Interview Questions:
SOLID Design Principles and Design Patterns

Q1	**Define the SOLID design principles.**
Reference	https://www.zeolearn.com/interview-questions/design-patterns
Ans	**SOLID** is an acronym that is used for the following design principles: • **S:** The **Single responsibility principle** states that each module or class should be responsible for a single piece of functionality, and that this functionality should be completely wrapped by the same class or module. • **O: The Open closed principle** states that software entities should be open for extension and closed for modification. • **L: The Liskov substitution principle** asserts that functions that employ pointers or references to base classes must be able to use objects from the derived classes, without being aware of it. • **I:** The **Interface segregation principle** states that no client should be forced to depend on methods that it does not use. • **D:** According to the **Dependency inversion principle**, high-level modules should not rely on low-level modules and abstractions should be used in both cases. Details should not be relied upon in abstracts. Instead, abstractions should determine the details.

Q2	**What is the difference between abstraction and encapsulation in Java?**
Reference	https://www.java67.com/2012/09/top-10-java-design-pattern-interview-question-answer.html#ixzz6zd189fAS
Ans	Despite the fact that both abstraction and encapsulation hide complexity and simplify the external interface, there is a slight distinction between them. Physical complexity is hidden by abstraction, while logical complexity is hidden by encapsulation.

Q3	**How are design principles different from design patterns?**
Reference	https://www.qfles.com/interview-question/solid-principles-interview-questions

Ans	• The fundamental guidelines that must be followed while building any software system on any platform and in any programming language are known as design principles. Consider the following scenario:
	  ○ Abstraction-based rather than concrete-based dependency
	  ○ Variable encapsulation
	  ○ Interfaces, not implementations, are the focus of this programme.
	• Design patterns are solutions to generic problems that arise frequently. They are not, however, the identical programmes that can be utilised to solve your problem. Instead, you must modify them to fit your requirements. Pre-existing solutions that have been thoroughly studied and confirmed to be safe to use are known as design patterns. Consider the following scenario:
	  ○ When you only want one instance of a class, use the single design pattern.
	  ○ Design template for a repository: To distinguish between the various layers of the application (Business repository, Data repository)

Q4	**Explain the Liskov substitution principle (LSP).**
Reference	https://www.qfles.com/interview-question/solid-principles-interview-questions
Ans	According to the Liskov substitution principle (LSP), objects in a program can be substituted by instances of their subtypes without affecting the program's correctness. To put it in another way, if A is a subtype of B, instances of B can be replaced with instances of A without affecting the program's validity.

Q5	**Explain the Interface segregation principle (ISP).**
Reference	https://www.qfles.com/interview-question/solid-principles-interview-questions
Ans	The Interface segregation principle (ISP) asserts that instead of a single general-purpose interface, numerous client-specific interfaces should be used. In other words, no client should be forced to use techniques that it does not require. It means that creating a different interface and allowing your classes to implement several interfaces is preferable.

Q6	**Elaborate on the Java design patterns.**
Ans	Design patterns describe the best practises used by experienced object-oriented software programmers. They are solutions to typical problems that software developers face while writing a code.

Q7	**Which design pattern is used to sequentially access the elements of a collection object?**
Reference	https://www.guru99.com/java-design-patterns-interview-questions.html
Ans	The iterator pattern is used to get a way to access the elements of a collection object in a sequential manner.

Q8	**What are antipatterns?**
Reference	https://www.zeolearn.com/interview-questions/design-patterns
Ans	Antipatterns are the polar opposites of design patterns. They add a solution to an issue that occurs frequently and has negative repercussions. This might occur as a result of a developer not understanding how to apply a design pattern properly or applying it in the incorrect context. They outline a strategy to reverse the underlying causes and apply product solutions in place.

Q9	**Which pattern is used when we need to decouple an abstraction from its implementation?**
Reference	https://www.guru99.com/java-design-patterns-interview-questions.html
Ans	The bridge pattern is used to isolate an abstraction from its implementation, so that the two can alter independently.

Q10	How can you create a singleton class in Java?
Reference	https://www.guru99.com/java-design-patterns-interview-questions.html
Ans	Creating a singleton class in Java involves a two step process: • Make the constructor private to prevent the new operator from being used to instantiate the class. • If the object is not null, return an object of the object. Otherwise, construct the object and return it via a method.

Q11	How does a static and a singleton class differ?
Reference	https://www.zeolearn.com/interview-questions/design-patterns
Ans	The differences between a static class and the class that implements the singleton pattern are as follows: • The static class's members must all be static. The singleton class, on the other hand, does not have such a requirement. • Static classes are initialised when they are loaded by the class loader, hence they cannot be initialised lazily. Singleton classes, on the other hand, can be loaded slowly. • The objects of static classes are stored on stack , while the objects of singleton class are stored in heap memory space. • A static class cannot be a top-level class and it cannot implement interfaces, whereas a singleton class can.

Q12	**Can you write a thread-safe singleton in Java?**
Reference	https://www.java67.com/2012/09/top-10-java-design-pattern-interview-question-answer.html#ixzz6zcyBnBqn
Ans	In Java, there are several approaches to construct thread-safe singletons, such as employing double-checked locking or using a static singleton instance that is initialised during class loading. Further, the simplest method to design a thread-safe singleton is to use Java enum. For additional information, see Why Enum singleton is better in Java.

Q13	**Explain the factory pattern in Java. What is the advantage of using a static factory method to create an object?**
Reference	https://www.java67.com/2012/09/top-10-java-design-pattern-interview-question-answer.html#ixzz6zcyu22zo
Ans	The factory pattern is a basic Java design pattern that comes up frequently in Java interviews. By providing static factory methods, the factory pattern is utilised to generate an object. There are numerous benefits to offering factory methods, including caching immutable objects, making it easier to introduce new objects and so on. For further information, see What is Factory pattern in Java and its benefits .

Q14	**Can you create the clone of a singleton object?**
Ans	Yes, the clone of a singleton object can be created.

Q15	**Mention why access to the non-static variable is not allowed from a static method in Java?**
Reference	https://www.guru99.com/java-design-patterns-interview-questions.html
Ans	This is because non-static variables are associated with a specific instance of an object, whereas static is not. You cannot access non-static data from a static context.

Q16	**What is the difference between a decorator and a proxy pattern in Java?**
Reference	https://www.java67.com/2012/09/top-10-java-design-pattern-interview-question-answer.html#ixzz6zczkSpqC
Ans	The fact that both decorator and proxy implement the interface of the object they adorn or encapsulate is yet another tough Java design pattern challenge and trick. As mentioned earlier, numerous Java design patterns can have a similar or identical structure, but their aim differs. The decorator pattern is used to implement functionality on a previously created object, while a proxy pattern is used to control access to an object. Another distinction between the decorator and proxy design patterns is that the decorator does not build an object; rather, it obtains the object in its constructor, whereas proxy does so.

Q17	**What are some design patterns that are used in the JDK library?**
Ans	The following are some of the design patterns found in the JDK library: • Wrapper classes employ the decorator pattern. • Calendar classes employ the singleton pattern (Runtime). • Factory patterns like Integer.valueOf are used in the wrapper classes. • Observer patterns such as swing and awt are used in event handling frameworks.

Q18	**What is the difference between a factory and an abstract factory in Java?**
Reference	https://www.java67.com/2012/09/top-10-java-design-pattern-interview-question-answer.html#ixzz6zd0DImYJ
Ans	The fundamental distinction is that the abstract factory pattern constructs a factory, whereas the factory pattern constructs objects. So, they both abstract the creation logic, but one is for the factory and the other is for the things. To answer this Java design pattern interview question, visit here.

Q19	**What is the builder design pattern in Java? When do you use it?**
Reference	https://www.java67.com/2012/09/top-10-java-design-pattern-interview-question-answer.html#ixzz6zd0NBdsD
Ans	The builder pattern in Java is a creational design pattern that is frequently questioned in Java interviews because of its specialised use while building an object with numerous properties, some of which are optional and some of which are required. For further information, see When to use Builder pattern in Java.

Q20	**Explain the advantages of the chain of responsibilities pattern and list the instances in which it is used.**
Ans	The advantages of the chain of responsibility pattern are as follows: • It minimises coupling. • It provides flexibility while assigning the responsibilities to objects. • It allows a group of classes to function as if they were one. Composition allows events generated in one class to be transmitted to other handler classes. **Usage of the chain of responsibility pattern** It is used in the following cases: • When there are many objects prepared to process a request and the handler is unclear. • If the collection or group of objects that can handle the request needs to be supplied dynamically.