1. Яке призначення класу CountDownLatch і яку задачу синхронізації він розв’язує?
2. Як працює лічильник у CountDownLatch і яким чином він впливає на блокування потоків?
3. Які основні методи надає клас CountDownLatch (await, countDown) і як вони використовуються?
4. Чи можна повторно використовувати об’єкт CountDownLatch після досягнення лічильником нуля і чому?
5. У яких сценаріях CountDownLatch є більш доцільним за join() або wait/notify?
6. Що таке бар’єрна синхронізація і яке призначення класу CyclicBarrier?
7. У чому полягає можливість перевикористання (cyclic) бар’єру в CyclicBarrier?
8. Яке призначення бар’єрної дії (barrier action) та коли вона виконується?
9. Що відбувається, якщо один із потоків не досягає бар’єру або перериваєтьcя?
10. У яких випадках доцільно використовувати CyclicBarrier у паралельних алгоритмах?
11. Яке призначення класу Phaser і чим він концептуально відрізняється від CyclicBarrier?
12. Як Phaser підтримує динамічне додавання та видалення «сторін» (participants)?
13. Що таке фаза (phase) в Phaser і як відбувається перехід між фазами?
14. Які основні методи класу Phaser використовуються для керування синхронізацією (arrive, arriveAndAwaitAdvance, register)?
15. У яких сценаріях Phaser є більш гнучким та ефективним рішенням порівняно з CountDownLatch і CyclicBarrier?

16. Яке призначення класу CompletableFuture і які переваги він надає порівняно з інтерфейсом Future?
17. Які методи CompletableFuture використовуються для запуску асинхронних задач?
18. У чому полягає різниця між методами thenApply, thenAccept та thenRun?
19. Як працюють методи композиції thenCompose та thenCombine і в яких випадках їх доцільно використовувати?
20. Які можливості обробки помилок надає клас CompletableFuture (exceptionally, handle, whenComplete)?
21. Яке призначення класу ReentrantLock і чим він відрізняється від механізму synchronized?
22. Які основні методи класу ReentrantLock забезпечують керування блокуванням?
23. Що таке справедливе (fair) блокування в ReentrantLock і як воно впливає на продуктивність?
24. У чому полягає перевага використання ReentrantReadWriteLock у багатопотокових сервісах із переважанням операцій читання?
25. Як використання окремих блокувань на читання та запис (readLock, writeLock) впливає на масштабованість і продуктивність кешів?
26. У чому полягає суть рекурсивного підходу до паралельних обчислень у Java?
27. Яка ідея алгоритму «розділити – виконати – об’єднати» (Divide and Conquer) у контексті ForkJoinPool?
28. Які переваги використання ForkJoinPool порівняно зі звичайними пулами потоків?
29. Яке призначення абстрактного класу ForkJoinTask і які основні методи він надає?

30. Як працює алгоритм «крадіжки задач» і яку роль він відіграє в балансуванні навантаження між потоками?

31. Що таке локальна черга задач у ForkJoinPool і як вона використовується під час виконання рекурсивних завдань?

32. У чому полягає відмінність між класами RecursiveTask<V> та RecursiveAction?

33. У яких випадках доцільно використовувати RecursiveTask, а в яких — RecursiveAction?

34. Яке значення має порогове значення (threshold) у рекурсивних задачах і як воно впливає на продуктивність?

35. Які типові помилки виникають при реалізації рекурсивних паралельних алгоритмів у ForkJoinPool?

36. Що таке інтерфейс ExecutorService і яку роль він відіграє в керуванні виконанням задач у пулі потоків?

37. У чому різниця між методами execute() та submit() інтерфейсу ExecutorService?

38. Які можливості надають методи Future, що повертаються методом submit()?

39. Яке призначення методів shutdown() та shutdownNow() і чим вони відрізняються?

40. Як працює метод awaitTermination() і в яких випадках його доцільно використовувати?

41. Які стани може мати потік у Java та як їх можна визначити за допомогою класу Thread?

42. Яке призначення методу getState() і яку інформацію він повертає?

43. Чим відрізняються методи isAlive() та getState() при аналізі стану потоку?

44. Чому метод Thread.stop() вважається небезпечним і не рекомендований до використання?

45. Як коректно реалізувати механізм зупинки потоку за допомогою interrupt()?

46. Які переваги використання пулів потоків порівняно зі створенням потоків вручну?

47. Для яких задач доцільно використовувати FixedThreadPool і які його обмеження?

48. Чим CachedThreadPool відрізняється від FixedThreadPool з точки зору управління кількістю потоків?

49. Яке призначення ScheduledThreadPool і які типи планування задач він підтримує?

50. У яких сценаріях варто застосовувати SingleThreadExecutor і які гарантії порядку виконання він забезпечує?

51. Що таке блокуючі черги в Java та яку проблему багатопотокового програмування вони розв’язують?

52. Які основні відмінності між інтерфейсами BlockingQueue, TransferQueue та BlockingDeque?

53. У чому полягає концепція блокування при роботі з чергами та деками?

54. Які методи додавання елементів у BlockingQueue існують і чим вони відрізняються (add, offer, put)?

55. Які методи отримання елементів із блокуючої черги використовуються та в яких випадках вони блокують потік (poll, take, peek)?

56. Як працюють тайм-аути при виклику методів offer і poll у блокуючих чергах?

57. Які особливості реалізації та використання класу ArrayBlockingQueue?

58. Для яких задач призначена DelayQueue та які вимоги висуваються до елементів, що в неї додаються?
59. Чим відрізняється LinkedBlockingQueue від ArrayBlockingQueue з точки зору структури та продуктивності?
60. Які особливості має PriorityBlockingQueue та як у ній визначається порядок обробки елементів?
61. У чому специфіка роботи SynchronousQueue і чому вона не зберігає елементи?
62. Яке призначення інтерфейсу TransferQueue і як працюють методи transfer() та tryTransfer()?
63. Які можливості надає LinkedTransferQueue порівняно зі звичайними блокуючими чергами?
64. Чим BlockingDeque відрізняється від BlockingQueue та які операції дозволяють працювати з обох кінців структури даних?
65. Які сценарії використання є типовими для LinkedBlockingDeque у багатопотокових додатках?
66. Що таке потік (thread) у Java та яку роль він відіграє в багатопотоковому програмуванні?
67. У чому полягає різниця між потоками рівня користувача та потоками рівня ядра? Які їхні переваги та недоліки?
68. Як JVM відображає Java-потоки на потоки операційної системи?
69. Яке призначення інтерфейсу Runnable і як він використовується для створення потоку в Java?
70. Які можливості надає інтерфейс Callable порівняно з Runnable?
71. Які основні відмінності між Callable і Runnable щодо повернення результату та обробки винятків?

72. Які ключові методи містить клас java.lang.Thread і для чого вони використовуються (start(), run(), sleep(), join(), interrupt())?

73. Чим відрізняється виклик методу run() від виклику методу start() у класі Thread?

74. Які стани може мати потік (задача) у Java і що означає кожен із них (NEW, RUNNABLE, BLOCKED, WAITING, TIMED_WAITING, TERMINATED)?

75. За яких умов потік переходить зі стану RUNNABLE у BLOCKED, WAITING або TIMED_WAITING?