Intro: Creates a communication software that lets someone be a listener and someone else is a speaker. Words are passed from one speaker to one listener without exception. Extra speakers must wait until there are listeners to hear them and vice versa.

Pseudocode Communicator.java

PROCEDURE Communicator() :

communicator() {

creates readyspeaker

creates ready listener

}

void speak(int word){

get the word;

increase number of speakers;

while (no listeners)

sleep();

remove a listener;

get message from word

signal speak is taken

}

Int listen(){

increase number of listeners

while (no speaker)

Sleep;

remove a speaker

return the word

if(speaker waiting)

call speak

}

void sleep()

{

enable

while(always)

{

if(speaker added & sleep for speaker)

counter increment

wake

if(speaker added & sleep for listener)

if(listener added & sleep for speaker)

if(listener added & sleep for listener)

counter increment

wake

}

{

END PROCEDURE

Test Cases

1. Calling listen with no one waiting on the listen side would result in a pile-up of people waiting to speak and cause the thread to sleep
2. Having no one waiting would result in Calling sleep() on multiple threads which makes it so that they are put in a situation where they will move into a waiting state
3. When sleep is called it waits for if there are changes to listeners or speakers.