Chen Wang Assignment 12_15

Q1. What is an abstract class? Why cannot an abstract class be instantiated directly? Can an abstract class have a constructor? If yes, what access modifier is typically used for it?

- An abstract class is a class in java, which cannot be instantiated directly, and designed to be subclassed for a behavior or purpose.
- An abstract class cannot be instantiated directly because it has abstract method(s) which don't have implementation in it. The abstract class design is not completed and so it cannot be instantiated directly.
- An abstract class can have a constructor, and the access modifier can be public, protected, and default. But it should not be private, since abstract class is designed to be subclassed and private constructor or any methods make it impossible or meaningless.

Q2. What types of methods and member variables can an abstract class contain? List at least 4 types. Is it correct to say "an abstract class can only contain abstract methods"? Explain.

- An abstract class can have regular fields, abstract fields, regular methods, and abstract methods. It must contain at least one abstract method.
- Not correct. An abstract class can contain a regular method to provide some common behaviors.

Q3. What are the default modifiers for methods and variables in an interface? Why can't an interface have a constructor?

- Default modifier for variable is "public static final" which make the default variable to be constant.
- Default modifier for method is public.
- An interface cannot have a constructor because constructor is used to help create objects for class, and an interface cannot create object, so it's meaningless. Also an interface is used to define behavior not implementation, so we should not have constructor for an interface.

Q4. Can a class extend multiple classes in Java? Can a class implement multiple interfaces? Explain the syntax for each case.

- A class cannot extend multiple classes in Java. Because if a class extend multiple classes, then there might be diamond issues, which means if the parent classes to be extended extend the same super parent class and override the same method from it, then the child class doesn't know which version to use when call this overridden method.
- But a class can extend multiple interfaces. Because

Q5. Explain the relationship between interfaces: can an interface extend another interface? If yes, what keyword is used? Can an interface extend multiple interfaces?

- an interface extend another interface, and the keyword is "extend"
- an interface extend multiple interfaces as well.

Q6. Compare abstract classes and interfaces from the following three perspectives: (1) Design intent (what it represents), (2) Multiple inheritance support, (3) Constructors and member variables.

- 1) An abstract class is designed for subclasses to share some common behaviors. An interface is designed for subclasses to support some behaviors, while the details of implementation can vary.
- 2) An abstract class doesn’t support multiple inheritance, but interface support.
- 3) An abstract class could contain a constructor and member variables, but an interface shouldn’t have a constructor, but it can have member variables.

Q7. Why does Java not allow multiple inheritance of classes but allows multiple implementation of interfaces? How does this design avoid the Diamond Problem?

- Multiple inheritance in classes could have overridden method conflicts, but interfaces work well bacause it doesn’t have implementation so no conflicts.
- The only conflicts could happen in the interfaces is the default methods. At this time java require the subclass to override to avoid the Diamond Problem

Q8. Explain the difference between Aggregation and Composition. Focus on the difference in object lifetime management and provide one practical example for each.

- Aggregation: the lifetime of the owned object does not depend on the lifetime of the owner
- Composition: the lifetime of the owned object depends on the lifetime of the owner

Q9. In design patterns and SOLID principles, there is an important principle: "Favor composition over inheritance." Explain why composition is generally more flexible than inheritance.

- Because composition gives us more flexibility and fewer long-term problems than inheritance.
- Composition avoids tight coupling, and composition allows behavior to be changed at runtime.

Q10. Describe the three core characteristics of the Singleton design pattern. Why is the Static Inner Class implementation of Singleton thread-safe and efficient? How does this approach achieve lazy initialization?

- In Singleton design pattern, there is only one instance through the project. Others cannot create instance of Singleton class. And the constructor should be private.
- The inner class is not loaded until it’s called. When the JVM loads a class, the JVM ensures that the static field instance is created once, safely, and without race conditions.
- Also it has no synchronized keyword, only creates the instance when needed. It’s fast and efficient.