# Chapter 8 – Threads

## Answers

1. A

   The method run() has no version that accepts a parameter, and here the method run is accepting an int parameter. Therefore the class "Hevron" does not properly implement the Runnable interface, and a compilation error will occur.

2. A

   The word "Hello" will be printed first, and then if the thread has to yield it will. If so, eventually it will continue running and the word " Israel" will be printed as well.

3. A

   The word "Hello" will be printed first, and then if the thread has to yield it will. If so, eventually it will continue running. The resume called does nothing here since suspend was never called, so then the word " Israel" will be printed as well.

4. B

   In this example there is a class called Jerusalem that is a sub class of Thread, and redefines the "run()" method. In the "main" method a Jersualem object is created and its reference is placed into a variable of type Thread, and then "start" is called on the thread. The "run" method then executes and the text "Hello" is printed. After this "suspend" is called on the thread, so it stops executing and waits for a "resume" to be called on it. Since "resume" is never called the thread never continues and the text " Israel" is never printed.

5. B

   In this example there is a class called Jerusalem that is a sub class of Thread, and redefines the "run()" method. In the "main" method a

Jersualem object is created and its reference is placed into a variable of type Thread, and then “start” is called on the thread. The “run” method then executes and the text “Hello” is printed. After this “suspend” is called on the thread, so it stops executing and waits for a “resume” to be called on it. The statement “resume()” on the next line does not execute since this thread has stopped running for now. Another thread needs to call “resume” on this thread for it to actually resume. Therefore the text “ Israel” is never printed.

6. C, D

   The Windows platform uses a time sliced implementation of thread scheduling, so in this case both thread will take equal turns running. The Solaris platform uses a preemptive implementation of thread scheduling, therefore only t1 will get to run. This is because with a preemptive implementation the thread with the highest priority runs until it is done, and here the while loop is never ending, so since t1 is set to run first, it will start and will never end.

7. B

   There is no way to specify which waiting thread is to be called with the notify() method.

8. A

   A monitor is an instance of any class that has all of its variables marked as private, and all of its methods that are not private marked with the “synchronized” keyword. The purpose of monitors is that at times a thread cannot do what it needs to do until the object that it is working with reaches a certain state. Monitors have tight control over the data, so the thread can be run at the proper times, so everything can run as is expected.

9. B

   A thread that is sent to sleep will sleep for at least time that was set. After that is goes into “ready” state, waiting to get a chance to run, which may be right away or take some time.

10. B

    There is no way to specify which waiting thread is to be called with the notify() method.

11. A

    The resume method is a deprecated method that only works with the suspend method on a thread. So if there is no suspend called on the thread then the resume does nothing.

12. A

    When the suspend method is called on a thread that thread will not run until the resume method is called on it.

13. B

    A thread that is sent to sleep will sleep for at least time that was set. After that is goes into “ready” state, waiting to get a chance to run, which may be right away or take some time.

14. B

    The method run() has no version that accepts a parameter, and here the method run is accepting an int parameter. Therefore the class “Hevron” does not properly implement the Runnable interface, and a compilation error will occur.

15. B

    The method start() has not version that accepts a parameter, and here the method start is accepting a number parameter. This will cause a compilation error.

    Also the “num” variable in the parenthesis of the “for” loop was never declared or initialized, so this will cause a compilation error as well.

16. A,B,E

    When a thread finishes what is in the run() method, it is considered dead. The start() method cannot be called again on the same object. The only way to run that method again is to create a new object and call start() again. Although the thread is dead, the object created is still an object, so it's other data and methods can still be accessed.

17. A,B,C,D,E

All of the answers here are correct statements.

18. A

The word "TANGA" is printed each time a deposit is made, that is each time the method "deposit" is called.

There are 2 "Depositer" objects created "dep1" and "dep2". This Depositer class implements the interface Runnable, and therefore defines a run() method.

Then there are 2 Thread objects created that are passed a Depositer object as their runnable objects, dep1Th and dep2Th.

Next the method start() is called for each thread, so each one in turn called the run() method of its Runnable object, which is a Depositer object for threads dep1 and dep2s. In the run() method of the class Depositer, the deposit method is called 4 times in total, twice in each "for" loop. So since there are 2 threads running that are doing the same thing, the Depositer method is called 8 times in total.

19. A

The word "BINGO" is printed each time a withdrawal is made, that is each time the method "withdraw" is called.

There are 2 "Withdrawer" objects created "with1" and "with2". This Withrawer class implements the interface Runnable, and therefore defines a run() method.

Then there are 2 Thread objects created that are passed a Withdrawer object as their runnable objects, with1Th and with2Th.

Next the method start() is called for each thread, so each one in turn called the run() method of its Runnable object, which is a Withdrawer object for threads with1Th and with2Th.

In the run() method of the class Withdrawer, the Withdraw method is called 4 times in total, in one “for” loop. So since there are 2 threads running that are doing the same thing, the withdraw method is called 8 times in total.

20. A,B,C,D

When a thread calls wait() it gives up the CPU and the lock and goes into the waiting pool of its Monitor object, and waits there until it is notified with the method notify() or notifyAll(). If a thread in the waiting pool gets interrupted with the interrupt() method, or is affected by notify() or notifyAll(), it goes into the ready state, where it waits its turn to run again.

The method notify() moves one thread from the waiting pool to the ready state, and the method notifyAll() moves all of the threads from the waiting pool to the ready state.

21. B

This example defines a new class called “Thready”, which extends from the Thread class, and redefines the “run()” method. This class holds a static variable called “numOfThreadys”, which is initialized to 1, and an instance variable of type Object called “ob”, that holds a reference to a new Object instance.

In the “main” method first an object of type Object is created and its reference is put into a variable called “locky”, but it is never used in the rest of the code here, so it is irrelevant. Then an array of type Thread is created, and its length is set to be 50. After this there is a “for” loop that creates 50 new Thready objects and places their references in the array. After each object is created the “start()” method is called on it, which calls the “run()” method. Each time the “run()” method is called it prints out the value in the static class variable “numberOfThreadys” and then increases the value of this variable by 1. Since the “start()” method is called for each of the 50

Thready objects, the “numberOfThreadys” variable value will be increased 50 times, and 50 numbers will be printed to the screen, each between the value of 1 and 50.

A separate instance of the object reference held by the “ob” variable is created for each Thready instance. Therefore the synchronized line, which is passed “ob” as a parameter, is being passed a different version of “ob” each time called here. So each of the Thready objects is synchronized on a different “ob” object, so one does not have to wait for the other to finish the synchronized code in order to run the “run()” method. Due to this it is possible that the value of “numberOfThreadys” could be increased by more than one thread at the same time, so it is possible that some of the numbers printed out will be duplicated. But, each of the numbers printed out will be either the same as the previous number printed, or greater, ending with 50.

22. A

This example defines a new class called “Thready”, which extends from the Thread class, and redefines the “Run()” method. This class holds a static int variable called “numOfThreadys”, which is initialized to 1, and a static Object variable called “ob”, which holds a reference to a new Object instance.

In the “main” method first an object of type Object is created and its reference is put into a variable called “locky”, but it is never used in the rest of the code here, so it is irrelevant. Then an array of type Thread is created, and its length is set to be 50. After this there is a “for” loop that creates 50 new Thready objects and places their references in the array. After each object is created the “start()” method is called on it, which calls the “run()” method. Each time the “run()” method is called it prints out the value of the static class variable “numberOfThreadys” and then increases the value of this variable by 1. Since the “start()” method is called on each of the 50 Thready objects, the “numberOfThreadys” variable value will be increased 50 times, and 50 numbers will be printed to the screen.

In this example, since the “ob” variable is static, there is only 1 instance of the object that it refers to for the Thready class. Therefore the synchronized line, which is passed “ob” as a parameter, is being passed the same object that “ob” refers to each time called. So only one of the Thready objects can access the synchronized code at a time.

As a result of this the value in the “numberOfThreadys” variable is printed and increased in order, one at a time, so the numbers 1 to 50 are printed out in order.

23. B

    The code compiles fine but does not do anything. Since a “Tara” object is a subclass of Thread, it automatically has a default run() method, which is empty. When start() is called for each of these threads, the default empty run() is called, so it runs but nothing happens.

24. A

    In this example there are 2 “Tara” objects created, which are actually Thread objects since the Tara class is a subclass of Thread. Then 2 Thread objects are created, t1 and t2, and they are each assigned the reference of one of the Tara objects. This assignment is fine, since a Tara object is a subclass of a Thread object. The variable “lucky” is an instance variable, so each Tara object has its own version of “lucky”.

    Then the start() method is called for each thread. Since the actual object that the references t1 and t2 refer to are Tara objects, the run() method of Tara is called once for each of the threads. Since the “lucky” variable is an instance variable, and run() is called twice, once for each thread, the numbers 1 to 100 are printed out twice.

    Since there are 2 Tara objects here, the “this” in the synchronized code is a different reference for each of the 2 objects, so each of them runs the “run()” method independently, and at the same time. Therefore there is no way to know which thread will print first, second, etc., but they will each print the 100 numbers in order. The synchronized code actually has no effect at all in this example.

25. A

    An array of Thread objects is created, that can hold 10 Thread objects. This array is filled with Tara objects, which is legal since the Tara class is a subclass of the Thread class. The Tara class has a static int variable called “lucky” that is initialized to 1.

    When start() is called on each thread, the run() method is called. Inside the run() method there is synchronized code that accepts the current object as a parameter. Since each Tara object is a separate object, the run() method is called 10 times, and there are 10 objects working with the synchronized code at the same time. Since the variable “lucky” is a static class variable, each of the Tara objects is printing the value of the same variable in the run() code. And since the synchronized code is running many times at the same time, and different objects are increasing the value of “lucky” and then checking if its value is greater than 25, all at overlapping times, there is no way to guarantee how many times the value in the “lucky” variable will be printed. Once the value of “lucky” does reach 25 though, soon later the code will break for all of the 10 times that is has run.

26. B

    The start() method of a thread can only be called once for each Thread object. In order to call the run() again, a new Thread object needs to be created.

27. D

    Since the variable “lupu” is static, the same variable is being accessed from the “for” loop in “main” and in the “for” loop in “run”. There is no way to know the order in which each of these loops is accessing the “lupu” variable, so the number of the times that the word “TOPAZ” will be printed to the screen can vary.

28. C

    This example has a class called Magic, which extends from Thread. This Magic class holds a static int variable called "num" that starts out with a value of 0. In the "main" method an array of threads is created with a length of 10, and 10 new Magic objects are created and their references are placed into the array. Then the method start() is called on each of these Magic threads, so the "while" loop in the run() method will be running many times concurrently.

    The code in the run() method prints out the value in the static variable num, and then increases the value of num by 1. Since this code may be running many times at once, and since the " while" loop in the "run" method has it's condition based on the static variable that is being increased by many threads, there is no way to know exactly how many numbers will be printed.  But it is certain that less than 1000 numbers will be printed since in the "while" loop parenthesis the condition "num<100" will have each thread stop running when the 100 value is hit. And since "num" is a static variable all of the threads are referring to the same "num", so the "num" value will never even get close to 1000.

29. E

    This example has a class called Magic, which extends from Thread. This Magic class holds an instance variable of type int called "num" that starts out with a value of 0. This class also defines a method called "run", but this method accepts an int as a parameter so it does not override the "run" method of Thread. In "main" when start() is called on each of the threads in the array, the default no-constructor "run" method of the Thread class is called, and not the new run method that accepts an argument. Therefore the value of num never changes and remains as 0, so that is what is printed.

30. A

    This example has a class called Magic, which extends from Thread. This Magic class holds an instance variable called "num" of type int. In the "main" method an array of threads is created with a length of

10, and 10 new Magic objects are created and their references are placed into the array. Then the method start() is called on each of these Magic threads, so the “while” loop in the run() method will be running many times concurrently.  Since each of these 10 Magic objects has  its own copy of the variable “num”, each one prints out exactly 100 numbers, so in the end exactly 1000 numbers are printed.

31. A

    The class “ThreadDemo1” is a subclass of Thread, and redefines the method run(). Two “ThreadDemo1” objects are created and start() is called on both of them, which calls their run() method. When each of these threads is created, it passes in a String to print in the run() method, and an int, which is used in run() to determine the number of times to print the String passed.

    Since the int variable “count” is a static variable, only one copy of this variable exists for the class. When the first thread is created, this variable is set to the value of 3, but then when the second thread it created it is set to the value of 6. Since both classes access the same variable, since the 6 was the last number defined to “count”, each object prints out the text 6 times, which makes a total of 12.

32. C

    The class “ThreadDemo1” is a subclass of Thread, and redefines the method run(). Two “ThreadDemo1” objects are created and start() is called on both of them, which calls their run() method. When each of these threads is created, it passes in a String to print in the run() method, and an int, which is used in run() to determine the number of times to print the String passed.  The int variable “count” is an instance variable, so each of the objects contains its own “count”. Both times the word “GEZER” is passed as the String to be printed. One object is passed a 3 and the other a 6, so the String “GEZER” is printed out a total of 9 times here.