1. What will be the state of a thread immediately after calling the start() method on a Thread object?

a. NEW
b. RUNNABLE
c. BLOCKED
d. TERMINATED

2. What will happen if a thread calls wait() on an object without owning its intrinsic lock?

a. The thread enters the WAITING state.
b. A java.lang.IllegalMonitorStateException is thrown.
c. The thread moves to the BLOCKED state.
d. Nothing; the thread continues execution.

3. Consider the following program:
What will be printed at Line 1, Line 2, and Line 3?

a. NEW, RUNNABLE, TERMINATED
b. NEW, TIMED_WAITING, TERMINATED
c. RUNNABLE, RUNNABLE, TERMINATED
d. NEW, RUNNABLE, RUNNABLE

```
public class Main {
    public static void main(String[] args) throws InterruptedException {
        Thread t = new Thread(() -> {
            for (int i = 0; i < 5; i++) {
                try {
                    Thread.sleep(100);
                } catch (InterruptedException e) {
                    System.out.println("Interrupted!");
                }
            }
        });
        System.out.println(t.getState()); // Line 1
        t.start();
        Thread.sleep(50);
        System.out.println(t.getState()); // Line 2
        t.join();
        System.out.println(t.getState()); // Line 3
    }
}
```

4. What happens when a thread acquires a lock on an object and attempts to acquire another lock on the same object?

```
class Example {
    synchronized void performTask() {
        synchronized (this) {
            System.out.println("Task performed.");
        }
    }
}
```

a. A java.lang.IllegalMonitorStateException will be thrown.
b. The thread will enter a deadlock situation.
c. The code will execute without any issue.
d. The program will terminate due to a stack overflow.

5. What happens in the following code?

```
class Example {
    static final Object lock1 = new Object();
    static final Object lock2 = new Object();
    public static void main(String[] args) {
        Thread t1 = new Thread(() -> {
            synchronized (lock1) {
                try { Thread.sleep(50); } catch (InterruptedException e) {}
                synchronized (lock2) {}
            }
        });
        Thread t2 = new Thread(() -> {
            synchronized (lock2) {
                try { Thread.sleep(50); } catch (InterruptedException e) {}
                synchronized (lock1) {}
            }
        });
        t1.start();
        t2.start();
    }
}
```

a. Both threads execute without issue.

b. A deadlock will occur as both threads wait indefinitely.

c. Thread t2 will finish execution first.

d. The program will terminate immediately after execution.

6. What happens if a thread calls join() with a timeout, but the other thread does not finish within the specified timeout?

```
class Main {
    public static void main(String[] args) throws InterruptedException {
        Thread t = new Thread(() -> {
            try {
                Thread.sleep(5000);
            } catch (InterruptedException e) {}
        });
        t.start();
        t.join(1000);
        System.out.println("Main thread finished waiting.");
    }
}
```

a. The main thread waits indefinitely.

b. The main thread throws a java.lang.InterruptedException.

c. The main thread continues after 1 second.

d. The program will crash.

7. What is the state of a thread after it is interrupted while sleeping?

a. RUNNABLE

b. WAITING

c. TERMINATED

d. BLOCKED

8. What happens if an exception is thrown inside a synchronized block?

a. The lock is not released, and the second thread waits indefinitely.

b. The lock is released, allowing the second thread to execute.

c. A java.lang.IllegalMonitorStateException is thrown.

d. Both threads execute concurrently.

9. What is the correct output of the following code?

```
class Main {
    public static void main(String[] args) throws InterruptedException {
        Object lock = new Object();
        Thread t1 = new Thread(() -> {
            synchronized (lock) {
                try {
                    lock.wait();
                    System.out.println("Thread 1 resumed");
                } catch (InterruptedException e) {}
            }
        });
        Thread t2 = new Thread(() -> {
            synchronized (lock) {
                try {
                    lock.wait();
                    System.out.println("Thread 2 resumed");
                } catch (InterruptedException e) {}
            }
        });
        t1.start();
        t2.start();
        Thread.sleep(100);
        synchronized (lock) {
            lock.notifyAll();
        }
    }
}
```

a. "Thread 1 resumed" followed by "Thread 2 resumed".

b. "Thread 2 resumed" followed by "Thread 1 resumed".

c. Both threads will resume execution, but the order is unpredictable.

d. The program will deadlock.

10. Which of the following statements is true about the difference between Thread.sleep() and Object.wait()?

a. Both methods release the lock on the monitor object.

b. Thread.sleep() does not release the lock, but Object.wait() does.

c. Thread.sleep() releases the lock, but Object.wait() does not.

d. Both methods block the thread and release the lock.

11. In the following program, can you think of a scenario in which the Final balance will be different from the expected balance? Explain!

```
class BankAccount {
    private int balance;
    public BankAccount(int initialBalance) {
        this.balance = initialBalance;
    }
    public void deposit(int amount) {
        balance += amount;
    }
    public void withdraw(int amount) {
        if (balance >= amount) {
            balance -= amount;
        }
    }
    public int getBalance() {
        return balance;
    }
}
public class BankTransferBug {
    public static void main(String[] args) throws InterruptedException {
        BankAccount account = new BankAccount(100);
        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 100; i++) {
                account.withdraw(10);
            }
        });
        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 100; i++) {
                account.deposit(10);
            }
        });
        t1.start();
        t2.start();
        t1.join();
        t2.join();
        System.out.println("Final balance: " + account.getBalance());
    }
}
```

Under certain thread interleavings, the getBalance operation in withdraw and deposit may not see a consistent state due to a **data race**, leading to an incorrect final balance.
For example:

1. Thread t1 reads the current balance as 100.
2. Before t1 subtracts 10, Thread t2 adds 10, making the balance 110.
3. t1 proceeds to subtract 10, leaving the balance as 100, effectively ignoring the deposit operation.

1. What happens if you override the run method in a Thread class but call the run() method directly instead of start()?

a. A new thread will be created and start executing the run method.

b. The run method will execute in the same thread that called it.

c. A RuntimeException will be thrown.

d. A compile-time error will occur.

2. What happens if a thread is waiting for a monitor and the monitor becomes available?

a. The thread moves to the RUNNABLE state.

b. The thread moves to the BLOCKED state.

c. The thread moves to the TERMINATED state.

d. The thread immediately acquires the monitor and starts execution.

3. What is the correct way to check if a thread has been interrupted during its execution?

a. Thread.isInterrupted()

b. Thread.currentThread().isInterrupted()

c. Thread.checkInterrupt()

d. Thread.currentThread().interrupted()s

4. Consider the following program:

```
class MyThread extends Thread {
    public void run() {
        while (!Thread.interrupted()) {
            System.out.println("Running...");
        }
        System.out.println("Interrupted");
    }
}
public class Main {
    public static void main(String[] args) throws InterruptedException {
        MyThread t = new MyThread();
        t.start();
        Thread.sleep(10);
        t.interrupt();
    }
}
```

What will be the output?

a. "Running..." printed continuously.

b. "Interrupted" will be printed after some "Running...".

c. The program will terminate without any output.

d. A java.lang.InterruptedException will be thrown.

5. What will be the state of a thread after it calls wait() on an object?

a. WAITING

b. BLOCKED

c. RUNNABLE

d. TERMINATED

6. What will happen in the following scenario?

```java
class Example {
    synchronized static void methodA() {
        System.out.println(Thread.currentThread().getName() + " is in methodA.");
        try { Thread.sleep(1000); } catch (InterruptedException e) {}
    }

    synchronized void methodB() {
        System.out.println(Thread.currentThread().getName() + " is in methodB.");
    }
}

public class Main {
    public static void main(String[] args) {
        Example obj1 = new Example();
        Example obj2 = new Example();
        Thread t1 = new Thread(() -> obj1.methodA());
        Thread t2 = new Thread(() -> obj2.methodB());
        t1.start();
        t2.start();
    }
}
```

a. Both threads execute concurrently.

b. t1 waits for t2 to finish because both methods are synchronized.

c. t2 waits for t1 to finish because both methods are synchronized.

d. Both threads execute without interfering with each other.

7. What is the output of the following code snippet?

```java
class Example {
    private final Object lock1 = new Object();
    private final Object lock2 = new Object();

    void method1() {
        synchronized (lock1) {
            System.out.println("Lock1 acquired by method1");
            try { Thread.sleep(100); } catch (InterruptedException e) {}
            synchronized (lock2) {
                System.out.println("Lock2 acquired by method1");
            }
        }
    }
    void method2() {
        synchronized (lock2) {
            System.out.println("Lock2 acquired by method2");
            try { Thread.sleep(100); } catch (InterruptedException e) {}
            synchronized (lock1) {
                System.out.println("Lock1 acquired by method2");
            }
        }
    }

    public static void main(String[] args) {
        Example example = new Example();
        Thread t1 = new Thread(example::method1);
        Thread t2 = new Thread(example::method2);
        t1.start();
        t2.start();
    }
}
```

a. Both threads execute successfully and print all messages.

b. Deadlock occurs, and the program hangs.

c. Only method1 executes, as method2 waits indefinitely.

d. A java.lang.IllegalMonitorStateException is thrown.

8. What happens when wait() is called on an object, but no thread calls notify() or notifyAll() on the same object?

a. The waiting thread continues after a default timeout.

b. The waiting thread remains in the WAITING state indefinitely.

c. The thread throws a java.lang.InterruptedException after some time.

d. The JVM terminates the program due to a deadlock.

9. Which of the following statements is true about the difference between Thread.sleep() and Object.wait()?

a. Both methods release the lock on the monitor object.

b. Thread.sleep() does not release the lock, but Object.wait() does.

c. Thread.sleep() releases the lock, but Object.wait() does not.

d. Both methods block the thread and release the lock.

10. What happens if a thread tries to synchronize on a null object?

```
public class Main {
    public static void main(String[] args) {
        Object lock = null;
        synchronized (lock) {
            System.out.println("In synchronized block.");
        }
    }
}
```

a. Both methods release the lock on the monitor object.

b. Thread.sleep() does not release the lock, but Object.wait() does.

c. Thread.sleep() releases the lock, but Object.wait() does not.

d. Both methods block the thread and release the lock.

a. The code compiles and runs without issues.

b. A java.lang.NullPointerException is thrown.

c. The thread waits indefinitely.

d. The JVM crashes due to an invalid monitor.

11. In the following program, can you think of a scenario in which the Final balance will be different from the expected balance? Explain!

```
class BankAccount {
    private int balance;
    public BankAccount(int initialBalance) {
        this.balance = initialBalance;
    }
    public void deposit(int amount) {
        balance += amount;
    }
    public void withdraw(int amount) {
        if (balance >= amount) {
            balance -= amount;
        }
    }
    public int getBalance() {
        return balance;
    }
}
public class BankTransferBug {
    public static void main(String[] args) throws InterruptedException {
        BankAccount account = new BankAccount(100);
        Thread t1 = new Thread(() -> {
            for (int i = 0; i < 100; i++) {
                account.withdraw(10);
            }
        });
        Thread t2 = new Thread(() -> {
            for (int i = 0; i < 100; i++) {
                account.deposit(10);
            }
        });
        t1.start();
        t2.start();
        t1.join();
        t2.join();
        System.out.println("Final balance: " + account.getBalance());
    }
}
```

Under certain thread interleavings, the getBalance operation in withdraw and deposit may not see a consistent state due to a **data race**, leading to an incorrect final balance.

For example:

1. Thread t1 reads the current balance as 100.
2. Before t1 subtracts 10, Thread t2 adds 10, making the balance 110.
3. t1 proceeds to subtract 10, leaving the balance as 100, effectively ignoring the deposit operation.