary satisfies which of the following design requirements? (Choose two)

A. defines IDL interfaces so that model interchange is fully supported

B. architecturally aligns UML, MOF, and OCL so that model interchange is fully supported

C. architecturally aligns UML, MOF, and CWM so that model interchange is fully supported

D. allows customization of UML through profiles and new languages based on the same metalanguage core as UML

E. defines a metalanguage core that can be used to define a variety of metamodels, including UML, MOF, and CWM

ANSWER: D E

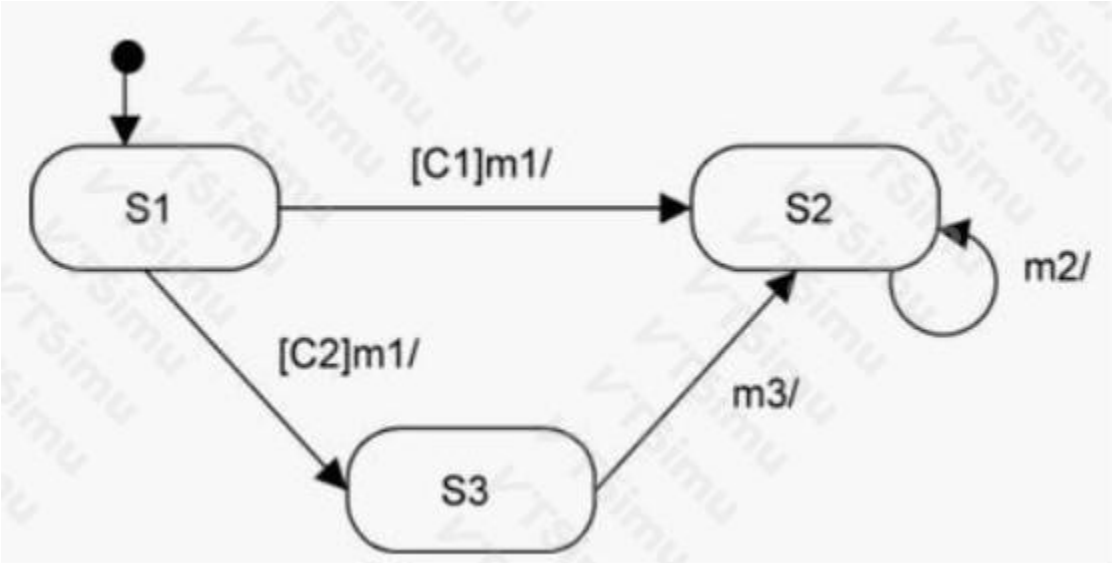
Explanation:

The InfrastructureLibrary is intended to provide a common, reusable foundation for defining UML and related modeling languages. Its design supports customization of UML through profiles, which add domain-specific terminology, constraints, and extensions without changing the underlying UML metamodel. It also supports the definition of new languages that share the same foundational metalanguage constructs. This enables languages to remain conceptually and architecturally compatible with UML while tailoring their vocabulary and semantics to a particular domain.

The InfrastructureLibrary also supplies the core metalanguage concepts used in the construction of metamodels. These foundational concepts provide the basis on which UML, MOF, and CWM can be defined or aligned. Reusing this common core promotes consistent treatment of fundamental modeling concepts and provides an architectural basis for interoperability among OMG modeling specifications. The UML specification describes the InfrastructureLibrary as a reusable core library for the definition of UML and other metamodels, while the MOF specification defines the framework used for specifying and managing metamodels. See the OMG UML 2.5.1 specification and the OMG MOF 2.5.1 specification.

QUESTION NO: 13

In the exhibit, what does the protocol state machine in the diagram specify?



A. Either condition C1 or condition C2 must be true before the associated classifier can be invoked.

B. Both condition C1 or condition C2 must be true before the associated classifier can be invoked.

C. Only behavioral features m1, m2, and m3 can be invoked on instances of the associated classifier.

D. A client of the associated classifier must always first invoke behavioral feature m1.

ANSWER: D

Explanation:

A client of the associated classifier must always first invoke behavioral feature m1. A protocol state machine defines the permitted order of operation invocations on instances of its associated classifier. Its transitions represent the valid protocol: an instance begins in its initial protocol state, and the trigger on the transition leaving that initial state identifies the operation that is permitted to move the instance into the next state. In the diagram, that initial transition is triggered by *m1*; consequently, a client must invoke *m1* before it can legally use operations whose transitions are enabled only in later protocol states. This is a constraint on the externally visible usage protocol of the classifier, rather than an implementation description of how the operations execute. Protocol state machines are specifically intended to express legal sequences of calls, including any guards or conditions that affect which subsequent call is permitted after a state has been reached. See the OMG UML specification's definition of ProtocolStateMachine and its use in specifying the legal behavior of classifier instances: OMG Unified Modeling Language (UML), Version 2.5.1. The OMG UML specification landing page is available at https://www.omg.org/spec/UML/.