PROGRAMMING IN JAVA

Assignment 06

TYPE OF QUESTION: MCQ

Number of questions: 10 Total marks**: 10 × 1 = 10**

QUESTION 1:

Consider the following code snippet.

```
class MyThread extends Thread {
    public void run() {
        for (int i = 0; i < 3; i++) {
            System.out.println("Running thread: " + i);
        }
    }
}

public class Main {
    public static void main(String[] args) {
        MyThread thread = new MyThread();
        thread.run();
        System.out.println("Main method complete.");
    }
}
```

What will be the output of the program?

a. **Running thread: 0**
   Running thread: 1
   Running thread: 2
   Main method complete.

b. **Running thread: 0**
   Main method complete.
   Running thread: 1
   Running thread: 2

c. **Main method complete.**

d. **Error: The thread was not started using `start()`**

Correct Answer:

a. **Running thread: 0**
   Running thread: 1
   Running thread: 2
   Main method complete

Detailed Solution:

In this program, the run() method is called directly instead of using `start()`. When `run()` is called like this, it does not create a new thread; it behaves like a regular method call and runs on the main thread. As a result, the output is sequential. To create a separate thread, you must call the `start()` method.For a more detailed understanding of thread methods like `run()` and `start()`, refer to the book Joy with Java.

QUESTION 2:

Which of the following best describes the concept of multithreading in Java?

a. **Multiple threads execute concurrently, sharing the same memory space.**

b. **Only one thread executes at a time, ensuring sequential execution.**

c. **Threads in Java cannot communicate with each other.**

d. **Threads require separate memory allocation for each thread to run.**

Correct Answer:

a. **Multiple threads execute concurrently, sharing the same memory space.**

Detailed Solution:

Multithreading in Java allows multiple threads to run concurrently within the same program, sharing the same memory space. This feature makes Java programs more efficient, especially for tasks like multitasking or background processing. However, proper synchronization is crucial to avoid issues like data inconsistency. For a more detailed explanation of multithreading concepts, refer to the book Joy with Java.

QUESTION 3:

What will happen when the following code is executed?

```
class ExampleThread extends Thread {
    public void run() {
        System.out.println("Thread is running.");
    }
}

public class Main {
    public static void main(String[] args) {
        ExampleThread thread = new ExampleThread();
        thread.start();
        thread.start();
    }
}
```

a. **The program will execute successfully, printing "Thread is running." twice.**

b. **The program will throw an error when attempting to start the thread a second time.**

c. **The program will terminate without any output.**

d. **The thread will run only once, and the second start() call will be ignored.**

Correct Answer:

b. The program will throw an error when attempting to start the thread a second time.

Detailed Solution:

In Java, a thread can only be started once. If you try to call start() on the same thread object more than once, the JVM will throw an IllegalThreadStateException. This is because a thread's lifecycle allows it to transition to the "terminated" state after completing execution, and it cannot be restarted.
For more details about the thread lifecycle, refer to the book Joy with Java.

QUESTION 4:

Find the error in the following program:

```
class RunnableExample implements Runnable {
    public void run() {
        for (int i = 1; i <= 3; i++) {
            System.out.println("Runnable thread: " + i);
        }
    }
}

public class Main {
    public static void main(String[] args) {
        RunnableExample runnable = new RunnableExample();
        runnable.run();
        System.out.println("Main method ends.");
    }
}
```

a. **The program will throw an error because Runnable cannot be executed directly.**

b. **The program will run successfully but will not create a new thread.**

c. **The program will create a new thread and execute concurrently.**

d. **The program will throw a runtime error because Thread is not used.**

Correct Answer:

b. **The program will run successfully but will not create a new thread.**

Detailed Solution:

In this code, the Runnable object is created, but it is executed directly using the run() method instead of passing it to a Thread object. To create a new thread, you need to use:

```
Thread t = new Thread(runnable);
t.start();
```

Without this, the run() method behaves like a regular method call, and no new thread is created.
For more detailed guidance on creating threads using Runnable, refer to the book Joy with Java.

QUESTION 5:

What will happen when the following code is executed?

```
 class RunnableExample implements Runnable {
    public void run() {
        for (int i = 1; i <= 3; i++) {
            System.out.println("Thread running: " + i);
        }
    }
}

public class Main {
    public static void main(String[] args) {
        RunnableExample task = new RunnableExample();
        Thread thread = new Thread(task);
        thread.start();
    }
}
```

a. **The program will throw a compile-time error because `Runnable` is not a thread.**

b. **The program will execute successfully, and the `run()` method will run in a new thread.**

c. **The program will execute the `run()` method directly on the main thread.**

d. **The program will throw a runtime error because `Runnable` is not properly implemented.**

Correct Answer:

b. **The program will execute successfully, and the `run()` method will run in a new thread.**

Detailed Solution:

The Runnable interface is implemented by the class RunnableExample. To create a thread, the Runnable object is passed to the Thread constructor, and calling start() ensures that the run() method is executed in a new thread.For more details on creating threads using Runnable, refer to the book Joy with Java.

QUESTION 6:

Which of the following states can a thread enter during its lifecycle in Java?

a. **New, Runnable, Running, Blocked**

b. **New, Runnable, Waiting, Blocked, Terminated**

c. **New, Runnable, Running, Sleeping, Dead**

d. **New, Active, Waiting, Suspended, Terminated**

Correct Answer:

b. **New, Runnable, Waiting, Blocked, Terminated**

Detailed Solution:

The states of a thread in Java are:

- **New:** Thread is created but not started.
- **Runnable:** Thread is ready to run but waiting for CPU time.
- **Waiting/Blocked:** Thread is paused or waiting for some resource.
- **Terminated:** Thread has completed execution.

For a detailed explanation of thread states, refer to the book *Joy with Java*.

QUESTION 7:

What does the thread scheduler use to decide which thread to run when multiple threads are in the runnable state?

a. **Thread priority**

b. **Thread's execution time**

c. **Thread name**

d. **Thread creation order**

Correct Answer:

a. **Thread priority**

Detailed Solution:

The thread scheduler uses thread priorities as a hint to determine the execution order of threads in the runnable state. However, thread scheduling also depends on the operating system's scheduling policies and may not strictly follow priority.
For more on thread scheduling, refer to the book Joy with Java.

QUESTION 8:

Consider the following program:

```
class PriorityExample extends Thread {
    public void run() {
        System.out.println(Thread.currentThread().getName() +
                           " with priority " +
Thread.currentThread().getPriority());
    }
}

public class Main {
    public static void main(String[] args) {
        PriorityExample t1 = new PriorityExample();
        PriorityExample t2 = new PriorityExample();

        t1.setPriority(Thread.MIN_PRIORITY);
        t2.setPriority(Thread.MAX_PRIORITY);

        t1.start();
        t2.start();
    }
}
```

Which of the following is true about the output?

a. **The thread with the higher priority is guaranteed to execute first.**

b. **The thread with the lower priority will never execute.**

c. **The order of execution depends on the JVM and OS scheduling policies.**

d. **The program will throw an error due to invalid priority values.**

Correct Answer:

c. **The order of execution depends on the JVM and OS scheduling policies.**

Detailed Solution:

Thread priority is a hint to the thread scheduler but does not guarantee execution order. The actual behavior depends on the JVM and the underlying OS.For more on thread priorities, refer to the book *Joy with Java*.

QUESTION 9:

What is the primary purpose of thread synchronization in Java?

a. **To allow multiple threads to execute a method at the same time**

b. **To ensure thread execution follows a specific order**

c. **To prevent race conditions and ensure data consistency**

d. **To allow threads to communicate with each other**

Correct Answer:

c. **To prevent race conditions and ensure data consistency**

Detailed Solution:

Thread synchronization is used to control the access of multiple threads to shared resources, ensuring data consistency and preventing race conditions. This is typically done using synchronized methods or blocks. For more details on thread synchronization, refer to the book *Joy with Java*.

QUESTION 10:

What is the primary difference between Byte Streams and Character Streams in Java?

a. **Byte Streams handle characters, while Character Streams handle bytes.**

b. **Byte Streams are used for binary data, while Character Streams are used for text data.**

c. **Character Streams are faster than Byte Streams in all cases.**

d. **Character Streams cannot handle international characters like Unicode.**

Correct Answer:

b. **Byte Streams are used for binary data, while Character Streams are used for text data.**

Detailed Solution:

Byte Streams: Handle raw binary data like images or files (InputStream, OutputStream).
Character Streams: Handle text data and support encoding like Unicode (Reader, Writer).
Character Streams are better for text processing, especially when handling international characters.
For a detailed understanding of Java I/O streams, refer to the book *Joy with Java*.