Lecture #37

xcursion into nitty-gritty stuff: Threads.

Threads

ır programs consist of single sequences of instructions.

equence is called a *thread* (for "thread of control") in

's programs containing *multiple* threads, which (conceptually) ntly.

gram access to threads, Java provides the type `Thread` . Each `Thread` contains information about, and controls,

s access to data from two threads can cause chaos, so lso constructs for controlled communication, allowing *ock* objects, to *wait* to be notified of events, and to her threads.

But Why?

a uniprocessor, only one thread at a time actually runs, wait, but this is largely invisible. So why bother with

programs always have > 1 thread: besides the main ers clean up garbage objects, receive signals, update other stuff.

ms deal with asynchronous events, it is sometimes convenient into subprograms, one for each independent, related events.

w us to insulate one such subprogram from another. ovide a form of modularization.

rganized like this: application is doing some computation her thread waits for mouse clicks (like 'Stop'), another on to updating the screen as needed.

s like search engines may be organized this way, with er request.

se, sometimes we *do* have a real multiprocessor.

Java Mechanics

he actions "walking" and "chewing gum":

```
l implements Runnable {
id run()
 (true) ChewGum(); }

l implements Runnable {
id run()
 (true) Walk(); }
```

```
// Walk and chew gum
Thread chomp
  = new Thread(new Chewer1());
Thread clomp
  = new Thread(new Walker1());
chomp.start(); clomp.start();
```

rnative (uses fact that Thread implements Runnable):

```
 extends Thread {
 run()
true) ChewGum(); }

 extends Thread {
 run()
true) Walk(); }
```

```
Thread chomp = new Chewer2(),
       clomp = new Walker2();
chomp.start();
clomp.start();
```


Avoiding Interference

read has data for another, one must wait for the other

two threads use the same object, generally only one y it at a time; other must wait.

ould happen if two threads simultaneously inserted an nked list at the same point in the list?

d conceivably execute

```
w ListCell(x, p.next);
```

e values of `p` and `p.next`; one insertion is lost.

for only one thread at a time to execute a method on a ject with either of the following equivalent definitions:

```
) {
onized (this) {
dy of f
```

```
synchronized void f(...) {
    body of f
}
```


Communicating the Hard Way

ng data is tricky: the faster party must wait for the

roaches for sending data from thread to thread don't

```
changer {
lue = null;
ceive() {
 r; r = null;
(r == null)
= value; }
= null;
 r;

sit(Object data) {
(value != null) { }
= data;
```

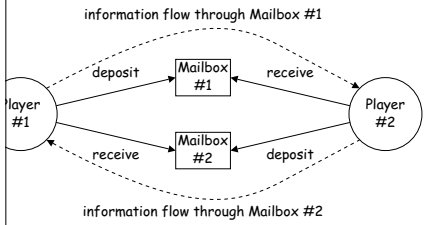
```
DataExchanger exchanger
  = new DataExchanger();

--------------------------------

// thread1 sends to thread2 with
exchanger.deposit("Hello!");

--------------------------------

// thread2 receives from thread1 with
msg = (String) exchanger.receive();
```

hread can monopolize the machine while waiting; two cuting `deposit` or `receive` simultaneously cause chaos.

Message-Passing Style

ava primitives very error-prone. CS162 goes into alternatives.

e higher-level, and allow the following program structure:



Player is a thread that looks like this:

```
ameOver()) {
Move())
tBox.deposit(computeMyMove(lastMove));

stMove = inBox.receive();
```


Primitive Java Facilities

method makes the current thread wait (not using processor) l by `notifyAll`, unlocking the Object while it waits.

`ucb.util.mailbox` has something like this (simplified):

```
ilbox {
osit(Object msg) throws InterruptedException;
eceive() throws InterruptedException;


Mailbox implements Mailbox {
List<Object> queue = new LinkedList<Object>();

ynchronized void deposit(Object msg) {
e.add(msg);
.notifyAll(); // Wake any waiting receivers


ynchronized Object receive() throws InterruptedException {
e (queue.isEmpty()) wait();
rn queue.remove(0);
```


Coroutines

is a kind of synchronous thread that explicitly hands o other coroutines so that only one executes at a time, generators. Can get similar effect with threads and

cursive inorder tree iterator:

```
or extends Thread {
ailbox r;
(Tree T, Mailbox r) {
t = T; this.dest = r;

run() {
(root);
t(End marker);

e(Tree t) {
 null) return;
(t.left);
t(t.label);
(t.right);
```

```
void treeProcessor(Tree T) {
    Mailbox m = new QueuedMailbox();
    new TreeIterator(T, m).start();
    while (true) {
        Object x = m.receive();
        if (x is end marker)
            break;
        do something with x;
    }
}
```


More Concurrency

mple can be done other ways, but mechanism is very

you want to think during opponent's move:

```
meOver()) {
Move())
tBox.deposit(computeMyMove(lastMove));

 {
  thinkAheadALittle();
  lastMove = inBox.receiveIfPossible();
while (lastMove == null);
```

ssible (written `receive(0)` in our actual package) doesn't s `null` if no message yet, perhaps like this:

```
chronized Object receiveIfPossible()
rows InterruptedException {
eue.isEmpty())
turn null;
 queue.remove(0);
```


Highlights of a GUI Component

```
hat draws multi-colored lines indicated by mouse. */
tends JComponent implements MouseListener {
<Point> lines = new ArrayList<Point>();

/ Main thread calls this to create one
edSize(new Dimension(400, 400));
stener(this);

ronized void paintComponent(Graphics g) { // Paint thread
(Color.white);   g.fillRect(0, 0, 400, 400);
  x = y = 200;
Color.black;
 p : lines)
or(c); c = chooseNextColor(c);
ne(x, y, p.x, p.y); x = p.x; y = p.y;


ronized void mouseClicked(MouseEvent e) // Event thread
d(new Point(e.getX(), e.getY())); repaint(); }
```


Use In GUIs

e library uses a special thread that does nothing but *nts* like mouse clicks, pressed keys, mouse movement,

ignate an object of your choice as a *listener;* which ava's event thread calls a method of that object whenever urs.

your program can do work while the UI continues to uttons, menus, etc.

cial thread does all the drawing. You don't have to en this takes place; just ask that the thread wake up u change something.

Interrupts

is an event that disrupts the normal flow of control of

ems, interrupts can be totally *asynchronous,* occurring
points in a program. The Java developers considered
nd arranged that interrupts would occur only at controlled

grams, one thread can interrupt another to inform it
ing unusual needs attention:

```
d.interrupt();
```

read does not receive the interrupt until it waits: methods
(wait for a period of time), join (wait for thread to
nd library methods like mailbox deposit and receive.

causes these methods to throw InterruptedException,
e is like this:

```
= inBox.receive();
nterruptedException e) { HandleEmergency(); }
```


mote Mailboxes (A Side Excursion)

e Method Interface allows one program to refer to
other program.

allow mailboxes in one program to be received from or
into in another.

you define an *interface* to the remote object:

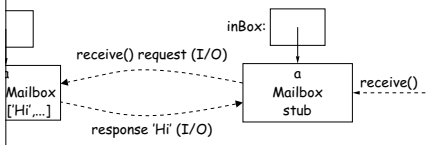
```
a.rmi.*;
Mailbox extends Remote {
eposit(Object msg)
rows InterruptedException, RemoteException;
receive()
rows InterruptedException, RemoteException;
```

ine that actually will contain the object, you define

```
edMailbox ... implements Mailbox {
mplementation as before, roughly
```


Remote Objects Under the Hood

```
#1:              // On Machine #2:
                 Mailbox inBox
edMailbox();       = get outBox from machine #1
```



lbox is an interface type, you don't see whether you are
mailbox or at a (remote) stub that stands in for it.

method calls are relayed by I/O to the machine that
ct.

nt or return type OK if it also implements Remote or
lized—turned into a stream of bytes and back, as can
es and String.

is involved, expect failures, hence every method can
eException (subtype of IOException).