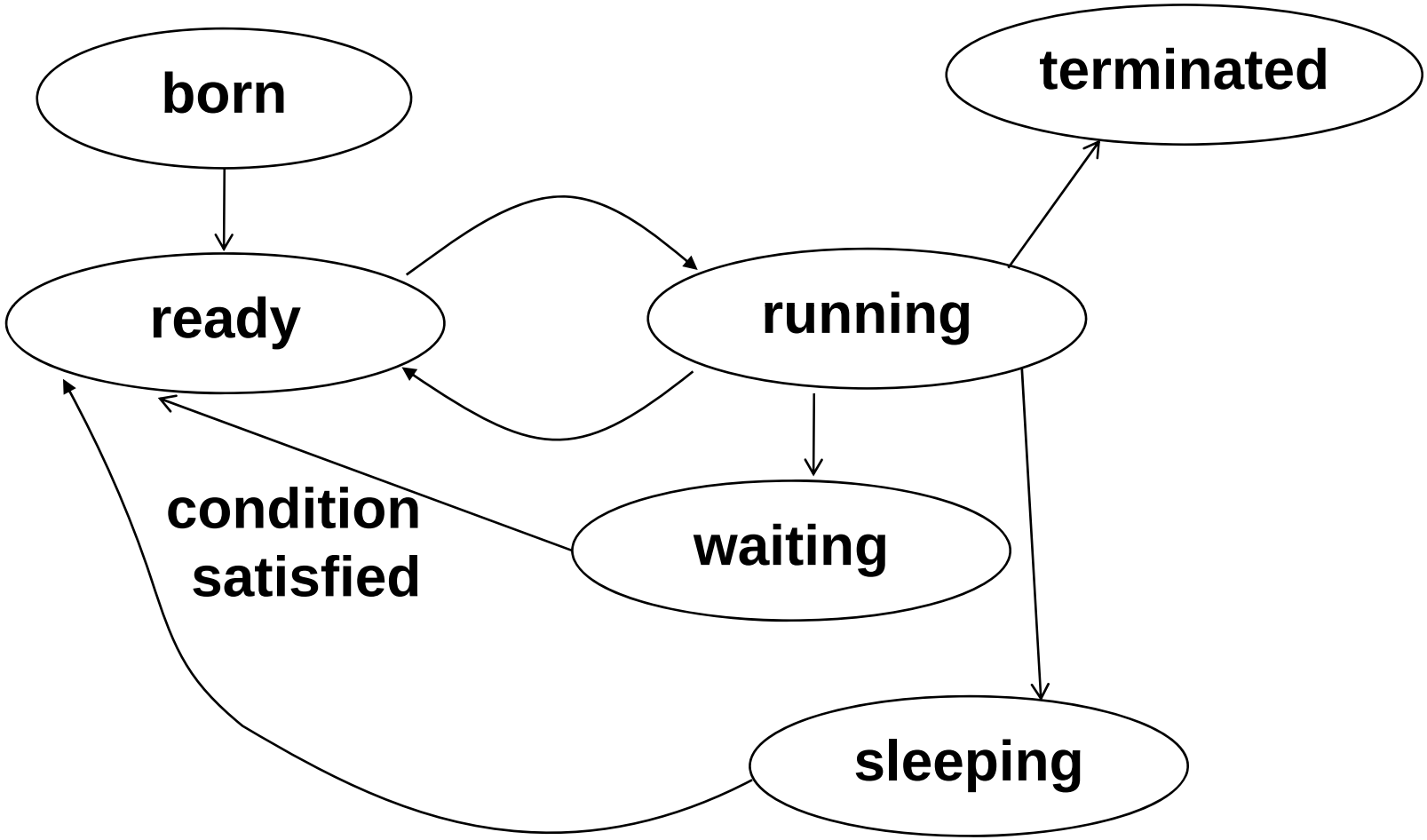
ECE 462
Object-Oriented Programming using C++ and Java

Scheduling and Critical Section

Yung-Hsiang Lu
yunglu@purdue.edu

Thread States

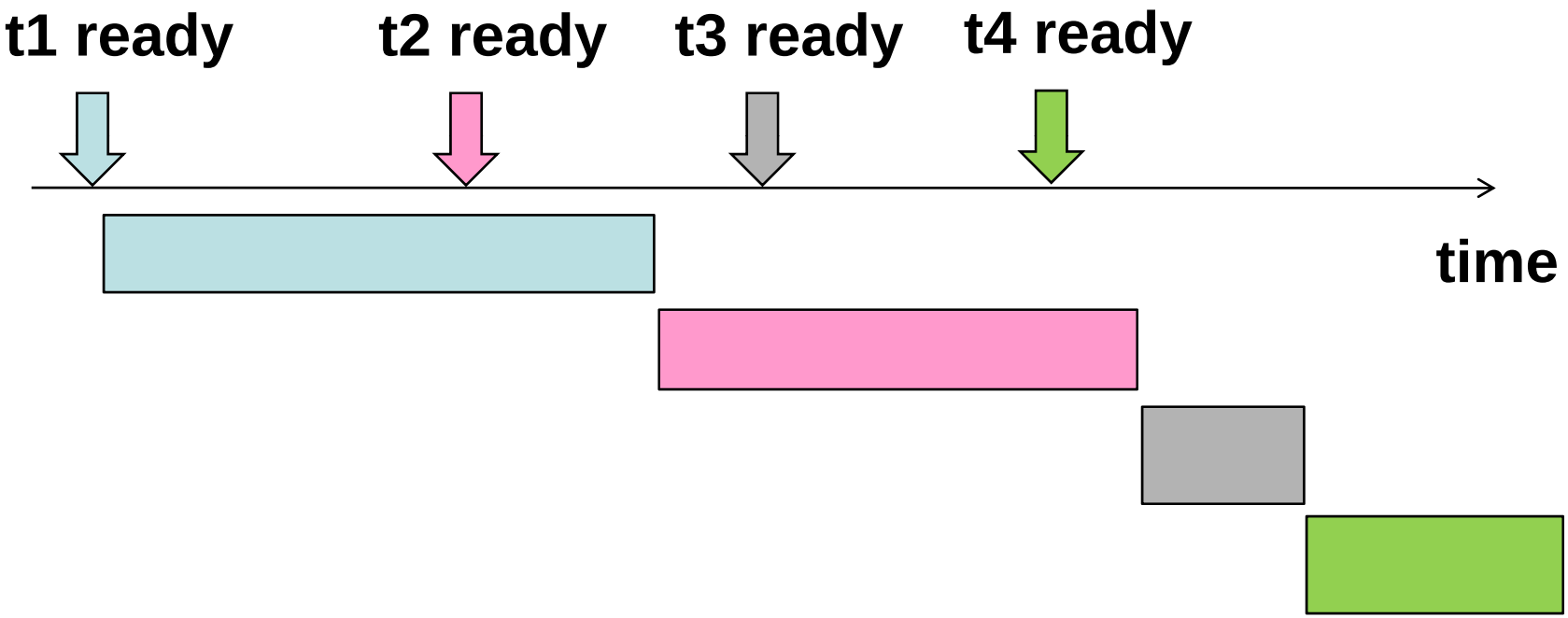


Scheduling

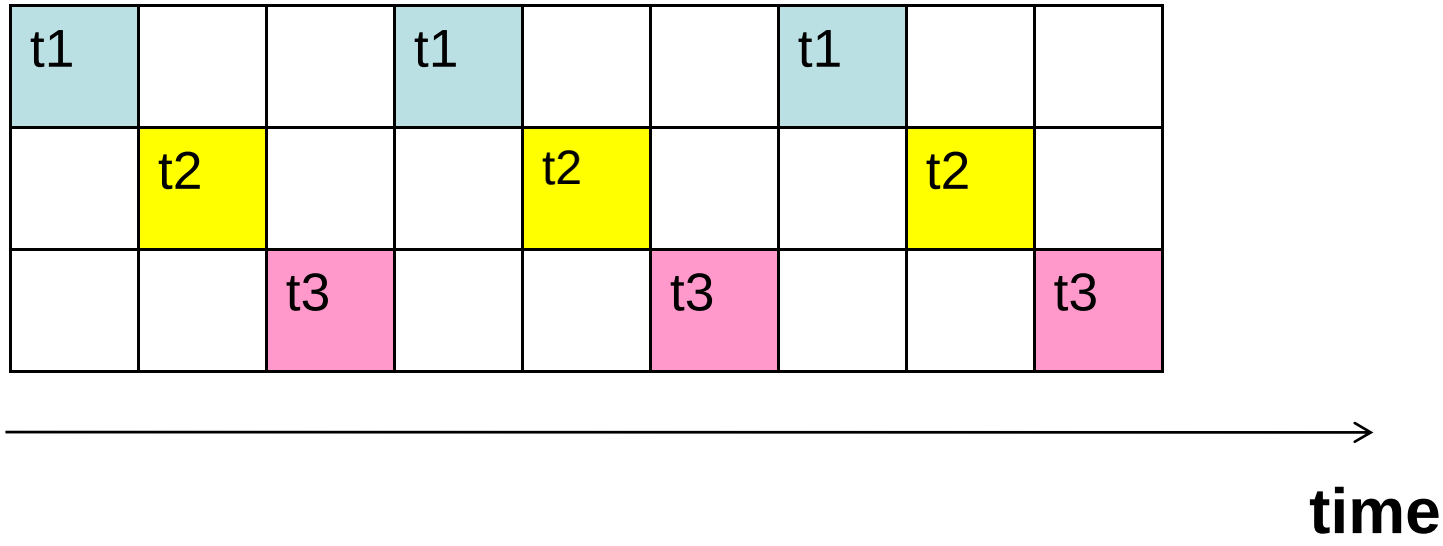
Scheduling
(select the thread to use a processor)

- If there are more ready threads than the number of processors, the computer's scheduling algorithm decides which threads to run next.
- scheduling criteria
  - utilization
  - throughput, turnaround time, and response time
  - waiting time
  - fairness
  - priority
  - deadline

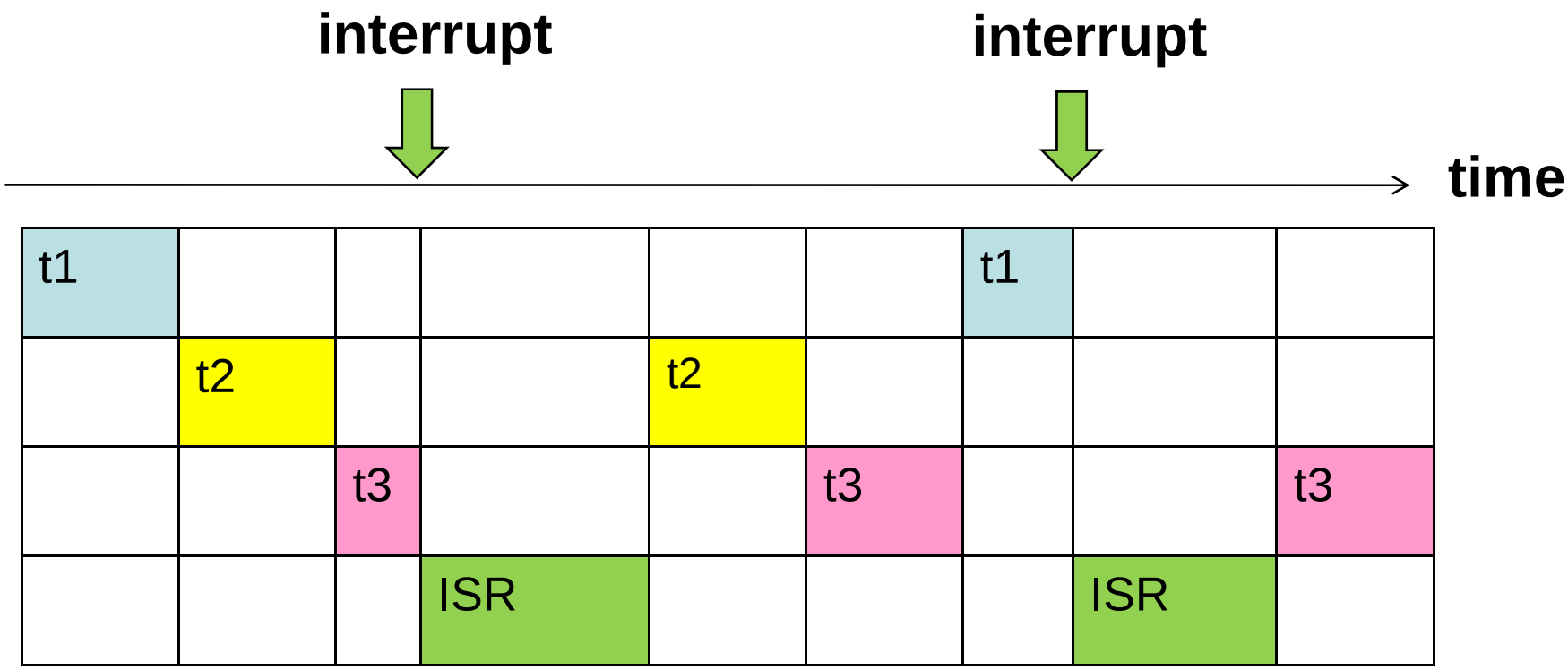
Scheduling Algorithm
First Come First Serve



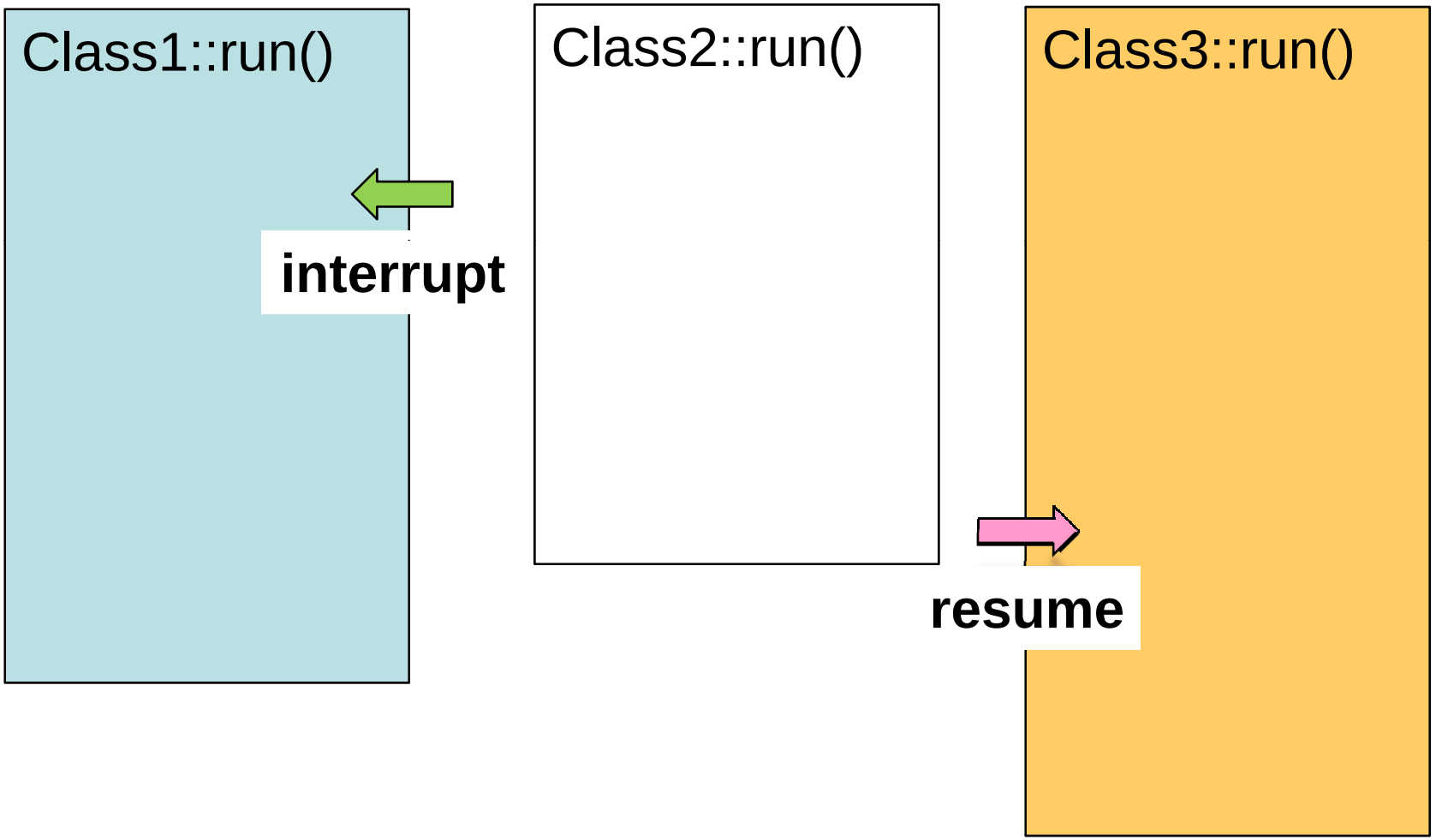
Round Robin + Time Slice



Scheduling + Interrupts



Thread Interleaving



Shared Data

int d = 50; // global variable

```
Class1::run()
{
    int x;
    x = d;
    x = x + 10;
    d = x;
}
```

```
Class2::run()
{
    int x;
    x = d;
    x = x + 5;
    d = x;
}
```

equivalent to
d += 5;

Interleaving and Shared Data

```
Class1::run()
{
    int x;
    x = d;

    x = x + 10;
    d = x;
}
```

interrupted

```
Class2::run()
{
    int x;
    x = d;
    x = x + 5;
    d = x;
}
```

```
Class1::run()
{
    int x;
    x = d;

    x = x + 10;
    d = x;
}
```

```
Class2::run()
{
    int x;
    x = d;

    x = x + 5;
    d = x;
}
```

interrupted

Conditions for Wrong Results

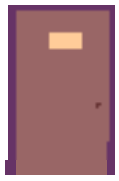
- multiple threads
- shared data and at least one thread modifies the data
- operations with multiple steps
- interrupts may occur between the steps
- another thread may execute while one thread is suspended

Atomic Operation

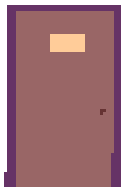
- An operation is "atomic" if it must execute to completion once the operation starts.
- In Java, only 32-bit (or smaller) assignments are atomic.
- Even x += y; is **not** atomic.
- The code (some statements in **run**) to modify a shared variable or object has to be atomic.

Mutual Exclusion (Mutex)

- Used 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.

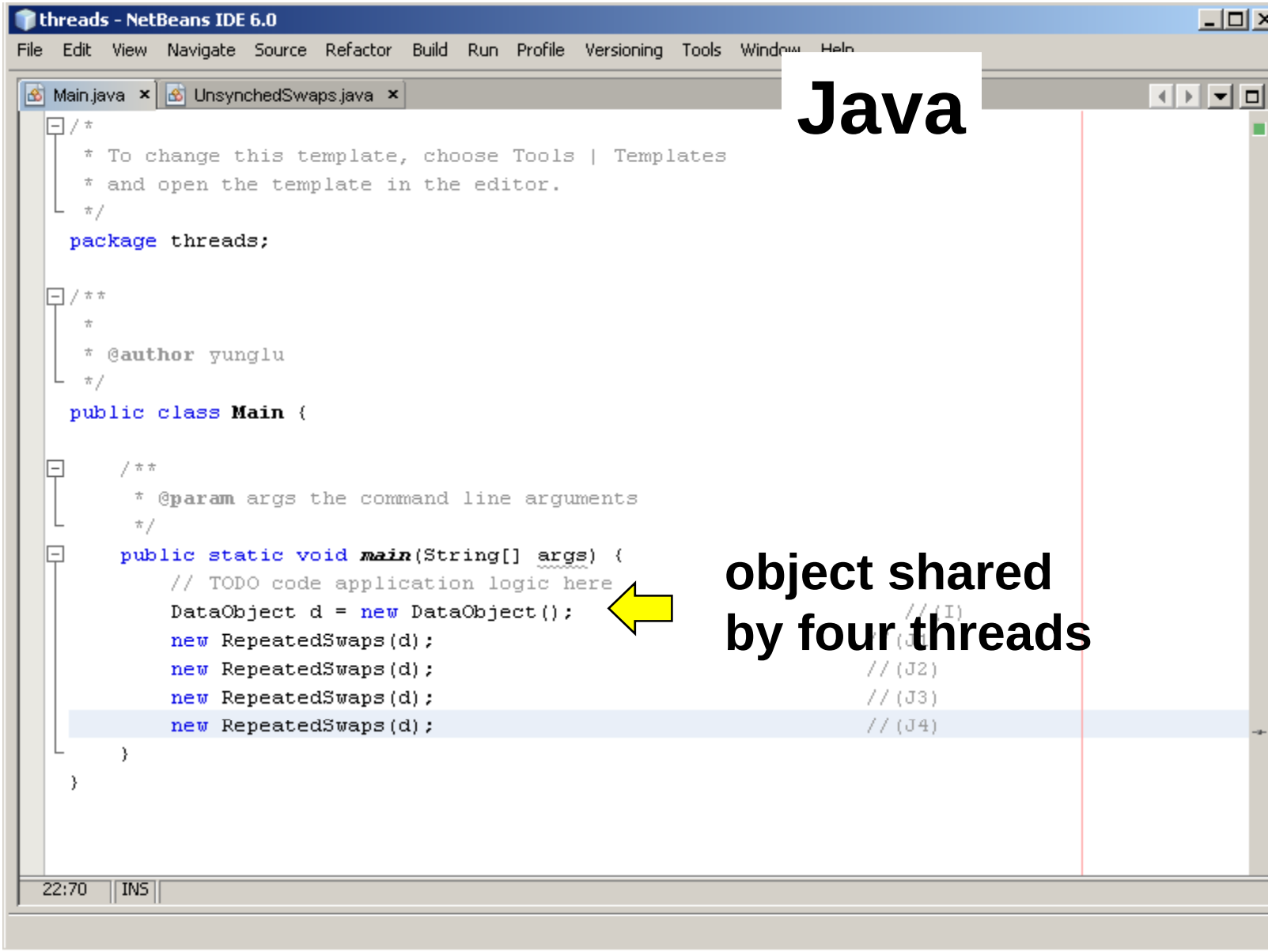


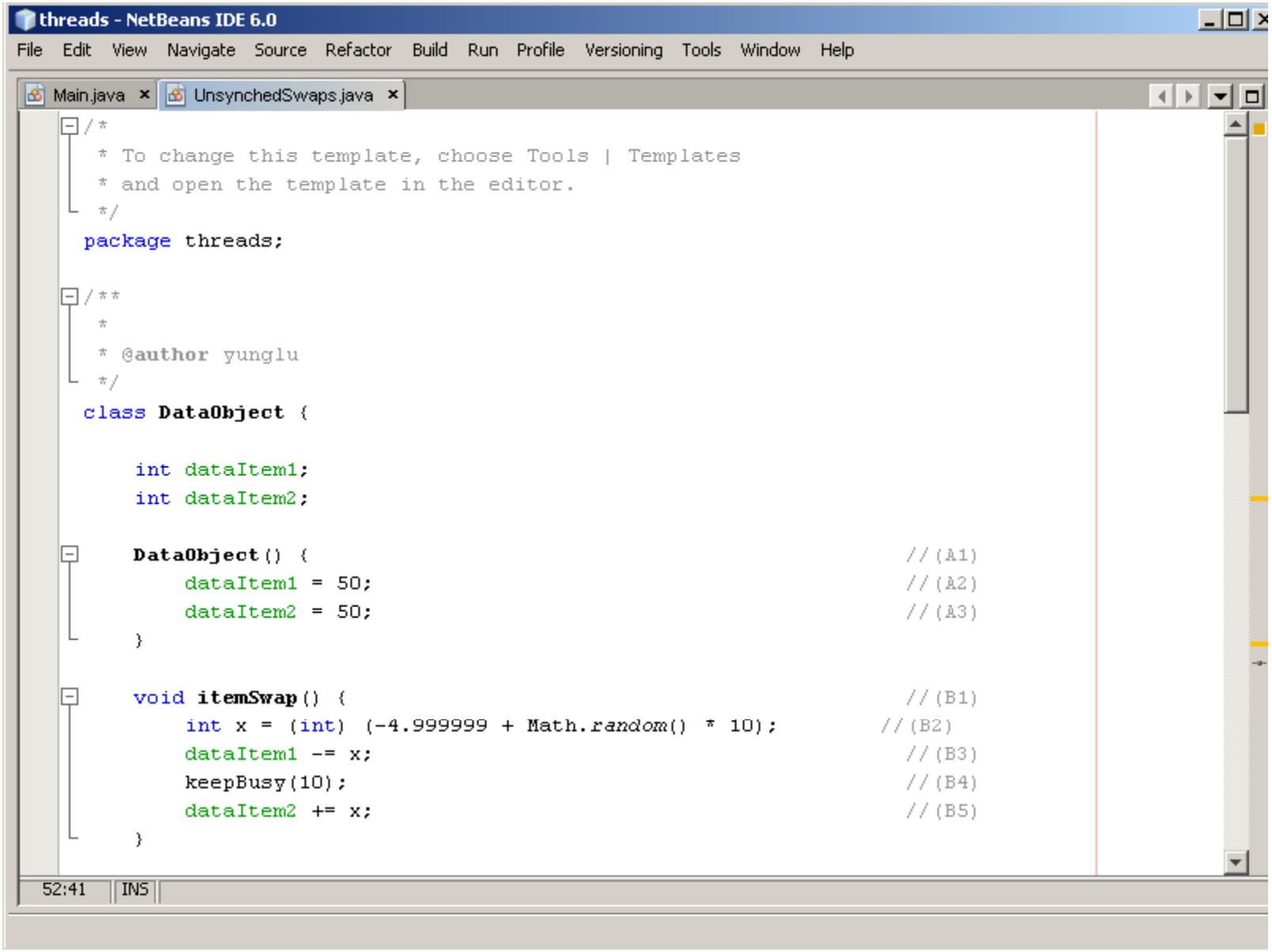
ECE 462
Object-Oriented Programming using C++ and Java

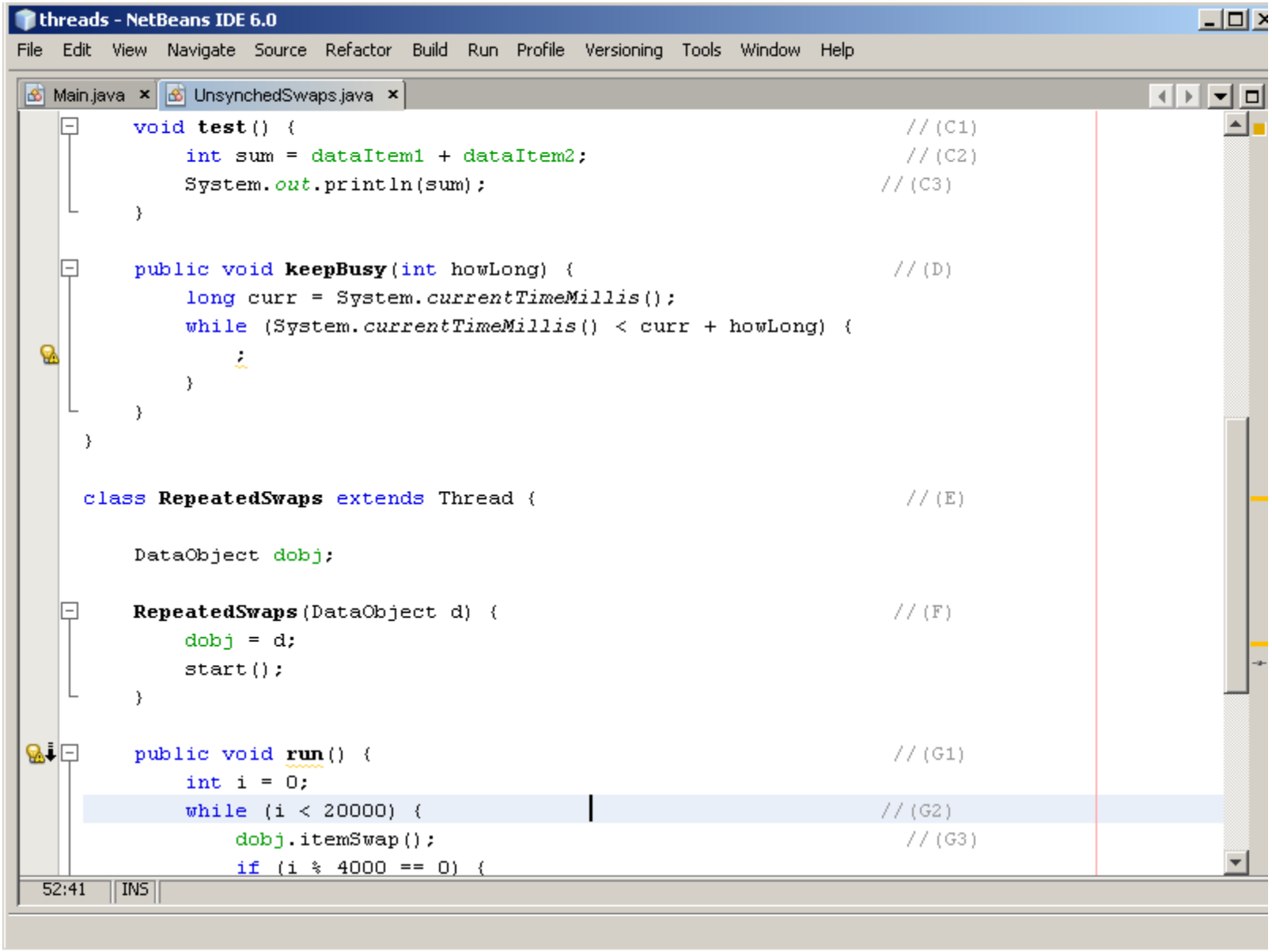
Synchronization

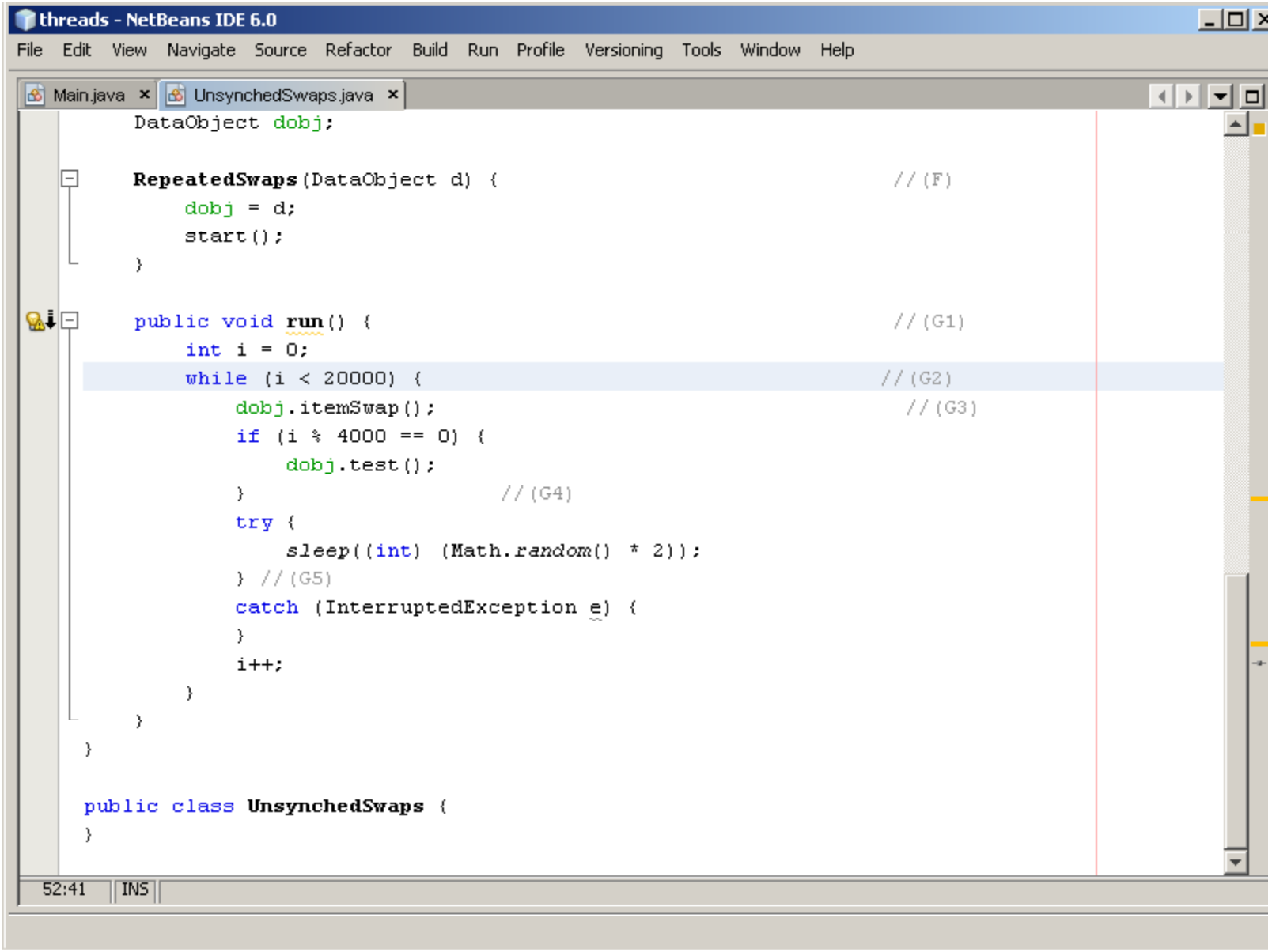
Yung-Hsiang Lu
yunglu@purdue.edu

Threads Sharing Objects


threads - NetBeans IDE 6.0
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Main.java
UnsynchedSwaps.java
/*
* To change this template, choose Tools | Templates
* and open the template in the editor.
*/
package threads;
/**
*
* @author yunglu
*/
public class Main {
/**
* @param args the command line arguments
*/
public static void main(String[] args) {
// TODO code application logic here
DataObject d = new DataObject();
new RepeatedSwaps(d);
new RepeatedSwaps(d);
new RepeatedSwaps(d);
new RepeatedSwaps(d);
}
}
//(I)
//(J2)
//(J3)
//(J4)
22:70
INS
Java
object shared
by four threads


threads - NetBeans IDE 6.0
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Main.java
UnsynchedSwaps.java
/*
* To change this template, choose Tools | Templates
* and open the template in the editor.
*/
package threads;
/**
*
* @author yunglu
*/
class DataObject {
int dataItem1;
int dataItem2;
DataObject() { //(A1)
dataItem1 = 50; //(A2)
dataItem2 = 50; //(A3)
}
void itemSwap() { //(B1)
int x = (int) (-4.999999 + Math.random() * 10); //(B2)
dataItem1 -= x; //(B3)
keepBusy(10); //(B4)
dataItem2 += x; //(B5)
}
52:41
INS


threads - NetBeans IDE 6.0
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Main.java
UnsynchedSwaps.java
void test() { //(C1)
int sum = dataItem1 + dataItem2; //(C2)
System.out.println(sum); //(C3)
}
public void keepBusy(int howLong) { //(D)
long curr = System.currentTimeMillis();
while (System.currentTimeMillis() < curr + howLong) {
;
}
}
}
class RepeatedSwaps extends Thread { //(E)
DataObject dobj;
RepeatedSwaps(DataObject d) { //(F)
dobj = d;
start();
}
public void run() { //(G1)
int i = 0;
while (i < 20000) { //(G2)
dobj.itemSwap(); //(G3)
if (i % 4000 == 0) {
52:41
INS

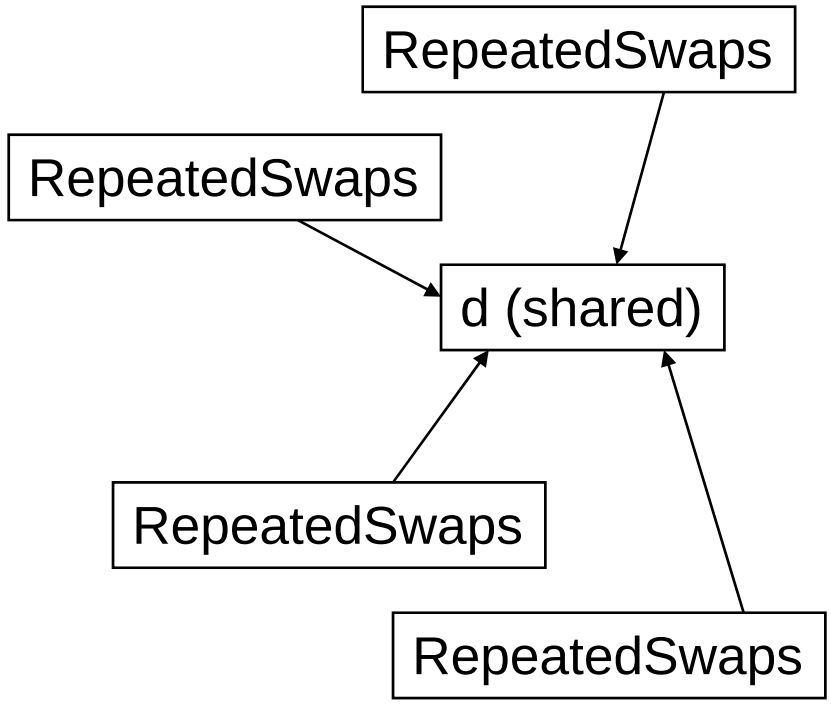

threads - NetBeans IDE 6.0
File Edit View Navigate Source Refactor Build Run Profile Versioning Tools Window Help
Main.java
UnsynchedSwaps.java
DataObject dobj;
RepeatedSwaps(DataObject d) { //(F)
dobj = d;
start();
}
public void run() { //(G1)
int i = 0;
while (i < 20000) { //(G2)
dobj.itemSwap(); //(G3)
if (i % 4000 == 0) {
dobj.test();
} //(G4)
try {
sleep((int) (Math.random() * 2));
} //(G5)
catch (InterruptedException e) {
}
i++;
}
}
}
public class UnsynchedSwaps {
}
52:41
INS

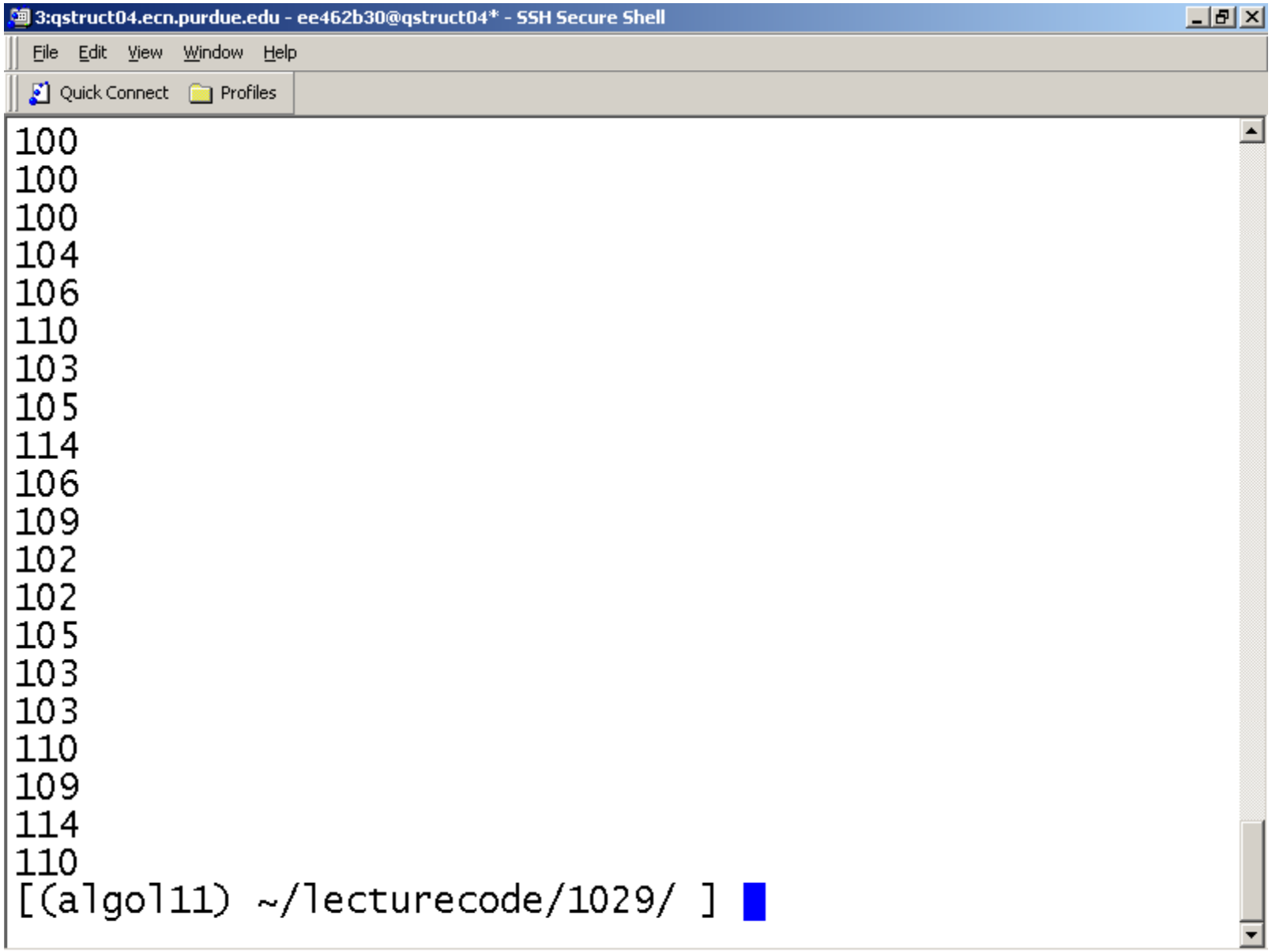
qstruct04.ecn.purdue.edu - ee462b30@qstruct04 - SSH Secure Shell

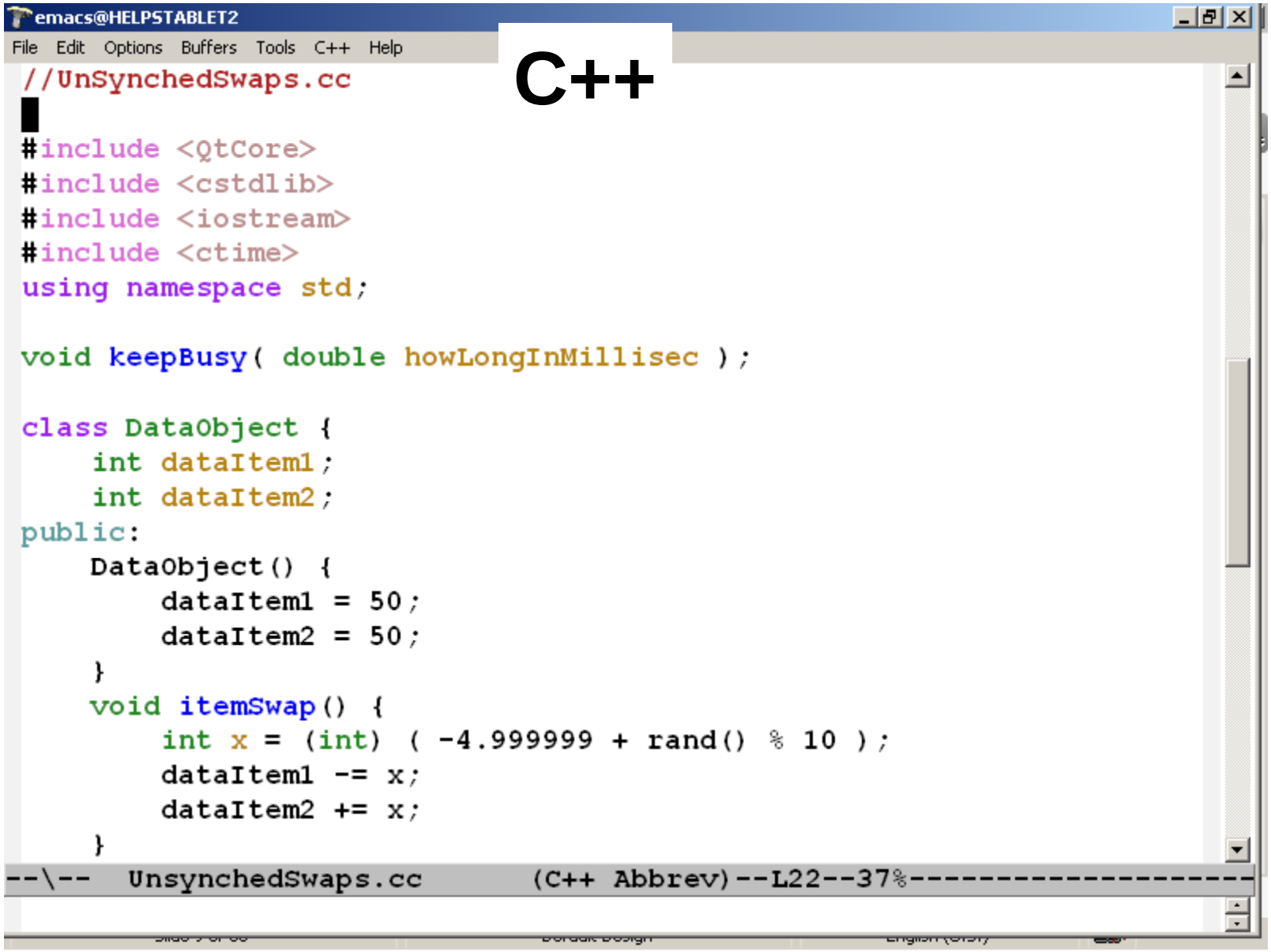
File Edit View Window Help

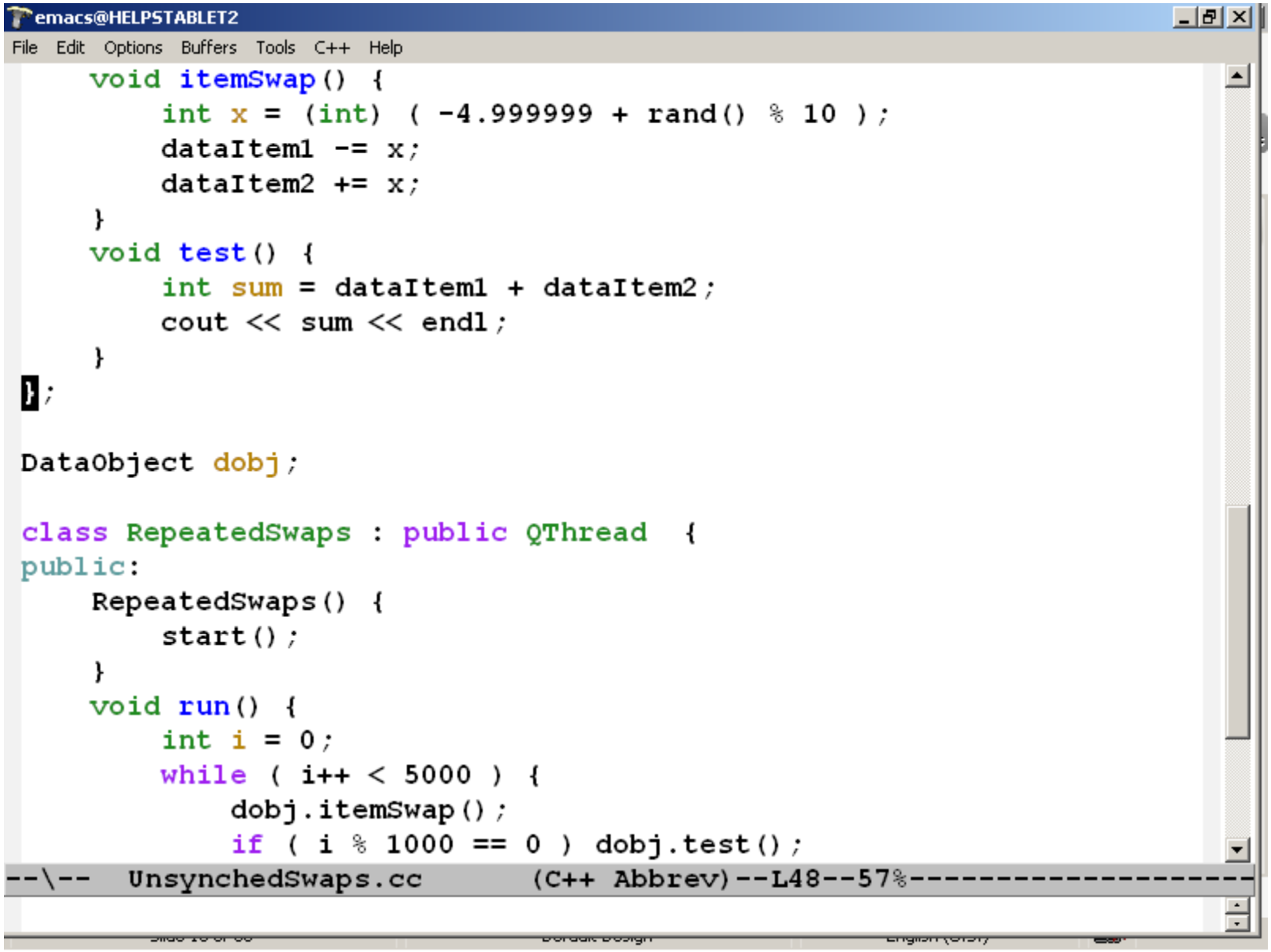
Quick Connect Profiles

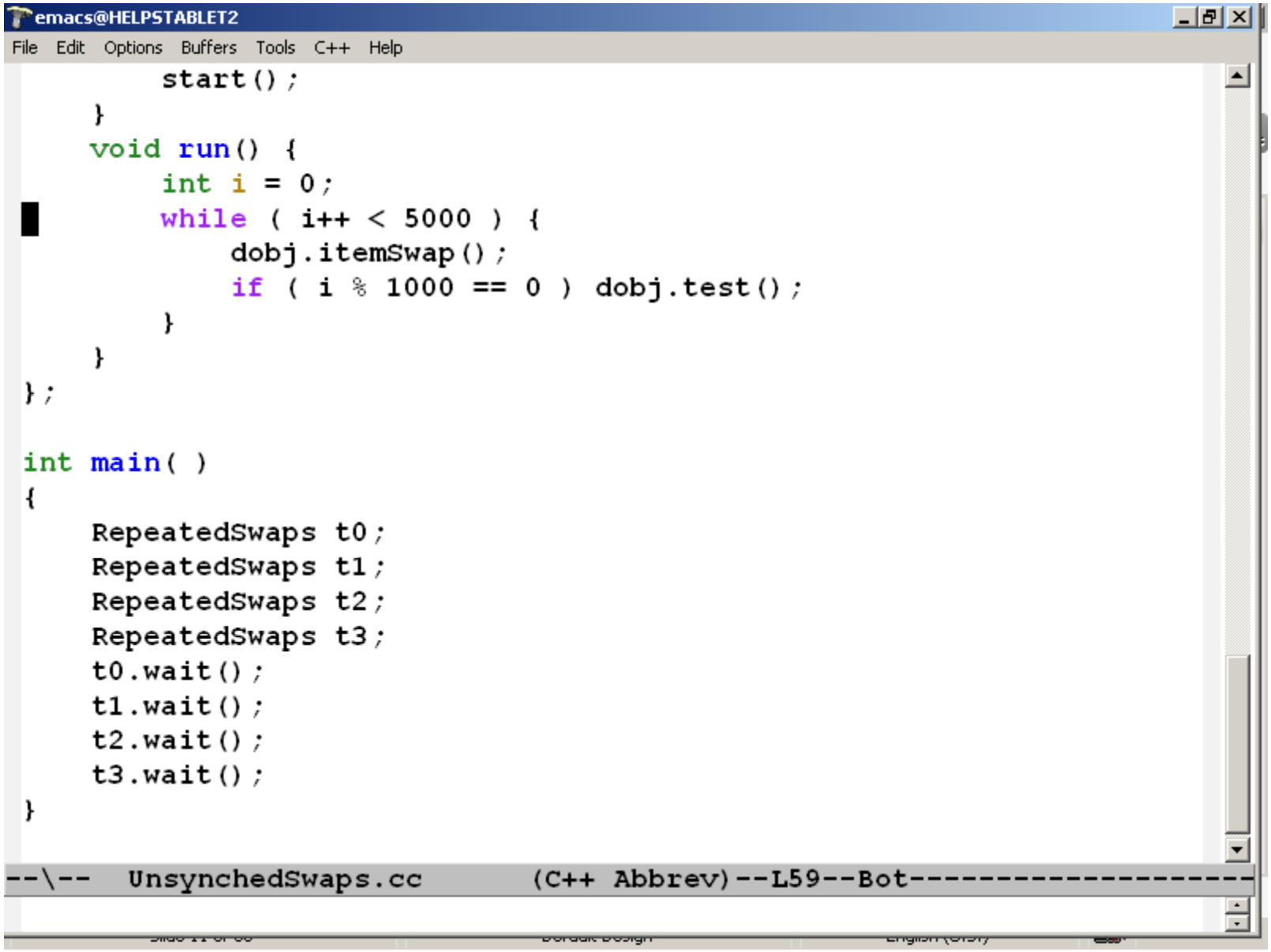
```
[(qstruct04) ~/lecturecode/1029/ ] java UnsynchedSwaps
94
100
100
102
134
129
134
137
155
165
162
165
```

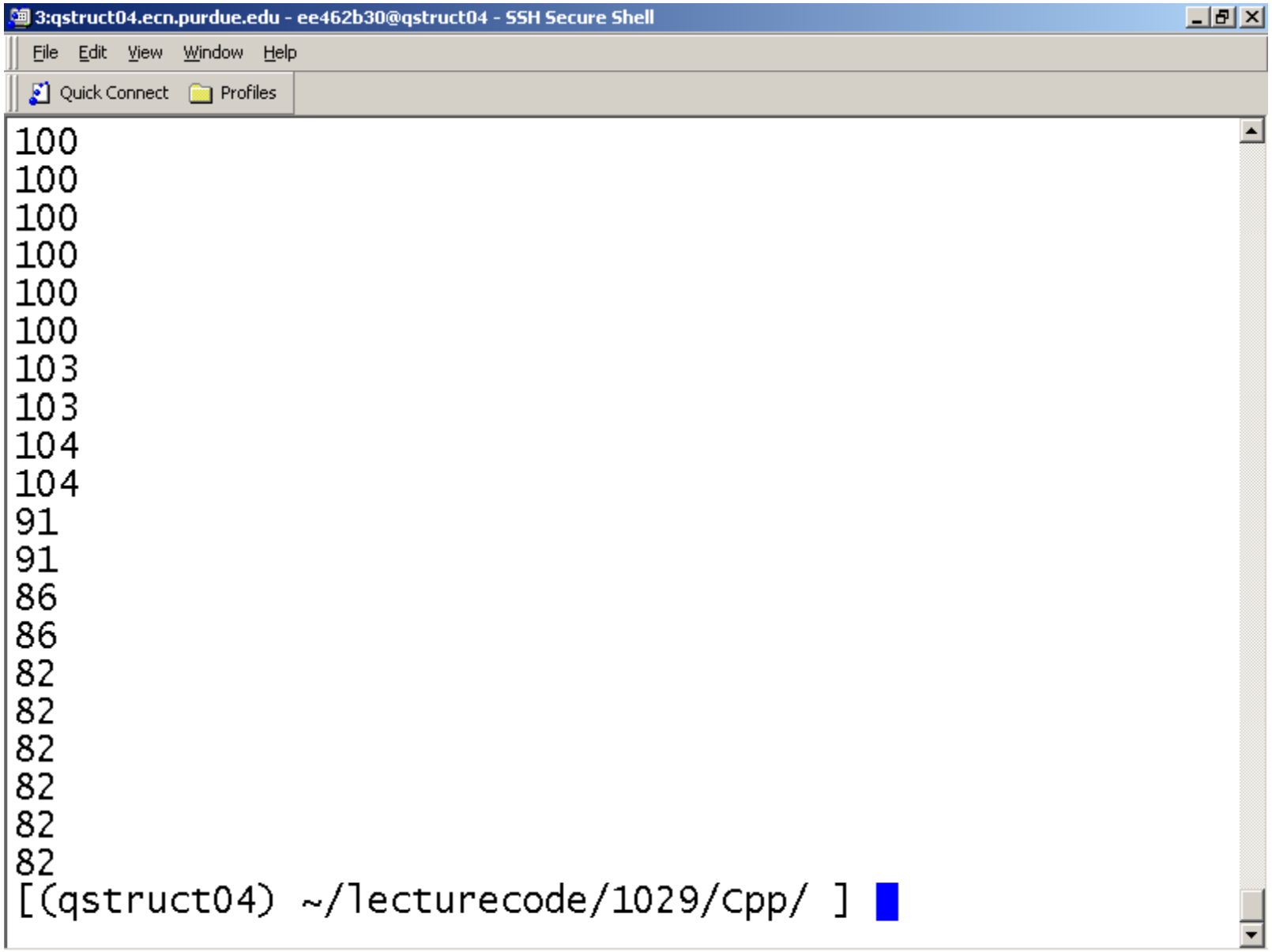



3:qstruct04.ecn.purdue.edu - ee462b30@qstruct04* - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles
100
100
100
104
106
110
103
105
114
106
109
102
102
105
103
103
110
109
114
110
[(algol11) ~/lecturecode/1029/]

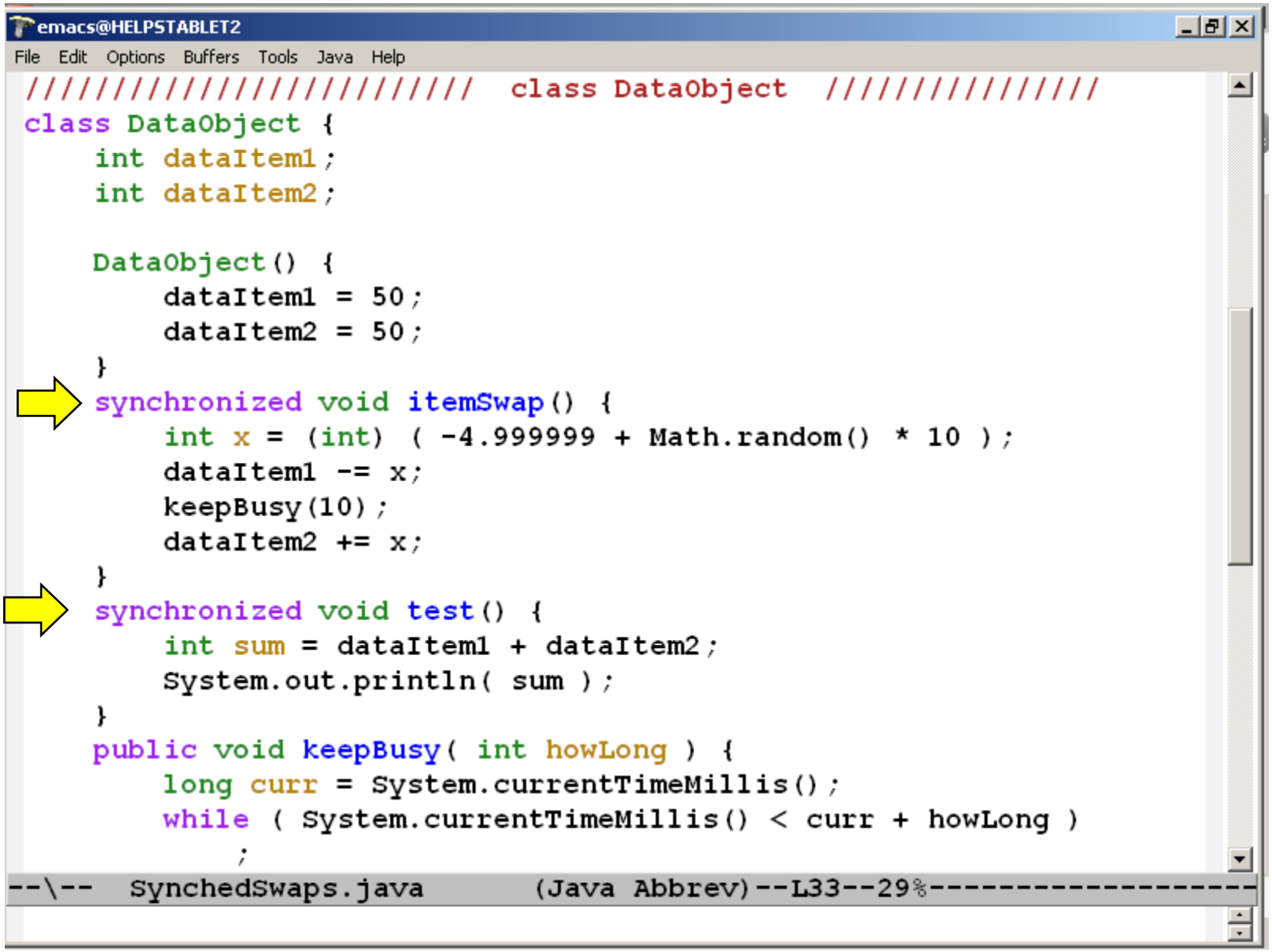

emacs@HELPSTABLET2
File Edit Options Buffers Tools C++ Help
C++
//UnSynchedSwaps.cc
#include <QtCore>
#include <cstdlib>
#include <iostream>
#include <ctime>
using namespace std;
void keepBusy(double howLongInMillisec);
class DataObject {
int dataItem1;
int dataItem2;
public:
DataObject() {
dataItem1 = 50;
dataItem2 = 50;
}
void itemSwap() {
int x = (int) (-4.999999 + rand() % 10);
dataItem1 -= x;
dataItem2 += x;
}
--\-- UnsynchedSwaps.cc (C++ Abbrev)--L22--37%


emacs@HELPSTABLET2
File Edit Options Buffers Tools C++ Help
void itemSwap() {
int x = (int) (-4.999999 + rand() % 10);
dataItem1 -= x;
dataItem2 += x;
}
void test() {
int sum = dataItem1 + dataItem2;
cout << sum << endl;
}
};
DataObject dobj;
class RepeatedSwaps : public QThread {
public:
RepeatedSwaps() {
start();
}
void run() {
int i = 0;
while (i++ < 5000) {
dobj.itemSwap();
if (i % 1000 == 0) dobj.test();
--\-- UnsynchedSwaps.cc (C++ Abbrev)--L48--57%


emacs@HELPSTABLET2
File Edit Options Buffers Tools C++ Help
start();
}
void run() {
int i = 0;
while (i++ < 5000) {
dobj.itemSwap();
if (i % 1000 == 0) dobj.test();
}
}
};
int main( )
{
RepeatedSwaps t0;
RepeatedSwaps t1;
RepeatedSwaps t2;
RepeatedSwaps t3;
t0.wait();
t1.wait();
t2.wait();
t3.wait();
}
--\-- UnsynchedSwaps.cc (C++ Abbrev)--L59--Bot

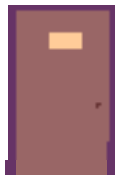

3:qstruct04.ecn.purdue.edu - ee462b30@qstruct04 - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles
100
100
100
100
100
100
103
103
104
104
91
91
86
86
82
82
82
82
82
82
[(qstruct04) ~/lecturecode/1029/Cpp/]

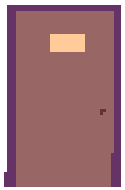
Synchronization

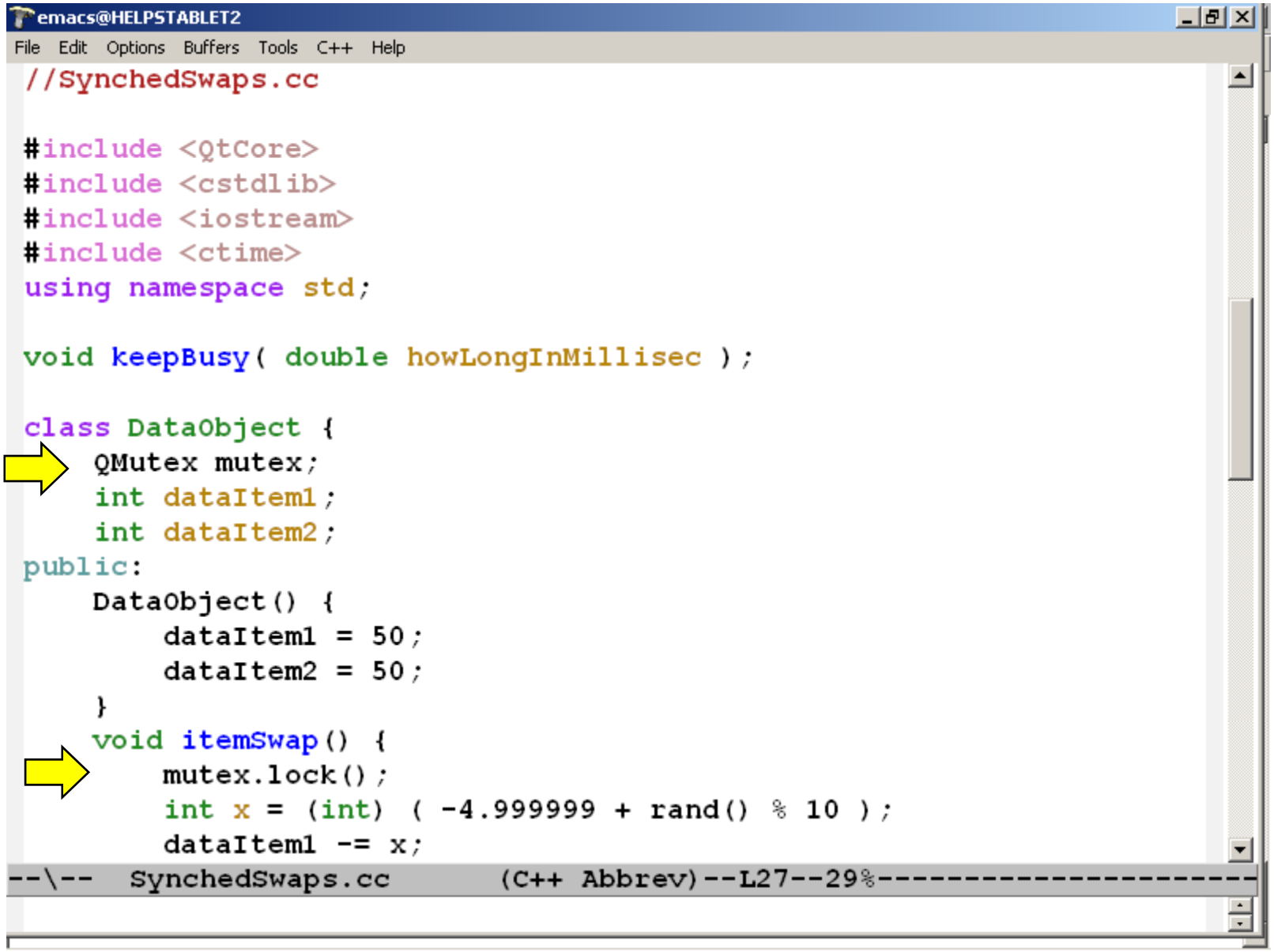

emacs@HELPSTABLET2
File Edit Options Buffers Tools Java Help
////////////////////////// class DataObject ////////////////
class DataObject {
int dataItem1;
int dataItem2;
DataObject() {
dataItem1 = 50;
dataItem2 = 50;
}
synchronized void itemSwap() {
int x = (int) (-4.999999 + Math.random() * 10);
dataItem1 -= x;
keepBusy(10);
dataItem2 += x;
}
synchronized void test() {
int sum = dataItem1 + dataItem2;
System.out.println(sum);
}
public void keepBusy(int howLong) {
long curr = System.currentTimeMillis();
while (System.currentTimeMillis() < curr + howLong)
;
--\-- SynchedSwaps.java (Java Abbrev)--L33--29%

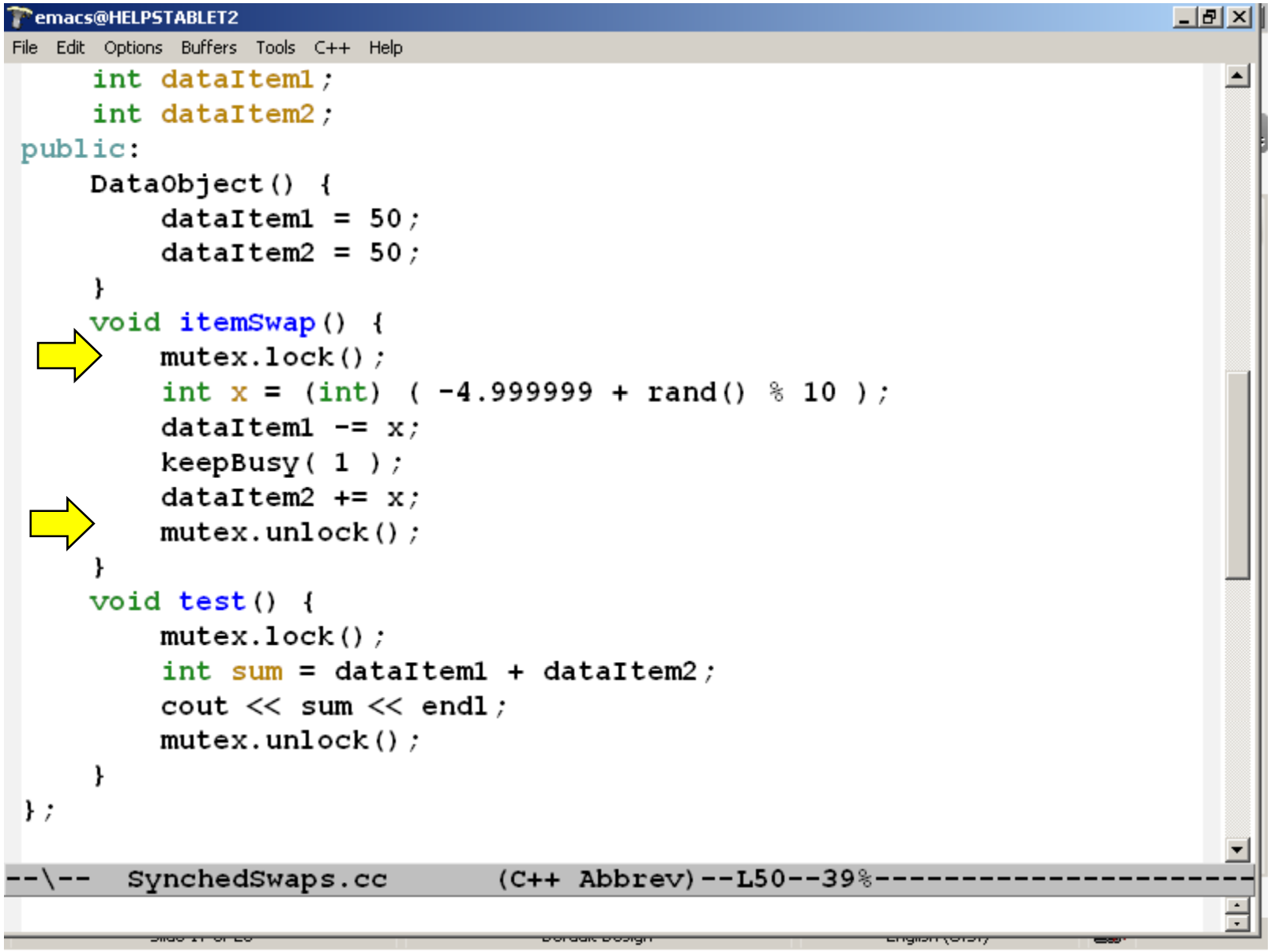
synchronized Methods

- 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

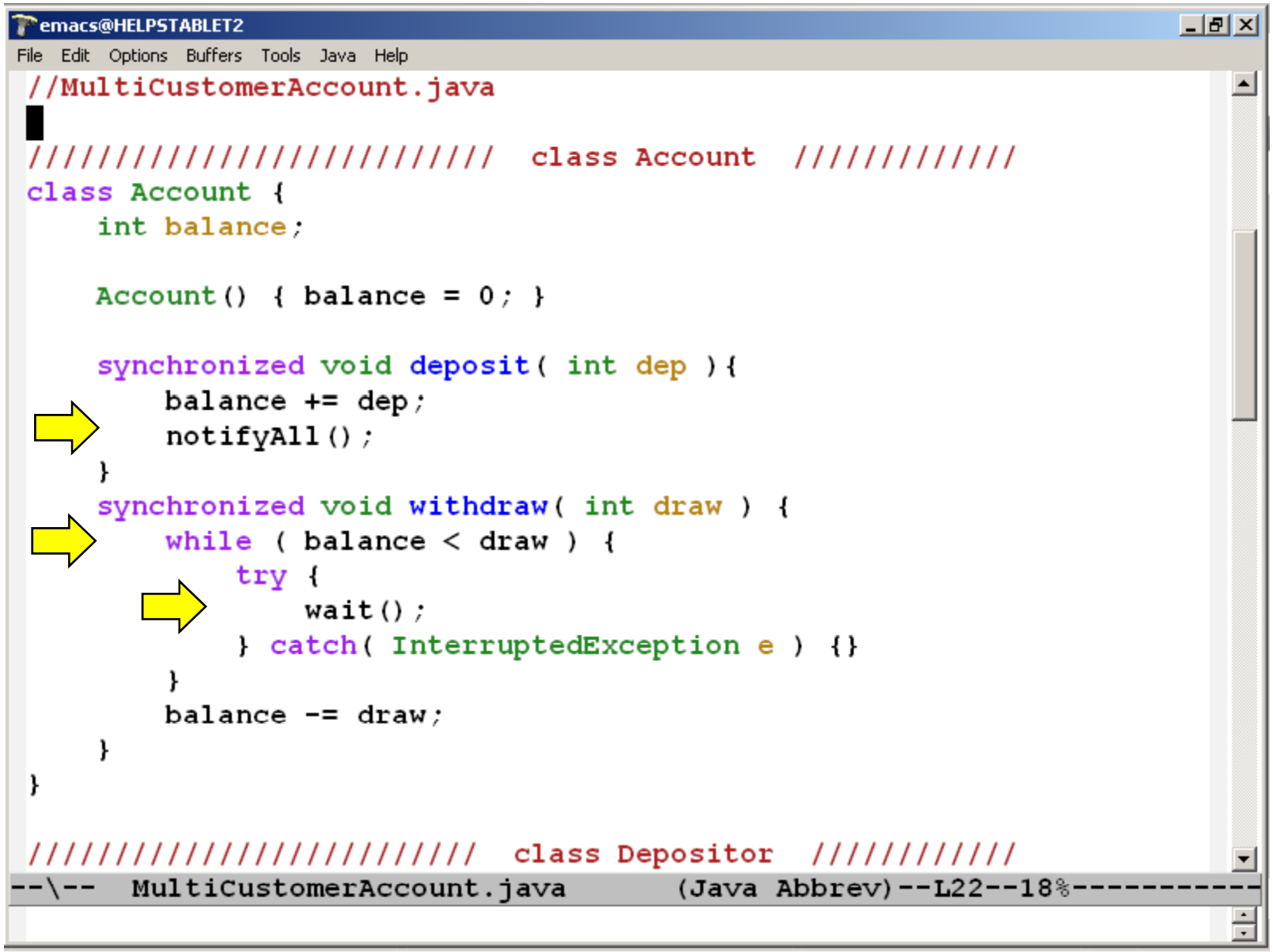





emacs@HELPSTABLET2
File Edit Options Buffers Tools C++ Help
//SynchedSwaps.cc
#include <QtCore>
#include <cstdlib>
#include <iostream>
#include <ctime>
using namespace std;
void keepBusy(double howLongInMillisec);
class DataObject {
QMutex mutex;
int dataItem1;
int dataItem2;
public:
DataObject() {
dataItem1 = 50;
dataItem2 = 50;
}
void itemSwap() {
mutex.lock();
int x = (int) (-4.999999 + rand() % 10);
dataItem1 -= x;
--\-- SynchedSwaps.cc (C++ Abbrev)--L27--29%


emacs@HELPSTABLET2
File Edit Options Buffers Tools C++ Help
int dataItem1;
int dataItem2;
public:
DataObject() {
dataItem1 = 50;
dataItem2 = 50;
}
void itemSwap() {
mutex.lock();
int x = (int) (-4.999999 + rand() % 10);
dataItem1 -= x;
keepBusy(1);
dataItem2 += x;
mutex.unlock();
}
void test() {
mutex.lock();
int sum = dataItem1 + dataItem2;
cout << sum << endl;
mutex.unlock();
}
};
--\-- SynchedSwaps.cc (C++ Abbrev)--L50--39%

Conditional Wait

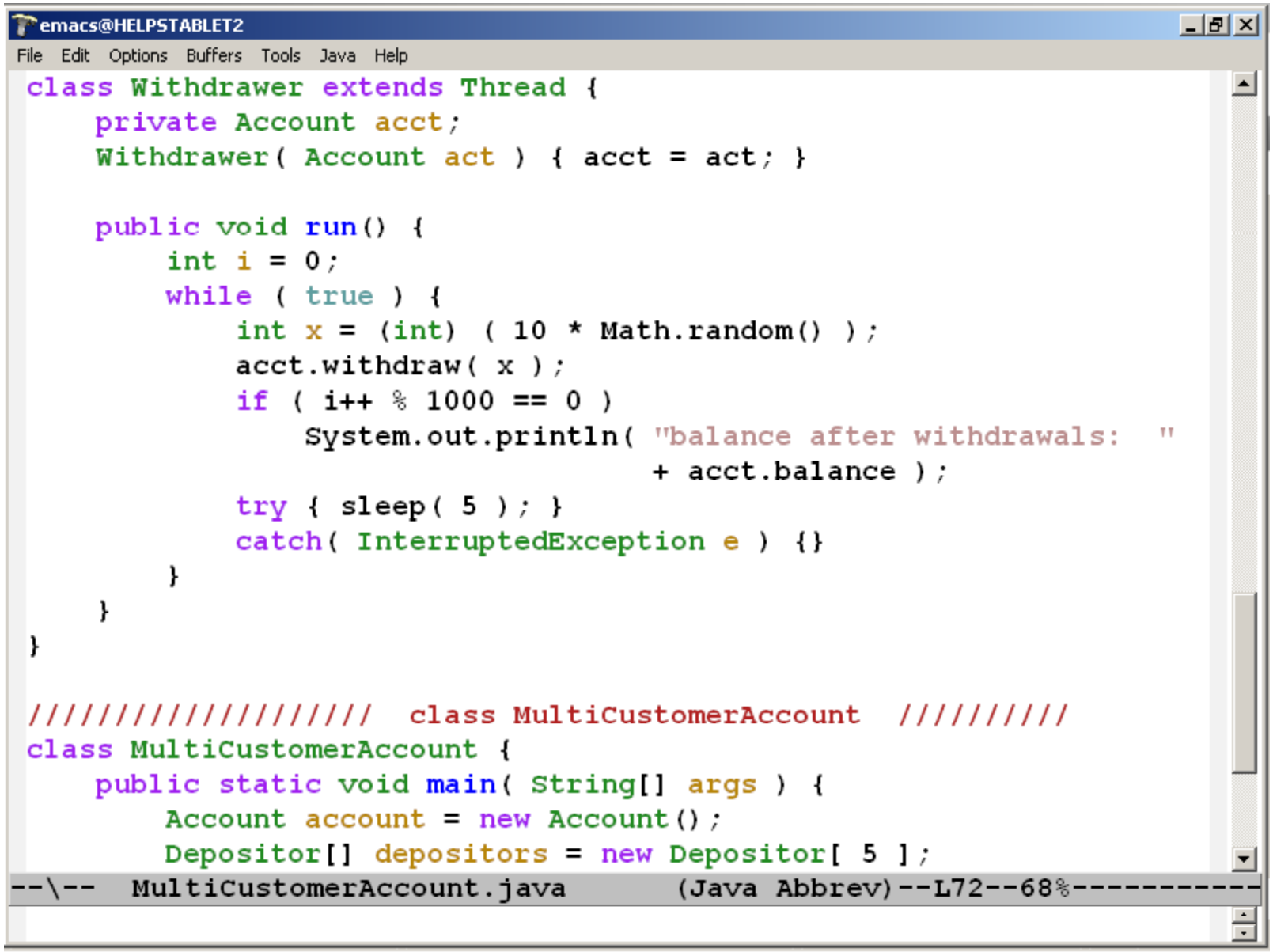

emacs@HELPSTABLET2
File Edit Options Buffers Tools Java Help
//MultiCustomerAccount.java
////////////////////////// class Account //////////////
class Account {
int balance;
Account() { balance = 0; }
synchronized void deposit(int dep){
balance += dep;
notifyAll();
}
synchronized void withdraw(int draw) {
while (balance < draw) {
try {
wait();
} catch(InterruptedException e) {}
}
balance -= draw;
}
}
////////////////////////// class Depositor ////////////
--\-- MultiCustomerAccount.java (Java Abbrev)--L22--18%----------

emacs@HELPSTABLET2

File Edit Options Buffers Tools Java Help

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

//////////////////////////  class Withdrawer  ///////////////
class Withdrawer extends Thread {
    private Account acct;
    Withdrawer( Account act ) { acct = act; }

    public void run() {
```

--\-- MultiCustomerAccount.java   (Java Abbrev)--L53--43%----------


emacs@HELPSTABLET2
File Edit Options Buffers Tools Java Help
class Withdrawer extends Thread {
private Account acct;
Withdrawer(Account act) { acct = act; }
public void run() {
int i = 0;
while (true) {
int x = (int) (10 * Math.random());
acct.withdraw(x);
if (i++ % 1000 == 0)
System.out.println("balance after withdrawals: "
+ acct.balance);
try { sleep(5); }
catch(InterruptedException e) {}
}
}
}
//////////////////// class MultiCustomerAccount //////////
class MultiCustomerAccount {
public static void main(String[] args) {
Account account = new Account();
Depositor[] depositors = new Depositor[5];
--\-- MultiCustomerAccount.java (Java Abbrev)--L72--68%----------

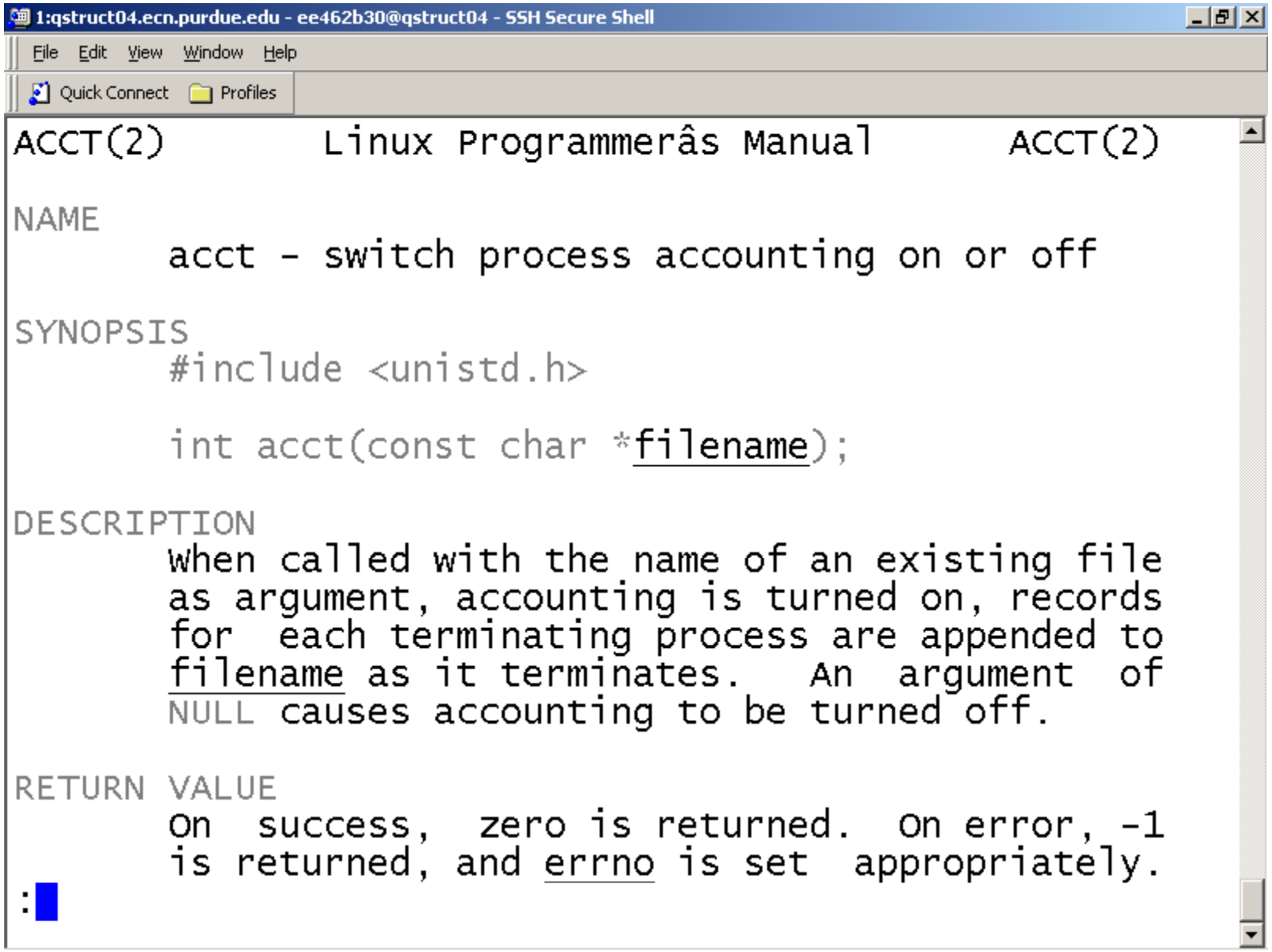
shared object

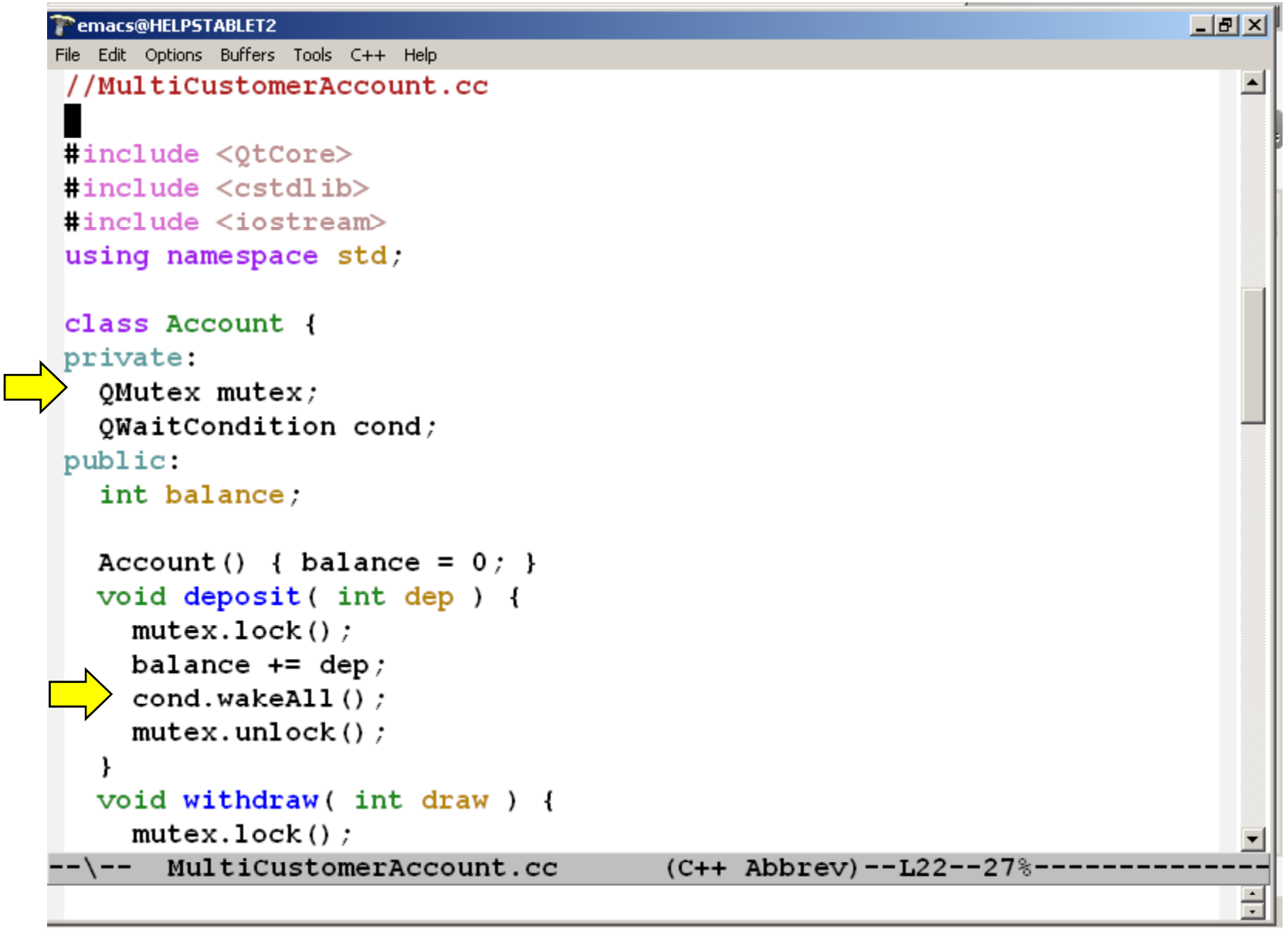
emacs@HELPSTABLET2

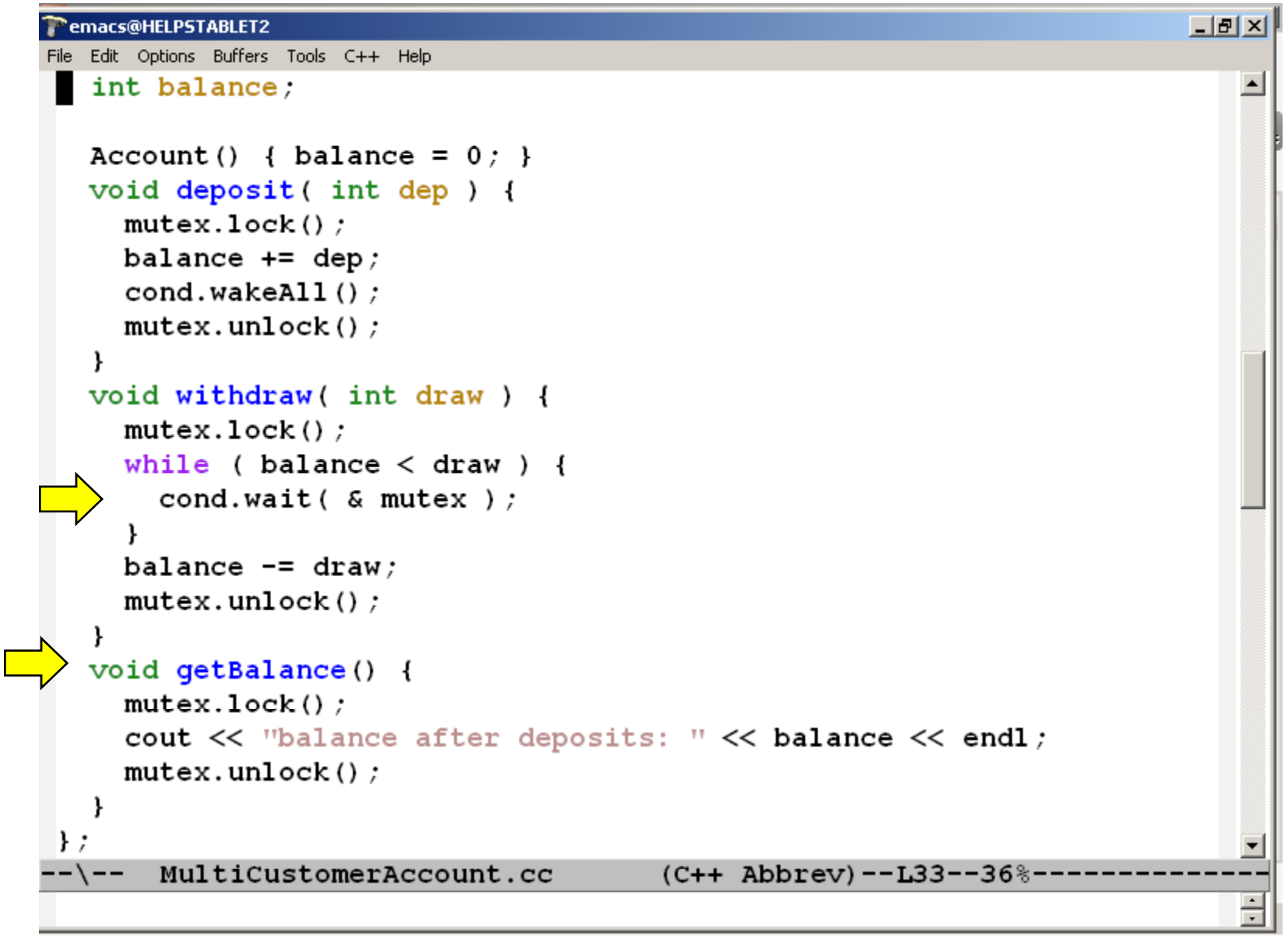
File Edit Options B

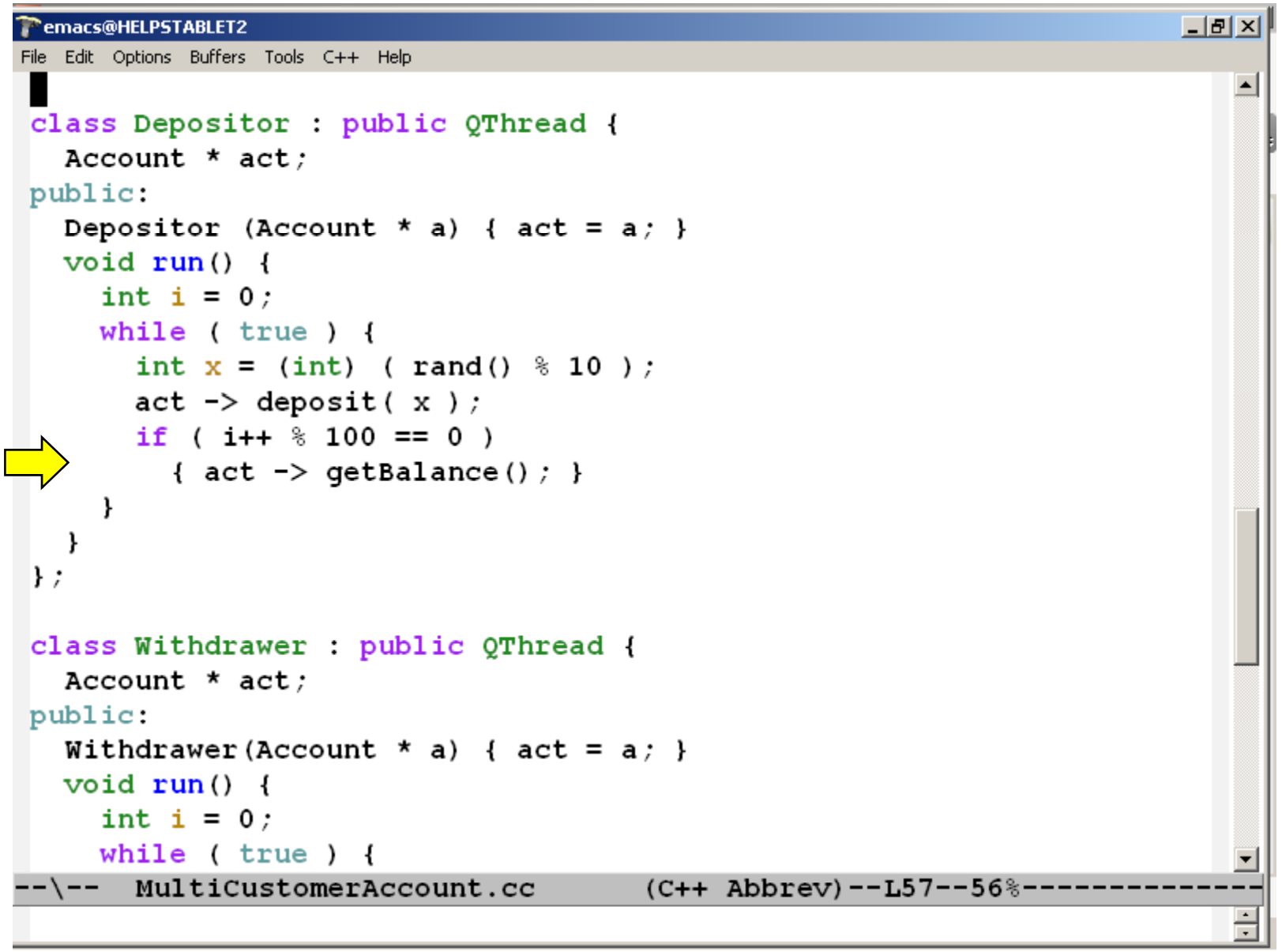
```
}

//////////////////////  class MultiCustomerAccount  //////////
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();
        }
    }
}
```

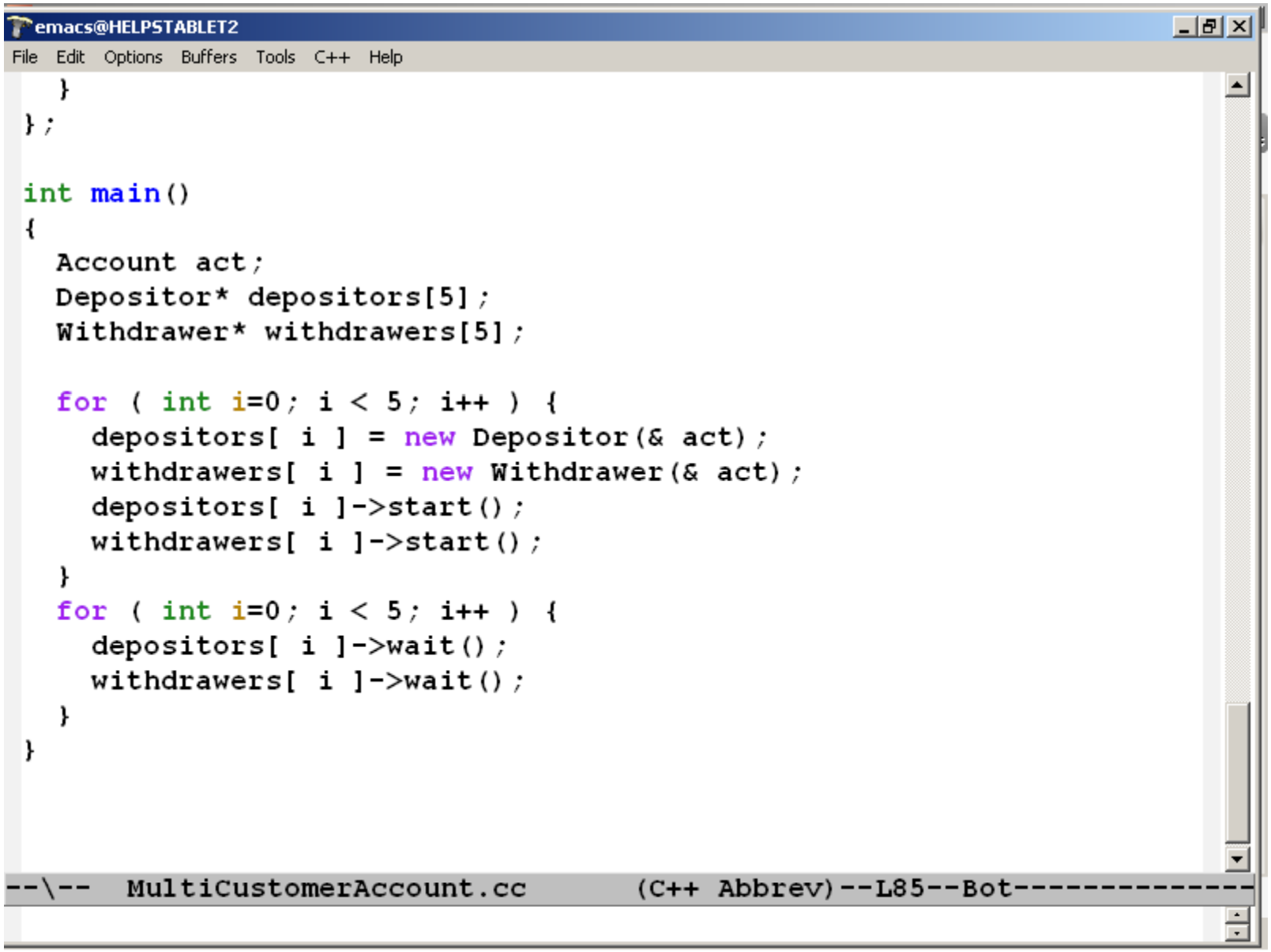
--\-- MultiCustomerAccount.java   (Java Abbrev)--L78--Bot----------


1:qstruct04.ecn.purdue.edu - ee462b30@qstruct04 - SSH Secure Shell
File Edit View Window Help
Quick Connect Profiles
ACCT(2) Linux Programmerâs Manual ACCT(2)
NAME
acct - switch process accounting on or off
SYNOPSIS
#include <unistd.h>
int acct(const char *filename);
DESCRIPTION
When called with the name of an existing file as argument, accounting is turned on, records for each terminating process are appended to filename as it terminates. An argument of NULL causes accounting to be turned off.
RETURN VALUE
On success, zero is returned. On error, -1 is returned, and errno is set appropriately.
:


emacs@HELPSTABLET2
File Edit Options Buffers Tools C++ Help
//MultiCustomerAccount.cc
#include <QtCore>
#include <cstdlib>
#include <iostream>
using namespace std;
class Account {
private:
QMutex mutex;
QWaitCondition cond;
public:
int balance;
Account() { balance = 0; }
void deposit(int dep) {
mutex.lock();
balance += dep;
cond.wakeAll();
mutex.unlock();
}
void withdraw(int draw) {
mutex.lock();
--\-- MultiCustomerAccount.cc (C++ Abbrev)--L22--27%


emacs@HELPSTABLET2
File Edit Options Buffers Tools C++ Help
int balance;
Account() { balance = 0; }
void deposit(int dep) {
mutex.lock();
balance += dep;
cond.wakeAll();
mutex.unlock();
}
void withdraw(int draw) {
mutex.lock();
while (balance < draw) {
cond.wait(& mutex);
}
balance -= draw;
mutex.unlock();
}
void getBalance() {
mutex.lock();
cout << "balance after deposits: " << balance << endl;
mutex.unlock();
}
};
--\-- MultiCustomerAccount.cc (C++ Abbrev)--L33--36%


emacs@HELPSTABLET2
File Edit Options Buffers Tools C++ Help
class Depositor : public QThread {
Account * act;
public:
Depositor (Account * a) { act = a; }
void run() {
int i = 0;
while (true) {
int x = (int) (rand() % 10);
act -> deposit(x);
if (i++ % 100 == 0)
{ act -> getBalance(); }
}
}
};
class Withdrawer : public QThread {
Account * act;
public:
Withdrawer(Account * a) { act = a; }
void run() {
int i = 0;
while (true) {
--\-- MultiCustomerAccount.cc (C++ Abbrev)--L57--56%


emacs@HELPSTABLET2
File Edit Options Buffers Tools C++ Help
}
};
int main()
{
Account act;
Depositor* depositors[5];
Withdrawer* withdrawers[5];
for (int i=0; i < 5; i++) {
depositors[i] = new Depositor(& act);
withdrawers[i] = new Withdrawer(& act);
depositors[i]->start();
withdrawers[i]->start();
}
for (int i=0; i < 5; i++) {
depositors[i]->wait();
withdrawers[i]->wait();
}
}
--\-- MultiCustomerAccount.cc (C++ Abbrev)--L85--Bot