Question **1**
Not yet answered
Marked out of 1.50

Which of the following design patterns are used to dynamically add/change functionality at run-time:

Select one or more:

- ☐ Composite Pattern
- ☑ Decorator Pattern
- ☐ Abstract Factory Pattern
- ☐ Observer Pattern
- ☐ Builder Pattern
- ☐ Template Pattern

Question **2**
Not yet answered
Marked out of 1.50

Which of the following statements is/are true?

Select one or more:

- ☐ The adapter class maps/joins functionality of two different types/interfaces and offers additional functionality.
- ☐ Decorative design patterns do not satisfy Open-Closed Principle.
- ☑ Tree structures are normally used to represent Composite Patterns.
- ☐ Graph structures are normally used to represent Builder Patterns.

Question **3**
Not yet answered
Marked out of 1.50

For the Template Pattern, which of the following statements is/are true?

Select one or more:

- ☐ Template Method lets subclasses redefine an algorithm, keeping certain steps invariants.
- ☑ Subclasses of the Template Method can redefine only certain parts of a behaviour without changing the algorithm's structure.
- ☐ A subclass calls the operations of a parent class and not the other way around.
- ☐ Template pattern works on the object level, letting you switch behaviours at runtime

Question **4**
Not yet answered
Marked out of 1.50

Which of the following statements is/are true?

Select one or more:

- ☐ In Java, errors (like OutOfMemoryError, VirtualMachineError, etc.) are Checked Exceptions.
- ☐ The Java IO makes use of the strategy pattern.
- ☐ Pre-conditions in an inherited overridden method must be stronger.
- [✓ All other choices are incorrect.]

Question **5**
Not yet answered
Marked out of 1.00

The Factory Method design pattern solves problems like:

Select one or more:

- [✓ How can an object be created so that subclasses can redefine which class to instantiate?]
- ☐ How can a class defer instantiation to its superclass?
- ☐ How can the way an object is created be changed at run-time?
- ☐ How can object creation that is distributed across multiple classes be centralized?

Question **6**
Not yet answered
Marked out of 1.50

An online camping store, sells different kinds of camping equipment. Items selected by the customer are added to a shopping cart. If an item is not available, a user can request an email notification when that item is available. Which of the following patterns would be useful to design this scenario? Select the most suitable pattern.

Select one:

- ○ Strategy Pattern
- ○ Decorator Pattern
- ○ Template Pattern
- [✓ Visitor Pattern]
- ○ Observer Pattern
- ○ Builder Pattern

Clear my choice

Question **7**
Not yet answered
Marked out of 1.50

In the composite pattern, not placing child-related operations in the component interface does what?

Select one:

- [✓ Prioritises safety over uniformity]
- ○ Prioritises uniformity over safety
- ○ Prioritises polymorphism over uniformity
- ○ Prioritises efficiency over safety

Clear my choice

Question **8**
Not yet answered
Marked out of 1.50

Which of the following statements is/are correct?

Select one or more:

- ☐ Encapsulate what does not vary is a key design principle.
- ☐ Polymorphism requires multiple inheritance.
- ☐ Favour inheritance over composition is a key design principle.
- [✓] A subclass can offer more behaviour than its super class.

Question **9**
Not yet answered
Marked out of 1.50

Suppose the following classes/interfaces are defined:

```
public interface Car {...}
public class SportsCar implements Car {...}
public interface FamilyCar extends Car {...}
public abstract class CityCar implements FamilyCar {...}
```

Which of the following instantiations/statements are valid?

Select one or more:

- ☐
```
FamilyCar c = new Car(...);
```
- ☐
```
FamilyCar c = new SportsCar(...);
```
- ☐
```
FamilyCar c = new CityCar(...);
```
- [✓] None of the other three choices are correct.

Question **10**
Not yet answered
Marked out of 1.50

For Liskov Substitution Principle (LSP), which of the following is correct?

Select one:

- [●] LSP means subtypes must be substitutable for their base types.
- ○ LSP is only applicable for generic types.
- ○ LSP means super types must be substitutable by their subtypes.
- ○ How can a class create different representations of a complex object using the same construction code?

Clear my choice

Question **11**
Not yet answered
Marked out of 1.50

For generic types in Java, which of the following is/are *incorrect*?

Select one or more:

- [✓] List<Integer> is a subtype of List<Object>.
- ☐ List<?> matches List<Object> and List<Integer>.
- ☐ The wildcard < ? extends Foo > matches Foo and any subtype of Foo, where Foo is any type.
- [✓] The wildcard < ? extends Foo > matches Foo and any super type of Foo, where Foo is any type.