Multithreading & Network programming
Threads

• Program units that execute independently; multiple threads run “simultaneously”

• Virtual machine executes each thread for short time slice
  – Thread scheduler activates/deactivates threads
  – Illusion of threads running in parallel

• Multiprocessor computers: threads actually do run in parallel
Threads vs. processes

- Processes isolated from each other by operating system
  - Promotes safety
  - Makes switching slow

- Threads run within single process
  - Fast switching
  - Multiple threads share memory, can corrupt each other’s data
Running threads

• Define class that implements Runnable interface:
  
  ```java
  public interface Runnable
  {
      void run();
  }
  ```

• Place code for task in class’s run method
• Create object of the class
• Construct a Thread object, supplying Runnable object as argument
• Call Thread object’s start() method
Example

public class MyRunnable implements Runnable
{
    public void run()
    {
        // thread action goes here
    }
}

Runnable r = new MyRunnable();
Thread t = new Thread(r);
t.start();
Static method Thread.sleep()

- Every thread should occasionally yield control to other threads
- Otherwise, have selfish thread – could prevent other threads from making progress
- Thread.sleep() puts current thread to sleep for specified number of milliseconds
- A sleeping thread will throw an InterruptedException if terminated; so code using sleep() needs to handle such an exception
Expanded example

```java
public class MyRunnable implements Runnable {
    public void run() {
        try {
            // thread action goes here
            Thread.sleep(50);
        } catch (InterruptedException e) {
        }
    }
}

Runnable r = new MyRunnable();
Thread t = new Thread(r);
t.start();
```
Methods start() and run()

• Thread class has start() method that creates a new thread in the JVM
• The started thread calls the Runnable object’s run() method
• The thread terminates when run() method either returns or throws an uncaught exception
Running threads in parallel

- Construct and start two or more Thread objects
- `main()` method runs in its own thread; may terminate before the threads it starts
- Program doesn’t end until all threads terminate
Example

public class ThreadTest
{
    public static void main(String[] args)
    {
        Runnable r1 = new myRunnable();
        Runnable r2 = new myRunnable();
        ...
        Runnable rn = new myRunnable();
        Thread t1 = new Thread(r1);
        Thread t2 = new Thread(r2);
        ...
        Thread tn = new Thread(rn);
    }
}
**Subclassing Thread**

- Can extend Thread rather than implementing Runnable, if desired:
  ```java
  class someClass extends Thread {
    public void run() {
      try {
        while (something to do) {
          // do work
          sleep(50);
        }
      } catch (InterruptedException e) {} 
    }
  }
  ```
Scheduling threads

• Thread scheduler gives no guarantee about order of thread execution
• Running times vary slightly, depending on what else is going on in the system
• Order in which each thread gains control is somewhat random
Thread states

• Each thread has a state & a priority
• Possible states are:
  – new (before start() is called)
  – runnable
  – blocked
  – dead (after run() exits)
Blocked threads

• Thread enters a blocked state for several reasons; stays blocked until event it is waiting for occurs

• Reasons for entering blocked state include:
  – sleeping
  – waiting for I/O
  – waiting to acquire a lock
  – waiting for a notification
Scheduling threads

• Scheduler will activate new thread in each of the following cases:
  – thread completes its time slice
  – thread has blocked itself
  – thread with higher priority becomes runnable

• Scheduler chooses highest priority threads among those that are runnable

• Scheduling algorithm system-dependent

• Priority values not usually under application programmer’s control
Terminating threads

• Thread terminates automatically when run() method of its Runnable object returns

• Can terminate running thread “manually” by calling method interrupt()
  – stops thread
  – releases system resources
Checking for interrupted state

- run() method should occasionally check if its thread is interrupted
- Can use isInterrupted() method on current thread object; but sleeping thread can’t execute this (because it’s sleeping) – so sleep() method terminates with an InterruptedException if sleeping thread is interrupted – this is why code containing call to sleep() should be surrounded by try/catch block
Thread synchronization

• Threads that share access to a common object can conflict with each other
• Can result in corruption of a data structure
• Race condition
  – effect of multiple threads on shared data depends on order of thread execution
  – end result (normal vs. corrupted data structure) depends on which thread wins the race
• Need to ensure that only one thread manipulates shared structure at any given moment
Object locks

- To prevent problems like those caused by race conditions, thread can be set up to lock an object.
- While object is locked, no other thread can lock same object; attempt to do so temporarily blocks other thread.
- In Java, preserve object integrity by tagging sensitive methods with the **synchronized** keyword.
Deadlock

• Occurs if no thread can proceed because every thread is waiting for another to do some work first
• Common scenario: thread needs to lock an object before checking whether an action can be carried out, then needs to wait to see if check fails
• Can use wait() method to temporarily block current thread and release object lock; current thread is added to set of threads waiting for object access
Unblocking waiting thread

• When a thread calls wait(), it remains blocked until another thread executes the notifyAll() method
• notifyAll() unblocks all threads waiting for an object
• notifyAll() should be called whenever state of object changes in a way that might benefit waiting threads
Threads & Animation

• Animation: shows object moving or changing as time progresses
• Simple example: animated display of graphical file
  – reads series of GIFs into array of images
  – paint routine selects one to display, then update index so next image will be selected
  – calls Thread.sleep() to display current image for set amount of time
  – calls repaint() to ensure next image is displayed
Algorithm animation

- Thread runs algorithm, updates display then sleeps
- After brief rest, wakes up again, runs to next point of interest in algorithm, updates display, sleeps again
- Sequence repeats until algorithm finishes
Example: Mergesort animation

- Uses MergeSorter class to sort random integer array
- MergeSorter.sort takes an array and a Comparator as arguments
- For demo, supply Sorter class that implements Runnable
  - run() method calls MergeSorter.sort()
  - Comparator supplied to sort() contains a calls to sleep() – pauses thread long enough to see progress of algorithm in display
Comparator - pseudocode

Comparator comp = new Comparator()
{
    public int compare (Object o1, Object o2)
    {
        draw array contents
        pause thread
        return ((Integer) o1).compareTo(o2);
    }
};
Improvement – allow animation to pause until button is clicked

• Need to coordinate user interface thread & algorithm thread

• Use “gate” class to coordinate threads:
  – Step button opens gate, allows one operation through
  – Run button deactivates gate
Animation classes
Improved compare() method

public int compare(Object o1, Object o2) {
    
    ... 
    if (gate.isActive())
        gate.waitForOpen();
    else
        Thread.sleep(Delay);
    
    ... 
    return ((Integer) o1).compareTo(o2);
}
Client/Server connections

• Port: location, designated by integer number, where information can be exchanged between client & server
  – Port values less than 1024 typically reserved for predefined services (ftp, telnet, email, http, etc.)
  – User-defined ports use larger values
Client/Server connections

• Socket: combination of IP address & port
  – socket is where client “plugs in” to server, creating connection for flow of information
  – sockets also provide facilities for creating I/O streams
DateServer application

- Simple example of client/server interaction
- Establishes connection at client’s request
- Sends current data & time to client for output
Notes on DateServer

- Several methods (accept(), write() and ServerSocket()) can all throw an IOException; can simplify things by checking exceptions in the constructor interface, then handle in main(); exceptions thrown by library procedures pass up through constructor to calling procedure (main())
Notes on DateServer application

• Method accept() returns a Socket when the client makes a request
• The Socket is then used to create an output stream
• After making the connection, the server sends a message (this is the requested service) to the client
Notes on DateClient

• The client requests a socket to be created at given port on specific computer; in this example, we assume client & server are on same computer

• The IP address on which the application is running is accessed by method InetAddress.getLocalHost()

• A more general method would be: InetAddress.getByName(domainName), which takes the string representation (FQDN) and converts it into an IP address
Notes on DateClient

• Once created, a socket can be used to create an input stream, which can be converted to a BufferedReader, which provides a method (readLine()) for reading an entire line of input, which can then be printed out

• The DateServer/DateClient interaction is a simple example of a service provided to one client at a time, with one-way communication
Client/Server interaction with multiple clients

- Therapist class simulates a therapist, conducting a question/answer session with a client; works by the following rules:
  - answer any question with “Why do you want to know?”
  - answer any statement that begins with “I feel” with “Why do you feel that way?”
  - answer any statement that mentions a relative with “Tell me about your” name of relative
  - if none of the above apply, respond with “Tell me more”
TherapySession class

• In this program, several clients can be served simultaneously because each initial client request spawns a new Thread

• Actual service is provided via an instance of TherapySession, which gets both input and output streams passed to it - this permits 2-way communication between Client & Server
TherapySession class

- TherapySession extends Thread, so most of its action takes place in run():
  - prints generic greeting
  - flush is used to transfer output across network, rather than let buffering process wait for more output
  - loop reads a line of text, then determines and writes response
TherapySession class

• Other methods include:
  – constructor: converts I/O streams to buffered reader and writer to simplify I/O processing
  – response method: implements rules for Therapist; returns appropriate String based on user input
  – isRelative: used to determine if a relative’s name appears in text
TherapyClient class

• Creates readers & writers to handle socket I/O
• Reads responses from stdin and passed to server, then writes replies to stdout