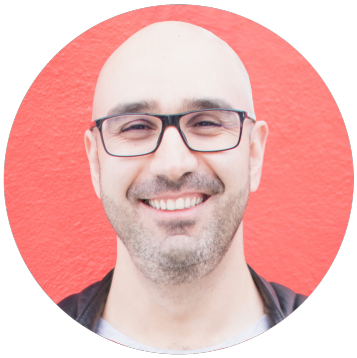
Inheritance

Quiz

Code with Mosh (codewithmosh.com)

1st Edition

About the Author



Hi! My name is Mosh Hamedani. I'm a software engineer with two decades of experience and I've taught over three million people how to code or how to become a professional software engineer. It's my mission to make software engineering simple and accessible to everyone.

https://codewithmosh.com

https://youtube.com/user/programmingwithmosh

https://twitter.com/moshhamedani

https://facebook.com/programmingwithmosh/

Questions

1- How can we have **ClassA** inherit from **ClassB**?

a) class ClassA inherits ClassB

b) class ClassA extends ClassB

c) class ClassA : ClassB

d) class ClassA implements ClassB

2- What will be printed on the console and why?

```
var point1 = new Point(1, 2);

var point2 = new Point(1, 2);

System.out.println(point1.equals(point2));
```

3- What does **hashCode()** method of the **Object** class return?

4- What is a default constructor?

5- How can we add a constructor to the **Customer** class?

a) public Customer(String name) { }

b) public void Customer(String name) {}

c) public Constructor(String name) {}

d) public void Constructor(String name) {}

6- What is the **super** keyword?

7- What is the difference between **private** and **protected** access modifiers?

8- How accessible is a field or method if it's declared without an access modifier?

9- What is method overriding? How is it different from method overloading?

10- What is the benefit of applying the **@Override** annotation when overriding a method?

11- The **Customer** class inherits from the **User** class. Can we pass a **Customer** object to this method? Why?

public void print(User user) {}

12- What is the usage of the **instanceof** operator?

13- What are the four principles of object-oriented programming?

14- When do we use abstract classes?

15- Can we have an abstract class without any abstract methods?

16- When do we use final classes?

17- What is the diamond problem?

18- Does Java support multiple inheritance?

Answers

1- B

2- False - even though point1 and point2 have the same coordinates, the default implementation of the **equals()** method compares objects for reference equality. These two objects are in two different locations in memory, that's why the **equals()** method returns false.

3- The **hashCode()** methods returns a numeric value that is calculated based on the address of the object in memory.

4- A constructor without any parameters. If we don't create it, the Java compiler will automatically add one to our classes.

5- A - constructors don't have a return type, not even void.

6- The **super** keyword is a reference to the base or parent class. We can use it to access the members (fields and methods) or call the constructors of the base class. In contrast, the **this** keyword returns a reference to the current object.

7- Members marked with **protected** or **private** access modifiers are only accessible inside of a class. Protected members are inherited and are accessible by child (derived) classes. Private members are not.

8- If we omit the access modifier, the member will have the default access modifier which makes that member public in package. In other words, that member will be public in the package but private outside of the package.

9- Method overriding means changing the implementation of an inherited method in a subclass. For example, we can override the **equals()** or **hashCode()** methods of the **Object** class. Method overloading means declaring a method with different signatures (different number, type and order of parameters).

10- It signals the Java compiler that we're overriding a method in the base class and this helps the compiler check our code for correctness. It will ensure the signature of the method in the subclass matches the on declared in the base class. Also, if we remove this method from the base class, the compiler will let us know and we can remove the method in the subclass as well.

11- Yes. We can pass an instance of any classes that inherit directly or indirectly from the **User** class. In this case, the customer object will get automatically upcast (meaning it'll get converted to its base type - User). If we need to work with members of the customer object in this method, we need to explicitly downcast it by prefixing the object with (Customer).

12- It tells us if an object is an instance of a class. We use it before casting an object to a different type to make sure we don't get a casting exception.

13- The four principles of object-oriented programming are:

- **Encapsulation**: bundling the data and operations on the data inside a single unit (class).
- **Abstraction**: reducing complexity by hiding unnecessary details (metaphor: the implementation detail of a remote control is hidden from us. We only work with its public interface.)
- **Inheritance**: a mechanism for reusing code.
- **Polymorphism**: a mechanism that allows an object to take many forms and behave differently. This will help us build extensible applications.

14- An abstract class is a partially-implemented (half-cooked) class. We cannot instantiate them. But we use them to share some common code across their subclasses.

15- Yes! An abstract class does not need abstract methods. But if we mark a method as abstract, we should mark the class as abstract as well.

16- Final classes cannot be inherited. We use them when we've made certain assumptions about a class and we want to prevent other classes extending our class and break those assumptions.

17- The diamond problem happens in languages that support multiple inheritance. If two classes (B, C) derive from A and are also the parents

of another class (D), we see a diamond. If the top class (A) declares a method (eg toString) and its children (B and C) override this method, it's not clear which implementation will be inherited by D.

18- No.