Q1. What will be the output of the following program?

int x = 100; double y = 100.1; boolean b = (x == y); System.out.println(b);

1. true
2. false
3. Compilation fails
4. Runtime exception

Answer: b — false
Explanation: `x` is promoted to `100.0` and compared to `100.1`; they are not equal.

Q2. What will be the output of the following code?

int x = 20; String sup = (x < 15) ? "small" : (x < 22) ? "tiny" : "huge";
System.out.println(sup);

1. small
2. tiny
3. huge
4. Compilation fails

Answer: b — tiny
Explanation: `x < 15` is false but `x < 22` is true, so the result is `"tiny"`.

Q3. What will be the output of this loop?

int x = 0, y = 0; for (int z = 0; z < 5; z++) { if ((++x > 2) && (++y > 2)) { x++; } }
System.out.println(x + " " + y);

1. 5 2
2. 5 3
3. 6 3
4. 6 4

Answer: c — 6 3
Explanation: The condition becomes true three times (z=2,3,4), each time incrementing x twice and y once.

Q4. What is the result of this constructor chain?

class Appette { Appette() { this(10); } Appette(int i) { this(); } public static void main(String[] a) { new Appette(); } }

1. No output
2. Compile-time error
3. Runtime exception
4. StackOverflowError

Answer: b — Compile-time error
Explanation: The two constructors call each other in a cycle, which is illegal.

Q5. You want subclasses in any package to access superclass members. Which modifier do you use?

1. public
2. private
3. protected
4. default (package-private)

Answer: c — protected
Explanation: `protected` permits subclass access across package boundaries.

Q6. Which classes allow reading/writing Java primitives in binary form?

1. DataInputStream & DataOutputStream
2. DataInput & DataOutput
3. InputStream & OutputStream
4. BufferedInput & BufferedOutput

Answer: a — DataInputStream & DataOutputStream
Explanation: They provide methods like `readInt()` and `writeDouble()`.

Q7. What does this try-finally print?

try { return; } finally { System.out.println("Finally"); }

1. Finally
2. Compilation fails
3. No output
4. Runtime exception

Answer: a — Finally
Explanation: The `finally` block always executes before the method returns.

Q8. Evaluate the bitwise operations:

int x = 11 & 9; // 1011 & 1001 = 1001 (9) int y = x ^ 3; // 1001 ^ 0011 = 1010 (10)
System.out.println(y | 12); // 1010 | 1100 = 1110 (14)

1. 0
2. 7
3. 8
4. 14

Answer: d — 14
Explanation: Final OR yields binary 1110, which is 14.

Q9. Which statement is FALSE about Java serialization?

1. A class must implement `Serializable` to be serialized.
2. Fields marked `transient` aren't serialized.
3. `IOException` may occur during serialization.
4. Serialization is platform-dependent.

Answer: d — Serialization is platform-dependent
Explanation: Java's serialization is designed to be platform-independent.

Q10. What does `Thread.join()` do?

1. Makes one thread wait for another to finish.
2. Interrupts another thread.
3. Suspends the current thread indefinitely.
4. Puts the thread to sleep.

Answer: a — Makes one thread wait for another
Explanation: `join()` causes the caller to pause until the target thread completes.

Q11. Which collection supports a dynamically resizable array?

1. Vector
2. ArrayList
3. Arrays
4. HashMap

Answer: b — ArrayList
Explanation: `ArrayList` automatically grows as elements are added.

Q12. Given these declarations, which assignment is INVALID?

interface GrandParent {} class Parent implements GrandParent {} class Baby extends Parent {}

1. `GrandParent g = new Parent();`
2. `GrandParent g = new Baby();`
3. `Parent p = new GrandParent();`
4. `Parent p = new Baby();`

Answer: c — `Parent p = new GrandParent();`
Explanation: You cannot instantiate an interface type.

Q13. What is the default priority of a new Java thread?

1. MIN_PRIORITY
2. MAX_PRIORITY
3. NORM_PRIORITY
4. Cannot be determined

Answer: c — NORM_PRIORITY
Explanation: By default, threads run at `Thread.NORM_PRIORITY` (value 5).

Q14. Which method must a class implementing `Runnable` define?

1. `public void run() throws Exception`
2. `public void run()`
3. `public void run(Runnable r)`
4. `public void run(Runnable r, String s)`

Answer: b — `public void run()`
Explanation: `Runnable` declares a single `void run()` method.

Q15. Which `Set` implementation maintains insertion order?

1. LinkedHashSet
2. HashSet
3. TreeSet
4. Vector

Answer: a — LinkedHashSet
Explanation: It preserves the order in which elements were added.

Q16. Which `Map` keeps its entries sorted by key?

1. HashMap
2. TreeMap
3. ArrayList
4. LinkedList

Answer: b — TreeMap
Explanation: `TreeMap` orders keys by their natural order or a comparator.

Q17. Which collection type does *not* allow duplicates?

1. ArrayList
2. LinkedList
3. HashSet
4. TreeList

Answer: c — HashSet
Explanation: Sets reject duplicate elements by definition.

Q18. To search a text file line by line and report line numbers, which classes fit best?

1. FileInputStream & PipedInputStream
2. FileInputStream & InputStreamReader
3. InputStreamReader & FilterInputStream

4. FileReader & BufferedReader

Answer: d — FileReader & BufferedReader
Explanation: `BufferedReader.readLine()` is optimized for text file processing.

Q19. Which code prints "Finally"?

try { return; } finally { System.out.println("Finally"); }

1. Finally
2. Compilation fails
3. No output
4. Runtime exception

Answer: a — Finally
Explanation: The `finally` block always executes before the return.

Q20. Evaluate this bitwise sequence:

int x = 11 & 9; // 9 int y = x ^ 3; // 10 System.out.println(y | 12); // 14

1. 0
2. 7
3. 8
4. 14

Answer: d — 14
Explanation: 10 OR 12 results in 14.

Q21. Which is FALSE regarding Java serialization?

1. Must implement `Serializable`
2. Transient fields aren't serialized
3. `IOException` can occur
4. Serialization depends on platform

Answer: d — Serialization depends on platform
Explanation: Java serialization is platform-independent.

Q22. What effect does `Thread.join()` have?

1. Caller waits until target thread finishes
2. Interrupts target thread
3. Suspends caller indefinitely
4. Puts thread to sleep

Answer: a — Caller waits until target thread finishes
Explanation: Ensures ordered thread execution.

Q23. Which class provides a dynamically resizable array?

1. Vector
2. ArrayList
3. Arrays
4. HashMap

Answer: b — ArrayList
Explanation: Automatically expands capacity as needed.

Q24. Which of these assignments is INVALID?

interface GrandParent {} class Parent implements GrandParent {} class Baby extends Parent {}

1. `GrandParent g = new Parent();`
2. `GrandParent g = new Baby();`
3. `Parent p = new GrandParent();`
4. `Parent p = new Baby();`

Answer: c — `Parent p = new GrandParent();`
Explanation: Cannot instantiate an interface.

Q25. Default thread priority in Java is:

1. MIN_PRIORITY
2. MAX_PRIORITY
3. NORM_PRIORITY
4. Cannot be determined

Answer: c — NORM_PRIORITY
Explanation: Default is `Thread.NORM_PRIORITY`.

Q26. Which signature must a `Runnable` class implement?

1. `public void run() throws Exception`
2. `public void run()`
3. `public void run(Runnable r)`
4. `public void run(Runnable r, String s)`

Answer: b — `public void run()`
Explanation: Matches `Runnable`'s single method.

Q27. Which is *not* a name for white-box testing?

1. Conformance testing
2. Structural testing
3. Glass-box testing
4. Clear-box testing

Answer: a — Conformance testing
Explanation: Conformance is a black-box technique; the others refer to white-box.

Q28. Black-box testing techniques include:

1. Boundary value analysis
2. Error guessing
3. Special value testing
4. All of the above

Answer: d — All of the above
Explanation: Each relies solely on inputs/outputs without internal code knowledge.

Q29. The final testing phase where the customer tests before acceptance is called:

1. Unit testing
2. Acceptance testing
3. System testing
4. None of the above

Answer: b — Acceptance testing
Explanation: Validates that the product meets business requirements.

Q30. Types of performance testing include:

1. Top-down approach testing
2. Alpha testing
3. Load & stress testing
4. None of the above

Answer: c — Load & stress testing
Explanation: Load tests normal usage; stress tests beyond capacity.

Q31. Which collection stores entries as key-value pairs?

1. HashMap
2. TreeSet
3. LinkedList
4. SortedSet

Answer: a — HashMap
Explanation: Implements the Map interface for key→value storage.

Q32. Which collection does *not* allow duplicates?

1. ArrayList
2. LinkedList
3. HashSet
4. TreeList

Answer: c — HashSet
Explanation: Sets reject duplicate elements.

Q33. Under which scenario is a checked exception thrown?

1. Accessing index 5 in an array of size 3
2. Opening a non-existent file for reading
3. Calling a method on a null `String` reference
4. Invalid database credentials

Answer: b — Opening a non-existent file
Explanation: `FileNotFoundException` is a checked exception.

Q34. Which statement about thread-control methods is TRUE?

1. `sleep()` requires owning an object lock
2. `join()` requires owning an object lock
3. `wait()` requires owning the monitor lock
4. `yield()` requires owning an object lock

Answer: c — `wait()` requires the monitor lock
Explanation: Must be in a `synchronized` block to call `wait()`.

Q35. What happens when you run this code?

Animal a = new Dog(); Cat c = (Cat) a; System.out.println(c.noise());

1. Prints “noise”
2. Prints “bark”
3. Prints

Q35. Determine the output of the following program:

class Animal { public String noise() { return "noise"; } } class Dog extends Animal { public String noise() { return "bark"; } } class Cat extends Animal { public String noise() { return "meow"; } } class MakeNoise { public static void main(String[] args) { Animal animal = new Dog(); Cat cat = (Cat) animal;
System.out.println(cat.noise()); } }

1. noise

1. bark

1. meow

1. ClassCastException

Answer: d — ClassCastException
Explanation: Although `animal` is declared as `Animal`, it's actually referencing a `Dog` object. When you attempt to cast it to `Cat`, the JVM throws a `ClassCastException` at runtime because `Dog` is not a subclass of `Cat`—they are siblings.

Q36. Which statement is INCORRECT regarding instance-initialization blocks?

1. A class can have more than one instance block
2. An instance block cannot initialise the class members
3. Instance blocks are executed before constructors
4. Instance blocks are executed for every created instance

Answer: B — An instance block cannot initialise the class members
Explanation: Instance blocks *can* initialize instance (not static) members; the rest of the statements are true.

Q37. What's the outcome when two threads run this code on the same `res` object?

synchronized(res) { System.out.println("Planet"); res.wait(); Thread.sleep(1000); res.notify(); System.out.println("Earth"); }

1. Planet Planet Earth Earth
2. Planet Earth Planet Earth
3. Deadlock after printing two "Planet"
4. Compilation error

Answer: c — Deadlock after printing two "Planet"
Explanation: Both threads enter, print "Planet" and call `wait()` before any `notify()`, so they block indefinitely.

Q38. For searching a String in a text file (counting occurrences and line numbers), which streams are best?

1. FileInputStream & PipedInputStream
2. FileInputStream & InputStreamReader
3. InputStreamReader & FilterInputStream
4. FileReader & BufferedReader

Answer: D — FileReader & BufferedReader
Explanation: `BufferedReader.readLine()` lets you read line by line efficiently and track line numbers easily.

Q39. Determine the output of this threading code:

class MyThread extends Thread { MyThread() { System.out.print("MyThread"); } public void run() { System.out.print("King"); } public static void main(String[] args) { new MyThread().start(); } }

1. Runtime exception

2. Compile-time error
3. MyThreadKing
4. MyThreadQueen

Answer: c — MyThreadKing
Explanation: Constructor prints "MyThread"; `start()` causes `run()` to print "King."

Q40. Which statement about Java garbage collection is true?

1. Programs can suggest GC but cannot force it
2. Garbage collection is platform-independent
3. The JVM's GC prevents programs from ever running out of memory
4. Java doesn't support garbage collection

Answer: A — Programs can suggest GC but cannot force it
Explanation: Calling `System.gc()` or `Runtime.getRuntime().gc()` is only a request; the JVM decides when to run garbage collection.