PassQueen
a.k.a dumpsqueen.com

OMG Certified UML Professional 2 (OCUP 2) - Advanced Level

OMG OMG-OCUP2-ADV300

Version Demo

Total Demo Questions: 11

Total Premium Questions: 114

Buy Premium PDF

https://passqueen.com

support@passqueen.com

Topic Break Down

Topic	No. of Questions
Topic 1, UML 2.5 Infrastructure	22
Topic 2, UML 2.5 Superstructure	92
Total	114

QUESTION NO: 1

Choose the correct answer:

What is correct about the modeling of individual things with UML?

A. A UML Model can contain individual things (e.g. Instances) UML does not prescribe the level of detail to be used in the description.

B. A UML Model can contain statements about individual things If a thing is an instance of a Classifier, these statements must be consistent (all mandatory Properties must be defined, all Constraints must be satisfied).

C. A UML Model can contain statements about individual things These statements can be incomplete, imprecise, and abstract, but not wrong.

D. A UML Model can contain statements about individual things. These statements can be incomplete, imprecise, abstract, may turn out to be wrong, or even be asserted as counterfactual

E. A UML Model can only contain statements about sets of individual things (e.g. Classifiers).

ANSWER: D

Explanation:

“A UML Model can contain statements about individual things. These statements can be incomplete, imprecise, abstract, may turn out to be wrong, or even be asserted as counterfactual” is correct because UML is a general-purpose modeling language whose models express claims made by a modeler about a subject matter; a model is not necessarily a complete, literal record of a running system. A model may describe a particular individual through an *InstanceSpecification*, which is the UML metaclass used to represent an instance in a modeled system. Such a representation can state only selected facts, use abstraction, or defer values and details that are not relevant to the model’s purpose.

Moreover, UML provides notation and semantics for expressing modeled information, including instances, classifiers, properties, slots, and constraints, but it does not make every assertion in a model a guaranteed fact about reality. A model may represent a proposed design, a hypothetical scenario, a test case, an analysis assumption, or an explicitly counterfactual situation. Its assertions can therefore later prove inaccurate when additional information becomes available or when the real system differs from what was modeled. This is consistent with UML’s role as a language for specifying and communicating models rather than as a mechanism that certifies truth or completeness. The normative UML definition of InstanceSpecification and the surrounding semantic framework are available in the OMG UML 2.5.1 specification; OMG also maintains the UML specification landing page.

QUESTION NO: 2

Choose the correct answer:

Which behavioral process can be modeled by a FunctionBehavior?

A. populating a file-based Spreadsheet with mathematical functions and their input values

B. turning on or off a light in room depending on whether people are or are not in that room

C. taking the next 5 elements out of a First in First Out queue

D. return an array containing the Fibonacci sequence (0, 1, 1, 2, 3, 5, …, n) for a given parameter n

E. UML has no concept of FunctionBehavior.

ANSWER: D

Explanation:

“return an array containing the Fibonacci sequence (0, 1, 1, 2, 3, 5, …, n) for a given parameter n” is a suitable FunctionBehavior because it describes a computation whose result is determined by its input parameter. A UML FunctionBehavior represents a Behavior that models a mathematical function: execution produces output values from input

values without causing observable side effects. Generating the Fibonacci values up to the requested bound can therefore be specified as a self-contained calculation. Its result can be conveyed through output parameters, while the calculation itself need not alter objects outside the behavior, consume items from a shared collection, or interact with an external device or file. UML permits a FunctionBehavior to have parameters, so the value of *n* can be an input parameter and the resulting array can be an output parameter. This makes the behavior especially appropriate where an activity, operation, or opaque expression requires a deterministic, side-effect-free derived value. The UML metamodel defines FunctionBehavior as a specialization of OpaqueBehavior for this purpose; its semantics align with a function evaluation rather than an interaction with the surrounding system. See the OMG's UML 2.5.1 specification and the UML specification overview.

QUESTION NO: 3

Choose the correct answer:

A composite State has two orthogonal Regions. Each Region reaches its own FinalState after completing its work. The composite State has an outgoing Transition with no trigger, and no doActivity remains executing for the composite State.

When can the triggerless Transition be taken?

A. After either Region reaches its FinalState.

B. After both Regions have completed.

C. Immediately when the composite State is entered.

D. Only when an external event explicitly triggers the Transition.

ANSWER: B

Explanation:

The Transition can be taken **after both Regions have completed**. A composite State with orthogonal Regions is complete only when all of its Regions are complete. Reaching a FinalState completes an individual Region; once the composite State is complete, its completion event can enable an outgoing triggerless Transition.

Completion of only one Region is insufficient because the other Region remains active. The Transition is not taken immediately when either Region is entered, and it does not require an external trigger because it is a completion Transition.

QUESTION NO: 4

Choose the correct answer:

Which statement is correct about a Decision Node?

A. A Decision Node shall have at most one incoming Activity Edge

B. A Decision Node shall have at most two incoming Control Flows

C. If a Decision Node has an incoming Control Flow and a decisionInput, then a decisionInput shall have a single in Parameter.

D. If a Decision Node has an incoming Object Flow, a decisionInput, and a decisionInputFlow, then a decisionInput shall have two in Parameters.

ANSWER: C

Explanation:

"If a Decision Node has an incoming Control Flow and a decisionInput, then a decisionInput shall have a single in Parameter." is correct. A UML DecisionNode directs the flow of tokens through outgoing ActivityEdges according to guards and, where defined, a decision-input Behavior. The decision-input Behavior provides a value that is used in making the routing decision. UML specifies the parameter count of that Behavior according to the kind of incoming flow available at the

node. Where the DecisionNode has an incoming ControlFlow, the decision-input Behavior receives a single input Parameter. This is the well-defined input needed to invoke the Behavior in the control-flow case and supports consistent decision evaluation during activity execution. The constraint is part of the normative semantics and well-formedness rules for DecisionNode in the UML Activities package; it is not merely a diagramming convention. It ensures that a model's decision-input Behavior has a parameter signature appropriate to the DecisionNode's incoming token context. See the OMG's UML 2.5.1 specification, particularly the DecisionNode constraints in the Activities chapter, and the OMG UML specification landing page for the current published UML specification materials.

QUESTION NO: 5

Choose the correct answer:

What should a modeler do lo represent specific hardware environments In a deployment model?

A. Create a profile with the applicable details

B. Create an instance of a deployment manifest with the applicable details

C. Create an instance of a deployment specification with the applicable details.

D. Create a mapping between the logical hardware description and the physical hardware description with the applicable details.

ANSWER: A

Explanation:

Create a profile with the applicable details is correct because a UML profile is the standard UML extension mechanism for adapting a model to a particular domain, platform, or implementation environment without changing the UML metamodel itself. A deployment model can use profile-defined stereotypes, tagged properties, and constraints to express the details that distinguish a target hardware environment, such as processor architecture, memory capacity, operating system, network interface characteristics, virtualization technology, or device-specific capabilities. The modeler can then apply those stereotypes to relevant deployment nodes and devices, producing a model that remains UML-conformant while conveying hardware information meaningful to stakeholders and tools. UML's deployment concepts provide the general semantic foundation for modeling nodes, devices, execution environments, artifacts, and their deployment relationships; profiles provide the controlled way to specialize those concepts when the standard elements do not capture the required platform detail. This approach also promotes consistent reuse: the same hardware vocabulary and constraints can be applied across multiple deployment diagrams and projects. The OMG UML specification defines profiles as a lightweight extension mechanism and defines the deployment-modeling concepts to which such extensions may be applied. See the OMG UML 2.5.1 specification and the OMG UML specification page.

QUESTION NO: 6

Choose the correct answer:

A composite structure contains a port on its boundary and a port on one of its internal Parts. Incoming requests received at the boundary port must be forwarded to the internal Part for handling. The connection is not intended to represent communication between two peer Parts.

What kind of Connector is required?

A. Assembly Connector

B. Delegation Connector

C. Dependency

D. Generalization

ANSWER: B

Explanation:

A **delegation Connector** links a Port of an encapsulated Classifier to a Port of one of its internal Parts. It expresses that requests or signals received through the external Port are delegated to the internal Part, or that behavior is exposed outward through that Port.

An assembly Connector connects the provided and required Interfaces of connectable elements, commonly between Parts. It does not express the boundary-to-internal delegation relationship. A Dependency and a Generalization likewise do not define communication paths in a composite structure.

QUESTION NO: 7

Choose the correct answer:

Consider the following diagram:

When this behavior is executed, which event will occur last?

A. finalizing HH

B. reception of z

C. reception of w

D. We cannot determine the last event from this diagram alone.

ANSWER: D

Explanation:

We cannot determine the last event from this diagram alone. UML sequence-diagram semantics define a partial ordering of occurrence specifications, rather than one universal chronological sequence for every event in the interaction. The ordering is established by the order of occurrences along an individual lifeline, by the send-before-receive relationship of a message, and by any explicit GeneralOrdering relationships. A page layout in which one message end happens to be drawn lower than another does not, by itself, impose an ordering between events on independent lifelines. Consequently, a reception on one pair of lifelines, a reception on another pair, and a destruction/finalization occurrence on a separate lifeline may be unordered relative to one another unless the diagram contains a causal chain or explicit ordering constraints connecting them. Different valid executions can therefore place such independent events in different relative orders. To identify one event as necessarily last, the interaction would need enough semantic ordering constraints to show that every other relevant occurrence precedes it. This follows UML's definition of an Interaction as a set of lifelines and message occurrences subject to ordering constraints; it is not simply read as a total top-to-bottom timeline. See the OMG's UML 2.5.1 specification, especially the Interaction and OccurrenceSpecification semantics, and the OMG UML specification index.

QUESTION NO: 8

Choose the correct answer: What is a minimal reflexive metamodel?

A. a metamodel whose interpretation maps completely onto itself

B. a metamodel that can define class and activity model elements

C. a metamodel that can define a modeling language of conformance level 1

D. a metamodel that can define a modeling language of conformance level 2

ANSWER: A

Explanation:

A minimal reflexive metamodel is correctly described as *"a metamodel whose interpretation maps completely onto itself."* Reflexivity means that the metamodel has sufficient modeling constructs to specify its own abstract syntax: its metaclasses,

properties, associations, and related structural features can themselves be represented using those same constructs. “Minimal” emphasizes that this self-description does not require a separate, richer meta-metamodel outside the metamodel being interpreted. In other words, the interpretation of the modeling concepts closes on the metamodel itself, producing a self-contained foundation for defining modeling languages. This is the essential idea behind the reflective architecture used by OMG modeling standards: MOF supplies a small set of core concepts for defining metamodels, and those concepts are themselves specified in a reflective manner. Such a foundation is valuable because it gives model elements and metamodel elements a uniform semantic basis while avoiding an endless sequence of additional meta-levels. The OMG UML specification describes UML’s relationship to its metamodeling infrastructure, while the MOF specification defines the core reflective facilities used to construct and exchange metamodels. See the OMG Meta Object Facility (MOF) 2.5.1 specification and the OMG UML 2.5.1 specification.

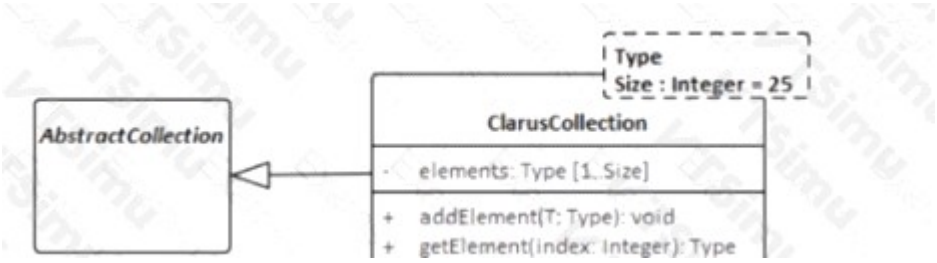
QUESTION NO: 9

Choose the correct answer:

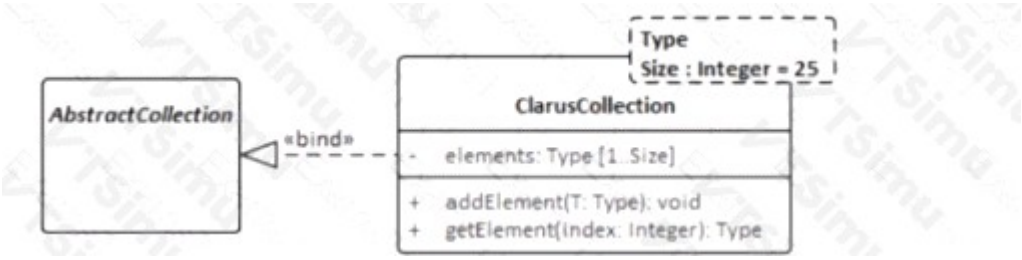
A project's requirements call for flexibility in the collection class design. Most of the collections will be a fixed length of 25 elements. However, allowance must be made in the design for collections that are a fixed length longer than 25.

Which model fragment supports the project's requirements?

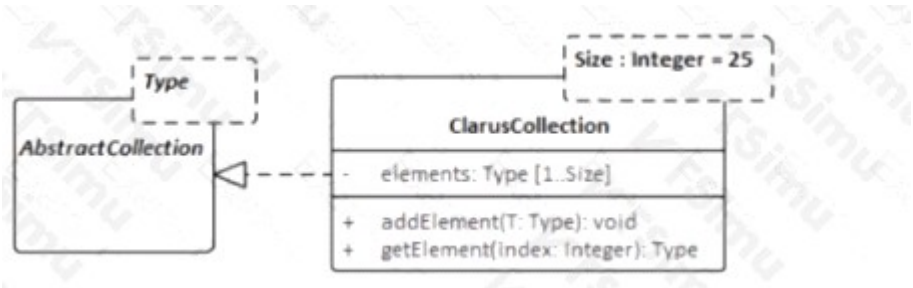
A)



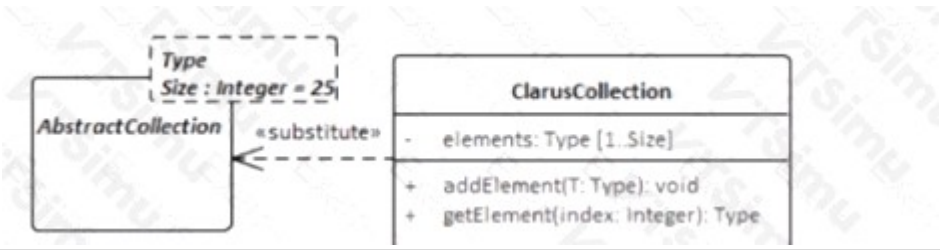
B)



C)



D)



A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: C

Explanation:

The model fragment represented by **Option C** correctly uses a UML template to make the collection's capacity a configurable design-time parameter. The collection template has a size parameter whose default value is 25. Consequently, bindings that do not explicitly supply a size produce collection types with a fixed capacity of 25, satisfying the common case without repetitive specification. When a larger fixed capacity is required, a template binding can supply a different value for that same size parameter, producing a distinct collection type with the requested fixed length. The type parameter similarly allows the collection to be bound for the relevant element type.

This is a suitable use of UML templates because template parameters represent formal, substitutable parameters of a model element. A default value makes a supplied argument optional while retaining a defined value whenever no overriding argument is provided. The resulting design expresses both requirements directly in the model: a normal fixed size of 25 and controlled variation to another fixed size, rather than an unconstrained runtime-resizable collection. UML specifies templates and template binding in its Common Structure package; see the OMG UML specification, especially its template-related definitions: OMG UML 2.5.1 Specification.

QUESTION NO: 10

Choose the correct answer:

What is true about the use of a Template Classifier to specify the Type of a Typed Element?

A. The Template Classifier needs to be modeled

B. The Template Classifier must have defaults for all of its Template Parameters

C. The Type Property of the Typed Element must directly name the Template Classifier.

D. Template Classifiers cannot be used in the specification of Types for Typed Elements.

E. A set of bound elements provides values for the Template Parameters of the Template Classifier.

ANSWER: E

Explanation:

A set of bound elements provides values for the Template Parameters of the Template Classifier. A template classifier defines a reusable, parameterized classifier rather than a fully specified concrete classifier on its own. To use that classifier in a context requiring a concrete type, its formal template parameters need corresponding actual parameter values. UML represents this specialization through template binding: the binding relates a bound element to the template signature and contains template parameter substitutions. Each substitution associates a formal template parameter with an actual parameterable element, thereby defining the particular instantiation of the template classifier that is intended. This is what makes a parameterized classifier usable as the specific type denoted for a TypedElement: its generic parameters are supplied with concrete values through the binding mechanism. UML specifies TemplateBinding, TemplateParameterSubstitution, and the relationship between a template signature and its bound element in its Templates package. See the OMG UML specification, especially the Templates package discussion, at OMG UML 2.5.1 Specification (PDF), and the official UML specification landing page at OMG UML Specification.

QUESTION NO: 11

Choose the correct answer:

A sequence diagram states that, after a client receives `authorizationApproved`, its local status must be `authorized` before the next message occurrence on that Lifeline. The requirement constrains the state of the participant at that exact point in the Interaction.

Which UML element should be used?

A. StateInvariant

B. StateMachine

C. TimeConstraint

D. DurationConstraint

ANSWER: A

Explanation:

A **StateInvariant** is correct. A StateInvariant is placed on a Lifeline and constrains the state or condition that must hold at the point in the Interaction where it occurs. It can express that the participant is in a specified state, or that a Boolean expression concerning it is true, before subsequent occurrences are considered.

A StateMachine defines a broader state-based behavior rather than a local assertion at a particular interaction point. A TimeConstraint constrains timing, and a DurationConstraint constrains elapsed time between occurrences, neither of which states the required participant condition.