**Lab 6 Assignment**

Concurrent and Cooperating Threads  
  
For the following two example programs, you are to describe what the output will be when each program is run. If there is more than one possible output, describe all the possibilities. Here are some important facts:  
  
\* Each program is made up of two concurrent threads.  
\* You don't know in what order they will run, nor do you know when the dispatcher will switch between threads. But the dispatcher will eventually and continually switch back and forth between threads(i.e., no threadswill starve). In other words, the instruction interleaving is unpredictable, so you need to trace through \*all\* the interleaving alternatives.  
\* The initialization code (setting the initial values for the shared variables), is completed before either of the two threads run.  
\* All the variables (X and Y) are shared between the two threads.  
\* Every time a variable is referenced (appears in an expression), it is read from memory.  
\* Every time a variable is set (appears on the left-hand size of an assignment operator), it is written to memory.  
\* Reading and writing single words (ints) is atomic.  
  
  
Problem 1  
  
Initialization  
  
int X = 0;  
int Y = 0;  
  
Thread A  
  
for (; X < 4; X++) {  
Y = 0;  
print(X);  
Y = 1;  
}  
  
Thread B  
  
while (X < 4) {  
if (Y == 1)  
print("a");  
}

*/\*\*  
 \* Created by JakeValino on 13/10/2015.  
 \*/***public class** ThreadProblem1 **extends** Thread{  
 *//Instance Variables* **int x** = 0;  
 **int y** = 0;  
  
 **public static void** main(String[] args) {  
 Thread a = **new** Thread(**new** ThreadProblem1(),**"Thread A"**);  
 a.start();  
 }  
  
 *//Default Constructor* **public** ThreadProblem1()  
 {  
 **x** = 0;  
 **y** = 0;  
 }  
  
 *//Overloaded Constructor* **public** ThreadProblem1(**int** xIn,**int** yIn)  
 {  
 **x** = xIn;  
 **y** = yIn;  
 }  
  
 Thread **a** = **new** Thread()  
 {  
 **public void** run()  
 {  
 **for** (; **x** < 4; **x**++) {  
 **y** = 0;  
 System.***out***.println(**x**);  
 **y** = 1;  
 }  
 }  
 };  
  
 Thread **b** = **new** Thread()  
 {  
 **public void** run()  
 {  
 **while** (**x** < 4) {  
 **if** (**y** == 1) {  
 System.***out***.println(**"a"**);  
 }  
 }  
 }  
 };  
  
 **public void** run()  
 {  
 **b**.start();  
  
 **a**.start();  
 }  
}

Describe the output here:

The output is: 0,1,2,3,a.

The output will always stay the same since that even if Thread B is run first, the statements inside its while loop cannot be executed because ‘y’ is initialised to 0. In order for its statements to be executed, ‘y’ must equals to 1. If Thread B is run first, it will not print ‘a’ until Thread A is run and Thread A changes the value of variable ‘y’ to 1. If we use a join in Thread B to cause it to run first and finish before running Thread A, this will create an infinite loop inside it’s while loop because the value of variable ‘y’ will never change from 0.  
  
  
Problem 2  
  
Initialization  
  
int X = 0;  
int Y = 0;  
  
Thread A  
  
while (X == 0) {  
// do nothing  
}  
print("a");  
Y = 1;  
Y = 0;  
print("d");  
Y = 1;  
  
Thread B  
  
print("b");  
X = 1;  
while (Y == 0) {  
// do nothing  
}  
print("c");

*/\*\*  
 \* Created by JakeValino on 13/10/2015.  
 \*/***public class** ThreadProblem2 **extends** Thread{  
 *//Instance Variables* **int x** = 0;  
 **int y** = 0;  
  
 **public static void** main(String[] args) {  
 Thread a = **new** Thread(**new** ThreadProblem2(),**"Thread B"**);  
 a.start();  
 }  
  
 *//Default Constructor* **public** ThreadProblem2()  
 {  
 **x** = 0;  
 **y** = 0;  
 }  
  
 *//Overloaded Constructor* **public** ThreadProblem2(**int** xIn,**int** yIn)  
 {  
 **x** = xIn;  
 **y** = yIn;  
 }  
  
 Thread **a** = **new** Thread()  
 {  
 **public void** run()  
 {  
 **while** (**x** == 0) {  
 *// do nothing* }  
 System.***out***.println(**"a"**);  
 **y** = 1;  
 **y** = 0;  
 System.***out***.println(**"d"**);  
 **y** = 1;  
 }  
 };  
  
 Thread **b** = **new** Thread()  
 {  
 **public void** run()  
 {  
 System.***out***.println(**"b"**);  
 **x** = 1;  
 **while** (**y** == 0) {  
 *// do nothing* }  
 System.***out***.println(**"c"**);  
 }  
 };  
  
  
 **public void** run()  
 {  
 **a**.start();  
 **b**.start();  
 }  
}

Describe the output here:

The output is: b,a,d,c.

The output is always going to stay the same. This is because when Thread A is run, it first enters an infinite while loop (i.e. variable x is initialised to 0 and it doesn’t change until Thread B is run) that does nothing. This causes the change of control to Thread B and Thread A is transitioned into a blocked state. Thread B now runs and prints out the letter ‘b’. It then changes the value of variable x to 1 which cause Thread A to run again. Thread A then prints out the letter ‘a’.It then changes the value of variable y to 1 and then to 0 again. This is to prevent it from leaving Thread A. Thread A then finishes it’s last two statement which prints the letter ‘d’ and changes the value of variable y to 1. The control is transferred to Thread B and with Y=1, it is able to bypass its while loop and print the letter ‘c’.