Emre KARAGÖZ

ASSIGNMENT 7
JAVA INHERITANCE
(HW 7)

Q1-) Option C

2 lines of code prevent this code from compiling. The first line is declaration of "name" variable. It is marked as private so, it is not accessible from Movie class. Furthermore, constructor of Movie class is missing "super()" statement. A call to a constructor in the direct parent class, using "super()" statement is needed. Since parent's constructor takes an argument, the super constructor would also takes an argument. **Option C** is the answer.

Q2-) Option D

In Java, all abstract interface methods are implicitly public even if it is not defined as "public". **Option D** the correct answer. Abstract methods does not have a body, but static methods must have so Option B is incorrect. "final" modifier cannot be applied to an abstract interface method because it prevents it from being implemented and this conflicts purpose of existence of abstract interface method. This is the reason why Option C is incorrect.

Q3-) Option C

The code does not compile because there are two methods with same method signature in this class. Java does not allow this. Besides, method "`int playMusic`" must return a value, this prevents the code from compiling. **Option C** is the answer.

Q4-) Option A

Inheritance allows objects to access commonly used attributes and methods. This is the main benefit of inheritance, it improves reusability. **Option A** is the answer. Inheritance does not guarantee to make code simpler; Option B is false. Primitives does not inherit methods, so Option C is incorrect. Due to writing methods that reference themselves is not about inheritance, Option D is also inheritance.

Q5-) Option A

```
package mammal;
interface Pet {}
public class Canine implements Pet {
     public__________ getDoggy() {
     return this;
     }
}
```

“this” refers an instance of the class Canine so, this class can return itself or any superclass of it. “Object”, “Pet” and, “Canine” are proper return types. **Option A** is the answer.

Q6-) Option B

Even if teams do not complete their code, the teams can develop the code that implements an ***“interface”*** simultaneously. An interface allows easy integration once the other team’s code is completed.

Q7-) Option B

The code compiles without an issue and prints “Driving electric car” because “car.drive()” calls method “drive()” from class “ElectricCar” owing to polymorphism.

Q8-) Option D

Java allows multiple inheritance by allowing classes to implement any number of interfaces. **Option D** is the answer.

Q9-) Option C

“final” keyword prevents this code from compiling. In order to implement method “watch()” in “LCD” class, “final” keyword must be removed. Furthermore, return types “void” and Objects conflicts each other. This causes a compilation error. Besides, “watch()” method in child class cannot have an access modifier narrower than it’s parent class. “protected” is broader than “package-private” and this is not allowed in Java. These three issue must be fixed to compile this code. **Option C** is the answer.

Q10-) Option C

- The method in the child class must have the same signature as the method in the parent class.
- The method in the child class must be at least as accessible or more accessible than the method in the parent class.
- The method in the child class may not throw a checked exception that is new or broader than the class of any exception thrown in the parent class method.
- If the method returns a value, it must be the same or a subclass of the method in the parent class, known as covariant return types.

Options A and B are correct because as explained in Q09 while overriding a method, the return types must be covariant and access modifier of the method in the child class must be the same or broader than the method in the superclass. Option D is also incorrect, the child class may not throw a checked exception that is new or broader than the class of any exception thrown in the parent class method. **Option C** is the answer because a checked exception thrown by the method in the parent class doesn't have be thrown by the method in the child class.

Q11-) Option C

This code does not compile. "process()" cannot override "process()" in "machines.Computer" because overridden method is final. **Option C** is the answer. If we remove "final" the code compiles and prints "3".

Q12-) Option A

The code compiles without an issue and prints "2" because "`school.getNumberOfStudentsPerClassroom()`"calls method from class "HighSchool" owing to polymorphism. Since "`FileNotFoundException`" is a subclass of "`IOException`" this is legal and does not cause an error. **Option A** is the answer. (https://docs.oracle.com/javase/8/docs/api/java/io/FileNotFoundException.html) .

Q13-) Option B

Java 8 support for static methods within interfaces. These methods are defined explicitly with the "static" keyword and function nearly identically to static methods defined in classes. In fact, there is really only one distinction between a static method in a class and an interface. A static method defined in an interface is not inherited in any classes that implement the interface.

1. Like all methods in an interface, a static method is assumed to be public and will not compile if marked as private or protected.

2. To reference the static method, a reference to the name of the interface must be used.

Q14-) – (There is no correct option)

The code compiles but throws an error at runtime because the signature of main method is invalid. If we fix this error by inserting proper parameters to main method, it compiles and prints “Sprinting” by overriding the “walk()” method.

Q15-) Option B

A class can extend another class, this is the essence of inheritance. Furthermore, Java allows multiple inheritance by allowing classes to implement any number of interfaces and an interface can extend another interface. So, options D, C and A are true but Option C is false. An interface cannot implement another interface. **Option B** is the answer.

Q16-) Option D

The code does not compile because "`private int height`" is not accessible from class “Rocket”. **Option D** is the answer. If we make variable "`height`” accesible by making it protected the code compiles without an issue outputs “3,5”.

Q17-) Option D

Excluding default and static methods, an ***“abstract class”*** can contain both abstract and concrete methods, while an ***“interface”*** contains only abstract methods. **Option D** is the answer.

Q18-) Option C

Abstract classes cannot be instantiated, only concrete classes can be. The line g3 attempts to instantiate class "`IsoscelesRightTriangle`" which is abstract is causes an error and prevents the code from compiling. **Option C** is the answer.

Q19-) Option D

The code does not compile regardless of the return type of “Saxoshone.play()” because return types of play methods in interface “Horn” and class “Woodwind” are incompatible with each other. **Option D** is the answer.

Q20-) Option C

A class “implements” an interface, while a class “extends” an abstract class. **Option C** is the answer.

Q21-) Option A

Static member “paper.Book.material” which is “papyrus” accessed via instance reference “super.material” by method “getMaterial” and the code prints it. **Option A** is the answer.

Q22-) Option B

Option D is true because "`unknownBunny`" could be an interface, a class, or an abstract class. Options A and C are also true and **Option B** is the answer. "`unknownBunny`" can be "`Bunny`" or a subclass of "`Bunny`" but it do not have to be. In order to access to the same variables, "`unknownBunny`" can be explicitly cast to these types.

Q23-) Option D

“final” and “private” access modifiers cannot be applied to an abstract method because this prevents these methods from overriding by concrete subclasses. Furthermore, “default” keyword also cannot be applied. It only be applied to concrete interfaces. Option D (“protected”) is the answer. Access modifiers “public”, “package-private” and “protected” modifiers can be applied to abstract methods.

Q24-) Option D

The code does not compile due to the declaration of class “Mars”. **Option D** is the answer. As explain in Q20, a class “implements” an interface, while a class “extends” an abstract class. Since “Sphere” is an interface it must be implemented. At the same time, “Planet” is a class and it must be extended. Swapping “Sphere” and “Planet” fixes the error and this code compiles.

Q25-) Option B

Option A is false because a reference to a class cannot be assigned to a subclass reference without an explicit cast. an explicit cast is needed. But the opposite is true. A reference to a class can be assigned to a superclass reference without an explicit cast so, Option B is the answer. Option C is false because a reference to an interface must explicitly cast be assigned to a reference of a class. Moreover, reference to a class that implements an interface can be assigned to an interface reference without an explicit cast. This is why Option D is also false.

Q26-) Option B

- Interface variables are assumed to be public, static, and final. Therefore, marking a variable as private or protected will trigger a compiler error, as will marking any variable as abstract.
- The value of an interface variable must be set when it is declared since it is marked as final.

Variables cannot be declared as abstract in interfaces, this is invalid. **Option B** is the answer.

Q27-) Option C

1. If there is a superclass, initialize it first.
2. Static variable declarations and static initializers in the order they appear in the file.
3. Instance variable declarations and instance initializers in the order they appear in the file.
4. The constructor.

This is the order of initialization. According to these rules:

The static initialization block is called and "1" is printed firstly. After that, class car which is the superlass is initialized and instance initialization block is executed and prints "2". Then the constructor is executed and it outputs "3". Finally, "BlueCar" class is loaded as follow the rules. Instance initialization block is executed and prints "4" and than "5" is printed. "13245" is the output of this code. **Option C** is the answer.

Q28-) Option C

Overloading a method and overriding a method are similar in that they both involve redefining a method using the "***same name"***. They differ in that an overloaded method will use a different signature than an overridden method. **Option C** is the answer.

Q29-) Option A

The code compiles without an issue, **Option A** is the answer. Class "SoccerBall" can be cast to class "Ball" and interface "Equipment" because it extends "Ball" and implements "Equipment". "`(SoccerBall)equipment).size`" equals to 5 and it is the ouput.

Q30-) Option C

A class that defines an instance variable with the same name as a variable in the parent class is referred to as ***"hiding"*** a variable, while a class that defines a static method with the same signature as a static method in a parent class is referred to as ***"hiding"*** a method. **Option C** is the answer.

Q31-) Option B

The code does not compile due to line "x2". Instance method "getEqualSides()"' in "shapes.Square" cannot override static method "getEqualSides()" in "shapes.Rectangle". **Option B** is the answer.

Q32-) Option C

The code does not compile. Method "Rotorcraft.fly()" is abstract but "Rotorcraft" cannot implement an abstract method because only abstract classes and interfaces implements abstract methods. Even if we marked "Rotorcraft" as abstract, the code does not compiles because the abstract 'Rotorcraft' cannot be instantiated.

Q33-) Option B

Here are some basic rules to keep in mind when casting variables:

1. Casting an object from a subclass to a superclass doesn't require an explicit cast.
2. Casting an object from a superclass to a subclass requires an explicit cast.
3. The compiler will not allow casts to unrelated types.
4. Even when the code compiles without issue, an exception may be thrown at runtime if the object being cast is not actually an instance of that class.

Therefore, a class may be assigned to a ***"superclass"*** reference variable automatically but requires an explicit cast when assigned to a ***"subclass"*** reference variable. **Option B** is the answer.

Q34-) Option C

A ***"concrete class"*** is is the first non-abstract subclass that is required to implement all of the inherited abstract methods. An abstract class becomes useful when it is extended by a concrete subclass. **Option C** is the answer.

Q35-) Option D

The code does not compile because of three errors. "public void fly()" cannot have body because interface abstract methods cannot have body. This is the first error. Moreover, a return statement is needed at "public int fly". This is the second error. Finally, since final classes cannot be extended, class "Eagle" cannot be inherited from "final class Bird". **Option D** is the answer.

Q36-) Option B

Both abstract classes and interfaces can contain static methods, abstract methods and static variables. An interface can also include default methods, but an abstract class cannot. **Option B** is the answer.

Q37-) Option C

The code does not compile. **Option C** is the answer. Java allows multiple inheritance via interfaces but the signatures of the two interface methods and the method defined in the class must be compatible to define a class that fulfills both interfaces simultaneously. A method "talk()" that has a same signature with interfaces' "talk()" method must be inserted to "Performance" class in order to compile this code.

Q38-) Option A

The most important feature of polymorphism—and one of the primary reasons we have class structure at all— is to support virtual methods. A ***virtual method*** is a method in which the specific implementation is not determined until runtime. In fact, **all non-final**, **non- static**, and **non-private** Java methods are considered virtual methods, since any of them can be overridden at runtime. **Option A *(protected)*** is the answer.

Q39-) Option B

An interface extends another interface, while a class extends another class. **Option B** is the answer.

Q40-) Option A

The code compiles and outputs "2.0". All variables "secret" are hidden from each-other.

Q41-) Option D

`protected void dance() throws FileNotFoundException {}`
Option A: `void dance() throws IOException`

Visibility of package-private access modifier is narrower than protected, this is invalid.

Option B: `public void dance() throws IOException`

Since `"IOException"` is superclass of `"FileNotFoundException",` this is invalid.

Option C: `private void dance() throws FileNotFoundException`

Visibility of private access modifier is narrower than private, this is invalid.

Option D: `public final void dance()`

This is the answer. Visibility of public access modifier is broder than protected. Moreover, overriden method do not have to throw an exception.

Q42-) Option C

An object which is “Dog” or one of subclass of “Dog” can be passed as an argument to method “setAnimal()”. “null” can be passed but an object which is a “Wolf” can not because it is not a subclass of “Dog”. **Option C** is the answer.

Q43-) Option A

An interface method can be abstract, default or static but it cannot be final. final” and “private” access modifiers cannot be applied to an interface method because it prevents these methods from overriding. **Option A** is the answer.

Q44-) Option A

There is no error which prevents the code from compiling. The code compiles and at runtime prints “Let's start the party!” **Option A** is the answer.

Q45-) Option D

Option D is the answer. None of the other options is proper for the blanks. We can write **“overloaded”** to first blank because overloaded methods must have a different list of parameters but **“overridden”** is not proper for the second blank. If an overridden method returns a value, it must be the same or a subclass of the method in the parent class, known as “***covariant return types”***.

Q46-) Option B

Constructor Definition Rules:

1. The first statement of every constructor is a call to another constructor within the class using this(), or a call to a constructor in the direct parent class using super().

2. The super() call may not be used after the first statement of the constructor.

3. If no super() call is declared in a constructor, Java will insert a no-argument super() as the first statement of the constructor.

4. If the parent doesn’t have a no-argument constructor and the child doesn’t define any constructors, the compiler will throw an error and try to insert a default no-argument constructor into the child class.

5. If the parent doesn’t have a no-argument constructor, the compiler requires an explicit call to a parent constructor in each child constructor.

Option A: If a parent class does not include a no-argument constructor, a child class can declare one explicitly.

Option B: If a parent class does not include a no-argument constructor (nor a default one inserted by the compiler), a child class must contain at least one constructor definition. (Otherwise, it will throw an error.)

Option C: If a parent class contains a no-argument constructor, a child class do not have to contain a no-argument constructor. Java inserts it automatically.

Option D: If a parent class contains a no-argument constructor, a child class do not have to contain any constructor.

Q47-) Option D

1. The type of the object determines which properties exist within the object in memory.

2. The type of the reference to the object determines which methods and variables are accessible to the Java program.

Namely, the ***"object type"*** determines which attributes exist in memory, while the ***"reference type"*** determines which attributes are accessible by the caller." **Option D** is the answer.

Q48-) – (There is no correct option)

Since the method "play" is overridden in interface "MusicCreator" and class "Violin", the return type of it must be ***covariant*** with the return types of methods in these class and interface. Namely it must be the same or a subclass of these return types. If we assume return value is "12L" or "12l", "Long" is a proper return type because it's the same with return type of the method of parent class and it is a subclass of Number which is the return type of the "play()" method in interface. Otherwise, none of these options is true.

Q49-) Option B

The purpose of adding default methods to the Java language was in part to help with code development and "***backward compatibility"***. Imagine you have an interface that is shared among dozens or even hundreds of users that you would like to add a new method to. If you just update the interface with the new method, the implementation would break among all of your subscribers, who would then be forced to update their code. **Option B** is the answer.

Q50-) Option C

- If the superclass method does not declare an exception, subclass overridden method cannot declare the checked exception but it can declare unchecked exception.

- If the superclass method declares an exception, subclass overridden method can declare same, subclass exception or no exception but cannot declare parent exception.

Since "`EOFEcxception`" is the subclass of "`IOException`" the code does not compile. **Option C** is the answer.
(https://docs.oracle.com/javase/8/docs/api/java/io/EOFException.html)