


Chapter 12
Advanced Java Class Design

THE OCP EXAM TOPICS COVERED IN THIS PRACTICE TEST INCLUDE THE FOLLOWING:

- ✓**Advanced Java Class**
- Develop code that uses abstract classes and methods
- Develop code that uses the final keyword
- Create inner classes including static inner class, local class, nested class, and anonymous inner class
- Use enumerated types including methods, and constructors in an enum type
- Develop code that declares, implements and/or extends interfaces and use the @Override annotation
- Create and use Lambda expressions

1. Which of the following is required for all valid lambda expressions?

1. ( )
2. ->
3. {}
4. Parameter data type(s)

2. What is the output of the following application?

```
package holiday;
enum DaysOff {
   Thanksgiving, PresidentsDay, ValentinesDay
}
public class Vacation {
   public static void main(String... unused) {
      final DaysOff input = DaysOff.Thanksgiving;
      switch(input) {
         default:
         case DaysOff.ValentinesDay:
            System.out.print("1");
         case DaysOff.PresidentsDay:
            System.out.print("2");
      }
   }
}
```

1. 1
2. 2
3. 12
4. None of the above

3. Fill in the blanks: A functional interface must contain or inherit _____and may optionally include_____ .

3. exactly one `abstract` method, the `@FunctionalInterface` annotation
4. at least one `static` method, at most one `default` method

4. Which of the following class types cannot be marked `final` or `abstract`?

1. Static nested class
2. Local inner class
3. Anonymous inner class
4. Member inner class

5. Which of the following is a valid lambda expression?

1. `r -> {return 1==2}`
2. `(q) -> true`
3. `(x,y) -> {int test; return test>0;}`
4. `a,b -> true`

6. Which of the following properties of an enum can be marked `abstract`?

1. The enum class definition
2. An enum method
3. An enum value
4. None of the above

7. What is the output of the following application?

```
package world;
public class Matrix {
   private int level = 1;
   class Deep {
      private int level = 2;
      class Deeper {
         private int level = 5;
         public void printReality() {
            System.out.print(level);
            System.out.print(" "+Matrix.Deep.this.level);
            System.out.print(" "+Deep.this.level);
         }
      }
   }
   public static void main(String[] bots) {
      Matrix.Deep.Deeper simulation = new Matrix().new Deep().new D
      simulation.printReality();
   }
}
```

1. `1 1 2`
2. `5 2 2`
3. `5 2 1`
4. The code does not compile.

8. A local inner class can access which type of local variables?

1. `final`
2. `private`
3. effectively final

1. I only
2. I and II
3. III only
4. I and III

9. What is the output of the following application?

```
package finance;

enum Currency {
```

```
   DOLLAR, YEN, EURO
}
abstract class Provider {
   protected Currency c = Currency.EURO;
}
public class Bank extends Provider {
   protected Currency c = Currency.DOLLAR;
   public static void main(String[] pennies) {
      int value = 0;
      switch(new Bank().c) {
         case 0:
            value--; break;
         case 1:
            value++; break;
      }
      System.out.print(value);
   }
}
```

1. `0`
2. `1`
3. The code does not compile.
4. The code compiles but throws an exception at runtime.

10. What statement best describes the notion of effectively final in Java?

1. A local variable that is marked `final`
2. A `static` variable that is marked `final`
3. A local variable that is not marked `final` but whose primitive value or object reference does not change after it is initialized
4. A local variable that is not marked `final` but whose primitive value or object reference does not change after a certain point in the method

11. What is the output of the following application?

```
package race;

interface Drive {
   int SPEED = 5;
   default int getSpeed() { return SPEED; }
}
interface Hover {
   int MAX_SPEED = 5;
   default int getSpeed() { return MAX_SPEED; }
}
public class Car implements Drive, Hover {
   public static void main(String[] gears) {
      class RaceCar extends Car {
         @Override public int getSpeed() { return 10; }
      };
      System.out.print(new RaceCar().getSpeed());
   }
}
```

1. `5`
2. `10`
3. The code does not compile due to the definition of `Racecar`.
4. The code does not compile for some other reason.

12. Fill in the blanks: It is possible to extend an _____but not an _____.

1. interface, abstract class
2. abstract class, enum
3. enum, interface
4. abstract class, interface

13. Which of the following results is not a possible output of this program?

```
package sea;
enum Direction { NORTH, SOUTH, EAST, WEST; };
public class Ship {
   public static void main(String[] compass) {
      System.out.print(Direction.valueOf(compass[0]));
   }
}
```

1. `WEST` is printed.
2. `south` is printed.

3. An `ArrayIndexOutOfBoundsException` is thrown at runtime.
4. An `IllegalArgumentException` is thrown at runtime.

14. Which of the following is not an advantage of using enumerated types in Java?

1. Ensure consistency of data across an application.
2. Offer ability to create new enumerated values at runtime.
3. Provide access to fixed constants whose value does not change during the course of the application.
4. Support cases where a value can only take one of a limited number of options.

15. Given the following enum declaration, how many lines contain compilation errors?

```
package rainbow;
enum Light {}
public enum Color extends Light {
   RED, BLUE, ORANGE, GREEN
   protected Color() {}
}
```

1. None, the code compiles as is.
2. One
3. Two
4. Three

16. Which of the following cannot include a `static` method in its definition?

1. Abstract class
2. Static nested class
3. Interface
4. Local inner class

17. What is the output of the following application?

```
package ai;

interface Pump {
   void pump(double psi);
}
interface Bend extends Pump {
   void bend(double tensileStrength);
}
public class Robot {
   public static final void apply(Bend instruction, double input) {
      instruction.bend(input);
   }
   public static void main(String... future) {
      final Robot r = new Robot();
      r.apply(x -> System.out.print(x+" bent!"), 5);
   }
}
```

1. `5.0 bent!`
2. The code does not compile because `Bend` is not a functional interface.
3. The code does not compile because of line `r1`.
4. None of the above.

18. What is the best reason for applying the `@Override` annotation to a method?

1. It is required to implement an interface method.
2. It is required to override a method.
3. The method will fail to compile if it is not actually overriding another method.
4. There are no good reasons other than as a form of documentation.

19. What is the output of the following application?

```
package space;

public class Bottle {
   public static class Ship {
      private enum Sail {  // w1
         TALL {protected int getHeight() {return 100;}},
         SHORT {protected int getHeight() {return 2;}};
         protected abstract int getHeight();
      }
      public Sail getSail() {
         return Sail.TALL;
      }
   }
   public static void main(String[] stars) {
      Bottle bottle = new Bottle();
      Ship q = bottle.new Ship();  // w2
      System.out.print(q.getSail());
   }
}
```

1. `TALL`
2. The code does not compile because of line `w1`.
3. The code does not compile because of line `w2`.
4. The code compiles but the application does not produce any output at runtime.

20. Which of the following is not a valid lambda expression?

1. `(Integer j, k) -> 5`
2. `(p,q) -> p+q`
3. `(Integer x, Integer y) -> x*y`
4. `(left,right) -> {return "null";}`

21. What is the output of the following application?

```
1:  package fruit;
2:
3:  interface Edible { void eat(); }
4:  public class ApplePicking {
5:     public static void main(String[] food) {
6:        Edible apple = new Edible() {
7:           @Override
8:           void eat() {
9:              System.out.print("Yummy!");
10:          }
11:       }
12:    }
13: }
```

1. The application completes without printing anything.
2. `Yummy!`
3. One line of this application fails to compile.
4. Two lines of this application fail to compile.

22. What is the output of the following application?

```
package forest;

public class Woods {
   static class Tree {}
   public static void main(String[] leaves) {
      int water = 10+5;
      final class Oak extends Tree {  // p1
         public int getWater() {
            return water;  // p2
         }
      }
      System.out.print(new Oak().getWater());
   }
}
```

1. `15`
2. The code does not compile because of line `p1`.
3. The code does not compile because of line `p2`.
4. None of the above

23. Fill in the blanks: _____allow Java to support multiple inheritance, and anonymous inner classes can _____of them.

1. Abstract classes, extend at most one
2. Abstract classes, extend any number
3. Interfaces, implement at most one
4. Interfaces, implement any number

24. What is the output of the following application?

```
package vessel;

class Problem extends Exception {}
abstract class Danger {
   protected abstract void isDanger() throws Problem;
}
public class SeriousDanger extends Danger {
   protected void isDanger() throws Exception {  // m1
      throw new RuntimeException();
   }
   public static void main(String[] will) throws Throwable {  // m2
      new SeriousDanger().isDanger();  // m3
   }
}
```

1. The code does not compile because of line `m1`
2. The code does not compile because of line `m2`.
3. The code does not compile because of line `m3`.
4. The code compiles but throws an exception at runtime.

25. Which of the following is not a true statement about interfaces and abstract classes?

1. Interfaces can only extend other interfaces, while abstract classes can extend both abstract and concrete classes.
2. Unlike abstract classes, interfaces can be marked `final`.
3. Abstract classes offer support for single inheritance, while interfaces offer support for multiple inheritance.
4. All methods and variables in interfaces are `public`, while abstract classes can use various access modifiers for their methods and variables, including `private` in some cases.

26. What is the output of the following application?

```
package weather;

public class Forecast {
   public enum Snow { BLIZZARD, SQUALL, FLURRY }
   public static void main(String[] modelData) {
      System.out.print(Snow.BLIZZARD.ordinal());
      System.out.print(" "+Snow.valueOf("flurry".toUpperCase()).nam
   }
}
```

1. `0 FLURRY`
2. `1 FLURRY`
3. The code does not compile.
4. The code compiles but throws an exception at runtime.

27. Fill in the blank: The primary reason `default` interface methods were added to Java is to support_____ .

1. polymorphism
2. concrete methods in interfaces
3. multiple inheritance
4. backward compatibility

28. What is the output of the following application?

```
package zoo;

public class Penguin {
   private int volume = 1;
```

```
   private class Chick {
      private static int volume = 3;
      void chick() {
         System.out.print("Honk("+Penguin.this.volume+")!");
      }
   }
   public static void main(String... eggs) {
      Penguin pen = new Penguin();
      final Penguin.Chick littleOne = pen.new Chick();
      littleOne.chick();
   }
}
```

1. `Honk(1)!`
2. `Honk(3)!`
3. The code does not compile.
4. The code compiles but the output cannot be determined until runtime.

29. Let's say `Dinosaur` is a class that contains a `public` member inner class called `Pterodactyl`. Given that `dino` is an instance of `Dinosaur`, how would you instantiate a new `Pterodactyl` from within a `static` method, such as `main()`?

1. `new Pterodactyl();`
2. `dino.Pterodactyl();`
3. `Dinosaur.new Pterodactyl();`
4. `dino.new Pterodactyl();`

30. What is the result of compiling the following program?

```
package desert;

interface CanBurrow {
   public abstract void burrow();
}
@FunctionalInterface interface HasHardShell extends CanBurrow {}  /
abstract class Tortoise implements HasHardShell {  // k2
   public abstract int toughness();
}
public class DesertTortoise extends Tortoise {  // k3
   public int toughness() { return 11; }
}
```

1. The code does not compile because of line `k1`.
2. The code does not compile because of line `k2`.
3. The code does not compile because of line `k3`.
4. The code compiles without issue.

31. Which statement(s) about the following `Twins` class are true?

```
package clone;

interface Alex {
   default void write() {}
   static void publish() {}
   void think();
}
interface Michael {
   public default void write() {}
   public static void publish() {}
   public void think();
}

public class Twins implements Alex, Michael {
   @Override public void write() {}
   @Override public static void publish() {}
   @Override public void think() {
      System.out.print("Thinking...");
   }
}
```

1. The class fails to compile because of the `write()` method.
2. The class fails to compile because of the `publish()` method.
3. The class fails to compile because of the `think()` method.

1. I only
2. II only
3. I and II
4. II and III

32. Fill in the blanks: A(n) _____and a(n) _____can define `static` methods.

    1. abstract class, local inner class
    2. anonymous inner class, interface
    3. member inner class, enum
    4. enum, static inner class

33. Which lambda expression can replace the instance of `new BiologyMaterial()` in the `Scientist` class and produce the same results under various inputted values?

```
package university;

@FunctionalInterface interface Study {
   abstract int learn(String subject, int duration);
}

class BiologyMaterial implements Study {
   @Override public int learn(String subject, int duration) {
      if(subject == null)
         return duration;
      else
         return duration+1;
   }
}

public class Scientist {
   public static void main(String[] courses) {
      final Study s = new BiologyMaterial() {};
      System.out.print(s.learn(courses[0], Integer.parseInt(courses
   }
}
```

    1. `(p,q) -> q==null ? p : p+1`
    2. `(c,d) -> {int d=1; return c!=null ? d+1 : d;}`
    3. `(x,y) -> {return x==null ? y : y+1;}`
    4. None of the above

34. Given the following enum declaration, how many lines contain compilation errors?

```
package myth;

public enum Proposition {
   TRUE(-10) { @Override String getNickName() { return "RIGHT"; }},
   FALSE(-10) { public String getNickName() { return "WRONG"; }},
   UNKNOWN(0) { @Override public String getNickName() { return "LOS
   private final int value;
   Proposition(int value) {
      this.value = value;
   }
   public int getValue() {
      return this.value;
   }
   protected abstract String getNickName();
}
```

    1. None. The code compiles as is.
    2. One
    3. Two
    4. Three

35. What is the output of the following application?

```
package math;

interface AddNumbers {
   int add(int x, int y);
   static int subtract(int x, int y) { return x-y; }
   default int multiply(int x, int y) { return x*y; }
}
public class Calculator {
   protected void calculate(AddNumbers add, int a, int b) {
      System.out.print(add.add(a, b));
   }
   public static void main(String[] moreNumbers) {
      final Calculator ti = new Calculator();
      ti.calculate((k,p) -> p+k+1, 2, 5);  // j1
   }
}
```

1. 8
2. The code does not compile because `AddNumbers` is not a functional interface.
3. The code does not compile because of line `j1`.
4. None of the above

36. Given the class declaration below, what expression can be used to fill in the blank to return the `size` variable defined in the `Bottle` class, printing `14` at runtime?

```
package baby;

final public class Bottle {
   final private int size = 14;
   final protected class Insert {
      private final int size = 25;
      public final int getSize() {
         return_____________ ;
      }
   }
   final Insert insert = new Insert();
   final public static void main(String[] feed) {
      System.out.print(new Bottle().insert.getSize());
   }
}
```

1. `Bottle.this.size`
2. `this.size`
3. `this.Bottle.size`
4. The code does not compile, regardless of what is placed in the blank.

37. What is the output of the following application?

```
package ocean;
abstract interface CanSwim {
   public void swim();
}
public class Turtle {
   public static void main(String[] seaweed) {
      int distance = 7;
      CanSwim seaTurtle = {
         @Override
         public void swim() {
            System.out.print(distance);
         }
      };
      seaTurtle.swim();
   }
}
```

1. The application completes without printing anything.
2. 7
3. One line of this application fails to compile.
4. Two lines of this application fail to compile.

38. What is the output of the following application?

```
package present;

interface Toy { String play(); }
public class Gift {
   public static void main(String[] matrix) {
      abstract class Robot {}
      class Transformer extends Robot implements Toy {
         public String name = "GiantRobot";
         public String play() {return "DinosaurRobot";}
      }
      Transformer prime = new Transformer () {
         public String play() {return name;}  // y1
      };
      System.out.print(prime.play()+" "+name);
   }
}
```

1. `GiantRobot GiantRobot`
2. `GiantRobot DinosaurRobot`
3. The code does not compile because of line `y1`.
4. None of the above

39. What is the result of compiling the following program?

```
package ballroom;

class Leader {}
class Follower {}
abstract public class Dancer {
   public Leader getPartner() { return new Leader(); }
   abstract public Leader getPartner(int count);  // u1
}

abstract class SwingDancer extends Dancer {
   public Follower getPartner() {  // u2
      return new Follower();  // u3
   }
}
```

1. The code does not compile because of line `u1`.
2. The code does not compile because of line `u2`.
3. The code does not compile because of line `u3`.
4. The code compiles without issue.

40. What is the output of the following application?

```
package prepare;
public class Ready {
   protected static int first = 2;
   private final short DEFAULT_VALUE = 10;
   private static class GetSet {
      int first = 5;
      static int second = DEFAULT_VALUE;
   }
   private GetSet go = new GetSet();
   public static void main(String[] begin) {
      Ready r = new Ready();
      System.out.print(r.go.first);
      System.out.print(", "+r.go.second);
   }
}
```

1. `2, 5`
2. `5, 10`
3. The code does not compile because of the `GetSet` class body.
4. The code does not compile for another reason.

PREV
Chapter 11 Java Class Design

NEXT
Chapter 13 Generics and Collections