**Quiz 3**

Questions on how threads can be created

**Question # 1**

***Give an example of creating a thread using the Runnable interface?***

The below snippet creates an instance of the **Thread** class by passing in a lambda expression to create an anonymous class implementing the **Runnable** interface.

        Thread t = new Thread(() -> {  
            System.out.println(this.getClass().getSimpleName());  
        });  
   
        t.start();  
        t.join();

1

2

3

4

5

6

7

8

9

10

11

12

class Demonstration {

    public static void main( String args[]) throws Exception {

        Thread t = new Thread(() -> {

            System.out.println("Hello from thread !");

        });

        t.start();

        t.join();

    }

}





RUN

SAVERESET

**Question # 2**

***Give an example of a thread running a task represented by the Callable<V> interface?***

There's no constructor in the **Thread** class that takes in a type of **Callable**. However, there is one that takes in a type of **Runnable**. We can't directly execute a callable task using an instance of the **Thread** class. However we can submit the callable task to an executor service. Both approaches are shown below:

**Callable with Thread Class**

        // Anoymous class  
        Callable<Void> task = new Callable<Void>() {  
   
            @Override  
            public Void call() throws Exception {  
                System.out.println("Using callable indirectly with instance of thread class");  
                return null;  
            }  
        };  
   
        // creating future task  
        FutureTask<Void> ft = new FutureTask<>(task);  
        Thread t = new Thread(ft);  
        t.start();  
        t.join();

**Callable with Executor Service**

        // Anoymous class  
        Callable<Void> task = new Callable<Void>() {  
   
            @Override  
            public Void call() throws Exception {  
                System.out.println("Using callable indirectly with instance of thread class");  
                return null;  
            }  
        };  
   
        ExecutorService executorService = Executors.newFixedThreadPool(5);  
        executorService.submit(task);  
        executorService.shutdown();

18

19

20

21

22

23

24

25

26

27

28

29

30

31

32

33

34

35

36

37

38

39

40

41

42

43

44

45

46

47

48

            public Void call() throws Exception {

                System.out.println("Using callable with executor service.");

                return null;

            }

        };

        ExecutorService executorService = Executors.newFixedThreadPool(5);

        executorService.submit(task);

        executorService.shutdown();

    }

    static void usingThread() throws Exception {

        // Anoymous class

        Callable<Void> task = new Callable<Void>() {

            @Override

            public Void call() throws Exception {

                System.out.println("Using callable indirectly with instance of thread class");

                return null;

            }

        };

        // creating future task

        FutureTask<Void> ft = new FutureTask<>(task);

        Thread t = new Thread(ft);

        t.start();

        t.join();

    }

}





RUN

SAVERESET

**Question # 3**

***Give an example of representing a class using the Thread class.***

We can extend from the **Thread** class to represent our task. Below is an example of a class that computes the square roots of given numbers. The **Task** class encapsulates the logic for the task being performed.

class Task<T extends Number> extends Thread {  
   
    T item;  
   
    public Task(T item) {  
        this.item = item;  
    }  
   
    public void run() {  
        System.out.println("square root is: " + Math.sqrt(item.doubleValue()));  
    }  
}

1

2

3

4

5

6

7

8

9

10

11

12

13

14

15

16

17

18

19

20

21

22

23

24

25

26

27

class Demonstration {

    public static void main( String args[] ) throws Exception{

      Thread[] tasks = new Thread[10];

      for(int i = 0;i<10;i++) {

        tasks[i] = new Task(i);

        tasks[i].start();

      }

      for(int i = 0;i<10;i++) {

        tasks[i].join();

      }

    }

}

class Task<T extends Number> extends Thread {

    T item;

    public Task(T item) {

        this.item = item;

    }

    public void run() {

        System.out.println("square root is: " + Math.sqrt(item.doubleValue()));

    }

}