Ch.4 Internet Addresses

Unit-2

Outline

4.1 The InetAddress Class

4.2 Inet4Address and Inet6Address

4.3 The NetworkInterface Class

4.4 Some Useful Programs

Internet Addresses

- IP (Internet Protocol) Addresses
  - IPv4 (4 Bytes): dotted quad format
    - www.csie.nuk.edu.tw 140.127.208.17
  - IPv6 (16 Bytes): 8 blocks of 4 hexadecimal digits separated by colons
    - www.csie.nuk.edu.tw **::ffff**:8c7f:d011
    - 2400:cb00:2048**:000**1**:0000:0000:**6ca2:c665 → 2400:cb00:2048**:**1**::**6ca2:c665
  - Mixed: last 4 bytes of the IPv6 written as an IPv4 dotted quad address
    - www.nuk.edu.tw ::ffff:140.127.208.17
    - FEDC:BA98:7654:3210:FEDC:BA98:7654:3210
      FEDC:BA98:7654:3210:FEDC:BA98:118.84.50.16
- Domain Names – Resolved by DNS Servers
  - FQDN: Fully Qualified Domain Name
    - www.csie.nuk.edu.tw.
  - One name can map to multiple IP addresses
  - One IP addresses can also have multiple names

4.1 The InetAddress Class

http://docs.oracle.com/javase/8/docs/api/java/net/InetAddress.html

- Creating new InetAddress objects
  - No public constructors; use static factory methods directly
    - Automatically connect to a DNS server to resolve a hostname
    - Throws an UnknownHostException, a subclass of IOException, if not found
- getByName(): lookup the name and the numeric address

```
InetAddress address = InetAddress.getByName("www.oreilly.com");
InetAddress address = InetAddress.getByName("208.201.239.100");
```

Example 4-1. A program that prints the address of www.oreilly.com

```
import java.net.*;

public class OReillyByName {

  public static void main (String[] args) {
    try {
      InetAddress address = InetAddress.getByName("www.oreilly.com");
      System.out.println(address);
    } catch (UnknownHostException ex) {
      System.out.println("Could not find www.oreilly.com");
    }
  }
}
```

```
% java OReillyByName
www.oreilly.com/208.201.239.36
```

- getAllByName(): lookup all the addresses of a host

```
try {
  InetAddress[] addresses = InetAddress.getAllByName("www.oreilly.com");
  for (InetAddress address : addresses) {
    System.out.println(address);
  }
} catch (UnknownHostException ex) {
  System.out.println("Could not find www.oreilly.com");
}
```

```
www.google.com/173.194.72.147
www.google.com/173.194.72.105
www.google.com/173.194.72.104
www.google.com/173.194.72.99
www.google.com/173.194.72.103
www.google.com/173.194.72.106
```

getLocalHost() and getByAddress()

- getLocalHost(): return an InetAddress object for the local host
  - Return ‘localhost/127.0.0.1’ if lookup failed

Example 4-2. Find the address of the local machine

```
import java.net.*;

public class MyAddress {

  public static void main (String[] args) {
    try {
      InetAddress address = InetAddress.getLocalHost();
      System.out.println(address);
    } catch (UnknownHostException ex) {
      System.out.println("Could not find this computer's address.");
    }
  }
}
```

```
% java MyAddress
titan.oit.unc.edu/152.2.22.14
```

- getByAddress(): create an InetAddress object from given address
  - Without talking to DNS

```
public static InetAddress getByAddress(byte[] addr) throws UnknownHostException
public static InetAddress getByAddress(String hostname, byte[] addr)
    throws UnknownHostException
```

  - Example

```
byte[] address = {107, 23, (byte) 216, (byte) 196};
InetAddress lessWrong = InetAddress.getByAddress(address);
InetAddress lessWrongWithname = InetAddress.getByAddress(
   "lesswrong.com", address);
```

Caching and Security Issues

- The InetAddress class caches the results of lookups
  - Java only caches unsuccessful DNS queries for 10 seconds by default
  - Times can be controlled and specified in Java security properties
    - networkaddress.cache.ttl: specifies the number of seconds a successful DNS lookup will remain in Java's cache
    - networkaddress.cache.negative.ttl: specifies the number of seconds an unsuccessful lookup will be cached

- Security Issues: a DNS lookup generates network traffic
  - Untrusted code
    - Prohibition against making network connections to hosts other than the codebase
    - An untrusted applet under the control of the default security manager will only be allowed to get the IP address of the host it came from (its codebase) and possibly the local host
  - Relaxed for trusted code
  - checkConnect(): test whether a host can be resolved

InetAddress: Create Objects and Getter Methods

static InetAddress[]	getAllByName(String host) Given the name of a host, returns an array of its IP addresses, based on the configured name service on the system
static InetAddress	getByAddress(byte[] addr) Returns an InetAddress object given the raw IP address
static InetAddress	getByAddress(String host, byte[] addr) Creates an InetAddress based on the provided host name and IP address
static InetAddress	getByName(String host) Determines the IP address of a host, given the host's name
static InetAddress	getLocalHost() Returns the address of the local host
static InetAddress	getLoopbackAddress() Returns the loopback address
byte[]	getAddress() Returns the raw IP address of this InetAddress object
String	getCanonicalHostName() Gets the fully qualified domain name for this IP address
String	getHostAddress() Returns the IP address string in textual presentation
String	getHostName() Gets the host name for this IP address

Examples – Find Hostname, IP and Version

Example 4-3. Given the address, find the hostname

```
import java.net.*;

public class ReverseTest {

  public static void main (String[] args) throws UnknownHostException {
    InetAddress ia = InetAddress.getByName("208.201.239.100");
    System.out.println(ia.getCanonicalHostName());
  }
}
```

```
% java ReverseTest
oreilly.com
```

Example 4-4. Find the IP address of the local machine

```
import java.net.*;

public class MyAddress {

  public static void main(String[] args) {
    try {
      InetAddress me = InetAddress.getLocalHost();
      String dottedQuad = me.getHostAddress();
      System.out.println("My address is " + dottedQuad);
    } catch (UnknownHostException ex) {
      System.out.println("I'm sorry. I don't know my own address.");
    }
  }
}
```

```
% java MyAddress
My address is 152.2.22.14.
```

Example 4-5. Determining whether an IP address is v4 or v6

```
import java.net.*;

public class AddressTests {

  public static int getVersion(InetAddress ia) {
    byte[] address = ia.getAddress();
    if (address.length == 4) return 4;
    else if (address.length == 16) return 6;
    else return -1;
  }
}
```

Address Types

boolean	isAnyLocalAddress()	Utility routine to check if the InetAddress in a wildcard address (0.0.0.0 / ::)
boolean	isLinkLocalAddress()	Utility routine to check if the InetAddress is an IPv6 link local address (Begin with FE80:0000:0000:0000 (8 Bytes) + Local address (often MAC))
boolean	isLoopbackAddress()	Utility routine to check if the InetAddress is a loopback address (127.0.0.1 / ::1)
boolean	isMCGlobal()	Utility routine to check if the multicast address has global scope (IPv4-all Multicast/IPv6-begin with FF0E or FF1E)
boolean	isMCLinkLocal()	Utility routine to check if the multicast address has subnet/link scope (IPv4-all Multicast/IPv6-begin with FF02 or FF12)
boolean	isMCNodeLocal()	Utility routine to check if the multicast address has node scope (for test) (IPv4-all Multicast/IPv6-begin with FF01 or FF11)
boolean	isMCOrgLocal()	Utility routine to check if the multicast address has organization scope (IPv6-begin with FF08 or FF18)
boolean	isMCSiteLocal()	Utility routine to check if the multicast address has site scope (IPv6-begin with FF05 or FF15)
boolean	isMulticastAddress()	Utility routine to check if the InetAddress is an IP multicast address (224.0.0.0~239.255.255.255 / FF00::)
boolean	isReachable(int timeout)	Test whether that address is reachable (Use traceroute/ICMP echo requests)
boolean	isReachable(NetworkInterface netif, int ttl, int timeout)	Test whether that address is reachable
boolean	isSiteLocalAddress()	Utility routine to check if the InetAddress is a IPv6 site local address Like LinkLocalAddress, but May be forwarded by routers (Begin with EEC0:0000:0000:0000 (8 Bytes) + Local address (often MAC))

Example 4-6. Testing characteristics of an IP

```
import java.net.*;

public class IPCharacteristics {

  public static void main(String[] args) {

    try {
      InetAddress address = InetAddress.getByName(args[0]);

      if (address.isAnyLocalAddress()) {
        System.out.println(address + " is a wildcard address.");
      }
      if (address.isLoopbackAddress()) {
        System.out.println(address + " is loopback address.");
      }
      if (address.isLinkLocalAddress()) {
        System.out.println(address + " is a link-local address.");
      } else if (address.isSiteLocalAddress()) {
        System.out.println(address + " is a site-local address.");
      } else {
        System.out.println(address + " is a global address.");
      }

      if (address.isMulticastAddress()) {
        if (address.isMCGlobal()) {
          System.out.println(address + " is a global multicast address.");
        } else if (address.isMCOrgLocal()) {
          System.out.println(address
           + " is an organization wide multicast address.");
        } else if (address.isMCSiteLocal()) {
          System.out.println(address + " is a site wide multicast
                             address.");
        } else if (address.isMCLinkLocal()) {
          System.out.println(address + " is a subnet wide multicast
                             address.");
        } else if (address.isMCNodeLocal()) {
          System.out.println(address
           + " is an interface-local multicast address.");
        } else {
          System.out.println(address + " is an unknown multicast
                             address type.");
        }
      } else {
        System.out.println(address + " is a unicast address.");
      }
    } catch (UnknownHostException ex) {
      System.err.println("Could not resolve " + args[0]);
    }
  }
}
```

```
$ java  IPCharacteristics 127.0.0.1
/127.0.0.1 is loopback address.
/127.0.0.1 is a global address.
/127.0.0.1 is a unicast address.
$ java  IPCharacteristics 192.168.254.32
/192.168.254.32 is a site-local address.
/192.168.254.32 is a unicast address.
$ java  IPCharacteristics www.oreilly.com
www.oreilly.com/208.201.239.37 is a global address.
www.oreilly.com/208.201.239.37 is a unicast address.
$ java  IPCharacteristics 224.0.2.1
/224.0.2.1 is a global address.
/224.0.2.1 is a global multicast address.
```

```
$ java  IPCharacteristics FF01:0:0:0:0:0:0:1
/ff01:0:0:0:0:0:0:1 is a global address.
/ff01:0:0:0:0:0:0:1 is an interface-local multicast address.
$ java  IPCharacteristics FF05:0:0:0:0:0:0:101
/ff05:0:0:0:0:0:0:101 is a global address.
/ff05:0:0:0:0:0:0:101 is a site wide multicast address.
$ java  IPCharacteristics 0::1
/0:0:0:0:0:0:0:1 is loopback address.
/0:0:0:0:0:0:0:1 is a global address.
/0:0:0:0:0:0:0:1 is a unicast address.
```

Object Methods

```
public boolean equals(Object o)
public int hashCode()
public String toString()
```

- equals(): both of InetAddress with the same IP address (not same hostname)
- hashCode(): solely from the IP address; consistent with the equals()
- toString(): has the form of `hostname/dotted quad address`

Example 4-7. Are www.ibiblio.org and helios.ibiblio.org the same?

```
public class IBiblioAliases {

  public static void main (String args[]) {
    try {
      InetAddress ibiblio = InetAddress.getByName("www.ibiblio.org");
      InetAddress helios = InetAddress.getByName("helios.ibiblio.org");
      if (ibiblio.equals(helios)) {
        System.out.println
            ("www.ibiblio.org is the same as helios.ibiblio.org");
      } else {
        System.out.println
            ("www.ibiblio.org is not the same as helios.ibiblio.org");
      }
    } catch (UnknownHostException ex) {
      System.out.println("Host lookup failed.");
    }
  }
}
```

```
% java IBiblioAliases
www.ibiblio.org is the same as helios.ibiblio.org
```

4.2 Inet4Address and Inet6Address

```
public final class Inet4Address extends InetAddress
public final class Inet6Address extends InetAddress
```

- Both overrides several of the methods in InetAddress but does not change their behavior in
  - Most of the time, simply not needed to know this
- Inet6Address.isIPv4CompatibleAddress(): one new method
  - Only the last four bytes are nonzero – IPv4 address stuffed into an IPv6
  - 0:0:0:0:0:0:d.d.d.d

4.3 The NetworkInterface Class

http://docs.oracle.com/javase/8/docs/api/java/net/NetworkInterface.html

- java.net.NetworkInterface objects represent physical hardware and virtual addresses

static **NetworkInterface**	**getByIndex**(int index) Get a network interface given its index
static **NetworkInterface**	**getByInetAddress**(**InetAddress** addr) Convenience method to search for a network interface that has the specified Internet Protocol (IP) address bound to it
static **NetworkInterface**	**getByName**(**String** name) Searches for the network interface with the specified name
Enumeration <**InetAddress**>	**getInetAddresses**() Convenience method to return an Enumeration with all or a subset of the InetAddresses bound to this network interface
List<**InterfaceAddress**>	**getInterfaceAddresses**() Get a List of all or a subset of the InterfaceAddresses of this network interface
static **Enumeration** <**NetworkInterface**>	**getNetworkInterfaces**() Returns all the interfaces on this machine
NetworkInterface	**getParent**() Returns the parent NetworkInterface of this interface if this is a subinterface, or null if it is a physical (non virtual) interface or has no parent
Enumeration <**NetworkInterface**>	**getSubInterfaces**() Get an Enumeration with all the subinterfaces (also known as virtual interfaces) attached to this network interface

NetworkInterface Examples

getByName()

```
try {
  NetworkInterface ni = NetworkInterface.getByName("eth0");
  if (ni == null) {
    System.err.println("No such interface:  eth0");
  }
} catch (SocketException ex) {
  System.err.println("Could not list sockets.");
}
```

getByInetAddress()

```
try {
  InetAddress local = InetAddress.getByName("127.0.0.1");
  NetworkInterface ni = NetworkInterface.getByInetAddress(local);
  if (ni == null) {
    System.err.println("That's weird. No local loopback address.");
  }
} catch (SocketException ex) {
  System.err.println("Could not list network interfaces." );
} catch (UnknownHostException ex) {
  System.err.println("That's weird. Could not lookup 127.0.0.1.");
}
```

Example 4-8. A program that lists all the network interfaces

```
import java.net.*;
import java.util.*;

public class InterfaceLister {

  public static void main(String[] args) throws SocketException {
    Enumeration<NetworkInterface> interfaces = NetworkInterface.
    getNetworkInterfaces();
    while (interfaces.hasMoreElements()) {
      NetworkInterface ni = interfaces.nextElement();
      System.out.println(ni);
    }
  }
}
```

```
% java InterfaceLister
name:eth1 (eth1) index: 3 addresses:
/192.168.210.122;

name:eth0 (eth0) index: 2 addresses:
/152.2.210.122;

name:lo (lo) index: 1 addresses:
/127.0.0.1;
```

NetworkInterface Getter Methods

boolean	**equals**(**Object** obj) Compares this object against the specified object
String	**getDisplayName**() Get the display name of this network interface
byte[]	**getHardwareAddress**() the hardware address (usually MAC) of the interface if it has one and if it can be accessed given the current privileges
int	**getIndex**() Returns the index of this network interface
Enumeration <**InetAddress**>	**getInetAddresses**() Convenience method to return an Enumeration with all or a subset of the InetAddresses bound to this network interface
List <**InterfaceAddress**>	**getInterfaceAddresses**() Get a List of all or a subset of the InterfaceAddresses of this network interface
int	**getMTU**() Returns the Maximum Transmission Unit (MTU) of this interface
String	**getName**() Get the name of this network interface
NetworkInterface	**getParent**() Returns the parent NetworkInterface of this interface if this is a subinterface, or null if it is a physical (non virtual) interface or has no parent
Enumeration <**NetworkInterface**>	**getSubInterfaces**() an Enumeration with all the subinterfaces (also known as virtual interfaces) attached to this network interface
int	**hashCode**() Returns a hash code value for the object.
boolean	**isLoopback**() Returns whether a network interface is a loopback interface.
boolean	**isPointToPoint**() Returns whether a network interface is a point to point interface.
boolean	**isUp**() Returns whether a network interface is up and running.
boolean	**isVirtual**() Returns whether this interface is a virtual interface (also called subinterface).
boolean	**supportsMulticast**() Returns whether a network interface supports multicasting or not.
String	**toString**() Returns a string representation of the object.

4.4 Some Useful Programs

- SpamCheck: asks sbl.spamhaus.org if an IPv4 is a spammer
  - i.e. A DNS query for 17.34.87.207.sbl.spamhaus.org succeeds (/returns 127.0.0.2) if 17.34.87.207 is a spammer

- Processing Web Server Logfiles: reads a web server logfile and prints each line with IP addresses converted to hostnames
  - Usually a Web server simply logs the IP addresses and converts them to hostnames at a later time
  - Common logfile format:

```
205.160.186.76 unknown - [17/Jun/2013:22:53:58 -0500] "GET /bgs/greenbg.gif HTTP 1.0" 200 50
```

Example 4-9. SpamCheck

```java
import java.net.*;

public class SpamCheck {

  public static final String BLACKHOLE = "sbl.spamhaus.org";
  public static void main(String[] args) throws UnknownHostException {
    for (String arg: args) {
      if (isSpammer(arg)) {
        System.out.println(arg + " is a known spammer.");
      } else {
        System.out.println(arg + " appears legitimate.");
      }
    }
  }

  private static boolean isSpammer(String arg) {
    try {
      InetAddress address = InetAddress.getByName(arg);
      byte[] quad = address.getAddress();
      String query = BLACKHOLE;
      for (byte octet : quad) {
        int unsignedByte = octet < 0 ? octet + 256 : octet;
        query = unsignedByte + "." + query;
      }
      InetAddress.getByName(query);
      return true;
    } catch (UnknownHostException e) {
      return false;
    }
  }
}
```

```
$ java SpamCheck 207.34.56.23 125.12.32.4 130.130.130.130
207.34.56.23 appears legitimate.
125.12.32.4 appears legitimate.
130.130.130.130 appears legitimate.
```

- Read IPv4 address list from the command line
- Send DNS query d.c.b.a.sbl.spamhaus.org for each IPv4 address of a.b.c.d
  – The query succeeds if it is a spammer

Example 4-10. Process Logfiles (Single Thread)

```
import java.io.*;
import java.net.*;

public class Weblog {

  public static void main(String[] args) {
    try (FileInputStream fin =  new FileInputStream(args[0]);
      Reader in = new InputStreamReader(fin);
      BufferedReader bin = new BufferedReader(in);) {

      for (String entry = bin.readLine();
        entry != null;
        entry = bin.readLine()) {    205.160.186.76 unknown - [17/Jun/2013:22:53:58 -0500]
        // separate out the IP address                          "GET /bgs/greenbg.gif HTTP 1.0" 200 50
        int index = entry.indexOf(' ');
        String ip = entry.substring(0, index);
        String theRest = entry.substring(index);

        // Ask DNS for the hostname and print it out
        try {
          InetAddress address = InetAddress.getByName(ip);
          System.out.println(address.getHostName() + theRest);
        } catch (UnknownHostException ex) {
          System.err.println(entry);
        }
      }
    } catch (IOException ex) {
      System.out.println("Exception: " + ex);
    }
  }
}
```

- It spends a huge amount of time sitting and waiting for DNS requests to return
- A thread pool is absolutely necessary
  - One main thread reads the logfile and
  - Passes off individual entries to other threads for processing

Example 4-11. Process Logfiles (Thread Pool)

```
import java.net.*;
import java.util.concurrent.Callable;

public class LookupTask implements Callable<String> {

  private String line;

  public LookupTask(String line) {
    this.line = line;
  }

  @Override
  public String call() {
    try {
      // separate out the IP address
      int index = line.indexOf(' ');
      String address = line.substring(0, index);
      String theRest = line.substring(index);
      String hostname = InetAddress.getByName(address).getHostName();
      return hostname + " " + theRest;
    } catch (Exception ex) {
      return line;
    }
  }
}
```

```
public class PooledWeblog {

  private final static int NUM_THREADS = 4;

  public static void main(String[] args) throws IOException {
    ExecutorService executor = Executors.newFixedThreadPool(NUM_THREADS);
    Queue<LogEntry> results = new LinkedList<LogEntry>();

    try (BufferedReader in = new BufferedReader(
      new InputStreamReader(new FileInputStream(args[0]), "UTF-8"));) {
      for (String entry = in.readLine(); entry != null; entry = in.readLine()) {
        LookupTask task = new LookupTask(entry);
        Future<String> future = executor.submit(task);
        LogEntry result = new LogEntry(entry, future);
        results.add(result);
      }
    }

    // Start printing the results. This blocks each time a result isn't ready..
    for (LogEntry result : results) {
      try {
        System.out.println(result.future.get());
      } catch (InterruptedException | ExecutionException ex) {
        System.out.println(result.original);
      }
    }

    executor.shutdown();
  }
  private static class LogEntry {
    String original;
    Future<String> future;

    LogEntry(String original, Future<String> future) {
     this.original = original;
     this.future = future;
    }
  }
}
```

1. Callback per entry
2. Add callback to queue
3. Wait results in order

- Processed in parallel
  - 10x-50x faster

```
import java.io.*;
import java.util.*;
import java.util.concurrent.*;

// Requires Java 7 for try-with-resources and multi-catch
```

Summary

4.1 The InetAddress Class

http://docs.oracle.com/javase/8/docs/api/java/net/InetAddress.html

- Create Objects and Getter Methods
- Example 4-1. Print the address of a host name (OreillyByName)
- Example 4-2. Print the address of the machine it's run on (MyAddress)
- Example 4-3. Given the address, find the hostname (ReverseTest)
- Example 4-4. Find the IP address of the local machine (MyAddress)
- Example 4-5. Determining whether an IP address is v4 or v6 (AddressTests)
- Example 4-6. Testing characteristics of an IP address (IPCharacteristics)
- Example 4-7. Check whether two host names are the same (IBiblioAliases)

4.2 Inet4Address and Inet6Address

http://docs.oracle.com/javase/8/docs/api/java/net/Inet4Address.html

http://docs.oracle.com/javase/8/docs/api/java/net/Inet6Address.html

4.3 The NetworkInterface Class

http://docs.oracle.com/javase/8/docs/api/java/net/NetworkInterface.html

- Example 4-8. List all the network interfaces (InterfaceLister)

4.4 Some Useful Programs

- Example 4-9. SpamCheck (SpamCheck)
- Example 4-10. Process LogFiles – Single Thread (Weblog)
- Example 4-11. Process LogFiles – Thread Pool (LookupTask, PooledWeblog)

Ch.8 Sockets for Clients

Unit-6

Outline

8.1 Using Sockets

8.2 Constructing and Connecting Sockets

8.3 Getting Information about Sockets

8.4 Setting Socket Options

8.5 Socket Exceptions

8.6 Sockets in GUI Applications

8.1 Using Sockets

- Datagram: transmitted data across the Internet in packets of finite size
  - Each datagram consists of
    - Header: address and port
    - Payload: data itself
  - It may split the data across multiple packets and reassemble at the destination
- Socket: connection between two hosts
  - An endpoint for communication between two machines (full-duplex)
  - Seven basic operations

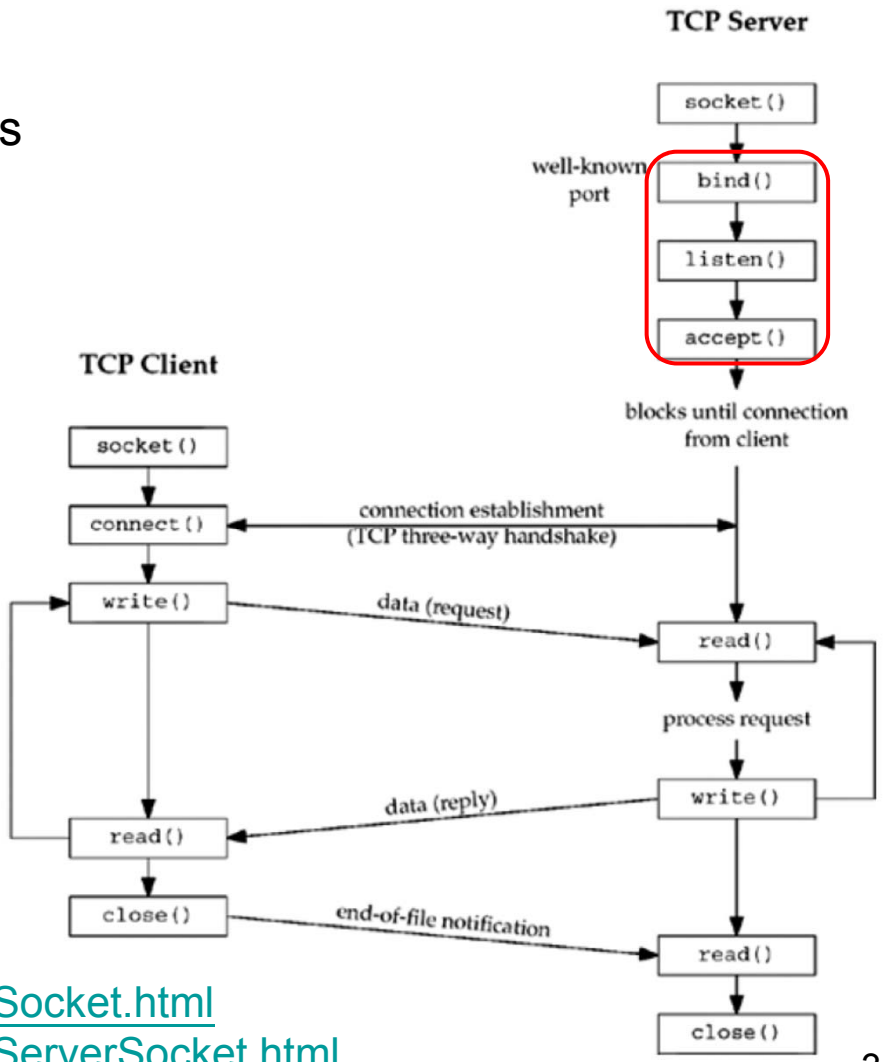
Java's Socket Class

- Connect to a remote machine
- Send data
- Receive data
- Close a connection

Java's ServerSocket Class

- Bind to a port
- Listen for incoming data
- Accept connections from remote machines on the bound port

http://docs.oracle.com/javase/8/docs/api/java/net/Socket.html
http://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html



Investigating Protocols with Telnet

% telnet <host name/IP> [<port (default: 23)>]

```
flare% telnet localhost 25
Trying 127.0.0.1 ...
Connected to localhost.sunspot.noao.edu.
Escape character is '^]'.
220 flare.sunspot.noao.edu Sendmail 4.1/SMI-4.1 ready at
Fri, 5 Jul 93 13:13:01 MDT
HELO sunspot.noao.edu
250 flare.sunspot.noao.edu Hello localhost [127.0.0.1], pleased to meet you
MAIL FROM: bart
250 bart... Sender ok
RCPT TO: local@sunspot.noao.edu
250 local@sunspot.noao.edu... Recipient ok
DATA
354 Enter mail, end with "." on a line by itself

In a pitiful attempt to reingratiate myself with the students
after their inevitable defeat of the staff on the volleyball
court at 4:00 P.M., July 24, I will be throwing a victory
party for the students at my house that evening at 7:00.
Everyone is invited.

Beer and Ben-Gay will be provided so the staff may drown
their sorrows and assuage their aching muscles after their
public humiliation.

Sincerely,

Bart
.
250 Mail accepted
QUIT
221 flare.sunspot.noao.edu delivering mail
Connection closed by foreign host.
```

Reading from Servers with Sockets

- Daytime Protocol (RFC 867): TCP Port 13 (Example 8-1 and Example 8-2)

```
$ telnet time.nist.gov 13
Trying 129.6.15.28...
Connected to time.nist.gov.
Escape character is '^]'.

56375 13-03-24 13:37:50 50 0 0 888.8 UTC(NIST) *
Connection closed by foreign host.
```

  - Format: JJJJJ YY-MM-DD HH:MM:SS TT L H msADV UTC(NIST) OTM
    - JJJJJ: Modified Julinan Date (days since Nov 17, 1858)
    - TT: 00 means standard time and 50 means daylight savings time
    - L: indicates whether a leap second will be added (1) or subtracted (2)
    - H: health of the server (0: healthy; 1: up to 5 seconds off; …)
    - msADV: how long (ms) it estimates it's going to take for the response to return
    - UTC (NIST): time-zone constant string
    - OTM: almost a constant (an asterisk)
- Time Protocol (RFC 868): TCP Port 37 (Example 8-3)
  - The server returns a 32-bit time value meaning the number of seconds since 00:00 (midnight) 1 January 1900 GMT
  - All network protocols use big-endian numbers
- Suggested to use NTP (RFC 5905)

Example 8-1. A *Daytime* Protocol Client

- Socket constructor

```
public Socket(String host, int port) throws UnknownHostException, IOException
public Socket(InetAddress host, int port) throws IOException
```

```
import java.net.*;
import java.io.*;

public class DaytimeClient {

  public static void main(String[] args) {

    String hostname = args.length > 0 ? args[0] : "time.nist.gov";
    Socket socket = null;
    try {
      socket = new Socket(hostname, 13);
      socket.setSoTimeout(15000);
      InputStream in = socket.getInputStream();
      StringBuilder time = new StringBuilder();
      InputStreamReader reader = new InputStreamReader(in, "ASCII");
      for (int c = reader.read(); c != -1; c = reader.read()) {
        time.append((char) c);
      }
      System.out.println(time);
    } catch (IOException ex) {
      System.err.println(ex);
    } finally {
      if (socket != null) {
        try {
          socket.close();
        } catch (IOException ex) {
          // ignore
        }
      }
    }
  }
}
```

- setSoTimeout(<ms>): set socket timeout
  - Prevent unpredicted hang by the server
  - SocketTimeoutException
- getInputStream(): return an InputStream for further input

Java 6-: need to release resources →
Java 7+: implement `Autocloseable` ↓

```
try (Socket socket = new Socket("time.nist.gov", 13)) {
  // read from the socket...
} catch (IOException ex) {
  System.err.println("Could not connect to time.nist.gov");
}
```

```
Socket socket = null;
try {
  socket = new Socket(hostname, 13);
  // read from the socket...
} catch (IOException ex) {
  System.err.println(ex);
} finally {
  if (socket != null) {
    try {
      socket.close();
    } catch (IOException ex) {
      // ignore
    }
  }
}
```

Example 8-2. Construct a Date by Talking to time.nist.gov

- Parse the date into a java.util.Date object instead
  - class Date represents a specific instant in time, with millisecond precision
  - DateFormat class: an abstract class for date/time formatting subclasses which formats and parses dates or time in a language-independent manner
  - SimpleDateFormat class: a concrete class for formatting and parsing dates in a locale-sensitive manner

```
import java.net.*;
import java.text.*;
import java.util.Date;
import java.io.*;

public class Daytime {

  public Date getDateFromNetwork() throws IOException, ParseException {
    try (Socket socket = new Socket("time.nist.gov", 13)) {
      socket.setSoTimeout(15000);
      InputStream in = socket.getInputStream();
      StringBuilder time = new StringBuilder();
      InputStreamReader reader = new InputStreamReader(in, "ASCII");
      for (int c = reader.read(); c != -1; c = reader.read()) {
        time.append((char) c);
      }
      return parseDate(time.toString());
    }
  }
```

```
56375 13-03-24 15:05:42 50 0 0 843.6 UTC(NIST) *
```

```
  static Date parseDate(String s) throws ParseException {
    String[] pieces = s.split(" ");
    String dateTime = pieces[1] + " " + pieces[2] + " UTC";
    DateFormat format = new SimpleDateFormat("yy-MM-dd hh:mm:ss z");
    return format.parse(dateTime);
  }
}
```

Letter	Component	Presentation	Examples
G	Era designator	Text	AD
y	Year	Year	1996; 96
Y	Week year	Year	2009; 09
M	Month in year (context sensitive)	Month	July; Jul; 07
L	Month in year (standalone form)	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day name in week	Text	Tuesday; Tue
u	Day number of week (1=Mon, ...,7=Sun)	Number	1
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800
X	Time zone	ISO 8601 time zone	-08; -0800; -08:00

Example 8-3. A *Time* Protocol Client

The server returns 32-bit number of seconds since 00:00 (midnight) 1 January 1900 GMT

```
public class Time {

  private static final String HOSTNAME = "time.nist.gov";

  public static void main(String[] args) throws IOException, ParseException {
    Date d = Time.getDateFromNetwork();
    System.out.println("It is " + d);
  }

  public static Date getDateFromNetwork() throws IOException, ParseException {
    long differenceBetweenEpochs = 2208988800L;
    Socket socket = null;
    try {
      socket = new Socket(HOSTNAME, 37);
      socket.setSoTimeout(15000);

      InputStream raw = socket.getInputStream();

      long secondsSince1900 = 0;
      for (int i = 0; i < 4; i++) {
        secondsSince1900 = (secondsSince1900 << 8) | raw.read();
      }
      long secondsSince1970
                = secondsSince1900 - differenceBetweenEpochs;
      long msSince1970 = secondsSince1970 * 1000;
      Date time = new Date(msSince1970);

      return time;
    } finally {
      try {
        if (socket != null) socket.close();
      }
      catch (IOException ex) {}
    }
  }
}
```

4-byte big-endian number

```
// The time protocol sets the epoch at 1900,
// the Java Date class at 1970. This number
// converts between them.

// If you'd rather not use the magic number, uncomment
// the following section which calculates it directly.
/*
TimeZone gmt = TimeZone.getTimeZone("GMT");
Calendar epoch1900 = Calendar.getInstance(gmt);
epoch1900.set(1900, 01, 01, 00, 00, 00);
long epoch1900ms = epoch1900.getTime().getTime();
Calendar epoch1970 = Calendar.getInstance(gmt);
epoch1970.set(1970, 01, 01, 00, 00, 00);
long epoch1970ms = epoch1970.getTime().getTime();

long differenceInMS = epoch1970ms - epoch1900ms;
long differenceBetweenEpochs = differenceInMS/1000;
*/
```

```
$ java Time
It is Sun Mar 24 12:22:17 EDT 2013
```

Writing to Servers with Sockets

- Dict Protocol (RFC 2229): TCP Port 2628 (Example 8-4)
  - Dictionary Server Protocol (DICT)

```
$ telnet dict.org 2628
Trying 216.18.20.172...
Connected to dict.org.
Escape character is '^]'.
220 pan.alephnull.com dictd 1.12.0/rf on Linux 3.0.0-14-server
    <auth.mime> <499772.29595.1364340382@pan.alephnull.com>
DEFINE eng-lat gold
150 1 definitions retrieved
151 "gold" eng-lat "English-Latin Freedict dictionary"
gold [gould]
   aurarius; aureus; chryseus
   aurum; chrysos

.
250 ok [d/m/c = 1/0/10; 0.000r 0.000u 0.000s]
DEFINE eng-lat computer
552 no match [d/m/c = 0/0/9; 0.000r 0.000u 0.000s]
quit
221 bye [d/m/c = 0/0/0; 42.000r 0.000u 0.000s]
```

  - dict.org databases changed:
    - Change eng-lat to fd-eng-lat
    - foldoc: computing dictionary
  - Control response lines begin with a three-digit code
  - The actual definition is plain text, terminated with a period on a line by itself

Example 8-4. A English-to-Latin Translator

```
public class DictClient {

  public static final String SERVER = "dict.org";
  public static final int PORT = 2628;
  public static final int TIMEOUT = 15000;

  public static void main(String[] args) {

    Socket socket = null;
    try {
      socket = new Socket(SERVER, PORT);
      socket.setSoTimeout(TIMEOUT);
      OutputStream out = socket.getOutputStream();
      Writer writer = new OutputStreamWriter(out, "UTF-8");
      writer = new BufferedWriter(writer);
      InputStream in = socket.getInputStream();
      BufferedReader reader = new BufferedReader(
          new InputStreamReader(in, "UTF-8"));

      for (String word : args) {
        define(word, writer, reader);
      }

      writer.write("quit\r\n");
      writer.flush();
    } catch (IOException ex) {
      System.err.println(ex);
    } finally { // dispose
      if (socket != null) {
        try {
          socket.close();
        } catch (IOException ex) {
          // ignore
        }
      }
    }
  }
```

```
  static void define(String word, Writer writer, BufferedReader reader)
      throws IOException, UnsupportedEncodingException {
    writer.write("DEFINE eng-lat " + word + "\r\n");
    writer.flush();

    for (String line = reader.readLine(); line != null; line = reader.readLine()) {
      if (line.startsWith("250 ")) { // OK
        return;
      } else if (line.startsWith("552 ")) { // no match
        System.out.println("No definition found for " + word);
        return;
      }
      else if (line.matches("\\d\\d\\d .*")) continue;
      else if (line.trim().equals(".")) continue;
      else System.out.println(line);
    }
  }
}
```

- getOutputStream(): return a raw OutputStream for writing data

```
$ java DictClient gold uranium silver copper lead
gold [gould]
   aurarius; aureus; chryseus
   aurum; chrysos

No definition found for uranium
silver [silvər]
   argenteus
   argentum

copper [kɔpər]
   æneus; aheneus; ærarius; chalceus
   æs

lead [led]
   ducere
   molybdus; plumbum
```

Databases provided by dict.org
http://www.dict.org/bin/Dict?Form=Dict4

Half-closed Sockets

- close(): shuts down both input and output from the socket
- shutdownInput() and shutdownOutput(): close only half the connection
  - Many protocols begin with the client sending a request to the server, and then reading the response

```
try (Socket connection = new Socket("www.oreilly.com", 80)) {
  Writer out = new OutputStreamWriter(
           connection.getOutputStream(), "8859_1");
  out.write("GET / HTTP 1.0\r\n\r\n");
  out.flush();
  connection.shutdownOutput();
  // read the response...
} catch (IOException ex) {
  ex.printStackTrace();
}
```

  - Further reads return -1; further writes throw an IOException
  - isInputShutdown() and isOutputShutdown(): return whether I/O streams are closed
    - Rather than isConnected() and isClosed() (discussed later)

8.2 Constructing and Connecting Sockets

- java.net.Socket: uses native code to communicate with the local TCP stack
- Basic Constructors

```
public Socket(String host, int port) throws UnknownHostException, IOException
public Socket(InetAddress host, int port) throws IOException
```

Example 8-5. Find out which of the first 1024 ports seem to be hosting TCP servers on a specified host

/etc/services or C:\Windows\System32\drivers\etc\services
https://en.wikipedia.org/wiki/List_of_TCP_and_UDP_port_numbers

```
import java.net.*;
import java.io.*;

public class LowPortScanner {

  public static void main(String[] args) {

    String host = args.length > 0 ? args[0] : "localhost";

    for (int i = 1; i < 1024; i++) {
      try {
        Socket s = new Socket(host, i);
        System.out.println("There is a server on port " + i + " of "
         + host);
        s.close();
      } catch (UnknownHostException ex) {
        System.err.println(ex);
        break;
      } catch (IOException ex) {
        // must not be a server on this port
      }
    }
  }
}
```

```
$ java LowPortScanner
There is a server on port 21 of localhost
There is a server on port 22 of localhost
There is a server on port 23 of localhost
There is a server on port 25 of localhost
There is a server on port 37 of localhost
There is a server on port 111 of localhost
There is a server on port 139 of localhost
There is a server on port 210 of localhost
There is a server on port 515 of localhost
There is a server on port 873 of localhost
```

- Three constructors create unconnected sockets

```
public Socket()
public Socket(Proxy proxy)
protected Socket(SocketImpl impl)
```

Picking a Local Interface to Connect From

- Two constructors specify
  - Host and port to connect to
  - Interface and localPort (0 for random between 1024 and 65535) to connect from

```
public Socket(String host, int port, InetAddress interface, int localPort)
      throws IOException, UnknownHostException
public Socket(InetAddress host, int port, InetAddress interface, int localPort)
      throws IOException
```

  - BindException occurs if not available
- Useful for
  - Router/firewall with dual Ethernet ports
  - Server/gateway that uses
    - One interface to provide outside services
    - Another one interface to connect local machines
  - Heterogeneous multiple simultaneous connections
  - Example: periodically dump error logs to a printer or send them over an internal mail server

```
try {
  InetAddress inward = InetAddress.getByName("router");
  Socket socket = new Socket("mail", 25, inward, 0);
  // work with the sockets...
} catch (IOException ex) {
  System.err.println(ex);
}
```

Constructing Without Connecting

- Constructor without arguments and connect/bind later
  - connect() with timeout (ms)
    - 0 means forever

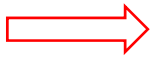
```
public Socket()
```

```
public void connect(SocketAddress endpoint, int timeout) throws IOException
```

```
try {
  Socket socket = new Socket();
  // fill in socket options
  SocketAddress address = new InetSocketAddress("time.nist.gov", 13);
  socket.connect(address);
  // work with the sockets...
} catch (IOException ex) {
  System.err.println(ex);
}
```

  - Set options further before connection
    - setSoTimeout(<ms>) for all operations
  - Noargs constructor throws no exception

```
Socket socket = null;
try {
  socket = new Socket(SERVER, PORT);
  // work with the socket...
} catch (IOException ex) {
  System.err.println(ex);
} finally {
  if (socket != null) {
    try {
      socket.close();
    } catch (IOException ex) {
      // ignore
    }
  }
}
```



```
Socket socket = new Socket();
SocketAddress address = new InetSocketAddress(SERVER, PORT);
try {
  socket.connect(address);
  // work with the socket...
} catch (IOException ex) {
  System.err.println(ex);
} finally {
  try {
    socket.close();
  } catch (IOException ex) {
    // ignore
  }
}
```

Socket Addresses

- SocketAddress class: abstract class with no protocol attachment
  - Socket Class: getRemoteSocketAddress() and getLocalSocketAddress() return socket end-points

```
public SocketAddress getRemoteSocketAddress()
public SocketAddress getLocalSocketAddress()
```

  - Example: reuse the end-point

```
Socket socket = new Socket("www.yahoo.com", 80);
SocketAddress yahoo = socket.getRemoteSocketAddress();
socket.close();
Socket socket2 = new Socket();
socket2.connect(yahoo);
```

- InetSocketAddress: implements an IP Socket Address
  - Basic constructors

```
public InetSocketAddress(InetAddress address, int port)
public InetSocketAddress(String host, int port)
public InetSocketAddress(int port)
```

  - Use static factory method createUnresolved() to skip looking up the host in DNS

```
public static InetSocketAddress createUnresolved(String host, int port)
```

  - Getter methods to inspect object

```
public final InetAddress getAddress()
public final int         getPort()
public final String      getHostName()
```

Proxy Servers

- Socket(Proxy) constructor specifies a proxy server for socket

```
public Socket(Proxy proxy)
```

  - Normally the proxy server is controlled by the socksProxyHost and socksProxyPort system properties
    - SOCKS is the only low-level proxy type Java understands

```
SocketAddress proxyAddress = new InetSocketAddress("myproxy.example.com", 1080);
Proxy proxy = new Proxy(Proxy.Type.SOCKS, proxyAddress)
Socket s = new Socket(proxy);
SocketAddress remote = new InetSocketAddress("login.ibiblio.org", 25);
s.connect(remote);
```

    - High-level proxy types
      - Proxy.Type.HTTP that works in the application layer rather than the transport layer and
      - Proxy.Type.DIRECT that represents proxyless connections

  - You can pass Proxy.NO_PROXY for the argument to bypass all proxy servers completely and connect directly to the remote host
    - Socket(Proxy.NO_PROXY)

8.3 Getting Information About a Socket

- Getter methods of Socket class

```
public InetAddress getInetAddress()
public int getPort()
public InetAddress getLocalAddress()
public int getLocalPort()
```

Address and port of the remote host

Example 8-6. Get a socket's information

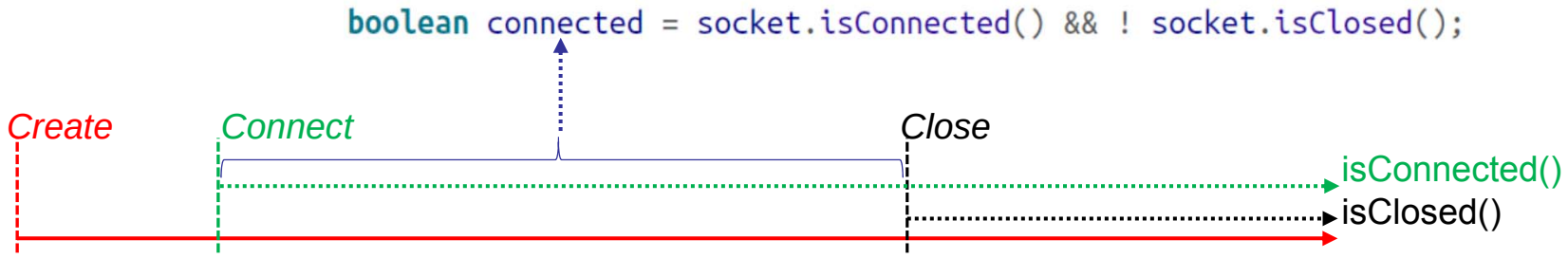
```
import java.net.*;
import java.io.*;

public class SocketInfo {

  public static void main(String[] args) {

    for (String host : args) {
      try {
        Socket theSocket = new Socket(host, 80);
        System.out.println("Connected to " + theSocket.getInetAddress()
            + " on port "  + theSocket.getPort() + " from port "
            + theSocket.getLocalPort() + " of "
            + theSocket.getLocalAddress());
      }  catch (UnknownHostException ex) {
        System.err.println("I can't find " + host);
      } catch (SocketException ex) {
        System.err.println("Could not connect to " + host);
      } catch (IOException ex) {
        System.err.println(ex);
      }
    }
  }
}
```

- Reads a list of hostnames from the command line
- Attempts to open a socket to each one, and then
- Uses these four methods to print
  the remote host, the remote port,
  the local address, and the local port

```
$ java SocketInfo www.oreilly.com www.oreilly.com www.elharo.com
  login.ibiblio.org
Connected to www.oreilly.com/208.201.239.37 on port 80 from port 49156 of
/192.168.254.25
Connected to www.oreilly.com/208.201.239.37 on port 80 from port 49157 of
/192.168.254.25
Connected to www.elharo.com/216.254.106.198 on port 80 from port 49158 of
/192.168.254.25
Could not connect to login.ibiblio.org
```

Closed or Connected?

- isClosed(): returns true if the ***opened*** socket is closed
  - If the socket has *never been connected* in the first place, isClosed() returns false, even though the socket isn't exactly open
- isConnected(): whether the socket has ***ever*** been connected to a remote host
  - If the socket was able to connect to the remote host at all, this method returns true, *even after* that socket has been closed
- To tell if a socket is currently open, you need to check that
  - isConnected() returns true and
  - isClosed() returns false



- Override toString() of Socket class

```
Socket[addr=www.oreilly.com/198.112.208.11,port=80,localport=50055]
```

  - Primarily for debugging; format may change in the future

8.4 Setting Socket Options

- Nine options for client-side sockets
  - Follow classic Unix C naming conventions
  - TCP_NODELAY, SO_BINDADDR, SO_TIMEOUT, SO_LINGER SO_SNDBUF, SO_RCVBUF, SO_KEEPALIVE, OOBINLINE, IP_TOS
- TCP_NODELAY
  - Packets are sent as quickly as possible regardless of their size (even 1 byte)

```
public void setTcpNoDelay(boolean on) throws SocketException
public boolean getTcpNoDelay() throws SocketException
```

  - setTcpNoDelay(true) turns off buffering for the socket

```
if (!s.getTcpNoDelay()) s.setTcpNoDelay(true);
```

- SO_LINGER
  - Specifies what to do with unsent data when a socket is closed

```
public void setSoLinger(boolean on, int seconds) throws SocketException
public int getSoLinger() throws SocketException
```

  - By default, close() returns immediately and the system still tries to send any remaining data
  - If the linger time is set to 0, any unsent packets are thrown away when closed
  - If the linger time > 0 and SO_LINGER is turned on, close() blocks while waiting the specified number of seconds for the data to be sent
    - Maximum linger time is 65,535 seconds
    - getTcpSoLinger() returns -1 if this option is disabled

```
if (s.getTcpSoLinger() == -1) s.setSoLinger(true, 240);
```

Socket Options – Timeout and Buffer Sizes

- SO_TIMEOUT
  - Specifies the timeout (ms) for socket calls that may block, i.e. read()
    - The default is 0, an infinite timeout

```
public void setSoTimeout(int milliseconds) throws SocketException
public int getSoTimeout() throws SocketException
```

  - Throws InterruptedIOException when the timeout expires

```
if (s.getSoTimeout() == 0) s.setSoTimeout(180000);
```

- SO_RCVBUF and SO_SNDBUF
  - Controls the suggested receive/send buffer sizes
    - BSD 4.2: 2KB; WinXP: 17,520 bytes; 128 KB is common

```
public void setReceiveBufferSize(int size)
    throws SocketException, IllegalArgumentException
public int getReceiveBufferSize() throws SocketException
public void setSendBufferSize(int size)
    throws SocketException, IllegalArgumentException
public int getSendBufferSize() throws SocketException
```

  - Roughly, maximum available bandwidth = BufferSize / Latency
    - Doubling the buffer size will double the bandwidth
    - Set the buffer too high than the network can handle, leading to congestion, dropped packets, and slower performance
  - The rule of thumb is not to do it until you've measured a problem
    - You can call InetAddress.isReachable() to check the latency first
    - Usually set to the same (smaller of these two)

Socket Options – Keep-alive, Urgent, and Reuse

- SO_KEEPALIVE
  - If turned on, the client occasionally sends a data packet over an idle connection to make sure the server hasn't crashed; The default is false

```
public void setKeepAlive(boolean on) throws SocketException
public boolean getKeepAlive() throws SocketException
```

    - Most commonly once every two hours
      - If the server fails to respond, the client keeps trying for a little more than 11 minutes
        - » If not response within 12 minutes, the client closes the socket

```
if (s.getKeepAlive()) s.setKeepAlive(false);
```

- OOBINLINE
  - Sends a single byte of '*urgent*' data out of band; The default is off

```
public void setOOBInline(boolean on) throws SocketException
public boolean getOOBInline() throws SocketException

public void sendUrgentData(int data) throws IOException
```

  - More modern approach is to place the urgent data in the regular received data queue in its proper order
    - Java does not distinguish it from non-urgent data

```
if (!s.getOOBInline()) s.setOOBInline(true);
```

- SO_REUSEADDR
  - If turned on, another socket is allowed to bind to the port even while data may be outstanding for the previous socket
    - setReuseAddress() must be called before the new socket binds to the port
    - The default is off; It may not immediately release the local port when the socket was closed

```
public void setReuseAddress(boolean on) throws SocketException
public boolean getReuseAddress() throws SocketException
```

Socket Option – Class of Service

- IP_TOS Class of Service
  - Specifies the class of service; stored in an eight-bit field in the IP header (0-255)

```
public int getTrafficClass() throws SocketException
public void setTrafficClass(int trafficClass) throws SocketException
```

    - High-order six bits contain a Differentiated Services Code Point (DSCP)
      - Up to 2^6 different traffic classes; Not hard and fast guarantees of services
      - Respected on some networks internally; A packet crosses ISPs is almost always ignored
    - Low-order two bits contain an Explicit Congestion Notification (ECN) value
      - Should be set to zero; usually set by intermediate routers
  - Many implementations ignore these values completely
    - Android treats it as a no-op

```
Socket s = new Socket("www.yahoo.com", 80);
s.setTrafficClass(0xB8); // 10111000 in binary
```

```
Socket s1 = new Socket("www.example.com", 80);
s1.setTrafficClass(0x26); // 00100110 in binary
Socket s2 = new Socket("www.example.com", 80);
s2.setTrafficClass(0x0A); // 00001010 in binary
Socket s3 = new Socket("www.example.com", 80);
s3.setTrafficClass(0x0E); // 00001110 in binary
```

Table 8-1. Common DSCP values and interpretations

PHB (Per Hop Behavior)	Binary value	Purpose
Default	00000	Best-effort traffic.
Expedited Forwarding (EF)	101110	Low-loss, low-delay, low-jitter traffic. Often limited to 30% or less of network capacity.
Assured Forwarding (AF)	*multiple*	Assured delivery up to a specified rate.
Class Selector	xxx000	Backward compatibility with the IPv4 TOS header, as stored in the first three bits.

Table 8-2. Assured forwarding priority classes

	Class 1 (lowest priority)	Class 2	Class 3	Class 4 (highest priority)
Low Drop Rate	AF11 (001010)	AF21 (010010)	AF31 (011010)	AF41 (100010)
Medium Drop Rate	AF12 (001100)	AF22 (010100)	AF32 (011100)	AF42 (100100)
High Drop Rate	AF13 (001110)	AF23 (010110)	AF33 (011110)	AF43 (100110)

- Use setPerformancePreferences() instead to assign preferences

```
public void setPerformancePreferences(int connectionTime, int latency, int bandwidth)
```

  - i.e. setPerformancePreferences(2, 1, 3) means Maximum bandwidth is most important

8.5 Socket Exceptions

- Most methods of the Socket class are declared to throw IOException or its subclass, java.net.SocketException

```
public class SocketException extends IOException
```

  - Several subclasses of SocketException that provide more information about what went wrong and why

```
public class BindException extends SocketException
public class ConnectException extends SocketException
public class NoRouteToHostException extends SocketException

public class ProtocolException extends IOException
```

    - BindException: a local port is in use or no privileges to use
    - ConnectException: connection refused at the remote
    - NoRouteToHostException: connection timed out
    - ProtocolException: received data violates the TCP/IP specification

8.6 Sockets in GUI Applications

- An example to discuss the considerations that arise when integrating networking code with Swing (GUI) applications
- Whois: a simple directory service protocol
  - Original defined in RFC 954 (centralized nature)
  - Whois++ documented in RFCs 1913 and 1914 (not widely implemented)
- Basic structure of the whois protocol
  1. The client opens a TCP socket to port 43 on the server.
  2. The client sends a search string terminated by (\r\n)
     - The search string can be a name, a list of names, or a special command
     - You can also search for domain names, like www.oreilly.com or netscape.com
  3. The server sends an unspecified amount of human-readable information in response to the command and closes the connection
  4. The client displays this information to the user
- Problem: the format is regrettably nonstandard
  - Different whois servers can and do send decidedly different output
  - Keywords of whois prefixes are supported to modifying a search

Non-standardized Output

Output varies on servers and inputs

```
$ telnet whois.internic.net 43
Trying 199.7.50.74...
Connected to whois.internic.net.
Escape character is '^]'.
Harold

Whois Server Version 2.0

Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

HAROLD.LUCKYLAND.ORG
HAROLD.FRUGAL.COM
HAROLD.NET
HAROLD.COM

To single out one record, look it up with "xxx", where xxx is one of the
of the records displayed above. If the records are the same, look them up
with "=xxx" to receive a full display for each record.
>>> Last update of whois database: Sat, 30 Mar 2013 15:15:05 UTC <<<
...
Connection closed by foreign host.
```

```
% telnet whois.internic.net 43
Trying 198.41.0.6...
Connected to whois.internic.net.
Escape character is '^]'.
oreilly.com

Whois Server Version 1.3

Domain names in the .com and .net domains can now be registered
with many different competing registrars. Go to http://www.internic.net
for detailed information.

   Domain Name: OREILLY.COM
   Registrar: BULKREGISTER, LLC.
   Whois Server: whois.bulkregister.com
   Referral URL: http://www.bulkregister.com
   Name Server: NS1.SONIC.NET
   Name Server: NS.OREILLY.COM
   Status: ACTIVE
   Updated Date: 17-oct-2002
   Creation Date: 27-may-1997
   Expiration Date: 26-may-2004


>>> Last update of whois database: Tue, 16 Dec 2003 18:36:16 EST <<<
...
Connection closed by foreign host.
```

```
% telnet whois.nic.fr 43
telnet whois.nic.fr 43
Trying 192.134.4.18...
Connected to winter.nic.fr.
Escape character is '^]'.
Harold

Tous droits reserves par copyright.
Voir http://www.nic.fr/outils/dbcopyright.ht
Rights restricted by copyright.
See http://www.nic.fr/outils/dbcopyright.htr

person:      Harold Potier
address:     ARESTE
address:     154 Avenue Du Brezet
address:     63000 Clermont-Ferrand
address:     France
phone:       +33 4 73 42 67 67
fax-no:      +33 4 73 42 67 67
nic-hdl:     HP4305-FRNIC
mnt-by:      OLEANE-NOC
changed:     hostmaster@oleane.net 20000510
changed:     migration-dbm@nic.fr 20001015
source:      FRNIC

person:      Harold Israel
address:     LE PARADIS LATIN
address:     28 rue du Cardinal Lemoine
address:     Paris, France 75005 FR
phone:       +33 1 43252828
fax-no:      +33 1 43296363
e-mail:      info@cie.fr
nic-hdl:     HI68-FRNIC
notify:      info@cie.fr
changed:     registrar@ns.il 19991011
changed:     migration-dbm@nic.fr 20001015
source:      FRNIC
```

Whois Prefixes

Table 8-3. Whois prefixes

Prefix	Meaning
Domain	Find only domain records.
Gateway	Find only gateway records.
Group	Find only group records.
Host	Find only host records.
Network	Find only network records.
Organization	Find only organization records.
Person	Find only person records.
ASN	Find only autonomous system number records.
Handle or !	Search only for matching handles.
Mailbox or @	Search only for matching email addresses.
Name or :	Search only for matching names.
Expand or *	Search only for group records and show all individuals in that group.
Full or =	Show complete record for each match.
Partial or suffix	Match records that start with the given string.
Summary or $	Show just the summary, even if there's only one match.
SUBdisplay or %	Show the users of the specified host, the hosts on the specified network, etc.

Hard to remember the prefixes;
GUI to help users specify them

Example: % whois Person Partial Elliot

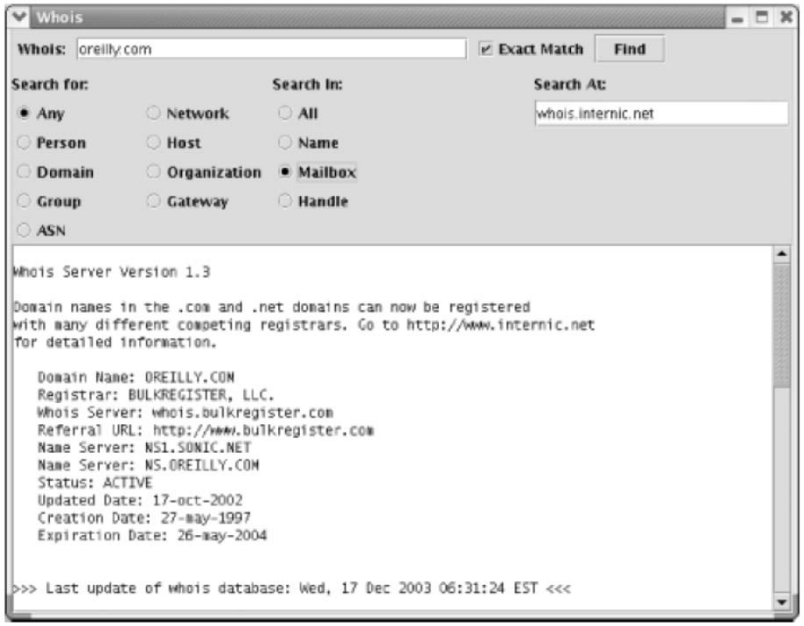
Example 8-7. Whois Class

Main function: lookUpNames()

```
import java.net.*;
import java.io.*;

public class Whois {

  public final static int DEFAULT_PORT = 43;
  public final static String DEFAULT_HOST = "whois.internic.net";

  private int port = DEFAULT_PORT;
  private InetAddress host;

  public Whois(InetAddress host, int port) {
    this.host = host;
    this.port = port;
  }

  public Whois(InetAddress host) {
    this(host, DEFAULT_PORT);
  }

  public Whois(String hostname, int port)
   throws UnknownHostException {
    this(InetAddress.getByName(hostname), port);
  }

  public Whois(String hostname) throws UnknownHostException {
    this(InetAddress.getByName(hostname), DEFAULT_PORT);
  }

  public Whois() throws UnknownHostException {
    this(DEFAULT_HOST, DEFAULT_PORT);
  }

  // Items to search for
  public enum SearchFor {
    ANY("Any"), NETWORK("Network"), PERSON("Person"), HOST("Host"),
    DOMAIN("Domain"), ORGANIZATION("Organization"), GROUP("Group"),
    GATEWAY("Gateway"), ASN("ASN");
    private String label;

    private SearchFor(String label) {
      this.label = label;
    }
  }

  // Categories to search in
  public enum SearchIn {
    ALL(""), NAME("Name"), MAILBOX("Mailbox"), HANDLE("!");

    private String label;

    private SearchIn(String label) {
      this.label = label;
    }
  }

  public String lookUpNames(String target, SearchFor category,
      SearchIn group, boolean exactMatch) throws IOException {

    String suffix = "";
    if (!exactMatch) suffix = ".";

    String prefix = category.label + " " + group.label;
    String query = prefix + target + suffix;

    Socket socket = new Socket();
    try {
      SocketAddress address = new InetSocketAddress(host, port);
      socket.connect(address);
      Writer out
          = new OutputStreamWriter(socket.getOutputStream(), "ASCII");
      BufferedReader in = new BufferedReader(new
          InputStreamReader(socket.getInputStream(), "ASCII"));
      out.write(query + "\r\n");
      out.flush();

      StringBuilder response = new StringBuilder();
      String theLine = null;
      while ((theLine = in.readLine()) != null) {
        response.append(theLine);
        response.append("\r\n");
      }
      return response.toString();
    } finally {
      socket.close();
    }
  }

  public InetAddress getHost() {
    return this.host;
  }

  public void setHost(String host)
      throws UnknownHostException {
    this.host = InetAddress.getByName(host);
  }
}
```

Graphical Whois Client

- To avoid an obscure race condition that can lead to deadlock, this needs to be done in the event dispatch thread
  - Events that the program must respond to
    - The user typing a name in the Whois: search box and either clicking the Find button or hitting Enter
      - Make the network connection
    - The user typing a new host in the server text field
  - Two rules to avoid deadlock and slowness
    - All updates to Swing components happen on the event dispatch thread
    - No slow blocking operations, especially I/O, happen on the event dispatch thread Otherwise a slow-to-respond server can hang the entire application



- Example 8-8. graphical whois client interface
  - Safely make network connections from a GUI program without blocking the event dispatch thread

Summary

8.1 Using Sockets

- Socket, SimpleDateFormat
- Daytime (Examples 8-1 and 8-2), Time (Example 8-3), DictClient (Example 8-4)

8.2 Constructing and Connecting Sockets

- LowPortScanner (Example 8-5)

8.3 Getting Information about Sockets

- SocketInfo (Example 8-6)

8.4 Setting Socket Options

- TCP_NODELAY, SO_BINDADDR, SO_TIMEOUT, SO_LINGER
  SO_SNDBUF, SO_RCVBUF, SO_KEEPALIVE, OOBINLINE, IP_TOS

8.5 Socket Exceptions

8.6 Sockets in GUI Applications

- Whois class (Example 8-7) and a graphical Whois client interface (Example 8-8)

Network Programming

Ch.9 Sockets for Servers

Unit-7

Outline

9.1 Using ServerSockets

9.2 Logging

9.3 Constructing Server Sockets

9.4 Getting Information About a Server Socket

9.5 Socket Options

9.6 HTTP Servers

9.1 Using ServerSockets

http://docs.oracle.com/javase/8/docs/api/java/net/ServerSocket.html

- ServerSocket: a server socket's job
  - Bind to a port
  - Listen for incoming data
  - Accept connections from remote machines
- Basic life cycle of a server program in Java
  1. Create a new ServerSocket on a particular port using a ServerSocket() constructor
  2. Listens for incoming connection attempts on that port using its accept() method
     - accept() blocks until a client attempts to make a connection
       - Returns a **Socket** object connecting the client and the server
  3. get input and output streams that communicate with the
     - Socket's getInputStream() method and getOutputStream() method
  4. The server and the client interact according to an agreed-upon protocol until it is time to close the connection
  5. The server, the client, or both close the connection
  6. The server returns to step 2 and waits for the next connection

Example 9-1. A Daytime Server – Iterative

- Daytime Protocol (RFC 867): TCP Port 13 (Examples 8-1 and 8-2 Client)
- Iterative server: there's one big loop, and in each pass through the loop a single connection is completely processed

```
import java.net.*;
import java.io.*;
import java.util.Date;

public class DaytimeServer {

  public final static int PORT = 13;

  public static void main(String[] args) {
   try (ServerSocket server = new ServerSocket(PORT)) {
     while (true) {
       try (Socket connection = server.accept()) {
         Writer out = new OutputStreamWriter(connection.getOutputStream());
         Date now = new Date();
         out.write(now.toString() +"\r\n");
         out.flush();
         connection.close();
       } catch (IOException ex) {}
     }
   } catch (IOException ex) {
     System.err.println(ex);
   }
  }
}
```

```
$ telnet localhost 13
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Sat Mar 30 16:15:10 EDT 2013
Connection closed by foreign host
```

1. creates a server socket that listens on port 13.
   - Note that ports 0-1023 are privileged by root
2. Calls accept(), stops here and waits. When a client does connect, it returns a Socket object
3. Serves the request: writes / returns the daytime
4. Closes the connection and loops back to accept next requests

- Java 7's try-with-resources used to autoclose the socket
- The client closes and the server throws Interrupted InterruptedIOException
  - The server gets ready to process the next incoming connection

Ctrl-C could terminate the program

Example 9-2. A Time Server

```
public class TimeServer {

  public final static int PORT = 37;

  public static void main(String[] args) {

   // The time protocol sets the epoch at 1900,
   // the Date class at 1970. This number
   // converts between them.
   long differenceBetweenEpochs = 2208988800L;

   try (ServerSocket server = new ServerSocket(PORT)) {
     while (true) {
       try (Socket connection = server.accept()) {
         OutputStream out = connection.getOutputStream();
         Date now = new Date();
         long msSince1970 = now.getTime();
         long secondsSince1970 = msSince1970/1000;
         long secondsSince1900 = secondsSince1970
             + differenceBetweenEpochs;
         byte[] time = new byte[4];
         time[0]
             = (byte) ((secondsSince1900 & 0x00000000FF000000L) >> 24);
         time[1]
             = (byte) ((secondsSince1900 & 0x0000000000FF0000L) >> 16);
         time[2]
             = (byte) ((secondsSince1900 & 0x000000000000FF00L) >> 8);
         time[3] = (byte) (secondsSince1900 & 0x00000000000000FFL);
         out.write(time);
         out.flush();
       } catch (IOException ex) {
         System.err.println(ex.getMessage());
       }
     }
   } catch (IOException ex) {
     System.err.println(ex);
   }
  }
}
```

- Time Protocol (RFC 868): TCP Port 37 (Example 8-3 Client)
  - The server sends a 4-byte, big-endian, unsigned integer
    - # of seconds since 12:00 A.M., January 1, 1900, GMT (the epoch)

Example 9-3. A Multithreaded Daytime Server

```
public class MultithreadedDaytimeServer {
  public final static int PORT = 13;
  public static void main(String[] args) {
   try (ServerSocket server = new ServerSocket(PORT)) {
     while (true) {
       try {
         Socket connection = server.accept();
         Thread task = new DaytimeThread(connection);
         task.start();
       } catch (IOException ex) {}
     }
    } catch (IOException ex) {
      System.err.println("Couldn't start server");
    }
  }
  private static class DaytimeThread extends Thread {
    private Socket connection;
    DaytimeThread(Socket connection) {
      this.connection = connection;
    }
    @Override
    public void run() {
      try {
        Writer out = new OutputStreamWriter(connection.getOutputStream());
        Date now = new Date();
        out.write(now.toString() +"\r\n");
        out.flush();
      } catch (IOException ex) {
        System.err.println(ex);
      } finally {
        try {
          connection.close();
        } catch (IOException e) {
          // ignore;
        }
      }
    }
  }
}
```

Does not use try-with-resources, or the main thread would close the socket as soon as it gets to the end of the while loop before the thread is spawned

A thread per connection design

Problem: Numerous roughly simultaneous incoming connections can cause it to spawn an indefinite number of threads

Example 9-4. A Thread-Pool Daytime Server

```
public class PooledDaytimeServer {
  public final static int PORT = 13;
  public static void main(String[] args) {
   ExecutorService pool = Executors.newFixedThreadPool(50);
   try (ServerSocket server = new ServerSocket(PORT)) {
     while (true) {
       try {
          Socket connection = server.accept();
          Callable<Void> task = new DaytimeTask(connection);
          pool.submit(task);
        } catch (IOException ex) {}
     }
    } catch (IOException ex) {
      System.err.println("Couldn't start server");
    }
  }
  private static class DaytimeTask implements Callable<Void> {
    private Socket connection;
    DaytimeTask(Socket connection) {
      this.connection = connection;
    }
    @Override
    public Void call() {
      try {
        Writer out = new OutputStreamWriter(connection.getOutputStream());
        Date now = new Date();
        out.write(now.toString() +"\r\n");
        out.flush();
      } catch (IOException ex) {
        System.err.println(ex);
      } finally {
        try {
          connection.close();
        } catch (IOException e) {
          // ignore;
        }
      }
      return null;
    }
  }
}
```

Does not use try-with-resources

A thread pool with 50 threads
– Private static class DaytimeTask

Example 9-5. An Echo Server (Thread Pool)

- Echo Protocol (RFC 862): TCP Port 7
  - The server sends the data back
  - the client is responsible for closing the connection

```
import java.net.*;
import java.io.*;
import java.util.concurrent.*;

public class EchoServer {
  public final static int PORT = 7;

  public static void main(String[] args) {

   ExecutorService pool = Executors.newFixedThreadPool(500);

   try (ServerSocket server = new ServerSocket(PORT)) {
     while (true) {
       try {
         Socket connection = server.accept();
         Callable<Void> task = new EchoTask(connection);
         pool.submit(task);
       } catch (IOException ex) {}
     }
    } catch (IOException ex) {
      System.err.println("Couldn't start server");
    }
  }
```

```
  private static class EchoTask implements Callable<Void> {

    private Socket connection;

    EchoTask(Socket connection) {
      this.connection = connection;
    }

    @Override
    public Void call() throws IOException {
      try {
        InputStream in = new BufferedInputStream(connection.getInputStream());
        OutputStream out = connection.getOutputStream();
        int c;
        while ((c = in.read()) != -1) {
          out.write(c);
          out.flush();
        }
      } catch (IOException ex) {
        System.err.println(ex);
      } finally {
        connection.close();
      }
      return null;
    }
  }
}
```

```
$ telnet rama.poly.edu 7
Trying 128.238.10.212...
Connected to rama.poly.edu.
Escape character is '^]'.
This is a test
This is a test
This is another test
This is another test
9876543210
9876543210
^]
telnet> close
Connection closed.
```

Closing Server Sockets

- Closing a ServerSocket
  - Frees a port on the local host, allowing another server to bind to the port
    - Closing a Socket object just frees the spawned end-to-end TCP connection
  - It also *breaks all currently open sockets* that the ServerSocket has accepted
  - ServerSocket is closed automatically when a program dies
    - However, it's good to close it as is no longer needed: three ways to close

1. Typical constrouctor

```
ServerSocket server = null;
try {
  server = new ServerSocket(port);
  // ... work with the server socket
} finally {
  if (server != null) {
    try {
      server.close();
    } catch (IOException ex) {
      // ignore
    }
  }
}
```

2. Uses noargs constructor and calls bind() later to prevent exception

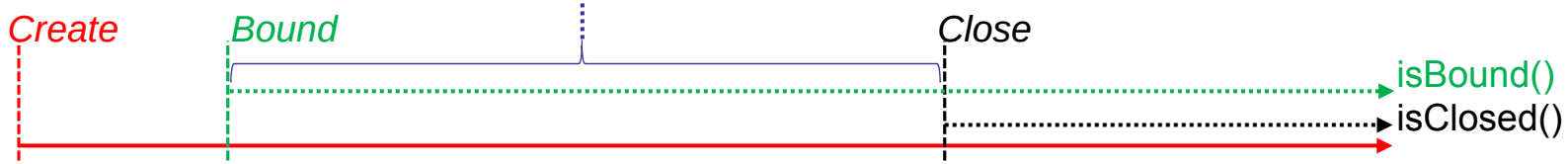
```
ServerSocket server = new ServerSocket();
try {
  SocketAddress address = new InetSocketAddress(port);
  server.bind(address);
  // ... work with the server socket
} finally {
  try {
    server.close();
  } catch (IOException ex) {
    // ignore
  }
}
```

3. Java 7+: AutoCloseable try-with-resources

```
try (ServerSocket server = new ServerSocket(port)) {
  // ... work with the server socket
}
```

- isBound(): whether the ServerSocket has been bound to a port

```
public static boolean isOpen(ServerSocket ss) {
  return ss.isBound() && !ss.isClosed();
}
```



9.2 Logging

http://docs.oracle.com/javase/8/docs/api/java/util/logging/package-frame.html
http://docs.oracle.com/javase/8/docs/api/java/util/logging/Logger.html

- Two primary things to store in the logs
  - Audit log: requests
  - Error log: server errors
    - The general rule of thumb: every line in the error log should be looked at and resolved
    - Do not keep debug logs in production; put it in another separate file
- **java.util.logging** package since Java 1.4
  - **Logger.getLogger()**: create one per class with a (dot-separated) log name

```
private final static Logger auditLogger = Logger.getLogger("requests");
```

    - Normally based on the packet name or class name
    - Loggers are thread safe
    - Multiple Logger objects can output to the same log, but usually exactly one log
  - log(): log messages with specified levels

```
catch (RuntimeException ex) {
  logger.log(Level.SEVERE, "unexpected error " + ex.getMessage(), ex);
}
```

    - Seven levels defined as named constants in **java.util.logging.Level**
      Level.SEVERE > .WARNING > .INFO > .CONFIG > .FINE > .FINER > Level.FINEST
      –Examples: Level.INFO for audit logs, and Level.WARNING or Level.SEVERE for error logs
      or Logger.info(), Logger.warning()/Logger.severe() instead
  - By default, the logs are just output to the console
    - Set log to file when launching the JVM
      –Djava.util.logging.config.file=_filename_

Example 9-6. A Daytime Server with Logs

- RuntimeException: cover most of the code and all of the network connections

```
public class LoggingDaytimeServer {

  public final static int PORT = 13;
  private final static Logger auditLogger = Logger.getLogger("requests");
  private final static Logger errorLogger = Logger.getLogger("errors");

  public static void main(String[] args) {

   ExecutorService pool = Executors.newFixedThreadPool(50);

   try (ServerSocket server = new ServerSocket(PORT)) {
     while (true) {
       try {
         Socket connection = server.accept();
         Callable<Void> task = new DaytimeTask(connection);
         pool.submit(task);
       } catch (IOException ex) {
         errorLogger.log(Level.SEVERE, "accept error", ex);
       } catch (RuntimeException ex) {
         errorLogger.log(Level.SEVERE, "unexpected error "
       }                                       + ex.getMessage(), ex);
     }
   } catch (IOException ex) {
     errorLogger.log(Level.SEVERE, "Couldn't start server", ex);
   } catch (RuntimeException ex) {
     errorLogger.log(Level.SEVERE, "Couldn't start server: "
   }                                         + ex.getMessage(), ex);
  }
```

```
Apr 13, 2013 8:54:50 AM LoggingDaytimeServer$DaytimeTask call
INFO: Sat Apr 13 08:54:50 EDT 2013 /0:0:0:0:0:0:0:1:56665
Apr 13, 2013 8:55:08 AM LoggingDaytimeServer$DaytimeTask call
INFO: Sat Apr 13 08:55:08 EDT 2013 /0:0:0:0:0:0:0:1:56666
Apr 13, 2013 8:55:16 AM LoggingDaytimeServer$DaytimeTask call
INFO: Sat Apr 13 08:55:16 EDT 2013 /0:0:0:0:0:0:0:1:56667
```

```
private static class DaytimeTask implements Callable<Void> {

  private Socket connection;

  DaytimeTask(Socket connection) {
    this.connection = connection;
  }
  @Override
  public Void call() {
    try {
      Date now = new Date();
      // write the log entry first in case the client disconnects
      auditLogger.info(now + " " + connection.getRemoteSocketAddress());
      Writer out = new OutputStreamWriter(connection.getOutputStream());
      out.write(now.toString() +"\r\n");
      out.flush();
    } catch (IOException ex) {
        // client disconnected; ignore;
    } finally {
      try {
        connection.close();
      } catch (IOException ex) {
        // ignore;
      }
    }
    return null;
  }
}
```

9.3 Constructing ServerSockets

- 4 constructors, throw BindException

```
public ServerSocket(int port) throws BindException, IOException
public ServerSocket(int port, int queueLength)
    throws BindException, IOException
public ServerSocket(int port, int queueLength, InetAddress bindAddress)
    throws IOException
public ServerSocket() throws IOException
```

  - port: the port to listen to
    - 0: the system will select an available port (anonymous port)
  - queuelength: hold incoming connection request
  - bindAddress: specify the local network interface to bind to
    - By default, the server socket listens on all the interfaces and IP addresses of the host
- Constructing without binding
  - bind() later; port 0 for anonymous port

```
public void bind(SocketAddress endpoint) throws IOException
public void bind(SocketAddress endpoint, int queueLength) throws IOException
```

```
ServerSocket ss = new ServerSocket();
// set socket options...
SocketAddress  http = new InetSocketAddress(80);
ss.bind(http);
```

Example 9-8. LocalPortScanner

- Look for local ports (for ports 1024 and above)
  - Attempt to open a server on that port

```
import java.io.*;
import java.net.*;

public class LocalPortScanner {
  public static void main(String[] args) {

    for (int port = 1; port <= 65535; port++) {
      try {
        // the next line will fail and drop into the catch block if
        // there is already a server running on the port
        ServerSocket server = new ServerSocket(port);
      } catch (IOException ex) {
        System.out.println("There is a server on port " + port + ".");
      }
    }
  }
}
```

9-4. Getting Information About a Server Socket

- Two getter methods
  - getInetAddress(): return the address being used for an accepted connection
    - Return null if not yet bound

```
ServerSocket httpd = new ServerSocket(80);
InetAddress ia = httpd.getInetAddress();
```

  - getLocalPort(): find out what port is listening on (for anonymous port)

```
public int getLocalPort()
```

- toString(): for debugging

```
ServerSocket[addr=0.0.0.0,port=0,localport=5776]
```

- Example 9-9. RandomPort
  - Bind to an anonymous port

```
import java.io.*;
import java.net.*;

public class RandomPort {

  public static void main(String[] args) {
    try {
      ServerSocket server = new ServerSocket(0);
      System.out.println("This server runs on port "
          + server.getLocalPort());
    } catch (IOException ex) {
      System.err.println(ex);
    }
  }
}
```

```
$ java RandomPort
This server runs on port 1154
D:\JAVA\JNP4\examples\9>java RandomPort
This server runs on port 1155
D:\JAVA\JNP4\examples\9>java RandomPort
This server runs on port 1156
```

9-5 Socket Options

Three options supported for server sockets

- SO_TIMEOUT: timeout in ms for accept()

```
public void setSoTimeout(int timeout) throws SocketException
public int  getSoTimeout() throws IOException
```

  - Set before calling accept(). Can't change while accept() is waiting
  - SocketTimeoutException thrown

```
try (ServerSocket server = new ServerSocket(port)) {
  server.setSoTimeout(30000); // block for no more than 30 seconds
  try {
    Socket s = server.accept();
    // handle the connection
    // ...
  } catch (SocketTimeoutException ex) {
    System.err.println("No connection within 30 seconds");
  }
} catch (IOException ex) {
  System.err.println("Unexpected IOException: " + e);
}
```

```
public void printSoTimeout(ServerSocket server) {
  int timeout = server.getSoTimeOut();
  if (timeout > 0) {
    System.out.println(server + " will time out after "
        + timeout + "milliseconds.");
  } else if (timeout == 0) {
    System.out.println(server + " will never time out.");
  } else {
    System.out.println("Impossible condition occurred in " + server);
    System.out.println("Timeout cannot be less than zero." );
  }
}
```

- SO_REUSEADDR: allowed to bind a used port even there might still be data

```
public boolean getReuseAddress() throws SocketException
public void setReuseAddress(boolean on) throws SocketException
```

  - Default is platform dependent; true on Linux and Mac OS X by default
- SO_RCVBUF: set the default receive buffer size for the accepted socket

```
public int  getReceiveBufferSize() throws SocketException
public void setReceiveBufferSize(int size) throws SocketException
```

- Class of Service: setPerformancePreferences(), hint for TCP stack

```
public void setPerformancePreferences(int connectionTime, int latency,
    int bandwidth)
```

```
ss.setPerformancePreferences(2, 1, 3);
```

  - Many implementations including Android ignore it

9-6 HTTP Servers

- Custom servers to optimize for a particular task instead of full function servers
- Example 9-10. An HTTP server that serves a single file (thread pool)
  - A server that always sends out the same file, no matter what the request
  - Command line arguments: filename, local-port (def: 80), encoding (def: UTF-8)
- Example 9-11. A Redirector (a thread per connection; simple to implement)
  - Reads a URL and a port number from the command line, opens a server socket on the port, and redirects all requests to the URL
  - Command line arguments: URL, local-port (def: 80)
- A Full-Fledged HTTP Server
  - Example 9-12. JHTTP web server (thread pool)
    - Command line arguments: Document Root Directory, local-port (def: 80)
  - Example 9-13. The runnable class that handles HTTP requests

Example 9-10. An HTTP Server That Serves a Single File

```
public class SingleFileHTTPServer {
  private static final Logger logger = Logger.getLogger("SingleFileHTTPServer");
  private final byte[] content;  private final int port;
  private final byte[] header;   private final String encoding;
  public SingleFileHTTPServer(String data, String encoding,
      String mimeType, int port) throws UnsupportedEncodingException {
    this(data.getBytes(encoding), encoding, mimeType, port);
  }
  public SingleFileHTTPServer(
      byte[] data, String encoding, String mimeType, int port) {
    this.content = data;  this.port = port;  this.encoding = encoding;
    String header = "HTTP/1.0 200 OK\r\n"
        + "Server: OneFile 2.0\r\n"
        + "Content-length: " + this.content.length + "\r\n"
        + "Content-type: " + mimeType + "; charset=" + encoding + "\r\n\r\n";
    this.header = header.getBytes(Charset.forName("US-ASCII"));
  }
  public void start() {
    ExecutorService pool = Executors.newFixedThreadPool(100);
    try (ServerSocket server = new ServerSocket(this.port)) {
      logger.info("Accepting connections on port " + server.getLocalPort());
      logger.info("Data to be sent:");
      logger.info(new String(this.content, encoding));
      while (true) {
        try {
          Socket connection = server.accept();
          pool.submit(new HTTPHandler(connection));
        } catch (IOException ex) {
          logger.log(Level.WARNING, "Exception accepting connection", ex);
        } catch (RuntimeException ex) {
          logger.log(Level.SEVERE, "Unexpected error", ex);
        } }
    } catch (IOException ex) {
      logger.log(Level.SEVERE, "Could not start server", ex);
    }
  }
  public static void main(String[] args) {
    // set the port to listen on
    int port;
    try {
      port = Integer.parseInt(args[1]);
      if (port < 1 || port > 65535) port = 80;
    } catch (RuntimeException ex) {
      port = 80;
    }
    String encoding = "UTF-8";
    if (args.length > 2) encoding = args[2];
    try {
      Path path = Paths.get(args[0]);;
      byte[] data = Files.readAllBytes(path);
      String contentType = URLConnection.getFileNameMap().getContentTypeFor(args[0]);
      SingleFileHTTPServer server = new SingleFileHTTPServer(data, encoding,
          contentType, port);
      server.start();
    } catch (ArrayIndexOutOfBoundsException ex) {
      System.out.println(
          "Usage: java SingleFileHTTPServer filename port encoding");
    } catch (IOException ex) {
      logger.severe(ex.getMessage());
    } }
}
```

Thread pool

Arguments:
- filename
- local-port (def: 80)
- encoding (def: UTF-8)

Map file extension

```
private class HTTPHandler implements Callable<Void> {
  private final Socket connection;

  HTTPHandler(Socket connection) {
    this.connection = connection;
  }

  @Override
  public Void call() throws IOException {
    try {
      OutputStream out = new BufferedOutputStream(
                           connection.getOutputStream()
                         );
      InputStream in = new BufferedInputStream(
                           connection.getInputStream()
                         );
      // read the first line only; that's all we need
      StringBuilder request = new StringBuilder(80);
      while (true) {
        int c = in.read();
        if (c == '\r' || c == '\n' || c == -1) break;
        request.append((char) c);
      }
      // If this is HTTP/1.0 or later send a MIME header
      if (request.toString().indexOf("HTTP/") != -1) {
        out.write(header);
      }
      out.write(content);
      out.flush();
    } catch (IOException ex) {
      logger.log(Level.WARNING, "Error writing to client", ex);
    } finally {
      connection.close();
    }
    return null;
  }
}
```

```
% telnet macfaq.dialup.cloud9.net 80
Trying 168.100.203.234...
Connected to macfaq.dialup.cloud9.net.
Escape character is '^]'.
GET / HTTP/1.0
HTTP/1.0 200 OK
Server: OneFile 2.0
Content-length: 959
Content-type: text/html; charset=UTF-8

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<HEAD>;
<TITLE>Under Construction</TITLE>
</HEAD>

<BODY>
...
```

Example 9-11. An HTTP Redirector

```
public class Redirector {
  private static final Logger logger = Logger.getLogger("Redirector");
  private final int port;
  private final String newSite;
  public Redirector(String newSite, int port) {
    this.port = port;
    this.newSite = newSite;
  }
  public void start() {
    try (ServerSocket server = new ServerSocket(port)) {
      logger.info("Redirecting connections on port "
          + server.getLocalPort() + " to " + newSite);
      while (true) {
        try {
          Socket s = server.accept();
          Thread t = new RedirectThread(s);
          t.start();
        } catch (IOException ex) {
          logger.warning("Exception accepting connection");
        } catch (RuntimeException ex) {
          logger.log(Level.SEVERE, "Unexpected error", ex);
        }
      }
    } catch (BindException ex) {
      logger.log(Level.SEVERE, "Could not start server.", ex);
    } catch (IOException ex) {
      logger.log(Level.SEVERE, "Error opening server socket", ex);
    }
  }
  public static void main(String[] args) {
    int thePort;
    String theSite;
    try {
      theSite = args[0];
      // trim trailing slash
      if (theSite.endsWith("/")) {
        theSite = theSite.substring(0, theSite.length() - 1);
      }
    } catch (RuntimeException ex) {
      System.out.println(
          "Usage: java Redirector http://www.newsite.com/ port");
      return;
    }
    try {
      thePort = Integer.parseInt(args[1]);
    } catch (RuntimeException ex) {
      thePort = 80;
    }
    Redirector redirector = new Redirector(theSite, thePort);
    redirector.start();
  }
```

A thread per connection

Arguments:
URL
local-port (def: 80)

```
private class RedirectThread extends Thread {
  private final Socket connection;
  RedirectThread(Socket s) {
    this.connection = s;
  }
  public void run() {
    try {
      Writer out = new BufferedWriter(
                   new OutputStreamWriter(
                    connection.getOutputStream(),"US-ASCII"));
      Reader in = new InputStreamReader(
                  new BufferedInputStream(
                   connection.getInputStream()));
      // read the first line only; that's all we need
      StringBuilder request = new StringBuilder(80);
      while (true) {
        int c = in.read();
        if (c == '\r' || c == '\n' || c == -1) break;
        request.append((char) c);
      }
      String get = request.toString();
      String[] pieces = get.split("\\w*");
      String theFile = pieces[1];
      // If this is HTTP/1.0 or later send a MIME header
      if (get.indexOf("HTTP") != -1) {
        out.write("HTTP/1.0 302 FOUND\r\n");
        Date now = new Date();
        out.write("Date: " + now + "\r\n");
        out.write("Server: Redirector 1.1\r\n");
        out.write("Location: " + newSite + theFile + "\r\n");
        out.write("Content-type: text/html\r\n\r\n");
        out.flush();
      } // Not all browsers support redirection so we need to
        // produce HTML that says where the document has moved to.
      out.write("<HTML><HEAD><TITLE>Document moved</TITLE></HEAD>\r\n");
      out.write("<BODY><H1>Document moved</H1>\r\n");
      out.write("The document " + theFile
       + " has moved to\r\n<A HREF=\"" + newSite + theFile + "\">"
       + newSite  + theFile
       + "</A>.\r\n Please update your bookmarks<P>");
      out.write("</BODY></HTML>\r\n");
      out.flush();
      logger.log(Level.INFO,
          "Redirected " + connection.getRemoteSocketAddress());
    } catch(IOException ex) {
      logger.log(Level.WARNING,
          "Error talking to " + connection.getRemoteSocketAddress(), ex);
    } finally {
      try {
        connection.close();
      } catch (IOException ex) {}
} } }
```

Bug: “ “

```
D:\JAVA\JNP4\examples\09> java Redirector http://www.cafeconleche.org/
Redirecting connections on port 80 to http://www.cafeconleche.org/
```

```
% telnet macfaq.dialup.cloud9.net 80
Trying 168.100.203.234...
Connected to macfaq.dialup.cloud9.net.
Escape character is '^]'.
GET / HTTP/1.0
HTTP/1.0 302 FOUND
Date: Sun Mar 31 12:38:42 EDT 2013
Server: Redirector 1.1
Location: http://www.cafeconleche.org/
Content-type: text/html

<HTML><HEAD><TITLE>Document moved</TITLE></HEAD>
<BODY><H1>Document moved</H1>
The document / has moved to
<A HREF="http://www.cafeconleche.org/">http://www.cafeconleche.
 org/</A>.
 Please update your bookmarks<P></BODY></HTML>
Connection closed by foreign host.
```

Example 9-12. JHTTP Web Server

```
public class JHTTP {

  private static final Logger logger = Logger.getLogger(
      JHTTP.class.getCanonicalName());
  private static final int NUM_THREADS = 50;
  private static final String INDEX_FILE = "index.html";

  private final File rootDirectory;
  private final int port;

  public JHTTP(File rootDirectory, int port) throws IOException {

    if (!rootDirectory.isDirectory()) {
      throw new IOException(rootDirectory
          + " does not exist as a directory");
    }
    this.rootDirectory = rootDirectory;
    this.port = port;
  }

  public void start() throws IOException {
    ExecutorService pool = Executors.newFixedThreadPool(NUM_THREADS);
    try (ServerSocket server = new ServerSocket(port)) {
      logger.info("Accepting connections on port " + server.getLocalPort());
      logger.info("Document Root: " + rootDirectory);

      while (true) {
        try {
          Socket request = server.accept();
          Runnable r = new RequestProcessor(
              rootDirectory, INDEX_FILE, request);
          pool.submit(r);
        } catch (IOException ex) {
          logger.log(Level.WARNING, "Error accepting connection", ex);
        }
      }
    }
  }
```

Thread pool

```
  public static void main(String[] args) {

    // get the Document root
    File docroot;
    try {
      docroot = new File(args[0]);
    } catch (ArrayIndexOutOfBoundsException ex) {
      System.out.println("Usage: java JHTTP docroot port");
      return;
    }

    // set the port to listen on
    int port;
    try {
      port = Integer.parseInt(args[1]);
      if (port < 0 || port > 65535) port = 80;
    } catch (RuntimeException ex) {
      port = 80;
    }

    try {
      JHTTP webserver = new JHTTP(docroot, port);
      webserver.start();
    } catch (IOException ex) {
      logger.log(Level.SEVERE, "Server could not start", ex);
    }
  }
}
```

Arguments:
DocRootDirectory
local-port (def: 80)

Example 9-13. The Runnable Class That Handles HTTP Requests

```
public class RequestProcessor implements Runnable {

  private final static Logger logger
                          = Logger.getLogger(
      RequestProcessor.class.getCanonicalName());
  private File rootDirectory;
  private String indexFileName = "index.html";
  private Socket connection;

  public RequestProcessor(File rootDirectory,
      String indexFileName, Socket connection) {

    if (rootDirectory.isFile()) {
      throw new IllegalArgumentException(
"rootDirectory must be a directory, not a file");
    }
    try {
      rootDirectory = rootDirectory.getCanonicalFile();
    } catch (IOException ex) {
    }
    this.rootDirectory = rootDirectory;
    if (indexFileName != null)
        this.indexFileName = indexFileName;
    this.connection = connection;
  }
  private void sendHeader(Writer out, String responseCode,
      String contentType, int length)
      throws IOException {
        out.write(responseCode + "\r\n");
        Date now = new Date();
        out.write("Date: " + now + "\r\n");
        out.write("Server: JHTTP 2.0\r\n");
        out.write("Content-length: " + length + "\r\n");
        out.write("Content-type: " + contentType + "\r\n\r\n");
        out.flush();
      }
  }
```

```
@Override
public void run() {
  // for security checks
  String root = rootDirectory.getPath();
  try {
    OutputStream raw = new BufferedOutputStream(
                        connection.getOutputStream()
                       );
    Writer out = new OutputStreamWriter(raw);
    Reader in = new InputStreamReader(
                new BufferedInputStream(
                 connection.getInputStream()
                ),"US-ASCII"
               );
    StringBuilder requestLine = new StringBuilder();
    while (true) {
      int c = in.read();
      if (c == '\r' || c == '\n') break;
      requestLine.append((char) c);
    }

    String get = requestLine.toString();

    logger.info(connection.getRemoteSocketAddress() + " " + get);

    String[] tokens = get.split("\\s+");
    String method = tokens[0];
    String version = "";
    if (method.equals("GET")) {
      String fileName = tokens[1];
      if (fileName.endsWith("/")) fileName += indexFileName;
      String contentType =
          URLConnection.getFileNameMap().getContentTypeFor(fileName);
      if (tokens.length > 2) {
        version = tokens[2];
      }

      File theFile = new File(rootDirectory,
          fileName.substring(1, fileName.length()));

      if (theFile.canRead()
          // Don't let clients outside the document root
          && theFile.getCanonicalPath().startsWith(root)) {
```

Case: normal

Map file extension

```
        byte[] theData = Files.readAllBytes(theFile.toPath());
        if (version.startsWith("HTTP/")) { // send a MIME header
          sendHeader(out, "HTTP/1.0 200 OK",
                                  contentType, theData.length);
        }
        // send the file; it may be an image or other binary data
        // so use the underlying output stream
        // instead of the writer
        raw.write(theData);
        raw.flush();
      } else { // can't find the file
        String body = new StringBuilder("<HTML>\r\n")
            .append("<HEAD><TITLE>File Not Found</TITLE>\r\n")
            .append("</HEAD>\r\n")
            .append("<BODY>")
            .append("<H1>HTTP Error 404: File Not Found</H1>\r\n")
            .append("</BODY></HTML>\r\n").toString();
        if (version.startsWith("HTTP/")) { // send a MIME header
          sendHeader(out, "HTTP/1.0 404 File Not Found",
              "text/html; charset=utf-8", body.length());
        }
        out.write(body);
        out.flush();
      }
    } else { // method does not equal "GET"
      String body = new StringBuilder("<HTML>\r\n")
          .append("<HEAD><TITLE>Not Implemented</TITLE>\r\n")
          .append("</HEAD>\r\n")
          .append("<BODY>")
          .append("<H1>HTTP Error 501: Not Implemented</H1>\r\n")
          .append("</BODY></HTML>\r\n").toString();
      if (version.startsWith("HTTP/")) { // send a MIME header
        sendHeader(out, "HTTP/1.0 501 Not Implemented",
                  "text/html; charset=utf-8", body.length());
      }
      out.write(body);
      out.flush();
    }
    catch (IOException ex) {
    logger.log(Level.WARNING,"Error talking to "
                    + connection.getRemoteSocketAddress(),ex);
    finally {
    try {
      connection.close();
    }
    catch (IOException ex) {}
  }
}
```

Summary

9.1 Using ServerSockets
- ServerSocket
- Iterative Server: DaytimeServer (Example 9-1), TimeServer (Example 9-2)
- Multitheaded: MultithreadedDaytimeServer (Example 9-3)
- Thread-Pool: PooledDaytimeServer (Example 9-4), EchoServer (Example 9-5)

9.2 Logging
- Logger
- seven levels: SEVERE > WARNING > INFO > CONFIG > FINE > FINER > FINEST
- LoggingDaytimeServer (Example 9-6, pooled)

9.3 Constructing Server Sockets
- LocalPortScanner (Example 9-8)

9.4 Getting Information About a Server Socket
- getInetAddress(), getLocalPort()
- RandomPort (Example 9-9)

9.5 Socket Options
- SO_TIMEOUT, SO_REUSEADDR, SO_RCVBUF

9.6 HTTP Servers
- SingleFileHTTPServer (Example 9-10, pooled)
- Redirector (Example 9-11, a thread per connection)
- A Full-Fledged HTTP Server
  - JHTTP (Example 9-12, pooled) + RequestProcessor (Example 9-13)

Ch.10 Secure Sockets

Unit-8

Outline

10.1 Secure Communications

10.2 Creating Secure Client Sockets

10.3 Choosing the Cipher Suites

10.4 Event Handlers

10.5 Session Management

10.6 Client Mode

10.7 Creating Secure Server Sockets

10.8 Configuring SSL Server Sockets

10.1 Secure Communications

- Issues: it's easy to copy packets by crackers and governments
- Java Secure Sockets Extension (JSSE) supports
  - Secure Sockets Layer (SSL) Version 3
  - Transport Layer Security (TLS)

  Option in Java 1.2/1.3 (javax.* package); Integrated in Java 1.4+
  http://docs.oracle.com/javase/8/docs/technotes/guides/security/jsse/JSSERefGuide.html
- Traditional methods
  - Secret key (symmetric) encryption: same key is used to encrypt and decrypt
  - Public key (asymmetric) encryption: different keys are used
    - Used for authentication and message integrity checking
      - Angela: Encrypt(Msg, *PriKeyA*) → Gus: Decrypt(Encrypt(Msg, *PriKeyA*), *PubKeyA*) = Msg
      - Angela: ***Msg'*** = Encrypt(Msg, *PriKeyA*, *PubKeyG*)
        → Gus: Decrypt(**Msg'**, *PubKeyA* , *PriKeyG*) = Msg
      - Man-in-the-Middle: Angela and Gus can not verify each other's public key, and GadMan pretends to be
        Angela: ***Msg'*** = Encrypt(Msg, *PriKeyA*, *PubKeyB*)
        BadMan: Decrypt(**Msg'**, *PubKeyA*, *PriKeyB*) = Msg
        Encrypt(Msg, *PriKeyB*, *PubKeyG*)
        → Gus: Decrypt(**Msg'**, *PubKeyB*, *PriKeyG*) = Msg
    - Trusted third-party certification authority to verify their public keys
    - Different authentication and encryption algorithms as well as the key length are provided

JSSE Four Packages

- javax.net.ssl
  - The abstract classes that define Java's API for secure network communication
- javax.net
  - The abstract socket factory classes used instead of constructors to create secure sockets
- java.security.cert
  - The classes for handling the public-key certificates needed for SSL
- com.sun.net.ssl
  - The concrete classes that implement the encryption algorithms and protocols in Sun's reference implementation of the JSSE
  - Technically, these are not part of the JSSE standard

10.2 Creating Secure Client Sockets

- createSocket() from javax.net.ssl.SSLSocketFactory

```
SocketFactory factory = SSLSocketFactory.getDefault();
Socket socket = factory.createSocket("login.ibiblio.org", 7000);
```

```
Socket socket = new Socket("time.nist.gov", 13);
```

- Five createSocket() methods to build an SSLSocket

```
public abstract Socket createSocket(String host, int port)
    throws IOException, UnknownHostException
public abstract Socket createSocket(InetAddress host, int port)
    throws IOException
public abstract Socket createSocket(String host, int port,
    InetAddress interface, int localPort)
    throws IOException, UnknownHostException
public abstract Socket createSocket(InetAddress host, int port,
    InetAddress interface, int localPort)
    throws IOException, UnknownHostException
public abstract Socket createSocket(Socket proxy, String host, int port,
    boolean autoClose) throws IOException
```

  - Actually return javax.net.ssl.SSLSocket, a subclass of java.net.Socket
  - Call getInputStream() and getOutputStream() as usual to get streams
    - Output/Write as usual

```
SSLSocketFactory factory
    = (SSLSocketFactory) SSLSocketFactory.getDefault();
Socket socket = factory.createSocket("login.ibiblio.org", 7000);

Writer out = new OutputStreamWriter(socket.getOutputStream(),
    "US-ASCII");
out.write("Name: John Smith\r\n");
out.write("Product-ID: 67X-89\r\n");
out.write("Address: 1280 Deniston Blvd, NY NY 10003\r\n");
out.write("Card number: 4000-1234-5678-9017\r\n");
out.write("Expires: 08/05\r\n");
out.flush();
```

Example 10-1. HTTPSClient

```
import java.io.*;
import javax.net.ssl.*;

public class HTTPSClient {

  public static void main(String[] args) {

    if (args.length == 0) {
      System.out.println("Usage: java HTTPSClient2 host");
      return;
    }

    int port = 443; // default https port
    String host = args[0];
    SSLSocketFactory factory
        = (SSLSocketFactory) SSLSocketFactory.getDefault();
    SSLSocket socket = null;
    try {
      socket = (SSLSocket) factory.createSocket(host, port);

      // enable all the suites
      String[] supported = socket.getSupportedCipherSuites();
      socket.setEnabledCipherSuites(supported);

      Writer out = new OutputStreamWriter(socket.getOutputStream(), "UTF-8");
      // https requires the full URL in the GET line
      out.write("GET http://" + host + "/ HTTP/1.1\r\n");
      out.write("Host: " + host + "\r\n");
      out.write("\r\n");
      out.flush();

      // read response
      BufferedReader in = new BufferedReader(
          new InputStreamReader(socket.getInputStream()));

      // read the header
      String s;
      while (!(s = in.readLine()).equals("")) {
        System.out.println(s);
      }
      System.out.println();

      // read the length
      String contentLength = in.readLine();
      int length = Integer.MAX_VALUE;
      try {
        length = Integer.parseInt(contentLength.trim(), 16);
      } catch (NumberFormatException ex) {
        // This server doesn't send the content-length
        // in the first line of the response body
      }
      System.out.println(contentLength);

      int c;
      int i = 0;
      while ((c = in.read()) != -1 && i++ < length) {
        System.out.write(c);
      }

      System.out.println();
    } catch (IOException ex) {
      System.err.println(ex);
    } finally {
      try {
        if (socket != null) socket.close();
      } catch (IOException e) {}
    }
  }
}
```

See next

```
% java HTTPSClient www.usps.com
HTTP/1.1 200 OK
Server: IBM_HTTP_Server
Cache-Control: max-age=0
Expires: Sun, 31 Mar 2013 17:29:33 GMT
Content-Type: text/html
Date: Sun, 31 Mar 2013 18:00:14 GMT
Transfer-Encoding:  chunked
Connection: keep-alive
Connection: Transfer-Encoding

00004000

<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
```

- It may be slower to respond
- If see exception like “No trusted certificate Found,” try upgrading JDK

10.3 Choosing the Cipher Suites

- Different implementations of the JSSE support *different combinations of authentication and encryption algorithms*
  - Oracle bundles with Java 7 only supports 128-bit AES encryption
- getSupportedCipherSuites()

```
public abstract String[] getSupportedCipherSuites()
```

  - tells which combination of algorithms is available on a given socket
- getEnabledCipherSuites()

```
public abstract String[] getEnabledCipherSuites()
```

  - tells which suites this socket is willing to use
- setEnabledCipherSuites()

```
public abstract void setEnabledCipherSuites(String[] suites)
```

  - change the suites the client attempts to use
  - Support list of Oracle's JDK 1.7 (the first 28 members are default enabled)
    - Four parts: protocol, key exchange algorithm, encryption algorithm, and checksum

SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA

- Secure Sockets Layer Version 3;
- Diffie-Hellman method for key agreement;
- no authentication;
- DES encryption with 40-bit keys;
- Cipher Block Chaining, and
- the Secure Hash Algorithm checksum

- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_RSA_WITH_AES_128_CBC_SHA256
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA256
- TLS_ECDH_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA256
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA256
- TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
- TLS_RSA_WITH_AES_128_CBC_SHA
- TLS_ECDH_ECDSA_WITH_AES_128_CBC_SHA
- TLS_ECDH_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_RSA_WITH_AES_128_CBC_SHA
- TLS_DHE_DSS_WITH_AES_128_CBC_SHA
- TLS_ECDHE_ECDSA_WITH_RC4_128_SHA
- TLS_ECDHE_RSA_WITH_RC4_128_SHA
- SSL_RSA_WITH_RC4_128_SHA
- TLS_ECDH_ECDSA_WITH_RC4_128_SHA
- TLS_ECDH_RSA_WITH_RC4_128_SHA
- TLS_ECDHE_ECDSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDH_ECDSA_WITH_3DES_EDE_CBC_SHA
- TLS_ECDH_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA
- SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA
- SSL_RSA_WITH_RC4_128_MD5
- TLS_EMPTY_RENEGOTIATION_INFO_SCSV
- TLS_DH_anon_WITH_AES_128_CBC_SHA256
- TLS_ECDH_anon_WITH_AES_128_CBC_SHA
- TLS_DH_anon_WITH_AES_128_CBC_SHA
- TLS_ECDH_anon_WITH_RC4_128_SHA
- SSL_DH_anon_WITH_RC4_128_MD5
- TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA
- SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
- TLS_RSA_WITH_NULL_SHA256
- TLS_ECDHE_ECDSA_WITH_NULL_SHA
- TLS_ECDHE_RSA_WITH_NULL_SHA
- SSL_RSA_WITH_NULL_SHA
- TLS_ECDH_ECDSA_WITH_NULL_SHA
- TLS_ECDH_RSA_WITH_NULL_SHA
- TLS_ECDH_anon_WITH_NULL_SHA
- SSL_RSA_WITH_NULL_MD5
- SSL_RSA_WITH_DES_CBC_SHA
- SSL_DHE_RSA_WITH_DES_CBC_SHA
- SSL_DHE_DSS_WITH_DES_CBC_SHA
- SSL_DH_anon_WITH_DES_CBC_SHA
- SSL_RSA_EXPORT_WITH_RC4_40_MD5
- SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
- SSL_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA
- SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA
- SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
- TLS_KRB5_WITH_RC4_128_SHA
- TLS_KRB5_WITH_RC4_128_MD5
- TLS_KRB5_WITH_3DES_EDE_CBC_SHA
- TLS_KRB5_WITH_3DES_EDE_CBC_MD5
- TLS_KRB5_WITH_DES_CBC_SHA
- TLS_KRB5_WITH_DES_CBC_MD5
- TLS_KRB5_EXPORT_WITH_RC4_40_SHA
- TLS_KRB5_EXPORT_WITH_RC4_40_MD5
- TLS_KRB5_EXPORT_WITH_DES_CBC_40_SHA
- TLS_KRB5_EXPORT_WITH_DES_CBC_40_MD5

Cipher Suite

- If you want nonauthenticated transactions or authenticated but unencrypted transactions, you must enable those suites explicitly with the setEnabledCipherSuites() method

```
String[] strongSuites = {"TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256"};
socket.setEnabledCipherSuites(strongSuites);
```

  - If the other side of the connection doesn't support this encryption protocol, the socket will throw an exception when they try to read from or write to it
    - Ensuring that no confidential information is accidentally transmitted over a weak channel
- Avoid any of these suites that contain NULL, ANON, or EXPORT in their names
- Suggested: TLS_ECDHE_****_SHA256/384
  - TLS_ECDHE_ECDSA_WITH_AES_128_CBC_SHA256
    - Reasonably secure
  - TLS_ECDHE_ECDSA_WITH_AES_256_CBC_SHA256
    - Stronger
  - DES/AES: block ciphers (certain number of bits at a time)
    - DES: 64 bits
    - AES: 128, 192 or 256 bits
    - Faster, it's problematic for user-centered protocols such as chat and telnet
  - RC4: stream cipher that encrypts one byte at a time

10.4 Event Handler

- Authenticated network communications are slower
  - It may take several seconds, so you may deal with the connection asynchronously
- JSSE uses the standard Java event model to notify programs when the handshaking between client and server is complete
  - SSLSocket registers HandshakeCompletedListener objects

```
public abstract void addHandshakeCompletedListener(
    HandshakeCompletedListener listener)
public abstract void removeHandshakeCompletedListener(
    HandshakeCompletedListener listener) throws IllegalArgumentException
```

  - HandshakeCompletedListener interface
    - Declare handshakeCompleted() method to receive an argument HandshakeCompletedEvent

```
public void handshakeCompleted(HandshakeCompletedEvent event)
public class HandshakeCompletedEvent extends java.util.EventObject
```

    - Four methods for getting information about the event

```
public SSLSession getSession()
public String getCipherSuite()
public X509Certificate[] getPeerCertificateChain()
    throws SSLPeerUnverifiedException
public SSLSocket getSocket()
```

10.5 Session Management

- SSL allows sessions to extend over multiple sockets
  - Multiple sockets within the same session use the same set of public/private keys
- JSSE represents by instances of the SSLSession interface
  - Reuses the session's keys automatically if
    - Multiple secure sockets to one host on one port are opened
    - Within a reasonably short period of time
  - In high security applications, you may want to disallow session-sharing between sockets or force reauthentication of a session
- getSession() method of SSLSocket returns the Session

```
public abstract SSLSession getSession()
```

  - Get various information about the session

```
public byte[] getId()
public SSLSessionContext getSessionContext()
public long getCreationTime()
public long getLastAccessedTime()
public void invalidate()
public void putValue(String name, Object value)
public Object getValue(String name)
public void removeValue(String name)
public String[] getValueNames()
public X509Certificate[] getPeerCertificateChain()
 throws SSLPeerUnverifiedException
public String getCipherSuite()
public String getPeerHost()
```

- setEnableSessionCreation() method
  - To allow/disallow session

```
public abstract void setEnableSessionCreation(boolean allowSessions)
public abstract boolean getEnableSessionCreation()
```

- startHandshake() method
  - To reauthenticate a connection

```
public abstract void startHandshake() throws IOException
```

10.6 Client Mode

- Usually the server is required to authenticate itself, but the client doesn't
  - i.e. Amazon proves to the browsers that it is indeed Amazon
- setUseClientMode() method determines whether the socket needs to use authentication in its first handshake
  - When true is passed in, the socket is in client mode (won't offer to authenticate itself)

```
public abstract void setUseClientMode(boolean mode)
    throws IllegalArgumentException
public abstract boolean getUseClientMode()
```

  - Can be set only once for any given socket
- setNeedClientAuth() method requires that all clients also need to authenticate themselves

```
public abstract void setNeedClientAuth(boolean needsAuthentication)
    throws IllegalArgumentException
public abstract boolean getNeedClientAuth()
```

10.7 Creating Secure Server Sockets

- javax.net.SSLServerSocket (SSL-enabled server sockets) created by abstract factory class javax.net.SSLServerSocketFactory with createServerSocket()

```
public abstract class SSLServerSocketFactory
    extends ServerSocketFactory
public static ServerSocketFactory getDefault()

public abstract class SSLServerSocket extends ServerSocket
```

```
public abstract ServerSocket createServerSocket(int port)
    throws IOException
public abstract ServerSocket createServerSocket(int port,
    int queueLength) throws IOException
public abstract ServerSocket createServerSocket(int port,
    int queueLength, InetAddress interface) throws IOException
```

  - SSLServerSocketFactory.getDefault() generally only supports server authentication. It doesn't support encryption. More initialization and setup are required
    - Sun's reference implementation, a com.sun.net.ssl.SSLContext object is responsible for creating fully configured and initialized secure server sockets
    - Steps to create a secure server socket in the reference implementation
      1. Generate public keys and certificates using *keytool*
      2. Pay money to have your certificates authenticated by a trusted third party such as Comodo
      3. Create an SSLContext for the algorithm you'll use
      4. Create a TrustManagerFactory for the source of certificate material you'll be using (Oracle/Sun default)
      5. Create a KeyManagerFactory for the type of key material you'll be using
      6. Create a KeyStore object for the key and certificate database (Oracle's default is JKS)
      7. Fill the KeyStore object with keys and certificates; for instance, by loading them from the filesystem using the passphrase they're encrypted with
      8. Initialize the KeyManagerFactory with the KeyStore and its passphrase
      9. Initialize the context with the necessary key managers from the KeyManagerFactory, trust managers from the TrustManagerFactory, and a source of randomness

      (The last two can be null if you're willing to accept the defaults.)

Example 10-2. SecureOrderTaker

Accept orders and print on System.out

```
import java.io.*;
import java.net.*;
import java.security.*;
import java.security.cert.CertificateException;
import java.util.Arrays;
import javax.net.ssl.*;
public class SecureOrderTaker {
  public final static int PORT = 7000;
  public final static String algorithm = "SSL";
  public static void main(String[] args) {
    try {
     SSLContext context = SSLContext.getInstance(algorithm);
     // The reference implementation only supports X.509 keys
     KeyManagerFactory kmf = KeyManagerFactory.getInstance("SunX509");
     // Oracle's default kind of key store
     KeyStore ks = KeyStore.getInstance("JKS");
     // For security, every key store is encrypted with a
     // passphrase that must be provided before we can load
     // it from disk. The passphrase is stored as a char[] array
     // so it can be wiped from memory quickly rather than
     // waiting for a garbage collector.
     char[] password = System.console().readPassword();
     ks.load(new FileInputStream("jnp4e.keys"), password);
     kmf.init(ks, password);
     context.init(kmf.getKeyManagers(), null, null);
     // wipe the password
     Arrays.fill(password, '0');

     SSLServerSocketFactory factory
         = context.getServerSocketFactory();
     SSLServerSocket server
         = (SSLServerSocket) factory.createServerSocket(PORT);
     // add anonymous (non-authenticated) cipher suites
     String[] supported = server.getSupportedCipherSuites();
     String[] anonCipherSuitesSupported = new String[supported.length];
     int numAnonCipherSuitesSupported = 0;
     for (int i = 0; i < supported.length; i++) {
       if (supported[i].indexOf("_anon_") > 0) {
         anonCipherSuitesSupported[numAnonCipherSuitesSupported++] =
     } }                                              supported[i];
     String[] oldEnabled = server.getEnabledCipherSuites();
     String[] newEnabled = new String[oldEnabled.length
         + numAnonCipherSuitesSupported];
     System.arraycopy(oldEnabled, 0, newEnabled, 0, oldEnabled.length);
     System.arraycopy(anonCipherSuitesSupported, 0, newEnabled,
         oldEnabled.length, numAnonCipherSuitesSupported);
     server.setEnabledCipherSuites(newEnabled);
     // Now all the set up is complete and we can focus
     // on the actual communication.
     while (true) {
       // This socket will be secure,
       // but there's no indication of that in the code!
       try (Socket theConnection = server.accept()) {
         InputStream in = theConnection.getInputStream();
         int c;
         while ((c = in.read()) != -1) {
           System.out.write(c);
         }
       } catch (IOException ex) {
         ex.printStackTrace();
       }
     } catch (IOException | KeyManagementException
         | KeyStoreException | NoSuchAlgorithmException
         | CertificateException | UnrecoverableKeyException ex) {
       ex.printStackTrace();
     }
   }
```

S1. Generate public keys and certificates using *keytool*

```
$ keytool -genkey -alias ourstore -keystore jnp4e.keys
Enter keystore password:  2andnotafnord
Re-enter new password:
What is your first and last name?
  [Unknown]:  Elliotte Harold
What is the name of your organizational unit?
  [Unknown]:  Me, Myself, and I
What is the name of your organization?
  [Unknown]:  Cafe au Lait
What is the name of your City or Locality?
  [Unknown]:  Brooklyn
What is the name of your State or Province?
  [Unknown]:  New York
What is the two-letter country code for this unit?
  [Unknown]:  NY
Is <CN=Elliotte Harold, OU="Me, Myself, and I",
O=Cafe au Lait, L=Brooklyn,ST=New York, C=NY> correct?
  [no]:  y

Enter key password for <ourstore>
        (RETURN if same as keystore password):
```

S3. Create an SSLContext for the algorithm you'll use
S5. Create a KeyManagerFactory for the type of key material
S6. Create a KeyStore object for the database (default is JKS)
S7. Fill the KeyStore object with keys and certificates by loading them from the filesystem
S8. Initialize the KeyManagerFactory with the KeyStore and its passphrase
S9. Initialize the context with the necessary key managers

Enable *no authentication* ("_anon_") suites (see next section)

Another Approach: Use Cipher Suites That Don't Require Authentication

- Not enabled by default because vulnerable to a man-in-the-middle attack
- Allow you to write simple programs without paying money
- See next section to enable them

SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA
SSL_DH_anon_EXPORT_WITH_RC4_40_MD5
SSL_DH_anon_WITH_3DES_EDE_CBC_SHA
SSL_DH_anon_WITH_DES_CBC_SHA
SSL_DH_anon_WITH_RC4_128_MD5
TLS_DH_anon_WITH_AES_128_CBC_SHA
TLS_DH_anon_WITH_AES_128_CBC_SHA256
TLS_ECDH_anon_WITH_3DES_EDE_CBC_SHA
TLS_ECDH_anon_WITH_AES_128_CBC_SHA
TLS_ECDH_anon_WITH_NULL_SHA
TLS_ECDH_anon_WITH_RC4_128_SHA

10.8 Configuring SSLServerSockets

- Like SSLSocket, SSLServerSocket provides methods to choose cipher suites, manage sessions, and establish whether clients are required to authenticate themselves
  - Most of these methods have the similar names in SSLSocket
- Choosing the Cipher Suites

```
public abstract String[] getSupportedCipherSuites()
public abstract String[] getEnabledCipherSuites()
public abstract void     setEnabledCipherSuites(String[] suites)
```

  - newEnabled[]
    = anonCipherSuitesSupported[]
    + oldEnabled[]

```
// add anonymous (non-authenticated) cipher suites
String[] supported = server.getSupportedCipherSuites();
String[] anonCipherSuitesSupported = new String[supported.length];
int numAnonCipherSuitesSupported = 0;
for (int i = 0; i < supported.length; i++) {
  if (supported[i].indexOf("_anon_") > 0) {
    anonCipherSuitesSupported[numAnonCipherSuitesSupported++] =
                                                  supported[i];
} }
String[] oldEnabled = server.getEnabledCipherSuites();
String[] newEnabled = new String[oldEnabled.length
    + numAnonCipherSuitesSupported];
System.arraycopy(oldEnabled, 0, newEnabled, 0, oldEnabled.length);
System.arraycopy(anonCipherSuitesSupported, 0, newEnabled,
    oldEnabled.length, numAnonCipherSuitesSupported);
server.setEnabledCipherSuites(newEnabled);
```

- Session Management
  - Session creation is enabled by default

```
public abstract void setEnableSessionCreation(boolean allowSessions)
public abstract boolean getEnableSessionCreation()
```

- Client Mode
  - setNeedClientAuth() method (default is false)

```
public abstract void setNeedClientAuth(boolean flag)
public abstract boolean getNeedClientAuth()
```

    - True: only connections in which the client is able to authenticate itself will be accepted
  - setUseClientMode() method (default is false for SSLServerSocket)

```
public abstract void setUseClientMode(boolean flag)
public abstract boolean getUseClientMode()
```

    - True if the SSLSeverSocket should be treated as a client in the communication wrt authentication and other negotiations

Summary

10.1 Secure Communications
- javax.net.ssl, javax.net, java.security.cert, com.sun.net.ssl

10.2 Creating Secure Client Sockets
- javax.net.ssl.SSLSocket
- HTTPSClient (Example 10-1)

10.3 Choosing the Cipher Suites
- getSupportedCipherSuites(), getEnabledCipherSuites(), setEnabledCipherSuites()

10.4 Event Handlers
- HandshakeCompletedListener

10.5 Session Management
- SSLSession
- getSession(), setEnableSessionCreation(), startHandshake()

10.6 Client Mode
- setUseClientMode(), getUseClientMode(), setNeedClientAuth(), getNeedClientAuth()

10.7 Creating Secure Server Sockets
- javax.net.ssl.SSLServerSocket
- SecureOrderTaker (Example 10-2)

10.8 Configuring SSL Server Sockets
- Similar to SSL client sockets

Ch.11 Nonblocking I/O

Unit-9

Outline

11.1 An Example Client

11.2 An Example Server

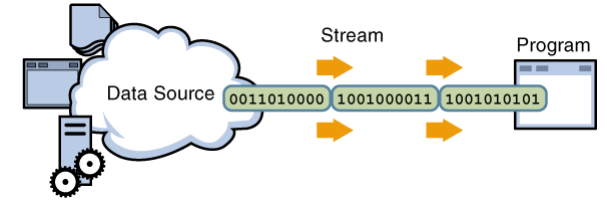
11.3 Buffers

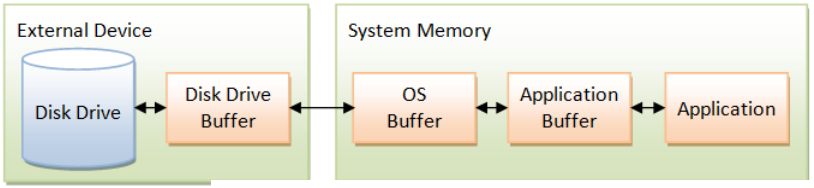
11.4 Channels

11.5 Readiness Selection

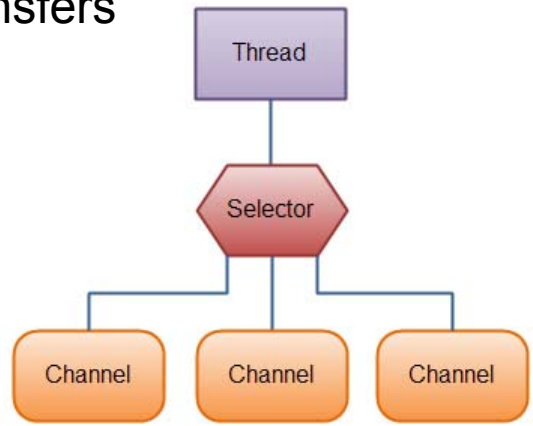
Java I/O

- Two typical I/O models
  - Stream-oriented I/O
    - Movement of single bytes, one at a time
    - Byte streams and character streams
    - Simple
  - Block-oriented I/O
    - Dealing with data in blocks, especially for bulk data transfers
      - A low-level data transfer mechanism
    - Channels and buffers
    - Faster



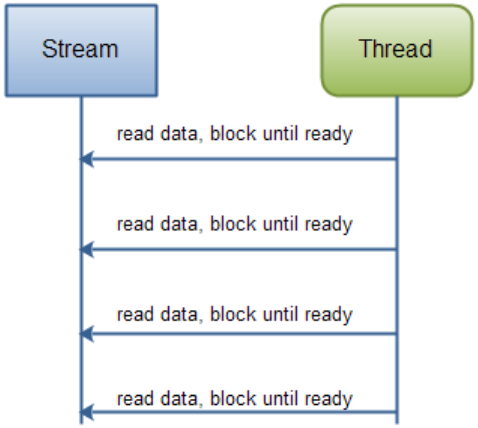


- Java I/O
  - Original I/O package: java.io.*
  - New I/O package: NIO (JDK 1.4+)
    - Channels and Buffers
      - Data is always read from a channel into a buffer, or written from a buffer to a channel
    - **Non-blocking I/O**
      - After asking a channel to read data into a buffer, a thread can do something else while the channel reads data into the buffer
    - Selector: an object that can monitor multiple channels for events
      - A thread can monitor multiple channels for data

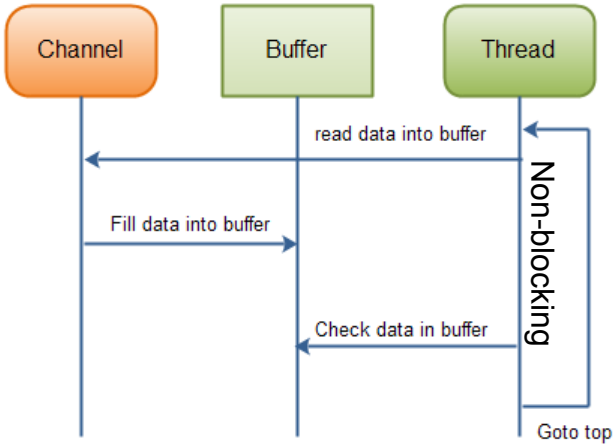


Java IO vs. NIO

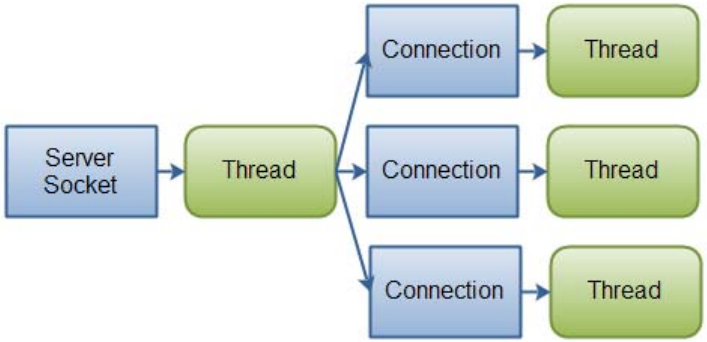
Java IO: Reading data from a blocking stream



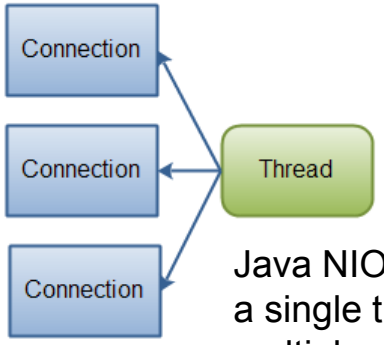
Java NIO: Reading data from a **channel** until all needed data is in **buffer**



Java IO: A classic IO server design - one connection handled by one thread



Java NIO: A single thread managing multiple connections



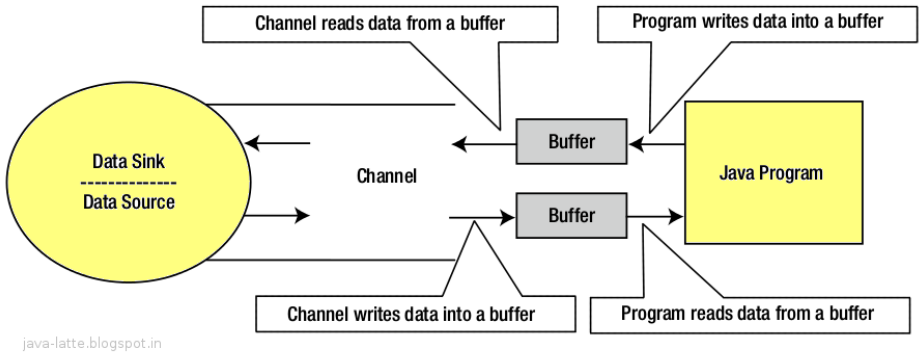
Java NIO's **selectors** allow a single thread to monitor multiple channels of input

http://tutorials.jenkov.com/java-nio/nio-vs-io.html

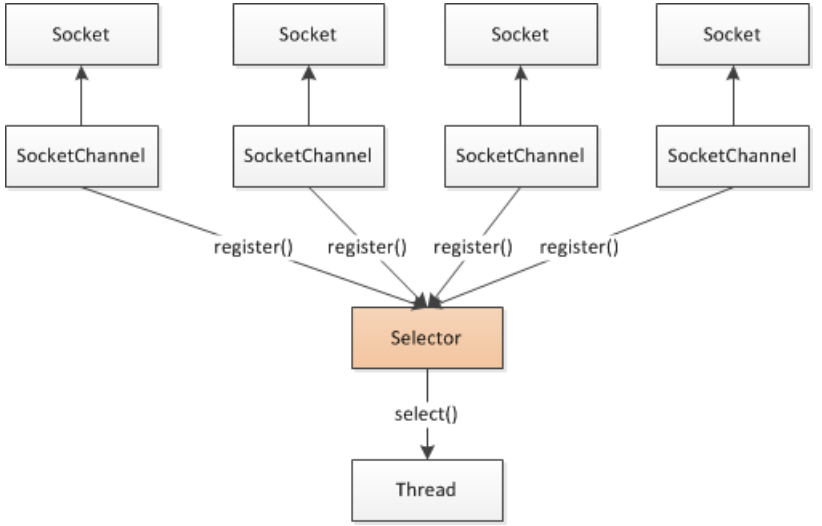
java.nio Package

http://docs.oracle.com/javase/8/docs/api/java/nio/package-summary.html

- Central abstractions of the NIO APIs
  - **Buffers**: containers for for a fixed amount of data of a specific primitive type
    - ByteBuffer (MappedByteBuffer), CharBuffer
    - ShortBuffer, IntBuffer, LongBuffer
    - FloatBuffer, DoubleBuffer
  - **Channels**: represent connections to entities capable of performing I/O operations
    - FileChannel, DatagramChannel, SocketChannel, ServerSocketChannel
  - **Selectors** and selection keys, which together with selectable channels: define a multiplexed, non-blocking I/O facility
  - **Charsets** and their associated decoders and encoders: translate between bytes and Unicode characters



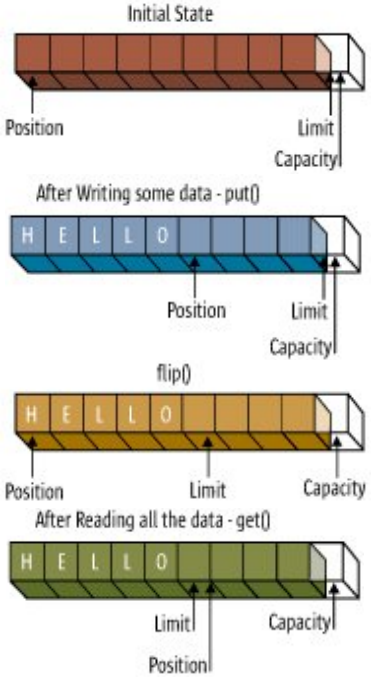
Interaction between a channel, buffers, a Java program, a data source, and a data sink



Buffers

- Essential properties of a buffer
  - **Capacity**: the number of elements it contains
    - Specified when the Buffer is constructed and cannot be changed (similar to an array)
    - Never changes
  - **Limit**: specifies the current occupancy (valid data in the range of 0 to limit-1)
    - Never greater than its capacity
  - **Position**: the index of the next element to be read or written
    - Never greater than its limit
- Example

http://www2.sys-con.com/itsg/virtualcd/java/archives/0902/krishnan/




```
ByteBuffer buffer = ByteBuffer.allocate(512);
```

```
String str = "Hello";
byte[] data = str.getBytes();
buffer.put(data);
```

```
buffer.flip();
```

`flip()` : set `Position` to 0 and `Limit` to last `Position`
`clear()`: set `Position` to 0 and `Limit` to `Capacity`
`rewind()`:set `Position` to 0 only

```
int limit = buffer.limit();
byte[] data = new byte[limit];
buffer.get(data);
System.out.println(new String(data));
```

Byte Buffers

- Byte buffer: the most commonly used kind of buffer
  - Can allocate as a direct buffer: the JVM makes a best effort to perform native I/O operations directly upon it
    - Avoid copying the buffer's content to (or from) an intermediate buffer
    - Implementation dependent: e.g. DMA to devices or map directly onto underlying OS buffers
  - Can create by mapping a region of a file directly into memory: a few additional file-related operations defined in the MappedByteBuffer class are available
    - Mapped I/O managed by the OS instead of the Java application
    - Can be used to wrap a region of OS buffer
  - Provides access to its content as a sequence of binary data of any non-boolean primitive type, in either big-endian or little-endian byte order

Channels

- Bi-directional object opened for reading, writing, or both
  - Simply create a buffer and then ask a channel to read data into it
    - For example, three steps to read from a file
      1. Getting the Channel from FileInputStream
      2. Creating the Buffer
      3. Reading from the Channel into the Buffer
  - Create a buffer, fill it with data, and then ask a channel to write from it
- Typical flow
  - **Read** from input **channel** into `Buffer`
  - **Flip** the `Buffer`
  - **Write** `Buffer` into output **channel** or get data from `Buffer` for further processing
  - **Clear** the `Buffer` for the next read

```
DumpFile(FileInputStream fin) {
  try (FileChannel in = fin.getChannel()) {
    ByteBuffer buf = ByteBuffer.allocate(bufferSize);
    while (in.read(buf) > 0) { // Read data from file into ByteBuffer
      buf.flip();     // set position to 0
      while(buf.hasRemaining()) {                // probably multiple lines
        System.out.print((char) buf.get());      // Print one line to screen
      }
      buf.clear();     // For the next read
} } }
```

Performance Comparison of File Copies

```
CopyWithIndirectByteBuffer(FileInputStream fin, FileOutputStream fout) {
  try (FileChannel in = fin.getChannel();
       FileChannel out = fout.getChannel()) {
    ByteBuffer bytebuf = ByteBuffer.allocate(bufferSize);
    while (in.read(bytebuf) > 0) { // Read data from file into ByteBuffer
      bytebuf.flip();     // set position to 0
      out.write(bytebuf);  // Write data from ByteBuffer to file
      bytebuf.clear();     // For the next read
} } }
CopyWithDirectByteBuffer(FileInputStream fin, FileOutputStream fout) {
  try (FileChannel in = fin.getChannel();
       FileChannel out = fout.getChannel()) {
    ByteBuffer bytebuf = ByteBuffer.allocateDirect(bufferSize);
    while (in.read(bytebuf) > 0) { // Read data from file into ByteBuffer
      bytebuf.flip();     // set position to 0
      out.write(bytebuf);  // Write data from ByteBuffer to file
      bytebuf.clear();     // For the next read
} } }
CopyWithDirectByteBuffer(FileInputStream fin, FileOutputStream fout) {
  try (FileChannel in = fin.getChannel();
       FileChannel out = fout.getChannel()) {
    in.transferTo(0, in.size(), out);
} }
```

File-Copy Method BufSize/Time(ms)	4KB	16KB	32KB	64KB	128KB	256KB	1024KB
(a) FileChannel with an indirect ByteBuffer	16.67	6.92	3.95	3.26	2.77	2.49	3.57
(b) FileChannel with a direct ByteBuffer	9.73	3.39	2.75	2.15	2.11	*1.66*	*1.86*
(c) FileChannel with transferTo()	*3.33*	*1.86*	*1.76*	*1.88*	*2.02*	1.80	1.97
(d) Buffered Stream	124.21	110.85	109.60	109.77	109.64	109.10	109.08
(e) Programmer-managed byte-array	7.72	4.06	2.90	2.96	2.59	2.55	5.88

http://www3.ntu.edu.sg/home/ehchua/programming/java/J5b_IO_advanced.html

Non-blocking I/O

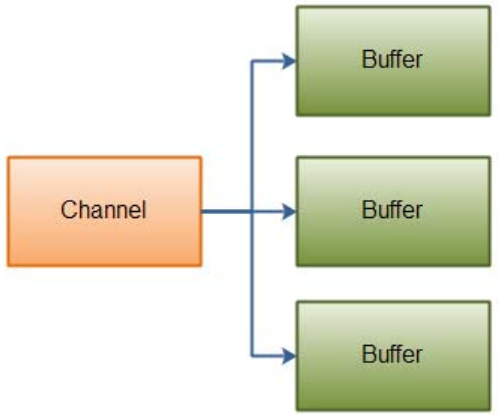
- Motivation: the overhead of spawning multiple threads and switching between them can be nontrivial
  - Compared to CPUs and memory or even disks, networks are slow
    - The blazingly fast CPU waits for the molasses-slow network
  - A server to process thousands of requests per second
    - Cost a lot if assigning a thread to each connection
    - Faster if one thread can take responsibility for multiple connections, and move on to the next ready connection as quickly as possible

```
ClientSocketChannel(String host, int port) throws Exception {
  ByteBuffer buf = ByteBuffer.allocateDirect(1024);
  SocketChannel sChannel = SocketChannel.open();
  sChannel.configureBlocking(false);
  sChannel.connect(new InetSocketAddress(host, port)); // non-blocking returns false and
           // the connection operation must later be completed by invoking finishConnect()

  while(sChannel.read(buf) >= 0) {  // non-blocking, or blocked if not connected yet
    buf.flip();
    while(buf.hasRemaining()) {                  // probably multiple lines
       System.out.print((char) buf.get());       // Print one line to screen
    }
    buf.clear();     // For the next read
  }
  sChannel.close();
}
```

Scattering Reads and Gathering Writes

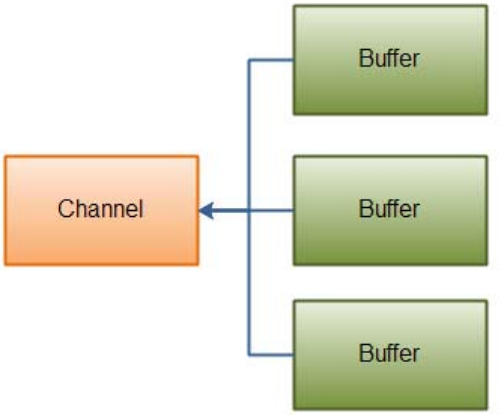
- Scattering Read: reads data from a single channel into multiple buffers

```
ByteBuffer tcpHeader = ByteBuffer.allocate(20);
ByteBuffer tcpBody   = ByteBuffer.allocate(1220);

ByteBuffer[] bufferArray = { tcpHeader, tcpBody };

channel.read(bufferArray);
```



- Gathering Write: writes data from multiple buffers into a single channel

```
ByteBuffer tcpHeader = ByteBuffer.allocate(20);
ByteBuffer tcpBody   = ByteBuffer.allocate(1220);

//write data into buffers

ByteBuffer[] bufferArray = { tcpHeader, tcpBody };

channel.write(bufferArray);
```



http://tutorials.jenkov.com/java-nio/scatter-gather.html

Selector

```
Selector selector = Selector.open();
channel.configureBlocking(false);
SelectionKey key = channel.register(selector, SelectionKey.OP_READ);

while(true) {
  int readyChannels = selector.select();
  if(readyChannels == 0) continue;

  Set<SelectionKey> selectedKeys = selector.selectedKeys();
  Iterator<SelectionKey> keyIterator = selectedKeys.iterator();

  while(keyIterator.hasNext()) {
    SelectionKey key = keyIterator.next();

    if(key.isAcceptable()) {
        // a connection was accepted by a ServerSocketChannel.
        ServerSocketChannel server = (ServerSocketChannel) key.channel();
        SocketChannel client = server.accept();
    } else if (key.isConnectable()) {
        // a connection was established with a remote server.
    } else { // it may be Readable, Writable, or Readable+Writable at once
        if (key.isReadable()) {
            // a channel is ready for reading
        }
        if (key.isWritable()) {
            // a channel is ready for writing
        }
    }
    keyIterator.remove();
} }
```

- Create a selector
- Channels register to the selector as a SelectionKey

- Check if the selector has channel events

- Turn the key set into Java Iterator

- Travel Iterator: loop to process SelectionKey having an event
  - 4 events: OP_CONNECT, OP_ACCEPT, OP_READ, OP_WRITE

11.1 An Example Client

- Character Generator Protocol (RFC 864): TCP Port 19 (Example 11-1)
  - The server sends a continuous sequence of characters until the client disconnects
    - Common pattern: rotating 72-character + “\r\n” lines of the 95 ASCII printing characters

```
!"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefgh
"#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghi
#$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghij
$%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijk
%&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijkl
&'()*+,-./0123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklm
```

    - Not commonly used these days since it’s vulnerable to DoS attacks
  - The client receives a continuous sequence of characters after connection
    - Simply outputs whatever it receives to the console

Example 11-1. A Channel-based chargen client

```
import java.nio.*;
import java.nio.channels.*;
import java.net.*;
import java.io.IOException;
public class ChargenClient {

  public static int DEFAULT_PORT = 19;

  public static void main(String[] args) {

    if (args.length == 0) {
      System.out.println("Usage: java ChargenClient host [port]");
      return;
    }

    int port;
    try {
      port = Integer.parseInt(args[1]);
    } catch (RuntimeException ex) {
      port = DEFAULT_PORT;
    }

    try {
      SocketAddress address = new InetSocketAddress(args[0], port);
      SocketChannel client = SocketChannel.open(address);
      client.configureBlocking(false);
      ByteBuffer buffer = ByteBuffer.allocate(74);
      WritableByteChannel out = Channels.newChannel(System.out);

      while (client.read(buffer) != -1) {
        buffer.flip();
        out.write(buffer);
        buffer.clear();
      }
    } catch (IOException ex) {
      ex.printStackTrace();
} } }
```

- Client: create a socket `channel`
  - Default in blocking mode
- Prepare the Buffer
- WritableByteChannel: turn console output-only stream into a `channel`
  - Only one write operation upon a writable channel may be in progress at any time
- Copy data from input `channel` to output `channel` with `buffer`
- Change to non-blocking mode

```
while (true) {
  // Put whatever code here you want to
  //    run every pass through the loop
  // whether anything is read or not
  int n = client.read(buffer);
  if (n > 0) {
     buffer.flip();
     out.write(buffer);
     buffer.clear();
  } else if (n == -1) {
    // This shouldn't happen unless
    //   the server is misbehaving.
    break;
} }
```

11.2 An Example Server

Example 11-2. A Chargen Server

```
import java.nio.*;
import java.nio.channels.*;
import java.net.*;
import java.util.*;
import java.io.IOException;
public class ChargenServer {
  public static int DEFAULT_PORT = 19;
  public static void main(String[] args) {
    int port;
    try {
      port = Integer.parseInt(args[0]);
    } catch (RuntimeException ex) {port = DEFAULT_PORT;}
    System.out.println("Listening for connections on port "+port);
    byte[] rotation = new byte[95*2];
    for (byte i = ' '; i <= '~'; i++) {
      rotation[i -' '] = i;   rotation[i + 95 - ' '] = i;
    }
    ServerSocketChannel serverChannel;
    Selector selector;
    try {
      serverChannel = ServerSocketChannel.open();
      ServerSocket ss = serverChannel.socket();
      InetSocketAddress address = new InetSocketAddress(port);
      ss.bind(address);
      serverChannel.configureBlocking(false);
      selector = Selector.open();
      serverChannel.register(selector, SelectionKey.OP_ACCEPT);
    } catch (IOException ex) {
      ex.printStackTrace();
      return;
    }
    while (true) {
      try {
        selector.select();
      } catch (IOException ex) {
        ex.printStackTrace();
        break;
      }
      Set<SelectionKey> readyKeys = selector.selectedKeys();
      Iterator<SelectionKey> iterator = readyKeys.iterator();
      while (iterator.hasNext()) {
        SelectionKey key = iterator.next();
        iterator.remove();
        try {
          if (key.isAcceptable()) {
            ServerSocketChannel server = (ServerSocketChannel) key.channel();
            SocketChannel client = server.accept();
            System.out.println("Accepted connection from " + client);
            client.configureBlocking(false);
            SelectionKey key2=client.register(selector,SelectionKey.OP_WRITE);
            ByteBuffer buffer = ByteBuffer.allocate(74);
            buffer.put(rotation, 0, 72);
            buffer.put((byte) '\r'); buffer.put((byte) '\n');
            buffer.flip();
            key2.attach(buffer);
          } else if (key.isWritable()) {
            SocketChannel client = (SocketChannel) key.channel();
            ByteBuffer buffer = (ByteBuffer) key.attachment();
            if (!buffer.hasRemaining()) {
              // Refill the buffer with the next line
              buffer.rewind();
              // Get the old first character
              int first = buffer.get();
              // Get ready to change the data in the buffer
              buffer.rewind();
              // Find the new first characters position in rotation
              int position = first - ' ' + 1;
              // copy the data from rotation into the buffer
              buffer.put(rotation, position, 72);
              // Store a line break at the end of the buffer
              buffer.put((byte) '\r'); buffer.put((byte) '\n');
              // Prepare the buffer for writing
              buffer.flip();
            }
            client.write(buffer);
          }
        } catch (IOException ex) {
          key.cancel();
          try { key.channel().close();}
          catch (IOException cex) {}
        }
      }
    }
  }
}
```

Create Selector and register Channel

Check each event/Key

New channel/key for an accepted connection

Each connection's key attached a buffer

11.3 Buffers

- Key differences between streams and channels
  - Byte-based vs. block-based
    - A Channel passes blocks of data around in buffers, a fixed-size list of elements
  - Channels support bidirectional reading and writing on the same object
    - Streams separate input and output
- Key pieces of information associated with each buffer
  - *capacity*: The maximum number of elements the buffer can hold

```
public final int capacity()
```

  - *limit*: The end of accessible data in the buffer

```
public final int    limit()
public final Buffer limit(int newLimit)
```

  - *position*: the next location in the buffer that will be read from or written to
    - Auto increased after read, write, put and get

```
public final int    position()
public final Buffer position(int newPosition)
```

  - *mark*: A client-specified index in the buffer for returning back

```
public final Buffer mark()
public final Buffer reset()
```

- Common Buffer methods to change properties
  - clear(): set *position* to 0 and *limit* to *capacity*
  - rewind(): set *position* to 0; *limit* not changed
  - flip(): set *limit* to current *position* and *position* to 0
  - remaining(): return number of elements in the buffer between *position* and *limit*
  - hasRemaining(): return true if remaining() is greater than 0

```
buffer.flip();
String result = "";
while (buffer.hasRemaining())
{ result += buffer.get();}
```

Creating Buffers

- Factory methods to create typed buffers
  - allocate(): create a new, empty buffer with specified fixed capacity

```
ByteBuffer buffer1 = ByteBuffer.allocate(100);
IntBuffer  buffer2 = IntBuffer.allocate(100);
```

    - The buffer can be accessed by the array() and arrayOffset() methods

```
byte[] data1 = buffer1.array();
int[]  data2 = buffer2.array();
```

      - Changes to the backing array are reflected in the buffer and vice versa
    - order(): inspect and set the buffer's byte order (def: BIG_ENDIAN)

```
if (buffer.order().equals(ByteOrder.BIG_ENDIAN))
{ buffer.order(ByteOrder.LITTLE_ENDIAN);}
```

  - ByteBuffer.allocateDirect(): using direct memory access to the buffer on Ethernet card, kernel memory, or something else if possible (implementation/JVM dependent)
  - wrap(): use an existing array of data instead of creating a new buffer

```
byte[] data = "Some data".getBytes("UTF-8");
ByteBuffer buffer1 = ByteBuffer.wrap(data);
char[] text = "Some text".toCharArray();
CharBuffer buffer2 = CharBuffer.wrap(text);
```

    - Changes to the array are reflected in the buffer and vice versa
  - View buffers: create new typed object from an underlying ByteBuffer beginning with the current position

```
public abstract ShortBuffer  asShortBuffer()
public abstract CharBuffer   asCharBuffer()
public abstract IntBuffer    asIntBuffer()
public abstract LongBuffer   asLongBuffer()
public abstract FloatBuffer  asFloatBuffer()
public abstract DoubleBuffer asDoubleBuffer()
```

    - Changes to the view buffer are reflected in the underlying buffer and vice versa

Buffer Access Methods

- Filling and draining: position is incremented automatically
  - put(): filling one element or array of elements
  - get(): draining

```
public abstract byte       get(int index)
public abstract ByteBuffer put(int index, byte b)
public ByteBuffer get(byte[] dst, int offset, int length)
public ByteBuffer get(byte[] dst)
public ByteBuffer put(byte[] array, int offset, int length)
public ByteBuffer put(byte[] array)
```

- Data conversion while accessing

```
public abstract char       getChar()
public abstract ByteBuffer putChar(char value)
public abstract char       getChar(int index)
public abstract ByteBuffer putChar(int index, char value)
public abstract short      getShort()
public abstract ByteBuffer putShort(short value)
public abstract short      getShort(int index)
public abstract ByteBuffer putShort(int index, short value)
public abstract int        getInt()
public abstract ByteBuffer putInt(int value)
public abstract int        getInt(int index)
public abstract ByteBuffer putInt(int index, int value)
public abstract long       getLong()
public abstract ByteBuffer putLong(long value)
public abstract long       getLong(int index)
public abstract ByteBuffer putLong(int index, long value)
public abstract float      getFloat()
public abstract ByteBuffer putFloat(float value)
public abstract float      getFloat(int index)
public abstract ByteBuffer putFloat(int index, float value)
public abstract double     getDouble()
public abstract ByteBuffer putDouble(double value)
public abstract double     getDouble(int index)
public abstract ByteBuffer putDouble(int index, double value)
```

Integer Generator Server (Intgen)

- Generating four-byte, big-endian integers indefinitely
  1. The client connects to the server
  2. The server immediately begins sending four-byte, big-endian integers, starting with 0 and incrementing by 1 each time
     - The server will eventually wrap around into the negative numbers
  3. The server runs indefinitely. The client closes the connection when it's had enough

- Implementation
  - The server stores the current int in a four-byte-long direct ByteBuffer
    - One buffer would be attached to each channel
  - The client creates ByteBuffer but reads with an integer View buffer by asIntBuffer()

Example 11-3. Intgen Server

```
import java.nio.*;
import java.nio.channels.*;
import java.net.*;
import java.util.*;
import java.io.IOException;
public class IntgenServer {

  public static int DEFAULT_PORT = 1919;

  public static void main(String[] args) {

    int port;
    try {
      port = Integer.parseInt(args[0]);
    } catch (RuntimeException ex) {
      port = DEFAULT_PORT;
    }
    System.out.println("Listening for connections on port " + p

    ServerSocketChannel serverChannel;
    Selector selector;
    try {
      serverChannel = ServerSocketChannel.open();
      ServerSocket ss = serverChannel.socket();
      InetSocketAddress address = new InetSocketAddress(port);
      ss.bind(address);
      serverChannel.configureBlocking(false);
      selector = Selector.open();
      serverChannel.register(selector, SelectionKey.OP_ACCEPT);
    } catch (IOException ex) {
      ex.printStackTrace();
      return;
    }
```

Create Server channel and register to selector

Check each event/Key

```
while (true) {
  try {
    selector.select();
  } catch (IOException ex) {
    ex.printStackTrace();
    break;
  }
  Set<SelectionKey> readyKeys = selector.selectedKeys();
  Iterator<SelectionKey> iterator = readyKeys.iterator();
  while (iterator.hasNext()) {
    SelectionKey key = iterator.next();
    iterator.remove();
    try {
      if (key.isAcceptable()) {
        ServerSocketChannel server = (ServerSocketChannel) key.channel();
        SocketChannel client = server.accept();
        System.out.println("Accepted connection from " + client);
        client.configureBlocking(false);
        SelectionKey key2=client.register(selector,SelectionKey.OP_WRITE);
        ByteBuffer output = ByteBuffer.allocate(4);
        output.putInt(0);
        output.flip();
        key2.attach(output);
      } else if (key.isWritable()) {
        SocketChannel client = (SocketChannel) key.channel();
        ByteBuffer output = (ByteBuffer) key.attachment();
        if (! output.hasRemaining()) {
          output.rewind();
          int value = output.getInt();
          output.clear();
          output.putInt(value + 1);
          output.flip();
        }
        client.write(output);
      }
    } catch (IOException ex) {
      key.cancel();
      try {key.channel().close();}
      catch (IOException cex) {}
} } }
```

New channel/key for an accepted connection

Each connection's key attached a buffer

Example 11-4. Intgen Client

```
public class IntgenClient {
  public static int DEFAULT_PORT = 1919;
  public static void main(String[] args) {

    if (args.length == 0) {
      System.out.println("Usage: java IntgenClient host [port]");
      return;
    }

    int port;
    try {
      port = Integer.parseInt(args[1]);
    } catch (RuntimeException ex) {
      port = DEFAULT_PORT;
    }

    try {
      SocketAddress address = new InetSocketAddress(args[0], port);
      SocketChannel client  = SocketChannel.open(address);
      ByteBuffer    buffer  = ByteBuffer.allocate(4);
      IntBuffer     view    = buffer.asIntBuffer();

      for (int expected = 0; ; expected++) {
        client.read(buffer);
        int actual = view.get();
        buffer.clear();
        view.rewind();

        if (actual != expected) {
          System.err.println("Expected " + expected + "; was " + actual);
          break;
        }
        System.out.println(actual);
      }
    } catch(IOException ex) {
      ex.printStackTrace();
  }}}
```

Read with buffer
View as Int

Example 8-3. A *Time* Protocol Client

```
public class Time {

  private static final String HOSTNAME = "time.nist.gov";

  public static void main(String[] args) throws IOException, ParseException {
    Date d = Time.getDateFromNetwork();
    System.out.println("It is " + d);
  }

  public static Date getDateFromNetwork() throws IOException, ParseException {
    long differenceBetweenEpochs = 2208988800L;
    Socket socket = null;
    try {
      socket = new Socket(HOSTNAME, 37);
      socket.setSoTimeout(15000);

      InputStream raw = socket.getInputStream();

      long secondsSince1900 = 0;
      for (int i = 0; i < 4; i++) {
        secondsSince1900 = (secondsSince1900 << 8) | raw.read();
      }

      long secondsSince1970
               = secondsSince1900 - differenceBetweenEpochs;
      long msSince1970 = secondsSince1970 * 1000;
      Date time = new Date(msSince1970);

      return time;
    } finally {
      try {
        if (socket != null) socket.close();
      }
      catch (IOException ex) {}
    }
  }
```

4-byte big-endian number

Duplicating and Slicing Buffers

- compact(): shift any remaining data (from *position* to *limit*) to the start
  - *position* is set to the end of the data

```
public abstract ByteBuffer   compact()
public abstract IntBuffer    compact()
public abstract ShortBuffer  compact()
public abstract FloatBuffer  compact()
public abstract CharBuffer   compact()
public abstract DoubleBuffer compact()
```

- duplicate(): make a copy of a buffer to be shared by two or more channels

```
public abstract ByteBuffer   duplicate()
public abstract IntBuffer    duplicate()
public abstract ShortBuffer  duplicate()
public abstract FloatBuffer  duplicate()
public abstract CharBuffer   duplicate()
public abstract DoubleBuffer duplicate()
```

  - Changes to the data in one buffer are reflected in the other buffer
  - Useful to transmit the same data over multiple channels, roughly in parallel
    - Same *capacity* (and *limit*), different *position*
- slice(): only copy a subsequence of the buffer from current *position* to *limit*

```
public abstract ByteBuffer   slice()
public abstract IntBuffer    slice()
public abstract ShortBuffer  slice()
public abstract FloatBuffer  slice()
public abstract CharBuffer   slice()
public abstract DoubleBuffer slice()
```

  - Useful to divide a long buffer into multiple parts such as a protocol header followed by the data

Example 11-6. Nonblocking HTTP Server

```
public class NonblockingSingleFileHTTPServer {

  private ByteBuffer contentBuffer;
  private int port = 80;

  public NonblockingSingleFileHTTPServer(
      ByteBuffer data, String encoding, String MIMEType, int port) {

    this.port = port;
    String header = "HTTP/1.0 200 OK\r\n"
        + "Server: NonblockingSingleFileHTTPServer\r\n"
        + "Content-length: " + data.limit() + "\r\n"
        + "Content-type: " + MIMEType + "\r\n\r\n";
    byte[] headerData = header.getBytes(Charset.forName("US-ASCII"));

    ByteBuffer buffer = ByteBuffer.allocate(
        data.limit() + headerData.length);
    buffer.put(headerData);
    buffer.put(data);
    buffer.flip();
    this.contentBuffer = buffer;
  }
  public static void main(String[] args) {
    if (args.length == 0) {
      System.out.println(
        "Usage: java NonblockingSingleFileHTTPServer file port encoding");
      return;
    }
    try {
      // read the single file to serve
      String contentType =
          URLConnection.getFileNameMap().getContentTypeFor(args[0]);
      Path file = FileSystems.getDefault().getPath(args[0]);
      byte[] data = Files.readAllBytes(file);
      ByteBuffer input = ByteBuffer.wrap(data);
      // set the port to listen on
      int port;
      try {
        port = Integer.parseInt(args[1]);
        if (port < 1 || port > 65535) port = 80;
      } catch (RuntimeException ex) { port = 80;}
      String encoding = "UTF-8";
      if (args.length > 2) encoding = args[2];
      NonblockingSingleFileHTTPServer server
          = new NonblockingSingleFileHTTPServer(
          input, encoding, contentType, port);
      server.run();
    } catch (IOException ex) {
      System.err.println(ex);
} } }
```

Read a file into a ByteBuffer

```
public void run() throws IOException {
  ServerSocketChannel serverChannel = ServerSocketChannel.open();
  ServerSocket  serverSocket = serverChannel.socket();
  Selector selector = Selector.open();
  InetSocketAddress localPort = new InetSocketAddress(port);
  serverSocket.bind(localPort);
  serverChannel.configureBlocking(false);
  serverChannel.register(selector, SelectionKey.OP_ACCEPT);
  while (true) {
    selector.select();
    Iterator<SelectionKey> keys = selector.selectedKeys().iterator();
    while (keys.hasNext()) {
      SelectionKey key = keys.next();
      keys.remove();
      try {
        if (key.isAcceptable()) {
          ServerSocketChannel server = (ServerSocketChannel) key.channel();
          SocketChannel channel = server.accept();
          channel.configureBlocking(false);
          channel.register(selector, SelectionKey.OP_READ);
        } else if (key.isWritable()) {
          SocketChannel channel = (SocketChannel) key.channel();
          ByteBuffer buffer = (ByteBuffer) key.attachment();
          if (buffer.hasRemaining()) {
            channel.write(buffer);
          } else {channel.close();}// we're done
        } else if (key.isReadable()) {
          // Don't bother trying to parse the HTTP header.
          // Just read something.
          SocketChannel channel = (SocketChannel) key.channel();
          ByteBuffer buffer = ByteBuffer.allocate(4096);
          channel.read(buffer);
          // switch channel to write-only mode
          key.interestOps(SelectionKey.OP_WRITE);
          key.attach(contentBuffer.duplicate());
        }
      } catch (IOException ex) {
        key.cancel();
        try {key.channel().close();}
        catch (IOException cex) {}
} } } }
```

11-4 Channels

- Move blocks of data into and out of buffers, to and from various I/O sources
  - Three channels for network programming
    - SocketChannel, ServerSocketChannel, and DatagramChannel
- SocketChannel: for TCP sockets
  - open()
    ```
    public static SocketChannel open(SocketAddress remote) throws IOException
    public static SocketChannel open() throws IOException
    ```
    - Connect with blocked open
      ```
      SocketAddress address = new InetSocketAddress("www.cafeaulait.org",  80);
      SocketChannel channel = SocketChannel.open(address);
      ```
    - Create with unconnected socket and connect later with connect()
      ```
      SocketChannel channel = SocketChannel.open();
      SocketAddress address = new InetSocketAddress("www.cafeaulait.org",  80);
      channel.connect(address);
      ```
    - Non-blocking connect
      ```
      SocketChannel channel = SocketChannel.open();
      SocketAddress address = new InetSocketAddress("www.cafeaulait.org",  80);
      channel.configureBlocking(false);
      channel.connect();
      ```
    - Call finishConnect() to check
      ```
      public abstract boolean finishConnect() throws IOException
      ```
    - isConnected() returns true if the connection is open; isConnectionPending() if still being set up
  - read() and write()
    ```
    public abstract int read(ByteBuffer dst) throws IOException
    public final long read(ByteBuffer[] dsts) throws IOException
    public final long read(ByteBuffer[] dsts, int offset, int length)
        throws IOException
    ```
    ```
    public abstract int write(ByteBuffer src) throws IOException
    public final long write(ByteBuffer[] dsts) throws IOException
    public final long write(ByteBuffer[] dsts, int offset, int length)
        throws IOException
    ```
  - close() `public void close() throws IOException`
    - isOpen() return false if channel is closed

ServerSocketChannel

- open(): create a new ServerSocketChannel object
- bind(): bind to a port

```
try {
  ServerSocketChannel server = ServerSocketChannel.open();
  SocketAddress address = new InetSocketAddress(80);
  server.bind(address);
} catch (IOException ex) {
  System.err.println("Could not bind to port 80 because " + ex.getMessage());
}
```

- accept(): listen for incoming connections

```
public abstract SocketChannel accept() throws IOException
```

  - Operate in either blocking or nonblocking mode by configureBlocking()
  - Exceptions
    - ClosedChannelException
      - You cannot reopen a ServerSocketChannel after closing it
    - AsynchronousCloseException
      - Another thread closed this ServerSocketChannel while accept() was executing
    - ClosedByInterruptException
      - Another thread interrupted this thread while a blocking ServerSocketChannel was waiting
    - NotYetBoundException
      - You called open() but did not bind the ServerSocketChannel's peer ServerSocket to an address before calling accept(). This is a runtime exception, not an IOException
    - SecurityException
      - The security manager refused to allow this application to bind to the requested port

Channels Class

- Channels: a simple utility class for wrapping channels around I/O streams, readers and writers

```
public static InputStream newInputStream(ReadableByteChannel ch)
public static OutputStream newOutputStream(WritableByteChannel ch)
public static ReadableByteChannel newChannel(InputStream in)
public static WritableByteChannel newChannel(OutputStream out)
public static Reader newReader (ReadableByteChannel channel,
    CharsetDecoder decoder, int minimumBufferCapacity)
public static Reader newReader (ReadableByteChannel ch, String encoding)
public static Writer newWriter (WritableByteChannel ch, String encoding)
```

- Example: Read HTTP request bodies using channels and parse XML using SAX (Java Simple API for XML) for performance

```
SocketChannel channel = server.accept();
processHTTPHeader(channel);
XMLReader parser = XMLReaderFactory.createXMLReader();
parser.setContentHandler(someContentHandlerObject);
InputStream in = Channels.newInputStream(channel);
parser.parse(in);
```

Asynchronous Channels (Java 7)

- AsynchronousSocketChannel and AsynchronousServerSocketChannel: Read and write to asynchronous channels return immediately
  - Even before the I/O is complete (connect, accept, read, and write)
  - The data is further process by a Future (get())or a CompletionHandler
    - Network connections run in parallel while the program does other things

get()

```
SocketAddress address = new InetSocketAddress(args[0], port);
AsynchronousSocketChannel client = AsynchronousSocketChannel.open();
Future<Void> connected = client.connect(address);

ByteBuffer buffer = ByteBuffer.allocate(74);

// wait for the connection to finish
connected.get();

// read from the connection
Future<Integer> future = client.read(buffer);

// do other things...

// wait for the read to finish...
future.get();

// flip and drain the buffer
buffer.flip();
WritableByteChannel out = Channels.newChannel(System.out);
out.write(buffer);
```

Pass a handler while reading

```
class LineHandler implements CompletionHandler<Integer, ByteBuffer> {

  @Override
  public void completed(Integer result, ByteBuffer buffer) {
    buffer.flip();
    WritableByteChannel out = Channels.newChannel(System.out);
    try {
      out.write(buffer);
    } catch (IOException ex) {
      System.err.println(ex);
    }
  }

  @Override
  public void failed(Throwable ex, ByteBuffer attachment) {
    System.err.println(ex.getMessage());
  }
}

ByteBuffer buffer = ByteBuffer.allocate(74);
CompletionHandler<Integer, ByteBuffer> handler = new LineHandler();
channel.read(buffer, buffer, handler);
```

Socket Options (Java 7)

- Beginning in Java 7, the new NetworkChannel interface is implemented to support various TCP options
  - SocketChannel, ServerSocketChannel, AsynchronousServer SocketChannel, AsynchronousSocketChannel, and DatagramChannel

```
NetworkChannel channel = SocketChannel.open();
channel.setOption(StandardSocketOptions.SO_LINGER, 240);
```

  - Options (see Chapter 8 and Chapter 9)
    - SocketOption<NetworkInterface> StandardSocketOptions.IP_MULTICAST_IF
    - SocketOption<Boolean> StandardSocketOptions.IP_MULTICAST_LOOP
    - SocketOption<Integer> StandardSocketOptions.IP_MULTICAST_TTL
    - SocketOption<Integer> StandardSocketOptions.IP_TOS
    - SocketOption<Boolean> StandardSocketOptions.SO_BROADCAST
    - SocketOption<Boolean> StandardSocketOptions.SO_KEEPALIVE
    - SocketOption<Integer> StandardSocketOptions.SO_LINGER
    - SocketOption<Integer> StandardSocketOptions.SO_RCVBUF
    - SocketOption<Boolean> StandardSocketOptions.SO_REUSEADDR
    - SocketOption<Integer> StandardSocketOptions.SO_SNDBUF
    - SocketOption<Boolean> StandardSocketOptions.TCP_NODELAY

Example 11-7. Listing Supported Options

- List all supported socket options for different types of network channels

```
import java.io.*;
import java.net.*;
import java.nio.channels.*;

public class OptionSupport {

  public static void main(String[] args) throws IOException {
    printOptions(SocketChannel.open());
    printOptions(ServerSocketChannel.open());
    printOptions(AsynchronousSocketChannel.open());
    printOptions(AsynchronousServerSocketChannel.open());
    printOptions(DatagramChannel.open());
  }

  private static void printOptions(NetworkChannel channel)
                                          throws IOException {
    System.out.println(channel.getClass().getSimpleName()
                                            + " supports:");
    for (SocketOption<?>option:channel.supportedOptions()){
      System.out.println(option.name() + ": "
                          + channel.getOption(option));
    }
    System.out.println();
    channel.close();
}}
```

```
SocketChannelImpl supports:
SO_OOBINLINE: false
SO_REUSEADDR: false
SO_LINGER: -1
SO_KEEPALIVE: false
IP_TOS: 0
SO_SNDBUF: 131072
SO_RCVBUF: 131072
TCP_NODELAY: false

ServerSocketChannelImpl supports:
SO_REUSEADDR: true
SO_RCVBUF: 131072

UnixAsynchronousSocketChannelImpl supports:
SO_KEEPALIVE: false
SO_REUSEADDR: false
SO_SNDBUF: 131072
TCP_NODELAY: false
SO_RCVBUF: 131072

UnixAsynchronousServerSocketChannelImpl supports:
SO_REUSEADDR: true
SO_RCVBUF: 131072

DatagramChannelImpl supports:
IP_MULTICAST_TTL: 1
SO_BROADCAST: false
SO_REUSEADDR: false
IP_MULTICAST_IF: null
IP_TOS: 0
IP_MULTICAST_LOOP: true
SO_SNDBUF: 9216
SO_RCVBUF: 196724
```

11.5 Readiness Selection

- Selector.open(): create a new selector

```
public static Selector open() throws IOException
```

- The channel register() with a selector with operations

```
public final SelectionKey register(Selector sel, int ops)
     throws ClosedChannelException
public final SelectionKey register(Selector sel, int ops, Object att)
     throws ClosedChannelException
```

  - OP_ACCEPT, OP_CONNECT, OP_READ, OP_WRITE

```
channel.register(selector,  SelectionKey.OP_READ | SelectionKey.OP_WRITE);
```

  - A channel can be registered with multiple selectors
- Query the selector to find out which channels are ready to be processed
  - Find how many are ready
    - Nonblocking selectNow()

```
public abstract int selectNow() throws IOException
```

    - Blocking select() methods

```
public abstract int select() throws IOException
public abstract int select(long timeout) throws IOException
```

  - Retrieve the ready channels using selectedKeys()

```
public abstract Set<SelectionKey> selectedKeys()
```

- close() the selector if no longer needed

```
public abstract void close() throws IOException
```

The SelectionKey Class

- A SelectionKey object serves as a pointer to channels
  - Returned by register() when registering a channel with a selector
- selectedKeys() returns the same objects again inside a Set
  - Each key have four possibilities

```
public final boolean isAcceptable()
public final boolean isConnectable()
public final boolean isReadable()
public final boolean isWritable()
```

  - channel(): retrieve the channel with the key

```
public abstract SelectableChannel channel()
```

  - attachment(): retrieve the object stored in the SelectionKey

```
public final Object attachment()
```

  - cancel(): deregister the object out from the selector

```
public abstract void cancel()
```

  - Closing a channel automatically deregisters all keys for that channel in all selectors
  - Closing a selector invalidates all keys in the selectors

Summary

Java New I/O (NIO): buffers, channels, selectors and non-blocking I/O

11.1 An Example Client
- ChargenClient (Example 11-1)

11.2 An Example Server
- ChargenServer (Example 11-2)

11.3 Buffers
- ByteBuffer, Direct ByteBuffer, View
- IntgenServer (Example 11-3), IntgenClient (Example 11-4)
- *EchoServer (Example 11-5),* NonblockingSingleFileHTTPServer (Example 11-6)

11.4 Channels
- SocketChannel, ServerSocketChannel, and DatagramChannel
- OptionSupport (Example 11-7)

11.5 Readiness Selection
- Selector, SelectionKey

Ch.12 UDP

Unit-10

Outline

12.1 The UDP Protocol

12.2 UDP Clients

12.3 UDP Servers

12.4 The DatagramPacket Class

12.5 The DatagramSocket Class

12.6 Socket Options

12.7 Some Useful Applications

12.8 DatagramChannel

12.1 The UDP Protocol

- UDP: alternative transport layer protocol over IP
  - Connectionless, not reliable
  - DNS, NFS, TFTP
- Java's implementation of UDP: split into two classes
  - DatagramPacket: stuffs bytes of data into UDP packets called datagrams
  - DatagramSocket: sends/receives datagrams
    - To send: put the data in a datagramPacket and send it using a DatagramSocket
    - To receive: take a DatagramPacket object from a DatagramSocket
    - Have no notion of a unique connection between two hosts
    - Always work with individual datagram packets

12.2 UDP Clients

- Common steps to write UDP clients
  1. Open a socket on port 0

```
DatagramSocket socket = new DatagramSocket(0);
```

  2. (Optional, highly recommended) setSoTimeout(): set a timeout on the connection

```
socket.setSoTimeout(10000);
```

  3. Set up the packets: two packets, one to send (request) and one to receive
     - Specify the destination host and port in the sent/request packet

```
InetAddress host = InetAddress.getByName("time.nist.gov");
DatagramPacket request = new DatagramPacket(new byte[1], 1, host, 13);
byte[] data = new byte[1024];
DatagramPacket response = new DatagramPacket(data, data.length);
```

  4. Send the request packet and then receive the response

```
socket.send(request);
socket.receive(response);
```

  5. Extract the bytes from the response

```
String daytime = new String(response.getData(), 0, response.getLength(),
    "US-ASCII");
System.out.println(daytime);
```

  6. Close: Java 7+ Autocloseable or Java 6-

```
try (DatagramSocket socket = new DatagramSocket(0)) {
  // connect to the server...
} catch (IOException ex) {
  System.err.println("Could not connect to time.nist.gov");
}
```

```
DatagramSocket socket = null;
try {
  socket = new DatagramSocket(0);
  // connect to the server...
} catch (IOException ex) {
  System.err.println(ex);
} finally {
  if (socket != null) {
    try {
      socket.close();
    } catch (IOException ex) {
      // ignore
} } }
```

Example 12-1. A Daytime Protocol Client

```
import java.io.*;
import java.net.*;

public class DaytimeUDPClient {

  private final static int PORT = 13;
  private static final String HOSTNAME = "time.nist.gov";

  public static void main(String[] args) {
    try (DatagramSocket socket = new DatagramSocket(0)) {
      socket.setSoTimeout(10000);
      InetAddress host = InetAddress.getByName(HOSTNAME);
      DatagramPacket request = new DatagramPacket(new byte[1], 1, host , PORT);
      DatagramPacket response = new DatagramPacket(new byte[1024], 1024);
      socket.send(request);
      socket.receive(response);
      String result = new String(response.getData(), 0, response.getLength(),
                                 "US-ASCII");
      System.out.println(result);
    } catch (IOException ex) {
      ex.printStackTrace();
    }
  }
}
```

```
$ java DaytimeUDPClient
56375 13-04-11 19:55:22 50 0 0 843.6 UTC(NIST) *
```

- Note: the request or response datagram may be lost

12.3 UDP Servers

- Steps to create a UDP server
  1. Open a datagram socket on a well-known port
     - No separate datagram server socket

```
DatagramSocket socket = new DatagramSocket(13);
```

  2. Create a packet into which to receive a request

```
DatagramPacket request = new DatagramPacket(new byte[1024], 0, 1024);
socket.receive(request);
```

  3. Create a response packet

```
String daytime = new Date().toString() + "\r\n";
byte[] data = daytime.getBytes("US-ASCII");
InetAddress host = request.getAddress();
int port = request.getPort();
DatagramPacket response = new DatagramPacket(data, data.length, host, port);
```

  4. Send the response packet back over the same socket that received it

```
socket.send(response);
```

- UDP servers tend not to be as multithreaded as TCP servers
  – They usually don't do a lot of work for any one client

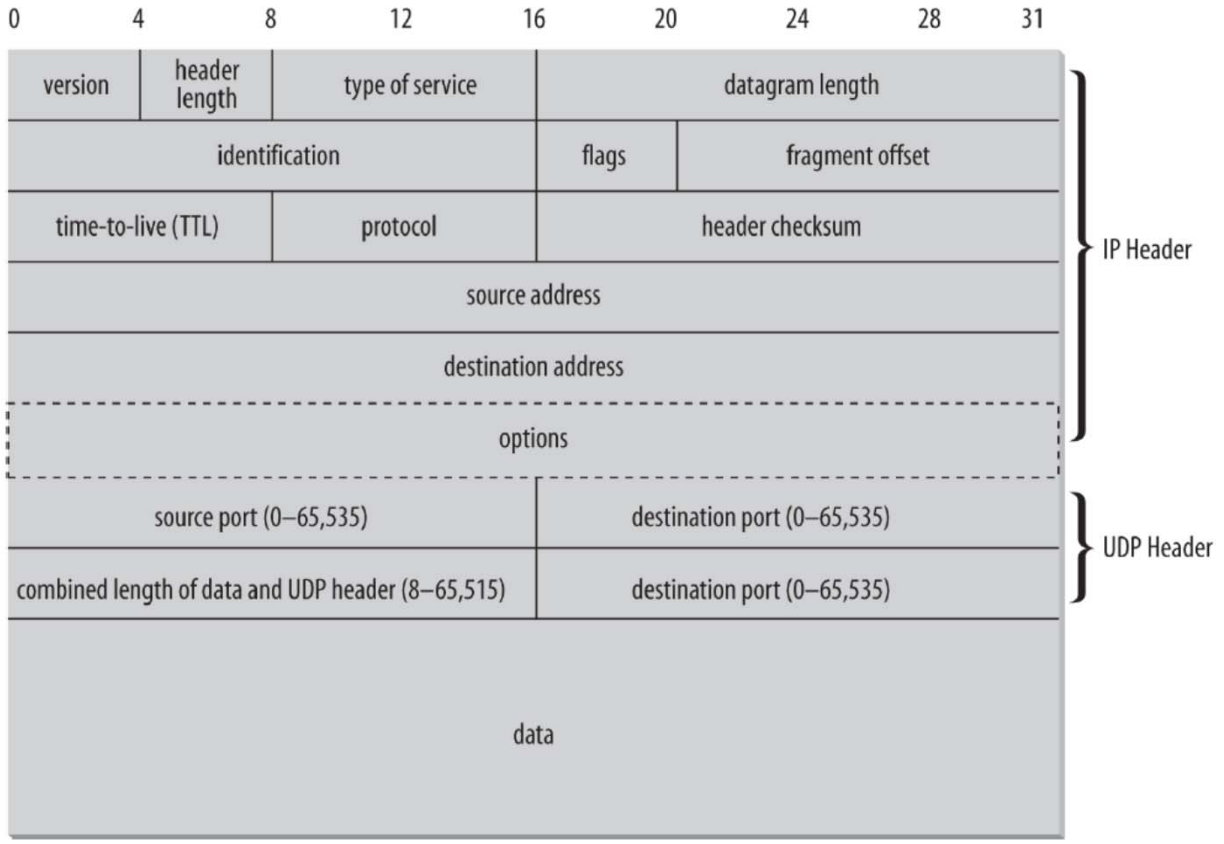
Example 12-2. A Daytime Protocol Server

```
import java.net.*;
import java.util.Date;
import java.util.logging.*;
import java.io.*;
public class DaytimeUDPServer {
  private final static int PORT = 13;
  private final static Logger audit = Logger.getLogger("requests");
  private final static Logger errors = Logger.getLogger("errors");
  public static void main(String[] args) {
    try (DatagramSocket socket = new DatagramSocket(PORT)) {
      while (true) {
        try {
          DatagramPacket request = new DatagramPacket(new byte[1024], 1024);
          socket.receive(request);

          String daytime = new Date().toString();
          byte[] data = daytime.getBytes("US-ASCII");
          DatagramPacket response = new DatagramPacket(data, data.length,
              request.getAddress(), request.getPort());
          socket.send(response);
          audit.info(daytime + " " + request.getAddress());
        } catch (IOException | RuntimeException ex) {
          errors.log(Level.SEVERE, ex.getMessage(), ex);
      } }
    } catch (IOException ex) {
      errors.log(Level.SEVERE, ex.getMessage(), ex);
} } }
```

- Note: the server doesn't care the input data

12.4 The DatagramPacket Class

- Theoretical max UDP size is 65,507 bytes (65,535 – 28)
  - IP header = 20 bytes, UDP header = 8 bytes
  - Usually many platforms set the default limit to 8,192 bytes
- In Java, a UDP datagram is represented by an instance of the DatagramPacket class



DatagramPacket Constructors

http://docs.oracle.com/javase/8/docs/api/java/net/DatagramPacket.html

- 6 Constructors
  - Two constructors for receiving data

```
public DatagramPacket(byte[] buffer, int length)
public DatagramPacket(byte[] buffer, int offset, int length)
```

    - Example: receiving a datagram of up to 8,192 bytes

```
byte[] buffer = new byte[8192];
DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
```

  - Four constructors for sending data: assigning destination host and port

```
public DatagramPacket(byte[] data, int length,
    InetAddress destination, int port)
public DatagramPacket(byte[] data, int offset, int length,
    InetAddress destination, int port)
public DatagramPacket(byte[] data, int length,
    SocketAddress destination)
public DatagramPacket(byte[] data, int offset, int length,
    SocketAddress destination)
```

    - Example: using InetAddress object

```
String s = "This is a test";
byte[] data = s.getBytes("UTF-8");

try {
  InetAddress ia = InetAddress.getByName("www.ibiblio.org");
  int port = 7;
  DatagramPacket dp = new DatagramPacket(data, data.length, ia, port);
  // send the packet...
} catch (IOException ex)
}
```

The get Methods

- Six methods to retrieve different parts of a datagram

InetAddress	getAddress() Returns the IP address of the machine to which this datagram is being sent or from which the datagram was received
int	getPort() Returns the port number on the remote host to which this datagram is being sent or from which the datagram was received
SocketAddress	getSocketAddress() Gets the SocketAddress (usually IP address + port number) of the remote host that this packet is being sent to or is coming from

byte[]	getData()
	Returns a byte array containing the data from the datagram
	`String s = new String(dp.getData(), "UTF-8");`
	The datagram can be converted to ByteArrayInputStream
	• Must specify the offset and the length to specify the available data
	`InputStream in = new ByteArrayInputStream(packet.getData(), packet.getOffset(), packet.getLength());`
	• Can be chained to DataInputStream to be read using readInt(), readLong(), …
	`DataInputStream din = new DataInputStream(in);`

int	getLength() Returns the length of the data to be sent or the length of the data received
int	getOffset() Returns the offset of the data to be sent or the offset of the data received

Example 12-3. Construct a DatagramPacket to Receive Data

```
import java.io.*;
import java.net.*;

public class DatagramExample {

  public static void main(String[] args) {

    String s = "This is a test.";

    try {
      byte[] data = s.getBytes("UTF-8");
      InetAddress ia = InetAddress.getByName("www.ibiblio.org");
      int port = 7;
      DatagramPacket dp
          = new DatagramPacket(data, data.length, ia, port);
      System.out.println("This packet is addressed to "
          + dp.getAddress() + " on port " + dp.getPort());
      System.out.println("There are " + dp.getLength()
          + " bytes of data in the packet");
      System.out.println(
          new String(dp.getData(), dp.getOffset(), dp.getLength(), "UTF-8"));
    } catch (UnknownHostException | UnsupportedEncodingException ex) {
      System.err.println(ex);
    }
  }
}
```

Create an artificial datagram not really sent

```
% java DatagramExample
This packet is addressed to www.ibiblio.org/152.2.254.81 on port 7
There are 15 bytes of data in the packet
This is a test.
```

The setter Methods

void setData(byte[] buf)
Set the data buffer for this packet; changes the payload of the datagram

void setData(byte[] buf, int offset, int length)
Set the data buffer for this packet;
sends different parts of a buffer

```
int offset = 0;
DatagramPacket dp = new DatagramPacket(bigarray, offset, 512);
int bytesSent = 0;
while (bytesSent < bigarray.length) {
  socket.send(dp);
  bytesSent += dp.getLength();
  int bytesToSend = bigarray.length - bytesSent;
  int size = (bytesToSend > 512) ? 512 : bytesToSend;
  dp.setData(bigarray, bytesSent, size);
}
```

```
String s = "Really Important Message";
byte[] data = s.getBytes("UTF-8");
DatagramPacket dp = new DatagramPacket(data, data.length);
dp.setPort(2000);
int network = "128.238.5.";
for (int host = 1; host < 255; host++) {
  try {
    InetAddress remote = InetAddress.getByName(network + host);
    dp.setAddress(remote);
    socket.send(dp);
  } catch (IOException ex) {
    // skip it; continue with the next host
  }
}
```

void setAddress(InetAddress iaddr)
Sets the IP address of the machine to which this datagram is being sent
Allow to send the same datagram to many different recipients

void setPort(int iport)
Sets the port number on the remote host to which this datagram is being sent

void setSocketAddress(SocketAddress address)
Sets the SocketAddress (usually IP address + port number) of the remote host to which this datagram is being sent.
Use this when replying

```
DatagramPacket input = new DatagramPacket(new byte[8192], 8192);
socket.receive(input);
DatagramPacket output = new DatagramPacket(
    "Hello there".getBytes("UTF-8"), 11);
SocketAddress address = input.getSocketAddress();
output.setAddress(address);
socket.send(output);
```

void setLength(int length)
Set the length for this packet

12.5 The DatagramSocket Class

- DatagramSocket: all datagram sockets bind to a local port, on
  - which they listen for incoming data and
  - which they place in the header of outgoing datagrams
- The constructors
  - DatagramSocket(): Constructs a datagram socket and binds it to any available port on the local host machine

```
try {
  DatagramSocket client = new DatagramSocket();
  // send packets...
} catch (SocketException ex) {
  System.err.println(ex);
}
```

  - DatagramSocket(DatagramSocketImpl impl): Creates an unbound datagram socket with the specified DatagramSocketImpl
  - DatagramSocket(int port): Constructs a datagram socket and binds it to the specified port on the local host machine
  - DatagramSocket(int port, InetAddress laddr): Creates a datagram socket, bound to the specified *local* address
  - DatagramSocket(SocketAddress bindaddr): Creates a datagram socket, bound to the specified *local* socket address

```
SocketAddress address = new InetSocketAddress("127.0.0.1", 9999);
DatagramSocket socket = new DatagramSocket(address);
```

Example 12-4. Look for Local UDP Ports

Local port scanner that looks at ports from 1024 and up

```
import java.net.*;

public class UDPPortScanner {

  public static void main(String[] args) {
    for (int port = 1024; port <= 65535; port++) {
      try {
        // the next line will fail and drop into the catch block if
        // there is already a server running on port i
        DatagramSocket server = new DatagramSocket(port);
        server.close();
      } catch (SocketException ex) {
        System.out.println("There is a server on port " + port + ".");
      }
    }
  }
}
```

```
% java UDPPortScanner
There is a server on port 2049.
There is a server on port 32768.
There is a server on port 32770.
There is a server on port 32771.
```

Sending, Receiving and Managing Methods

void	send(DatagramPacket p) Sends a datagram packet from this socket
void	receive(DatagramPacket p) Receives a datagram packet from this socket
void	close() Closes this datagram socket
void	bind(SocketAddress addr) Limits to this DatagramSocket to a specific address and port Usually does not bind. If binded, only packets from/to the specified addr are allowed
void	connect(InetAddress address, int port) Limits to connect the socket to a remote address for this socket
void	connect(SocketAddress addr) Limits to connect this socket to a remote socket address (IP address + port number)
void	disconnect() Disconnects the socket.

Example 12-5. UDPDiscardClient

```
import java.net.*;
import java.io.*;

public class UDPDiscardClient {

  public final static int PORT = 9;

  public static void main(String[] args) {

    String hostname = args.length > 0 ?  args[0] : "localhost";

    try (DatagramSocket theSocket = new DatagramSocket()) {
      InetAddress server = InetAddress.getByName(hostname);
      BufferedReader userInput
          = new BufferedReader(new InputStreamReader(System.in));
      while (true) {
        String theLine = userInput.readLine();
        if (theLine.equals(".")) break;
        byte[] data = theLine.getBytes();
        DatagramPacket theOutput
            = new DatagramPacket(data, data.length, server, PORT);
        theSocket.send(theOutput);
      } // end while
    } catch (IOException ex) {
      System.err.println(ex);
} } }
```

Read lines from console and send to server

Example 12-6. UDPDiscardServer

```
import java.net.*;
import java.io.*;
public class UDPDiscardServer {
  public final static int PORT = 9;
  public final static int MAX_PACKET_SIZE = 65507;
  public static void main(String[] args) {

    byte[] buffer = new byte[MAX_PACKET_SIZE];
    try (DatagramSocket server = new DatagramSocket(PORT)) {
      DatagramPacket packet =
                  new DatagramPacket(buffer, buffer.length);
      while (true) {
        try {
          server.receive(packet);
          String s = new String(packet.getData(), 0,
                                packet.getLength(), "8859_1");
          System.out.println(packet.getAddress() + " at port "
              + packet.getPort() + " says " + s);
          // reset the length for the next packet
          packet.setLength(buffer.length);
        } catch (IOException ex) {
          System.err.println(ex);
        }}// end while
    } catch (SocketException  ex) {
      System.err.println(ex);
} } }
```

Read strings from client and output to console

The get Methods

int	getLocalPort() Returns the port number on the local host to which this socket is bound
InetAddress	getLocalAddress() Gets the local address to which the socket is bound
SocketAddress	getLocalSocketAddress() Returns the address of the endpoint this socket is bound to
DatagramChannel	getChannel() Returns the unique DatagramChannel object associated with this datagram socket, if any
int	getPort() Returns the port number to which this socket is connected
SocketAddress	getRemoteSocketAddress() Returns the address of the endpoint this socket is connected to, or null if it is unconnected
InetAddress	getInetAddress() Returns the address to which this socket is connected

12.6 Socket Options

- SO_TIMEOUT: receive() waits for an incoming datagram (or InterruptedIOException) (in ms)

```
public void setSoTimeout(int timeout) throws SocketException
public int getSoTimeout() throws IOException
```

- SO_RCVBUF: the max size of datagram packets can be received
  - Platform dependent, ie. 64K for Linux

```
public void setReceiveBufferSize(int size) throws SocketException
public int getReceiveBufferSize() throws SocketException
```

- SO_SNDBUF: suggested send buffer size used for network output

```
public void setSendBufferSize(int size)  throws SocketException
public int getSendBufferSize() throws SocketException
```

- SO_REUSEADDR: controls whether multiple datagram sockets can bind to the same port and address at the same time
  - If multiple sockets are bound to the same port, received packets will be copied to all bound sockets

```
public void setReuseAddress(boolean on) throws SocketException
public boolean getReuseAddress() throws SocketException
```

  - setReuseAddress() must be called before the new socket binds to the port
- SO_BROADCAST: controls whether a socket is allowed to send packets to and receive packets from broadcast addresses

```
public void setBroadcast(boolean on) throws SocketException
public boolean getBroadcast() throws SocketException
```

- IP_TOS: set or inspect TOS field (class of service) of IP header for a socket

```
public int getTrafficClass() throws SocketException
public void setTrafficClass(int trafficClass) throws SocketException
```

12.7 Some Useful Applications

- Simple UDP clients
  - Base Class: UDPPoke Class (Example 12-7)
    - Sends an empty UDP packet to a specified host and port and reads a response packet from the same host
      - BufferSize: default 8192 bytes
      - Timeout: default 30,000 ms
  - Example 12-8: A UDP Time Client (UDPTimeClient)
    - Convert returned four raw bytes into a java.util.Date object
- UDP Server
  - Base Class: UDPServer Class (Example 12-9)
    - A simple iterative UDP server class that can be sublcassed to provide specific servers for different protocols
      - BufferSize: default 8192 bytes
      - DatagramSocket: the constructor opens a socket on a specified local port to receive datagrams
  - Example 12-10: A UDP Discard Server (UDPDiscardServer)
  - Example 12-11: A UDP Echo Server (UDPEchoServer)
- A UDP Echo Client: three threads use the same DatagramSocket
  - UDPEchoClient (Main thread): create the socket and threads
  - SenderThread: process user input and send it to the echo server
  - ReceiverThread: receive echoed packet and output to console

Example 12-7. UDPPoke

```
import java.io.*;
import java.net.*;
public class UDPPoke {
  private int bufferSize; // in bytes
  private int timeout; // in milliseconds
  private InetAddress host;
  private int port;

  public UDPPoke(InetAddress host, int port, int bufferSize,int timeout){
    this.bufferSize = bufferSize;
    this.host = host;
    if (port < 1 || port > 65535) {
      throw new IllegalArgumentException("Port out of range");
    }
    this.port = port;
    this.timeout = timeout;
  }
  public UDPPoke(InetAddress host, int port, int bufferSize) {
    this(host, port, bufferSize, 30000);
  }

  public UDPPoke(InetAddress host, int port) {
    this(host, port, 8192, 30000);
  }

  public byte[] poke() {
    try (DatagramSocket socket = new DatagramSocket(0)) {
      DatagramPacket outgoing = new DatagramPacket(new byte[1], 1, host, port);
      socket.connect(host, port);
      socket.setSoTimeout(timeout);

      socket.send(outgoing);
      DatagramPacket incoming
          = new DatagramPacket(new byte[bufferSize], bufferSize);
      // next line blocks until the response is received
      socket.receive(incoming);
      int numBytes = incoming.getLength();
      byte[] response = new byte[numBytes];
      System.arraycopy(incoming.getData(), 0, response, 0, numBytes);
      return response;
    } catch (IOException ex) {
      return null;
  } }
```

Send 1-byte and wait to receive up to 8KB

```
  public static void main(String[] args) {
    InetAddress host;
    int port = 0;
    try {
      host = InetAddress.getByName(args[0]);
      port = Integer.parseInt(args[1]);
    } catch (RuntimeException | UnknownHostException ex) {
      System.out.println("Usage: java UDPPoke host port");
      return;
    }

    try {
      UDPPoke poker = new UDPPoke(host, port);
      byte[] response = poker.poke();
      if (response == null) {
        System.out.println("No response within allotted time");
        return;
      }
      String result = new String(response, "US-ASCII");
      System.out.println(result);
    } catch (UnsupportedEncodingException ex) {
      // Really shouldn't happen
      ex.printStackTrace();
    }
  }
}
```

```
$ java UDPPoke rama.poly.edu 13
Sun Oct  3 13:04:22 2009
$ java UDPPoke rama.poly.edu 19
123456789:;<=>?@ABCDEFGHIJKLMNOPQRSTUVWXYZ[\]^_`abcdefghijklmnopqrstuv
```

Example 12-8. UDPTimeClient

```
import java.net.*;
import java.util.*;
public class UDPTimeClient {
  public final static int PORT = 37;
  public final static String DEFAULT_HOST = "time.nist.gov";
  public static void main(String[] args) {
    InetAddress host;
    try {
      if (args.length > 0) {
        host = InetAddress.getByName(args[0]);
      } else {
        host = InetAddress.getByName(DEFAULT_HOST);
      }
    } catch (RuntimeException | UnknownHostException ex) {
      System.out.println("Usage: java UDPTimeClient [host]");
      return;
    }
    UDPPoke poker = new UDPPoke(host, PORT);
    byte[] response = poker.poke();
    if (response == null) {
      System.out.println("No response within allotted time");
      return;
    } else if (response.length != 4) {
      System.out.println("Unrecognized response format");
      return;
    }
    // The time protocol sets the epoch at 1900,
    // the Java Date class at 1970. This number
    // converts between them.
    long differenceBetweenEpochs = 2208988800L;
    long secondsSince1900 = 0;
    for (int i = 0; i < 4; i++) {
      secondsSince1900
          = (secondsSince1900 << 8) | (response[i] & 0x000000FF);
    }
    long secondsSince1970=secondsSince1900-differenceBetweenEpochs;
    long msSince1970 = secondsSince1970 * 1000;
    Date time = new Date(msSince1970);
    System.out.println(time);
} }
```

Call UDPPoke (Example 12-7) to send/receive datagrams

Convert returned 4 raw bytes into a java.util.Date object

Example 12-9. The UDPServer Class

```
import java.io.*;
import java.net.*;
import java.util.logging.*;
public abstract class UDPServer implements Runnable {
  private final int bufferSize; // in bytes
  private final int port;
  private final Logger logger=Logger.getLogger(UDPServer.class.getCanonicalName());
  private volatile boolean isShutDown = false;

  public UDPServer(int port, int bufferSize) {
    this.bufferSize = bufferSize;
    this.port = port;
  }
  public UDPServer(int port) {
    this(port, 8192);
  }
  @Override
  public void run() {
    byte[] buffer = new byte[bufferSize];
    try (DatagramSocket socket = new DatagramSocket(port)) {
      socket.setSoTimeout(10000); // check every 10 seconds for shutdown
      while (true) {
        if (isShutDown) return;
        DatagramPacket incoming = new DatagramPacket(buffer, buffer.length);
        try {
          socket.receive(incoming);
          this.respond(socket, incoming);
        } catch (SocketTimeoutException ex) {
          if (isShutDown) return;
        } catch (IOException ex) {
          logger.log(Level.WARNING, ex.getMessage(), ex);
        }
      } // end while
    } catch (SocketException ex) {
      logger.log(Level.SEVERE, "Could not bind to port: " + port, ex);
  } }
  public abstract void respond(DatagramSocket socket, DatagramPacket request)
      throws IOException;

  public void shutDown() {
    this.isShutDown = true;
} }
```

- Implements Runnable
  - Repeatedly receives an incoming datagram and responds by passing it to the abstract respond() method
    - Does not spawn a thread
  - shutDown() sets a flag to exit

Example 12-10. A UDP Discard Server

```
import java.net.*;

public class FastUDPDiscardServer extends UDPServer {

  public final static int DEFAULT_PORT = 9;

  public FastUDPDiscardServer() {
    super(DEFAULT_PORT);
  }

  public static void main(String[] args) {
    UDPServer server = new FastUDPDiscardServer();
    Thread t = new Thread(server);
    t.start();
  }
  @Override
  public void respond(DatagramSocket socket, DatagramPacket request) {
  }
}
```

Extends UDPServer(Example 12-9) by providing respond() to discard the datagram

Example 12-11. A UDP Echo Server

```
import java.io.*;
import java.net.*;

public class UDPEchoServer extends UDPServer {

  public final static int DEFAULT_PORT = 7;

  public UDPEchoServer() {
    super(DEFAULT_PORT);
  }

  @Override
  public void respond(DatagramSocket socket, DatagramPacket packet)
      throws IOException {
    DatagramPacket outgoing = new DatagramPacket(packet.getData(),
        packet.getLength(), packet.getAddress(), packet.getPort());
    socket.send(outgoing);
  }

  public static void main(String[] args) {
    UDPServer server = new UDPEchoServer();
    Thread t = new Thread(server);
    t.start();
  }
}
```

Create a DatagramPacket to send back the received data

A Multi-threaded UDP Echo Client

```
import java.net.*;

public class UDPEchoClient {

  public final static int PORT = 7;

  public static void main(String[] args) {

    String hostname = "localhost";
    if (args.length > 0) {
      hostname = args[0];
    }

    try {
      InetAddress ia = InetAddress.getByName(hostname);
      DatagramSocket socket = new DatagramSocket();
      SenderThread sender = new SenderThread(socket, ia, PORT);
      sender.start();
      Thread receiver = new ReceiverThread(socket);
      receiver.start();
    } catch (UnknownHostException ex) {
      System.err.println(ex);
    } catch (SocketException ex) {
      System.err.println(ex);
    }
  }
}
```

- UDPEchoClient (Example 12-12)
  - Create the DatagramSocket
  - Create SenderThread to process user input (Example 12-13)
  - Create ReceiverThread to receive echo (Example 12-14)

Example 12-13. SenderThread

```
import java.io.*;
import java.net.*;
class SenderThread extends Thread {
  private InetAddress server;
  private DatagramSocket socket;
  private int port;
  private volatile boolean stopped = false;
  SenderThread(DatagramSocket socket, InetAddress address, int port) {
    this.server = address;
    this.port = port;
    this.socket = socket;
    this.socket.connect(server, port);
  }
  public void halt() {
    this.stopped = true;
  }
  @Override
  public void run() {
    try {
      BufferedReader userInput
          = new BufferedReader(new InputStreamReader(System.in));
      while (true) {
        if (stopped) return;
        String theLine = userInput.readLine();
        if (theLine.equals(".")) break;
        byte[] data = theLine.getBytes("UTF-8");
        DatagramPacket output
            = new DatagramPacket(data, data.length, server, port);
        socket.send(output);
        Thread.yield();
      }
    } catch (IOException ex) {
      System.err.println(ex);
} } }
```

Ensure to specify the destination

Read line from console

Example 12-14. ReceiverThread

```
import java.io.*;
import java.net.*;
class ReceiverThread extends Thread {
  private DatagramSocket socket;
  private volatile boolean stopped = false;
  ReceiverThread(DatagramSocket socket) {
    this.socket = socket;
  }
  public void halt() {
    this.stopped = true;
  }
  @Override
  public void run() {
    byte[] buffer = new byte[65507];
    while (true) {
      if (stopped) return;
      DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
      try {
        socket.receive(dp);
        String s = new String(dp.getData(), 0, dp.getLength(), "UTF-8");
        System.out.println(s);
        Thread.yield();
      } catch (IOException ex) {
        System.err.println(ex);
} } } }
```

Both export halt() and call Thread.yield()

12.8 DatagramChannel

- DatagramChannel class: support nonblocking UDP applications
  - Methods return quickly if network isn't immediately ready to receive or send data
    - Since UDP is more asynchronous than TCP, the net effect is smaller
  - Can be registered with a Selector
    - Monitors multiple sockets
- Using DatagramChannel
  - Java 6-: need to use the DatagramSocket class to bind a channel to a port
  - Java 7+: need not to use DatagramSocket, nor DatagramPacket
    - Read and write byte buffers as do with a SocketChannel

Using DatagramChannel

- open(): open a socket
  - Not initially bound to any port. Bind it with the channel's socket object

Java 6-

```
DatagramChannel channel = DatagramChannel.open();
SocketAddress address = new InetSocketAddress(3141);
DatagramSocket socket = channel.socket();
socket.bind(address);
```

Java 7+

```
DatagramChannel channel = DatagramChannel.open();
SocketAddress address = new InetSocketAddress(3141);
channel.bind(address);
```

- receive(): read one datagram packet from the channel into a ByteBuffer
  - If the packet has more data than the buffer can hold, the extra data is thrown away with no notification of the problem

```
public SocketAddress receive(ByteBuffer dst) throws IOException
```

- send(): write one datagram packet into the channel from a ByteBuffer

```
public int send(ByteBuffer src, SocketAddress target) throws IOException
```

  - Return the number of bytes written; the source ByteBuffer can be reused
    - 0 if the channel is in nonblocking mode and the data can't be sent immediately
- close(): close the socket
  - isOpen(): return false if the channel is closed

Java 6-

```
DatagramChannel channel = null;
try {
  channel = DatagramChannel.open();
  // Use the channel...
} catch (IOException ex) {
  // handle exceptions...
} finally {
  if (channel != null) {
    try {
      channel.close();
    } catch (IOException ex){// ignore
} } }
```

Java 7+ (AutoCloseable)

```
try (DatagramChannel channel = DatagramChannel.open()) {
  // Use the channel...
} catch (IOException ex) {
  // handle exceptions...
}
```

Reading and Writing on Connected Channels

- connect(): enforce to connect to a particular remote address
  - Does not block in any meaningful sense

```
SocketAddress remote = new InetSocketAddress("time.nist.gov", 37);
channel.connect(remote);
```

  - The channel will only send data to or receive data from this address
  - isConnected(): tell if the DatagramChannel is limited to one host

```
public boolean isConnected()
```

  - Disconnect(): break the connection

```
public DatagramChannel disconnect() throws IOException
```

- read(): like receive(), but only used on connected channels

```
public int read(ByteBuffer dst) throws IOException
public long read(ByteBuffer[] dsts) throws IOException
public long read(ByteBuffer[] dsts, int offset, int length)throws IOException
```

  - Only read a single datagram packet from the network
    - If the packet has more data than the ByteBuffer can hold, the extra data is thrown away
  - Return the number of bytes, -1 if the channel has been closed, or 0 if
    - The channel is nonblocking and no packet was ready
    - A datagram packet contained no data
    - The buffer is full
- write(): like send(), but also only used on connected channels

```
public int write(ByteBuffer src) throws IOException
public long write(ByteBuffer[] dsts) throws IOException
public long write(ByteBuffer[] dsts, int offset, int length)throws IOException
```

  - Not guaranteed to write the complete contents of the buffer
    - But can call again and again until the buffer is fully drained and sent

```
while (buffer.hasRemaining() && channel.write(buffer) != -1) ;
```

Example 12-15. UDPDiscardServerWithChannels

```
import java.io.*;
import java.net.*;
import java.nio.*;
import java.nio.channels.*;
public class UDPDiscardServerWithChannels {
  public final static int PORT = 9;
  public final static int MAX_PACKET_SIZE = 65507;
  public static void main(String[] args) {
    try {
      DatagramChannel channel = DatagramChannel.open();
      DatagramSocket socket = channel.socket();
      SocketAddress address = new InetSocketAddress(PORT);
      socket.bind(address);
      ByteBuffer buffer = ByteBuffer.allocateDirect(MAX_PACKET_SIZE);
      while (true) {
        SocketAddress client = channel.receive(buffer);
        buffer.flip();
        System.out.print(client + " says ");
        while (buffer.hasRemaining()) System.out.write(buffer.get());
        System.out.println();
        buffer.clear();
      }
    } catch (IOException ex) {
      System.err.println(ex);
} } }
```

Open and bind to a port

Loop and receive to a ByteBuffer

Output to the console (Discard; no return)

Example 12-16. UDPEchoServerWithChannels

```
import java.io.*;
import java.net.*;
import java.nio.*;
import java.nio.channels.*;
public class UDPEchoServerWithChannels {
  public final static int PORT = 7;
  public final static int MAX_PACKET_SIZE = 65507;
  public static void main(String[] args) {
    try {
      DatagramChannel channel = DatagramChannel.open();
      DatagramSocket socket = channel.socket();
      SocketAddress address = new InetSocketAddress(PORT);
      socket.bind(address);
      ByteBuffer buffer = ByteBuffer.allocateDirect(MAX_PACKET_SIZE);
      while (true) {
        SocketAddress client = channel.receive(buffer);
        buffer.flip();
        channel.send(buffer, client);
        buffer.clear();
      }
    } catch (IOException ex) {
      System.err.println(ex);
} } }
```

Loop and receive to a ByteBuffer

Echo back to client

Example 12-17. UDPEchoClientWithChannels

- Send one hundred ints from 0 to 99
- Print out the values returned
- Figure out if any packets are being lost

```
import java.io.*;
import java.net.*;
import java.nio.*;
import java.nio.channels.*;
import java.util.*;
public class UDPEchoClientWithChannels {
  public  final static int PORT = 7;
  private final static int LIMIT = 100;
  public static void main(String[] args) {
    SocketAddress remote;
    try {
      remote = new InetSocketAddress(args[0], PORT);
    } catch (RuntimeException ex) {
      System.err.println("Usage: java UDPEchoClientWithChannels host");
      return;
    }
    try (DatagramChannel channel = DatagramChannel.open()) {
      channel.configureBlocking(false);
      channel.connect(remote);
      Selector selector = Selector.open();
      channel.register(selector,SelectionKey.OP_READ|SelectionKey.OP_WRITE);
      ByteBuffer buffer = ByteBuffer.allocate(4);
      int n = 0;
      int numbersRead = 0;
```

Create socket & channel and register to selector

```
Wrote: 0
Read: 0
Wrote: 1
Wrote: 2
Read: 1
Wrote: 3
...
Read: 97
Read: 98
Read: 99
Echoed 92 out of 100 sent
Success rate: 92.0%
```

Loop to check selector's keys

```
      while (true) {
        if (numbersRead == LIMIT) break;
        // wait one minute for a connection
        selector.select(60000);
        Set<SelectionKey> readyKeys = selector.selectedKeys();
        if (readyKeys.isEmpty() && n == LIMIT) {
          // All packets have been written and it doesn't look like any
          // more are will arrive from the network
          break;
        } else {
          Iterator<SelectionKey> iterator = readyKeys.iterator();
          while (iterator.hasNext()) {
            SelectionKey key = (SelectionKey) iterator.next();
            iterator.remove();
            if (key.isReadable()) {
              buffer.clear();
              channel.read(buffer);
              buffer.flip();
              int echo = buffer.getInt();
              System.out.println("Read: " + echo);
              numbersRead++;
            }
            if (key.isWritable()) {
              buffer.clear();
              buffer.putInt(n);
              buffer.flip();
              channel.write(buffer);
              System.out.println("Wrote: " + n);
              n++;
              if (n == LIMIT) {
                // All packets have been written; switch to read-only mode
                key.interestOps(SelectionKey.OP_READ);
      } } } } }
      System.out.println("Echoed "+numbersRead+" out of "+LIMIT+" sent");
      System.out.println("Success rate: " + 100.0*numbersRead/LIMIT+"%");
    } catch (IOException ex) {
      System.err.println(ex);
    } } }
```

Print out and count returned value

Send, print out and count the sent value

Print out statistics

DatagramChannel Socket Options (Java 7+)

- The first 5 options are the same as general Socket Options; the last 3 options are used by multicast sockets

Table 12-1. Socket options supported by datagram sockets

Option	Type	Constant	Purpose
SO_SNDBUF	StandardSocketOptions.	Integer	Size of the buffer used for sending datagram packets
SO_RCVBUF	StandardSocketOptions.SO_RCVBUF	Integer	Size of the buffer used for receiving datagram packets
SO_REUSEADDR	StandardSocketOptions.SO_REUSEADDR	Boolean	Enable/disable address reuse
SO_BROADCAST	StandardSocketOptions.SO_BROADCAST	Boolean	Enable/disable broadcast messages
IP_TOS	StandardSocketOptions.IP_TOS	Integer	Traffic class
IP_MULTICAST_IF	StandardSocketOptions.IP_MULTICAST_IF	NetworkInterface	Local network interface to use for multicast
IP_MULTICAST_TTL	StandardSocketOptions.IP_MULTICAST_TTL	Integer	Time-to-live value for multicast datagrams
IP_MULTICAST_LOOP	StandardSocketOptions.IP_MULTICAST_LOOP	Boolean	Enable/disable loopback of multicast datagrams

- setOptions(), getOptions() and supportedOptions()

```
public <T> DatagramChannel setOption(SocketOption<T> name, T value)
    throws IOException
public <T> T getOption(SocketOption<T> name) throws IOException
public Set<SocketOption<?>> supportedOptions()
```

```
try (DatagramChannel channel = DatagramChannel.open()) {
  channel.setOption(StandardSocketOptions.SO_BROADCAST, true);
  // Send the broadcast message...
} catch (IOException ex) {
  // handle exceptions...
}
```

Example 12-18. DefaultSocketOptionValues

- Open a channel to check the default values of socket options

```
import java.io.IOException;
import java.net.SocketOption;
import java.nio.channels.DatagramChannel;

public class DefaultSocketOptionValues {

  public static void main(String[] args) {
    try (DatagramChannel channel = DatagramChannel.open()) {
      for (SocketOption<?> option : channel.supportedOptions()) {
        System.out.println(option.name() + ": " + channel.getOption(option));
      }
    } catch (IOException ex) {
      ex.printStackTrace();
    }
  }
}
```

```
IP_MULTICAST_TTL: 1
SO_BROADCAST: false
SO_REUSEADDR: false
SO_RCVBUF: 196724
IP_MULTICAST_LOOP: true
SO_SNDBUF: 9216
IP_MULTICAST_IF: null
IP_TOS: 0
```

Summary

12.1 The UDP Protocol
12.2 UDP Clients
- DaytimeUDPClient (Example 12-1)

12.3 UDP Servers
- DaytimeUDPServer (Example 12-2)

12.4 The DatagramPacket Class
- DatagramExample (Example 12-3)

12.5 The DatagramSocket Class
- UDPPortScanner (Example 12-4)
- UDPDiscardClient (Example 12-5), UDPDiscardServer (Example 12-6)

12.6 Socket Options
12.7 Some Useful Applications
- UDPPoke (Example 12-7), UDPTimeClient (Example 12-8)
- UDPServer (Example 12-9), FastUDPDiscardServer (12-10), UDPEchoServer (12-11)
- A Multi-threaded UDP Echo Client
  UDPEchoClient (Example 12-12), SenderThread (12-13), ReceiverThread (12-14)

12.8 DatagramChannel
- UDPDiscardServerWithChannels (Example 12-15)
- UDPEchoServerWithChannels (Example 12-16)
- UDPEchoClientWithChannels (Example 12-17)
- DefaultSocketOptionValues (Example 12-18)

Ch.13 IP Multicast

Unit-11

Outline

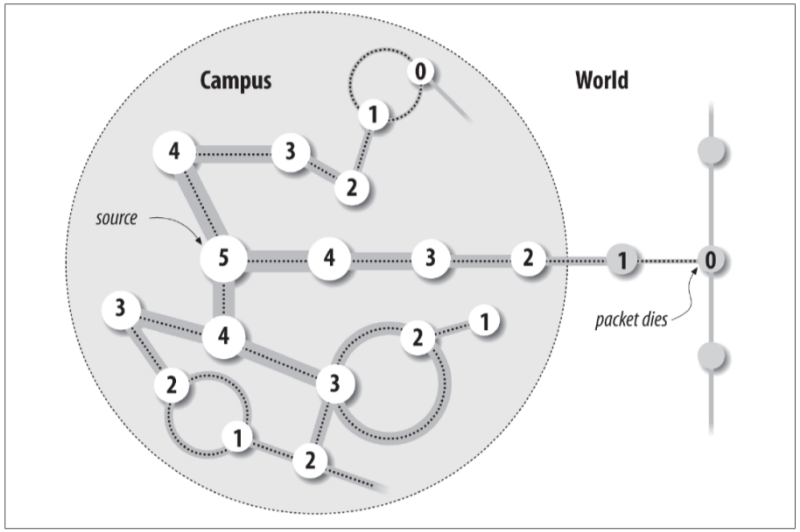
13.1 Multicasting

13.2 Working with Multicast Sockets

13.3 Two Simple Examples

13.1 Multicasting

- Multicasting: sends data from one host to many different hosts
  - Not to everyone: only clients expressing an interest by joining a multicast group
  - Examples:
    - Apple's Bonjour (a.k.a. Zeroconf) and Apache's River both use IP multicasting to dynamically discover services on the local network
- Most of the work is done by routers and transparent to programmers
  - The routers make sure the packet is delivered to all the hosts in the multicast group
    - The biggest problem is that not all routers support multicasting
  - Time-To-Live (TTL) of IP: maximum number of routers that the datagram is allowed



Destinations	TTL value
The local host	0
The local subnet	1
The local campus—that is, the same side of the nearest Internet router—but on possibly different LANs	16
High-bandwidth sites in the same country, generally those fairly close to the backbone	32
All sites in the same country	48
All sites on the same continent	64
High-bandwidth sites worldwide	128
All sites worldwide	255

Multicast Addresses and Groups

- A multicast address is the shared address of a group of hosts called a multicast group
  - IPv4 CIDR group: 224.0.0.0/4 (224.0.0.0 to 239.255.255.255)
    - All addresses have the leading four binary digits 1110
  - IPv6 CIDR group: ff00::/8
    - All start with the byte 0xFF, or 1111 1111 in binary
- A multicast group
  - Open: enter or leave at any time
  - Either permanent or transient
    - Permanent: the assigned address remains constant
      - Assigned: ***224.1.*** or ***224.2.***
      - The complete list is available from iana.org
    - Transient: exist only as long as they have members
      - Create a new multicast group address from ***225.***0.0.0 to ***238.***255.255.255
- Link-local multicast addresses: from ***224.0.0.***0 to ***224.0.0.***255
  - Packets addressed to a multicast group from 224.0.0.0 to 224.0.0.255 are never forwarded beyond the local subnet

Table 13-1. Link-local Multicast Addresses

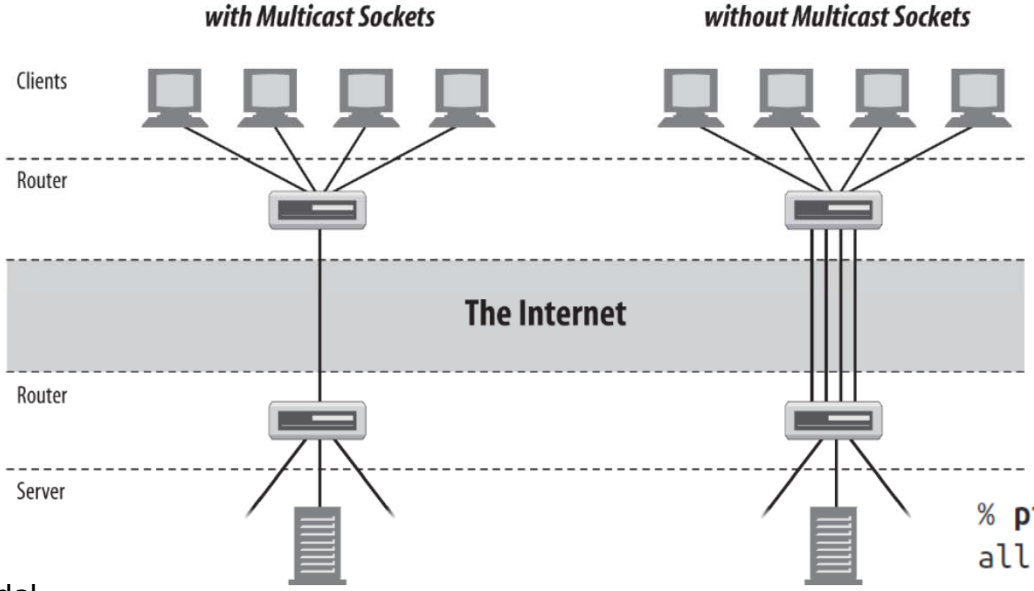
Domain name	IP address	Purpose
BASE-ADDRESS.MCAST.NET	*224.0.0.0*	The reserved base address. This is never assigned to any multicast group.
ALL-SYSTEMS.MCAST.NET	*224.0.0.1*	All systems on the local subnet.
ALL-ROUTERS.MCAST.NET	*224.0.0.2*	All routers on the local subnet.
DVMRP.MCAST.NET	*224.0.0.4*	All Distance Vector Multicast Routing Protocol(DVMRP) routers on this subnet.
MOBILE-AGENTS.MCAST.NET	*224.0.0.11*	Mobile agents on the local subnet.
DHCP-AGENTS.MCAST.NET	*224.0.0.12*	This multicast group allows a client to locate a Dynamic Host Configuration Protocol (DHCP) server or relay agent on the local subnet.
RSVP-ENCAPSULATION.MCAST.NET	*224.0.0.14*	RSVP encapsulation on this subnet. RSVP stands for Resource reSerVation setup Protocol, an effort to allow people to reserve a guaranteed amount of Internet bandwidth in advance for an event.
VRRP.MCAST.NET	*224.0.0.18*	Virtual Router Redundancy Protocol (VRRP) Routers
	224.0.0.35	DXCluster is used to announce foreign amateur (DX) stations.
	224.0.0.36	Digital Transmission Content Protection (DTCP), a digital restrictions management (DRM) technology that encrypts interconnections between DVD players, televisions, and similar devices.
	224.0.0.37-224.0.0.68	zeroconf addressing
	224.0.0.106	Multicast Router Discovery
	224.0.0.112	Multipath Management Agent Device Discovery
	224.0.0.113	Qualcomm's AllJoyn
	224.0.0.114	Inter RFID Reader Protocol
	224.0.0.251	Multicast DNS self assigns and resolves hostnames for multicast addresses.
	224.0.0.252	Link-local Multicast Name Resolution, a precursor of mDNS, allows nodes ot self-assign domain names strictly for the local network, and to resolve such domain names on the local network.
	224.0.0.253	Teredo is used to tunnel IPv6 over IPv4. Other Teredo clients on the same IPv4 subnet respond to this multicast address.
	224.0.0.254	Reserved for experimentation.

Table 13-2. Common Permanent Multicast Addresses

Domain name	IP address	Purpose
NTP.MCAST.NET	*224.0.1.1*	The Network Time Protocol.
NSS.MCAST.NET	*224.0.1.6*	The Name Service Server.
AUDIONEWS.MCAST.NET	*224.0.1.7*	Audio news multicast.
MTP.MCAST.NET	*224.0.1.9*	The Multicast Transport Protocol.
IETF-1-LOW-AUDIO.MCAST.NET	*224.0.1.10*	Channel 1 of low-quality audio from IETF meetings.
IETF-1-AUDIO.MCAST.NET	*224.0.1.11*	Channel 1 of high-quality audio from IETF meetings.
IETF-1-VIDEO.MCAST.NET	*224.0.1.12*	Channel 1 of video from IETF meetings.
IETF-2-LOW-AUDIO.MCAST.NET	*224.0.1.13*	Channel 2 of low-quality audio from IETF meetings.
IETF-2-AUDIO.MCAST.NET	*224.0.1.14*	Channel 2 of high-quality audio from IETF meetings.
IETF-2-VIDEO.MCAST.NET	*224.0.1.15*	Channel 2 of video from IETF meetings.
MLOADD.MCAST.NET	*224.0.1.19*	MLOADD measures the traffic load through one or more network interfaces over a number of seconds. Multicasting is used to communicate between the different interfaces being measured.
EXPERIMENT.MCAST.NET	*224.0.1.20*	Experiments.
	224.0.23.178	JDP Java Discovery Protocol, used to find manageable JVMs on the network.
MICROSOFT.MCAST.NET	*224.0.1.24*	Used by Windows Internet Name Service (WINS) servers to locate one another.
MTRACE.MCAST.NET	*224.0.1.32*	A multicast version of traceroute.
JINI-ANNOUNCEMENT.MCAST.NET	*224.0.1.84*	JINI announcements.
JINI-REQUEST.MCAST.NET	*224.0.1.85*	JINI requests.
	224.0.1.143	Emergency Managers Weather Information Network.
	224.2.0.0-224.2.255.255	The Multicast Backbone on the Internet (MBONE) addresses are reserved for multimedia conference calls (i.e., audio, video, whiteboard, and shared web browsing between many people).
	224.2.2.2	Port 9875 on this address is used to broadcast the currently available MBONE programming. You can look at this with the X Window utility sdr or the Windows/Unix multikit program.
	239.0.0.0-239.255.255.255	Organization local scope, in contrast to TTL scope, uses different ranges of multicast addresses to constrain multicast traffic to a particular region or group of routers. For example, when a Universal Plug and Play (UPnP) device joins a network, it sends an HTTPU (HTTP over UDP) message to the multicast address 239.255.255.250 on port 1900. The idea is to allow the possible group membership to be established in advance without relying on less-than-reliable TTL values.

Clients and Servers

- Send
  - Once the data has been stuffed into one or more datagrams, the sending host launches the datagrams onto the Internet like sending regular UDP data
- Receive
  - When data arrives at a host in a multicast group, the host receives it as it receives any other UDP datagram
- Mrouters: the biggest restriction on multicasting is the availability of special multicast routers
  - For packets to reach any given host, there must be a path of multicast capable routers between your host and the remote host




```
% ping all-routers.mcast.net
all-routers.mcast.net is alive
```

13-2. Working with Multicast Sockets

Java.net.MulticastSocket

```
public class MulticastSocket extends DatagramSocket
    implements Closeable, AutoCloseable
```

- Steps to receive multicast data
  1. MulticastSocket(): create a MulticastSocket with its constructor

```
MulticastSocket ms = new MulticastSocket(2300);
```

  2. joinGroup(): join a multicast group

```
InetAddress group = InetAddress.getByName("224.2.2.2");
ms.joinGroup(group);
```

  3. Receive UDP data just as with a DatagramSocket and DatagramPacket

```
byte[] buffer = new byte[8192];
DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
ms.receive(dp);
```

  4. leaveGroup(): leave the multicast group
  5. Close socket with close()

```
ms.leaveGroup(group);
ms.close();
```

- Steps to send multicast data: similar to sending UDP data
  - Create a MulticastSocket and directly send data to it
    - Do not need to join a multicast group

```
InetAddress ia = InetAddress.getByName("experiment.mcast.net");
byte[] data = "Here's some multicast data\r\n".getBytes("UTF-8");
int port = 4000;
DatagramPacket dp = new DatagramPacket(data, data.length, ia, port);
MulticastSocket ms = new MulticastSocket();
ms.send(dp);
```

Communicating with a Multicast Group

- 3 constructors

```
public MulticastSocket() throws SocketException
public MulticastSocket(int port) throws SocketException
public MulticastSocket(SocketAddress bindAddress) throws IOException
```

  - bind()

```
MulticastSocket ms = new MulticastSocket(null);
ms.setReuseAddress(false);
SocketAddress address = new InetSocketAddress(4000);
ms.bind(address);
```

- joinGroup(): join a group

```
public void joinGroup(InetAddress address) throws IOException
public void joinGroup(SocketAddress address, NetworkInterface interface)
    throws IOException
```

  - A single MulticastSocket can join multiple multicast groups
  - Multiple multicast sockets (on same or different JVMs) can all join the same group

- leaveGroup(): leave a group

```
public void leaveGroup(InetAddress address) throws IOException
public void leaveGroup(SocketAddress multicastAddress,
NetworkInterface interface)
    throws IOException
```

- close(): close the socket

Java 7+

```
try (MulticastSocket socket = new MulticastSocket()) {
  // connect to the server...
} catch (IOException ex) {
  ex.printStackTrace();
}
```

Java 6-

```
MulticastSocket socket = null;
try {
  socket = new MulticastSocket();
  // connect to the server...
} catch (IOException ex) {
  ex.printStackTrace();
} finally {
  if (socket != null) {
    try {
      socket.close();
    }catch(IOException ex){//ignore
} } }
```

Sending Multicast Data

- Sending: multicast sockets uses a TTL of 1 by default
  - setTimeToLive(): set the default TTL

```
public void setTimeToLive(int ttl) throws IOException
public int getTimeToLive() throws IOException
```

```
try {
  InetAddress ia = InetAddress.getByName("www.ibiblio.org");
  MulticastSocket ms = new MulticastSocket(2048);
  ms.setInterface(ia);
  // send and receive data...
} catch (UnknownHostException ue) {
  System.err.println(ue);
} catch (SocketException se) {System.err.println(se);}
```

  - setLoopBack(): Set not to receive packets you sends (default is platform dependent)

```
public void setLoopbackMode(boolean disable) throws SocketException
public boolean getLoopbackMode() throws SocketException
```

- setInterface(), setNetworkInterface(): specify a network interface for multicast

```
public void setInterface(InetAddress address) throws SocketException
public InetAddress getInterface() throws SocketException
public void setNetworkInterface(NetworkInterface interface) throws SocketException
public NetworkInterface getNetworkInterface() throws SocketException
```

```
try {
  InetAddress ia = InetAddress.getByName("www.ibiblio.org");
  MulticastSocket ms = new MulticastSocket(2048);
  ms.setInterface(ia);
  // send and receive data...
} catch (UnknownHostException ue) {
  System.err.println(ue);
} catch (SocketException se) {
  System.err.println(se);
}
```

13.3 Two Simple Examples

Example 13-1. MulticastSniffer

```
import java.io.*;
import java.net.*;
public class MulticastSniffer {
  public static void main(String[] args) {
    InetAddress group = null;
    int port = 0;
    // read the address from the command line
    try {
      group = InetAddress.getByName(args[0]);
      port = Integer.parseInt(args[1]);
    } catch (ArrayIndexOutOfBoundsException | NumberFormatException
        | UnknownHostException ex) {
      System.err.println("Usage: java MulticastSniffer multicast_address port");
      System.exit(1);
    }
    MulticastSocket ms = null;
    try {
      ms = new MulticastSocket(port);
      ms.joinGroup(group);
      byte[] buffer = new byte[8192];
      while (true) {
        DatagramPacket dp = new DatagramPacket(buffer, buffer.length);
        ms.receive(dp);
        String s = new String(dp.getData(), "8859_1");
        System.out.println(s);
      }
    } catch (IOException ex) {
      System.err.println(ex);
    } finally {
      if (ms != null) {
        try {
          ms.leaveGroup(group);
          ms.close();
        } catch (IOException ex) {}
} } } }
```

Leave the group and close the socket

– Read Multicast Address and Port
  • UPnP (HTTPU/HTTP over UDP) 239.255.255.250 on port 1900

– Create the MulticastSocket

– Join the group
– Receive/sniffer datagrams

```
$ java MulticastSniffer 239.255.255.250 1900
NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age=1800
LOCATION: http://192.168.1.2:23519/Ircc.xml
NT: upnp:rootdevice
NTS: ssdp:alive
SERVER: Android/3.2 UPnP/1.0 Internet TV Box NSZ-GT1/1.0
USN: uuid:34567890-1234-1010-8000-544249cb49fd::upnp:rootdevice
X-AV-Server-Info: av=5.0; hn=""; cn="Sony Corporation";
mn="Internet TV Box NSZ-GT1"; mv="1.0";

NOTIFY * HTTP/1.1
HOST: 239.255.255.250:1900
CACHE-CONTROL: max-age=1800
LOCATION: http://192.168.1.2:23519/Ircc.xml
NT: uuid:34567890-1234-1010-8000-544249cb49fd
NTS: ssdp:alive
SERVER: Android/3.2 UPnP/1.0 Internet TV Box NSZ-GT1/1.0
USN: uuid:34567890-1234-1010-8000-544249cb49fd
X-AV-Server-Info: av=5.0; hn=""; cn="Sony Corporation";
mn="Internet TV Box NSZ-GT1"; mv="1.0";
```

Example 13-2. MulticastSender

```
import java.io.*;
import java.net.*;
public class MulticastSender {
  public static void main(String[] args) {
    InetAddress ia = null;
    int port = 0;
    byte ttl = (byte) 1;
    // read the address from the command line
    try {
      ia = InetAddress.getByName(args[0]);
      port = Integer.parseInt(args[1]);
      if (args.length > 2) ttl = (byte) Integer.parseInt(args[2]);
    } catch (NumberFormatException | IndexOutOfBoundsException
        | UnknownHostException ex)  {
      System.err.println(ex);
      System.err.println(
          "Usage: java MulticastSender multicast_address port ttl");
      System.exit(1);
    }
    byte[] data = "Here's some multicast data\r\n".getBytes();
    DatagramPacket dp = new DatagramPacket(data, data.length, ia, port);
    try (MulticastSocket ms = new MulticastSocket()) {
      ms.setTimeToLive(ttl);
      ms.joinGroup(ia);
      for (int i = 1; i < 10; i++) {ms.send(dp);}
      ms.leaveGroup(ia);
    } catch (SocketException ex) {
      System.err.println(ex);
    } catch (IOException ex) {
      System.err.println(ex);
} } }
```

- Read input from the command line and send it to a multicast group
  - Read Multicast Address and Port
  - Prepare the packet to send
  - Create MulticastSocket
  - Join, send, and leave

```
% java MulticastSniffer all-systems.mcast.net 4000
% java MulticastSender all-systems.mcast.net 4000
```

```
Here's some multicast data
Here's some multicast data
Here's some multicast data
Here's some multicast data
Here's some multicast data
Here's some multicast data
Here's some multicast data
Here's some multicast data
Here's some multicast data
```

Summary

13.1 Multicasting

- Multicast group and addresses

13.2 Working with Multicast Sockets

- Java.net.MulticastSocket
- joinGroup(), leaveGroup(), bind(), close()

13.3 Two Simple Examples

- MulticastSniffer (Example 13-1), MulticastSender (Example 13-2)