---------------------------------------2.15

47. Why Java does not allow overriding a static method?

Method overriding is based on dynamic binding at runtime and the static methods are bonded using static binding at compile time. So, we cannot override static methods. The calling of method depends upon the type of object that calls the static method.

48. Is it allowed to override an overloaded method?

Yes, since the overloaded method is a completely different method in the eyes of the compiler. Overriding isn't the same thing at all. The decision as to which method to call is deferred to runtime

49. What is the difference between method overloading and method overriding in Java?

Overriding occurs when the method signature is the same in the superclass and the child class. It is an example of **runtime polymorphism**. Return type must be same in overriding

Overloading occurs when two or more methods in the same class have the same name but different parameters. Is an example of compile-time polymorphism. Return type can be different but you must change the parameters as well.

50. Does Java allow virtual functions?

Yes. **Every non-static method in Java is a virtual function except for final and private methods**. The methods that cannot be used for the polymorphism is not considered as a virtual function. A static function is not considered a virtual function because a static method is bound to the class itself.

51. What is meant by covariant return type in Java?

Covariant return type refers to **return type of an overriding method**. It allows to narrow down return type of an overridden method without any need to cast the type or check the return type. Covariant return type works only for non-primitive return types

52. What is Runtime Polymorphism?

Method overriding

53. Is it possible to achieve Runtime Polymorphism by data members in java?

No. A method is overridden, not the data members, so **runtime polymorphism can't be achieved by data members**.

54. Explain the difference between static and dynamic binding?

Static binding uses Type information for binding while Dynamic binding uses Objects to resolve binding. Overloaded methods are resolved (deciding which method to be called when there are multiple methods with same name) using static binding while overridden methods using dynamic binding.

Method overloading --- static binding.

Method overriding --- dynamic binding.

55. What is Abstraction in Object Oriented programming?

In simple terms, abstraction **“displays” only the relevant attributes of objects and “hides” the unnecessary details.**

In object-oriented programming, abstraction is one of three central principles (along with encapsulation and inheritance). Through the process of abstraction, a programmer hides all but the relevant data about an object in order to reduce complexity and increase efficiency.

56. How is Abstraction different from Encapsulation?

Abstraction is hiding the details and implementation of the code. Encapsulation is hiding the data and controlling the visibility of the code.

57. What is an abstract class in Java?

Abstract class is **a class which contain one or more abstract methods, which has to be implemented by sub classes**. An abstract class can contain no abstract methods also i.e. abstract class may contain concrete methods.

58. Is it allowed to mark a method abstract method without marking the class abstract?

An abstract class is not required to have an abstract method in it. But any class that has an abstract method in it or that does not provide an implementation for any abstract methods declared in its super classes must be declared as an abstract class.

59. Is it allowed to mark a method abstract as well as final?

An abstract method must be overridden to be useful and called but when you make the abstract method final **it cannot be overridden in Java**, hence there would be no way to use that method. Therefore, making an abstract method final in Java will result in a compile-time error.

60. Can we instantiate an abstract class in Java?

Abstract classes cannot be instantiated, but they can be subclassed. When an abstract class is subclassed, the subclass usually provides implementations for all the abstract methods I its parent class. However, if it does not, then the subclass must also be declared abstract.