

ECE 462

Object-Oriented Programming

using C++ and Java

Design Parallel Programs

Yung-Hsiang Lu
yunlu@purdue.edu

Compare Java and C++ (Qt) Threads

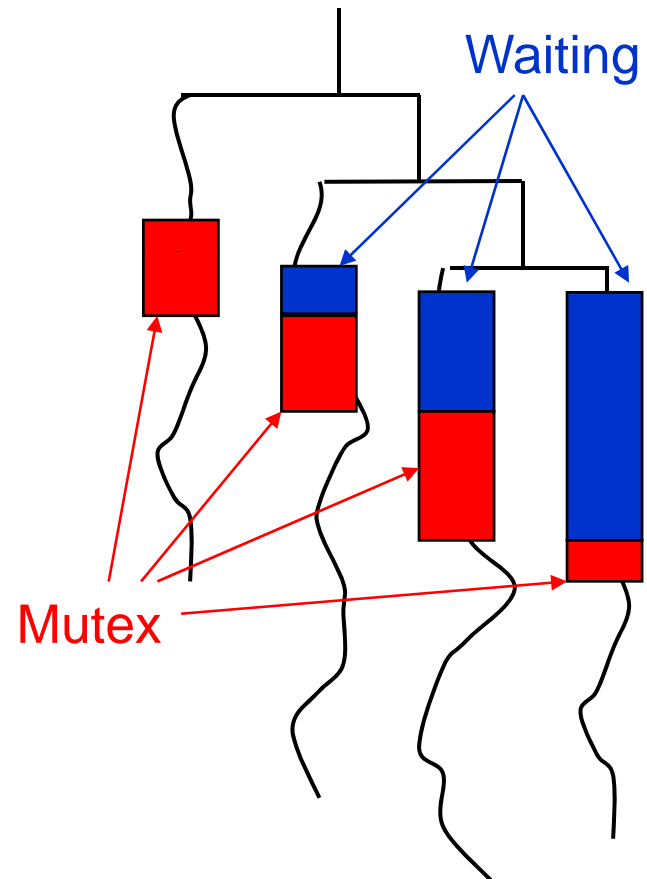
Java	C++
extends Thread or implements Runnable	public QThread
public void run()	public: void run()
thread1.start();	thread1.start();
synchronized	mutex.lock() ... mutex.unlock()
try { wait (); } catch ...	cond.wait(&mutex);

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.



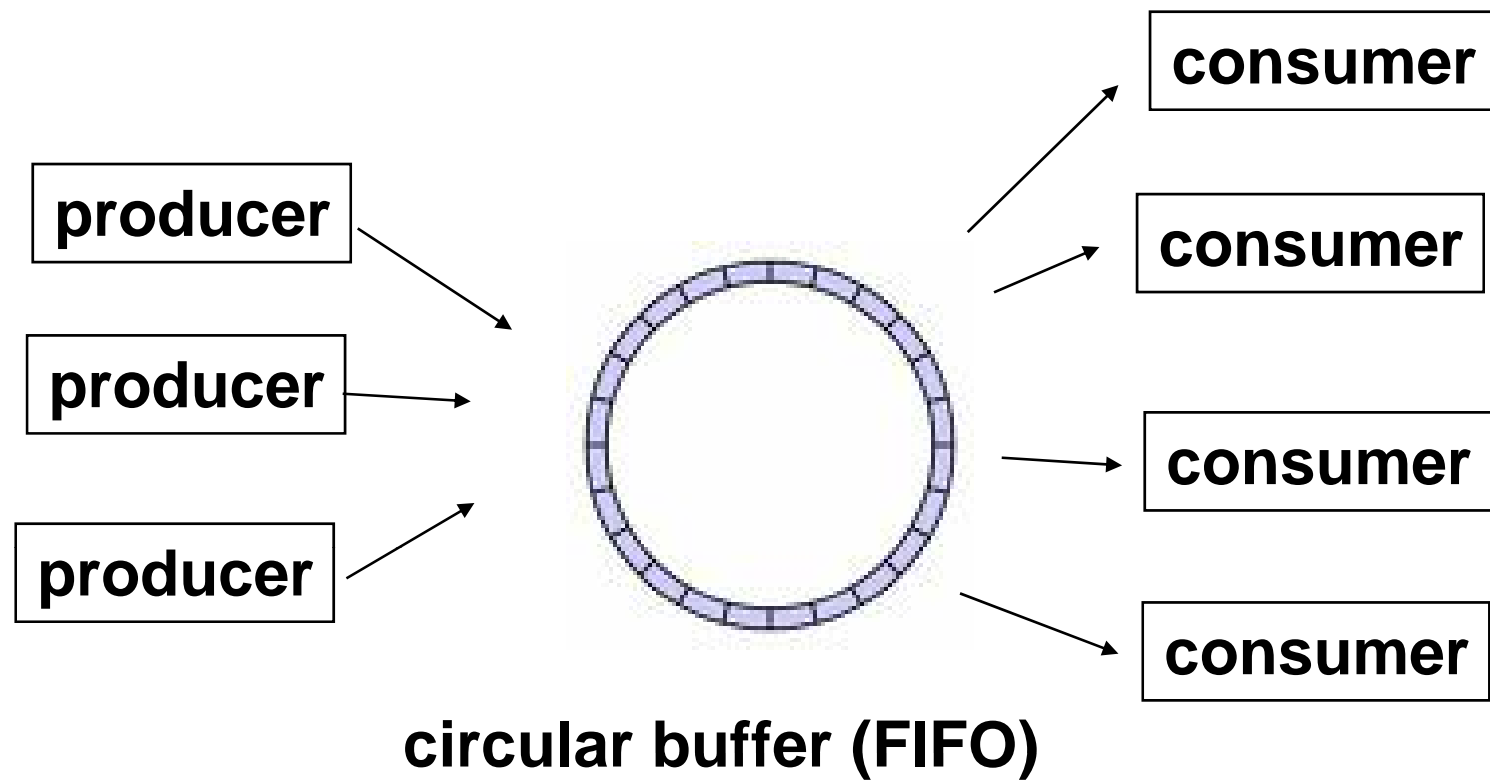
Improve Thread Efficiency

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 \Rightarrow individual customer's account
- shrink critical sections \Rightarrow 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

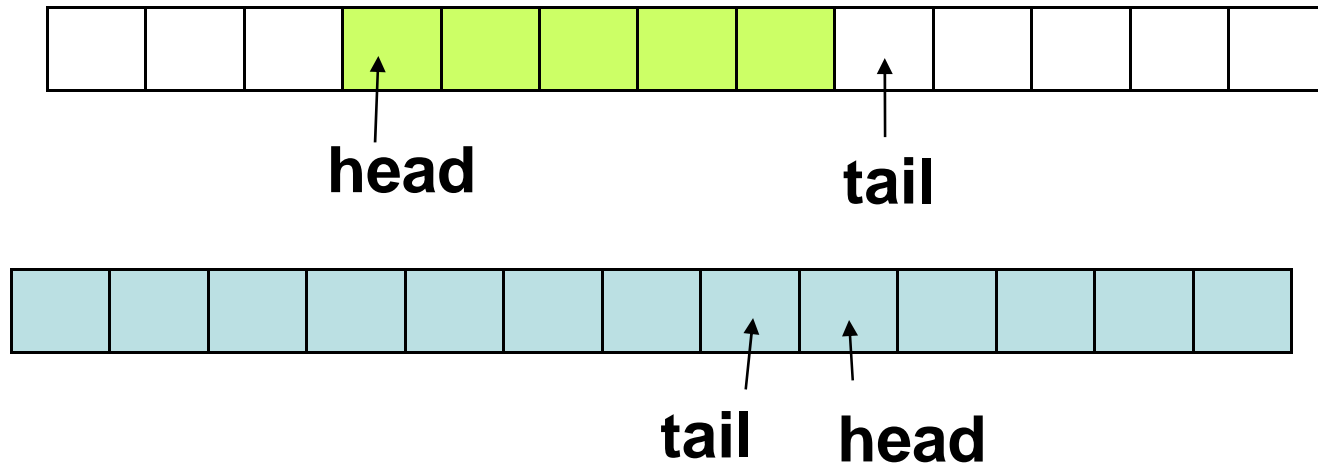


Producers and Consumers with a Finite Buffer

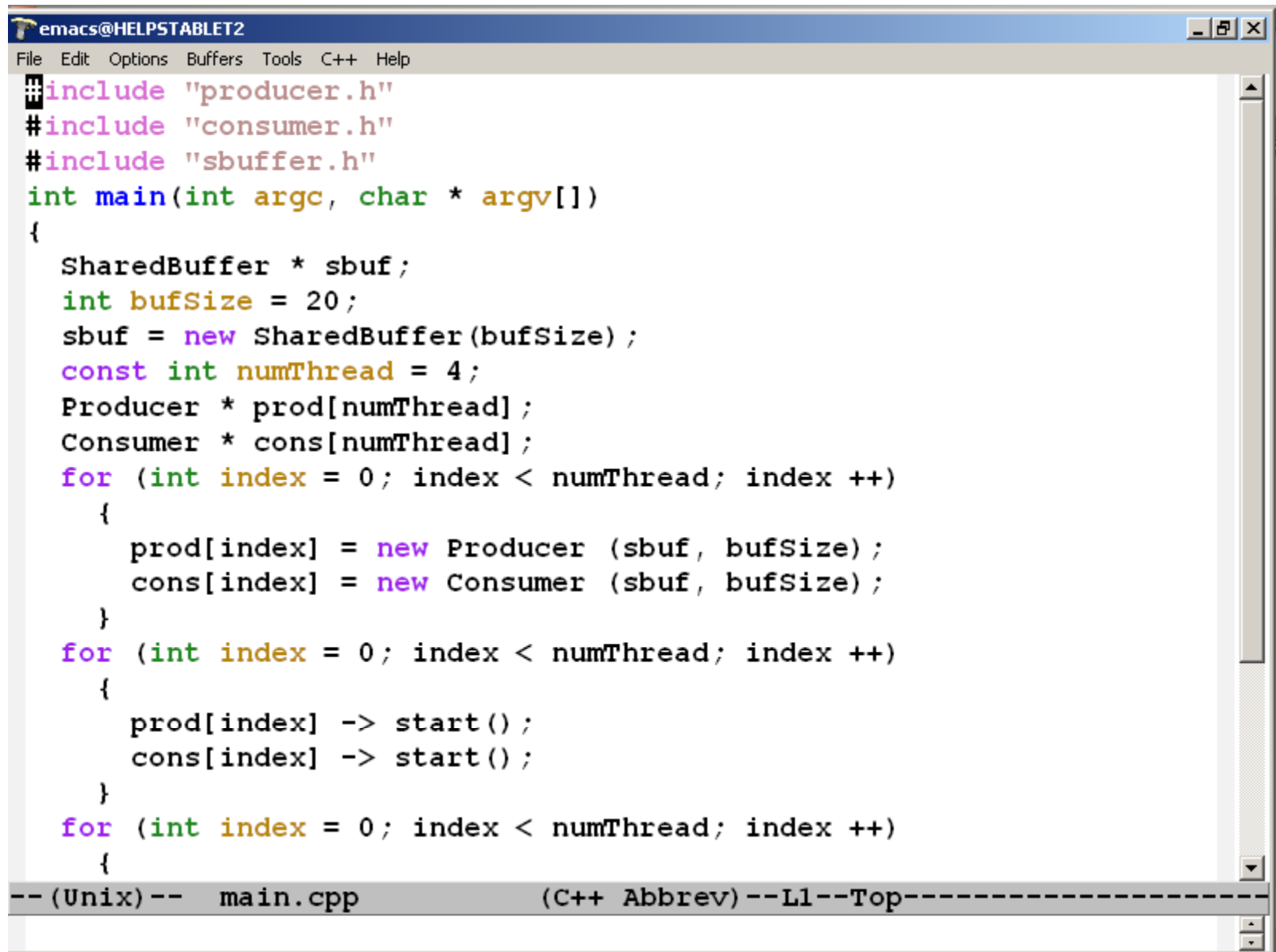


A producer can put an element in the buffer if it is not full. A consumer can get an element the buffer if it is not empty.

Circular Buffer



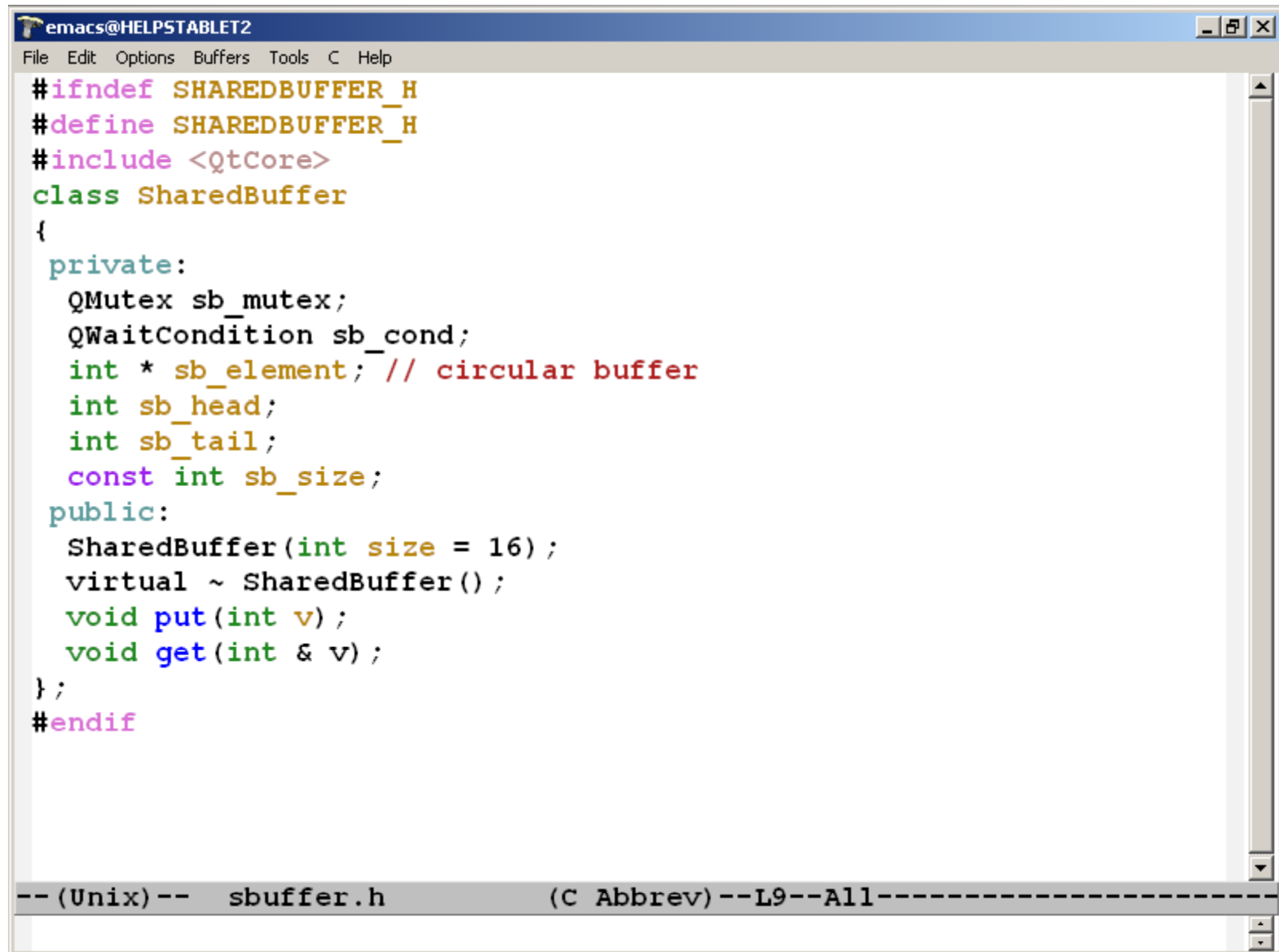
- $\text{head} = \text{tail} \Rightarrow \text{buffer empty}$
- $\text{tail} > \text{head} \text{ and } (\text{tail} - \text{head}) == \text{size} - 1 \Rightarrow \text{buffer full}$
- $\text{head} > \text{tail} \text{ and } (\text{head} - \text{tail}) == 1 \Rightarrow \text{buffer full}$



```
emacs@HELPSTABLET2
File Edit Options Buffers Tools C++ Help
#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++)
    {

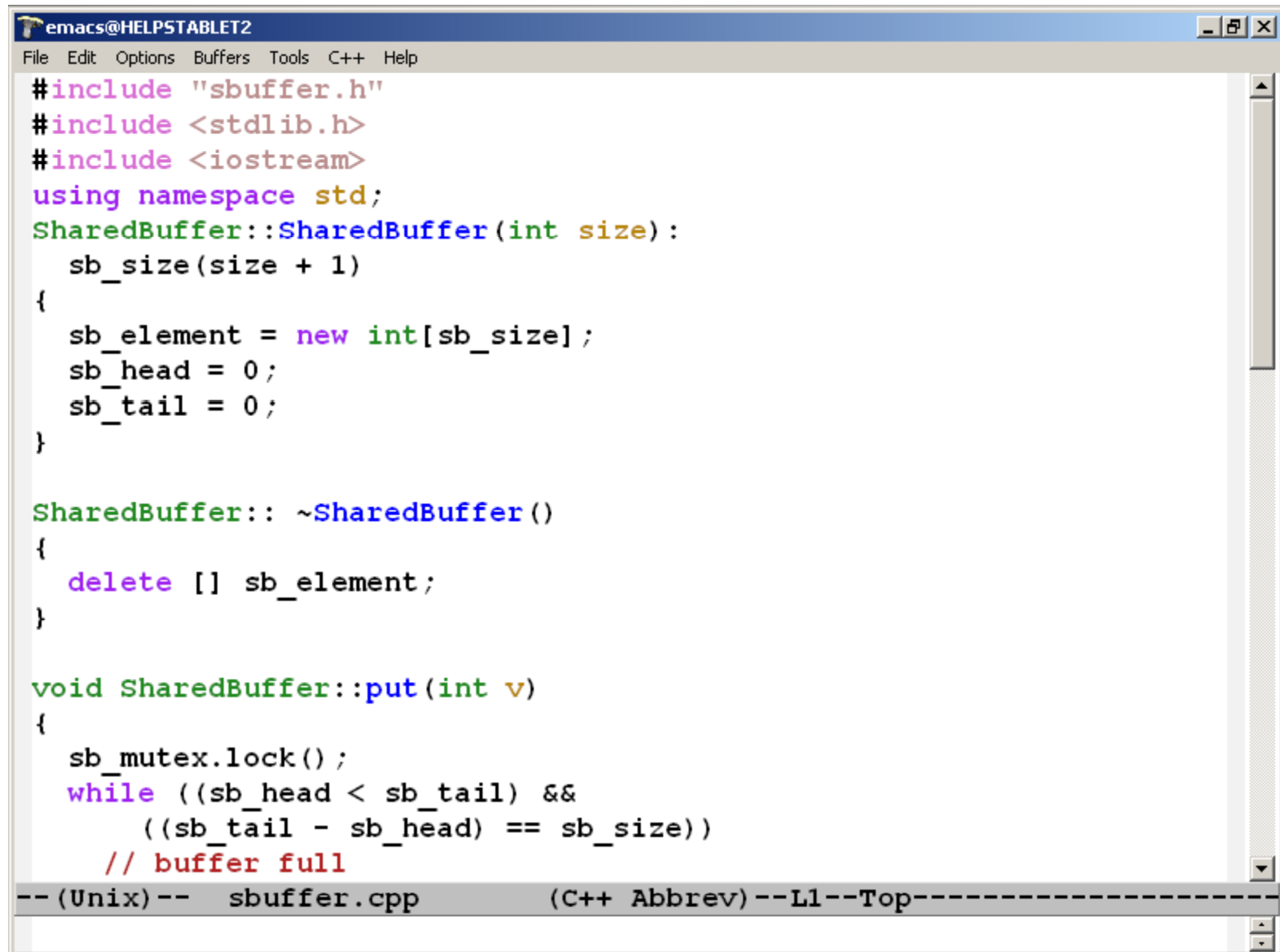
```

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



```
emacs@HELPSTABLET2
File Edit Options Buffers Tools C Help
#ifndef SHARED_BUFFER_H
#define SHARED_BUFFER_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 -----
```



The image shows a screenshot of an Emacs editor window. The title bar at the top reads "emacs@HELPSTABLET2". Below the title bar is a menu bar with the following items: "File", "Edit", "Options", "Buffers", "Tools", "C++", and "Help". The main editing area contains C++ code for a class named "SharedBuffer". The code is as follows:

```
#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;
}

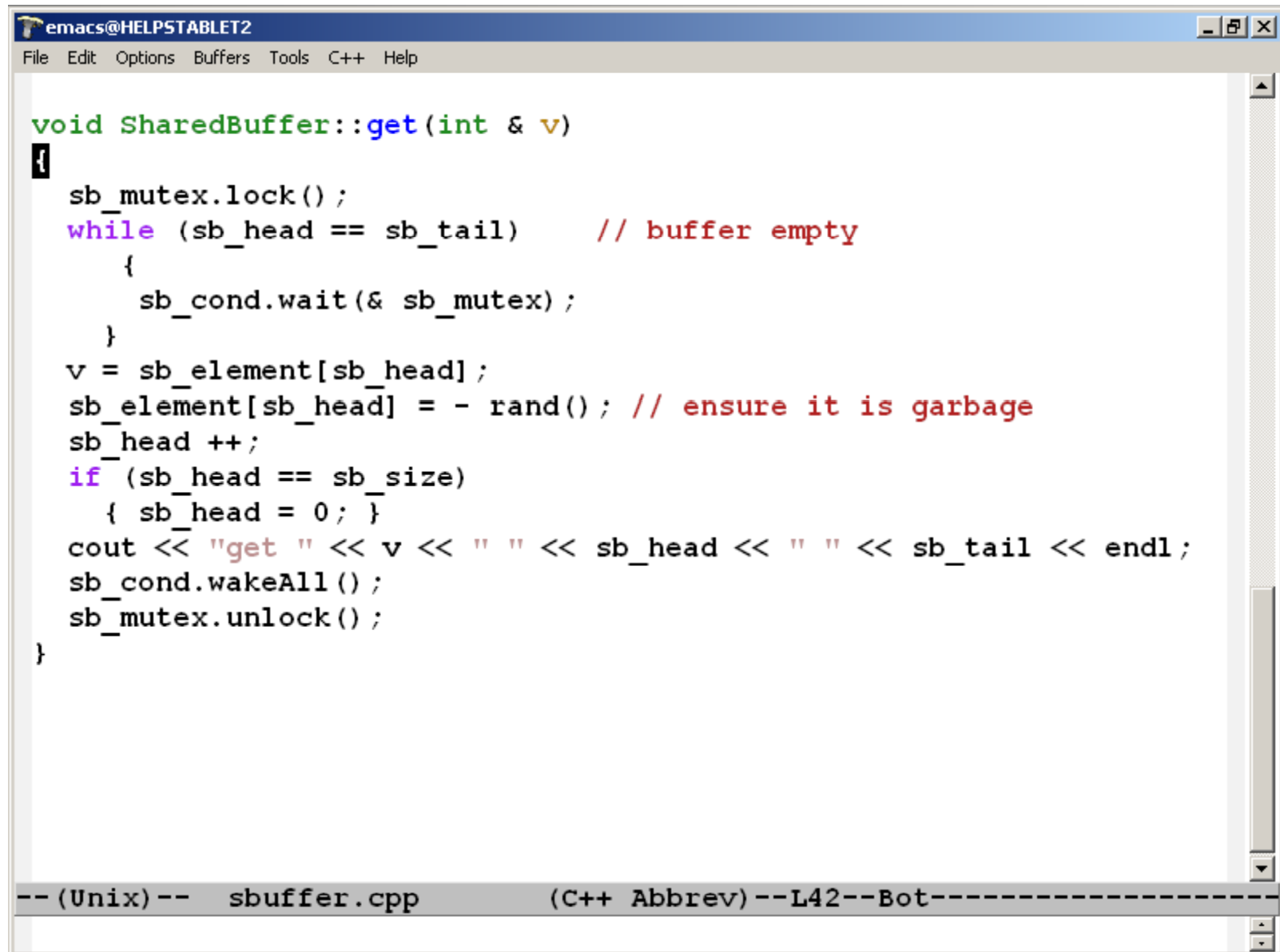
void SharedBuffer::put(int v)
{
    sb_mutex.lock();
    while ((sb_head < sb_tail) &&
           ((sb_tail - sb_head) == sb_size))
        // buffer full
}
```

At the bottom of the window, there is a status bar with the text: "--(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();
}

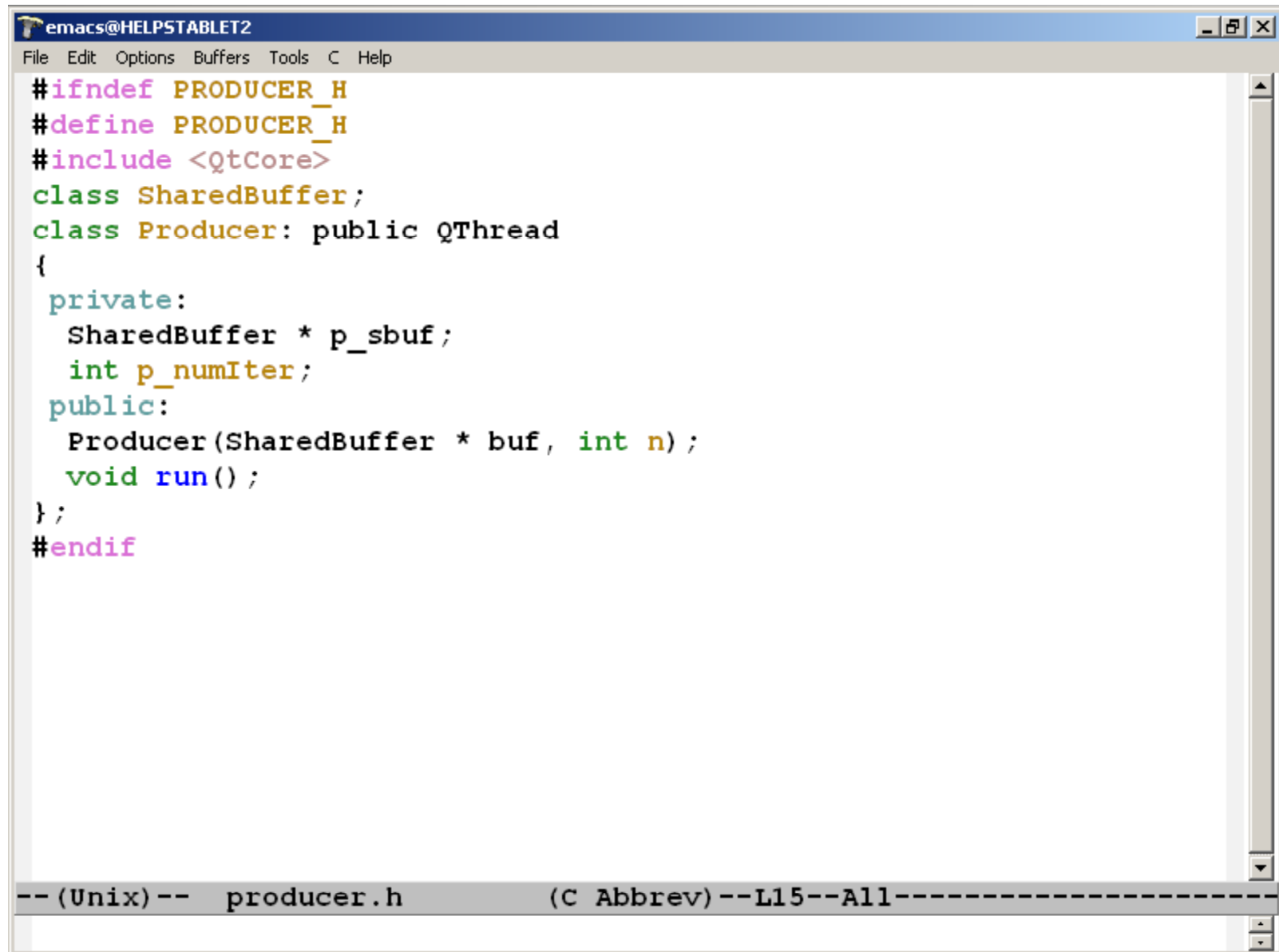
--(Unix)-- sbuffer.cpp (C++ Abbrev) --L21--24%-----
```



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

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();
}

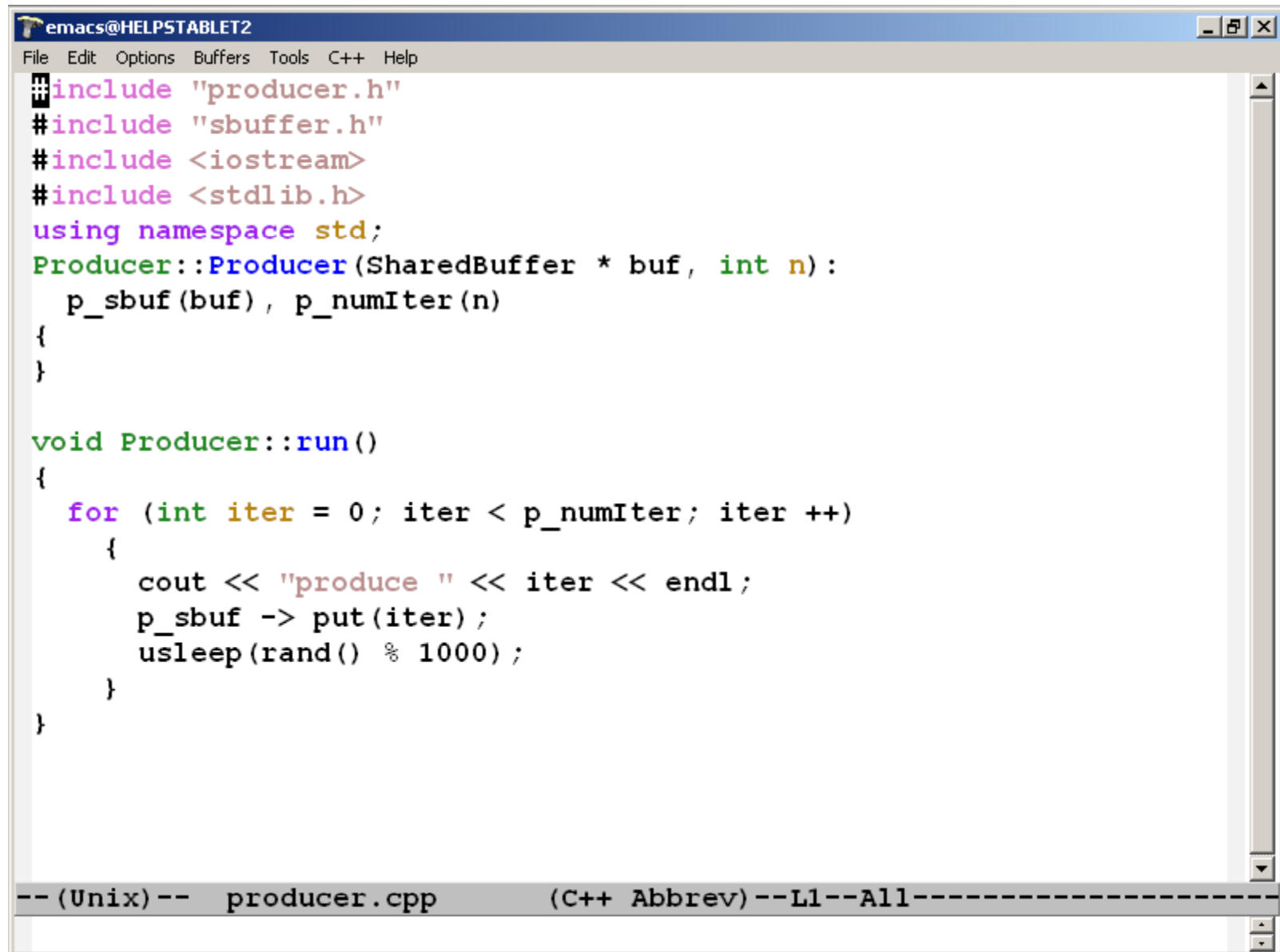
--(Unix)-- sbuffer.cpp (C++ Abbrev)--L42--Bot-----
```



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

#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

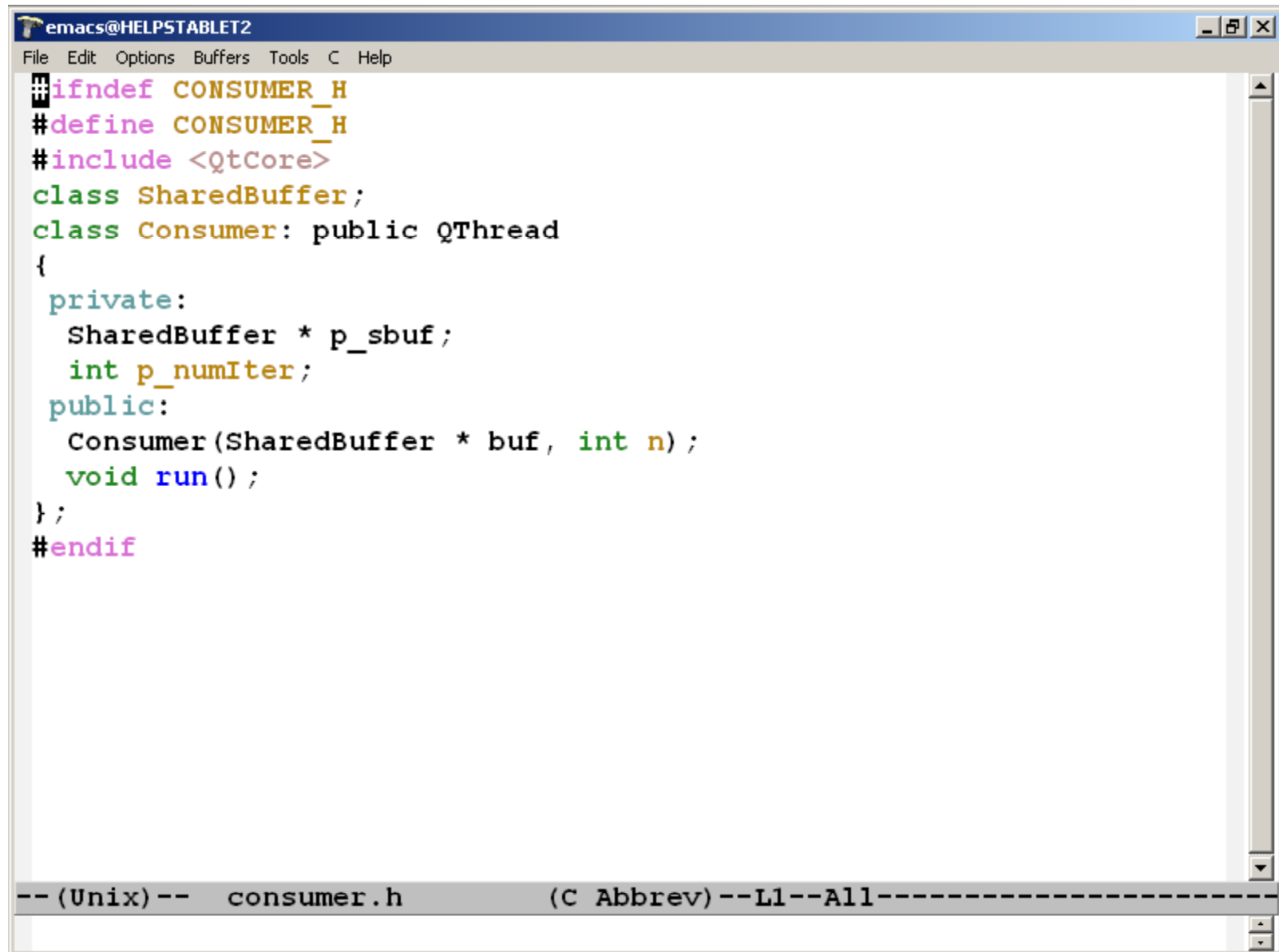
-- (Unix) -- producer.h (C Abbrev) --L15--All-----
```



```
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);
    }
}

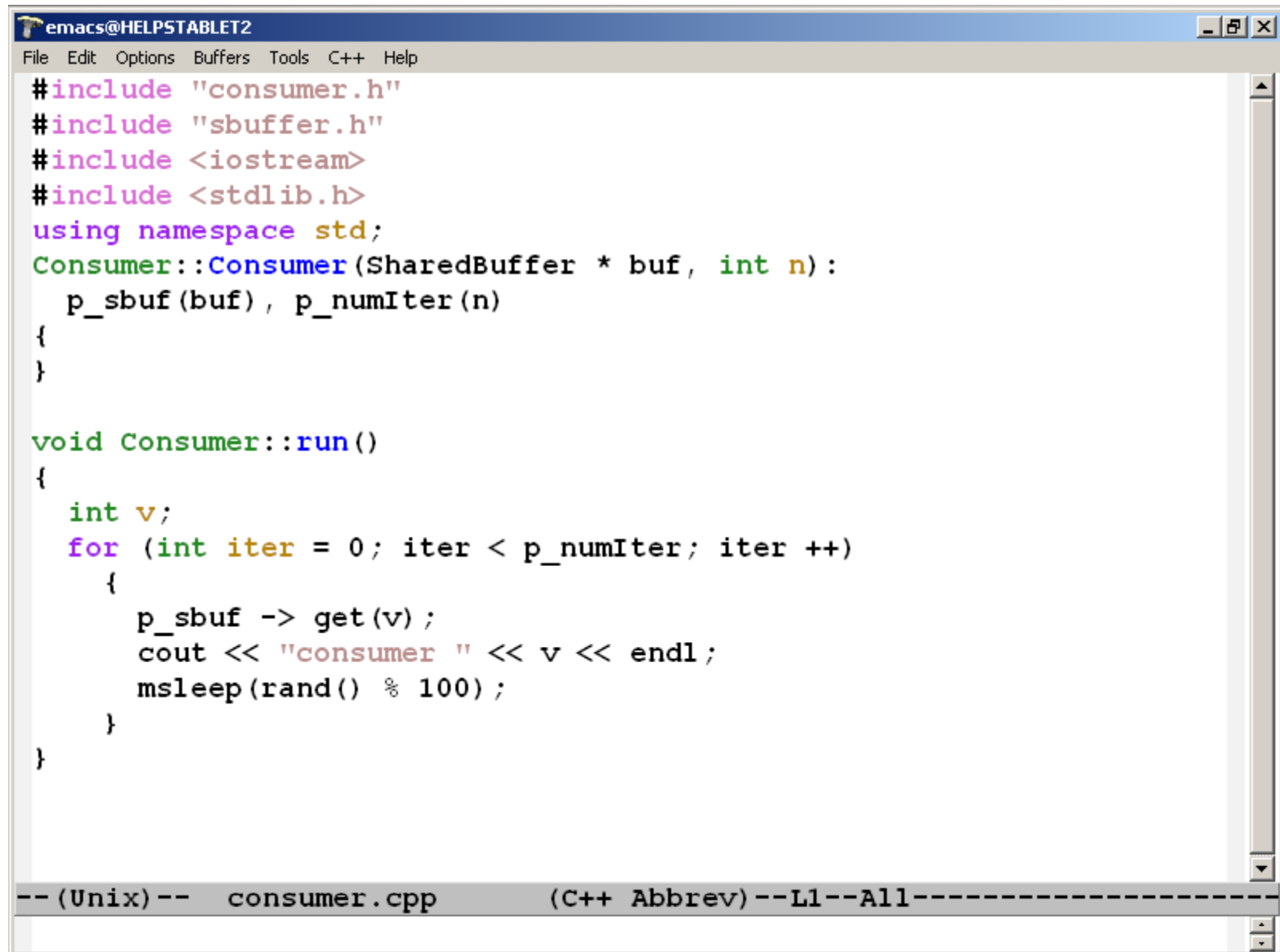
-- (Unix) -- producer.cpp (C++ Abbrev) -- L1 -- All -----
```



The image shows a screenshot of an Emacs editor window. The title bar at the top reads "emacs@HELPSTABLET2". Below the title bar is a menu bar with the following items: "File", "Edit", "Options", "Buffers", "Tools", "C", and "Help". The main editing area contains the following C++ code:

```
#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
```

At the bottom of the window, there is a status bar with the text: "-- (Unix) -- consumer.h (C Abbrev) -- L1 -- All -----".



The image shows a screenshot of an Emacs editor window. The title bar at the top reads "emacs@HELPSTABLET2". Below the title bar is a menu bar with the following items: "File", "Edit", "Options", "Buffers", "Tools", "C++", and "Help". The main editing area contains C++ code for a consumer thread. The code is as follows:

```
#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);
    }
}
```

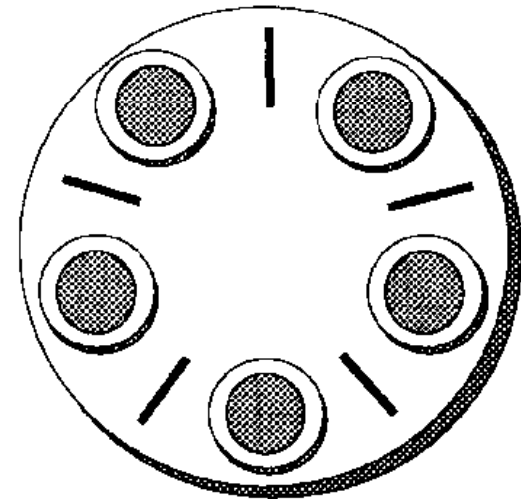
At the bottom of the window, there is a status bar with the text: "-- (Unix) -- consumer.cpp (C++ Abbrev) -- L1 -- All -----".

Check Correctness

- wrong result \Rightarrow the program has bugs
- correct result \Rightarrow maybe you are lucky
- know the correct results, even though the interleaving may be different
- head \neq 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

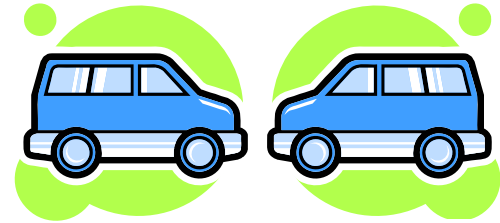
using C++ and Java

Deadlock

Yung-Hsiang Lu
yunlu@purdue.edu

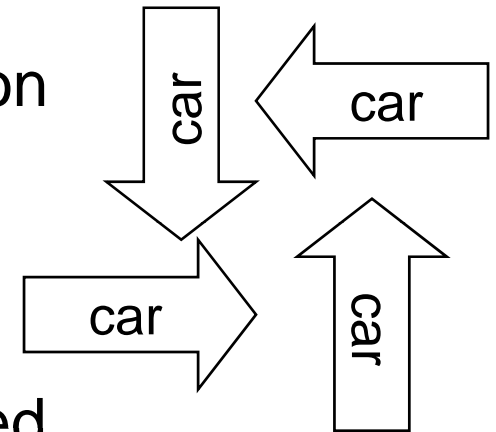


Deadlock

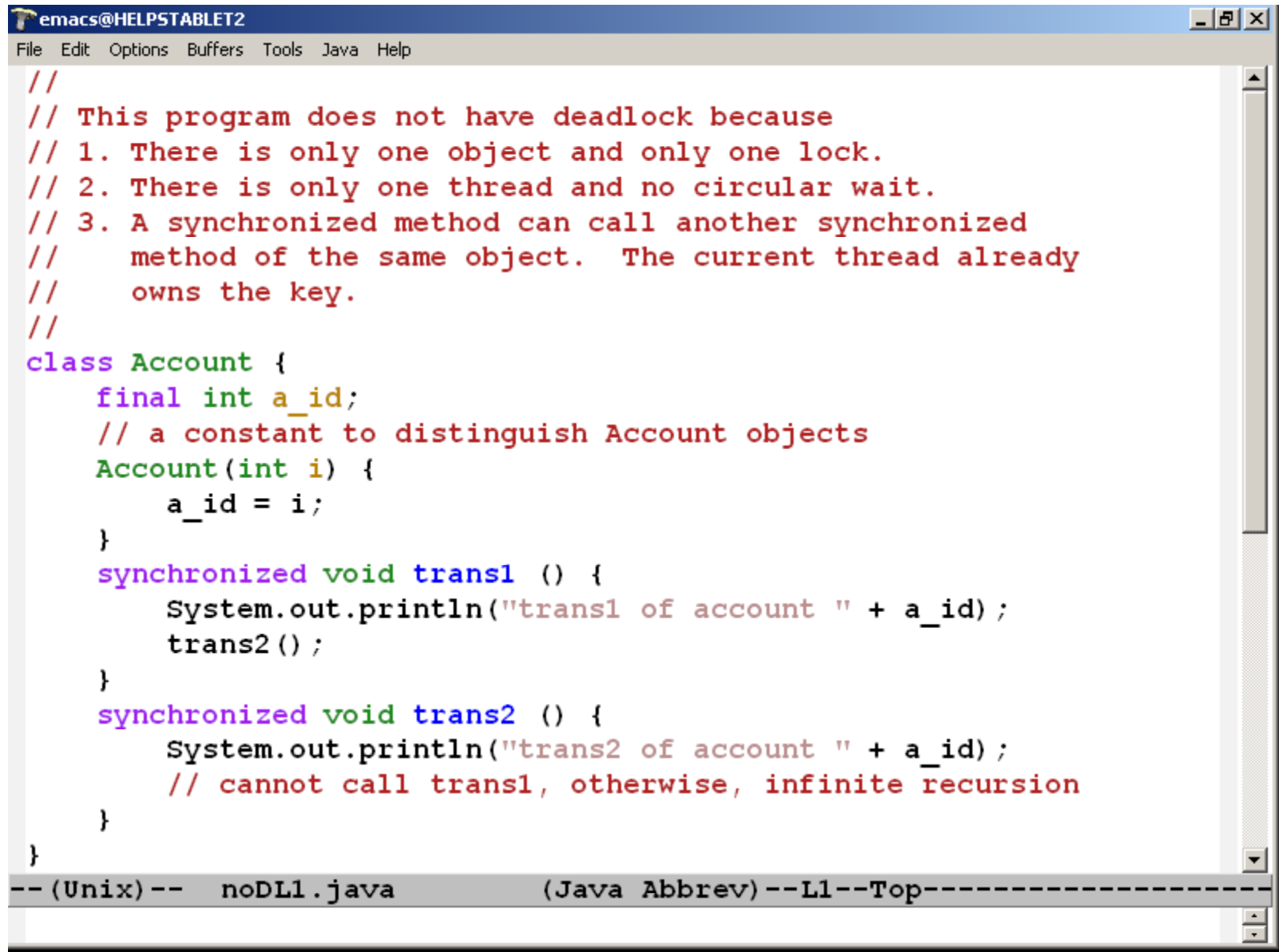


four necessary conditions for deadlock

- 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

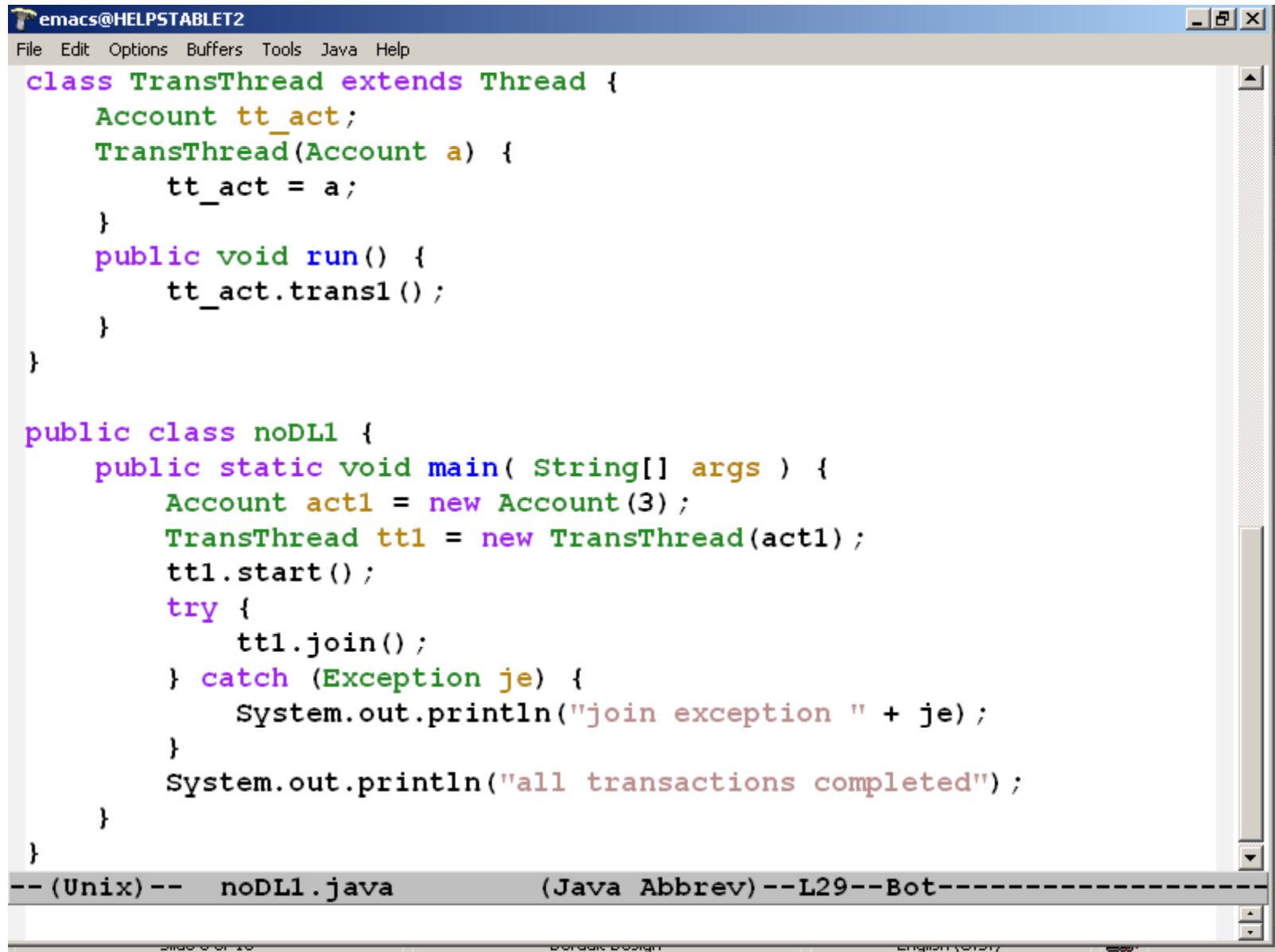


Programs without Deadlock

The image shows a screenshot of an Emacs editor window. The title bar at the top reads "emacs@HELPSTABLET2". Below the title bar is a menu bar with the following items: "File", "Edit", "Options", "Buffers", "Tools", "Java", and "Help". The main editing area contains Java code. The code starts with several comment lines explaining why the program does not have a deadlock. It then defines a class named "Account" with a final integer attribute "a_id". The constructor "Account(int i)" initializes "a_id" to "i". There are two synchronized methods: "trans1()" which prints "trans1 of account " followed by "a_id" and then calls "trans2()", and "trans2()" which prints "trans2 of account " followed by "a_id" and has a comment stating it cannot call "trans1" to avoid infinite recursion. The status bar at the bottom of the window displays "-- (Unix)-- noDL1.java (Java Abbrev) --L1--Top-----".

```
//  
// 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-----

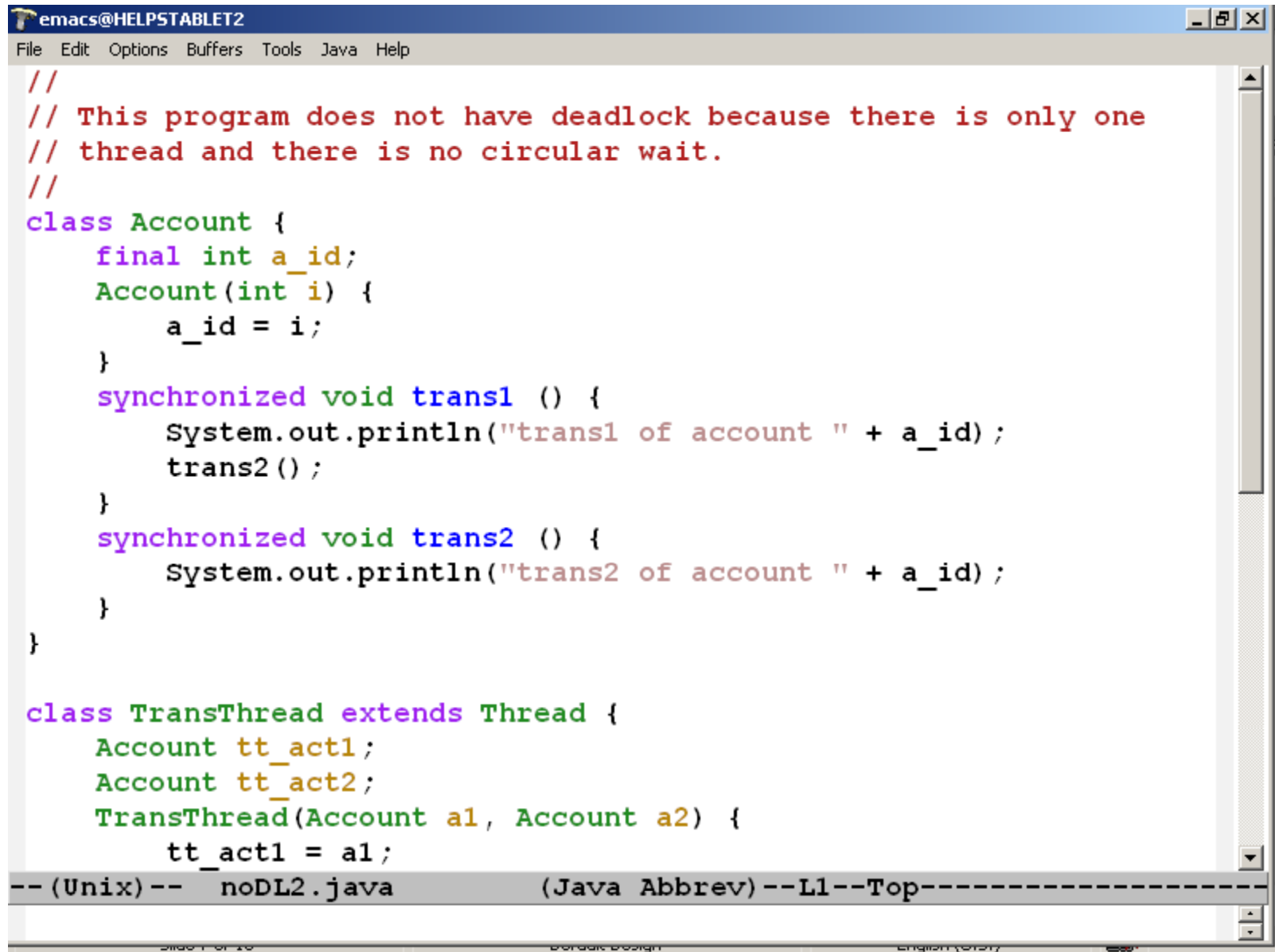


The screenshot shows an Emacs editor window titled 'emacs@HELPSTABLET2'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Java', and 'Help'. The code is written in Java and defines two classes: 'TransThread' and 'noDL1'. 'TransThread' is a subclass of 'Thread' that holds an 'Account' reference and calls 'trans1()' on it. 'noDL1' is a public class with a 'main' method that creates an 'Account' object, starts a 'TransThread' instance, and attempts to join it, with exception handling and status printing. The status bar at the bottom shows '(Unix)-- noDL1.java (Java Abbrev)--L29--Bot-----'.

```
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");
    }
}
```

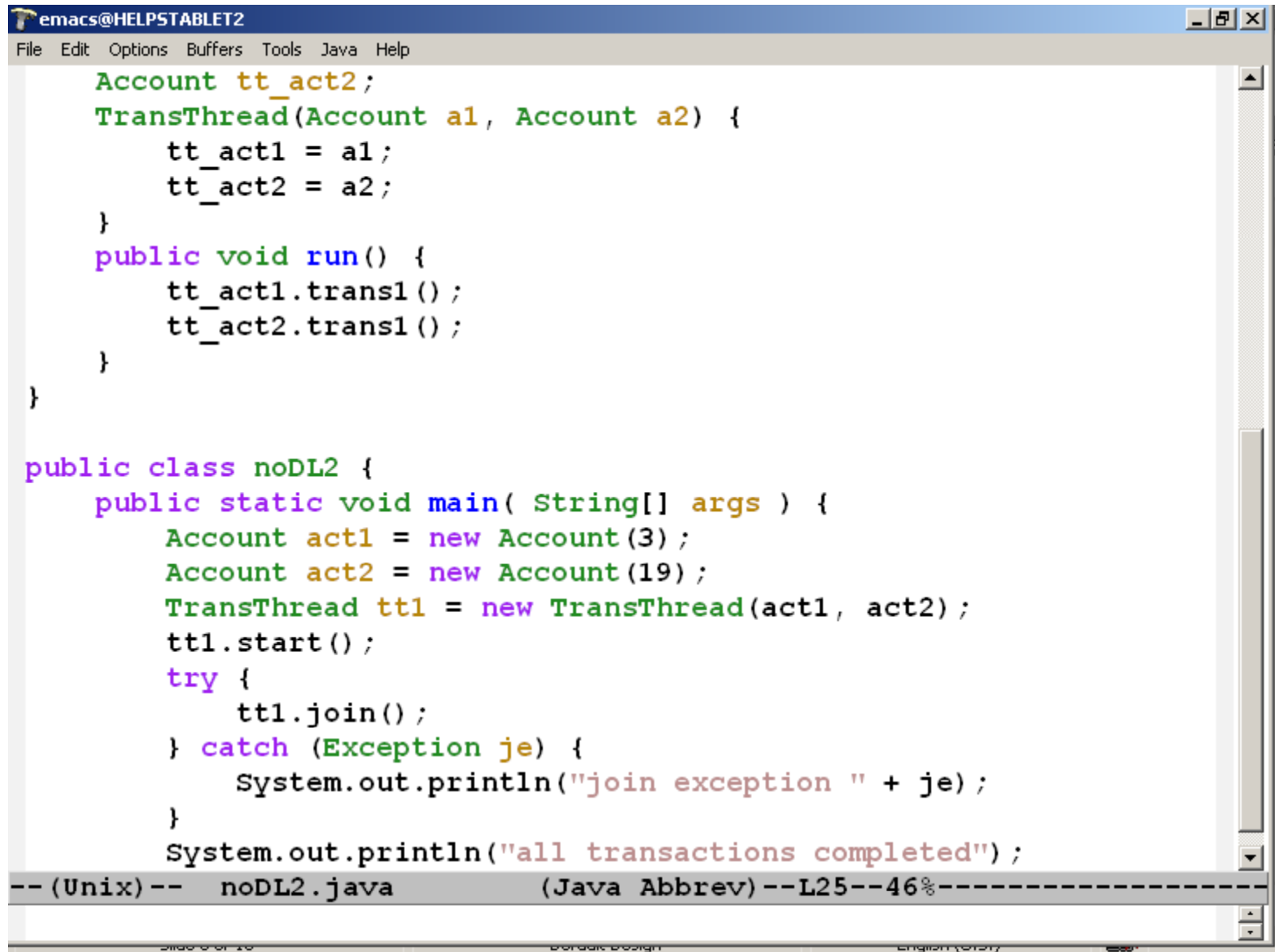
-- (Unix)-- noDL1.java (Java Abbrev)--L29--Bot-----



The screenshot shows an Emacs editor window titled 'emacs@HELPSTABLET2'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Java', and 'Help'. The code is written in Java and is as follows:

```
//  
// 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;  
    }  
}
```

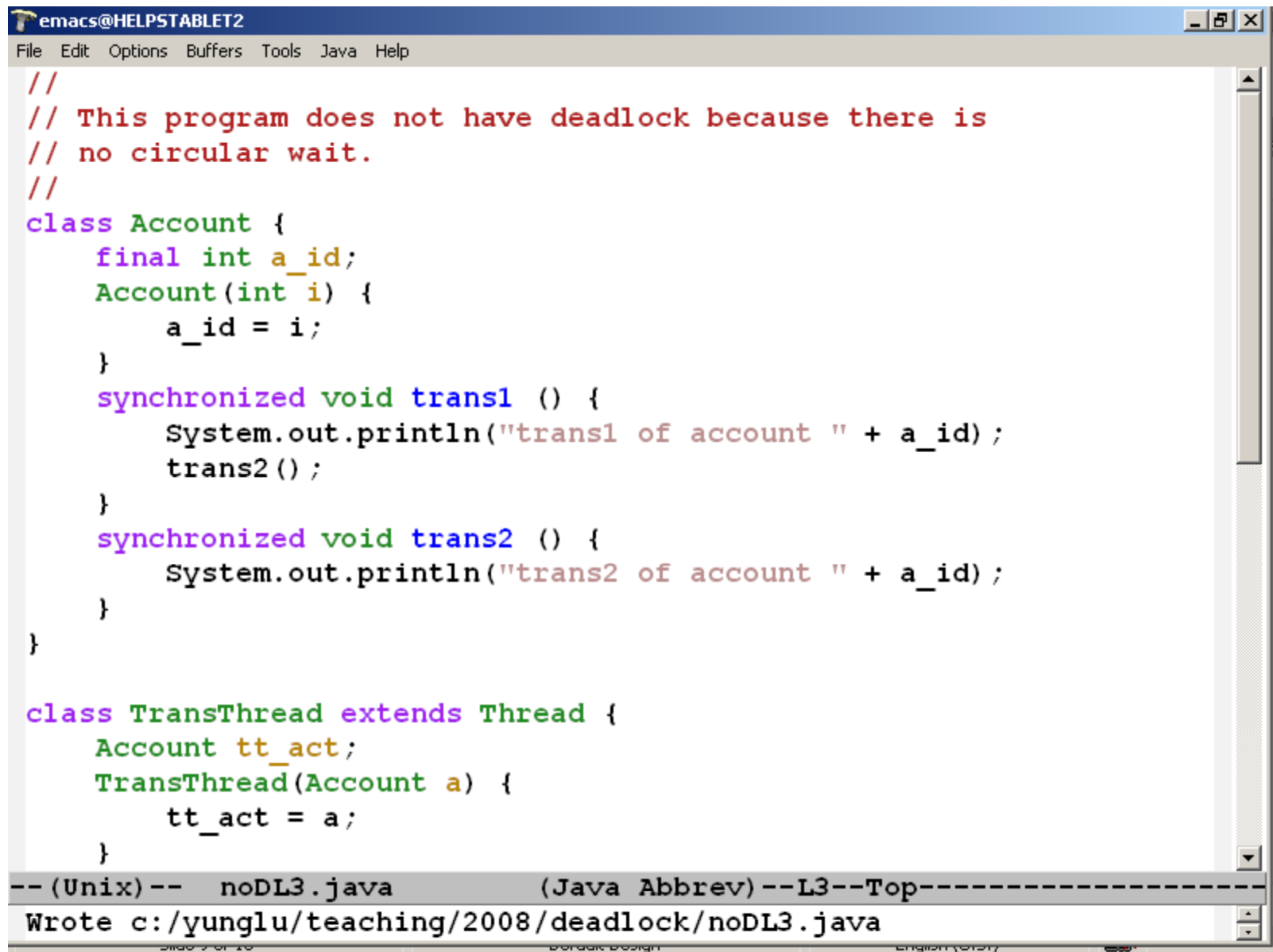
The status bar at the bottom of the editor displays: '-- (Unix)-- noDL2.java (Java Abbrev) --L1--Top-----'.



The screenshot shows an Emacs editor window titled 'emacs@HELPSTABLET2'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Java', and 'Help'. The code is written in Java and defines a 'TransThread' class and a 'noDL2' class. The 'TransThread' class has a constructor that takes two 'Account' objects and a 'run()' method that calls 'trans1()' on both. The 'noDL2' class has a 'main()' method that creates two 'Account' objects, a 'TransThread' object, starts it, and then joins it, printing out the results.

```
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");  
    }  
}
```

-- (Unix)-- noDL2.java (Java Abbrev) --L25--46%-----



The screenshot shows an Emacs editor window titled 'emacs@HELPSTABLET2'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Java', and 'Help'. The code is written in Java and is color-coded: comments are red, keywords are purple, and identifiers are green. The code defines an 'Account' class with a 'final int a_id' and two synchronized methods, 'trans1' and 'trans2', which print their respective transaction numbers for a given account. It also defines a 'TransThread' class that extends 'Thread' and holds a reference to an 'Account' object. The status bar at the bottom indicates the file is 'noDL3.java' and provides navigation options like '(Unix)--', '(Java Abbrev)', '--L3--', and 'Top--'. Below the status bar, it shows the file path 'Wrote c:/yunglu/teaching/2008/deadlock/noDL3.java'.

```
//  
// 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

```
emacs@HELPSTABLET2
File Edit Options Buffers Tools Java Help

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);
        }
    }
}

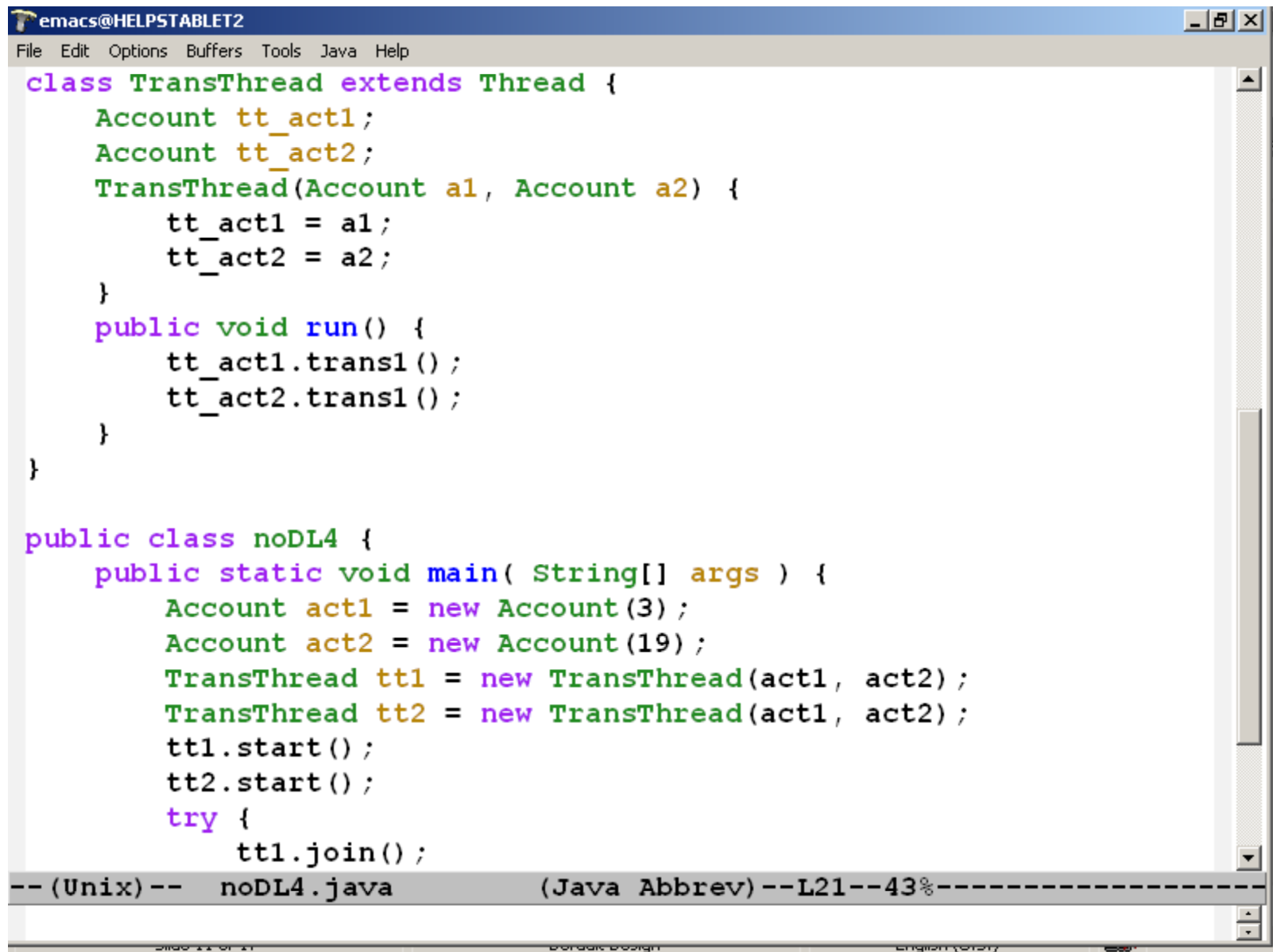
-- (Unix) -- noDL3.java (Java Abbrev) -- L28 -- 38% -----
```

```
emacs@HELPSTABLET2
File Edit Options Buffers Tools Java Help

//
// 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)
```

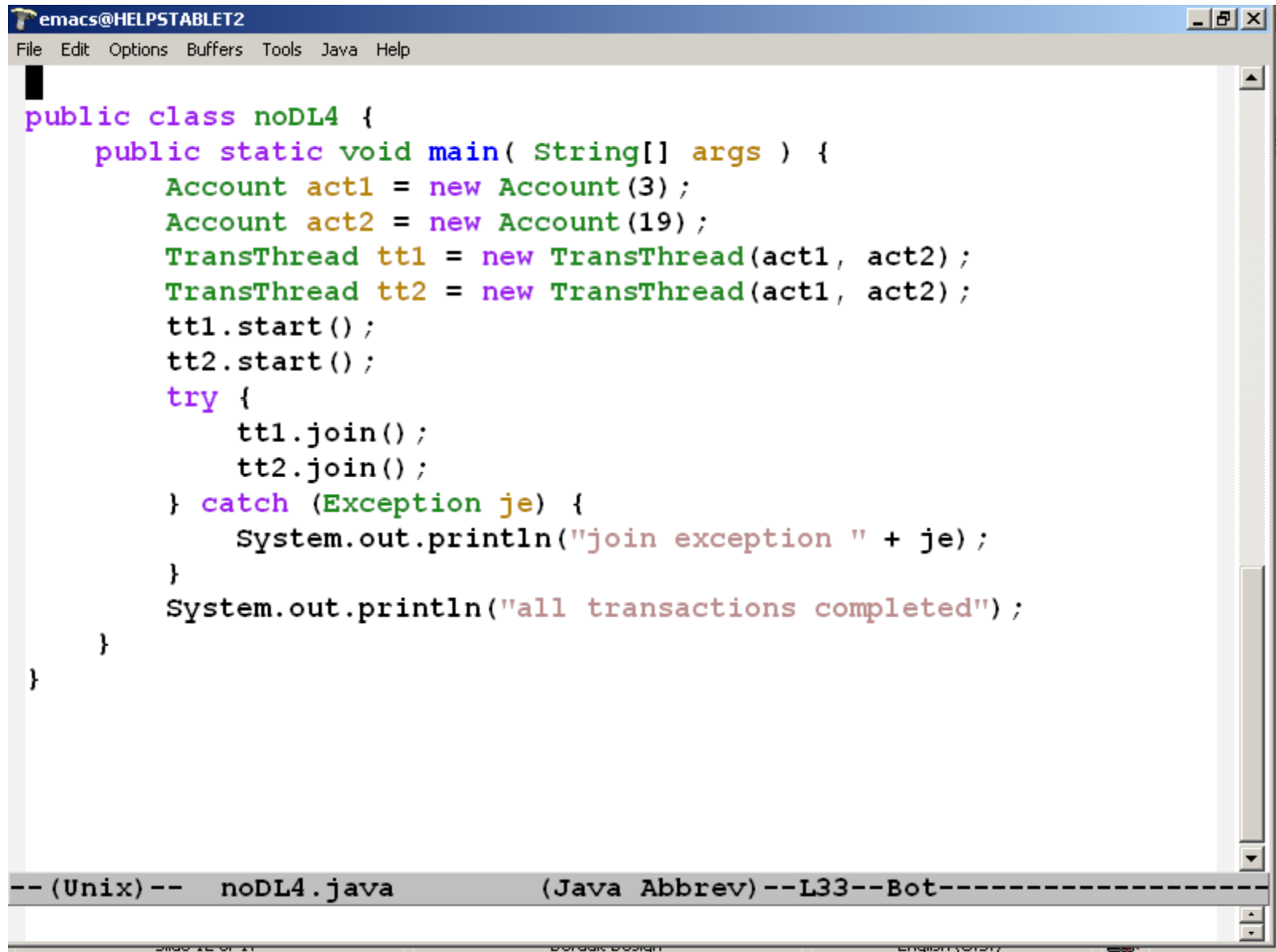


The screenshot shows an Emacs editor window titled 'emacs@HELPSTABLET2'. The menu bar includes 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Java', and 'Help'. The code is written in Java and is color-coded. It defines a 'TransThread' class that extends 'Thread' and has two 'Account' objects, 'tt_act1' and 'tt_act2'. The 'run()' method calls 'trans1()' on both. A 'noDL4' class has a 'main' method that creates two 'Account' objects, two 'TransThread' objects, starts them, and then joins 'tt1'. The status bar at the bottom shows '-- (Unix) -- noDL4.java (Java Abbrev) -- L21 -- 43% --'.

```
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

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");
    }
}

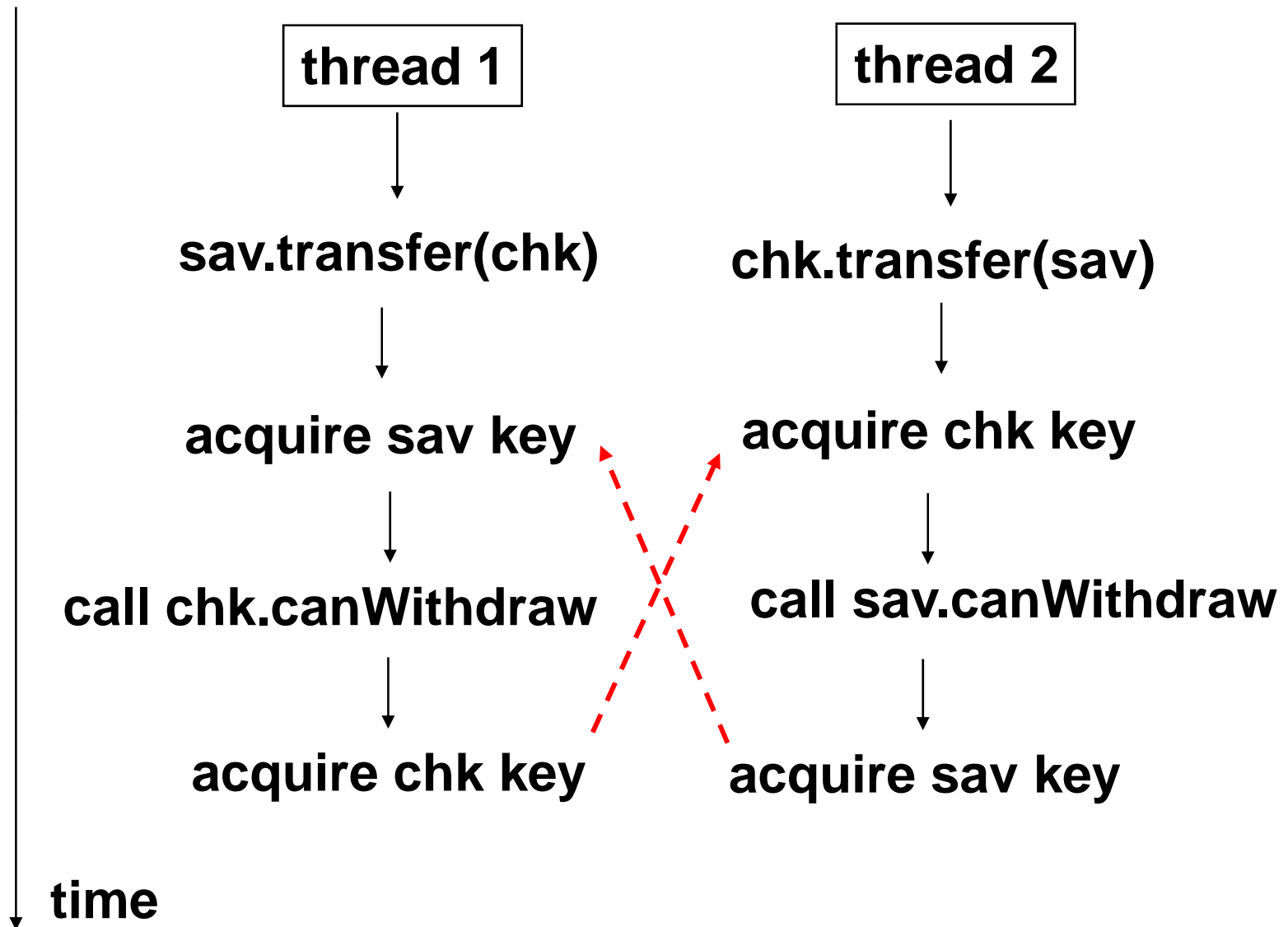
-- (Unix)-- noDL4.java (Java Abbrev)--L33--Bot-----
```

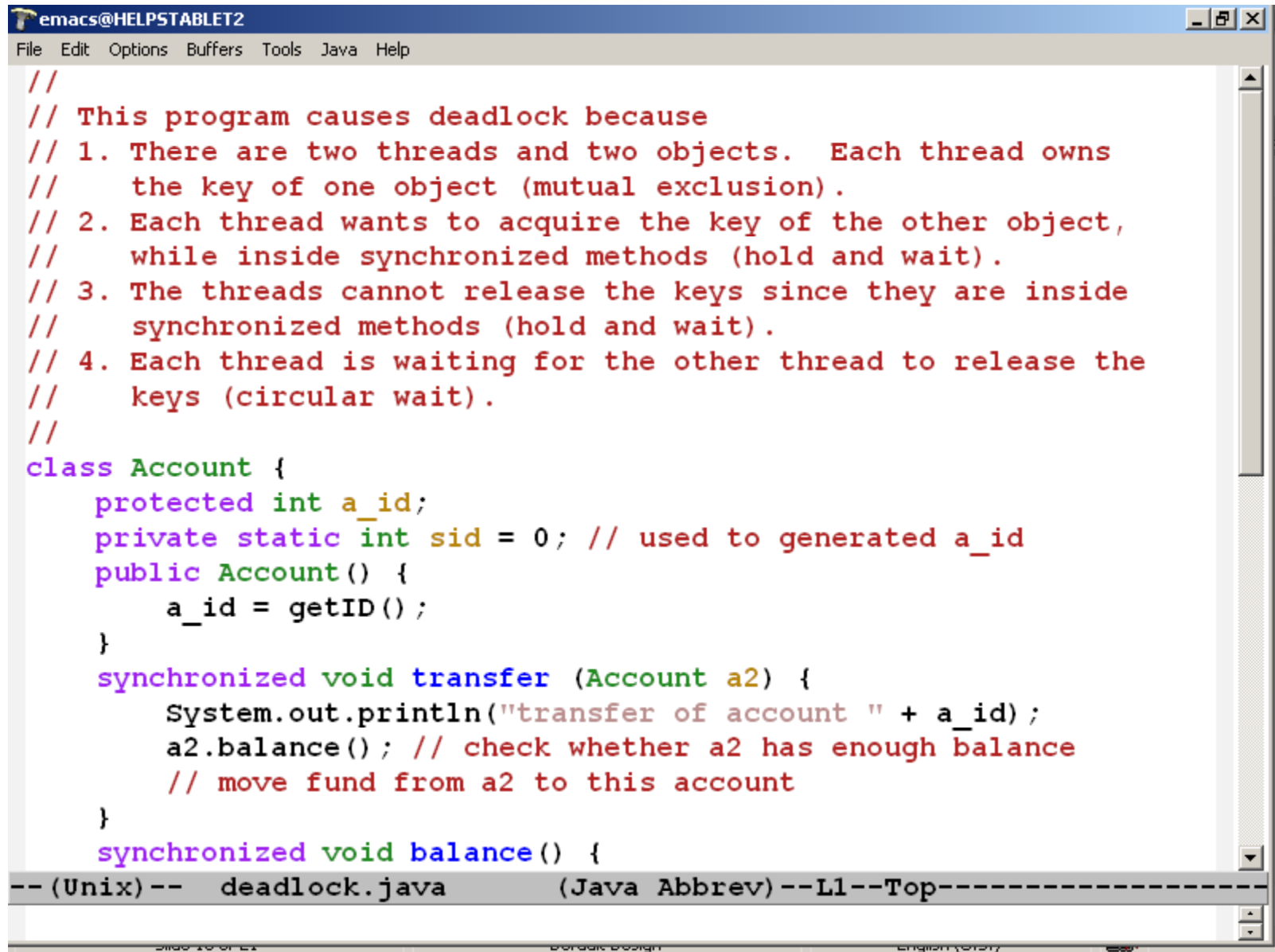
Java Example of Deadlock

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?

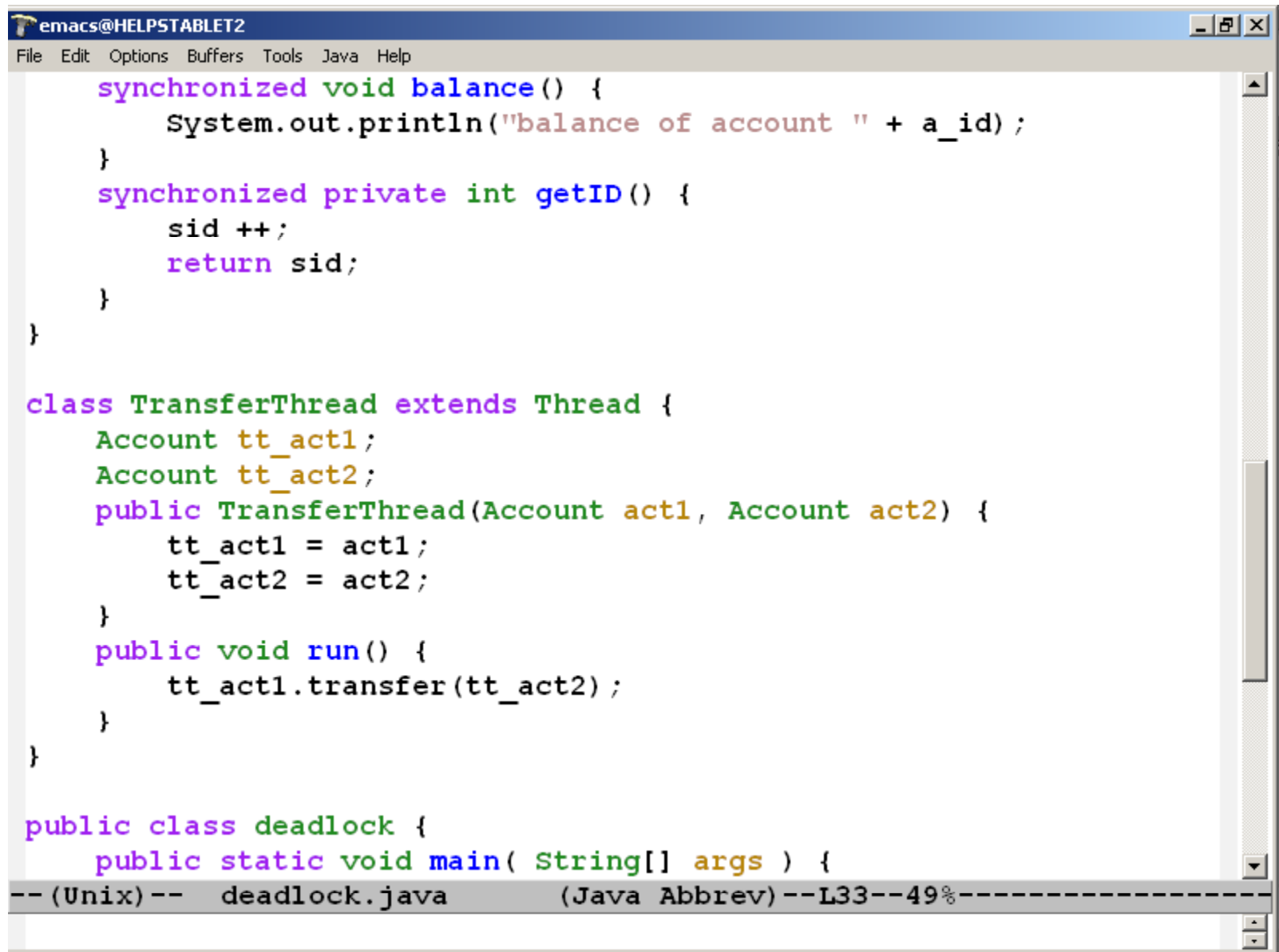


```
class Account {  
    private int balance;  
    synchronized void transfer (Account source, int amount) {  
        if (source.canWithdraw(amount)) {  
            source.withdraw(amount);  
            balance += amount;  
        }  
    }  
    synchronized boolean canWithdraw(amount) {  
  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



The image shows a screenshot of an Emacs editor window. The title bar at the top reads 'emacs@HELPSTABLET2'. Below the title bar is a menu bar with the following items: 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Java', and 'Help'. The main editing area contains Java code. The code starts with several comments explaining the deadlock scenario: two threads and two objects, each thread owns a key, each thread wants the other's key while holding its own, and they cannot release the keys because they are in synchronized methods. The code then defines a 'class Account' with a 'protected int a_id', a 'private static int sid' initialized to 0, and a 'public Account()' constructor that calls 'getID()'. It also has a 'synchronized void transfer(Account a2)' method that prints a message and calls 'a2.balance()', and a 'synchronized void balance()' method. At the bottom of the window, there is a status bar with the text '--(Unix)-- deadlock.java (Java Abbrev)--L1--Top-----'.

```
//  
// 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() {  
--(Unix)--  deadlock.java      (Java Abbrev)--L1--Top-----
```

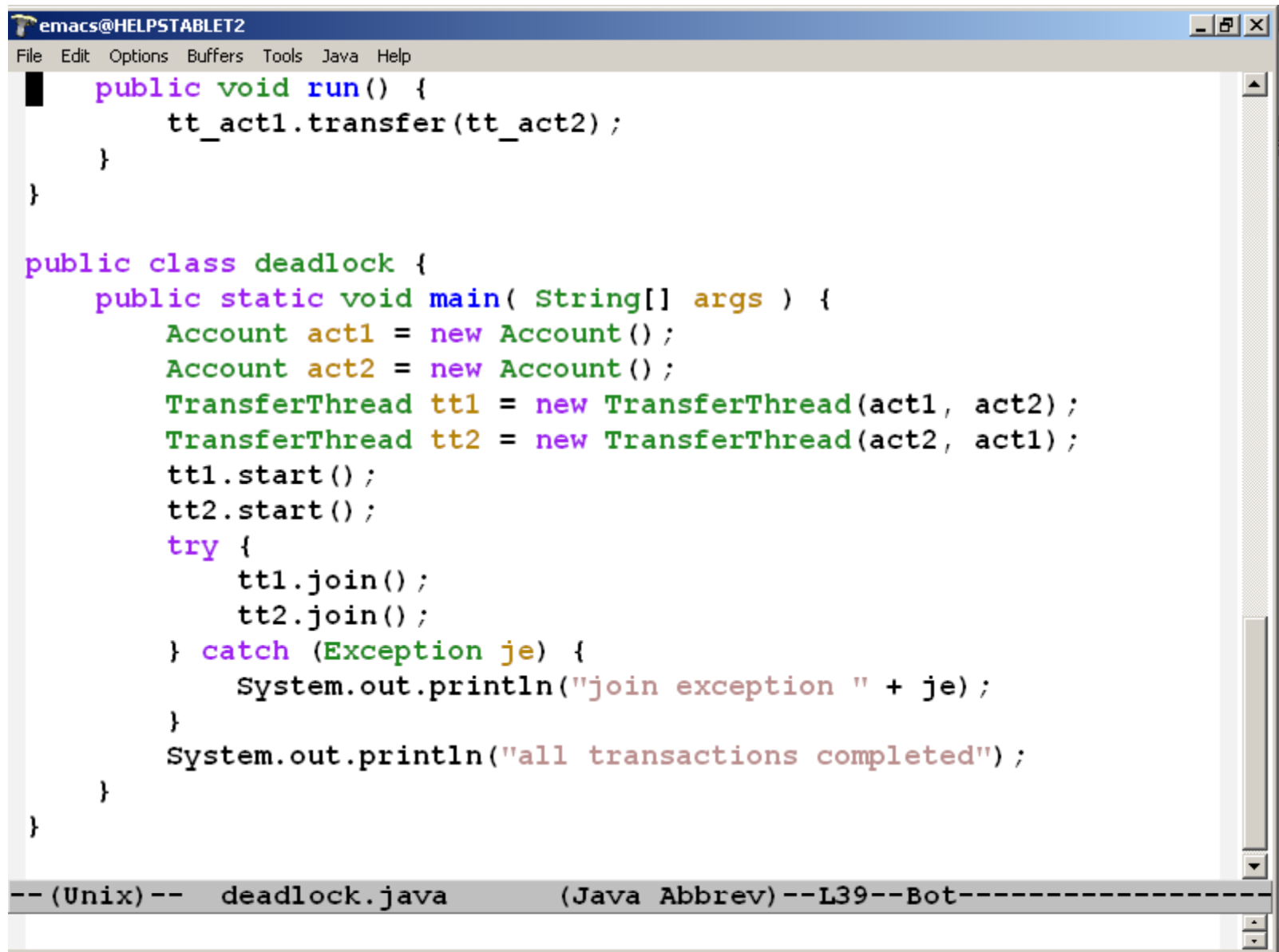


The image shows a screenshot of an Emacs editor window titled "emacs@HELPSTABLET2". The window contains Java code for a deadlock example. The code defines a `TransferThread` class that extends `Thread` and a `deadlock` class with a `main` method. The `TransferThread` class has two `Account` objects, `tt_act1` and `tt_act2`, and a `run` method that calls `tt_act1.transfer(tt_act2)`. The `deadlock` class has a `main` method that creates two `TransferThread` objects and starts them. The status bar at the bottom of the window shows "--(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 ) {
--(Unix)--  deadlock.java      (Java Abbrev)--L33--49%-----
```

The image shows a screenshot of an Emacs editor window. The title bar at the top reads "emacs@HELPSTABLET2". Below the title bar is a menu bar with the following items: "File", "Edit", "Options", "Buffers", "Tools", "Java", and "Help". The main editing area contains Java code. The code defines a method `run()` and a class `deadlock` with a `main` method. The `main` method creates two `Account` objects, `act1` and `act2`, and two `TransferThread` objects, `tt1` and `tt2`. `tt1` is created with `act1` and `act2` as arguments, and `tt2` is created with `act2` and `act1` as arguments. Both threads are started. A `try` block follows, containing `tt1.join()` and `tt2.join()`. A `catch` block for `Exception je` contains `System.out.println("join exception " + je);`. After the `try` block, `System.out.println("all transactions completed");` is executed. The code is color-coded: keywords are purple, class and method names are green, and variable names are orange. The status bar at the bottom of the window displays "-- (Unix) -- deadlock.java (Java Abbrev) --L39--Bot-----".

```
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");
    }
}
```

ECE 462

**Object-Oriented Programming
using C++ and Java**

Thread Performance

Yung-Hsiang Lu
yunlu@purdue.edu

Performance Measurement

ts1 = current time;

execute the program without creating threads;

ts2 = current time;

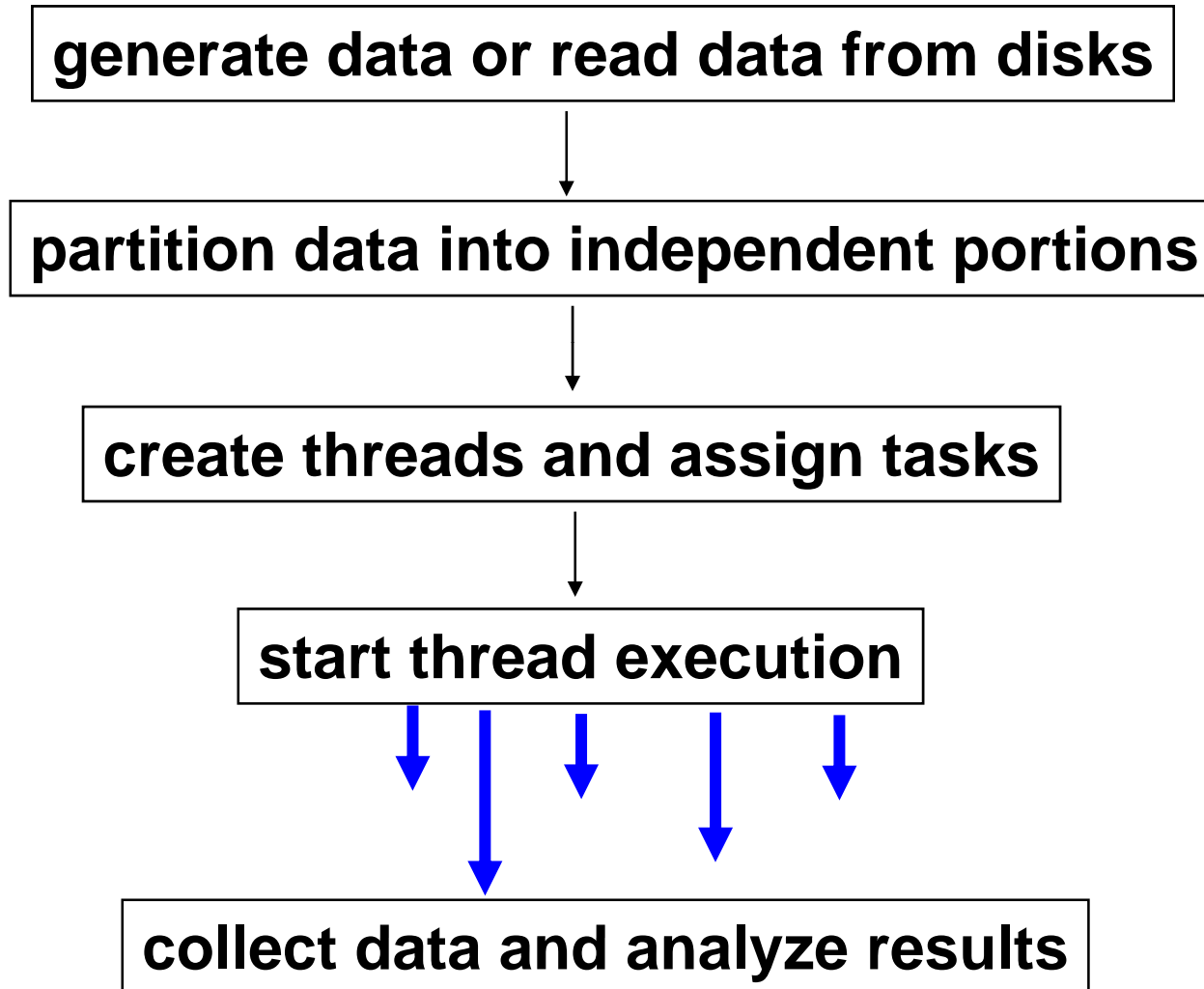
tp1 = current time;

execute the program with multiple threads

tp2 = current time;

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

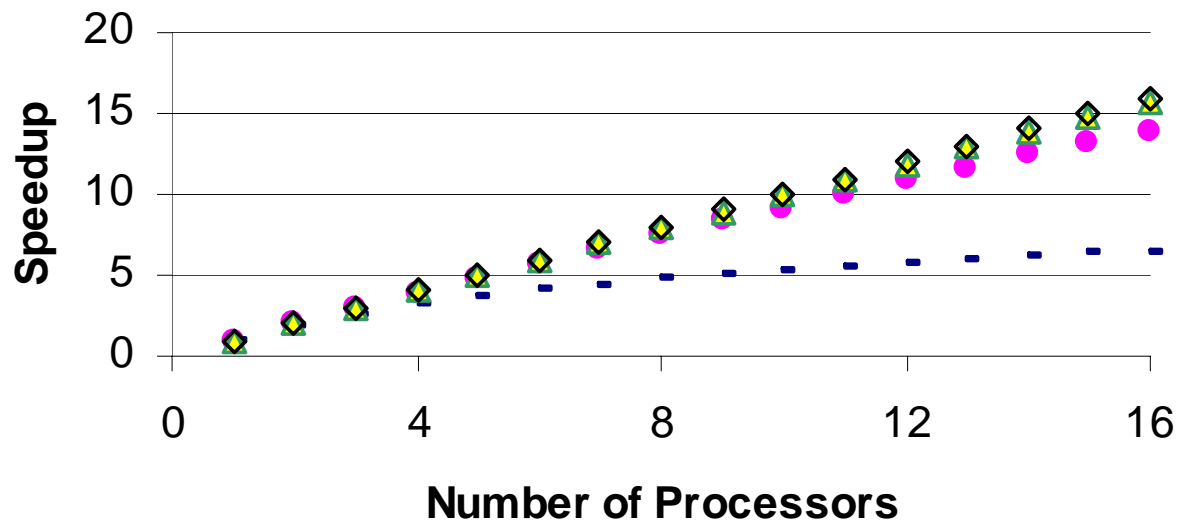
Structure of Multithread Programs



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

- $x = 0.9$ \bullet 0.99 \triangle 0.999 \diamond 1



Multi-Thread = High Performance?

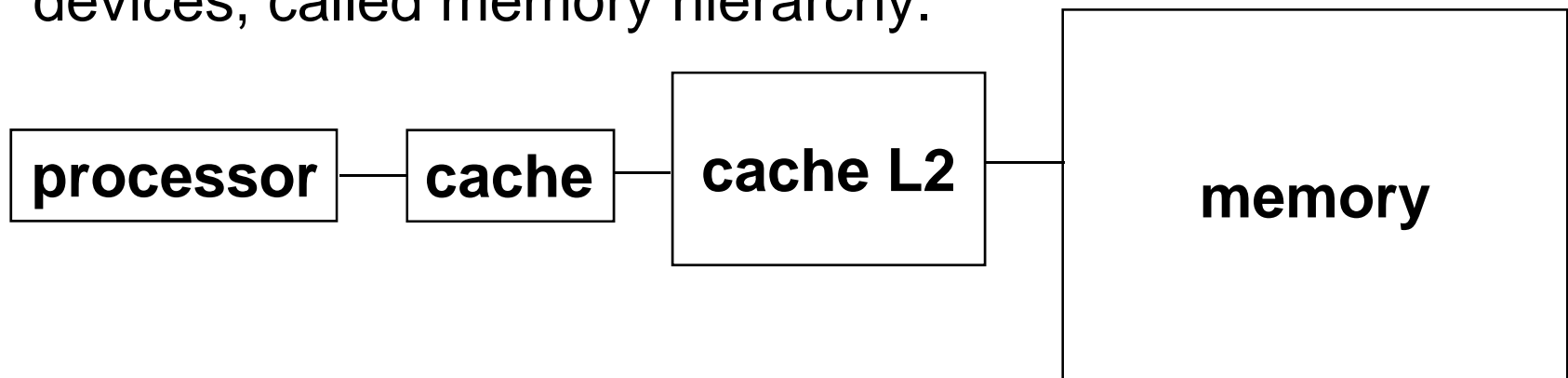
- multi-thread \neq high performance (faster)
- If a program is IO-bounded, multi-thread (or multi-core) does not really help.
- Finding sufficient parallelism (make x closer to 1) can be difficulty.
- Reduce the sequential parts as much as possible
 - read data from multiple, parallel sources
 - partition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

Superlinear Speedup

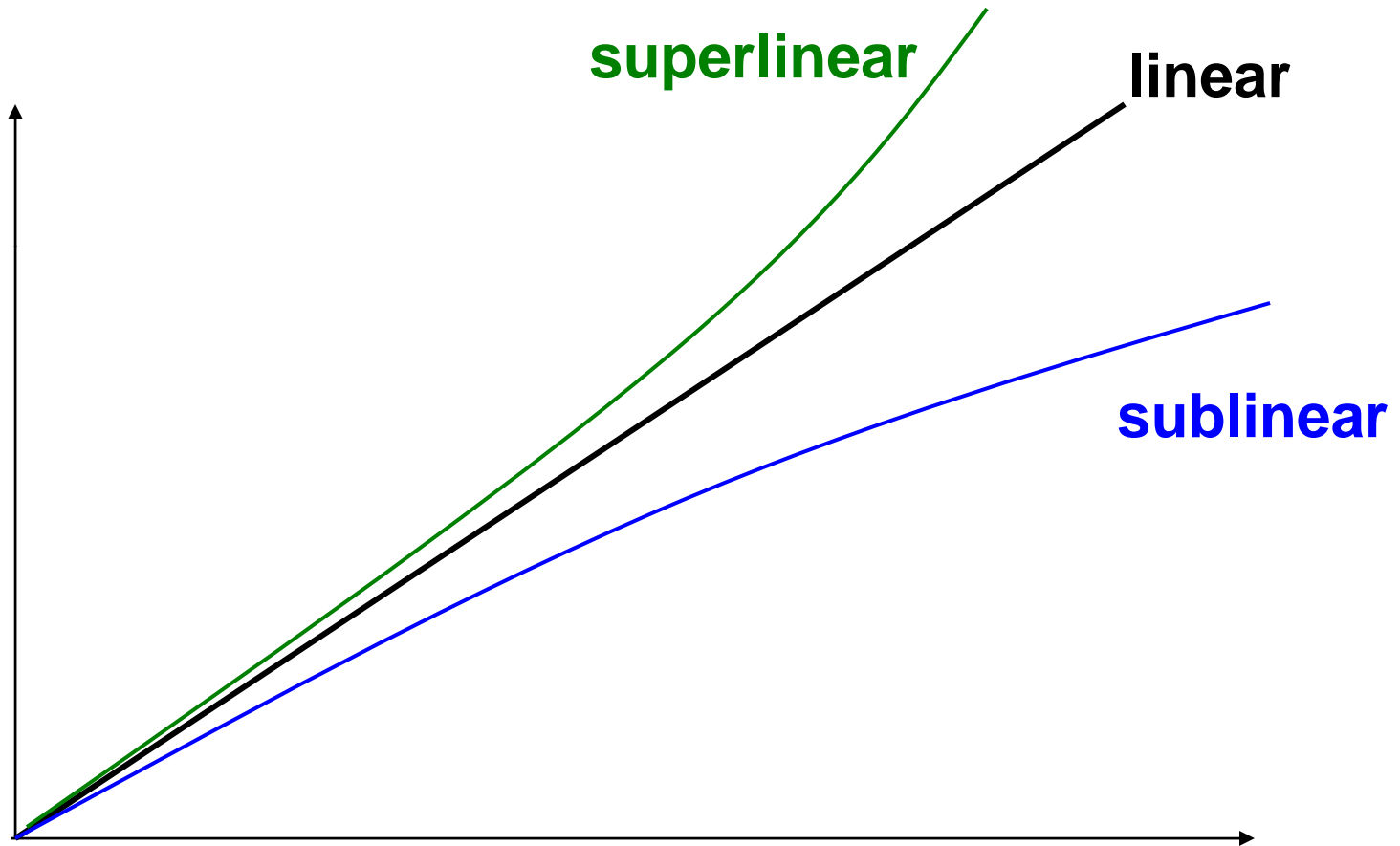
- Sometimes, a multithread program's performance exceeds the number of processors.

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

- 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

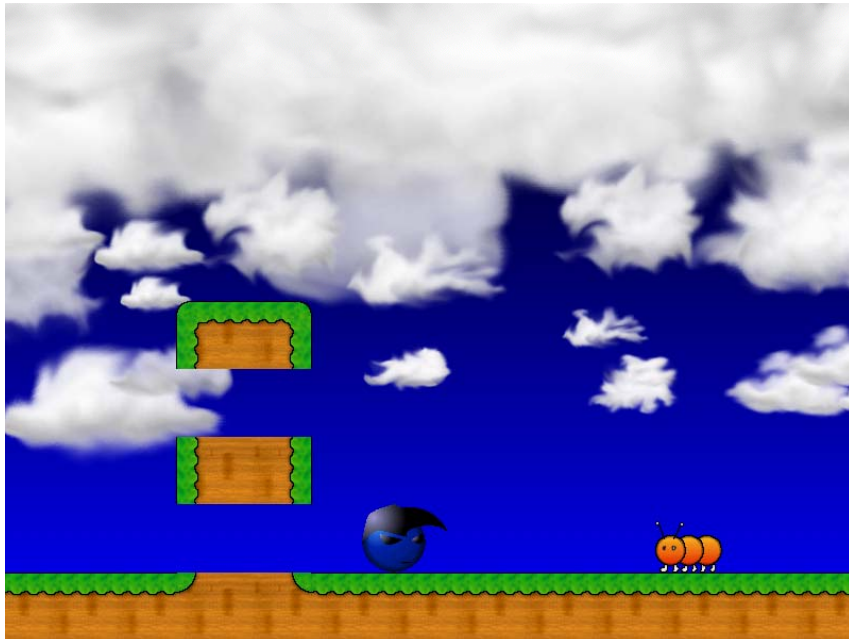
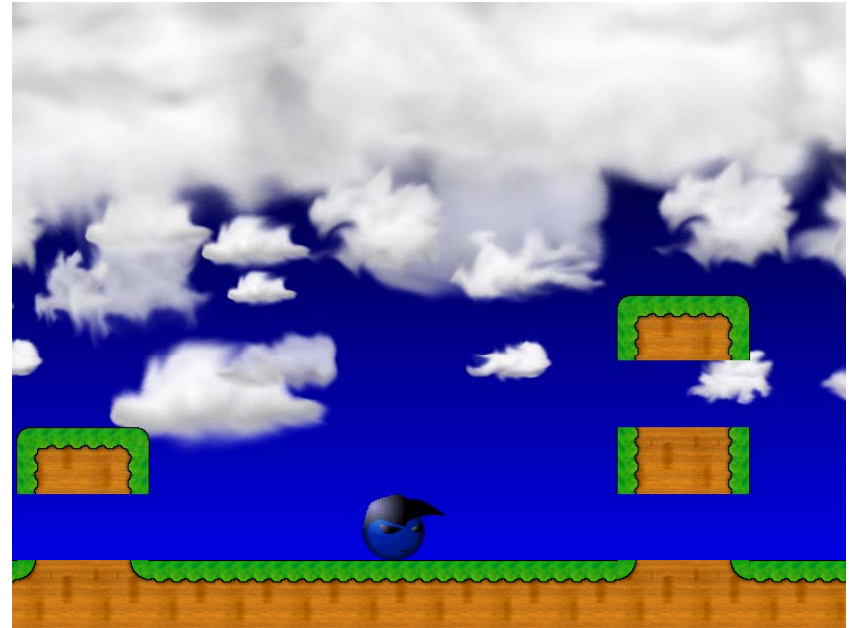
Object-Oriented Programming using C++ and Java

Tile-Based Platform Game

Yung-Hsiang Lu
yunlu@purdue.edu







orm G

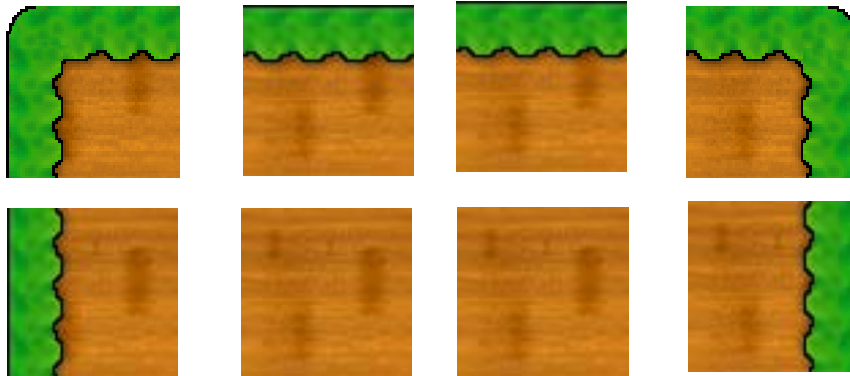
Changing Background

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

- sky



- tiles



Tile Map

- The tiles do not need individual copies of the images
- collision detection based on the type of a tile
- complex map \Rightarrow a special map editor
- simple map \Rightarrow 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.

```

emacs@HELPSTABLET2
File Edit Options Buffers Tools Help

# 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)

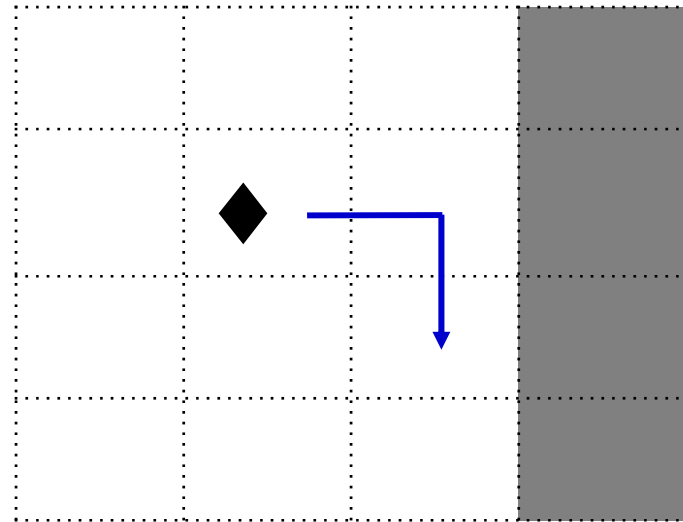
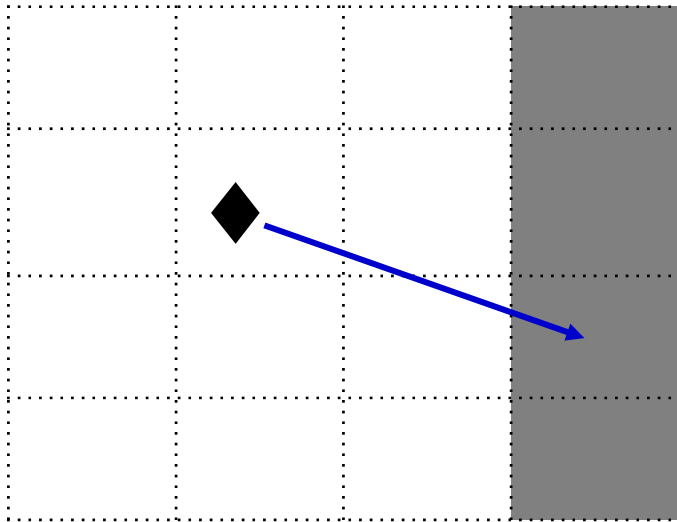
          o o o          o o o          o      o      o      o
          IIIIIII      IIIIIII      o      o      o      o
          2              2EF      2      2
          EF            EGD      2
          EF 1      CD      1      1      EGAD      *
BBBBBBBBBGBBBBBBBBGBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBBGAABBBBBBBBBB

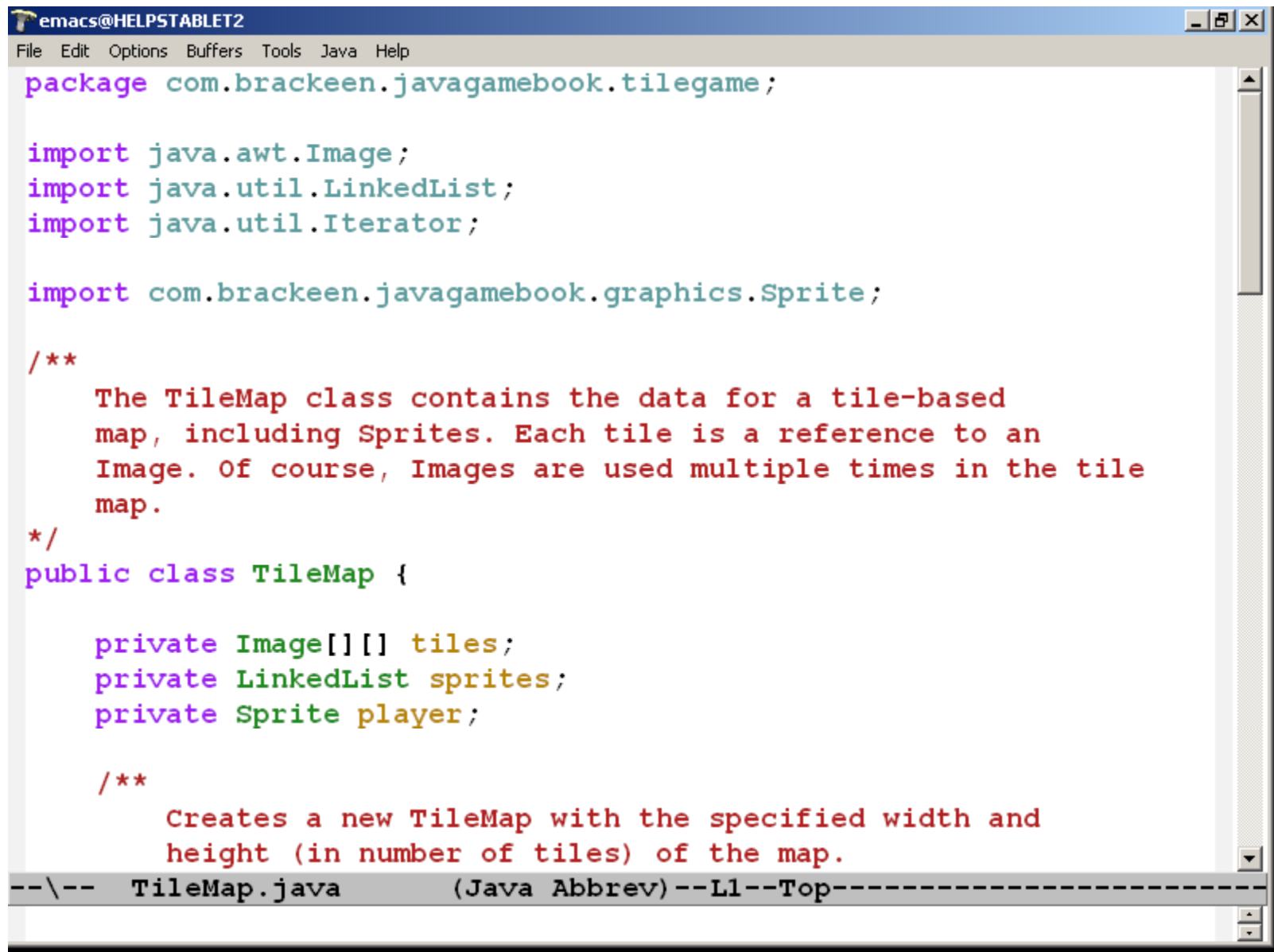
```

Tile-Based Collision Detection

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

`tileMap.get(x,y) \Rightarrow if a tile exists, collision`



The image shows a screenshot of an Emacs editor window. The title bar at the top reads 'emacs@HELPSTABLET2'. Below the title bar is a menu bar with the following items: 'File', 'Edit', 'Options', 'Buffers', 'Tools', 'Java', and 'Help'. The main editing area contains Java code for a class named 'TileMap'. The code is color-coded: package names and class names are in purple, imports are in purple, and comments are in red. The code includes package declarations, imports for 'java.awt.Image', 'java.util.LinkedList', 'java.util.Iterator', and 'com.brackeen.javagamebook.graphics.Sprite'. It also contains a multi-line comment describing the 'TileMap' class and its purpose. The class 'TileMap' is defined as a public class with three private attributes: 'tiles' (an array of 'Image' objects), 'sprites' (a 'LinkedList' of 'Sprite' objects), and 'player' (a 'Sprite' object). A second multi-line comment describes the constructor, which creates a new 'TileMap' with a specified width and height. At the bottom of the window, there is a status bar that reads '--\-- TileMap.java (Java Abbrev) --L1--Top-----'.

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

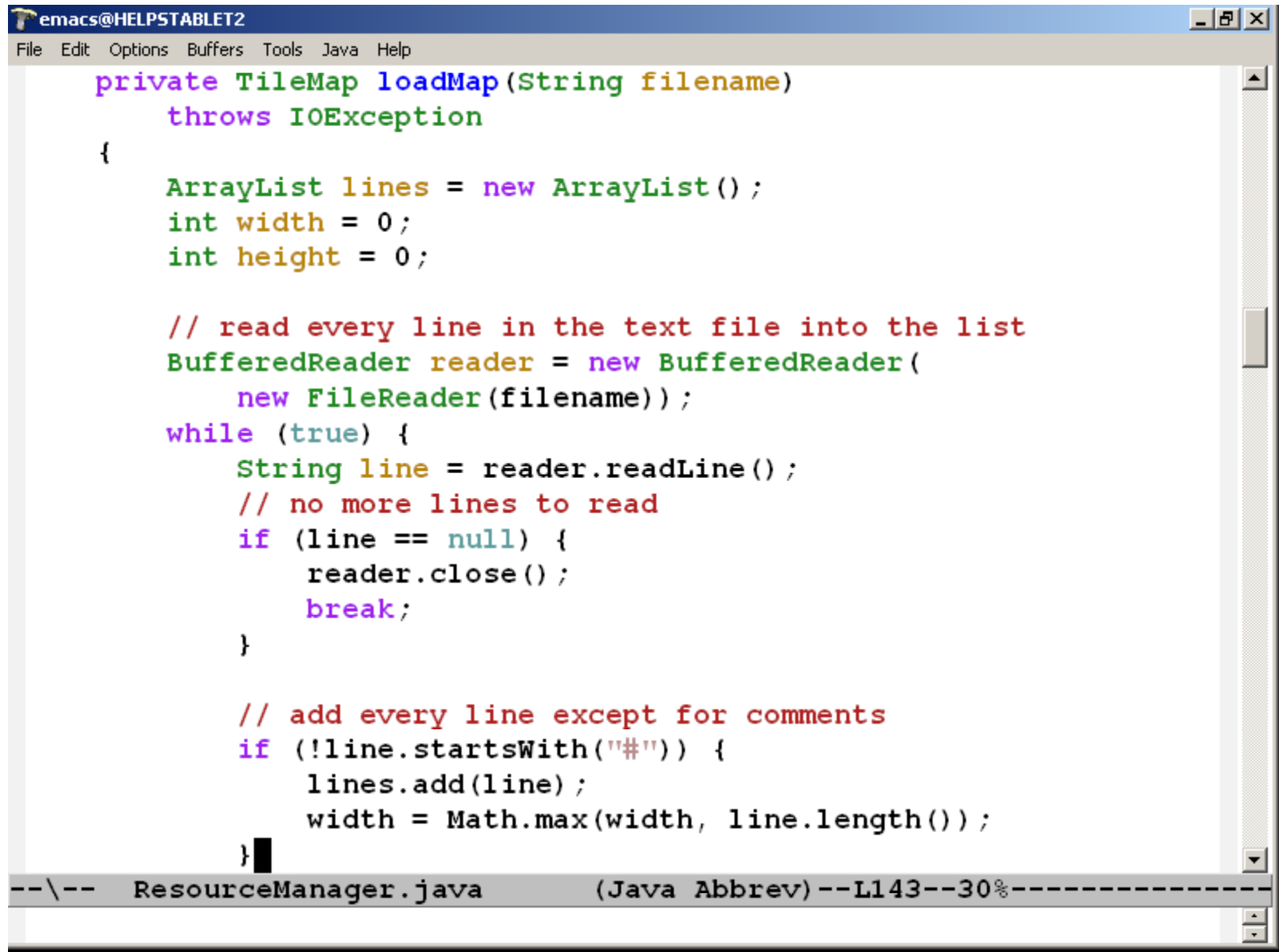
import java.awt.Image;
import java.util.LinkedList;
import java.util.Iterator;

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.
     */
}
```

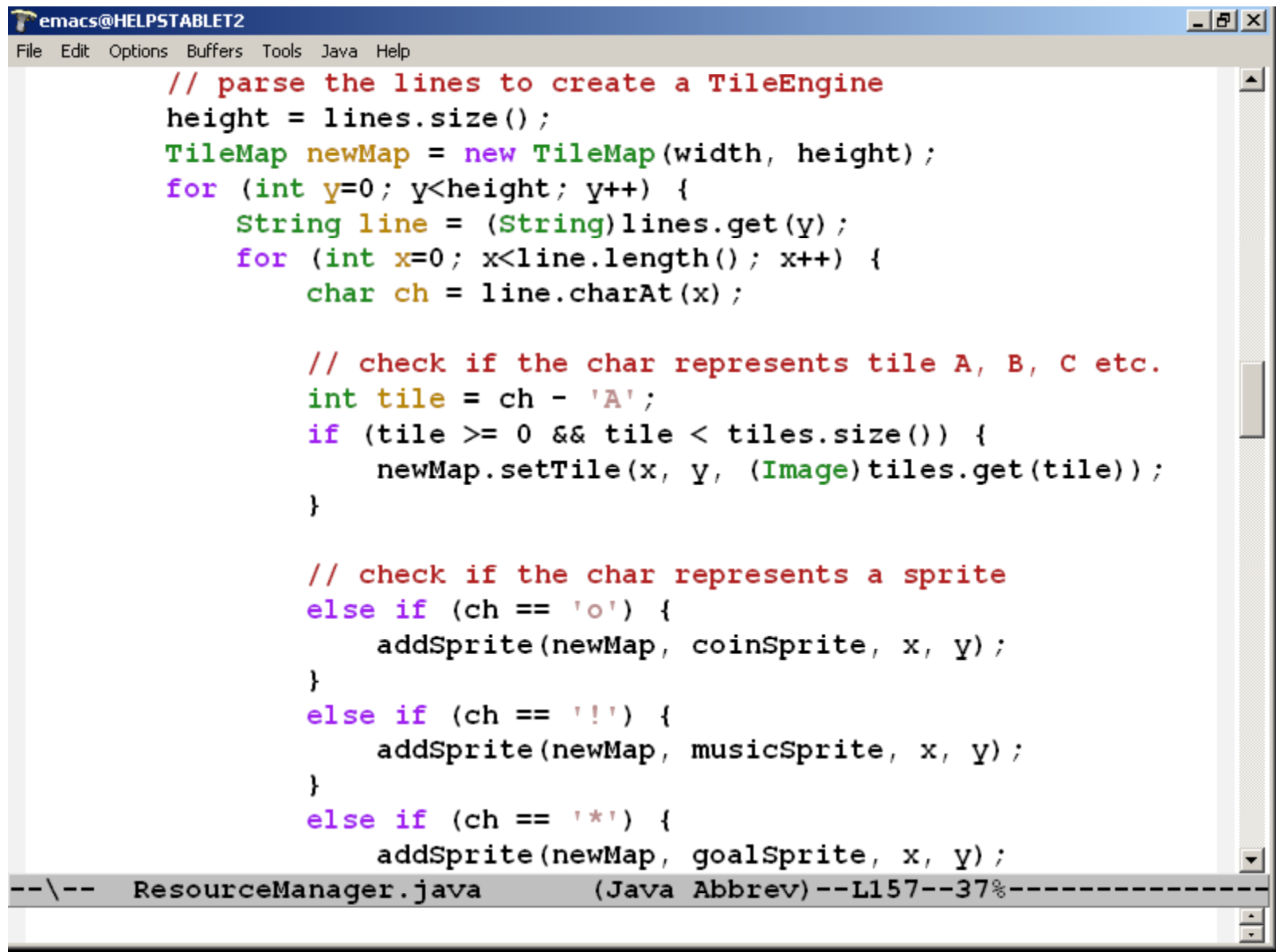
The image shows a screenshot of an Emacs editor window. The title bar at the top reads "emacs@HELPSTABLET2". Below the title bar is a menu bar with the following items: "File", "Edit", "Options", "Buffers", "Tools", "Java", and "Help". The main editing area contains Java code for a method named "loadMap". The code is as follows:

```
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());
        }
    }
}
```

The status bar at the bottom of the window displays "--\-- 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%-----