NPTEL Online Certification Courses

Indian Institute of Technology Kharagpur

PROGRAMMING IN JAVA

Assignment 6

TYPE OF QUESTION: MCQ

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

QUESTION 1:

Which of the following is NOT TRUE in Java?

a. Every thread has a priority.
b. JVM allows multiple threads of execution running concurrently.
c. Threads with higher priority are executed first.
d. You cannot set a maximum priority value that a thread can have.

Correct Answer: d

Detailed Solution:

A thread is a thread of execution in a program. The Java Virtual Machine allows an application to have multiple threads of execution running concurrently. Every thread has a priority. Threads with higher priority are executed in preference to threads with lower priority. The MAX_PRIORITY field can be used to set priority value of a thread in Java.

QUESTION 2:

Consider the following code:

```
class PrimeRun implements Runnable {
         long minPrime;
         PrimeRun(long minPrime) {
             this.minPrime = minPrime;
         }

         public void run() {
             // compute primes larger than minPrime
               . . .
         }
     }
```

Which of the following is TRUE regarding the above code?

a. Creating a thread in Java using Runnable interface
b. Thread creation by declaring a class to be a subclass of Thread
c. Overriding the run method of class Thread
d. The class implements the run method.

Correct Answer: a, d

Detailed Solution:

There are two ways to create a new thread of execution.

One is to declare a class to be a subclass of Thread.
This subclass should override the run method of class Thread. An instance of the subclass can then be allocated and started.

The other way to create a thread is to declare a class that implements the Runnable interface.
That class then implements the run method. An instance of the class can then be allocated, passed as an argument when creating Thread, and started. The same example is given here.

QUESTION 3:

Which of the following cannot be used to create an instance of Thread?

a. By implementing the Runnable interface.
b. By extending the Thread class.
c. By creating a new class named Thread and calling method run().
d. By importing the Thread class from the related package.

Correct Answer: c, d

Detailed Solution:

An application that creates an instance of Thread must provide the code that will run in that thread. There are two ways to do this:

- *Provide a Runnable object.* The Runnable interface defines a single method, run, meant to contain the code executed in the thread. The Runnable object is passed to the Thread constructor
- *Subclass Thread.* The Thread class itself implements Runnable, though its run method does nothing. An application can subclass Thread, providing its own implementation of run

Reference:https://docs.oracle.com/javase/tutorial/essential/concurrency/runthread.html

---

QUESTION 4:

What is the use of `sleep(long millis)` function in Java?

a. Pauses the thread execution for exactly the given time with no dependency.
b. Pauses the thread execution but depends on precision of system timers.
c. Pauses the thread execution but depends on precision of system schedulers.
d. Pauses the thread execution but depends on precision of both system timers and schedulers.

Correct Answer: d

Detailed Solution:

The `sleep(long millis)` function causes the currently executing thread to sleep (*temporarily cease execution*) for the specified number of milliseconds, subject to the precision and accuracy of system timers and schedulers.

QUESTION 5:

Which method `start()` will do which of the following?

a. Causes this thread to begin execution.
b. Either start the execution for new thread or pause an ongoing thread.
c. The JVM calls the run method of this thread.
d. Recovers a thread from deadlock and begin execution

Correct Answer: a, c

Detailed Solution: The `start()` method causes this thread to begin execution; the Java Virtual Machine calls the run method of this thread. This method cannot pause execution of a thread. Further, the threads are inherently deadlock-prone.

QUESTION 6:

Which of the following is not platform independent in Java?

a) Everything in Java thread is platform dependent.
b) The thread constructor with the stackSize parameter is platform dependent.
c) The inheritance in java is platform dependent as multiple classes are involved.
d) There is no platform dependency in Java.

Answer : **b**

Explanation:

public **Thread**(ThreadGroup *group*, Runnable *target*, String *name*, long *stackSize*)

The above constructor allocates a new Thread object so that it has target as its run object, has the specified name as its name, and belongs to the thread group referred to by group, and has the specified stack size. This constructor is identical to Thread(ThreadGroup,Runnable,String) with the exception of the fact that it allows the thread stack size to be specified. The stack size is the approximate number of bytes of address space that the virtual machine is to allocate for this thread's stack. The effect of the stackSize parameter, if any, is highly platform dependent. On some platforms, specifying a higher value for the stackSize parameter may allow a thread to achieve greater recursion depth before throwing a StackOverflowError. Similarly, specifying a lower value may allow a greater number of threads to exist concurrently without throwing an OutOfMemoryError (or other internal error). The details of the relationship between the value of the stackSize parameter and the maximum recursion depth and concurrency level are platform-dependent. On some platforms, the value of the stackSize parameter may have no effect whatsoever.

Due to the platform-dependent nature of the behavior of this constructor, extreme care should be exercised in its use. The thread stack size necessary to perform a given computation will likely vary from one JRE implementation to another. In light of this variation, careful tuning of the stack size parameter may be required, and the tuning may need to be repeated for each JRE implementation on which an application is to run.

QUESTION 7:

The following is a simple program using the concept of thread.

```
public class Question extends Thread{
 public void run(){
  for(int i=1;i<5;i++){
    System.out.println(++i);
  }
 }
 public static void main(String args[]){
    Question t1=new Question();
    t1.run();
 }
}
```

What is the output of the above program?

a. 1
   3
b. 2
   4
   6
   8
c. Runtime error
d. 2
   4

Correct Answer: b

Detailed Solution:

The increment operators increase the value of ***i* to 2** in the first run. Afterwards, two increments are happening till ***i* < 8** condition is not satisfied.

QUESTION 8:

For the program given below, what will be the output after its execution?

```
public class Main{
public static void main(String[]args){
        Thread thread=Thread.currentThread();
        thread.run();
        System.out.print(thread.activeCount());
       }
}
```

a. 1
b. 10
c. 01
d. 11

Correct Answer: a

Detailed Solution:

java.lang.Thread.activeCount() : Returns an estimate of the number of active threads in the current thread's thread group and its subgroups.

QUESTION 9:

Which of the following method returns a reference to the currently executing thread object?

a. public static boolean interrupted()
b. public static Thread currentThread()
c. public final boolean isAlive()
d. public final void suspend()

Correct Answer: b

Detailed Solution:

Only *public static Thread currentThread() method r*eturns a reference to the currently executing thread object among the options.

QUESTION 10:

Which of the following methods can be used to reduce over-utilization of CPU?

a. public static void yield()
b. public static void main(String args[])
c. public static void sleep(long millis)
d. public void start()

Answer: a

Explanation:

public static void yield() gives a hint to the scheduler that the current thread is willing to yield its current use of a processor. The scheduler is free to ignore this hint.

Yield is a heuristic attempt to improve relative progression between threads that would otherwise over-utilise a CPU. Its use should be combined with detailed profiling and benchmarking to ensure that it actually has the desired effect.

It is rarely appropriate to use this method. It may be useful for debugging or testing purposes, where it may help to reproduce bugs due to race conditions. It may also be useful when designing concurrency control constructs such as the ones in the java.util.concurrent.locks package.

**************END**************