|  |  |  |
| --- | --- | --- |
| Question Label | Error Code | Error Text |
| Realization 2 | ABS1 | The class is abstract and must be marked with the stereotype <<abstract>>. |
| Realization 2 | ABS2 | Abstract methods must be marked with the stereotype <<abstract>>. |
| C++ Virtual Methods 6 | ABS3 | Even one abstract method makes the class abstract, and the class must be marked abstract. |
|  | ABS4 | This is not an abstract class. All methods (if any) are concrete. |
| Attributes 1 (aggregation) Attributes 2 (composition) | ATTR1 | Show non-primitive attribute as a separate class with the aggregation/composition relationship. Use this format when you want to emphasize the relationship between the two classes. |
| Attributes 3 | ATTR10 | Association is backwards: when a class contains an attribute, the association is from the class to the attribute, not vice-versa. |
| Attributes 3 | ATTR11 | This is a bi-directional association: when a class contains an attribute, the association is from the class to the attribute, not vice-versa. |
|  | ATTR12 | The attribute belongs to the parent class, not the subclass. You can show it as a dependency, but not with aggregation/composition. |
|  | ATTR13 | Instead of an association, use aggregation/composition. |
|  | ATTR14 | Primitive attributes should appear in the middle compartment. |
|  | ATTR15 | Instead of showing the more general dependency, show the more specific aggregation/composition relationship. |
|  | ATTR16 | Aggregation/composition is backwards; diamond goes on the other end. |
|  | ATTR17 | Wrong relationship. Should be aggregation (solid line with a hollow triangle). The triangle is attached to the enclosing class. |
|  | ATTR18 | Protected attributes are marked with #. |
|  | ATTR2 | Show non-primitive attribute inside compartment. Use this format when you want to de-emphasize the relationship between the two classes. |
| Attributes 1 (aggregation)  Attributes 2 (composition) | ATTR3 | Show an attribute either in the middle compartment or using the composition/aggregation relationship, but not both. |
| Attributes 1 (aggregation) Attributes 2 (composition)  Attributes 3 | ATTR4 | Use aggregation rather than composition (stronger). |
| Attributes 1 (aggregation) Attributes 2 (composition) | ATTR5 | Use composition (stronger) rather than aggregation. |
| Attributes 1 (aggregation) | ATTR6 | Attributes are part of the enclosing class, not a subclass of the enclosing class. |
| Attributes 1 (aggregation) | ATTR7 | Missing attribute name and visibility on aggregation. |
| Attributes 2 (composition) | ATTR8 | Missing attribute name and visibility on composition. |
| Attributes 2 (composition) Attributes 3 | ATTR9 | Dependency is backwards: when a class contains an attribute, it depends on the attribute, not vice-versa. |
|  | CLAS1 | Attributes marked ‘static’ in the code are class attributes and must be underlined. Only these are underlined. |
|  | CLAS2 | Methods marked ‘static’ in the code are class methods and must be underlined. Only these are underlined. |
| Compartments & Visibility 1 Compartments & Visibility 3 Completeness 1 | CV1 | Two compartments are reversed. Middle compartment is for attributes and lower compartment is for methods. |
| Compartments & Visibility 1  Compartments & Visibility 2 | CV2 | Visibility: + means public and - means private. |
| Compartments & Visibility 3 | CV3 | All attributes go in the middle compartment, whether they’re private or public. |
|  | CV4 | If a class has methods but not attributes, you must still show an empty middle compartment. |
|  | CV5 | All methods go in the lower compartment, whether they’re private, protected, or public. |
| Generalization/Specialization 1 | GEN1 | Generalization (inheritance) relationship is backwards. |
|  | GEN10 | Wrong relationship: should be a solid line with a hollow triangle (not aggregation/composition). The triangle is at the end of the superclass. |
|  | GEN11 | Invalid relationship: generalization has a hollow triangle, not solid. |
| Generalization/Specialization 1 | GEN2 | Wrong relationship: should be a solid line with a hollow triangle (not dashed line). |
| Generalization/Specialization 1 | GEN3 | Wrong relationship: should be a solid line with a hollow triangle (not a diamond). |
| Generalization/Specialization 2  Interfaces 3 | GEN4 | The dependency is backwards. A subclass depends on its superclass; the superclass does not depend on any subclasses. |
| Generalization/Specialization 2 | GEN5 | The association is backwards. The association should go from the subclass to the superclass. |
| Generalization/Specialization 2 | GEN6 | This is a bi-directional association, but there is no association from a superclass to any of its subclasses. |
|  | GEN7 | While dependency is not incorrect, use the more precise relationship of generalization (inheritance). |
| Generalization/Specialization 3 | GEN7 | We don’t show a class’s inherited methods; show only methods it overrides or additional methods it provides. |
|  | GEN8 | Generalization and realization relationships use a hollow triangle, not a solid triangle. |
| Generalization/Specialization 3 | GEN8 | We don’t show default constructors or destructors (ones with no parameters) . Only show constructors or destructors that have parameters. |
|  | GEN9 | Wrong relationship: should be a solid line with a hollow triangle (not a dashed line with an arrowhead). |
| Interfaces 1 | IFAC1 | An interface cannot have any attributes. The class is abstract. |
|  | IFAC10 | Should say <<interface>> followed by the interface name, e.g. <<interface>> IShape |
|  | IFAC11 | With the stereotype <<interface>>, do not mark the methods as <<abstract>>. They can’t be anything else. |
|  | IFAC12 | This is an interface, so use the stereotype <<interface>>. |
|  | IFAC13 | An interface cannot contain attributes. There should not be an aggregation/composition relationships for an interface. |
| Interfaces 1 | IFAC2 | An interface cannot contain any private or protected methods. |
| Interfaces 1 | IFAC3 | All methods in an interface must be abstract (“pure virtual” in C++). |
| Interfaces 1 | IFAC4 | Our conventions dictate that an interface contains a destructor declared ‘virtual’ and with an empty body. |
| Interfaces 1 | IFAC5 | Our conventions dictate that interfaces do not declare constructors. |
| Interfaces 2 | IFAC6 | The dependency is backwards. The dependency is from an interface to the one it inherits from. |
| Interfaces 2 | IFAC7 | An interface can’t realize another interface. |
| Interfaces 3 | IFAC8 | An abstract class can’t realize an interface. |
| Interfaces 4 | IFAC9 | Dependency is backwards: when a concrete class realizes an interface, the dependency goes from the concrete class to the interface, not vice-versa. |
| Realization 1  Interfaces 4 | REAL1 | Wrong relationship: should be a dashed line with triangle (not a solid line). |
| Realization 2 | REAL2 | An abstract class can’t realize (implement) an interface. Use the generalization relationship. |
|  | REAL3 | Wrong relationship: should be a dashed line with a triangle (not an arrowhead). |
|  | REAL4 | Invalid relationship: realization has a hollow triangle, not solid. |
|  | REAL5 | Realization (implements) relationship is backwards. |
|  | REAL6 | Missing realization relationship. |
| Stereotypes 1 | STER1 | <<interface>> is part of the UML standard. |
| Stereotypes 1 | STER2 | <<create>> is part of the UML standard. |
| Stereotypes 1 | STER3 | <<destroy>> is part of the UML standard. |
| Stereotypes 1 | STER4 | <<abstract>> is part of the UML standard. |
| Stereotypes 1 | STER5 | Stereotypes are an extensibility mechanism for UML class diagrams. |
| Stereotypes 1 | STER6 | Creating your own stereotype is allowed by the UML standard. |
| Stereotypes 1 | STER7 | A stereotype can be applied to a method. |
| Stereotypes 1 | STER8 | A stereotype can be applied to an association. |
| Stereotypes 1 | STER9 | One or more stereotypes can be applied to a class. |
| C++ Virtual Methods 1  C++ Virtual Methods 2  C++ Virtual Methods 3  C++ Virtual Methods 4 | VIRT1 | ‘virtual’ is C++ terminology, not UML terminology, so it doesn’t belong in a class diagram. UML does not distinguish between dynamic method binding (virtual) and static method binding (non-virtual). |
| C++ Virtual Methods 1  C++ Virtual Methods 2  C++ Virtual Methods 3  C++ Virtual Methods 4  C++ Virtual Methods 5  C++ Virtual Methods 6 | VIRT2 | The … =0 syntax means that it’s a pure virtual method, aka an abstract method, so in UML it is shown with the <<abstract>> stereotype. |
| C++ Virtual Methods 5 | VIRT3 | The class follows our conventions for an interface, so it can be shown as an interface in UML. |
| C++ Virtual Methods 5 | VIRT4 | The class follows our conventions for an interface, but in UML we can show it as an abstract class with all abstract methods. |