EXERCISES

1. Write a Java program to demonstrate that as a high-priority thread executes, it will delay the execution of all lower priority threads.

2. If your system supports timeslicing, write a Java program that demonstrates timeslicing among several equal-priority threads. Show that a lower priority thread's execution is deferred by the timeslicing of the higher-priority threads.

3. Write a Java program that demonstrates a high priority thread using **sleep** to give lower priority threads a chance to run.

4. If your system does not support timeslicing, write a Java program that demonstrates two threads using **yield** to enable one another to execute.

5. Two problems that can occur in systems like Java, that allow threads to wait, are deadlock, in which one or more threads will wait forever for an event that cannot occur, and indefinite postponement, in which one or more threads will be delayed for some unpredictably long time. Give an example of how each of these problems can occur in a multithreaded Java program.

6. (Readers and Writers) This exercise asks you to develop a Java monitor to solve a famous problem in concurrency control. This problem was first discussed and solved by P. J. Courtois, F.Heymans and D. L. Parnas in their research paper, "Concurrent Control with Readers and Writers,"*Communications of the ACM*, Vol. 14, No. 10, October 1971, pp. 667–668. The interested student might also want to read C. A. R. Hoare's seminal research paper on monitors, "Monitors: An Operating System Structuring Concept," *Communications of the ACM*, Vol. 17, No. 10, October 1974, pp.549–557. Corrigendum, *Communications of the ACM*, Vol. 18, No. 2, February 1975, p. 95. [The readers and writers problem is discussed at length in Chapter 5 of the author's book: Deitel, H. M.,*Operating Systems*, Reading, MA: Addison-Wesley, 1990.]

a) With multithreading, many threads can access shared data; as we have seen, access to shared data needs to be synchronized carefully to avoid corrupting the data.

b) Consider an airline-reservation system in which many clients are attempting to book seats on particular flights between particular cities. All of the information about flights and seats is stored in a common database in memory. The database consists of many entries, each representing a seat on a particular flight for a particular day between particular cities. In a typical airline-reservation scenario, the client will probe around in the database looking for the "optimal" flight to meet that client's needs. So a client may probe the database many times before deciding to book a particular flight. A seat that was available during this probing phase could easily be booked by someone else before the client
has a chance to book it. In that case, when the client attempts to make the reservation, the client will discover that the data has changed and the flight is no longer available.

c) The client probing around the database is called a reader. The client attempting to book the flight is called a writer. Clearly, any number of readers can be probing shared data at once, but each writer needs exclusive access to the shared data to prevent the data from being corrupted.

d) Write a multithreaded Java program that launches multiple reader threads and multiple writer threads, each attempting to access a single reservation record. A writer thread has two possible transactions, **makeReservation** and **cancelReservation**. A reader has one possible transaction, **queryReservation**.

e) First implement a version of your program that allows unsynchronized access to the reservation record. Show how the integrity of the database can be corrupted. Next implement a version of your program that uses Java monitor synchronization with **wait** and **notify** to enforce a disciplined protocol for readers and writers accessing the shared reservation data. In particular, your program should allow multiple readers to access the shared data simultaneously when no writer is active. But if a writer is active, then no readers should be allowed to access the shared data.

f) Be careful. This problem has many subtleties. For example, what happens when there are several active readers and a writer wants to write? If we allow a steady stream of readers to arrive and share the data,

they could indefinitely postpone the writer (who may become tired of waiting and take his or her business elsewhere). To solve this problem, you might decide to favor writers over readers. But here, too, there is a trap, because a steady stream of writers could then indefinitely postpone the waiting readers, and they, too, might choose to take their business elsewhere! Implement your monitor with the following methods: **startReading**, which is called by any reader who wants to begin accessing a reservation; **stopReading** to be called by any reader who has finished reading a reservation; **startWriting** to be called by any writer who wants to make a reservation and **stopWriting** to be called by any writer who has finished making a reservation.

7. Write a program that bounces a blue ball inside an applet. The ball should be initiated with a **mousePressed** event. When the ball hits the edge of the applet, the ball should bounce off the edge and continue in the opposite direction.

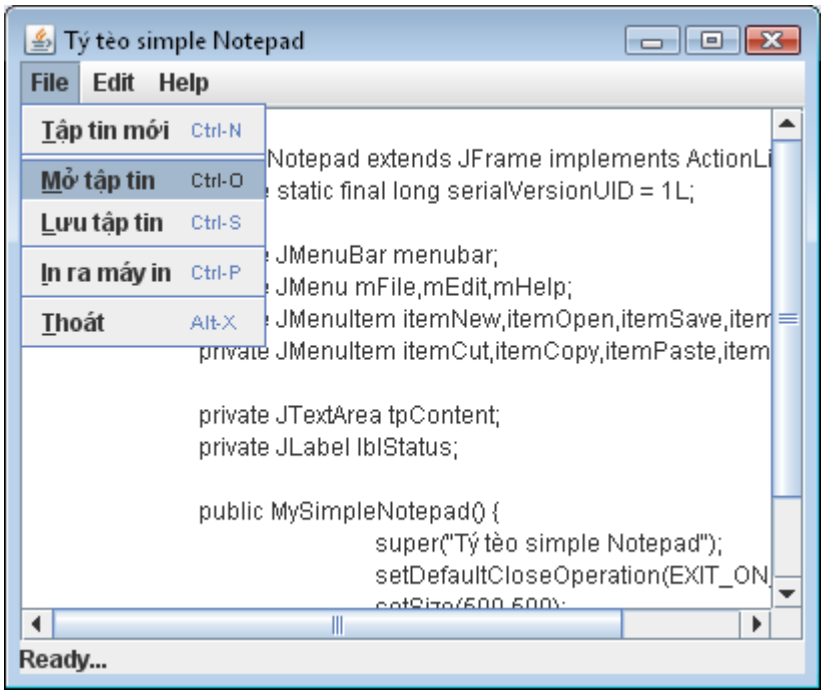
8. Modify the program of Exercise 7 to add a new ball each time the user clicks the mouse. Provide for a minimum of 20 balls. Randomly choose the color for each new ball.

9. Modify the program of Exercise 8 to add shadows. As a ball moves, draw a solid black oval at the bottom of the applet. You may consider adding a 3-D effect by increasing or decreasing the size of each ball when a ball hits the edge of the applet.

10. Modify the program of Exercise 8 or 9 to bounce the balls off each other when they collide.

11. Inherit a class from **Thread** and override the **run( )** method. Inside **run( )**, print a message, then call **sleep( )**. Repeat this three times, then return from **run( )**. Put a startup message in the constructor and override **finalize( )** to print a shut-down message.
Make a separate thread class that calls **System.gc( )** and **System.runFinalization( )** inside **run( )**, printing a message as it does so. Make several thread objects of both types and run them to see what happens.

12. Create three dasses: Storage, Counter and Printer. The Storage class should store an integer. The Counter class should create a thread that starts counting from 0 (0, 1, 2. 3 ...) and stores each value in the Storage class. The Printer class should create thread that keeps reading the value in the Storage class and printing it.
Create a program that creates an instance of the Storage class, and sets up a Counter and a Printer object to operate on it.
13. Modify the program from the previous exercise, to ensure that each number is printed exactly once, by adding suitable synchronization.
14. Write an application as an simple notepad demo how to use multithreading loading file. Main GUI as follow


Tý tèo simple Notepad
File
Edit
Help
Tập tin mới Ctrl-N
Mở tập tin Ctrl-O
Lưu tập tin Ctrl-S
In ra máy in Ctrl-P
Thoát Alt-X
Notepad extends JFrame implements ActionLi
static final long serialVersionUID = 1L;
JMenuBar menubar;
JMenu mFile,mEdit,mHelp;
JMenuItem itemNew,itemOpen,itemSave,item
private JMenuItem itemCut,itemCopy,itemPaste,item
private JTextArea tpContent;
private JLabel lblStatus;
public MySimpleNotepad() {
super("Tý tèo simple Notepad");
setDefaultCloseOperation(EXIT_ON
Ready...

Review Questions

1. List each of the three reasons given in the text for entering the blocked state. For each of these, describe how the program will normally leave the blocked state and enter the runnable state.

2. Distinguish between preemptive scheduling and nonpreemptive scheduling. Which does Java use?

3. What is timeslicing? Give a fundamental difference in how scheduling is performed on Java systems that support timeslicing vs. scheduling on Java systems that do not support timeslicing.

4. Why would a thread ever want to call **yield**?

5. What aspects of developing Java applets for the World Wide Web encourage applet designers to use **yield** and **sleep** abundantly?

6. If you choose to write your own **start** method, what must you be sure to do to ensure that your threads start up properly?

7. Distinguish among each of the following means of pausing threads:
a) busy wait
b) sleep
c) blocking I/O

8. Write a Java statement that tests if a thread is alive.

9. a) What is multiple inheritance?
b) Explain why Java does not offer multiple inheritance.
c) What feature does Java offer instead of multiple inheritance?
d) Explain the typical use of this feature.
e) How does this feature differ from **abstract** classes?

10. Distinguish between the notions of **extends** and **implements**.

11. Discuss each of the following terms in the context of monitors:
a) monitor
b) producer
c) consumer
d) wait
e) notify
f) InterruptedException
g) synchronized

12. (Tortoise and the Hare) In the Chapter 7 exercises, you were asked to simulate the legendary race of the tortoise and the hare. Implement a new version of that simulation, this time placing each of the animals in a separate thread. At the start of the race call the **start** methods for each of the threads. Use **wait**, **notify** and **notifyAll** to synchronize the animals' activities.

13. (Multithreaded, Networked, Collaborative Applications) In Chapter 17, we cover networking in Java. A multithreaded Java application can communicate concurrently with several host computers. This creates the possibility of being able to build some interesting kinds of collaborative applications. In anticipation of studying networking in Chapter 17, develop proposals for several possible multithreaded networked applications. After studying Chapter 17, implement some of those applications.

14. Which of these events will cause a thread to die?
Select all valid answers.
(a) The method sleep() is called.
(b) The method wait() is called.
(c) Execution of the start() method ends.
(d) Execution of the run() method ends.
(e) Execution of the thread's constructor ends.

15. What can be guaranteed by calling the method yield 0?
Select the one right answer.
(a) All lower priority threads will be granted CPU time.
(b) The current thread will sleep for some time while some other threads are doing some work.
(c) The current thread will not continue until other threads are finished with their work.
(d) The thread will sleep until it is notified.
(e) None of the above.

16. Where is the notify() method defined?
Select the one right answer.
(a) Thread
(b) Object
(c) Applet
(d) Runnable

17. What wifi calling the noti fy() method on an object implementing Runnable achieve? Select the one right answer.
(a) Will cause the thread executing the run() method of the object to continue.
(b) Will cause a thread that called the wait() method while owning the monitor of the object to be enabled for running.
(c) Will cause all the threads waiting for the monitor of the object to be enabled for running.
(d) Will cause an Illegal Moni torStateException to be thrown.
(e) None of the above.

18. How can you set the priority of a thread?
Select the one right answer.
(a) Using the setPriority() method in the class Thread.
(b) Give the priority as a parameter to the constructor of the thread.
(c) Both of the above.
(d) None of the above.

19. What will be the result of writing a method that attempts to call wait() without ensuring that the current thread owns the monitor of the object?
Select the one right answer.
(a) The code will fail to compile.
(b) Nothing special will happen.
(c) An IllegalMonitorStateException will be thrown whenever the method is called.
(d) An IllegalMorritorStateException will be thrown if the method is called at a time when the current thread does not have the monitor of the object.
(e) The thread will be blocked until it gains the monitor of the object.

20. Which of these are plausible reasons why a thread might be alive, but still not be running?
Select all valid answers.
(a) The thread is waiting for some condition as a result of a call to wait 0.
(b) The thread is waiting on a monitor for an object so that it may access a certain member variable of that object.

(c) The thread is not the highest priority thread and is currently not granted CPU time.
(d) The thread is sleeping as a result of a call to the sleep() method.

21. Which one of these events will cause a thread to die?
Select the one correct answer.

a. The method `sleep()` is called.
b. The method `wait()` is called.
c. Execution of the `start()` method ends.
d. Execution of the `run()` method ends.
e. Execution of the thread's constructor ends.

22. Which statements are true about the following code?

```
public class Joining {
    static Thread createThread(final int i, final Thread t1) {
        Thread t2 = new Thread() {
            public void run() {
                System.out.println(i+1);
                try {
                    t1.join();
                } catch (InterruptedException e) {
                }
                System.out.println(i+2);
            }
        };
        System.out.println(i+3);
        t2.start();
        System.out.println(i+4);
        return t2;
    }

    public static void main(String[] args) {
        createThread(10, createThread(20, Thread.currentThread()));
    }
}
```

Select the two correct answers.

a. The first number printed is `13`.
b. The number `14` is printed before the number `22`.
c. The number `24` is printed before the number `21`.
d. The last number printed is `12`.
e. The number `11` is printed before the number `23`.

23. What can be guaranteed by calling the method `yield()`?

Select the one correct answer.

a. All lower priority threads will be granted CPU time.
b. The current thread will sleep for some time while some other threads run.
c. The current thread will not continue until other threads have terminated.
d. The thread will wait until it is notified.
e. None of the above.

24. Where is the `notify()` method defined?

Select the one correct answer.

a. `Thread`
b. `Object`
c. `Applet`
d. `Runnable`

25. How can the priority of a thread be set?

Select the one correct answer.

a. By using the `setPriority()` method in the class `Thread`.
b. By passing the priority as a parameter to the constructor of the thread.
c. Both of the above.
d. None of the above.

26. Which statements are true about locks?

Select the two correct answers.

a. A thread can hold more than one lock at a time.
b. Invoking `wait()` on a `Thread` object will relinquish all locks held by the thread.
c. Invoking `wait()` on an object whose lock is held by the current thread will relinquish the lock.
d. Invoking `notify()` on a object whose lock is held by the current thread will relinquish the lock.
e. Multiple threads can hold the same lock at the same time.

27. What will be the result of invoking the `wait()` method on an object without ensuring that the current thread holds the lock of the object?

Select the one correct answer.

a. The code will fail to compile.
b. Nothing special will happen.
c. An `IllegalMonitorStateException` will be thrown if the `wait()` method is called while the current thread does not hold the lock of the object.
d. The thread will be blocked until it gains the lock of the object.

28. Which of these are plausible reasons why a thread might be alive, but still not be running?

Select the four correct answers.

a. The thread is waiting for some condition as a result of a `wait()` call.
b. The execution has reached the end of the `run()` method.

c. The thread is waiting to acquire the lock of an object in order to execute a certain method on that object.
d. The thread does not have the highest priority and is currently not executing.
e. The thread is sleeping as a result of a call to the `sleep()` method.

29. Which is the correct way to start a new thread?

Select the one correct answer.

a. Just create a new `Thread` object. The thread will start automatically.
b. Create a new `Thread` object and call the method `begin()`.
c. Create a new `Thread` object and call the method `start()`.
d. Create a new `Thread` object and call the method `run()`.
e. Create a new `Thread` object and call the method `resume()`.

30. When extending the `Thread` class to provide a thread's behavior, which method should be overridden?

Select the one correct answer.

a. `begin()`
b. `start()`
c. `run()`
d. `resume()`
e. `behavior()`

31. Which statements are true?

Select the two correct answers.

a. The class `Thread` is `abstract`.
b. The class `Thread` implements `Runnable`.
c. Classes implementing the `Runnable` interface must define a method named `start`.
d. Calling the method `run()` on an object implementing `Runnable` will create a new thread.
e. A program terminates when the last non-daemon thread ends.

32. What will be the result of attempting to compile and run the following program?

```
public class MyClass extends Thread {
    public MyClass(String s) { msg = s; }
    String msg;
    public void run() {
        System.out.println(msg);
    }

    public static void main(String[] args) {
        new MyClass("Hello");
        new MyClass("World");
    }
}
```

Select the one correct answer.

a. The program will fail to compile.
b. The program will compile without errors and will print `Hello` and `World`, in that order, every time the program is run.
c. The program will compile without errors and will print a never-ending stream of `Hello` and `World`.
d. The program will compile without errors and will print `Hello` and `World` when run, but the order is unpredictable.
e. The program will compile without errors and will simply terminate without any output when run.