DUMPSBOSS.

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

OMG OMG-OCUP2-FOUND100

Version Demo

Total Demo Questions: 10

Total Premium Questions: 109

Buy Premium PDF

https://dumpsboss.co

support@dumpsboss.co

support@dumpsboss.co
dumpsboss.co

Topic Break Down

Topic	No. of Questions
Topic 1, UML Basics	16
Topic 2, Structural Modeling	39
Topic 3, Behavioral Modeling	39
Topic 4, Interaction Modeling	15
Total	109

QUESTION NO: 1

Choose the correct answer:

Exactly two Player Instances are involved in the "Play Tennis" Use Case.

Which diagram depicts this scenario correctly?

A)

B)

C)

D)

A. Option A

B. Option B

C. Option C

D. Option D

ANSWER: C

Explanation:

Option C is correct because the described diagram places a multiplicity of `2` at the *Player* end of the communication association connecting the Player actor to the *Play Tennis* use case. In UML, an actor denotes a role played by an external entity, while an actor instance is a particular entity playing that role during a use-case occurrence. The multiplicity at the actor end of the association specifies how many instances of that actor participate in each occurrence of the use case. Therefore, the value `2` precisely expresses that every execution of *Play Tennis* involves exactly two Player instances. This is appropriate for modeling a tennis game as an interaction requiring two participants in the Player role, without requiring two separately defined actor types. UML defines a use case as a specification of behavior performed by a subject in collaboration with one or more actors, and actor/use-case associations may carry multiplicities to constrain participating instances. See the OMG UML specification's use-case notation and semantics in OMG Unified Modeling Language, Version 2.5.1 and the corresponding specification landing page at OMG UML 2.5.1.

QUESTION NO: 2

Choose the correct answer:

For projects involving complex and strategic systems, what is a key advantage of developing models before starting implementation?

A. Developing models ensures that all requirements will be addressed.

B. Models are useful to provide proof of progress to project management.

C. Models help to establish a consensus among all the project stakeholders.

D. Modeling helps to convince developers that models are necessary for good design.

ANSWER: C

Explanation:

Models help to establish a consensus among all the project stakeholders because they provide an accessible, shared representation of a proposed system before the cost and commitment of implementation. Complex, strategic systems usually involve people with different responsibilities and perspectives, such as business sponsors, domain experts, analysts, architects, developers, testers, and operations personnel. UML models give these groups a common vocabulary and visual basis for discussing the system's required behavior, structure, interactions, boundaries, and key design decisions.

Developing and reviewing models early makes assumptions visible and supports productive discussion about what the system is intended to accomplish. Stakeholders can validate whether the model reflects business goals and operational needs, while technical participants can assess feasibility and identify important architectural concerns. This shared understanding helps resolve ambiguities and align expectations before implementation work makes changes more expensive. The OMG describes UML as a language for specifying, visualizing, constructing, and documenting software-intensive systems, purposes that directly support communication and agreement among project participants. See the OMG UML 2.5.1 specification and OMG's UML overview.

QUESTION NO: 3

Choose the correct answer:

What is an advantage of modeling as a part of the software analysis and design process?

A. It reduces the risk of inconsistent or improper implementations.

B. It reduces the risk of incorporating technology constraints into a design.

C. It reduces the risk of using an incorrect or improper programming language.

D. It reduces the risk of the solution being strongly related to business practices.

ANSWER: A

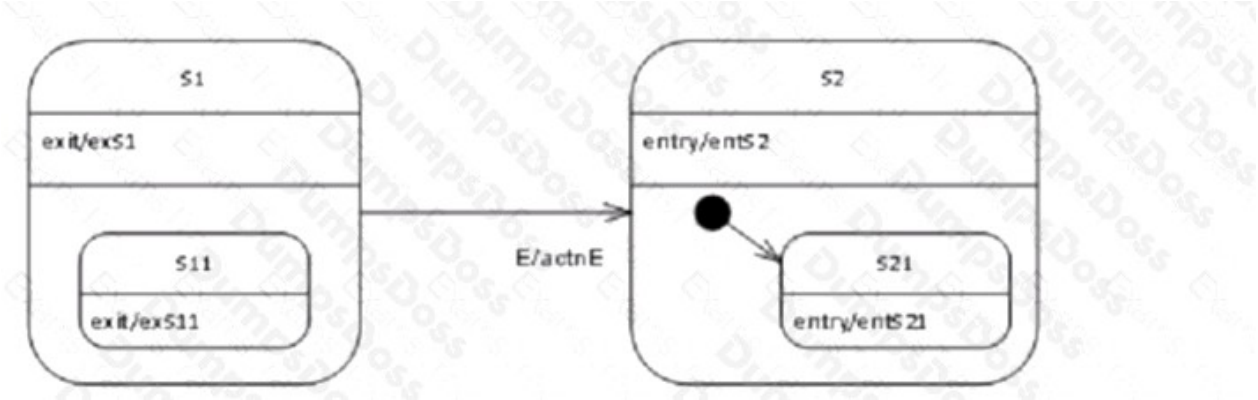
Explanation:

It reduces the risk of inconsistent or improper implementations. Modeling gives analysts, designers, developers, and stakeholders a shared, explicit representation of the system's structure, behavior, requirements, and key relationships before or alongside implementation. This makes assumptions and design decisions visible and reviewable, allowing omissions, contradictions, interface mismatches, and misunderstood requirements to be detected earlier than they might be through code alone. A model also provides a common vocabulary across roles, helping separate parts of a system be designed consistently and integrated according to the same intended behavior. UML is specifically defined by OMG as a visual modeling language for specifying, visualizing, constructing, and documenting software-intensive systems. Its standardized notation supports precise communication without requiring every participant to infer intent from source code or informal descriptions. Consequently, modeling improves traceability from analysis through design and implementation, and it provides a basis for validation before defects become costly to correct. The OMG UML specification is available at OMG Unified Modeling Language Version 2.5.1. OMG also describes UML as a standard visual modeling language at OMG UML.

QUESTION NO: 4

Choose the correct answer:

Which sequence of behavior executions occurs if the state machine below is in state S11 and an event of type E occurs?



A. actnE; exS1; exS11: entS21; entS2

B. actnE; exS1; exS11: entS2; entS21

C. exS11; actnE; entS2

D. gxSH; exS1; actnE; entS2

E. exS11; exS1; actnE; entS2; entS21

ANSWER: E

Explanation:

exS11; exS1; actnE; entS2; entS21 is correct because UML state-machine transition processing follows a defined exit–effect–entry order. The active configuration starts in nested state S11, which is contained by composite state S1. When the transition triggered by E leaves that configuration for S21, the machine first executes exit behavior for the innermost active state, exS11, and then exit behavior for its enclosing state, exS1. It then performs the transition's effect, actnE. Only after all required exits and the transition effect are complete does it enter the target configuration. Entry behavior is performed from the outer target state inward: entS2 executes before entS21, because S21 is nested within S2. This ordering preserves the hierarchical semantics of state machines: exit active states from deepest to shallowest, execute the transition effect, then enter target states from shallowest to deepest. The OMG UML specification defines this ordering as part of transition execution and state entry/exit behavior semantics. See the OMG UML 2.5.1 specification, especially the State Machines chapter, and the current UML specification landing page at OMG UML.

QUESTION NO: 5

Choose the correct answer:

Consider the following invalid state machine fragment:

Why is the diagram invalid?

A. A transition requires a trigger or guard.

B. A guard condition is not allowed on the initial transition.

C. A trigger is not allowed on the transition to the final state.

D. A transition is not allowed to leave and enter the same state.

E. A trigger is not allowed on the initial transition.

ANSWER: E

Explanation:

A trigger is not allowed on the initial transition. In a UML state machine, an initial pseudostate represents the point at which execution starts when its containing region is entered. Its single outgoing transition is taken automatically, rather than in response to an event; consequently, that transition must not have a trigger. It also must not have a guard, because selection among alternative guarded paths is not applicable from an initial pseudostate. Therefore, if the incoming transition to state `S1` originates at the initial pseudostate and is labeled `e`, the label denotes a trigger and makes the fragment invalid. A transition whose source and target are the same state is permitted in UML. Such a self-transition has defined run-to-completion behavior and is commonly used to model handling an event while leaving and re-entering a state. The relevant constraint for the initial pseudostate is specified in the UML State Machines chapter: an initial pseudostate has at most one outgoing transition, and that transition has neither a trigger nor a guard. See the OMG UML 2.5.1 specification, especially the State Machines and pseudostate definitions, and the official UML 2.5.1 specification landing page.

QUESTION NO: 6

Choose the correct answer:

Consider the following class diagram:

Which statement is true about the class diagram?

A. The preferred age to open a bank account is 18 years old or older

B. Only customers who are 18 years old or older can open a bank account.

C. The age condition should only hold when the setAge(Integer) function is called

D. An object of Customer with age set to 18 or greater will raise an exception.

ANSWER: B

Explanation:

Only customers who are 18 years old or older can open a bank account. The diagram's constraint, **{age >= 18}**, is a UML restriction on the modeled value of **age** for the relevant **Person** instances. A UML constraint expresses a condition that must be satisfied by the constrained model element or set of elements; it is not merely descriptive advice or a preferred value. Consequently, a person playing the **customer** role in the association with **BankAccount** must meet the stated age condition. This makes the age threshold a rule of the model for bank-account customers, so the model permits such a customer only when their age is at least 18. The wording "can open a bank account" appropriately captures the business-level implication of the association and its constraint: customer eligibility is restricted by the invariant. UML permits constraints to be written in braces and represented using a natural-language or formal expression. The OMG UML specification defines a Constraint as a restriction on the values or behavior of a model element, supporting the interpretation of **{age >= 18}** as a condition that must hold rather than an optional preference. See the OMG UML 2.5.1 specification and OMG's UML specification page.

QUESTION NO: 7

Choose the correct answer:

Which elements in the diagram are Features of the Car class?

A. drive(), Car

B. stop(). driver

C. name. stop()

D. drive(), stop()

ANSWER: D

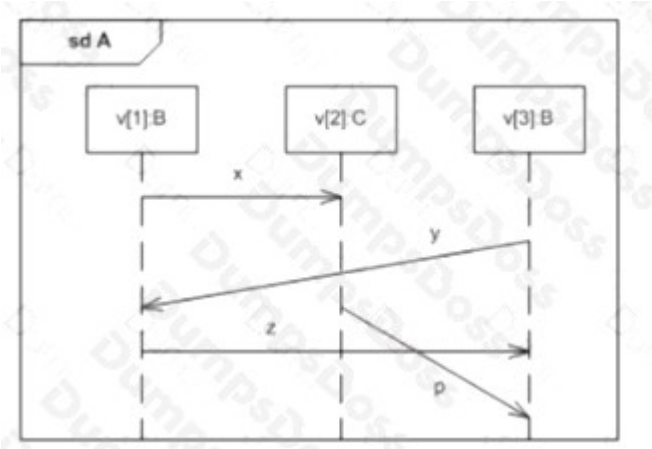
Explanation:

drive(), stop() is correct because these are operations owned by the *Car* class. In UML, a feature is a characteristic of a classifier, including a class. A class's features include its structural features, such as attributes represented by properties, and its behavioral features, such as operations. Therefore, each operation shown in the operations compartment of *Car* is a behavioral feature of that class. The operation names *drive()* and *stop()* denote behavior that instances of *Car* may perform, so both belong to the set of features of *Car*. This classification is based on the UML metamodel rather than on the visual proximity of labels in a diagram: an element is a feature when it is owned by, or inherited by, the classifier as an attribute/property or operation. The OMG UML specification defines *Feature* as an abstract metaclass and identifies structural and behavioral forms of features; *Operation* is the UML element used to specify behavior available on a class. See the OMG's UML 2.5.1 Specification, particularly the metamodel definitions for Feature, BehavioralFeature, and Operation, and the OMG UML 2.5.1 overview.

QUESTION NO: 8

Choose the correct answer:

Which statement is always true about the following sequence diagram?



A. Sending message x is the first occurrence that happens.

B. Sending message z will happen after receiving message x.

C. Sending message y will happen before sending message z.

D. Sending message p will happen before receiving message y.

ANSWER: B

Explanation:

Sending message z will happen after receiving message x. UML sequence diagrams define behavior in terms of occurrences, including message-send events and message-receive events. The vertical ordering of occurrences on an individual lifeline represents an ordering constraint: an occurrence lower on that lifeline occurs after one above it. In the diagram, the receive occurrence for *x* is on the lifeline of *v2[C]*, and the send occurrence for *z* is later, lower on that same lifeline. Therefore, every valid execution represented by this interaction orders receipt of *x* before transmission of *z*. This conclusion does not depend on an assumed global clock or on visually comparing events across separate lifelines; it follows directly from the required event ordering local to *v2[C]*. The UML specification describes a Lifeline as representing an interacting participant and a Message as specifying communication between lifelines, with message ends represented by occurrence specifications. It also defines sequence-diagram ordering through the partial order induced by occurrences and their placement on lifelines. See the OMG's UML 2.5.1 specification, particularly its Interaction, Lifeline, Message, and OccurrenceSpecification semantics, and the OMG UML specification overview.

QUESTION NO: 9

Choose the correct answer: OpaqueExpression can use which languages?

A. onlyOCL

B. only programming languages

C. only Mathematical Expressions

D. any language

ANSWER: D

Explanation:

any language is correct. UML defines an OpaqueExpression as an expression whose meaning is specified by one or more textual bodies, rather than by a UML expression notation with a predefined semantic interpretation. Its `body` property contains the text of the expression, while its optional `language` property identifies the language used for the corresponding body text. UML does not constrain that language to OCL, to a programming language, or to mathematical notation. It can therefore identify an expression written in any language appropriate to the model and its intended execution or interpretation environment, including a domain-specific language, a query language, an implementation language, or an informal textual

notation. The specification also permits multiple body strings and corresponding language strings, allowing an OpaqueExpression to provide equivalent or alternative forms in different languages when needed. This flexibility is the purpose of calling the expression “opaque”: UML records the expression and its declared language without requiring the UML metamodel itself to interpret the expression’s internal syntax. See the OMG UML specification’s definition of OpaqueExpression in the UML 2.5.1 specification and the current OMG UML specification page.

QUESTION NO: 10

Choose the correct answer:

Consider the following diagram fragment:

What makes this fragment invalid?

A. A list of elements is not allowed in a package body.

B. «class»one is missing a visibility.

C. An *interface» is not allowed in a package

D. Private elements are not allowed on the list.

E. Protected elements are not allowed in a package.

F. Element six is missing its stereotype.

ANSWER: E

Explanation:

Protected elements are not allowed in a package. A protected visibility is defined in terms of inheritance: it makes an element visible to elements related by generalization to the namespace that owns the element. A UML Package is a Namespace, but it is not a Classifier and does not participate in the classifier generalization mechanism that gives protected visibility its meaning. Consequently, protected visibility is not applicable to an element owned directly by a Package. Package contents may be represented as a list of packaged elements, and a visibility marker may be omitted when the default visibility is intended. Classes and Interfaces are both PackageableElements and can be packaged; UML keywords such as «class» and «interface» are optional notational aids where the element kind is otherwise clear. Likewise, an element shown without a keyword is not automatically invalid merely because its specific metaclass is not displayed in the package’s member list. The invalid aspect is specifically the protected member notation in the Package. The OMG UML specification defines Packages as namespaces for PackageableElements and defines visibility semantics, including the inheritance-based meaning of protected visibility; see the OMG UML 2.5.1 specification and the OMG UML 2.5.1 specification page.