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

Design Parallel Programs

Yung-Hsiang Lu

yunglu@purdue.edu

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()

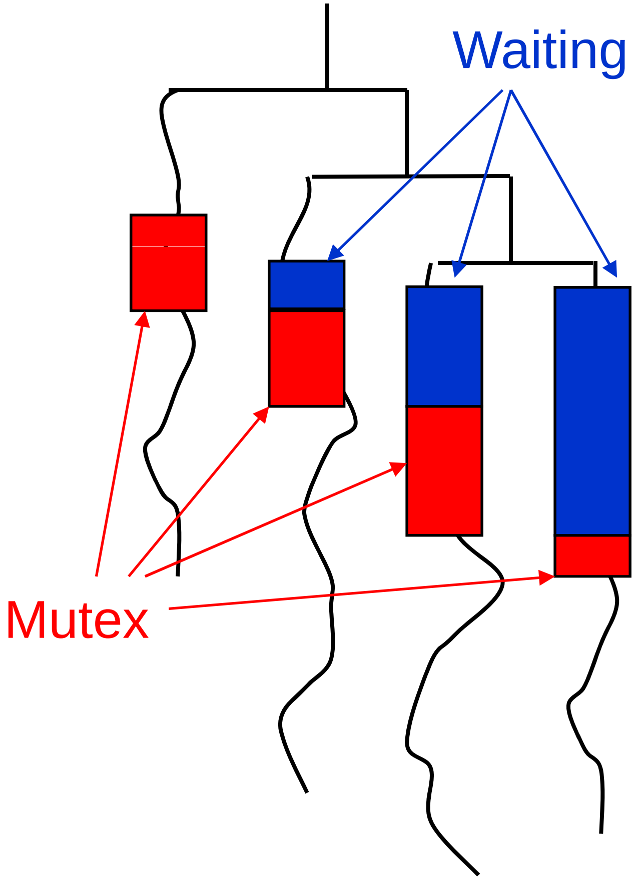
try {
wait ();
} catch …
	cond.wait(&mutex);

Design Principles

- separate data and threads
  - data: such as bank account, shared among threads, need protected by synchronized (Java) or mutex (Qt)
  - threads: such as depositor, extends Threads (Java) or : public QThread (C++). modify data
- choose the granularity of data
  - too coarse: most threads are serialized
  - too fine: mutex overhead dominant

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.

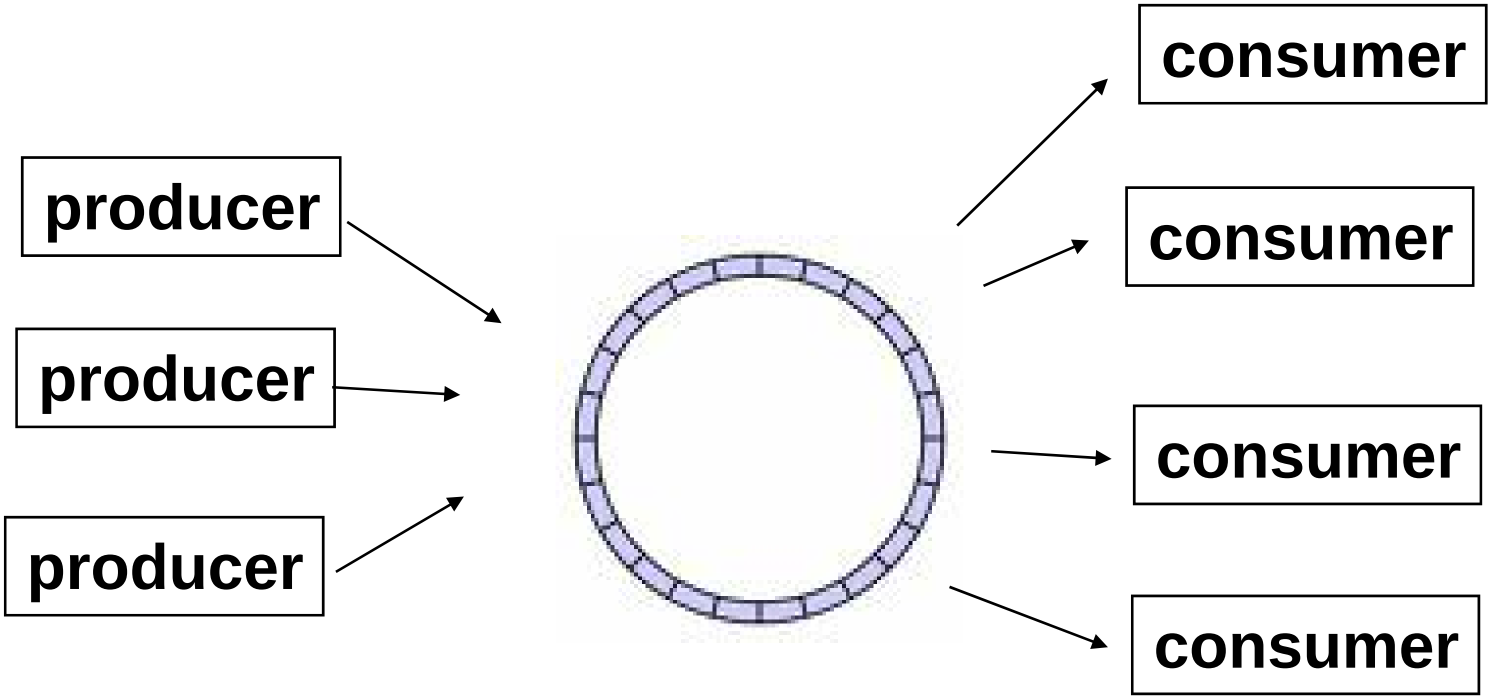


Improve Thread Efficiency

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 ⇒ individual customer's account
- shrink critical sections ⇒ only the operations that are related to the shared objects are in mutex
- use private (not shared) data for short-term storage
- do not guarantee correctness, remedy later

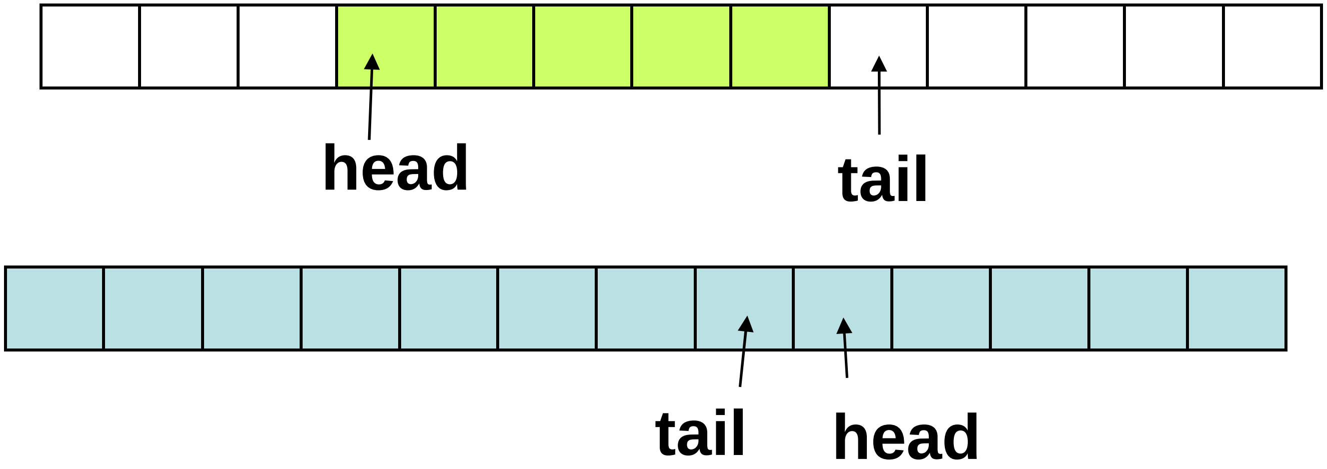
Producers and Consumers with a Finite Buffer



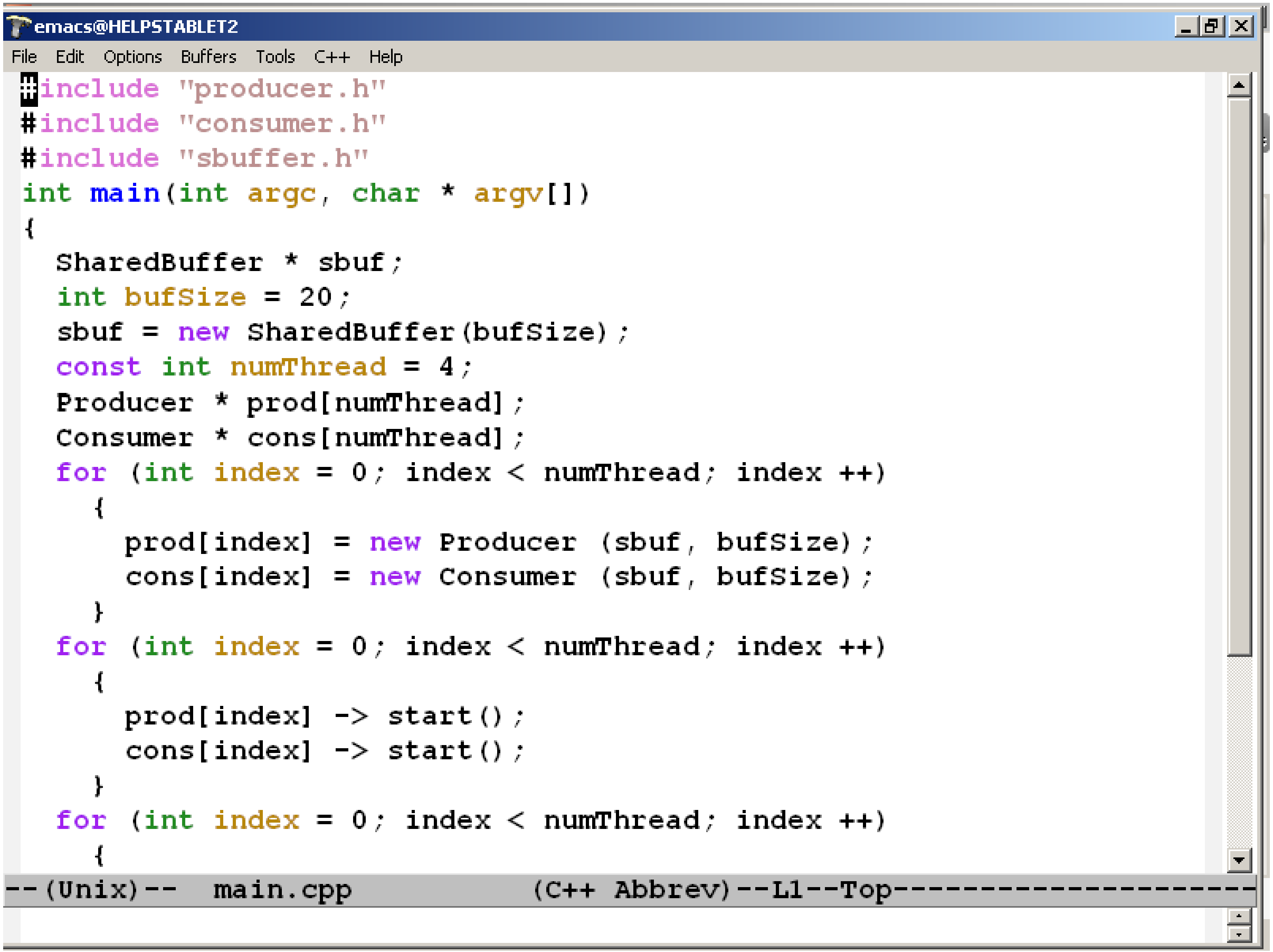
circular buffer (FIFO)

A producer can put an element in the buffer if it is not full. A consumer can get an element the buffer if it is not empty.

Circular Buffer

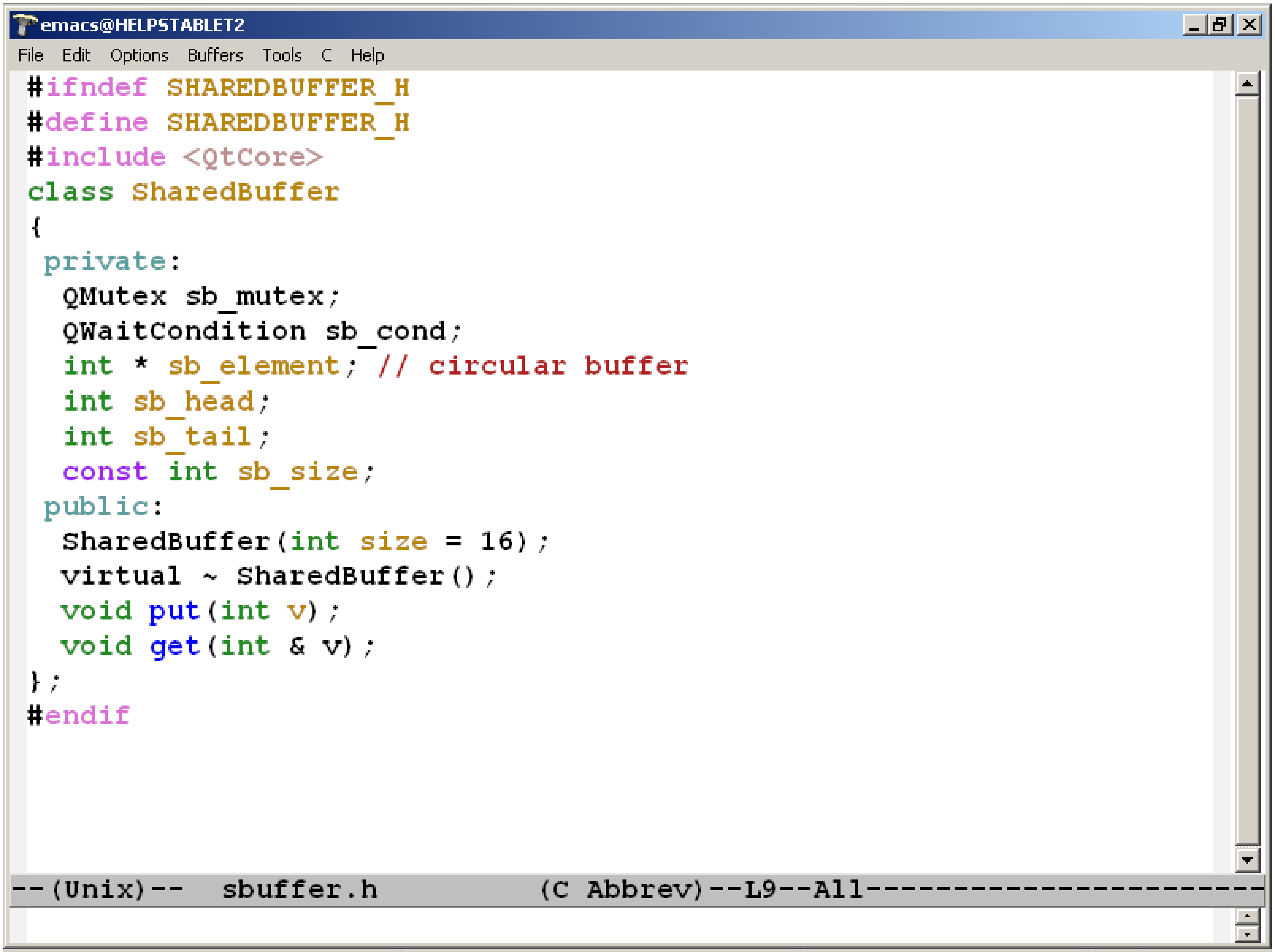


- head = tail ⇒ buffer empty
- tail > head and (tail – head) == size – 1 ⇒ buffer full
- head > tail and (head – tail) == 1 ⇒ buffer full



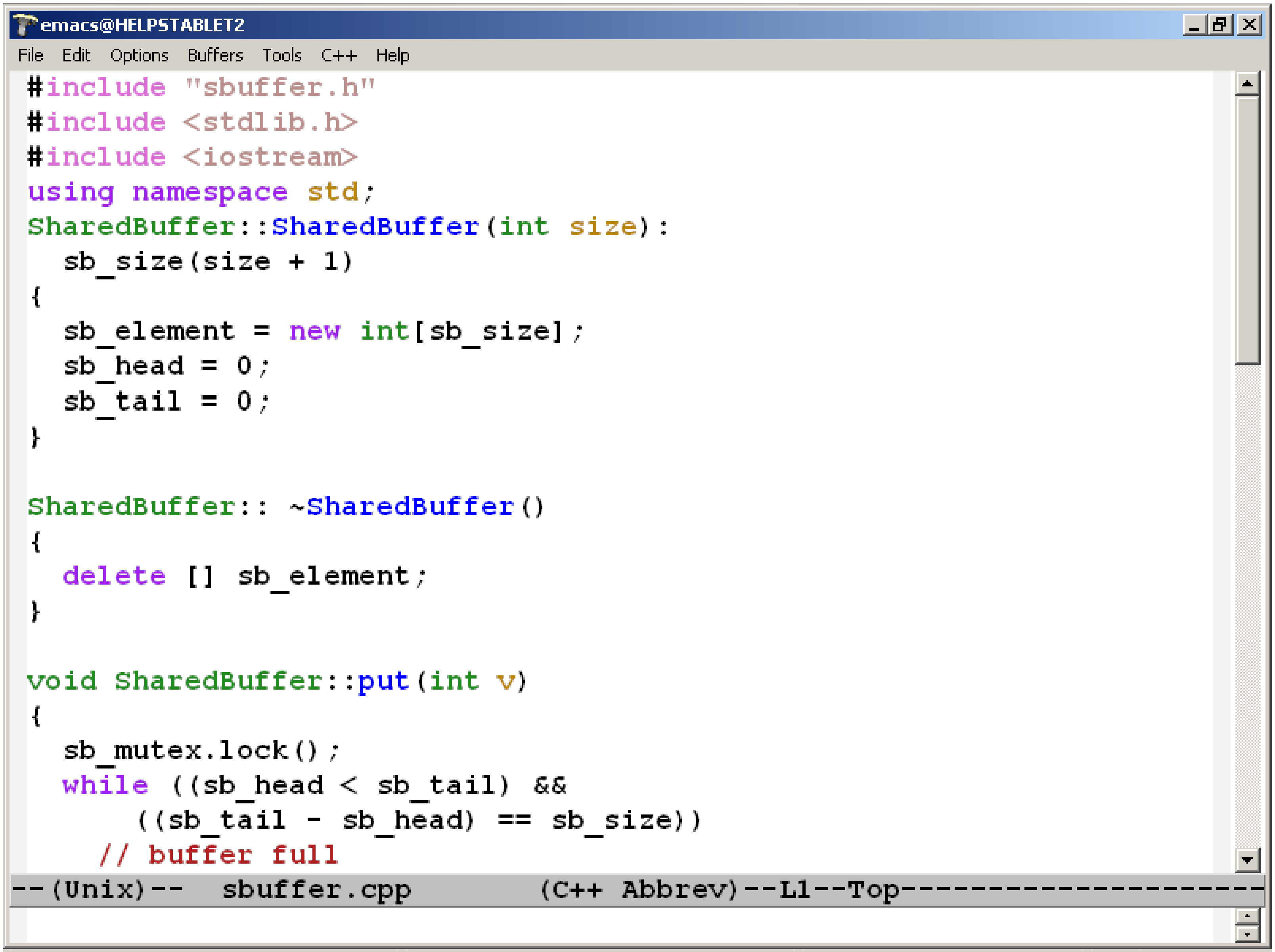
```
#include "producer.h"
#include "consumer.h"
#include "sbuffer.h"
int main(int argc, char * argv[])
{
  SharedBuffer * sbuf;
  int bufSize = 20;
  sbuf = new SharedBuffer(bufSize);
  const int numThread = 4;
  Producer * prod[numThread];
  Consumer * cons[numThread];
  for (int index = 0; index < numThread; index ++)
    {
      prod[index] = new Producer (sbuf, bufSize);
      cons[index] = new Consumer (sbuf, bufSize);
    }
  for (int index = 0; index < numThread; index ++)
    {
      prod[index] -> start();
      cons[index] -> start();
    }
  for (int index = 0; index < numThread; index ++)
    {
```

--(Unix)-- main.cpp   (C++ Abbrev)--L1--Top--------------------


emacs@HELPSTABLET2
File Edit Options Buffers Tools C Help

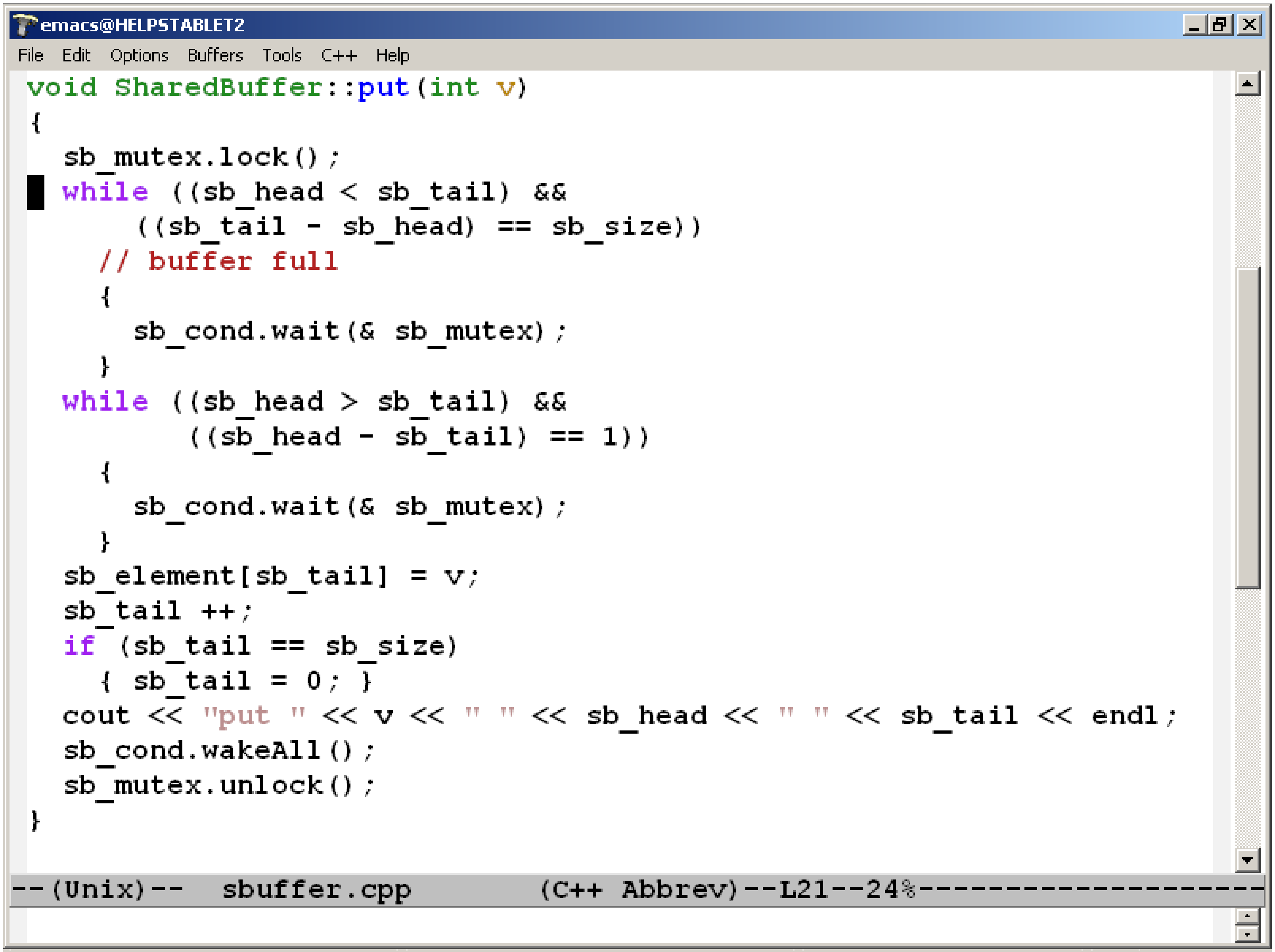
```
#ifndef SHAREDBUFFER_H
#define SHAREDBUFFER_H
#include <QtCore>
class SharedBuffer
{
 private:
  QMutex sb_mutex;
  QWaitCondition sb_cond;
  int * sb_element; // circular buffer
  int sb_head;
  int sb_tail;
  const int sb_size;
 public:
  SharedBuffer(int size = 16);
  virtual ~ SharedBuffer();
  void put(int v);
  void get(int & v);
};
#endif
```

--(Unix)--   sbuffer.h   (C Abbrev)--L9--All----------------------


emacs@HELPSTABLET2
File Edit Options Buffers Tools C++ Help

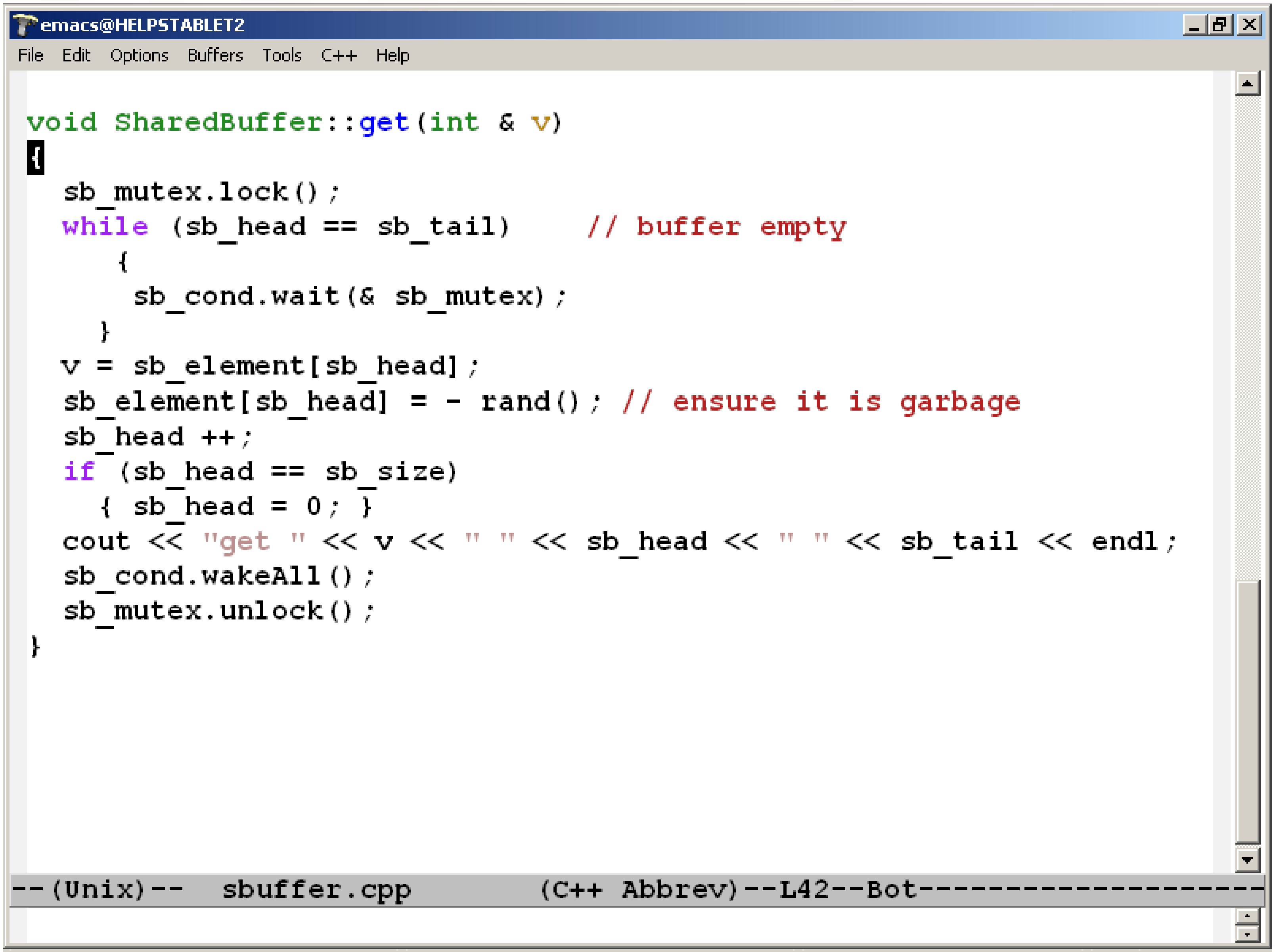
```
#include "sbuffer.h"
#include <stdlib.h>
#include <iostream>
using namespace std;
SharedBuffer::SharedBuffer(int size):
  sb_size(size + 1)
{
  sb_element = new int[sb_size];
  sb_head = 0;
  sb_tail = 0;
}

SharedBuffer:: ~SharedBuffer()
{
  delete [] sb_element;
}

void SharedBuffer::put(int v)
{
  sb_mutex.lock();
  while ((sb_head < sb_tail) &&
        ((sb_tail - sb_head) == sb_size))
    // buffer full
```

--(Unix)--   sbuffer.cpp   (C++ Abbrev)--L1--Top--------------------

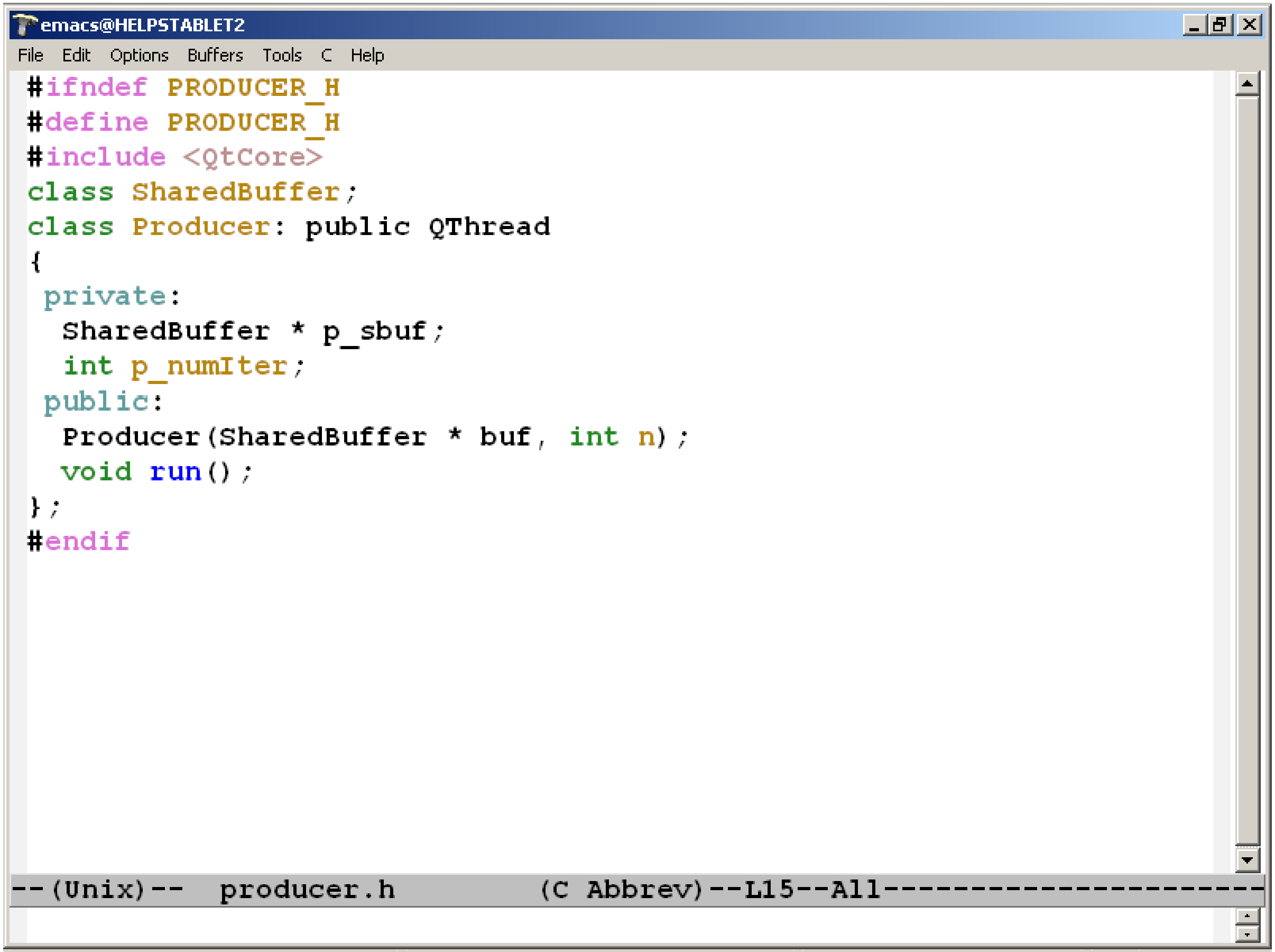

emacs@HELPSTABLET2
File Edit Options Buffers Tools C++ Help

```
void SharedBuffer::put(int v)
{
  sb_mutex.lock();
  while ((sb_head < sb_tail) &&
         ((sb_tail - sb_head) == sb_size))
    // buffer full
    {
      sb_cond.wait(& sb_mutex);
    }
  while ((sb_head > sb_tail) &&
            ((sb_head - sb_tail) == 1))
    {
      sb_cond.wait(& sb_mutex);
    }
  sb_element[sb_tail] = v;
  sb_tail ++;
  if (sb_tail == sb_size)
    { sb_tail = 0; }
  cout << "put " << v << " " << sb_head << " " << sb_tail << endl;
  sb_cond.wakeAll();
  sb_mutex.unlock();
}
```

--(Unix)--   sbuffer.cpp   (C++ Abbrev)--L21--24%-------------------

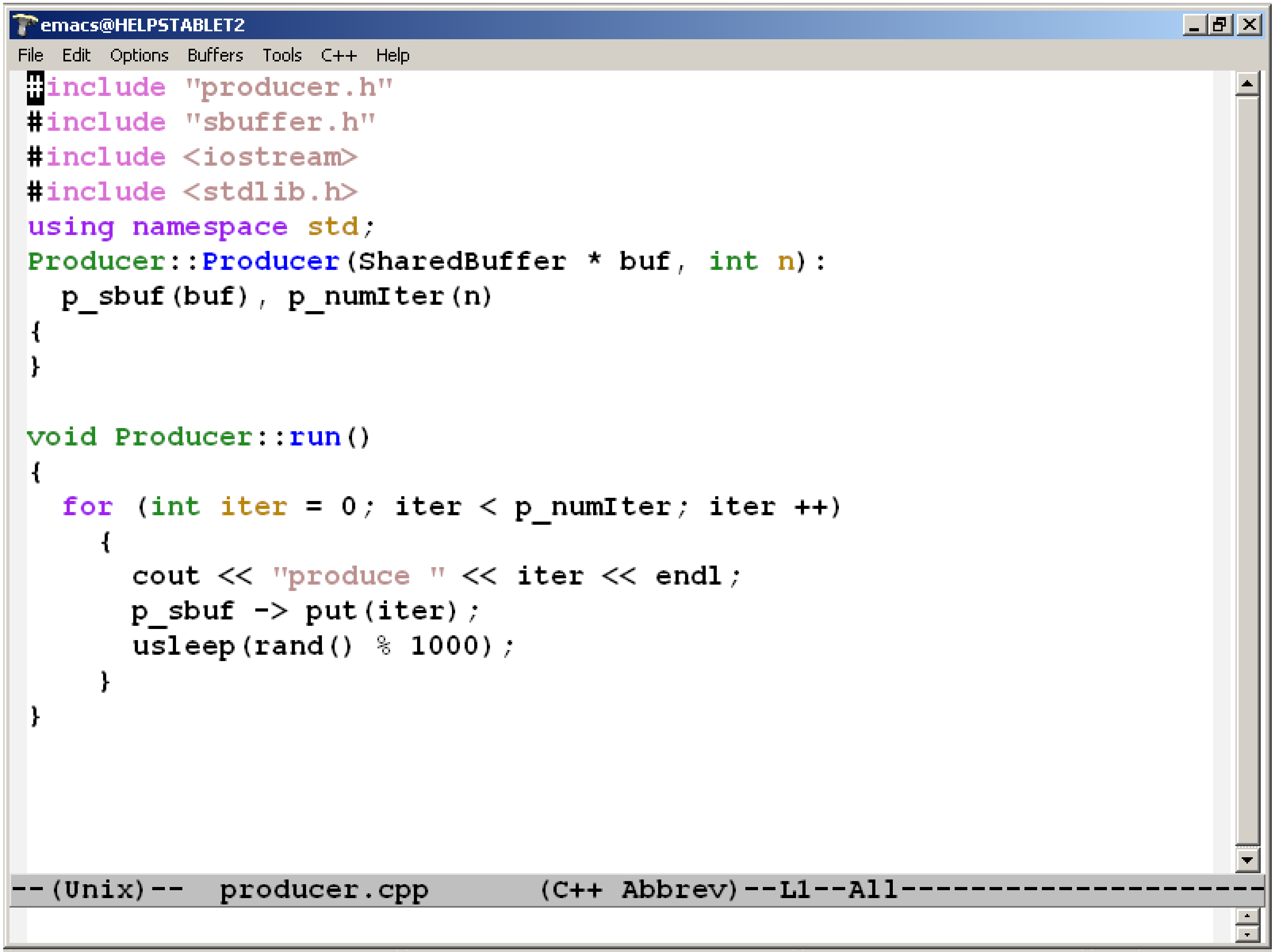

emacs@HELPSTABLET2
File Edit Options Buffers Tools C++ Help
--(Unix)--   sbuffer.cpp   (C++ Abbrev)--L42--Bot-------------------

```
void SharedBuffer::get(int & v)
{
   sb_mutex.lock();
   while (sb_head == sb_tail)      // buffer empty
      {
       sb_cond.wait(& sb_mutex);
     }
   v = sb_element[sb_head];
   sb_element[sb_head] = - rand(); // ensure it is garbage
   sb_head ++;
   if (sb_head == sb_size)
     { sb_head = 0; }
   cout << "get " << v << " " << sb_head << " " << sb_tail << endl;
   sb_cond.wakeAll();
   sb_mutex.unlock();
}
```



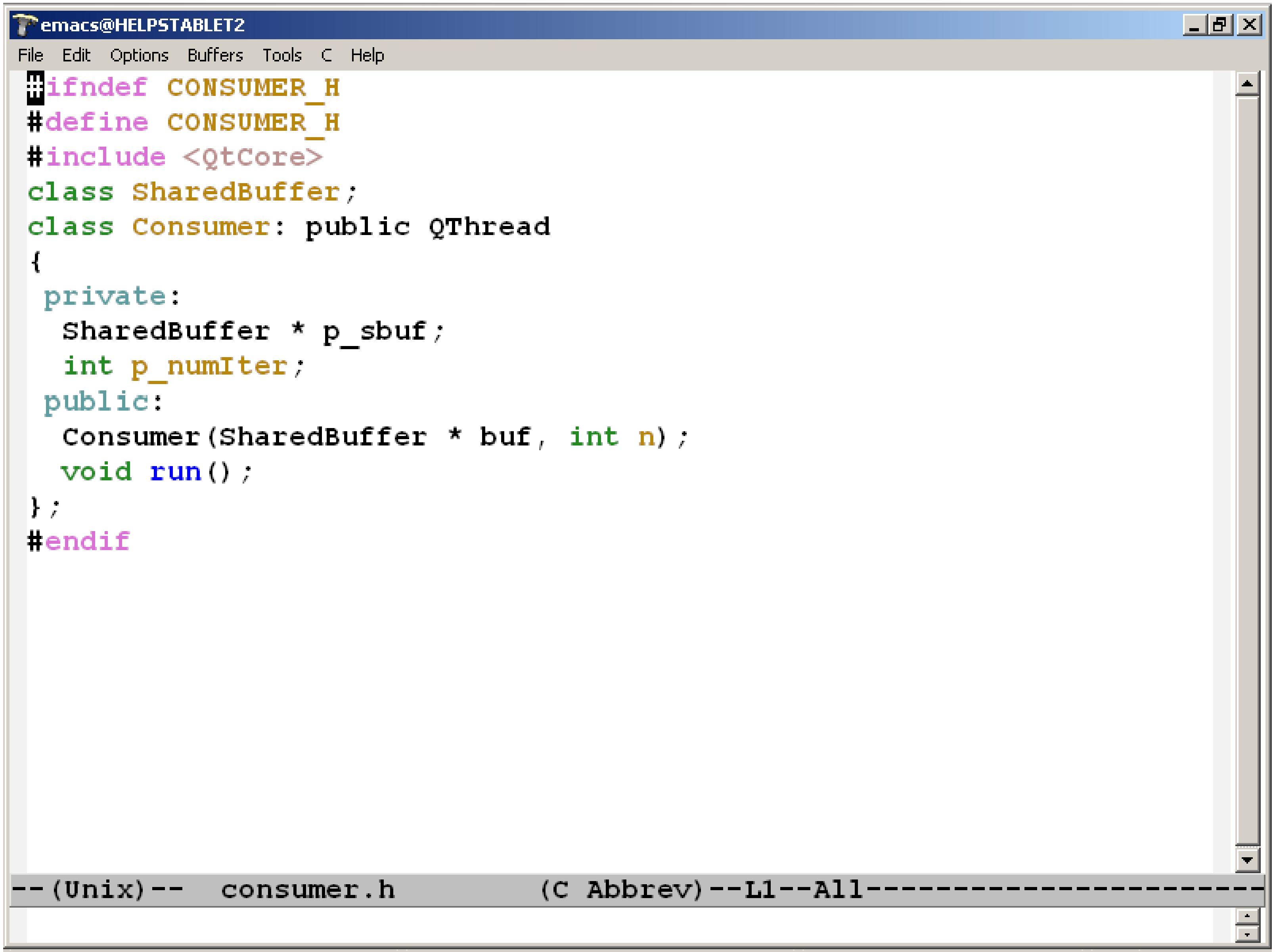
```
#ifndef PRODUCER_H
#define PRODUCER_H
#include <QtCore>
class SharedBuffer;
class Producer: public QThread
{
 private:
  SharedBuffer * p_sbuf;
  int p_numIter;
 public:
  Producer(SharedBuffer * buf, int n);
  void run();
};
#endif
```

```
--(Unix)--   producer.h        (C Abbrev)--L15--All---------------------
```


emacs@HELPSTABLET2
File Edit Options Buffers Tools C++ Help

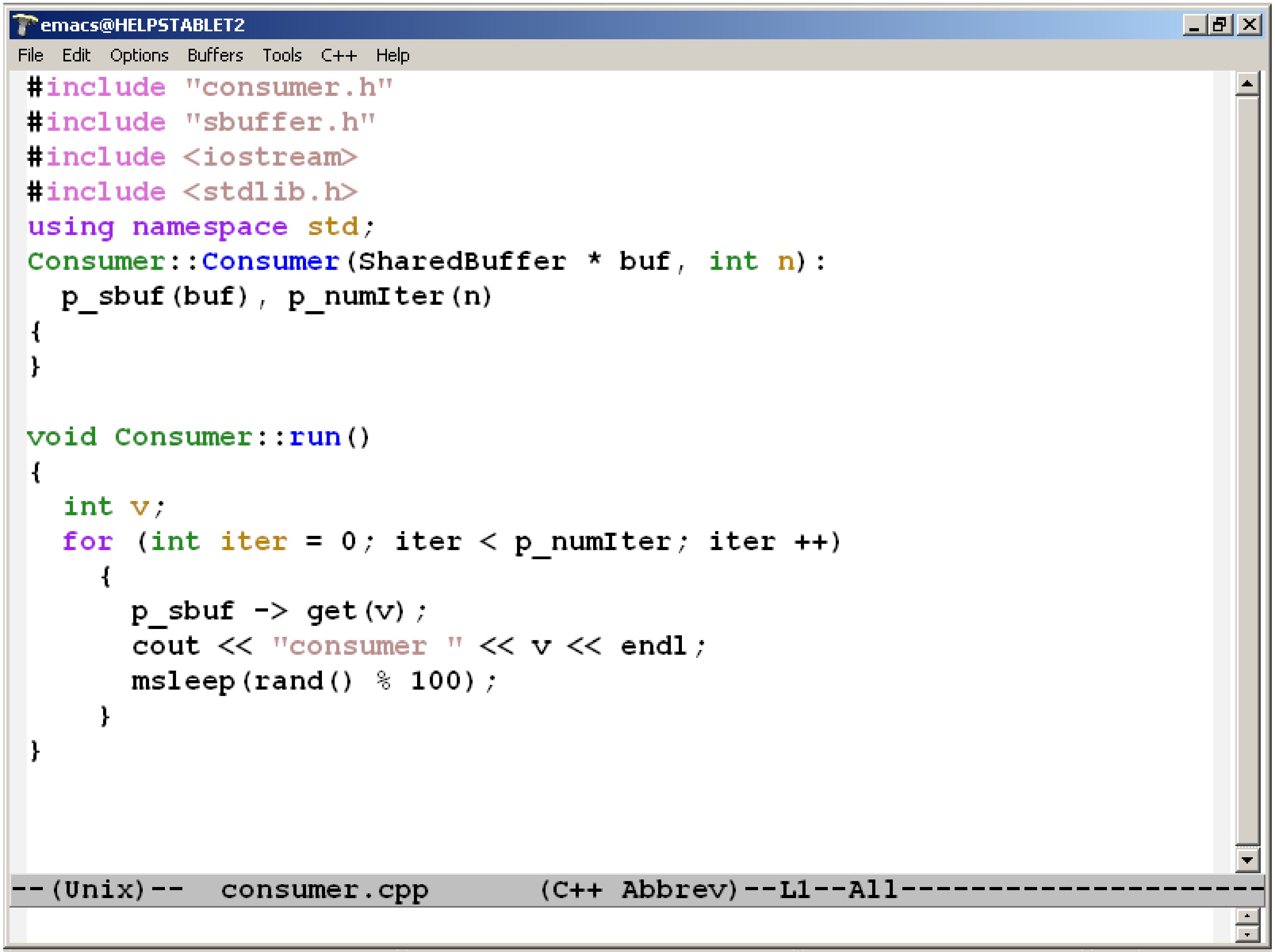
```
#include "producer.h"
#include "sbuffer.h"
#include <iostream>
#include <stdlib.h>
using namespace std;
Producer::Producer(SharedBuffer * buf, int n):
  p_sbuf(buf), p_numIter(n)
{
}

void Producer::run()
{
  for (int iter = 0; iter < p_numIter; iter ++)
    {
      cout << "produce " << iter << endl;
      p_sbuf -> put(iter);
      usleep(rand() % 1000);
    }
}
```

--(Unix)--   producer.cpp   (C++ Abbrev)--L1--All--------------------


emacs@HELPSTABLET2
File Edit Options Buffers Tools C Help

```
#ifndef CONSUMER_H
#define CONSUMER_H
#include <QtCore>
class SharedBuffer;
class Consumer: public QThread
{
 private:
  SharedBuffer * p_sbuf;
  int p_numIter;
 public:
  Consumer(SharedBuffer * buf, int n);
  void run();
};
#endif
```

--(Unix)-- consumer.h   (C Abbrev)--L1--All----------------------


emacs@HELPSTABLET2
File Edit Options Buffers Tools C++ Help
--(Unix)--   consumer.cpp   (C++ Abbrev)--L1--All--------------------

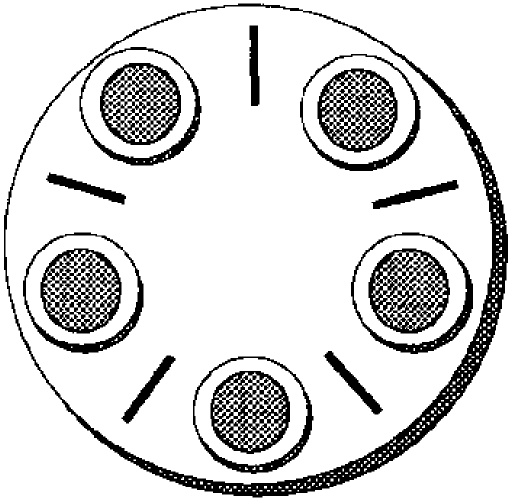
```
#include "consumer.h"
#include "sbuffer.h"
#include <iostream>
#include <stdlib.h>
using namespace std;
Consumer::Consumer(SharedBuffer * buf, int n):
  p_sbuf(buf), p_numIter(n)
{
}

void Consumer::run()
{
  int v;
  for (int iter = 0; iter < p_numIter; iter ++)
    {
      p_sbuf -> get(v);
      cout << "consumer " << v << endl;
      msleep(rand() % 100);
    }
}
```

Check Correctness

- wrong result ⇒ the program has bugs
- correct result ⇒ maybe you are lucky
- know the correct results, even though the interleaving may be different
- head ≠ tail after adding an element
- randomize interleaving if possible
- assign invalid values to data that should not appear

Deadlock and Livelock

- deadlock: none of the participating threads can execute any more code
- livelock: some (or all) of the participating threads can execute some more code, but no progress is made. for example, dinning philosophers



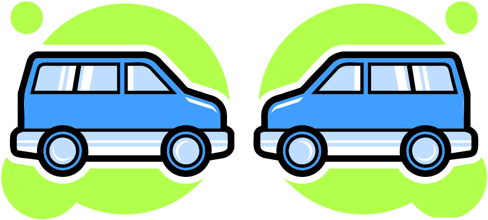
ECE 462
Object-Oriented Programming using C++ and Java

Deadlock

Yung-Hsiang Lu
yunglu@purdue.edu

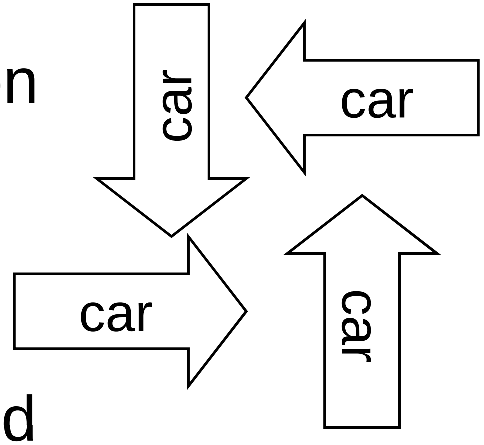
Deadlock

Deadlock

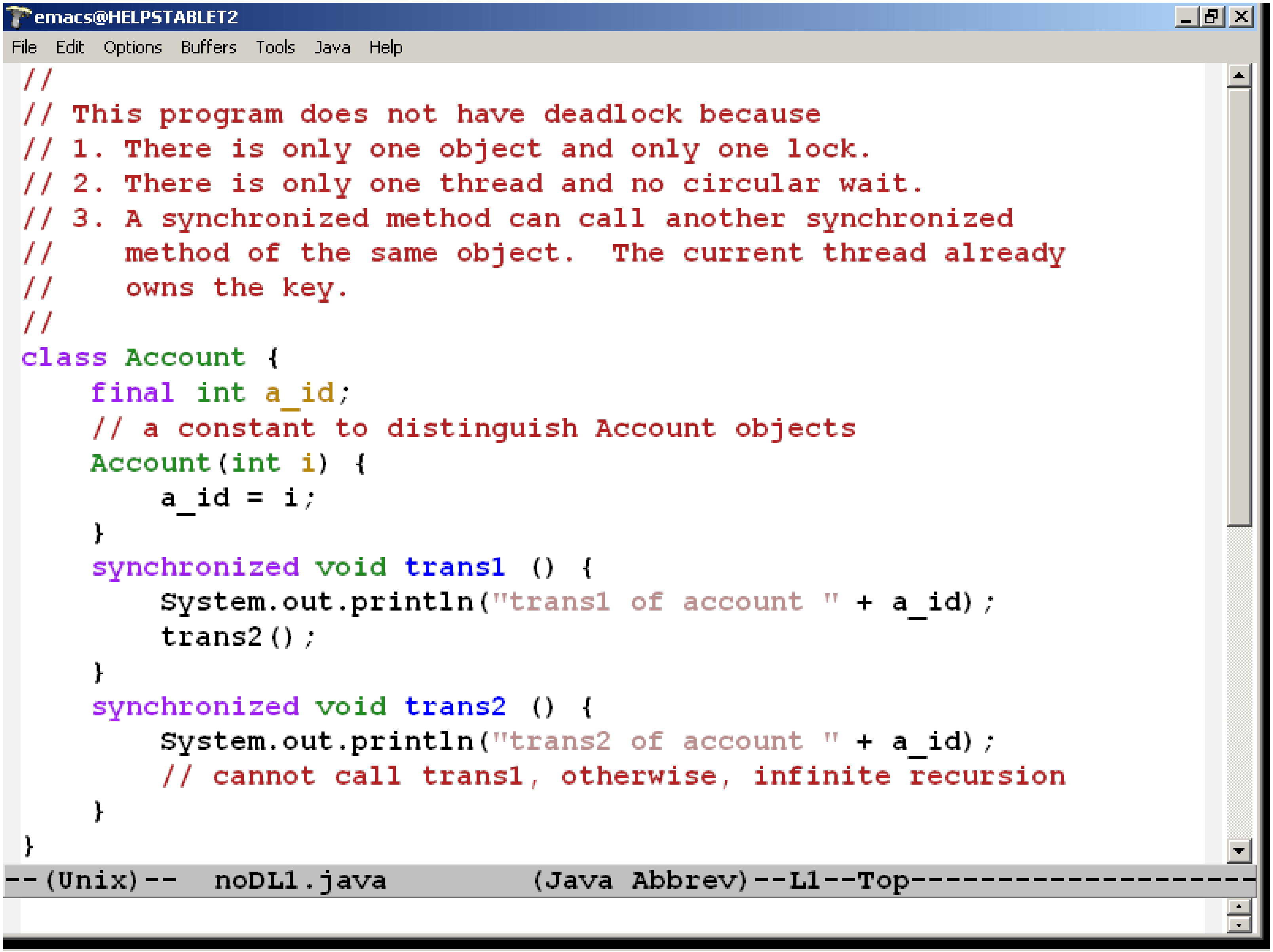


four necessary conditions for deadlock

- 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

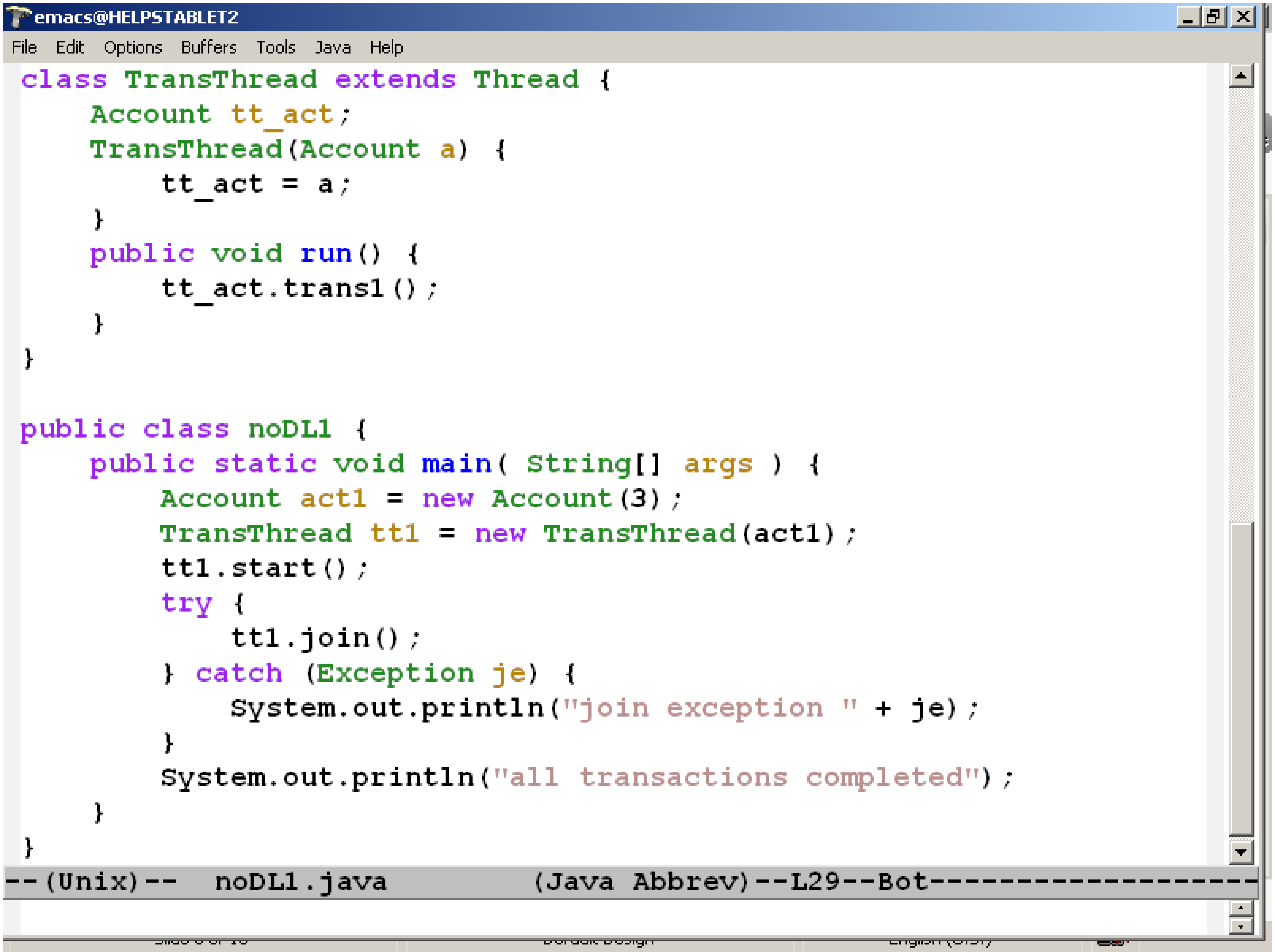


Programs without Deadlock



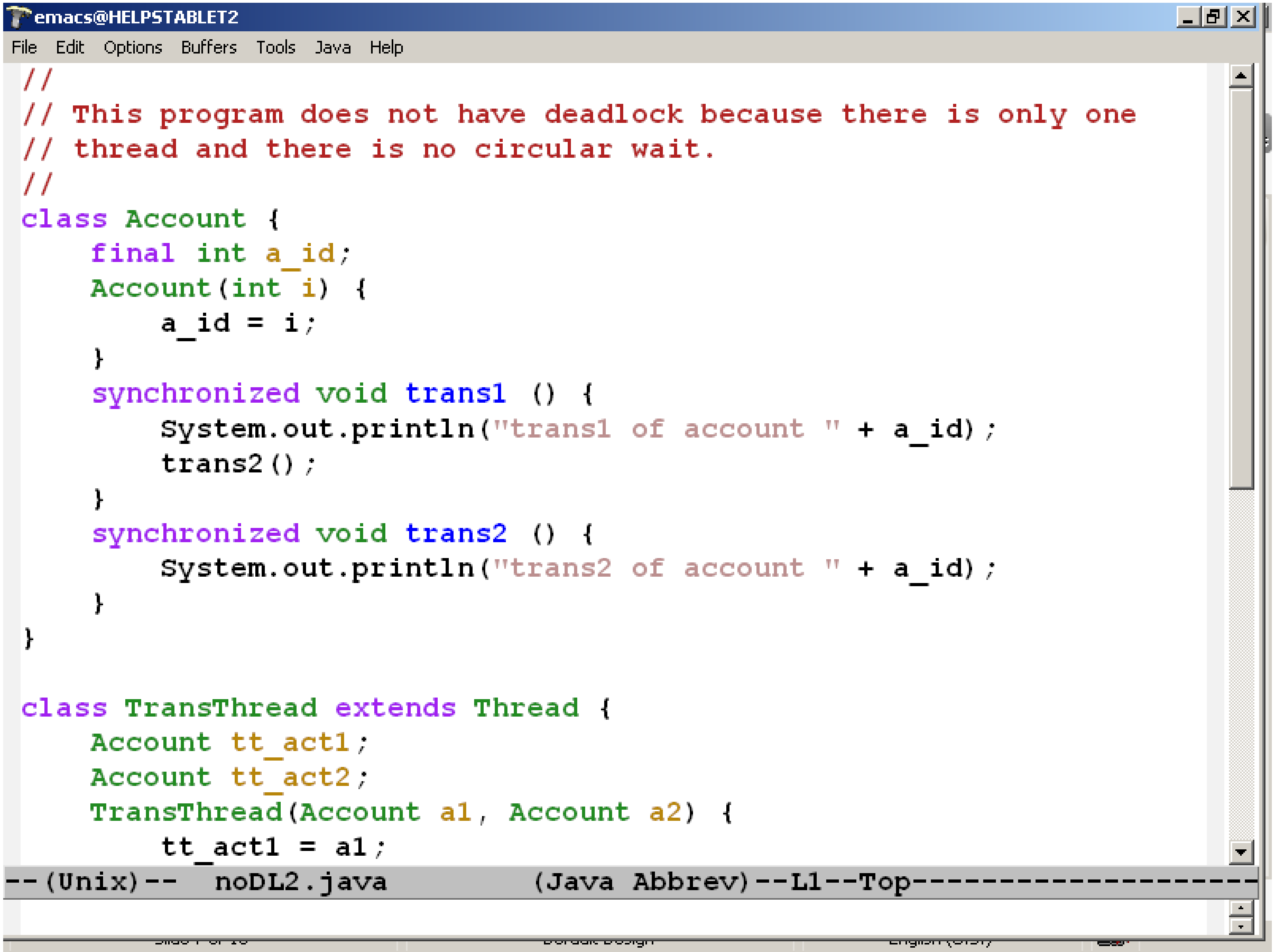
```
//
// This program does not have deadlock because
// 1. There is only one object and only one lock.
// 2. There is only one thread and no circular wait.
// 3. A synchronized method can call another synchronized
//    method of the same object.  The current thread already
//    owns the key.
//
class Account {
    final int a_id;
    // a constant to distinguish Account objects
    Account(int i) {
        a_id = i;
    }
    synchronized void trans1 () {
        System.out.println("trans1 of account " + a_id);
        trans2();
    }
    synchronized void trans2 () {
        System.out.println("trans2 of account " + a_id);
        // cannot call trans1, otherwise, infinite recursion
    }
}
```

--(Unix)--   noDL1.java   (Java Abbrev)--L1--Top-------------------



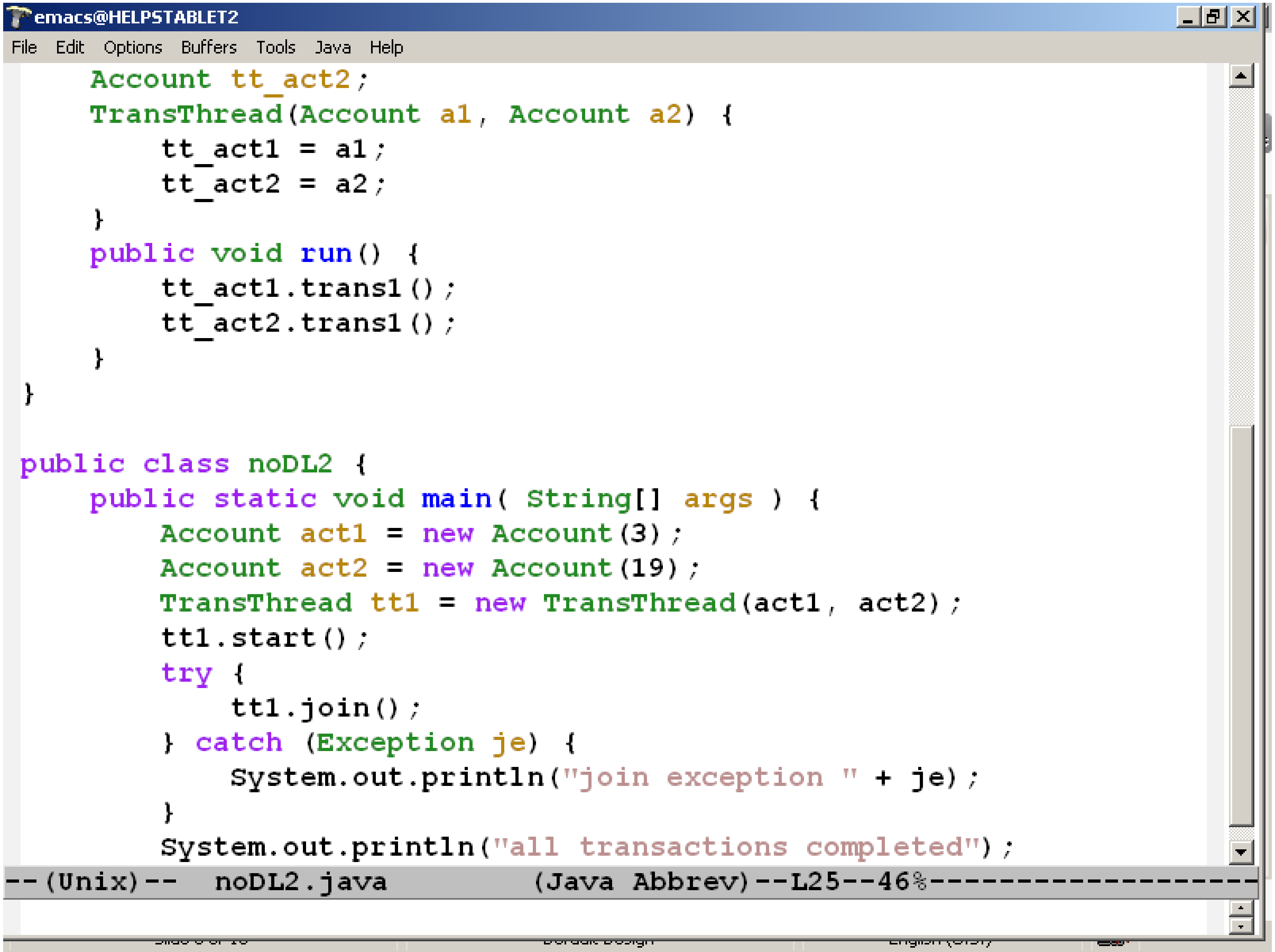
```java
class TransThread extends Thread {
    Account tt_act;
    TransThread(Account a) {
        tt_act = a;
    }
    public void run() {
        tt_act.trans1();
    }
}

public class noDL1 {
    public static void main( String[] args ) {
        Account act1 = new Account(3);
        TransThread tt1 = new TransThread(act1);
        tt1.start();
        try {
            tt1.join();
        } catch (Exception je) {
            System.out.println("join exception " + je);
        }
        System.out.println("all transactions completed");
    }
}
```

```
--(Unix)--  noDL1.java        (Java Abbrev)--L29--Bot------------------
```

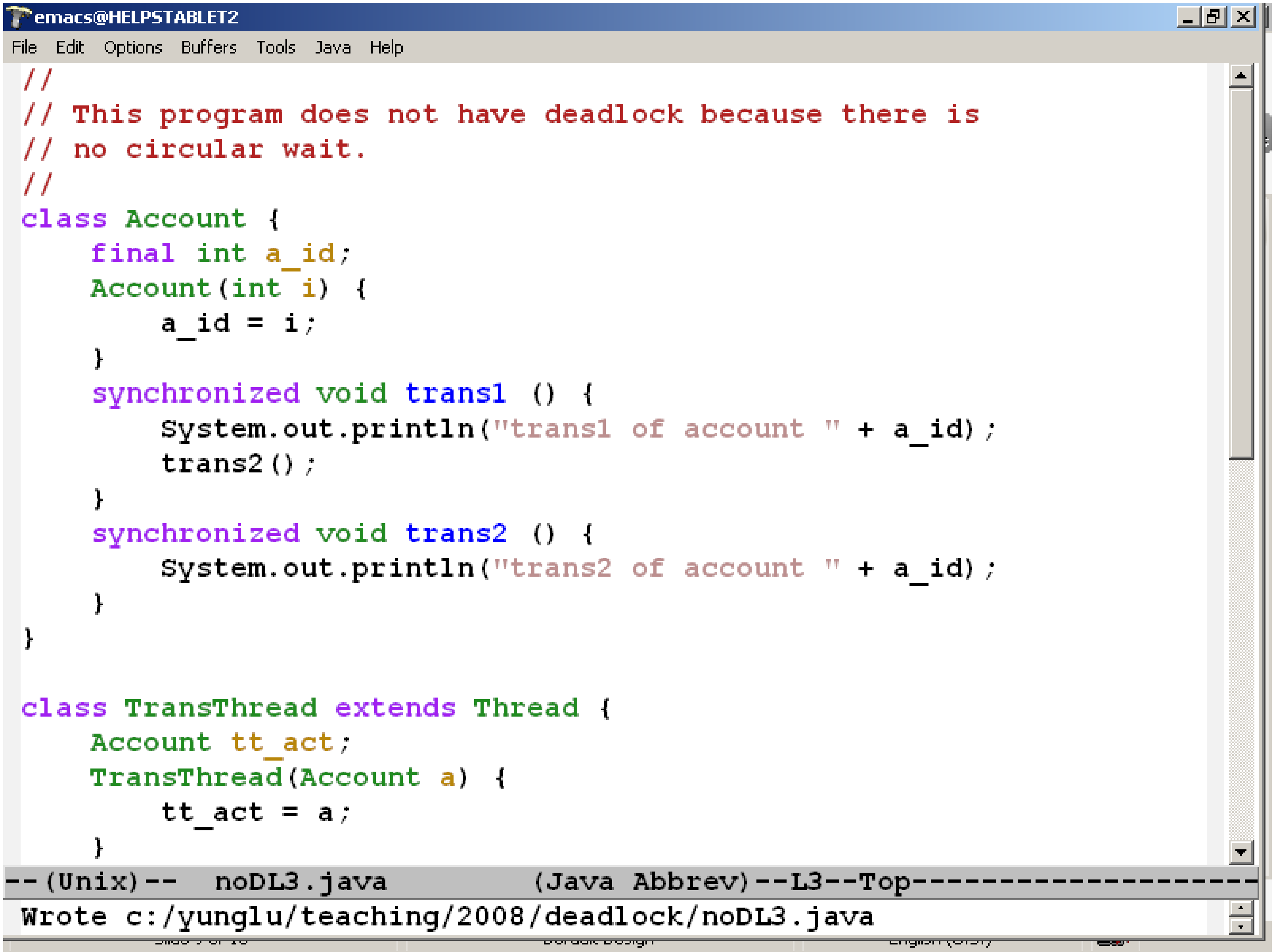


```
//
// This program does not have deadlock because there is only one
// thread and there is no circular wait.
//
class Account {
    final int a_id;
    Account(int i) {
        a_id = i;
    }
    synchronized void trans1 () {
        System.out.println("trans1 of account " + a_id);
        trans2();
    }
    synchronized void trans2 () {
        System.out.println("trans2 of account " + a_id);
    }
}

class TransThread extends Thread {
    Account tt_act1;
    Account tt_act2;
    TransThread(Account a1, Account a2) {
        tt_act1 = a1;
```

```
--(Unix)--  noDL2.java        (Java Abbrev)--L1--Top-------------------
```

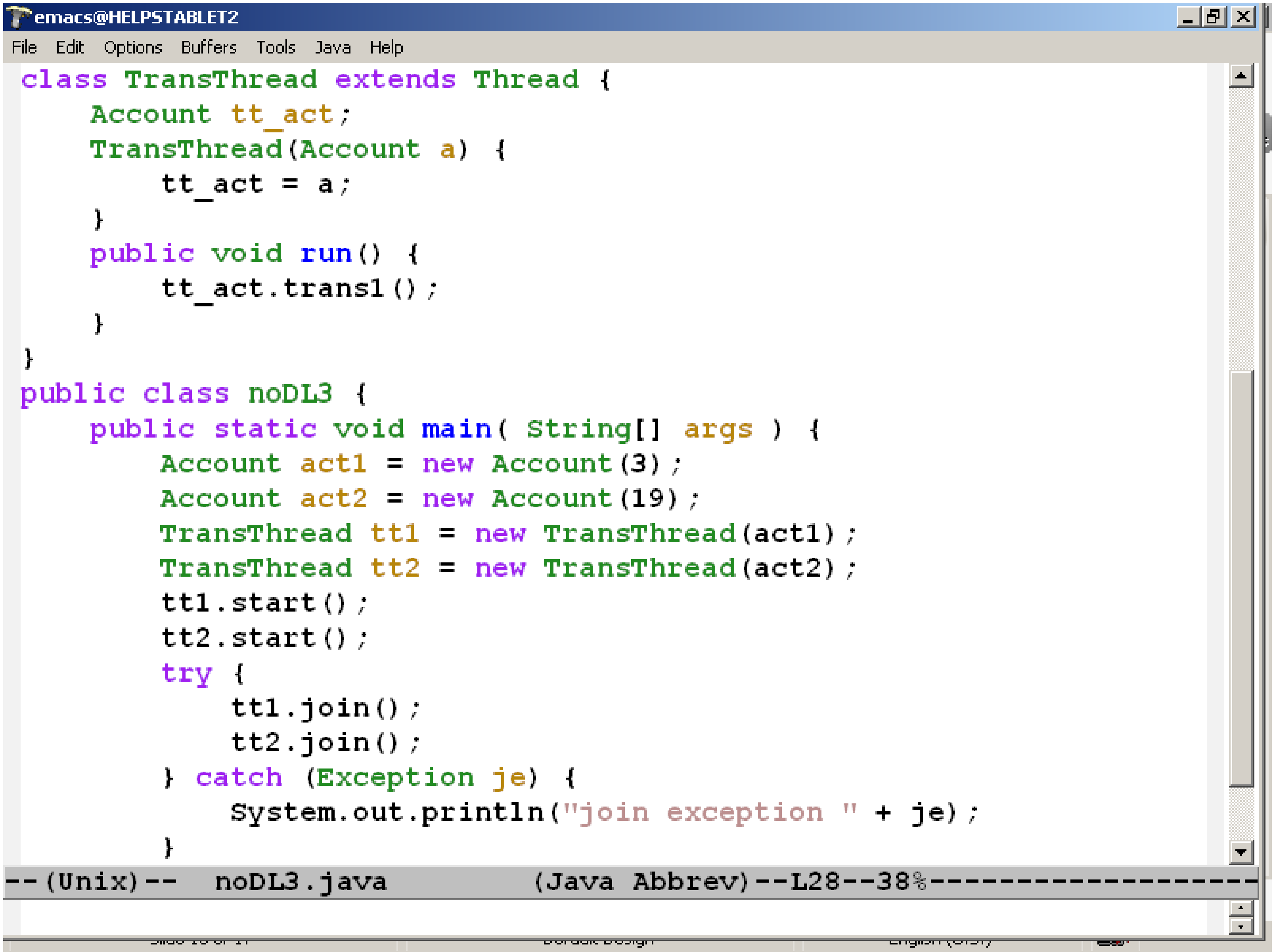

emacs@HELPSTABLET2
File Edit Options Buffers Tools Java Help
--(Unix)--   noDL2.java   (Java Abbrev)--L25--46%------------------

```
    Account tt_act2;
    TransThread(Account a1, Account a2) {
        tt_act1 = a1;
        tt_act2 = a2;
    }
    public void run() {
        tt_act1.trans1();
        tt_act2.trans1();
    }
}

public class noDL2 {
    public static void main( String[] args ) {
        Account act1 = new Account(3);
        Account act2 = new Account(19);
        TransThread tt1 = new TransThread(act1, act2);
        tt1.start();
        try {
            tt1.join();
        } catch (Exception je) {
            System.out.println("join exception " + je);
        }
        System.out.println("all transactions completed");
```



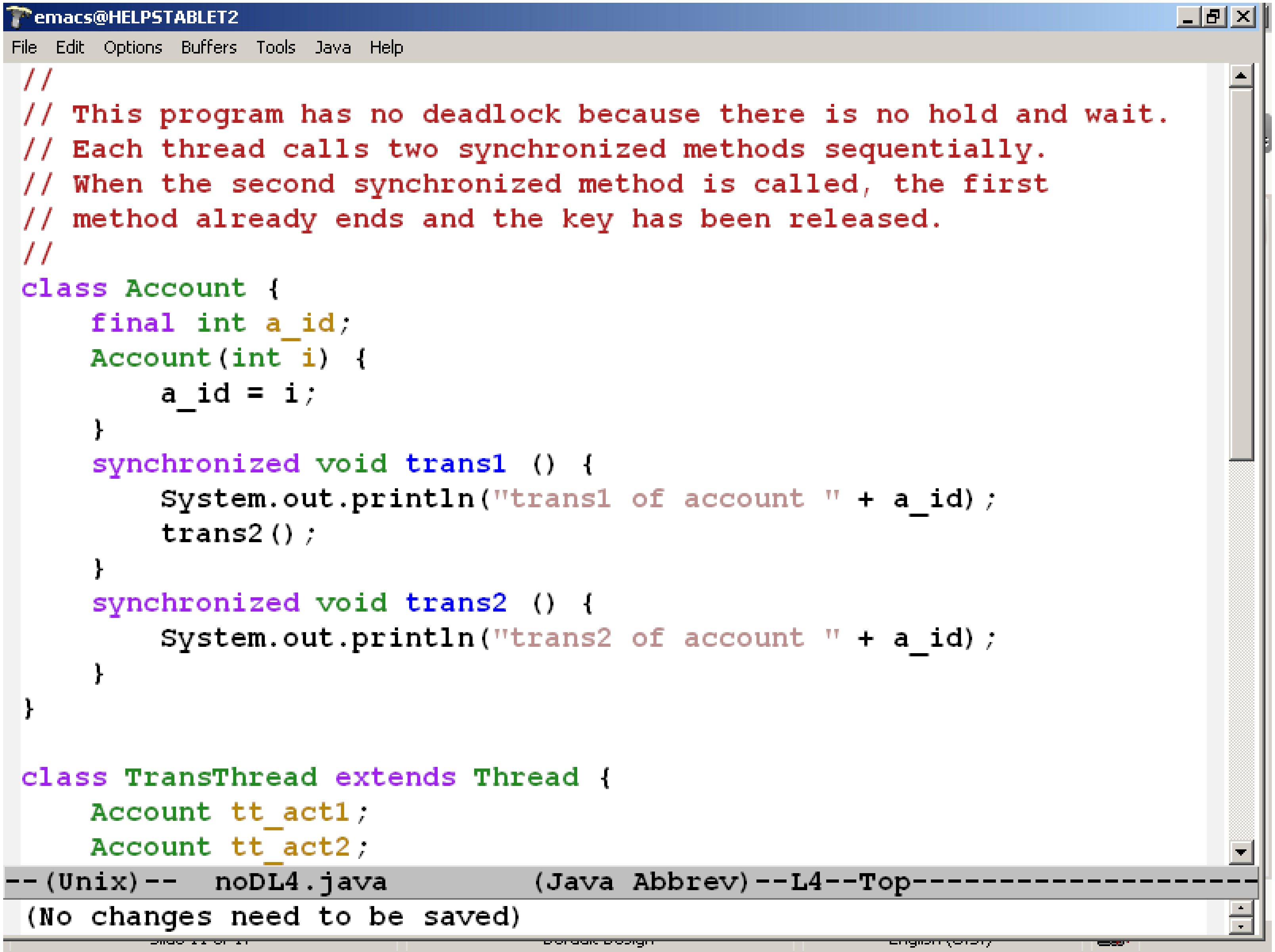
```
//
// This program does not have deadlock because there is
// no circular wait.
//
class Account {
    final int a_id;
    Account(int i) {
        a_id = i;
    }
    synchronized void trans1 () {
        System.out.println("trans1 of account " + a_id);
        trans2();
    }
    synchronized void trans2 () {
        System.out.println("trans2 of account " + a_id);
    }
}

class TransThread extends Thread {
    Account tt_act;
    TransThread(Account a) {
        tt_act = a;
    }
```

```
--(Unix)--  noDL3.java       (Java Abbrev)--L3--Top-------------------
Wrote c:/yunglu/teaching/2008/deadlock/noDL3.java
```



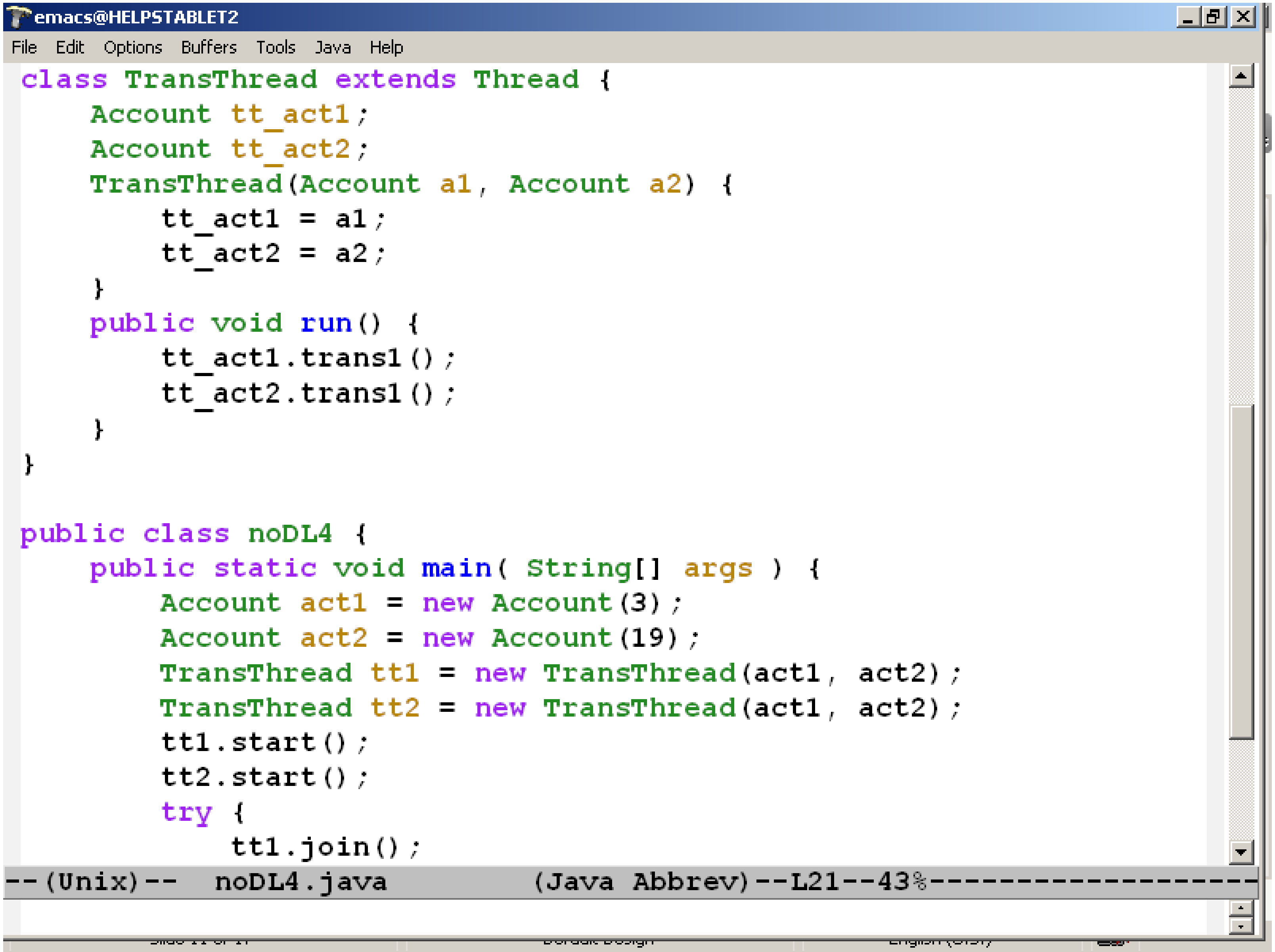
```
class TransThread extends Thread {
    Account tt_act;
    TransThread(Account a) {
        tt_act = a;
    }
    public void run() {
        tt_act.trans1();
    }
}
public class noDL3 {
    public static void main( String[] args ) {
        Account act1 = new Account(3);
        Account act2 = new Account(19);
        TransThread tt1 = new TransThread(act1);
        TransThread tt2 = new TransThread(act2);
        tt1.start();
        tt2.start();
        try {
            tt1.join();
            tt2.join();
        } catch (Exception je) {
            System.out.println("join exception " + je);
        }
```

```
--(Unix)--  noDL3.java      (Java Abbrev)--L28--38%------------------
```

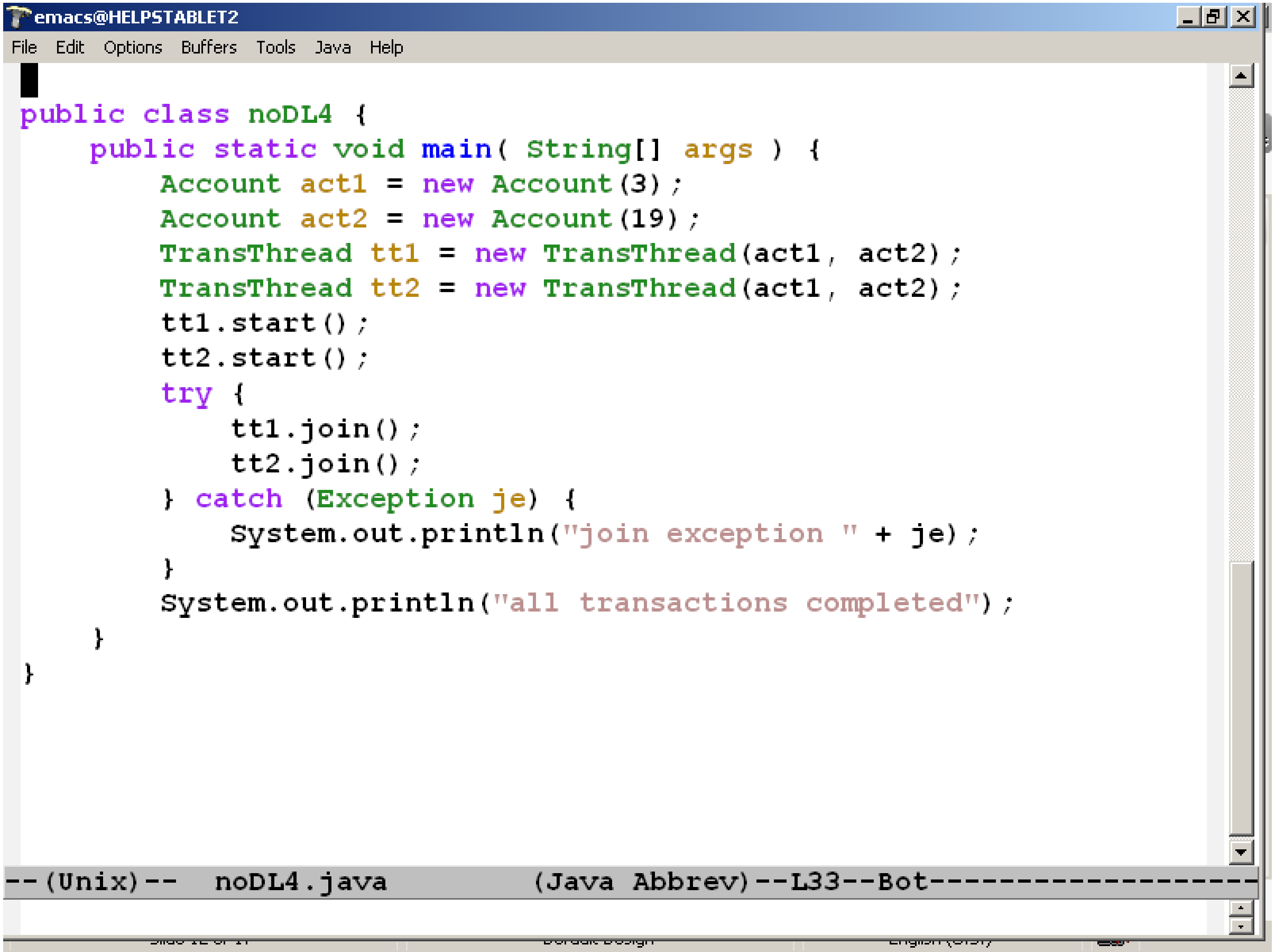


```
//
// This program has no deadlock because there is no hold and wait.
// Each thread calls two synchronized methods sequentially.
// When the second synchronized method is called, the first
// method already ends and the key has been released.
//
class Account {
    final int a_id;
    Account(int i) {
        a_id = i;
    }
    synchronized void trans1 () {
        System.out.println("trans1 of account " + a_id);
        trans2();
    }
    synchronized void trans2 () {
        System.out.println("trans2 of account " + a_id);
    }
}

class TransThread extends Thread {
    Account tt_act1;
    Account tt_act2;
```

```
--(Unix)--  noDL4.java       (Java Abbrev)--L4--Top-------------------
(No changes need to be saved)
```


emacs@HELPSTABLET2
File Edit Options Buffers Tools Java Help
--(Unix)--   noDL4.java   (Java Abbrev)--L21--43%------------------

```java
class TransThread extends Thread {
    Account tt_act1;
    Account tt_act2;
    TransThread(Account a1, Account a2) {
        tt_act1 = a1;
        tt_act2 = a2;
    }
    public void run() {
        tt_act1.trans1();
        tt_act2.trans1();
    }
}

public class noDL4 {
    public static void main( String[] args ) {
        Account act1 = new Account(3);
        Account act2 = new Account(19);
        TransThread tt1 = new TransThread(act1, act2);
        TransThread tt2 = new TransThread(act1, act2);
        tt1.start();
        tt2.start();
        try {
            tt1.join();
```

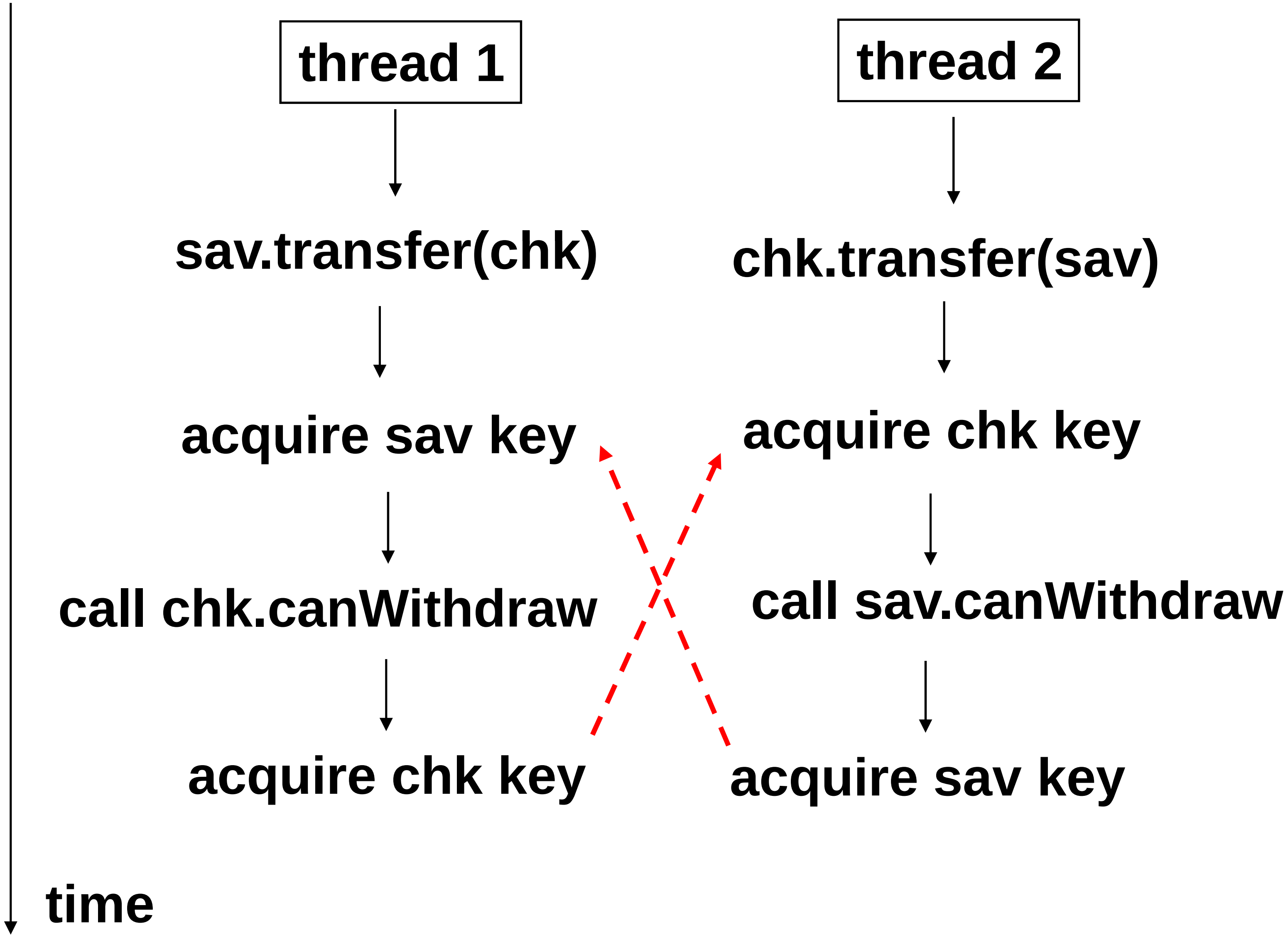

emacs@HELPSTABLET2
File Edit Options Buffers Tools Java Help
--(Unix)--   noDL4.java   (Java Abbrev)--L33--Bot------------------

```
public class noDL4 {
    public static void main( String[] args ) {
        Account act1 = new Account(3);
        Account act2 = new Account(19);
        TransThread tt1 = new TransThread(act1, act2);
        TransThread tt2 = new TransThread(act1, act2);
        tt1.start();
        tt2.start();
        try {
            tt1.join();
            tt2.join();
        } catch (Exception je) {
            System.out.println("join exception " + je);
        }
        System.out.println("all transactions completed");
    }
}
```

Java Example of Deadlock

John has a check and a saving accounts with a bank. He transferred $500 from the check account to the saving account for a high interest rate. Later in the day, he wrote a check and transferred $200 from the saving account back to the checking account. In the evening, he went to an ATM to withdraw cash. The ATM said, “Account not Found.” What happened?

```
class Account {
    private int balance;
    synchronized void transfer (Account source, int amount) {
          if (source.canWithdraw(amount)) {
                    source.withdraw(amount);
                    balance += amount;
          }
   }
    synchronized boolean canWithdraw(amount) {
          if (balance >= amount) { return true; }
          return false;
    }
}
```



thread 1

sav.transfer(chk)

acquire sav key

call chk.canWithdraw

acquire chk key

thread 2

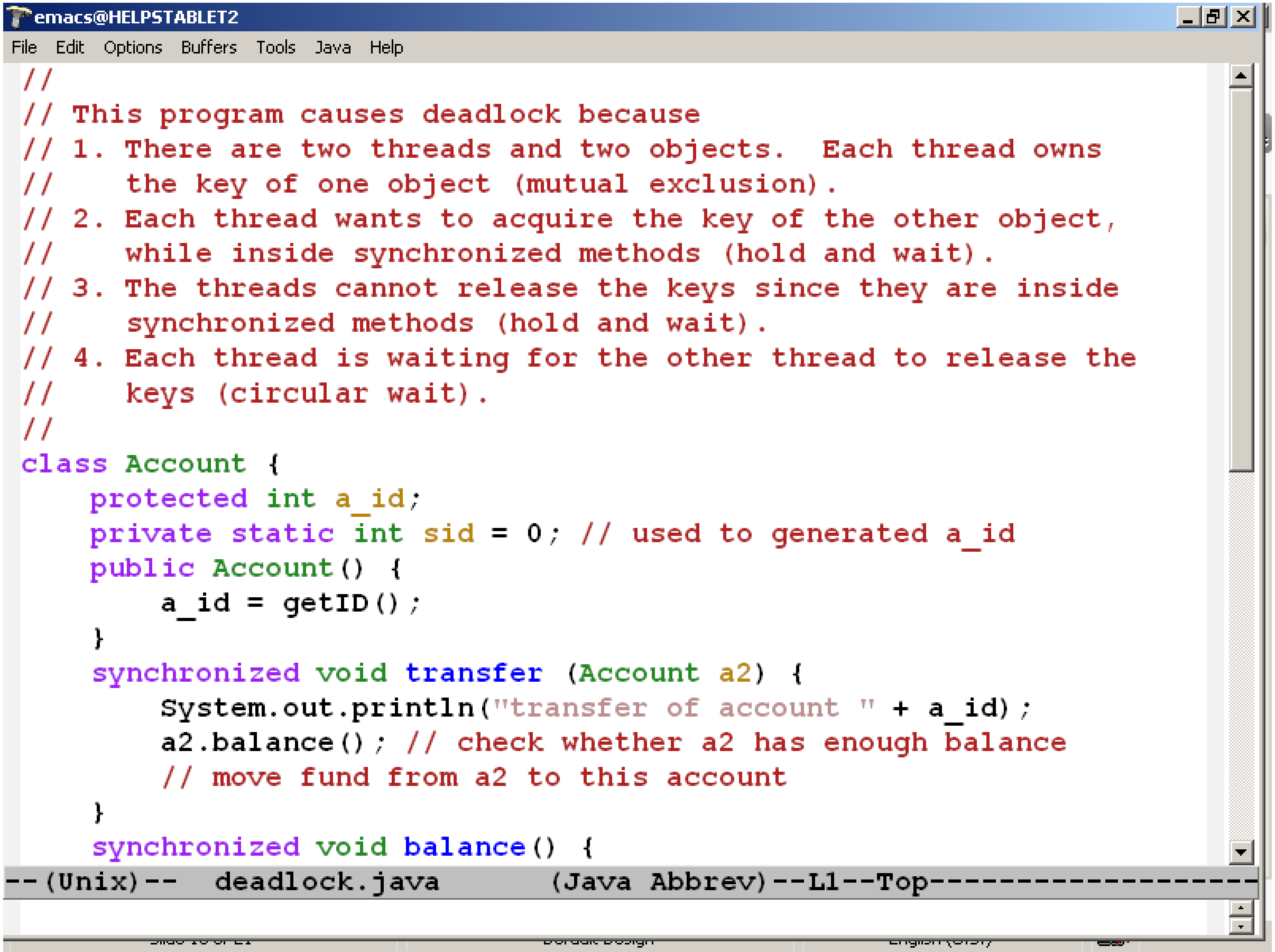
chk.transfer(sav)

acquire chk key

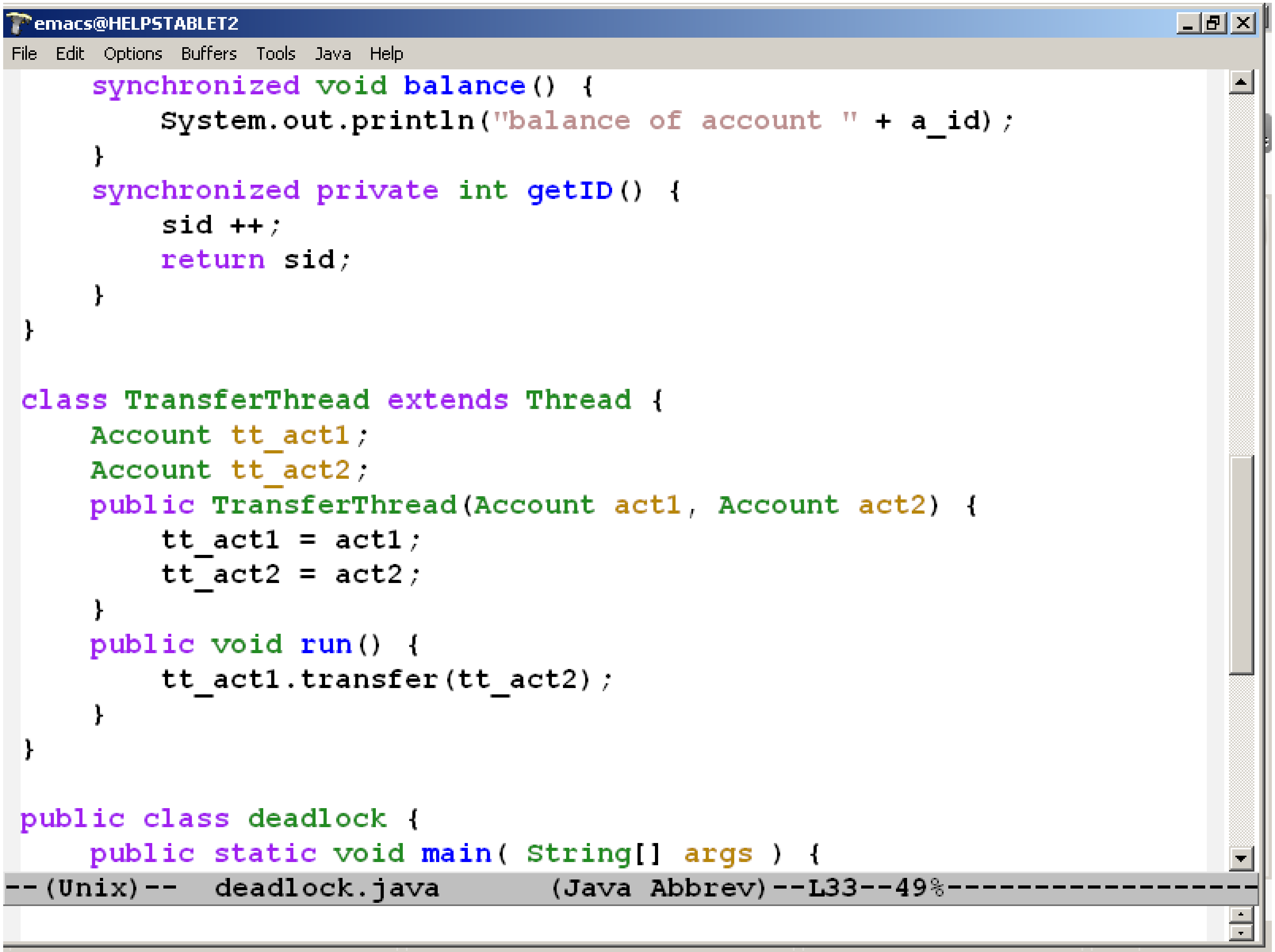
call sav.canWithdraw

acquire sav key

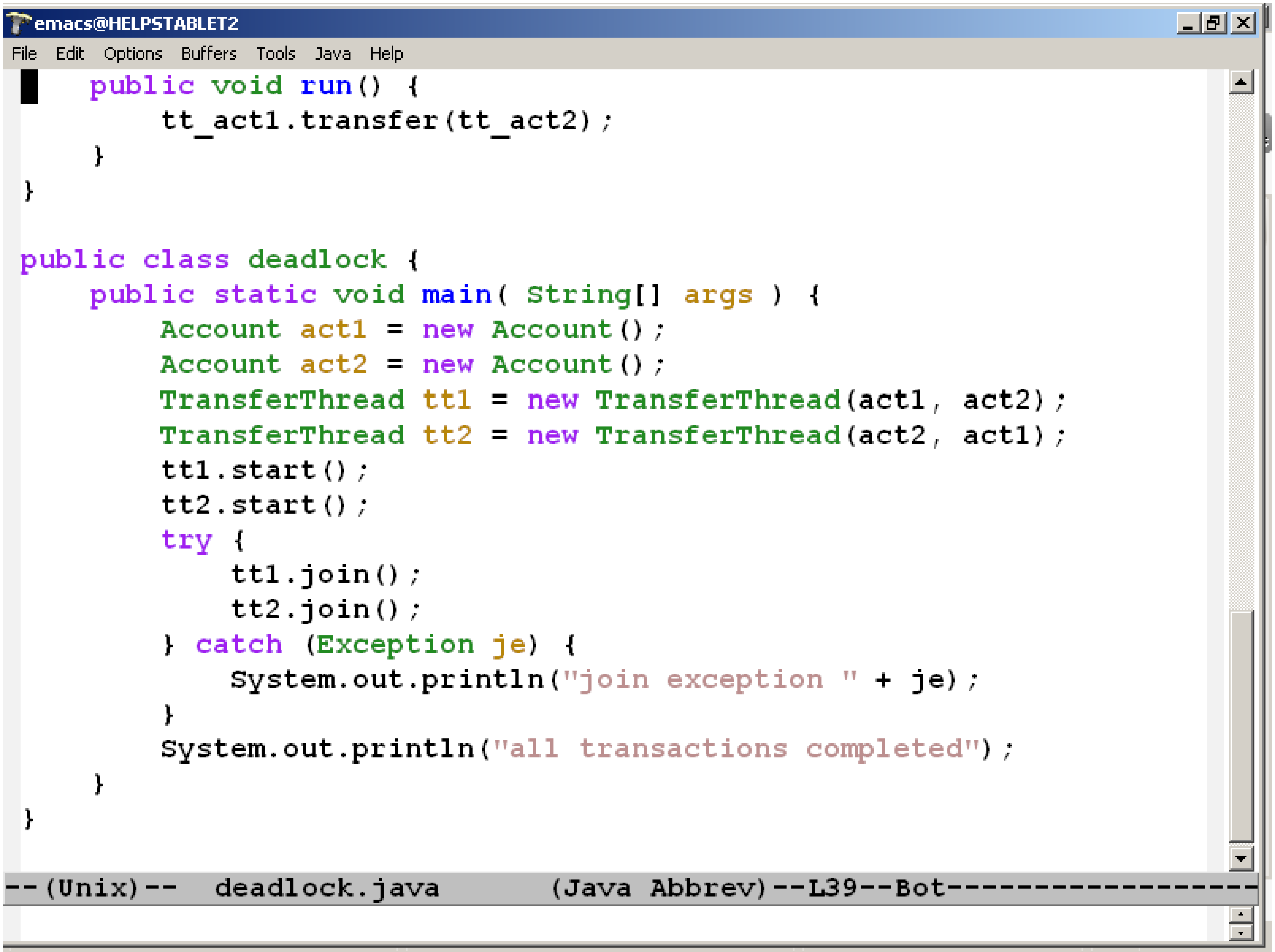
time


emacs@HELPSTABLET2
File Edit Options Buffers Tools Java Help
--(Unix)--   deadlock.java   (Java Abbrev)--L1--Top-------------------

```java
//
// This program causes deadlock because
// 1. There are two threads and two objects.  Each thread owns
//    the key of one object (mutual exclusion).
// 2. Each thread wants to acquire the key of the other object,
//    while inside synchronized methods (hold and wait).
// 3. The threads cannot release the keys since they are inside
//    synchronized methods (hold and wait).
// 4. Each thread is waiting for the other thread to release the
//    keys (circular wait).
//
class Account {
    protected int a_id;
    private static int sid = 0; // used to generated a_id
    public Account() {
        a_id = getID();
    }
    synchronized void transfer (Account a2) {
        System.out.println("transfer of account " + a_id);
        a2.balance(); // check whether a2 has enough balance
        // move fund from a2 to this account
    }
    synchronized void balance() {
```


emacs@HELPSTABLET2
File
Edit
Options
Buffers
Tools
Java
Help
--(Unix)--   deadlock.java   (Java Abbrev)--L33--49%-----------------

```
    synchronized void balance() {
        System.out.println("balance of account " + a_id);
    }
    synchronized private int getID() {
        sid ++;
        return sid;
    }
}

class TransferThread extends Thread {
    Account tt_act1;
    Account tt_act2;
    public TransferThread(Account act1, Account act2) {
        tt_act1 = act1;
        tt_act2 = act2;
    }
    public void run() {
        tt_act1.transfer(tt_act2);
    }
}

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



```
    public void run() {
        tt_act1.transfer(tt_act2);
    }
}

public class deadlock {
    public static void main( String[] args ) {
        Account act1 = new Account();
        Account act2 = new Account();
        TransferThread tt1 = new TransferThread(act1, act2);
        TransferThread tt2 = new TransferThread(act2, act1);
        tt1.start();
        tt2.start();
        try {
            tt1.join();
            tt2.join();
        } catch (Exception je) {
            System.out.println("join exception " + je);
        }
        System.out.println("all transactions completed");
    }
}
```

```
--(Unix)--   deadlock.java      (Java Abbrev)--L39--Bot-----------------
```

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

Thread Performance

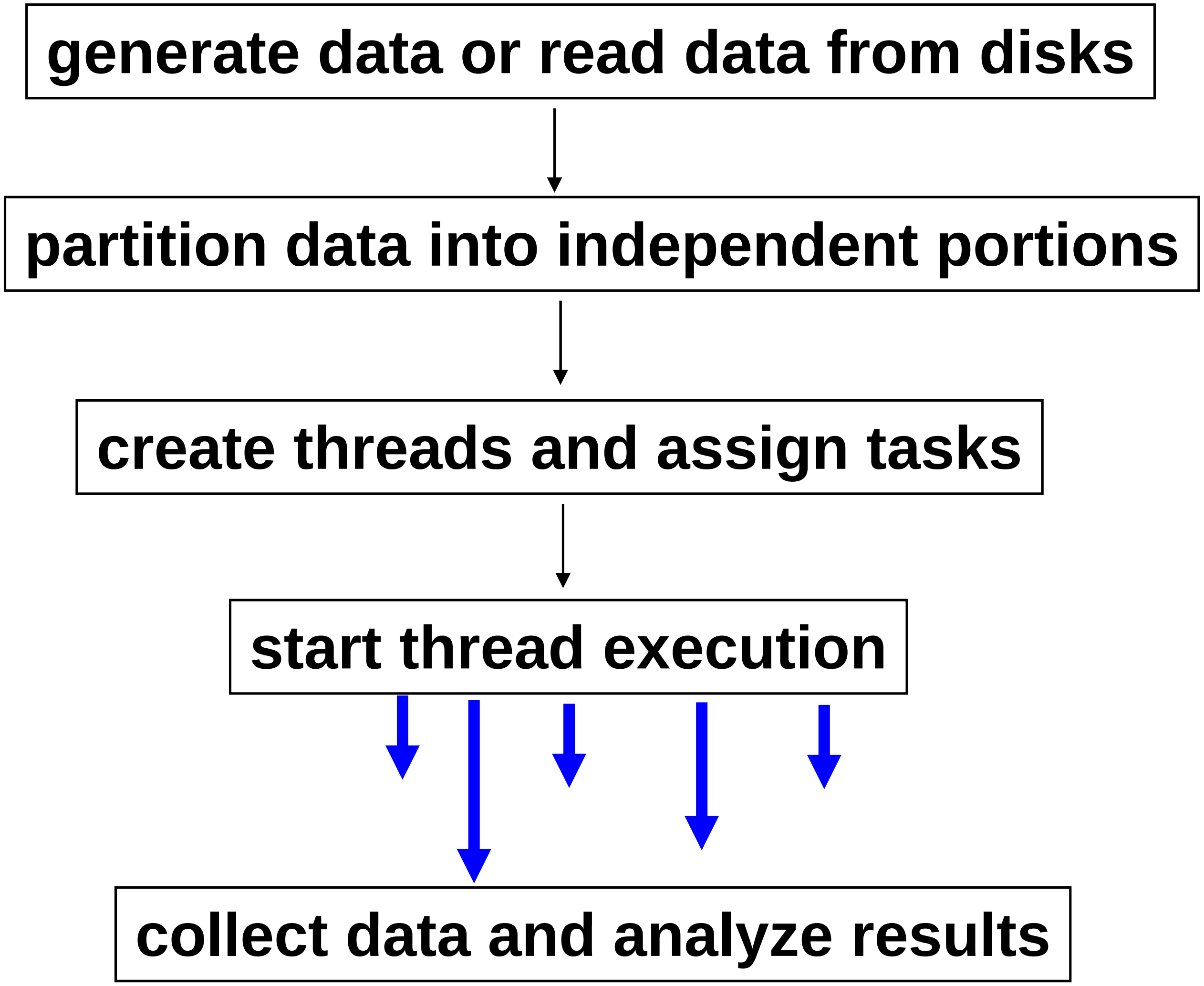
Yung-Hsiang Lu
yunglu@purdue.edu

Performance Measurement

```
ts1 = current time;
    execute the program without creating threads;
ts2 = current time;

tp1 = current time;
    execute the program with multiple threads
tp2 = current time;
```

$$\text{improvement} = \frac{\text{ts2 - ts1}}{\text{tp2} - \text{tp1}}$$

Structure of Multithread Programs



generate data or read data from disks

partition data into independent portions

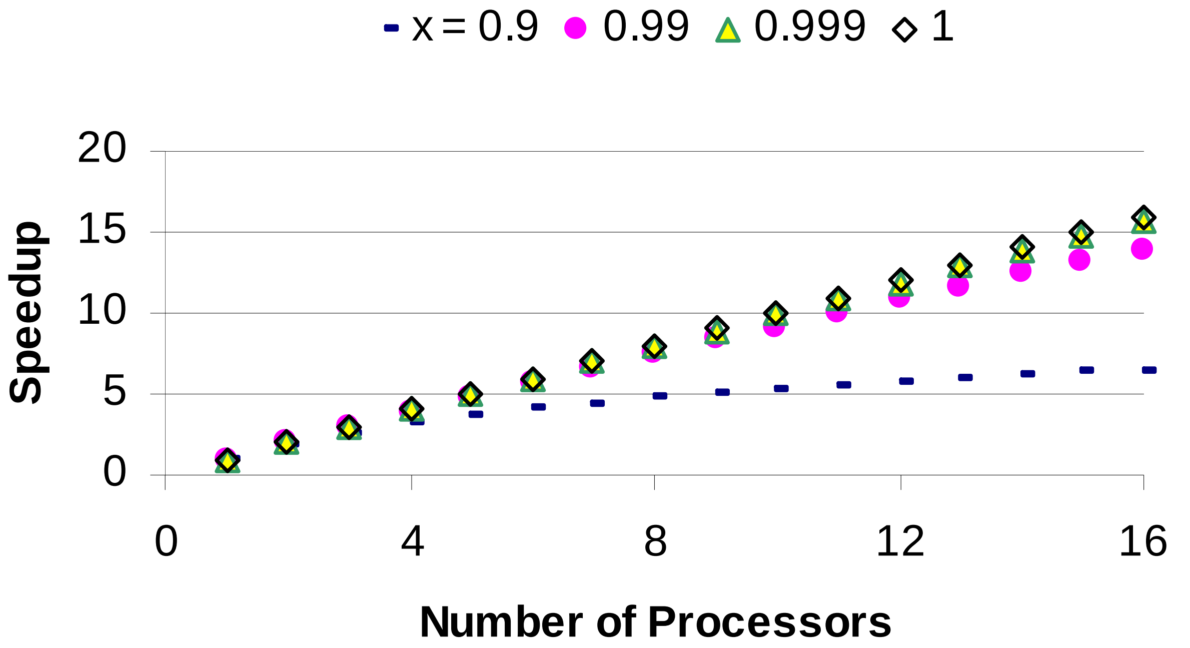
create threads and assign tasks

start thread execution

collect data and analyze results

Amdahl’s Law

- If a program has x (%) parallel code, 1-x sequential code
- the speedup of using n threads (no sync) is $\dfrac{1}{1\text{-}x+\dfrac{x}{n}}$
- if x = 0.9 and n →∞, speedup = 10



Multi-Thread = High Performance?

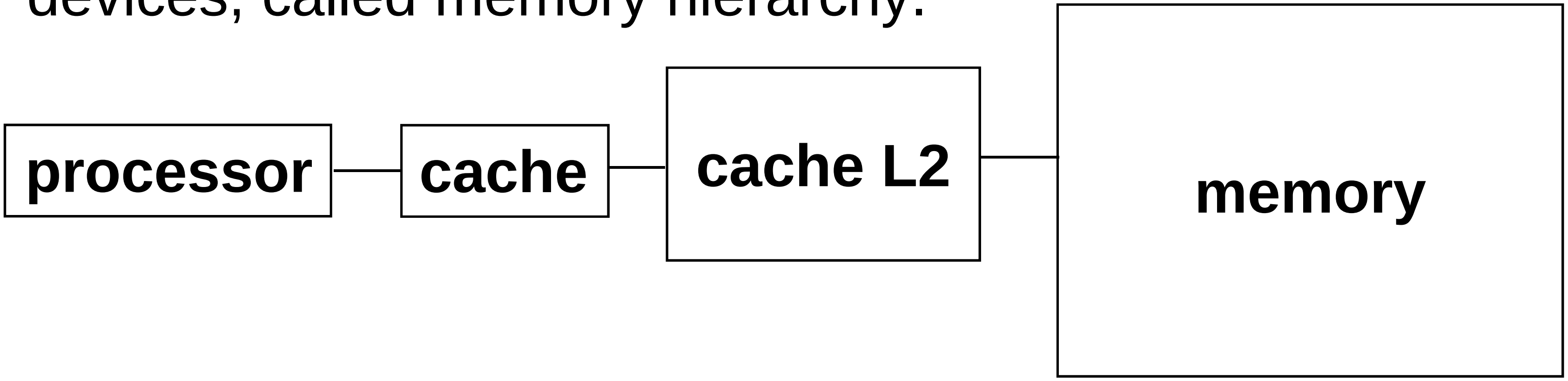
- multi-thread ≠ high performance (faster)
- If a program is IO-bounded, multi-thread (or multi-core) does not really help.
- Finding sufficient parallelism (make x closer to 1) can be difficulty.
- Reduce the sequential parts as much as possible
  - read data from multiple, parallel sources
  - partition data as they arrive
  - create long-living threads and reuse them
  - reduce competitions of mutex keys

Superlinear Speedup

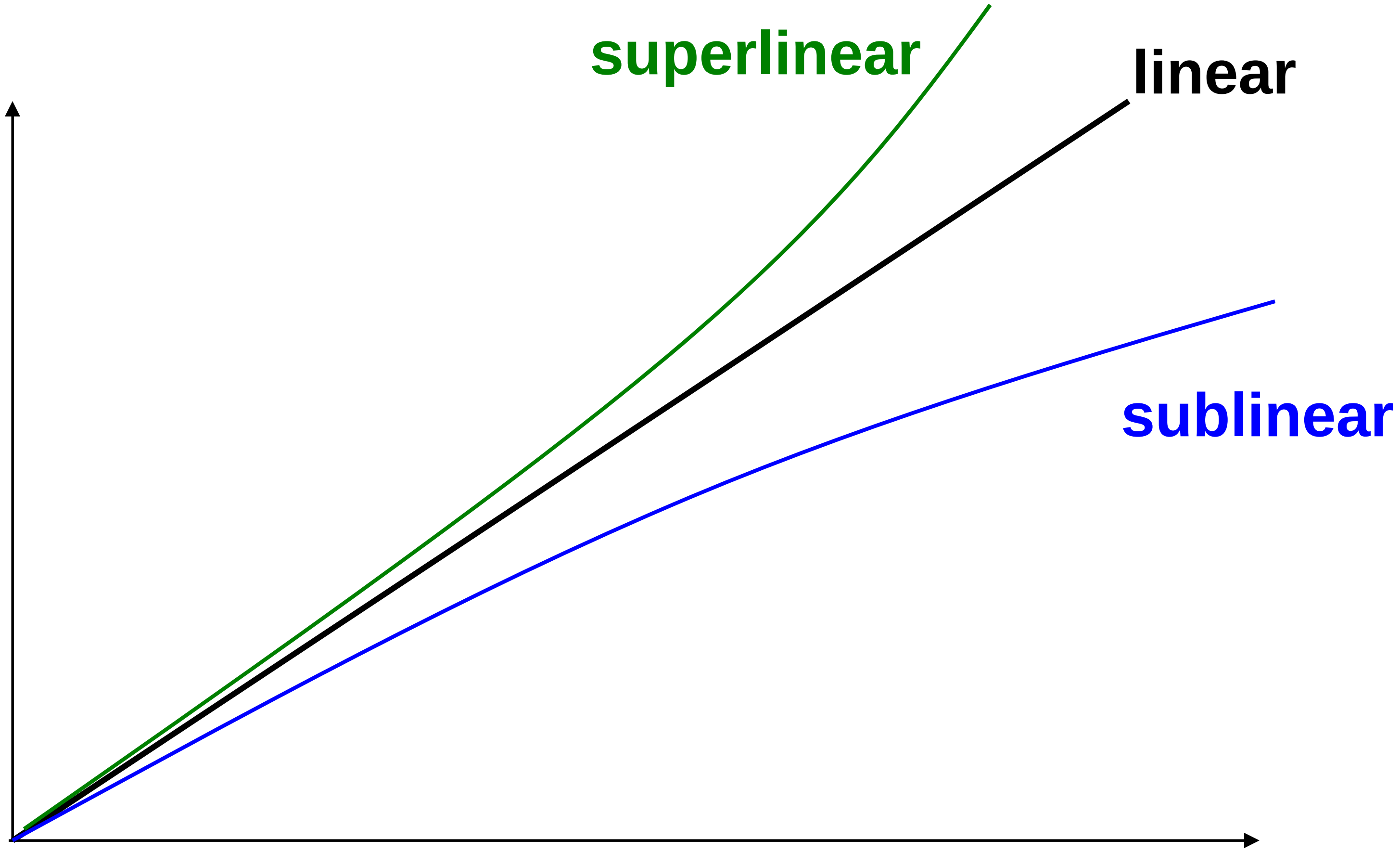
- Sometimes, a multithread program's performance exceeds the number of processors.

$$\frac{\text{execution time of single thread}}{\text{execution time of multiple thread}} > \text{number of processors}$$

- All modern processors are built upon a series of storage devices, called memory hierarchy.



Superlinear or Sublinear



Memory Hierarchy

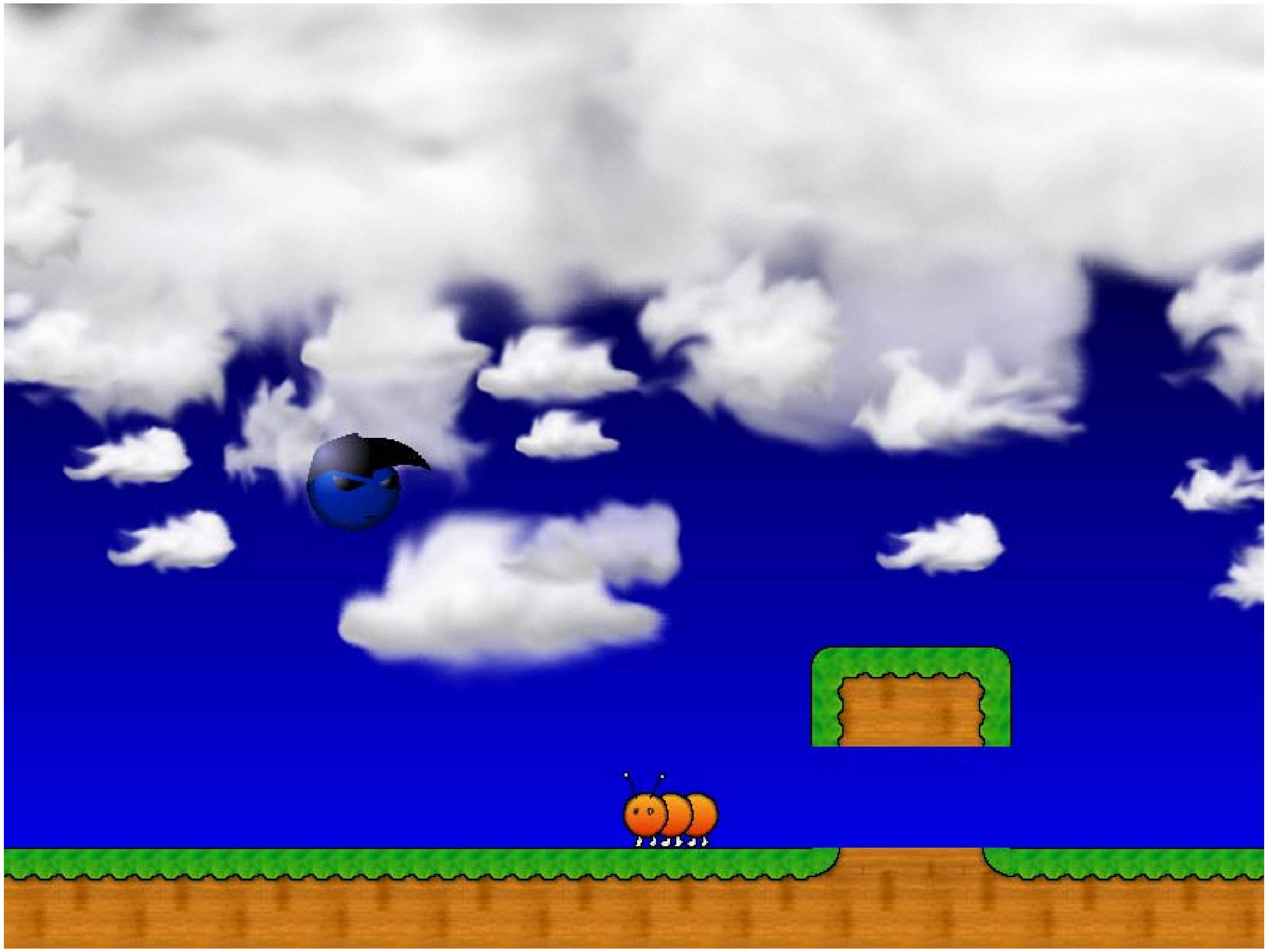
- Smaller and faster memory (cache) is installed closer to the processor. When data cannot fit into L1 cache, the data are stored in L2 cache, then memory.
- Multiple processors can have larger L1 cache collectively to accommodate more data for faster access. As a result, the program is faster.
- Multi-thread programs can also cause frequent data contention and trigger expensive (i.e. slow) consistency checking code. When this happens, the program can be much slower.

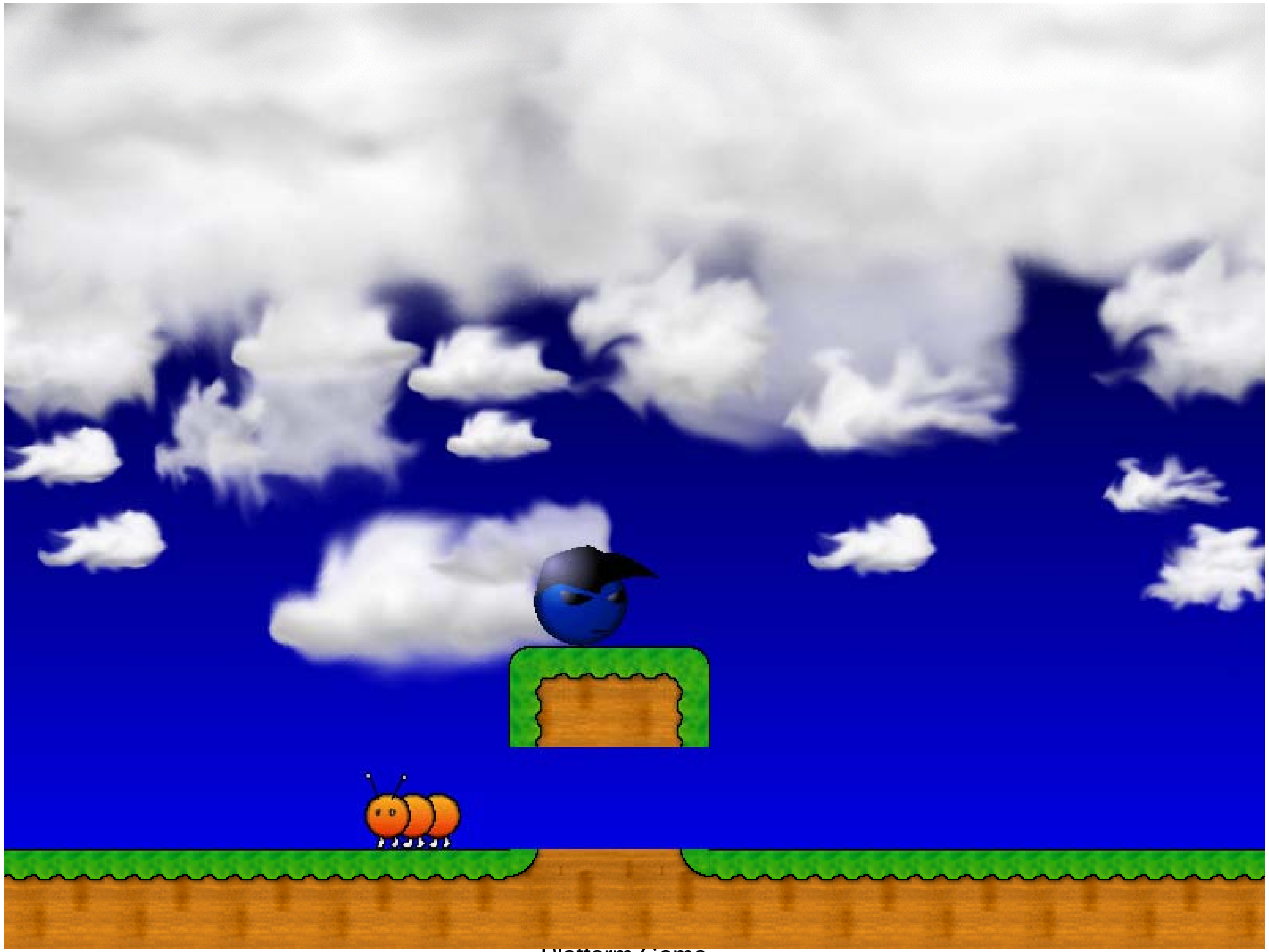
ECE 462
Object-Oriented Programming using C++ and Java

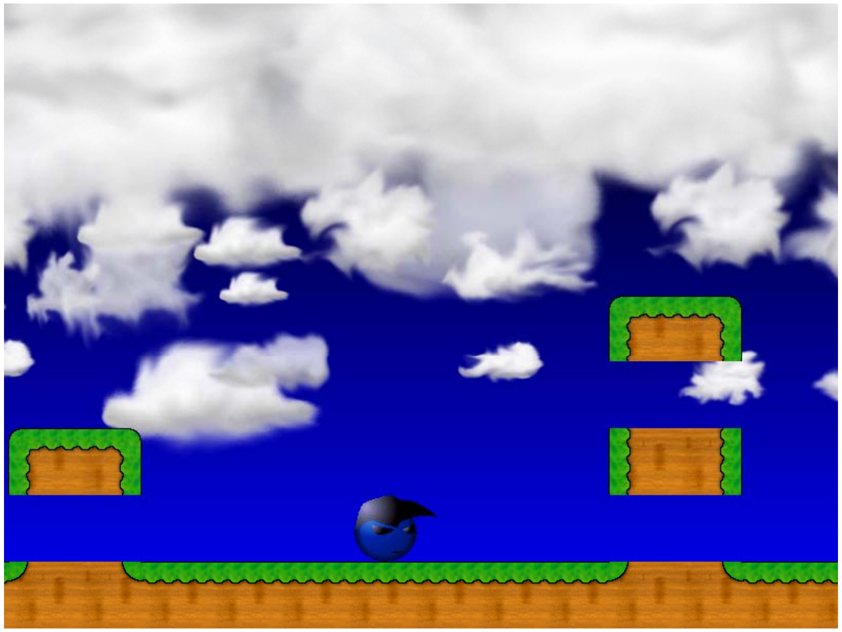
Tile-Based Platform Game

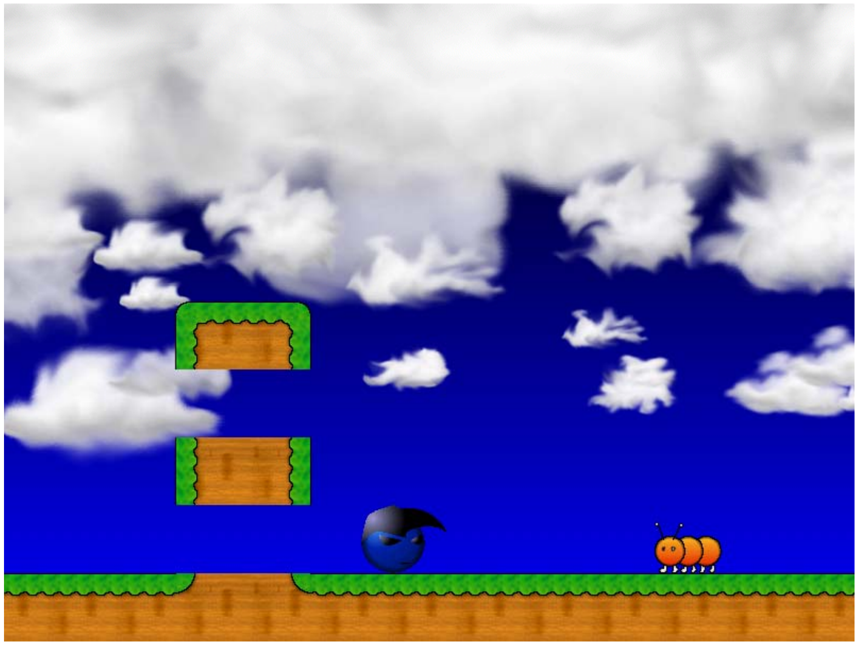
Yung-Hsiang Lu

yunglu@purdue.edu







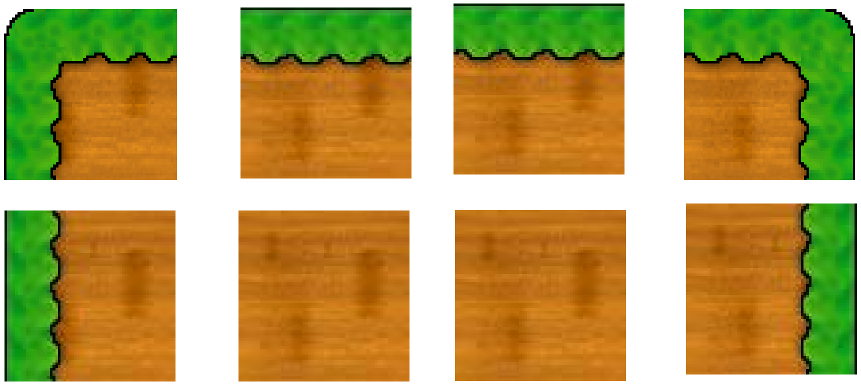


orm G

Changing Background

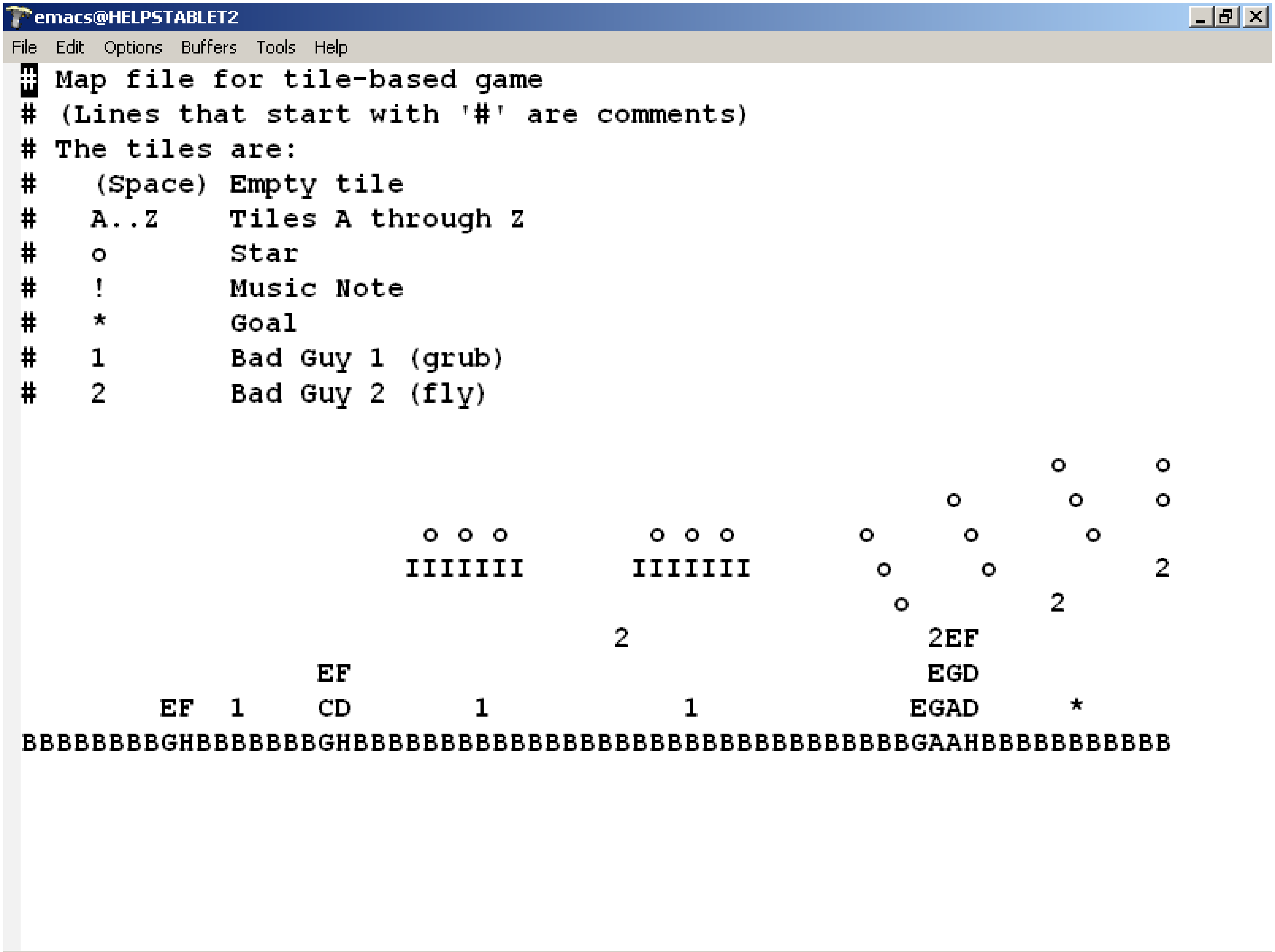
- background map: To provide a changing background without a very large actual background
- sky

- tiles



Tile Map

- The titles do not need individual copies of the images
- collision detection based on the type of a tile
- complex map ⇒ a special map editor
- simple map ⇒ text editor sufficient
- Each object's location is relative to the map (not to the screen) so that a player can scroll left or right easily.

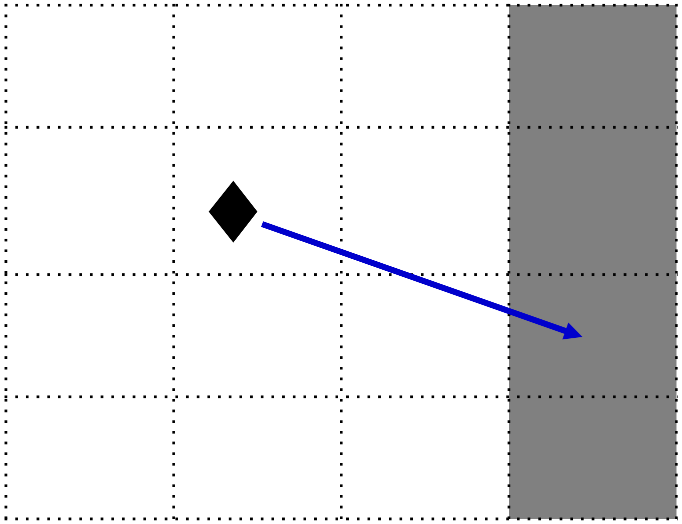


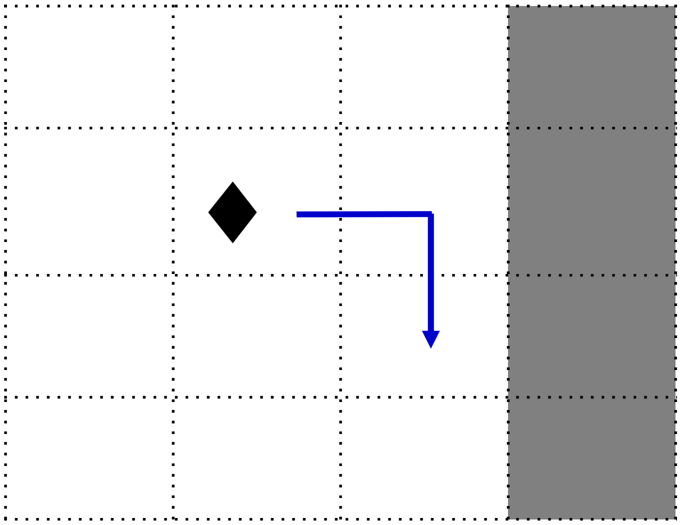
```
# Map file for tile-based game
# (Lines that start with '#' are comments)
# The tiles are:
#   (Space) Empty tile
#   A..Z    Tiles A through Z
#   o       Star
#   !       Music Note
#   *       Goal
#   1       Bad Guy 1 (grub)
#   2       Bad Guy 2 (fly)

                                                           o     o
                                                     o      o    o
                       o o o        o o o       o     o      o
                      IIIIIII      IIIIIII       o     o         2
                                                  o        2
                                  2                 2EF
                 EF                                 EGD
        EF  1    CD       1           1            EGAD     *
BBBBBBBBGHBBBBBBBGHBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBGAAHBBBBBBBBBBB
```

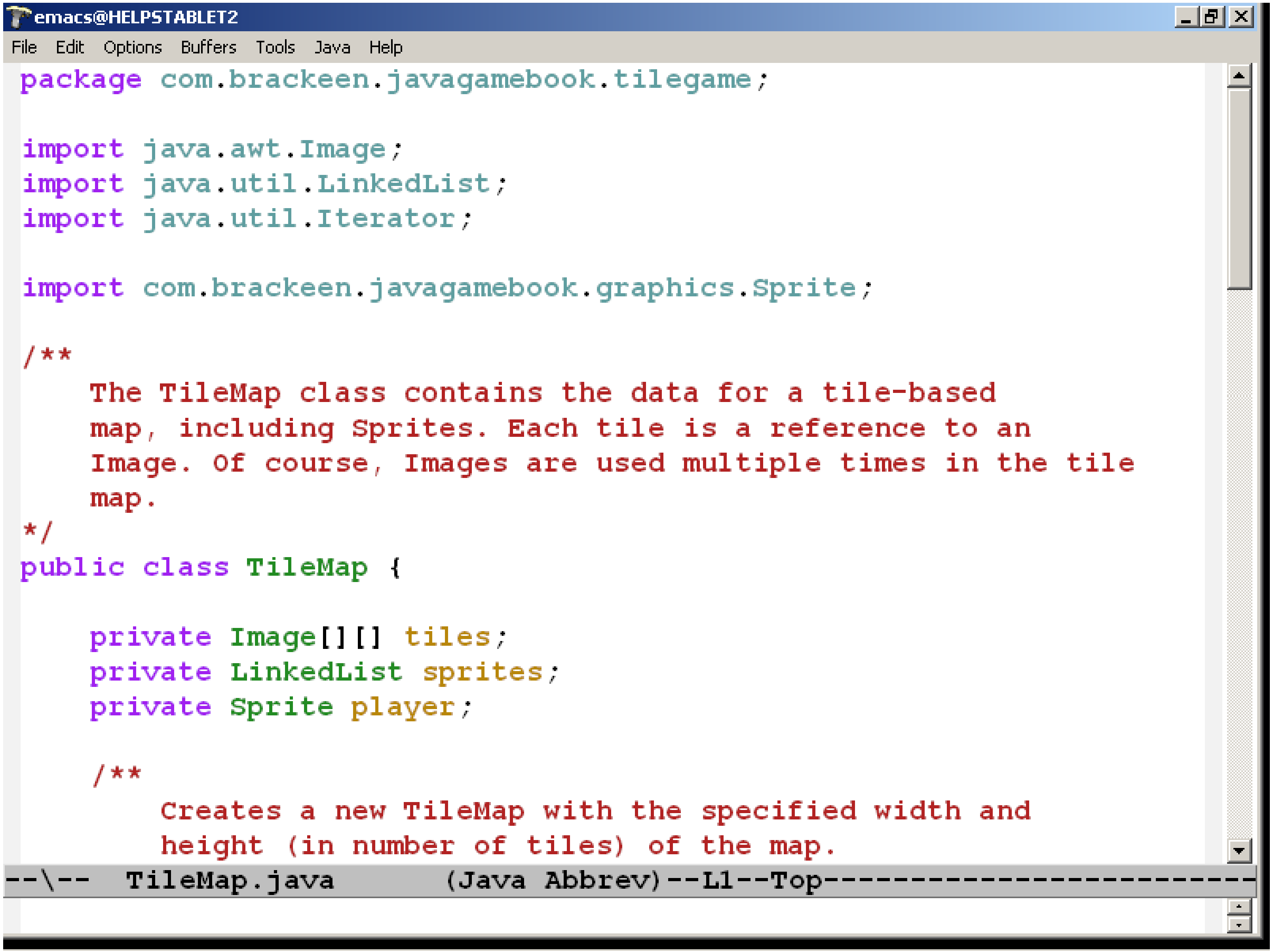
Tile-Based Collision Detection

If the sprite’s next location is (x,y), checking whether collision occurs is easy:

tileMap.get(x,y) ⇒ if a tile exists, collision

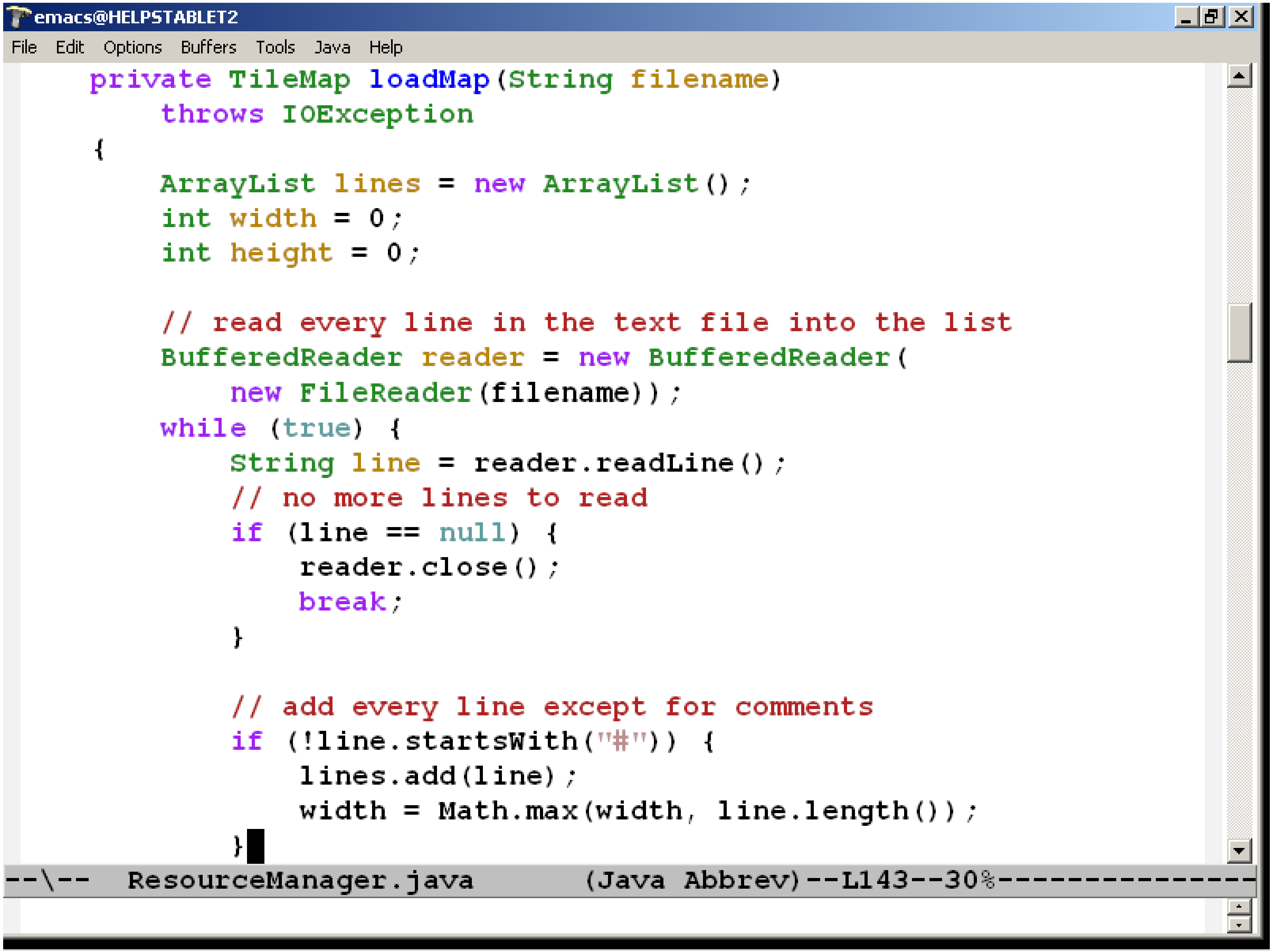






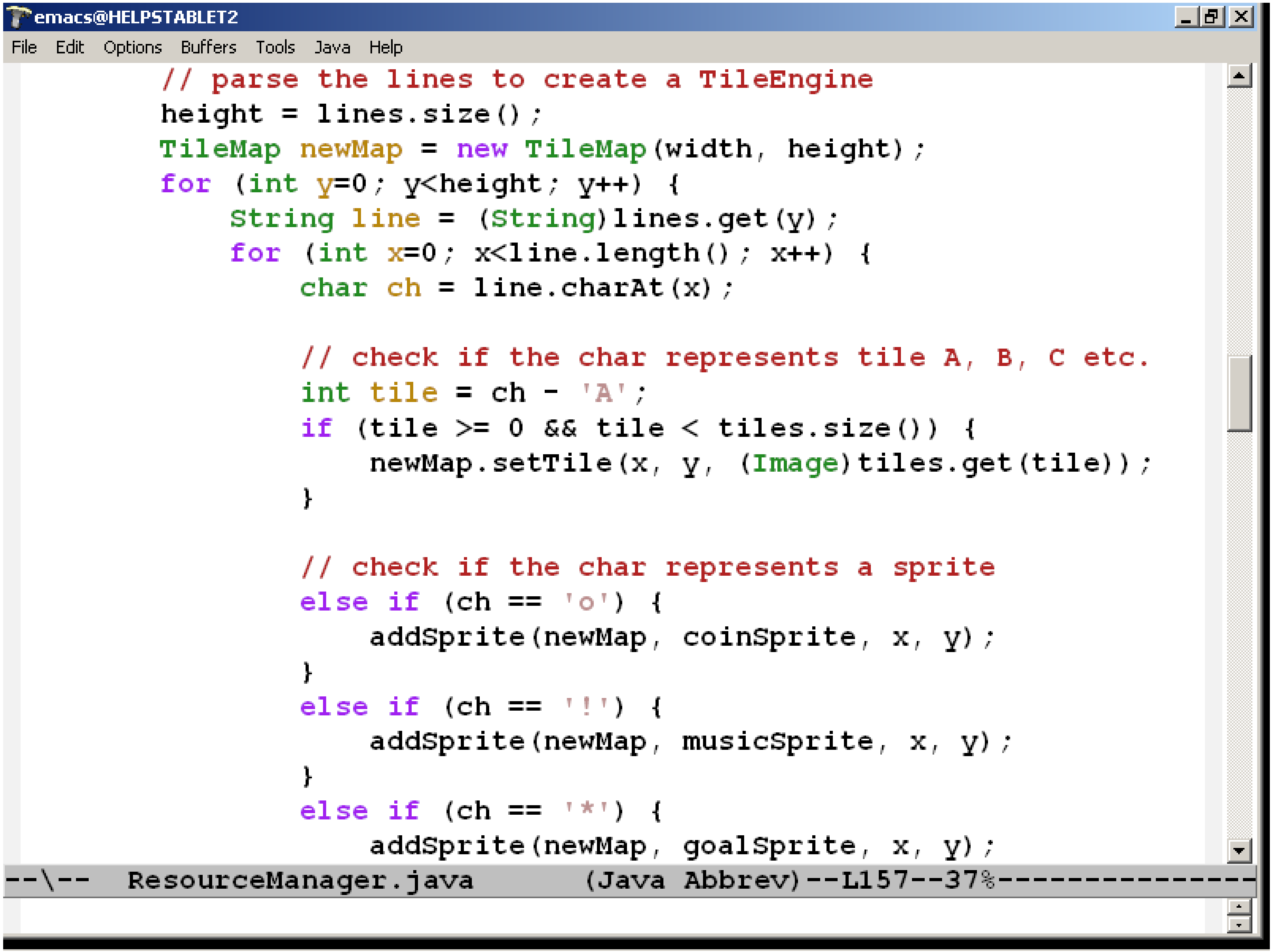
```java
package com.brackeen.javagamebook.tilegame;

import java.awt.Image;
import java.util.LinkedList;
import java.util.Iterator;

import com.brackeen.javagamebook.graphics.Sprite;

/**
    The TileMap class contains the data for a tile-based
    map, including Sprites. Each tile is a reference to an
    Image. Of course, Images are used multiple times in the tile
    map.
*/
public class TileMap {

    private Image[][] tiles;
    private LinkedList sprites;
    private Sprite player;

    /**
        Creates a new TileMap with the specified width and
        height (in number of tiles) of the map.
```

```
--\--  TileMap.java       (Java Abbrev)--L1--Top-------------------------
```



```
    private TileMap loadMap(String filename)
        throws IOException
    {
        ArrayList lines = new ArrayList();
        int width = 0;
        int height = 0;

        // read every line in the text file into the list
        BufferedReader reader = new BufferedReader(
            new FileReader(filename));
        while (true) {
            String line = reader.readLine();
            // no more lines to read
            if (line == null) {
                reader.close();
                break;
            }

            // add every line except for comments
            if (!line.startsWith("#")) {
                lines.add(line);
                width = Math.max(width, line.length());
            }
```

```
--\--  ResourceManager.java      (Java Abbrev)--L143--30%--------------
```



```
        // parse the lines to create a TileEngine
        height = lines.size();
        TileMap newMap = new TileMap(width, height);
        for (int y=0; y<height; y++) {
            String line = (String)lines.get(y);
            for (int x=0; x<line.length(); x++) {
                char ch = line.charAt(x);

                // check if the char represents tile A, B, C etc.
                int tile = ch - 'A';
                if (tile >= 0 && tile < tiles.size()) {
                    newMap.setTile(x, y, (Image)tiles.get(tile));
                }

                // check if the char represents a sprite
                else if (ch == 'o') {
                    addSprite(newMap, coinSprite, x, y);
                }
                else if (ch == '!') {
                    addSprite(newMap, musicSprite, x, y);
                }
                else if (ch == '*') {
                    addSprite(newMap, goalSprite, x, y);
--\--  ResourceManager.java       (Java Abbrev)--L157--37%----------------
```