**Structural patterns: definitions, problem and solution statements**

**Test:**

**1. What is the best definition of Structural patterns?**

A. This type of patterns provides a way to create interfaces and define ways to compose objects to obtain new functionalities.

B. This type of patterns provides an ability to hide creation logic instead of using new operator.

C. This type of patterns is specifically concerned with communication between objects.

D. This type of patterns is specifically concerned with the presentation tier.

**2. What is related to Structural patterns? Multiple options available.**

A. Structural patterns are concerned with how objects and classes communicate with each other.

B. Structural patterns hide how instances of classes are created and put together.

C. Structural patterns describe ways to compose objects to realize new functionality.

D. Structural patterns show ways how classes and objects are composed to form larger structures.

E. Structural patterns encapsulate knowledge about which concrete classes the system uses.

F. Structural patterns describe how different objects work together to accomplish a task.

**3. What from the following code issues can be solved with Structural Patterns? Multiple answers possible.**

A. Algorithm implementation contains too many special case logic and conditional statements.

B. Different methods in subclass do semantically similar steps except for creating objects.

C. Creation code is duplicated in different methods.

D. Class contains hard-coded logic to notify other classes.

E. Creation logic is sprawled among many classes.

F. Classes implement the same of similar steps and have different interface which make client code complicated, since it has to work with both interfaces.

G. Class has new responsibility which is additional to its base responsibility which makes the class very big.

**4. Which Design Pattern should you use when you want to avoid a permanent binding between an abstraction and its implementation. This might be the case, for example, when the implementation must be selected or switched at run-time.**

A. Adapter.

B. Composite.

C. Bridge.

D. Decorator.

**5. Which Design Pattern should you use when you want to represent part-whole hierarchies of objects.**

A. Adapter.

B. Composite.

C. Bridge.

D. Decorator.

**6. Which Design Pattern should you use when you want to provide a simple interface to a complex subsystem.**

A. Adapter.

B. Façade.

C. Decorator.

D. Composite.

**7. Which Design Pattern should you use when an application uses a large number of objects and the storage costs are high because of the sheer quantity of objects.**

A. Adapter.

B. Façade.

C. Flyweight.

D. Decorator.

**8. Which Design Pattern should you use when you want to add responsibilities to individual objects dynamically and transparently, that is, without affecting other objects.**

A. Adapter.

B. Façade.

C. Decorator.

D. Composite.