Inheritance

1. How many lines of the following program contain compilation errors?

```
package theater;
class Cinema {
private String name;
public Cinema(String name) {this.name = name;}
}
public class Movie extends Cinema {
public Movie(String movie) {}
public static void main(String[] showing) {
System.out.print(new Movie("Another Trilogy").name);
}
}
```

A. None
B. One
C. Two
D. Three

2. Which modifier can be applied to an abstract interface method?
A. protected
B. static
C. final
D. public

3. What is the output of the following application?

```
package radio;
public class Song {
public void playMusic() {
System.out.print("Play!");
}
private static int playMusic() {
System.out.print("Music!");
}
public static void main(String[] tracks) {
new Song().playMusic();
}
}
```

A. Play!
B. Music!
C. The code does not compile.
D. The code compiles but the answer cannot be determined until runtime.

4. Which of the following statements about inheritance is true?
A. Inheritance allows objects to access commonly used attributes and methods.
B. Inheritance always leads to simpler code.
C. All primitives and objects inherit a set of methods.
D. Inheritance allows you to write methods that reference themselves.

5. Given the class declaration below, which value cannot be inserted into the blank line that would allow the code to compile?

```
package mammal;
interface Pet {}
public class Canine implements Pet {
public getDoggy() {
return this;
}
}
```
A. Class
B. Pet
C. Canine
D. Object

6. Imagine you are working with another team to build an application. You are developing code that uses a class that the other team has not finished writing yet. Which element of Java would best facilitate this development, allowing easy integration once the other team's code is complete?
A. An abstract class
B. An interface
C. static methods
D. An access modifier

7. What is the output of the following application?
```
package vehicles;
class Automobile {
private final String drive() { return "Driving vehicle"; }
}
class Car extends Automobile {
protected String drive() { return "Driving car"; }
}
public class ElectricCar extends Car {
public final String drive() { return "Driving electric car"; }
public static void main(String[] wheels) {
final Car car = new ElectricCar();

System.out.print(car.drive());
}
}
```
A. Driving vehicle
B. Driving electric car
C. Driving car
D. The code does not compile.

8. Which of the following statements about inheritance is correct?
A. Java does not support multiple inheritance.
B. Java allows multiple inheritance using abstract classes.
C. Java allows multiple inheritance using non-abstract classes.
D. Java allows multiple inheritance using interfaces.

9. How many changes need to be made to the classes below to properly override the watch() method?
```
package entertainment;
```

```
class Television {
protected final void watch() {}
}
public class LCD extends Television {
Object watch() {}
}
```
A. One
B. Two
C. Three
D. None; the code compiles as is.

10. Which of the following statements about overriding a method is incorrect?
A. The return types must be covariant.
B. The access modifier of the method in the child class must be the same or broader than the method in the superclass.
C. A checked exception thrown by the method in the parent class must be thrown by the method in the child class.
D. A checked exception thrown by a method in the child class must be the same or narrower than the exception thrown by the method in the parent class.

11. What is the output of the following application?
```
package machines;
class Computer {
protected final int process() { return 5; }

}
public class Laptop extends Computer {
public final int process() { return 3; }
public static void main(String[] chips) {
System.out.print(new Laptop().process());
}
}
```
A. 5
B. 3
C. The code does not compile.
D. The code compiles but throws an exception at runtime.

12. Given that FileNotFoundException is a subclass of IOException, what is the output of the following application?
```
package edu;
import java.io.*;
class School {
public int getNumberOfStudentsPerClassroom(String... students)
throws IOException {
return 3;
}
public int getNumberOfStudentsPerClassroom() throws IOException {
return 9;
}
}
public class HighSchool extends School {
```

```
public int getNumberOfStudentsPerClassroom() throws FileNotFoundException {
return 2;
}
public static void main(String[] students) throws IOException {
School school = new HighSchool();
System.out.print(school.getNumberOfStudentsPerClassroom());
}
}
```
A. 2
B. 3
C. 9
D. The code does not compile.

13. Which modifier can be applied to an interface method?
A. protected
B. static
C. private
D. final

14. What is the output of the following application?
```
package track;
interface Run {
default void walk() {
System.out.print("Walking and running!");
}
}
interface Jog {
default void walk() {
System.out.print("Walking and jogging!");
}
}
public class Sprint implements Run, Jog {
public void walk() {
System.out.print("Sprinting!");
}
public static void main() {
new Sprint().walk();
}
}
```
A. Walking and running!
B. Walking and jogging!
C. Sprinting!
D. The code does not compile.

15. Which of the following statements about interfaces is not true?
A. An interface can extend another interface.
B. An interface can implement another interface.
C. A class can implement two interfaces.
D. A class can extend another class.

16. What is the output of the following application?

```
package transport;
class Ship {
protected int weight = 3;
private int height = 5;
public int getWeight() { return weight; }
public int getHeight() { return height; }
}
public class Rocket extends Ship {
public int weight = 2;
public int height = 4;
public void printDetails() {
System.out.print(super.getWeight()+","+super.height);
}
public static final void main(String[] fuel) {
new Rocket().printDetails();
}
}
```

A. 2,5
B. 3,4
C. 3,5
D. The code does not compile.

17. Fill in the blanks: Excluding default and static methods, a(n) _____ can contain both abstract and concrete methods, while a(n) _____ contains only abstract methods.
A. concrete class, abstract class
B. concrete class, interface
C. interface, abstract class

18. Which statement about the following class is correct?

```
package shapes;
abstract class Triangle {
abstract String getDescription();
}

class RightTriangle extends Triangle {
protected String getDescription() { return "rt"; } // g1
}
public abstract class IsoscelesRightTriangle extends RightTriangle { // g2
public String getDescription() { return "irt"; }
public static void main(String[] edges) {
final Triangle shape = new IsoscelesRightTriangle(); // g3
System.out.print(shape.getDescription());
}
}
```

A. The code does not compile due to line g1.
B. The code does not compile due to line g2.
C. The code does not compile due to line g3.
D. The code compiles and runs without issue.

19. Given that Short and Integer extend Number, what type can be used to fill in the blank in

the class below to allow it to compile?

```
package band;
interface Horn { public Integer play(); }
abstract class Woodwind { public Short play() {return 3;} }
public final class Saxophone extends Woodwind implements Horn {
public play() {
return null;
}
}
```

A. Integer
B. Short
C. Number
D. None of the above

20. Fill in the blanks: A class an interface, while a class an abstract class.
A. extends, implements
B. extends, extends
C. implements, extends
D. implements, implements

21. What is the output of the following application?

```
package paper;
abstract class Book {
protected static String material = "papyrus";
public Book() {}
public Book(String material) {this.material = material;}
}
public class Encyclopedia extends Book {
public static String material = "cellulose";
public Encyclopedia() {super();}
public String getMaterial() {return super.material;}
public static void main(String[] pages) {
System.out.print(new Encyclopedia().getMaterial());
}
}
```

A. papyrus
B. cellulose
C. The code does not compile.
D. The code compiles but throws an exception at runtime.

23. Which of the following modifiers can be applied to an abstract method?
A. final
B. private
C. default
D. protected

24. What is the output of the following application?

```
package space;
interface Sphere {
```

```
default String getName() { return "Unknown"; }
}
abstract class Planet {
abstract String getName();
}
public class Mars extends Sphere implements Planet {
public Mars() {
super();
}
public String getName() { return "Mars"; }
public static void main(final String[] probe) {
System.out.print(((Planet)new Mars()).getName());
}
}
```

A. Mars
B. Unknown
C. The code does not compile due to the declaration of Sphere.
D. The code does not compile for another reason.

25. Which of the following statements is correct?
A. A reference to a class can be assigned to a subclass reference without an explicit cast.
B. A reference to a class can be assigned to a superclass reference without an explicit cast.
C. A reference to an interface can be assigned to a reference of a class that implements the interface without an explicit cast.
D. A reference to a class that implements an interface can be assigned to an interface reference only with an explicit cast.

26. Of the following four modifiers, choose the one that is not implicitly applied to all interface variables.
A. final
B. abstract
C. static
D. public

27. What is the output of the following application?

```
package race;
abstract class Car {
static { System.out.print("1"); }
public Car(String name) {
super();
System.out.print("2");
}
{ System.out.print("3"); }
}
public class BlueCar extends Car {
{ System.out.print("4"); }
public BlueCar() {
super("blue");
System.out.print("5");
}
public static void main(String[] gears) {
```

```
new BlueCar();
}
}
```
A. 23451
B. 12354
C. 13245
D. The code does not compile.

28. Fill in the blank: Overloaded and overridden methods always have .
A. the same parameter list
B. different return types
C. the same method name
D. covariant return types

29. What is the output of the following application?
```
package sports;
abstract class Ball {
protected final int size;
public Ball(int size) {
this.size = size;
}
}

interface Equipment {}
public class SoccerBall extends Ball implements Equipment {
public SoccerBall() {
super(5);
}
public Ball get() { return this; }
public static void main(String[] passes) {
Equipment equipment = (Equipment)(Ball)new SoccerBall().get();
System.out.print(((SoccerBall)equipment).size);
}
}
```
A. 5
B. The code does not compile due an invalid cast.
C. The code does not compile for a different reason.
D. The code compiles but throws a ClassCastException at runtime.

30. Fill in the blanks: A class that defines an instance variable with the same name as a variable in the parent class is referred to as 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 a method.
A. hiding, overriding
B. overriding, hiding
C. hiding, hiding
D. replacing, overriding

Answers

1. C. The code does not compile, so Option A is incorrect. This code does not compile for two reasons. First, the name variable is marked private in the Cinema class, which means it cannot be accessed directly in the Movie class. Next, the Movie class defines a constructor that is missing an explicit super() statement. Since Cinema does not include a no-argument constructor, the no-argument super() cannot be inserted automatically by the compiler without a compilation error. For these two reasons, the code does not compile, and Option C is the correct answer.

2. D. All abstract interface methods are implicitly public, making Option D the correct answer. Option A is incorrect because protected conflicts with the implicit public modifier. Since static methods must have a body and abstract methods cannot have a body, Option B is incorrect. Finally, Option C is incorrect. A method, whether it be in an interface or a class, cannot be declared both final and abstract, as doing so would prevent it from ever being implemented.

3. C. A class cannot contain two methods with the same method signature, even if one is static and the other is not. Therefore, the code does not compile because the two declarations of playMusic() conflict with one another, making Option C the correct answer.

4. A. Inheritance is often about improving code reusability, by allowing subclasses to inherit commonly used attributes and methods from parent classes, making Option A the correct answer. Option B is incorrect. Inheritance can lead to either simpler or more complex code, depending on how well it is structured. Option C is also incorrect. While all objects inherit methods from java.lang.Object, this does not apply to primitives. Finally, Option D is incorrect because methods that reference themselves are not a facet of inheritance.

5. A. Recall that this refers to an instance of the current class. Therefore, any superclass of Canine can be used as a return type of the method, including Canine itself, making Option C an incorrect answer. Option B is also incorrect because Canine implements the Pet interface. An instance of a class can be assigned to any interface reference that it inherits. Option D is incorrect because Object is the superclass of instances in Java. Finally, Option A is the correct answer. Canine cannot be returned as an instance of Class because it does not inherit Class.

6. B. The key here is understanding which of these features of Java allow one developer to build their application around another developer's code, even if that code is not ready yet. For this problem, an interface is the best choice. If the two teams agree on a common interface, one developer can write code that uses the interface, while another developer writes code that implements the interface. Assuming neither team changes the interface, the code can be easily integrated once both teams are done. For these reasons, Option B is the correct answer.

7. B. The drive() method in the Car class does not override the version in the Automobile class since the method is not visible to the Car class. Therefore, the final attribute in the Automobile class does not prevent the Car class from implementing a method with the same signature. The drive() method in the ElectricCar class is a valid override of the method in the Car class, with the access modifier expanding in the subclass. For these reasons, the code compiles, and Option D is incorrect. In the main() method, the object created is an ElectricCar, even if it is assigned to a Car reference. Due to polymorphism,

the method from the ElectricCar will be invoked, making Option B the correct answer.

8. D. While Java does not allow a class to extend more than one class, it does allow a class to implement any number of interfaces. Multiple inheritance is, therefore, only allowed via interfaces, making Option D the correct answer.

9. C. There are three problems with this method override. First, the watch() method is marked final in the Television class. The final modifier would have to be removed from the method definition in the Television class in order for the method to compile in the LCD class. Second, the return types void and Object are not covariant. One of them would have to be changed for the override to be compatible. Finally, the access modifier in the child class must be the same or broader than in the parent class. Since package-private is narrower than protected, the code will not compile. For these reasons, Option C is the correct answer.

10. C. First off, the return types of an overridden method must be covariant. Next, it is true that the access modifier must be the same or broader in the child method. Using a narrower access modifier in the child class would not allow the code to compile. Overridden methods must not throw any new or broader checked exceptions than the method in the superclass. For these reasons, Options A, B, and D are true statements. Option C is the false statement. An overridden
method is not required to throw a checked exception defined in the parent class.

11. C. The process() method is declared final in the Computer class. The Laptop class then attempts to override this method, resulting in a compilation error, making Option C the correct answer.

12. A. The code compiles without issue, so Option D is incorrect. The rule for overriding a method with exceptions is that the subclass cannot throw any new or broader checked exceptions. Since FileNotFoundException is a subclass of IOException, it is considered a narrower exception, and therefore the overridden method is allowed. Due to polymorphism, the overridden version of the method in HighSchool is used, regardless of the reference type, and 2 is printed, making Option A the correct answer. Note that the version of the method that takes the varargs is not used in this application.

13. B. Interface methods are implicitly public, making Option A and C incorrect. Interface methods can also not be declared final, whether they are static, default, or abstract methods, making Option D incorrect. Option B is the correct answer because an interface method can be declared static.

14. C. Having one class implement two interfaces that both define the same default method signature leads to a compiler error, unless the class overrides the default method. In this case, the Sprint class does override the walk() method correctly, therefore the code compiles without issue, and Option C is correct.

15. B. Interfaces can extend other interfaces, making Option A incorrect. On the other hand, an interface cannot implement another interface, making Option B the correct answer. A class can implement any number of interfaces, making Option C incorrect. Finally, a class can extend another class, making Option D incorrect.

16. D. The code does not compile because super.height is not visible in the Rocket class, making Option D the correct answer. Even though the Rocket class defines a height value, the super keyword looks for an inherited version. Since there are none, the code does not

compile. Note that super.getWeight() returns 3 from the variable in the parent class, as polymorphism and overriding does not apply to instance variables.

17. D. An abstract class can contain both abstract and concrete methods, while an interface can only contain abstract methods. With Java 8, interfaces can now have static and default methods, but the question specifically excludes them, making Option D the correct answer. Note that concrete classes cannot contain any abstract methods.

18. C. The code does not compile, so Option D is incorrect. The IsoscelesRightTriangle class is abstract; therefore, it cannot be instantiated on line g3. Only concrete classes can be instantiated, so the code does not compile, and Option C is the correct answer. The rest of the lines of code compile without issue. A concrete class can extend an abstract class, and an abstract class can extend a concrete class. Also, note that the override of getDescription() has a widening access modifier, which is fine per the rules of overriding methods.

19. D. The play() method is overridden in Saxophone for both Horn and Woodwind, so the return type must be covariant with both. Unfortunately, the inherited methods must also be compatible with each other. Since Integer is not a subclass of Short, and vice versa, there is no subclass that can be used to fill in the blank that would allow the code to compile. In other words, the Saxophone class cannot compile regardless of its implementation of play(), making Option D the correct answer.

20. C. A class can implement an interface, not extend it. Alternatively, a class extends an abstract class. Therefore, Option C is the correct answer.

21. A. The code compiles and runs without issue, making Options C and D incorrect. Although super.material and this.material are poor choices in accessing static variables, they are permitted. Since super is used to access the variable in getMaterial(), the value papyrus is returned, making Option A the correct answer. Also, note that the constructor Book(String) is not used in the Encyclopedia class.

22. B. Options A and C are both true statements. Either the unknownBunny reference variable is the same as the object type or it can be explicitly cast to the Bunny object type, therefore giving it access to all its members. This is the key distinction between reference types and object types. Assigning a new reference does not change the underlying object. Option D is also a true statement since any superclass that Bunny extends or interface it implements could be used as the data type for unknownBunny. Option B is the false statement and the correct answer. An object can be assigned to a reference variable type that it inherits, such as Object unknownBunny = new Bunny().

23. D. An abstract method cannot include the final or private method. If a class contained either of these modifiers, then no concrete subclass would ever be able to override them with an implementation. For these reasons, Options A and B are incorrect. Option C is also incorrect because the default keyword applies to concrete interface methods, not abstract methods. Finally, Option D is correct. The protected, package-private, and public access modifiers can each be applied to abstract methods.

24. D. The declaration of Sphere compiles without issue, so Option C is incorrect. The Mars class declaration is invalid because Mars cannot extend Sphere, an interface, nor can Mars implement Planet, a class. In other words, they are reversed. Since the code does not compile, Option D is the correct answer. Note that if Sphere and Planet were swapped in

the Mars class definition, the code would compile and the output would be Mars, making Option A the correct answer.

25. B. A reference to a class can be implicitly assigned to a superclass reference without an explicit class, making Option B the correct answer. Assigning a reference to a subclass, though, requires an explicit cast, making Option A incorrect. Option C is also incorrect because an interface does not inherit from a class. A reference to an interface requires an explicit cast to be assigned to a reference of any class, even one that implements the interface. An interface reference requires an explicit cast to be assigned to a class reference. Finally, Option D is incorrect. An explicit cast is not required to assign a reference to a class that implements an interface to a reference of the interface.

26. B. Interface variables are implicitly public, static, and final. Variables cannot be declared as abstract in interfaces, nor in classes.

27. C. The class is loaded first, with the static initialization block called and 1 is outputted first. When the BlueCar is created in the main() method, the superclass initialization happens first. The instance initialization blocks are executed before the constructor, so 32 is outputted next. Finally, the class is loaded with the instance initialization blocks again being called before the constructor, outputting 45. The result is that 13245 is printed, making Option C the correct answer.

28. C. Overloaded methods share the same name but a different list of parameters and an optionally different return type, while overridden methods share the exact same name, list of parameters, and return type. For both of these, the one commonality is that they share the same method name, making Option C the correct answer.

29. A. Although the casting is a bit much, the object in question is a SoccerBall. Since SoccerBall extends Ball and implements Equipment, it can be explicitly cast to any of those types, so no compilation error occurs. At runtime, the object is passed around and, due to polymorphism, can be read using any of those references since the underlying object is a SoccerBall. In other words, casting it to a different reference variable does not modify the object or cause it to lose its underlying SoccerBall information. Therefore, the code compiles without issue, and Option A is correct.

30. C. Both of these descriptions refer to variable and static method hiding, respectively, making Option C correct. Only instance methods can be overridden, making Options A and B incorrect. Option D is also incorrect because *replacing* is not a real term in this context.