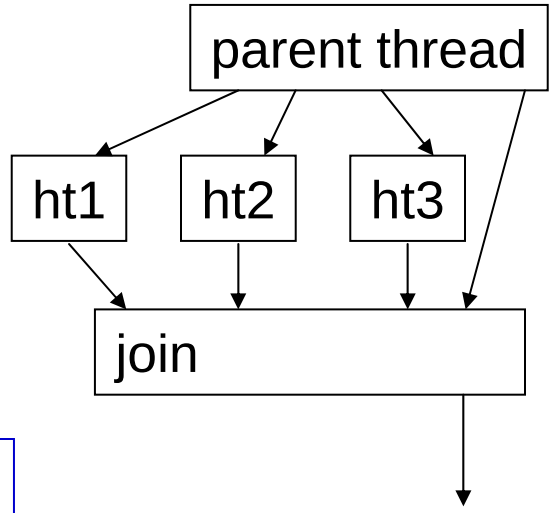
ECE 462
Object-Oriented Programming using C++ and Java

Lecture 15

Yung-Hsiang Lu

yunglu@purdue.edu

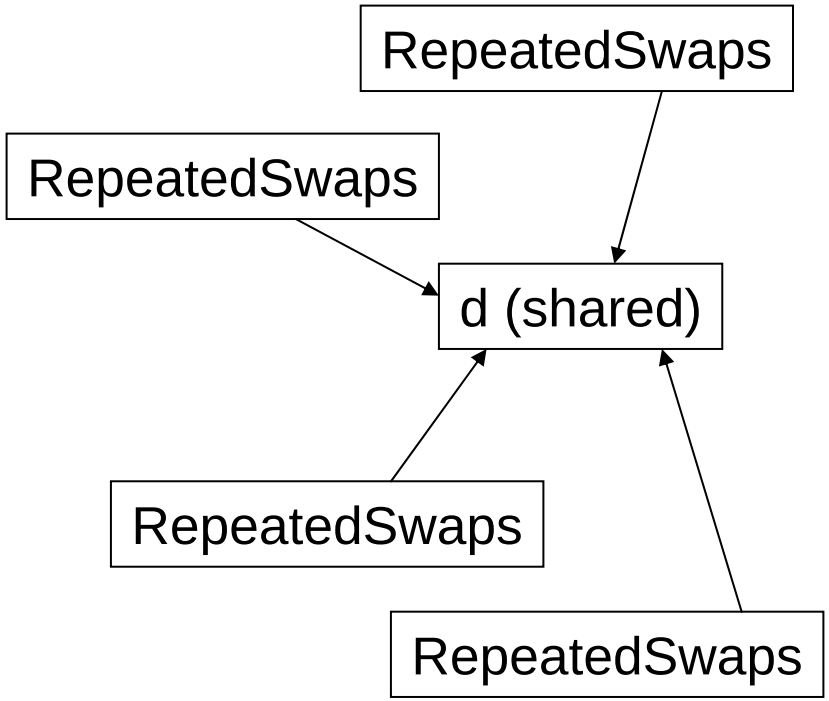
```
//ThreadsBasicWithJoin.java
class HelloThread extends Thread {
    String message;
    HelloThread( String message ) { this.message = message; }
    public void run() { System.out.print( message );   }
    public static void main(String[] args) throws InterruptedException
    {
        HelloThread ht1 = new HelloThread( "Good" );
        HelloThread ht2 = new HelloThread( " morning" );
        HelloThread ht3 = new HelloThread( " to" );
        ht1.start();
        ht2.start();
        ht3.start();
        ht1.join();
        ht2.join();
        ht3.join();
        System.out.println( " you!" );
    }
}
```



no shared data / object

Multi-Thread with Shared Object

```
//UnsynchedSwaps.java
class DataObject {
    int dataItem1;
    int dataItem2;
    DataObject() {
        dataItem1 = 50;
        dataItem2 = 50;
    }
   void test() {
        int sum = dataItem1 + dataItem2;
        System.out.println( sum );
    }
```



```
 void itemSwap() {
        int x = (int) ( -4.999999 + Math.random() * 10 );
        dataItem1 -= x;
        keepBusy(10);
        dataItem2 += x;    // should dataItem1 + dataItem2 be 100?
    }
 public void keepBusy( int howLong ) {
        long curr = System.currentTimeMillis();
        while ( System.currentTimeMillis() < curr + howLong ) ;
    }
}
class RepeatedSwaps extends Thread  {
    DataObject dobj;
    RepeatedSwaps( DataObject d ) {
        dobj = d;
        start();
    }
```

```
    public void run( ) {
        int i = 0;
        while ( i < 20000 ) {
            dobj.itemSwap();
            if ( i % 4000 == 0 ) dobj.test();
            try { sleep( (int) (Math.random() * 2 ) ); }
            catch( InterruptedException e ) {}
            i++;
        }
    }
}
public class UnsynchedSwaps {
    public static void main( String[] args ) {
        DataObject d = new DataObject();
        new RepeatedSwaps( d );
        new RepeatedSwaps( d );
        new RepeatedSwaps( d );
        new RepeatedSwaps( d );
    }
}
```

Atomic Operation

- An operation is "atomic" if it must execute to completion once the operation starts.
- In Java, only 32-bit (or smaller) assignments are atomic.
- Even x += y; is not atomic.
- Atomicity is important for consistency.

```
void itemSwap() {
  int x = (int) ( -4.999999 + Math.random() * 10 );
  dataItem1 -= x;
  keepBusy(10);
  dataItem2 += x;
}
```

Thread 1	Thread 2	Thread 3
int x = 3;		
dataItem1 -= x;		
	int x = 5;	
	dataItem1 -= x;	
		println(dataItem1 + dataItem2);
		// result < 100
	dataItem2 += x;	
dataItem2 += x;		

time

Does it help if x is a shared variable? No

This is also called "race condition".

Synchronization
(the source of many, if not most, problems)

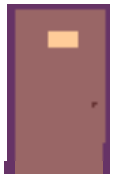
- Most threads access (read, write, or both) some **shared data** at some point of the execution.
- examples of shared data
  - how many copies of the same book in Amazon.com (many customers may order the same book)
  - seat assignment in a flight (several travelers may want the same seat)
  - credit card balance (if two or more cards are issued, or used for on-line purchase at the same time)
- If threads do not share anything, synchronization is unnecessary. These threads are independent.

Mutual Exclusion (Mutex)

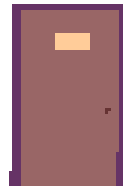
- If one has started, the others have to wait until the currently running one completes.
- Mutex can be achieved in many ways. All of them can be simplified to "test-and-set".
  - A thread tests a condition.
  - If the condition is not set, the thread sets the condition to prevent other mutex. After this thread finishes its mutex, reset the condition.
  - If the condition is already set, the thread waits.
- "test-and-set" has to be atomic.

What is in a Mutex

- When a shared (by multiple threads) variable / object may be modified and can lead to inconsistency. If the variable is read-only, mutex is unnecessary.
- If a variable / object is read by a thread and written by another thread, both threads must use mutex to protect the variable / object.
- Mutex **serializes** a multi-thread program.
- The code that is protected by Mutex is called **critical section**.



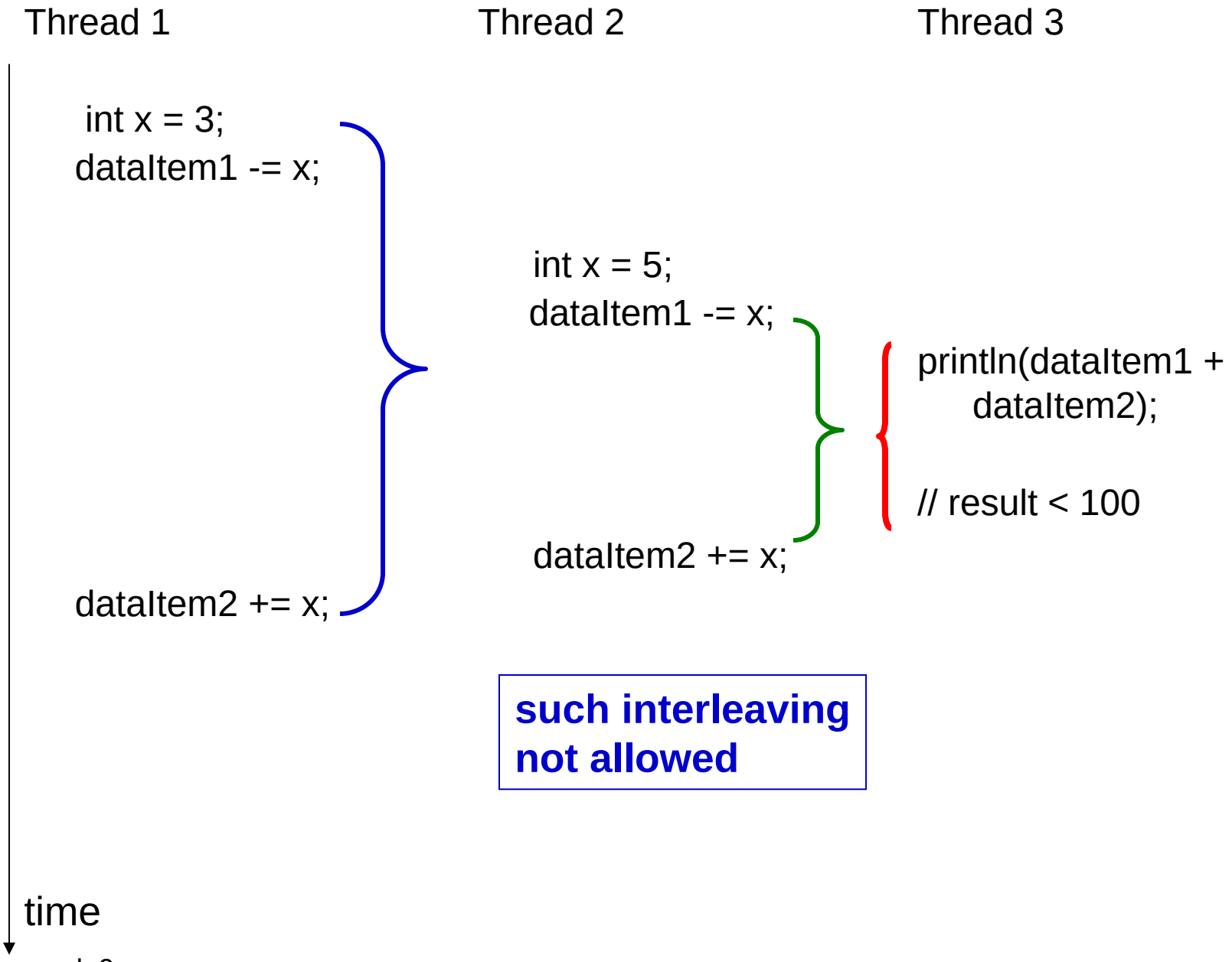
Only one is allowed in the room. If the room is locked by someone, no one else cannot enter the room, until the person in the room leaves.



```
//SynchedSwaps.java
class DataObject {
    int dataItem1;
    int dataItem2;
    DataObject() {
        dataItem1 = 50;
        dataItem2 = 50;
    }
    synchronized void itemSwap() {
        int x = (int) ( -4.999999 + Math.random() * 10 );
        dataItem1 -= x;
        keepBusy(10);
        dataItem2 += x;
    }
```

```
    synchronized void test() {
        int sum = dataItem1 + dataItem2;
        System.out.println( sum );
    }
    public void keepBusy( int howLong ) { // why not synchronized?
        long curr = System.currentTimeMillis();
        while ( System.currentTimeMillis() < curr + howLong )
            ;
    }
}
class RepeatedSwaps extends Thread  {
    DataObject dobj;
    RepeatedSwaps( DataObject d ) {
        dobj = d;
        start();
    }
```

```
 public void run( ) {
        int i = 0;
        while ( i++ < 20000 ) {
            dobj.itemSwap();
            if ( i % 4000 == 0 ) dobj.test();
            try { sleep( 1 ); } catch( InterruptedException e ) {}
        }
    }
}
public class SynchedSwaps {
    public static void main( String[] args ) {
        DataObject d = new DataObject();
        new RepeatedSwaps(  d );
        new RepeatedSwaps(  d );
        new RepeatedSwaps(  d );
        new RepeatedSwaps(  d );
    }
}
```


Thread 1
Thread 2
Thread 3
int x = 3;
dataItem1 -= x;
dataItem2 += x;
int x = 5;
dataItem1 -= x;
dataItem2 += x;
println(dataItem1 + dataItem2);
// result < 100
such interleaving not allowed
time

```
//MultiCustomerAccount.java
class Account {
    int balance;
    Account() { balance = 0; }
    synchronized void deposit( int dep ){
        balance += dep;
        notifyAll();
    }
    synchronized void withdraw( int draw ) {
        while ( balance < draw ) {
            try {
                wait();
            } catch( InterruptedException e ) {}
        }
        balance -= draw;
    }
}
```

```
class Depositor extends Thread {
    private Account sacct;
    Depositor( Account act ){ sacct = act; }
    public void run() {
        int i = 0;
        while ( true ) {     // the thread will never stop
            int x = (int) ( 10 * Math.random() );
            sacct.deposit( x );
            if ( i++ % 1000 == 0 )
                System.out.println(
                    "balance after deposits:  " + sacct.balance );
            try { sleep( 5 ); } catch( InterruptedException e ) {}
        }
    }
}
```

```
class Withdrawer extends Thread {
    private Account sacct;
    Withdrawer( Account act ) { sacct = act; }
    public void run() {
        int i = 0;
        while ( true ) {
            int x = (int) ( 10 * Math.random() );
            sacct.withdraw( x );
            if ( i++ % 1000 == 0 )
                System.out.println( "balance after withdrawals:  "
                                    + sacct.balance );
            try { sleep( 5 ); } catch( InterruptedException e ) {}
        }
    }
}
```

```
class MultiCustomerAccount {
    public static void main( String[] args ) {
        Account account = new Account();
        Depositor[] depositors = new Depositor[ 5 ];
        Withdrawer[] withdrawers = new Withdrawer[ 5 ];
        for ( int i=0; i < 5; i++ ) {
            depositors[ i ] = new Depositor( account );
            withdrawers[ i ] = new Withdrawer( account );
            depositors[ i ].start();
            withdrawers[ i ].start();
        } // the program runs indefinitely
    }
}
```

General Structure of synchronized

- If only one thread can execute a method, make the method synchronized. If only one thread can execute one of several methods, make all these methods synchronized.
- If several threads may modify a condition, the general structure is

```
synchronized method1 (...)
{
        while (condition false) {
            try {
             wait();
            } catch (InterrupedException
    exp) {}
        }
        execute some statements;
}
```

```
synchronized method2 (...)
{
        execute some
    statements;
        set condition to be true;
        notifyAll();
}
```

When a thread calls wait, it yields the key (release the lock).
After notifyAll, only one thread can enter either method1 or 2.

Thread class or Runnable interface

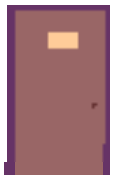
```
class HelloThread extends Thread {
        ...
        public void run() {
          ...
        }
    }
```

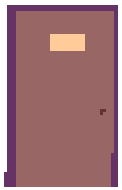
```
HelloThread t = new HelloThread(...);
...
t.start();
// do not call run directly
```

```
class HelloThread implements
    Runnable {
            ...
            public void run() {
                    ...
            }
    }
```

synchronized Methods

- All synchronized methods of the same object share the same lock ⇒ only one thread can acquire the lock
- Can one synchronized method calls another synchronized method? ⇒ Yes
- Can two threads calls the synchronized methods of **two** objects simultaneously? ⇒ Yes
- Can one thread calls one object's synchronized method that calls another object's synchronized method? ⇒ Yes
- Can a constructor be synchronized? ⇒ No





synchronized calls synchronized

```
class tclass extends Thread {
    synchronized void func1() {
     System.out.println("func1 start");
     func2();
     System.out.println("func1 end");
    }
    synchronized void func2() {
     System.out.println("func2 start");
     func3();
     System.out.println("func2 end");
    }
    synchronized void func3() {
     System.out.println("func3");
    }
    public void run() {
     func1();
    }
}
public class SyncCallSync {
    public static void main( String[]
     args ) {
     tclass xobj = new tclass();
     xobj.start();
    }
}
```

Interleaving synchronized

```
class Sclass
{
    static int syncount = 0;
    synchronized
     void func1() {
     syncount ++;
     long nextTime =
     System.currentTimeMillis() + 500;
     while (System.currentTimeMillis()
     < nextTime)
         { }
     func2();
    }
synchronized
      void func2() {
      syncount --;
     }
     synchronized public void print () {
      System.out.println("syncount = " +
      syncount);
     }
}
```

```
class Tclass extends Thread {
    Sclass sobj;
    public Tclass() {
     sobj = new Sclass();
    }
    public void run () {
     while (true) {
         sobj.func1();
         yield();
         sobj.print();
     }
    }
}
```

```
public class SyncMulti {
    public static void main( String[]
     args ) {
     int numThread = 100;
     Tclass [] tobjs = new
     Tclass[numThread];
     for (int i = 0; i < numThread; i
     ++)
         {
             tobjs[i] = new Tclass();
             tobjs[i].start();
         }
    }
}
```

synchronized calls another object's synchronized

```
class SClass {
    synchronized void func1() {
     System.out.println("S func1");
    }
}
class TClass extends Thread {
    SClass sobj;
    public TClass() {
     sobj = new SClass();
    }
    synchronized void func1() {
     System.out.println("T func1");
      func2();
       sobj.func1();
    }
    synchronized void func2() {
       System.out.println("T func2");
    }
    public void run() { func1();  }
}
public class SyncCallSync2 {
    public static void main( String[]
     args ) {
     TClass xobj = new TClass();
     xobj.start();
    }
}
```

```
//MultiCustomerAccount.cc
#include <qthread.h>
#include <cstdlib>
#include <iostream>
#include <ctime>
#include <QMutex>
#include <QWaitCondition>
using namespace std;
void keepBusy( double howLongInMillisec );
QMutex mutex;
QWaitCondition cond;
class Account : public QThread {
public:
    int balance;
    Account() { balance = 0; }
```

need modifications from the source code in the book

→ correspond to Java's synchronized

→ condition to wait at low balance

```
void deposit( int dep ) {
        mutex.lock();
        balance += dep;
        keepBusy( 1 );
        cond.wakeAll();
        mutex.unlock();
    }
    void withdraw( int draw ) {
        mutex.lock();
        while ( balance < draw ) {  cond.wait( &mutex );   }
        keepBusy( 1 );
        balance -= draw;
        mutex.unlock();
    }
    void run(){}
};
Account accnt;
```


sacct is a C library function. You need to rename this global / shared variable.

```
class Depositor : public QThread {
public:
    void run() {
        int i = 0;
        while ( true ) {
            int x = (int) ( rand() % 10 );
            accnt.deposit( x );
            if ( i++ % 100 == 0 )
                cerr << "balance after deposits: "
                     <<  sacctn.balance << endl;
            keepBusy( 1 );
        }
    }
};
```

```
class Withdrawer : public QThread {
public:
    void run() {
        int i = 0;
        while ( true ) {
            int x = (int) ( rand() % 10 );
            accnt.withdraw( x );
            if ( i++ % 100 == 0 )
                cerr << "balance after withdrawals:  "
                    << accnt.balance << endl;
            keepBusy( 1 );
        }
    }
};
```

```
int main()
{
    Depositor* depositors[5];
    Withdrawer* withdrawers[5];
    for ( int i=0; i < 5; i++ ) {
        depositors[ i ] = new Depositor();
        withdrawers[ i ] = new Withdrawer();
        depositors[ i ]->start();
        withdrawers[ i ]->start();
    }
    for ( int i=0; i < 5; i++ ) {
        depositors[ i ]->wait();
        withdrawers[ i ]->wait();
    }
}
```

main thread waits until the child threads complete, equivalent to join

```
void keepBusy( double howLongInMillisec ) {
    int ticksPerSec = CLOCKS_PER_SEC;
    int ticksPerMillisec = ticksPerSec / 1000;
    clock_t ct = clock();
    while ( clock() < ct + howLongInMillisec * ticksPerMillisec )
        ;
}
```

Qt creates Makefile for you:

qmake -project

qmake

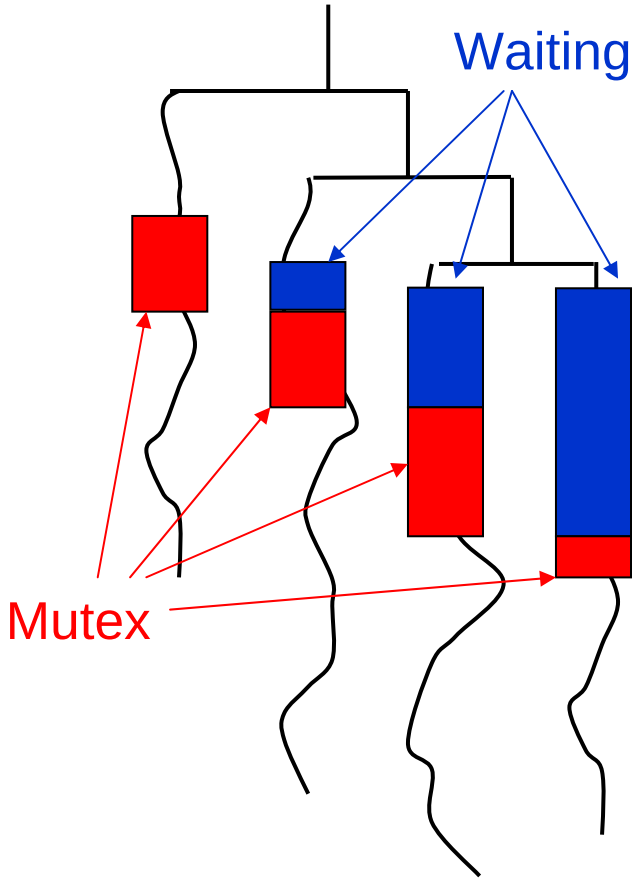
make

Compare Java and C++ (Qt) Threads

Java	C++
extends Thread or implements Runnable	public QThread
public void run()	public: void run()
thread1.start();	thread1 -> start();
synchronized	mutex.lock() … mutex.unlock()
try { wait (); } catch …	cond.wait(&mutex);

Serialization

- If all threads need to access one shared object often, the program essentially becomes a serial program.
- The program may actually run more slowly due to mutex and thread overhead.
- solution: reduce the "coupling" among threads.

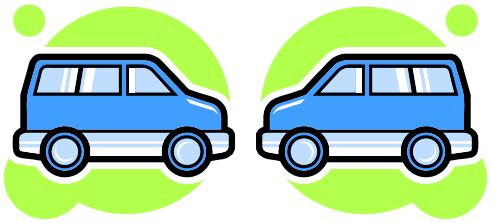


Improve Thread Efficiency

reduce the "coupling" among threads

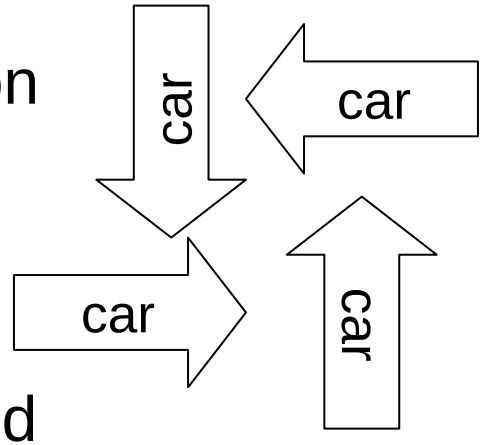
- do not share objects
- do not execute concurrently when objects are shared
- shrink the granularity of shared object. e.g. all bank accounts $\Rightarrow$ individual customer's account
- shrink critical sections $\Rightarrow$ only the operations that are related to the shared objects are in mutex
- coarse-grained synchronization (many statements within lock) $\Rightarrow$ serialize the threads
- fine-grained synchronization (few statements within lock) $\Rightarrow$ lock-release overhead

Deadlock



four necessary conditions for deadlock

- mutual exclusion: a car's current location cannot be occupied by another car
- hold and wait: a car must "hold" the current location while waiting
- no preemption: a car cannot be removed by a scheduler
- circular wait: a car can move if another car moves but another car is waiting for this car

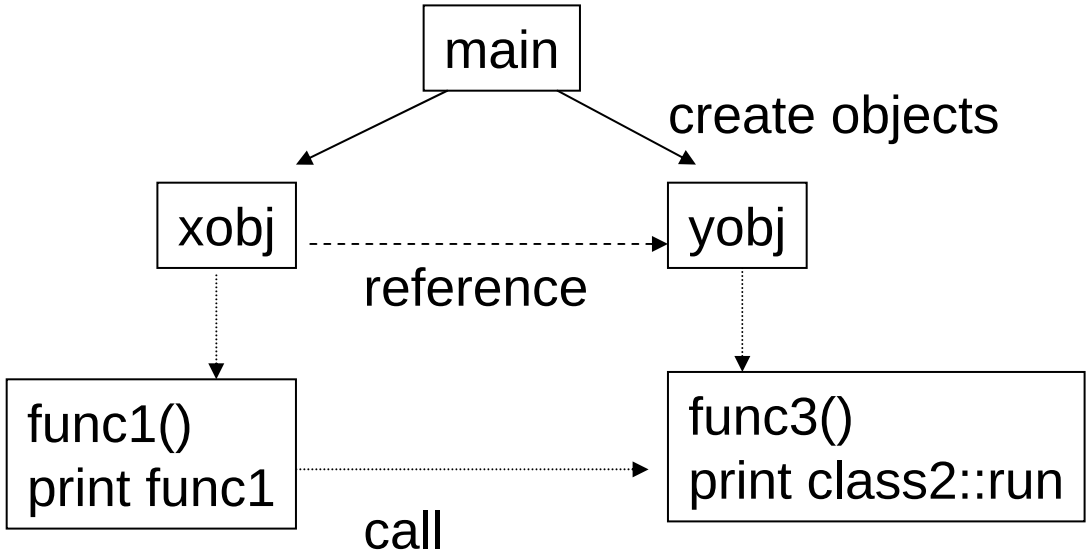


```
// deadlock?
class class1 extends Thread {
    synchronized void func1() {
        System.out.println("func1");
        func2();
    }
    synchronized void func2() { System.out.println("func2"); }
    public void run() { func1();  }
}
public class example {
    public static void main( String[] args ) {
        class1 xobj = new class1();
        xobj.start();
    }
}
```

No, only one thread

func1
func2
(terminate)

```
class class1 extends Thread {
    class2 obj2;
    public void assignY(class2 inobj) { obj2 = inobj; }
    public void func1() { System.out.println("func1");  obj2.func3();  }
    public void func2() {  System.out.println("\tfunc2");  }
    public void run() { func1(); }
}
class class2  extends Thread {
    void func3() {  System.out.println("func3"); }
    public void run() { System.out.println("class2::run"); }
}
public class example {
    public static void main( String[] args ) {
        class1 xobj = new class1();            class2 yobj = new class2();
        xobj.assignY(yobj);
        xobj.start();                          yobj.start();
    }
}
```



No deadlock, no lock, no circular call

func1
func3
class2::run
(terminate)

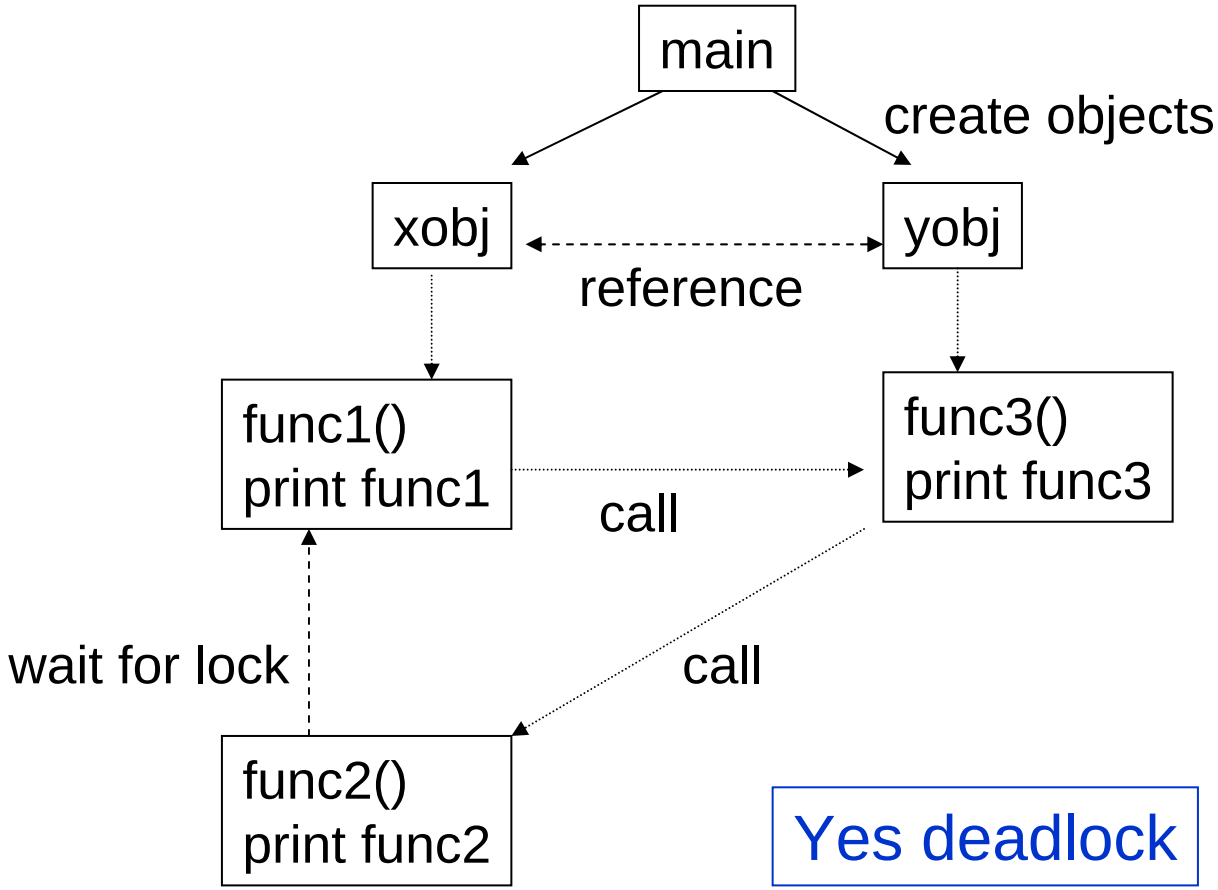
```
class class1 extends Thread {
    class2 obj2;
    int cnt;
    class1 () { cnt = 0; }
    public void assignY(class2 inobj) { obj2 = inobj; }
    synchronized void func1() {
        System.out.print("func1 " + cnt + "    ");        cnt ++;
        try { sleep(1); } catch( InterruptedException e ) {}
        obj2.func3();
    }
    synchronized void func2() {
        System.out.println("\tfunc2 " + cnt);
        try { sleep(1); } catch( InterruptedException e ) {}     cnt ++;
        func1();
    }
```

```
 public void run() {  while (true)   { func1(); }   }
}
class class2  extends Thread {
    class1 obj1;
    int cnt;
    class2 () { cnt = 0; }
    public void assignX(class1 inobj) { obj1 = inobj; }
    synchronized void func3() {
        System.out.print("func3 " + cnt + "    ");        cnt ++;
        try { sleep(1); } catch( InterruptedException e ) {}
        obj1.func2();
    }
    public void run() {  while (true)  { func3(); }   }
}
```

```
public class example {
    public static void main( String[] args ) {
        class1 xobj = new class1();
        class2 yobj = new class2();
        xobj.assignY(yobj);
        yobj.assignX(xobj);
        xobj.start();
        yobj.start();
    }
}
```

func1 0 func3 0

// Is this a deadlock?



```
synchronized void func2() {
        System.out.println("\tfunc2 " + cnt);
        try { sleep(1); } catch( InterruptedException e ) {}     cnt ++;
        func1();
    }
```

still deadlock

```
synchronized void func1() {
        System.out.print("func1 " + cnt + "    ");        cnt ++;
        try { sleep(1); } catch( InterruptedException e ) {}
        obj2.func3();
    }
```

no deadlock, no circular waiting

Multi-Thread = High Performance?

- multi-thread $\neq$ high performance (faster)
- If a program is IO-bounded, multi-thread (or multi-core) does not really help.
- If a program has a large portion of sequential code. Amdahl's Law:
  - If a program has x (%) parallel code, 1-x sequential code
  - the speedup of using n threads (no sync) is $\frac{1}{1-x+\frac{x}{n}}$
  - if x = 0.9 and n $\rightarrow\infty$, speedup = 10
- Finding sufficient parallelism (make x closer to 1) is difficulty.

When to Use Multi-Threads?

- To meet parallelism requirement: multiple users access their bank accounts on-line.
- To provide quick response to a user: graphical user interface responds to mouse click, even though computation is still being performed
- To expedite computation: simulate an airplane's wings
- To provide fault tolerance: whether different threads (with different algorithms) reach the same answer
- .... many more ...

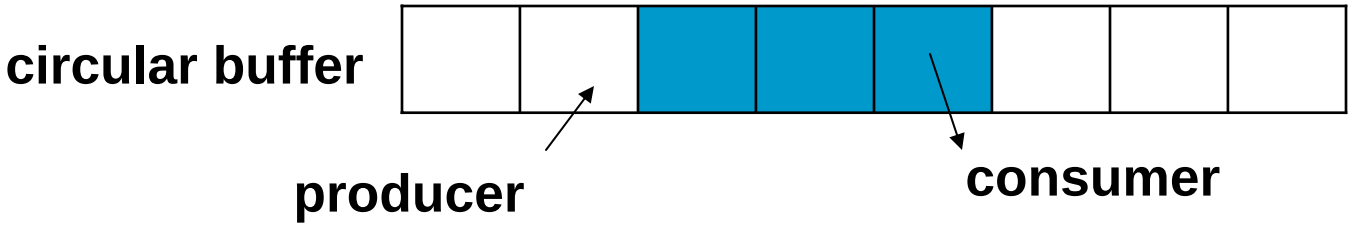
ECE 462
Object-Oriented Programming using C++ and Java

Lecture 16

Yung-Hsiang Lu

yunglu@purdue.edu

Producer-Consumer




```
Producer:
while (true)
{
    if the buffer is not full
    {
        add an element into the
        buffer;
    }
}
```

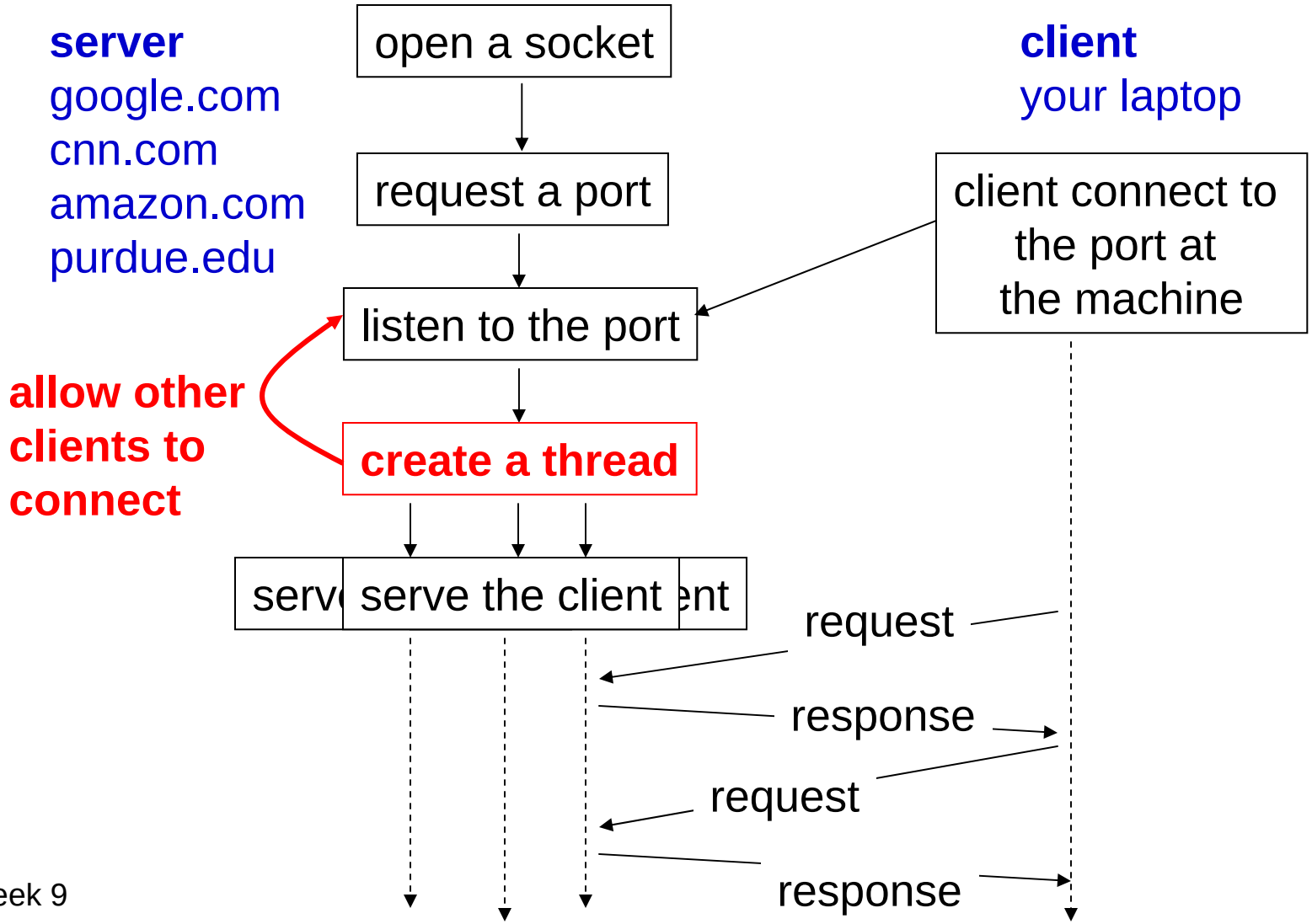
```
Consumer:
while (true)
{
    if the buffer is not empty
    {
        retrieve an element from
        the buffer
    }
}
```

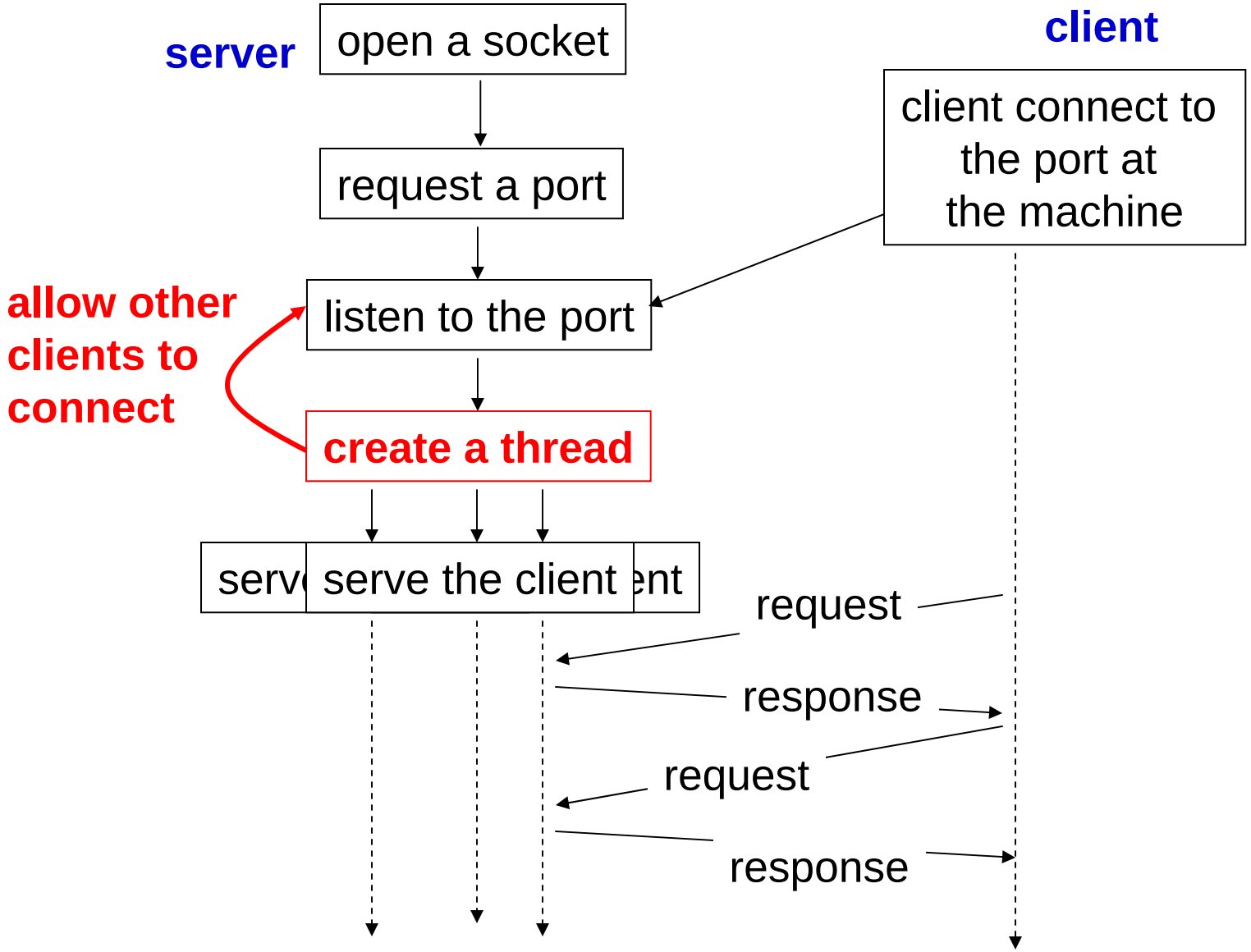
two threads sharing the buffer

Lab 9 Convert a C++ Multithread Program to Java

http://doc.trolltech.com/4.3/threads-mandelbrot.html

Networking, Client-Server Model





Internet port definitions:
http://www.spirit.com/Resources/ports.html

C++ Client using Qt

```
// Modified for Qt 4.3
//ClientSocket.h
#ifndef CLIENTSOCKET_H
#define CLIENTSOCKET_H
#include <QTcpSocket>
#include <string>
using namespace std;
class ClientSocket : public
    QTcpSocket {
  Q_OBJECT
  string wwwName;
  QTcpSocket* socket;
public:
  ClientSocket( string wwwName );
  string constructHttpRequest();
  void socketClosed();
  ~ClientSocket();
public slots:
  void reportConnected();
  void reportHostFound();
  void getWebPage();
  void socketConnectionClosed();
  void reportError( int );
};
#endif
```

```
// ClientSocket.cc
#include "ClientSocket.h"
#include <QApplication>
#include <QTcpSocket>
#include <string>
#include <iostream>
using namespace std;
ClientSocket::ClientSocket( string
    siteName ) : QTcpSocket( 0  )
{
  wwwName = siteName;
  socket = new QTcpSocket( );
  connect( socket, SIGNAL(
    connected() ),
      this,   SLOT( reportConnected()
    ) );
  connect( socket, SIGNAL(
    hostFound() ),
      this,   SLOT( reportHostFound()
    ) );
  connect( socket, SIGNAL(
    readyRead() ),
      this,   SLOT( getWebPage() ) );
  connect( socket, SIGNAL(
    connectionClosed() ),
      this,   SLOT(
    socketConnectionClosed() ) );
  connect( socket, SIGNAL( error( int )
    ),   this,   SLOT( reportError( int ) )
    );
  QString qstr( wwwName.c_str() );
  socket->connectToHost( qstr, 80 );
    // asynchronous call
}
```

```
ClientSocket::~ClientSocket() {
    delete socket; }
string
    ClientSocket::constructHttpReque
    st( ) {
  string hostHeader = "Host: " +
    wwwName;
  string urlString( hostHeader );
  string httpRequestString = "GET /
    HTTP/1.0\r\n" +
   urlString + "\r\n" + "\r\n";
  return httpRequestString;
}
void ClientSocket::reportHostFound()
 {
  cout << "host found" << endl;
}
```

```
void ClientSocket::reportConnected()
    {
  cout << "connection established" <<
    endl;
  string httpRequest =
    constructHttpRequest();
  int len = httpRequest.size();
  socket->write( httpRequest.c_str(),
    len );
}
void ClientSocket::getWebPage()
{
  int howManyBytes = socket-
    >bytesAvailable();
  char data[howManyBytes];
  socket->read( data, howManyBytes);
  cout << data;
  cout.flush();
}
```

```
void
    ClientSocket::socketConnectionCl
    osed() {
   if ( socket->state() ==
    QAbstractSocket::ClosingState ) {
    // delayed close
   connect( socket, SIGNAL(
    delayedCloseFinished() ),
         this,   SLOT( socketClosed() )
    );
  } else {
   // The socket is really closed
   socketClosed();
  }
}
```

```
void ClientSocket::reportError( int e ) {
  cout << "error report from
     connectToHost" << endl;
  cout << "error id: " << e << endl;
}
void ClientSocket::socketClosed() {
  cout << "Connection closed" <<
     endl;
  exit( 0 );
}
int main( int argc, char* argv[] )
{
  QApplication app( argc, argv );
  ClientSocket* sock = new
     ClientSocket( argv[1] );
  return app.exec();
}
```

C++ Server using Qt

```
// ChatServer.h
// Modified for Qt4.3
// similar structure as the Java version
// This program has memory leak.
// Do not worry about it now.
#ifndef CHATSERVER_H
#define CHATSERVER_H
#include <QtNetwork>
#include <QString>
#include <QThread>
#include <QApplication>
#include <vector>
using namespace std;
```

```
class ClientHandler : public QObject
{
  Q_OBJECT
private:
  QString chatName; // assume
     names of the clients are unique
  QTcpSocket* handlerSocket;
  QTextStream* os;
  static QList<QString*> * chatStore;
  void broadcastClient(QString);
public:
  ClientHandler( QTcpSocket* sock);
  virtual ~ClientHandler();
```

```
  static vector<ClientHandler *>
     clientVector;
private slots:
  void readFromClient();
};
class ChatServer: public QObject
{
  Q_OBJECT
private:
  QTcpServer * server;
public:
  ChatServer( );
  ChatServer(const ChatServer & cs);
  virtual ~ChatServer();
public slots:
  void connectNewClient( );
};
#endif
```

```
//ChatServer.cc
#include "ChatServer.h"
#include <iostream>
using namespace std;
// initialize static members
QList<QString*> *
     ClientHandler::chatStore = new
     QList<QString*>;
vector<ClientHandler *>
     ClientHandler::clientVector;
ChatServer::ChatServer( )
{
  server = new QTcpServer();
  if (! server->listen())
    {
      qWarning( "Failed to register the
      server port" );
      exit( 1 );
    }
```

```
  cout << "Server port " << server-
     >serverPort() << endl;
  connect(server,
     SIGNAL(newConnection()), this,
     SLOT(connectNewClient()));
}
void ChatServer::connectNewClient()
{
  QTcpSocket* socket = server-
     >nextPendingConnection();
  // socket->setSocket( socketFD );
  ClientHandler* clh = new
     ClientHandler( socket );
     ClientHandler::clientVector.push_
     back(clh);
  cout << "A new client connected "
     << endl;
}
```

```
ChatServer::~ChatServer(){}
ClientHandler::ClientHandler(
     QTcpSocket* socket)
  : chatName(""),
    handlerSocket( socket )
{
  os = new QTextStream(
     handlerSocket );
  (*os) << "Welcome to a chat room
     powered by C++\n";
  (*os) << ">>>>    Enter 'bye' to exit
     <<<\n";
  (*os) << "Enter chat name: ";
  os -> flush();
  connect( handlerSocket, SIGNAL(
     readyRead() ),
        this, SLOT( readFromClient() ) );
}
```

```
ClientHandler::~ClientHandler(){}
void
    ClientHandler::broadcastClient(QS
    tring message)
{
  ClientHandler * handler;
  for (unsigned int index = 0; index <
    clientVector.size(); index ++)
   {
    handler = clientVector[index];
    *(handler-> os) << message;
    (handler-> os) -> flush();
   }
}
void ClientHandler::readFromClient()
    {
  QTcpSocket* sock = (QTcpSocket*)
    sender();
while ( sock->canReadLine() ) {
  QString qstr = sock->readLine();
  qstr = qstr.trimmed(); // remove
   white space
  if ( chatName == "" )
   {
   chatName = qstr;
   QString outgoing = "\nMessage
   from chat server: " +
     chatName + " signed in\n";
   broadcastClient(outgoing);
   }
  else if ( qstr == "bye" )
   {
```

```
QString outgoing = "\nMessage from
    the chat server: " +
      chatName + " signed off\n";
    broadcastClient(outgoing);
    handlerSocket->close();
    handlerSocket = 0;
    }
   else
    {
    QString outgoing = "\n" +
    chatName + ": " + qstr + "\n";
    broadcastClient(outgoing);
    }
   // A chatter's terminal always
  // shows his/her own name at
  // beginning of a new line.
    for (unsigned int index = 0; index <
     clientVector.size(); index ++)
     {
     ClientHandler * handler =
     clientVector[index];
     *(handler-> os) << (handler-
     >chatName + ": ");
     (handler-> os) -> flush();
     }
  }
}
int main( int argc, char* argv[] )
{
  QApplication app( argc, argv );
  ChatServer* server = new
     ChatServer( );
  return app.exec();
}
```

We have finished the explanation of the C++ and Java languages.

In future lectures, we will discuss programming techniques and object-oriented designs.