

**You said:**

```
What is wrong here? import com.phidget22.DigitalInput;  
import com.phidget22.DigitalOutput;
```

```
public class TugOfWar
```

```
{
```

```
    public static void main(String[] args) throws Exception
```

```
{
```

```
{
```

```
        //Variables for each player's button presses
```

```
        int player1Press = 0;
```

```
        int player2Press = 0;
```

```
        //Create | Create objects for your buttons and LEDs.
```

```
        DigitalInput redButton = new DigitalInput();
```

```
        DigitalOutput redLED = new DigitalOutput();
```

```
        DigitalInput greenButton = new DigitalInput();
```

```
        DigitalOutput greenLED = new DigitalOutput();
```

```
        //Address | Address your four objects which lets your program know where to find  
them.
```

```
        redButton.setHubPort(0);
```

```
        redButton.setIsHubPortDevice(true);
```

```
        redLED.setHubPort(1);
```

```
        redLED.setIsHubPortDevice(true);
```

```
        greenButton.setHubPort(5);
```

```
        greenButton.setIsHubPortDevice(true);
```

```
        greenLED.setHubPort(4);
```

```
        greenLED.setIsHubPortDevice(true);
```

```
        //Open | Connect your program to your physical devices.
```

```
        redButton.open(1000);
```

```
        redLED.open(1000);
```

```
        greenButton.open(1000);
```

```
        greenLED.open(1000);
```

```
        //Count player presses
```

```
        while (player1Press < 10 || player2Press < 10)
```

```
{
```

```
        if (redButton.getState() == true)
        {
            player1Press++;
        }

        else if (greenButton.getState() == true)
        {
            player2Press++;
        }

        Thread.sleep(50);
    }

//Flash LEDs when game ends
redLED.setState(true);
greenLED.setState(true);
Thread.sleep(1500);

redLED.setState(false);
greenLED.setState(false);
Thread.sleep(500);

if (player1Press > player2Press)
{
    for (int f = 0; f < 5; f++)
    {
        redLED.setState(true);
        Thread.sleep(150);
        redLED.setState(false);
    }
}

else if (player1Press < player2Press)
{
    for (int f = 0; f < 5; f++)
    {
        greenLED.setState(true);
        Thread.sleep(150);
        greenLED.setState(false);
    }
}
```

```
    }
}
}
}
```

**ChatGPT said:**

There are a few concrete problems here. I will go from most serious to more subtle.

**1. Extra opening brace in main**

Your main method starts like **this**:

```
public static void main(String[] args) throws Exception
{
{
```

There are TWO opening braces, but only one is needed. This creates an extra block that serves no purpose **and** will confuse both you **and** anyone reading the code.

**Fix:**

Remove one of them so it becomes:

```
public static void main(String[] args) throws Exception
{
```

and **delete** the matching extra closing brace near the bottom.

Right now you have three closing braces at the end when only two are required.

**2. Button press counting logic causes runaway counts**

This part is logically wrong **for** a button-based game:

```
if (redButton.getState() == true)
{
    player1Press++;
}
else if (greenButton.getState() == true)
{
    player2Press++;
}
```

If a player holds the button down, `getState` will stay `true` for many loop cycles. Because your loop runs every 50 ms, a single press can add dozens of counts.

That means players do not need to tap 10 times. They can just hold the button for half a second and instantly win.

Correct approach:

You must detect a transition from `false` to `true`, not continuous `true`.

That requires storing the previous button state and only incrementing when it changes from `false` to `true`.

3. Loop condition allows one player to keep increasing after the other wins

Your loop condition is:

```
while (player1Press < 10 || player2Press < 10)
```

This means the loop keeps running as long as at least one player is below 10.

Example:

Player 1 reaches 10 first

Player 2 is still at 3

The loop keeps running and player 1 can keep increasing beyond 10

If your intention is to stop the game when either player reaches 10, this is wrong.

Correct logic would be:

```
while (player1Press < 10 && player2Press < 10)
```

That stops the loop as soon as one player reaches 10.

4. else if blocks simultaneous presses

This part:

```
if (redButton.getState() == true)
{
    player1Press++;
}
else if (greenButton.getState() == true)
{
```

```
    player2Press++;  
}
```

If both buttons are pressed at the same time, only player 1 will ever be counted, because `else if` prevents the second condition from being checked.

If you want both players to be able to score independently, these should be two separate `if` statements, not `if / else if`.

## 5. Missing close calls for Phidgets devices

You open all devices:

```
redButton.open  
redLED.open  
greenButton.open  
greenLED.open
```

But you never close them. This can cause problems when you rerun the program.

Proper cleanup would include calling `close` on all devices after the game ends.

Summary of what is actually wrong

Syntax issue:

Extra opening brace inside main

Logic issues:

Button presses counted continuously instead of per press

Wrong loop condition operator

`else if` blocking simultaneous input

No device cleanup

If you want, I can rewrite `this` entire program cleanly with proper button edge detection and correct game logic so it behaves exactly like a real tug of war game.

You said:

Why isn't this working?  
`import com.phidget22.DigitalInput;`  
`import com.phidget22.DigitalOutput;`

```
public class TugOfWar  
{
```

```
public static void main(String[] args) throws Exception
{
{
    //Variables for each player's button presses
    int player1Press = 0;
    int player2Press = 0;

    //Variables for previous state of button
    boolean prevStateRed;
    boolean prevStateGreen;

    //Create | Create objects for your buttons and LEDs.
    DigitalInput redButton = new DigitalInput();
    DigitalOutput redLED = new DigitalOutput();
    DigitalInput greenButton = new DigitalInput();
    DigitalOutput greenLED = new DigitalOutput();

    //Address | Address your four objects which lets your program know where to find them.
    redButton.setHubPort(0);
    redButton.setIsHubPortDevice(true);
    redLED.setHubPort(1);
    redLED.setIsHubPortDevice(true);
    greenButton.setHubPort(5);
    greenButton.setIsHubPortDevice(true);
    greenLED.setHubPort(4);
    greenLED.setIsHubPortDevice(true);

    //Open | Connect your program to your physical devices.
    redButton.open(1000);
    redLED.open(1000);
    greenButton.open(1000);
    greenLED.open(1000);

    //Count player presses
    while (player1Press < 10 && player2Press < 10)
    {
        //Set button states so you can't hold down the button
        prevStateRed = redButton.getState();
        prevStateGreen = greenButton.getState();
```

```
        if (redButton.getState() == true && prevStateRed == false)
    {
        player1Press++;
    }

        if (greenButton.getState() == true && prevStateGreen == false)
    {
        player2Press++;
    }

        Thread.sleep(50);
    }

//Flash LEDs when game ends
redLED.setState(true);
greenLED.setState(true);
Thread.sleep(1500);

redLED.setState(false);
greenLED.setState(false);
Thread.sleep(500);

if (player1Press > player2Press)
{
    for (int f = 0; f < 5; f++)
    {
        redLED.setState(true);
        Thread.sleep(150);
        redLED.setState(false);
    }
}

if (player1Press < player2Press)
{
    for (int f = 0; f < 5; f++)
    {
        greenLED.setState(true);
        Thread.sleep(150);
        greenLED.setState(false);
    }
}
```

```
    }
}
}
}
}
```

**ChatGPT said:**

There are TWO main reasons `this` still `is not` working, and one leftover structural issue. The logic error `is` the real killer.

I'll go step by step.

-----  
1. You are overwriting the previous state BEFORE you check it  
-----

This `is` the core bug.

Inside your `while` loop you `do this`:

```
prevStateRed = redButton.getState();
prevStateGreen = greenButton.getState();
```

then immediately after:

```
if (redButton.getState() == true && prevStateRed == false)
```

That can NEVER be `true`.

Why?

Because `prevStateRed` was just `set` to the CURRENT state.

So logically you are checking:

```
if (currentState == true AND currentState == false)
```

That condition `is` impossible.

So the counter will never increment.

-----  
2. How edge detection `is` supposed to work  
-----

You need to:

```
1) Read the CURRENT state  
2) Compare it to the PREVIOUS state  
3) Increment if it changed from false to true  
4) THEN update the previous state for the NEXT loop
```

Right now you are doing step 4 before step 2.

---

### 3. Correct button logic structure

---

Inside the loop it must look like this conceptually:

```
boolean currentRed = redButton.getState();  
boolean currentGreen = greenButton.getState();  
  
if (currentRed == true && prevStateRed == false)  
{  
    player1Press++;  
}  
  
if (currentGreen == true && prevStateGreen == false)  
{  
    player2Press++;  
}  
  
prevStateRed = currentRed;  
prevStateGreen = currentGreen;
```

---

### 4. You never initialized prevStateRed and prevStateGreen

---

These variables are declared but never assigned before the loop starts.

Java will not allow reading uninitialized local variables.

You must initialize them ONCE before the loop, usually right after opening the devices:

```
prevStateRed = redButton.getState();  
prevStateGreen = greenButton.getState();
```

---

### 5. You STILL have an extra brace in main

---

```
You still have this:
```

```
public static void main(String[] args) throws Exception
{
{
```

That extra opening brace is unnecessary and confusing.

It should be:

```
public static void main(String[] args) throws Exception
{
```

and remove the matching extra closing brace near the end.

```
-----
6. Minimal fixed loop (this WILL work)
-----
```

Replace your `while` loop logic with this pattern:

```
prevStateRed = redButton.getState();
prevStateGreen = greenButton.getState();

while (player1Press < 10 && player2Press < 10)
{
    boolean currentRed = redButton.getState();
    boolean currentGreen = greenButton.getState();

    if (currentRed && !prevStateRed)
    {
        player1Press++;
    }

    if (currentGreen && !prevStateGreen)
    {
        player2Press++;
    }

    prevStateRed = currentRed;
    prevStateGreen = currentGreen;

    Thread.sleep(50);
}
```

```
-----
```

Why `this` fixes everything

---

- Button presses are counted once per press
- Holding the button does nothing
- Both players can score simultaneously
- Game stops `as soon as` one player reaches `10`

If you want, I can rewrite the FULL program cleanly `with` comments removed and structure `fixed` so it looks like a polished final submission.

You said:

This still isn't working(ignore extra curly brace) import com.phidget22.DigitalInput;  
import com.phidget22.DigitalOutput;

```
public class TugOfWar
```

```
{
```

```
    public static void main(String[] args) throws Exception {
```

```
{
```

```
        //Variables for each player's button presses
```

```
        int player1Press = 0;
```

```
        int player2Press = 0;
```

```
        //Create | Create objects for your buttons and LEDs.
```

```
        DigitalInput redButton = new DigitalInput();
```

```
        DigitalOutput redLED = new DigitalOutput();
```

```
        DigitalInput greenButton = new DigitalInput();
```

```
        DigitalOutput greenLED = new DigitalOutput();
```

```
        //Address | Address your four objects which lets your program know where to find them.
```

```
        redButton.setHubPort(0);
```

```
        redButton.setIsHubPortDevice(true);
```

```
        redLED.setHubPort(1);
```

```
        redLED.setIsHubPortDevice(true);
```

```
        greenButton.setHubPort(5);
```

```
        greenButton.setIsHubPortDevice(true);
```

```
        greenLED.setHubPort(4);
```

```
        greenLED.setIsHubPortDevice(true);
```

```
//Open | Connect your program to your physical devices.
```

```
redButton.open(1000);
redLED.open(1000);
greenButton.open(1000);
greenLED.open(1000);

//Variables for current state of button
boolean currentRed = redButton.getState();
boolean currentGreen = greenButton.getState();

//Variables for previous state of button
boolean prevStateRed = redButton.getState();
boolean prevStateGreen = greenButton.getState();

//Count player presses
while (player1Press < 10 && player2Press < 10)
{
    if (currentRed == true && prevStateRed == false)
    {
        player1Press++;
    }

    if (currentGreen == true && prevStateGreen == false)
    {
        player2Press++;
    }

    prevStateRed = currentRed;
    prevStateGreen = currentGreen;
}

//Flash LEDs when game ends
redLED.setState(true);
greenLED.setState(true);
Thread.sleep(1500);

redLED.setState(false);
greenLED.setState(false);
Thread.sleep(500);

if (player1Press > player2Press)
```

```
    {
        for (int f = 0; f < 5; f++)
        {
            redLED.setState(true);
            Thread.sleep(150);
            redLED.setState(false);
        }
    }

    if (player1Press < player2Press)
    {
        for (int f = 0; f < 5; f++)
        {
            greenLED.setState(true);
            Thread.sleep(150);
            greenLED.setState(false);
        }
    }
}
```

**ChatGPT said:**

Good – now the problem is very clear. The code compiles, but the logic can NEVER change state, so the loop never works.

There is ONE fatal logic mistake and TWO secondary issues.

-----  
THE MAIN REASON IT DOES NOT WORK  
-----

You read the button states ONCE, before the loop:

```
boolean currentRed = redButton.getState();
boolean currentGreen = greenButton.getState();
```

Then inside the while loop, you NEVER update them.

That means for the entire game:

```
currentRed never changes
currentGreen never changes
```

So your loop is effectively doing this forever:

```
if (sameValue && sameValue) { ... }
```

No new button presses are ever detected.

---

#### WHY THIS BREAKS EDGE DETECTION

---

Edge detection requires this EVERY LOOP:

- 1) Read the current hardware state
- 2) Compare it to the previous state
- 3) Update previous state

You are doing steps 2 and 3, but you NEVER do step 1 again.

---

#### SECOND ISSUE: INFINITE TIGHT LOOP

---

There is no Thread.sleep inside your while loop.

So even if it \*did\* work, the program would:

- Run at full CPU speed
- Spam state checks thousands of times per second
- Make debugging nearly impossible

---

#### THE FIX (MINIMAL CHANGES)

---

Move the state reads INSIDE the loop.

Your loop MUST look like this:

```
while (player1Press < 10 && player2Press < 10)
{
    currentRed = redButton.getState();
    currentGreen = greenButton.getState();

    if (currentRed && !prevStateRed)
    {
        player1Press++;
    }
}
```

```
    if (currentGreen && !prevStateGreen)
    {
        player2Press++;
    }

    prevStateRed = currentRed;
    prevStateGreen = currentGreen;

    Thread.sleep(50);
}
```

---

WHY THIS VERSION WORKS

---

- Reads the hardware every cycle
- Detects false → true transitions correctly
- Prevents button holding from cheating
- Allows simultaneous presses
- Prevents CPU lockup

---

MENTAL MODEL (IMPORTANT)

---

currentState = what the button is doing RIGHT NOW  
prevState = what the button was doing LAST LOOP

If you don't update currentState every loop, nothing can ever change.

---

FINAL NOTE

---

You are actually VERY close. This is a classic beginner hardware-programming bug, not a Java syntax issue.

If you want, I can now:

- Add LED progress feedback
- Add a reset button
- Make the game best-of-3
- Clean this into a final submission-ready version

Just say the word.

You said:

In the for loop, my light only flashes once, but I want it to flash 5 times. How do i fix it?

ChatGPT said:

This is happening because your LED is turning ON and OFF too fast for your eyes to see more than one flash.

Right now your loop looks like this (conceptually) :

```
for (int f = 0; f < 5; f++)  
{  
    led.setState(true);  
    Thread.sleep(150);  
    led.setState(false);  
}
```

What's missing is a delay AFTER turning the LED OFF.

So what actually happens is:

- LED turns on
- stays on for 150 ms
- turns off
- immediately turns back on again (too fast to see the off time)

Your eyes interpret this as one long flash.

---

THE FIX

---

You must add a sleep AFTER turning the LED off.

Correct version:

```
for (int f = 0; f < 5; f++)  
{  
    redLED.setState(true);  
    Thread.sleep(150);  
  
    redLED.setState(false);  
    Thread.sleep(150);  
}
```

Now each flash has:

- ON time
- OFF time

---

WHY THIS WORKS

```
-----  
Human vision needs a noticeable OFF gap to distinguish flashes.
```

Without the second sleep:

- The LED technically flashes 5 times
- But visually appears as one flash

```
-----  
OPTIONAL IMPROVEMENT (CLEANER)  
-----
```

You can make the timing clearer by using constants:

```
int flashDelay = 150;  
  
for (int f = 0; f < 5; f++)  
{  
    redLED.setState(true);  
    Thread.sleep(flashDelay);  
  
    redLED.setState(false);  
    Thread.sleep(flashDelay);  
}
```

```
-----  
SUMMARY  
-----
```

Problem: No delay after turning LED off

Fix: Add Thread.sleep after setState(false)

Result: LED visibly flashes 5 times

This is a VERY common beginner hardware timing issue – you did nothing “wrong”, you just need visible timing gaps.

You said:

How could i make it so that the thread.sleep only affects the timer and not the rest of the loop?

//Add Phidgets Library | You added a file called phidget22 when configuring your project.

Import gives you access to the Phidgets library code inside that file.

```
import com.phidget22.*;
```

```
public class Thermostat  
{
```

```
public static void main(String[] args) throws Exception{

    //Create | This allows you to access data from your Phidget.
    HumiditySensor humiditySensor = new HumiditySensor();
    TemperatureSensor temperatureSensor = new TemperatureSensor();
    DigitalOutput greenLED = new DigitalOutput();
    DigitalOutput redLED = new DigitalOutput();
    DigitalInput redButton = new DigitalInput();
    DigitalInput greenButton = new DigitalInput();

    //Locate objects
    redButton.setHubPort(0);
    redButton.setIsHubPortDevice(true);
    redLED.setHubPort(1);
    redLED.setIsHubPortDevice(true);
    greenButton.setHubPort(5);
    greenButton.setIsHubPortDevice(true);
    greenLED.setHubPort(4);
    greenLED.setIsHubPortDevice(true);

    //Open | Open establishes a connection between your object and your physical
    Phidget. You provide a timeout value of 1000 to give the program 1000 milliseconds (1
    second) to locate your Phidget. If your Phidget can't be found, an exception occurs.
    humiditySensor.open(1000);
    temperatureSensor.open(1000);
    redLED.open(1000);
    greenLED.open(1000);
    redButton.open(1000);
    greenButton.open(1000);

    //Create set temperature variable
    double setTemp = 21;

    //Create timer variable
    int timer = 0;

    //Loop for thermostat function
    while (true)
    {
        //Determine what LED to light up depending on set temperature
    }
}
```

```
if (temperatureSensor.getTemperature() < (setTemp - 2) ||
temperatureSensor.getTemperature() > (setTemp + 2))
{
    greenLED.setState(false);
    redLED.setState(true);
}
else
{
    redLED.setState(false);
    greenLED.setState(true);
}

//Increase and decrease set temperature
if (redButton.getState() == true)
{
    setTemp--;
}
if (greenButton.getState() == true)
{
    setTemp++;
}

//Print current and set temperature every 10 seconds
if (timer == 10)
{
    System.out.println("Temperature: " + temperatureSensor.getTemperature()
+ " °C");
    System.out.println("Set Temperature: " + setTemp);
    timer = 0;
}
Thread.sleep(1000);
timer++;
}

}
```

**ChatGPT said:**

This is a GREAT question – and it shows you're hitting a real limitation of single-threaded programs.

Short answer:

You can't make `Thread.sleep` pause only part of a loop.

`Thread.sleep` ALWAYS pauses the entire thread.

So to fix `this` properly, you need to separate "timing" from "logic".

I'll explain `this` cleanly, then show TWO correct `solutions` (one simple, one more advanced).

#### =====

#### WHY YOUR CURRENT CODE BEHAVES THIS WAY

#### =====

Your program has ONE thread: the main thread.

This line:

```
Thread.sleep(1000);
```

literally tells Java:

"Pause EVERYTHING in `this` thread for 1 second."

That includes:

- Button checks
- LED updates
- Temperature reading
- Timer counting

There is no way to scope `Thread.sleep` to "just the timer" inside the same thread.

#### =====

#### SOLUTION 1 (RECOMMENDED FOR THIS LEVEL)

#### =====

Use time-based logic instead of sleep-based logic

Instead of:

- Sleeping for 1 second
- Incrementing a counter

You:

- Continuously run the loop
- Check elapsed time using the system clock

```
This keeps the loop responsive.
```

```
-----  
How this works conceptually  
-----
```

```
You store the last time something happened.
```

```
Each loop, you ask:
```

```
"Has 10 seconds passed yet?"
```

```
-----  
CODE CHANGE  
-----
```

```
Add this BEFORE the loop:
```

```
long lastPrintTime = System.currentTimeMillis();
```

```
Then replace your timer logic with this INSIDE the loop:
```

```
long currentTime = System.currentTimeMillis();

if (currentTime - lastPrintTime >= 10000)
{
    System.out.println("Temperature: " +
temperatureSensor.getTemperature() + " °C");
    System.out.println("Set Temperature: " + setTemp);
    lastPrintTime = currentTime;
}
```

```
And REMOVE these lines entirely:
```

```
int timer = 0;
Thread.sleep(1000);
timer++;
```

```
Optionally, add a very small sleep to prevent CPU abuse:
```

```
Thread.sleep(50);
```

```
-----  
WHY THIS WORKS  
-----
```

- Loop runs continuously
- Buttons respond instantly
- LEDs update immediately

- Printing still happens every 10 seconds
- No logic is blocked waiting

This is how real embedded systems work.

```
=====
SOLUTION 2 (ADVANCED, MULTITHREADING)
Use a second thread for the timer
=====
```

If you want the timer to run independently, you create a separate thread.

Example concept (not required for your course yet):

```
new Thread(() ->
{
    while (true)
    {
        try
        {
            Thread.sleep(10000);
            System.out.println("Temperature: " +
temperatureSensor.getTemperature() + " °C");
            System.out.println("Set Temperature: " + setTemp);
        }
        catch (Exception e) {}
    }
}).start();
```

Meanwhile, your main while loop runs with NO sleep or a tiny one.

```
-----
WHY THIS IS MORE COMPLEX
-----
```

- Shared variables need synchronization
- Harder to debug
- Overkill for a thermostat assignment

```
=====
WHAT YOUR TEACHER EXPECTS
=====
```

Almost certainly Solution 1.

Using:  
System.currentTimeMillis

```
or  
System.nanoTime
```

```
shows you understand:  
• Event timing  
• Non-blocking loops  
• Real-time logic
```

```
=====  
SUMMARY  
=====
```

```
Thread.sleep ALWAYS pauses the whole loop.  
You cannot partially sleep a thread.
```

```
Correct fix:  
Stop using sleep as a timer.  
Use elapsed-time checks instead.
```

```
If you want, I can rewrite your full thermostat loop cleanly using the  
time-based approach so you can submit it confidently.
```

You said:

Using this setup, the thing is displayed later than 10 seconds and it keeps getting longer everytime. why is that?

//Add Phidgets Library | You added a file called phidget22 when configuring your project.  
Import gives you access to the Phidgets library code inside that file.

```
import com.phidget22.*;
```

```
public class Thermostat  
{  
    public static void main(String[] args) throws Exception{
```

```
//Create | This allows you to access data from your Phidget.  
HumiditySensor humiditySensor = new HumiditySensor();  
TemperatureSensor temperatureSensor = new TemperatureSensor();  
DigitalOutput greenLED = new DigitalOutput();  
DigitalOutput redLED = new DigitalOutput();  
DigitalInput redButton = new DigitalInput();  
DigitalInput greenButton = new DigitalInput();
```

```
//Locate objects  
redButton.setHubPort(0);
```

```
redButton.setIsHubPortDevice(true);
redLED.setHubPort(1);
redLED.setIsHubPortDevice(true);
greenButton.setHubPort(5);
greenButton.setIsHubPortDevice(true);
greenLED.setHubPort(4);
greenLED.setIsHubPortDevice(true);
```

//Open | Open establishes a connection between your object and your physical Phidget. You provide a timeout value of 1000 to give the program 1000 milliseconds (1 second) to locate your Phidget. If your Phidget can't be found, an exception occurs.

```
humiditySensor.open(1000);
temperatureSensor.open(1000);
redLED.open(1000);
greenLED.open(1000);
redButton.open(1000);
greenButton.open(1000);
```

//Create set temperature variable  
double setTemp = 21;

//Create timer variable  
int timer = 0;

//Loop for thermostat function  
while (true)
{
 //Determine what LED to light up depending on set temperature
 if (temperatureSensor.getTemperature() < (setTemp - 2) ||
 temperatureSensor.getTemperature() > (setTemp + 2))
 {
 greenLED.setState(false);
 redLED.setState(true);
 }
 else
 {
 redLED.setState(false);
 greenLED.setState(true);
 }
}

```

//Increase and decrease set temperature
if (redButton.getState() == true)
{
    setTemp--;
}

if (greenButton.getState() == true)
{
    setTemp++;
}

//Print current and set temperature every 10 seconds
if (timer == 100)
{
    System.out.println("Temperature: " + temperatureSensor.getTemperature()
+ " °C");
    System.out.println("Set Temperature: " + setTemp);
    timer = 0;
}

Thread.sleep(100);
timer++;
}

}

```

**ChatGPT said:**

What you are seeing is \*\*timing drift\*\*, and it is happening for a very specific (and very important) reason.

Short version:

Your loop does \*\*not\*\* take exactly 100 ms to run.  
So `Thread.sleep(100)` + "everything else" > 100 ms.  
That extra time accumulates every loop, so your "10 seconds" slowly turns into 11, 12, 13... seconds.

Let's break it down clearly.

---

=====

WHAT YOU \*THINK\* IS HAPPENING

=====

```
You are assuming this:
```

- Each loop = 100 ms
- 100 loops = 10 seconds

```
So:
```

```
Thread.sleep(100);  
timer++;
```

```
→ print at timer == 100 → exactly 10 seconds
```

```
But that assumption is false.
```

```
=====  
WHAT IS ACTUALLY HAPPENING  
=====
```

```
Each loop really looks like this:
```

1. Read temperature sensor
2. Compare values
3. Set LED states
4. Read button states
5. Do comparisons
6. Print (sometimes)
7. Call Thread.sleep(100)
8. Wake up \*sometime after\* 100 ms (not exactly)
9. Loop again

```
So the real time per loop is:
```

```
100 ms (sleep)  
+ code execution time  
+ OS scheduling delay  
+ JVM overhead
```

```
Example (very realistic):
```

- Sleep = ~100 ms
- Code = ~5-15 ms
- OS delay = ~1-10 ms

```
Actual loop time ≈ **110-125 ms**
```

```
=====  
WHY THE DELAY GETS WORSE OVER TIME
```

```
=====
```

That extra time happens **every single loop**.

If you are off by just 10 ms per loop:

```
10 ms × 100 loops = 1000 ms (1 second late)
```

After a minute, you might be **6-8 seconds behind**.

That's exactly what you are observing.

This is NOT a bug in Java.

This is expected behavior in real systems.

```
=====
```

```
=====
```

WHY Thread.sleep IS NOT A TIMER

```
=====
```

Thread.sleep means:

```
"Do not run this thread for *at least* this long."
```

It does NOT mean:

"Wake me up exactly on time."

```
Java, Windows, and your CPU make no guarantees about precision.
```

```
=====
```

THE CORRECT WAY TO DO THIS (NO DRIFT)

```
=====
```

You must measure **real elapsed time**, not loop counts.

Replace your timer logic entirely.

```
-----
```

```
CORRECT TIMER APPROACH
```

```
-----
```

Before the loop:

```
long lastPrintTime = System.currentTimeMillis();
```

Inside the loop:

```
long now = System.currentTimeMillis();
```

```
if (now - lastPrintTime >= 10_000)
{
    System.out.println("Temperature: " +
temperatureSensor.getTemperature() + " °C");
    System.out.println("Set Temperature: " + setTemp);
    lastPrintTime += 10_000;
}
```

Then keep a \*\*small sleep\*\* just to reduce CPU load:

```
Thread.sleep(20);
```

```
=====
WHY `lastPrintTime += 10_000` MATTERS
=====
```

This keeps the timer aligned to real time.

If one cycle is late:

- It corrects itself
- It does NOT drift further

If you instead wrote:

```
lastPrintTime = now;
```

You would reintroduce drift.

```
=====
WHY YOUR TEACHER WILL LIKE THIS
=====
```

This shows understanding of:

- Real-time systems
- Non-blocking loops
- OS scheduling
- Why sleep-based timing is unreliable

This is exactly how professional control systems work.

```
=====
SUMMARY
=====
```

Your delay gets longer because:

- Thread.sleep is imprecise
- Your loop does extra work
- Small delays accumulate

Correct fix:

- Stop counting loops
- Measure real elapsed time
- Use System.currentTimeMillis

You didn't do anything "wrong" – you just hit a \*\*classic timing trap\*\* that almost everyone hits the first time they write real-time code.

If you want, I can rewrite your \*\*entire thermostat loop\*\* cleanly with drift-free timing and button debouncing.

You said:

i have to use thread.sleep though as i have not learned anything else

ChatGPT said:

That constraint actually helps – it means the answer your teacher expects is a \*\*sleep-based explanation\*\*, not a system-time solution.

So here is the correct explanation \*\*and\*\* the best fix you can do while still using `Thread.sleep`.

=====

WHY IT KEEPS GETTING LATER (IN SIMPLE TERMS)

=====

You are assuming:

Thread.sleep(100) = exactly 100 ms

That is \*\*not true\*\*.

What it really means is:

"Sleep for \*\*at least\*\* 100 ms."

Your loop time is actually:

sleep time  
+ code execution time  
+ OS scheduling delay

So every loop takes \*\*longer than 100 ms\*\*.

```
That extra time happens **every loop**, so it adds up.
```

That's why:

- First print might be ~10.3 s
- Next might be ~10.6 s
- Then ~11.0 s
- Then ~11.5 s

```
Nothing is "wrong" with your code – this is how `Thread.sleep` works.
```

```
=====
WHY YOU CAN'T FIX DRIFT PERFECTLY WITH SLEEP
=====
```

As long as you do this:

```
Thread.sleep(100);
timer++;
```

```
You are measuring **iterations**, not **time**.
```

```
And iterations are NOT constant-time.
```

So drift is unavoidable.

```
=====
BEST POSSIBLE FIX *USING ONLY Thread.sleep*
=====
```

```
Instead of sleeping 100 ms, you make the sleep **slightly shorter** to
compensate for execution time.
```

```
This does NOT eliminate drift, but it reduces it a lot.
```

```
-----  
Practical fix  
-----
```

Change:

```
Thread.sleep(100);
```

to something like:

```
Thread.sleep(80);
```

Then adjust `timer` accordingly:

```
if (timer == 125)    // 125 × 80 ms ≈ 10 s
{
    print stuff
    timer = 0;
}
```

Why this helps:

- Your loop work fills in the missing time
- Total loop time gets closer to 100 ms

This is a \*\*calibration approach\*\*, and it's acceptable at this level.

```
=====
EVEN BETTER (STILL USING Thread.sleep)
=====
```

Use a \*\*1-second sleep\*\*, not 100 ms.

That massively reduces accumulated error.

Example:

```
int timer = 0;

while (true)
{
    // logic

    Thread.sleep(1000);
    timer++;

    if (timer == 10)
    {
        print stuff
        timer = 0;
    }
}
```

Why this works better:

- 10 sleeps instead of 100
- Less accumulated scheduling error
- Much easier to reason about

```
=====
WHAT YOUR TEACHER EXPECTS YOU TO SAY
=====
```

```
If this is a written or verbal question, the correct answer is:
```

```
"The delay increases because Thread.sleep is not exact, and the loop takes extra time to run. That extra time adds up every cycle, causing drift."
```

```
That is the **exact correct explanation**.
```

```
=====
SUMMARY (VERY IMPORTANT)
=====
```

- Thread.sleep is NOT a precise timer
- Loop execution time adds delay
- Small delays accumulate
- This causes increasing lateness

```
Within your restrictions:
```

- Use longer sleep intervals
- Or compensate by sleeping slightly less
- Or accept small drift (this is realistic)

```
You are not misunderstanding anything – you've hit a **real-world timing limitation**, and recognizing it already puts you ahead of most beginners.
```

**You said:**

Why doesn't this work?

```
//Add Phidgets Library  
import com.phidget22.*;
```

```
public class LEDBrightness {  
    //Handle Exceptions  
    public static void main(String[] args) throws Exception{
```

```
        //Create  
        DigitalOutput redLED = new DigitalOutput();
```

```
        //Address  
        redLED.setHubPort(1);  
        redLED.setIsHubPortDevice(true);
```

```
        //Open  
        redLED.open(1000);
```

```
//Use your Phidgets with Duty Cycle | Duty Cycle controls the power to your  
Phidget (Digital Output object). Duty Cycle values range from 0.0 - 1.0.
```

```
//Create cycle variable  
double cycle = 0.1;  
  
//Loop for increase and decrease in brightness  
redLED.setDutyCycle(cycle);  
cycle += 0.1;  
Thread.sleep(50);  
  
if (cycle == 1.0)  
{  
    while (redLED.getDutyCycle() > 0)  
    {  
        redLED.setDutyCycle(cycle);  
        cycle -= 0.1;  
        Thread.sleep(50);  
    }  
}  
}  
}
```

**ChatGPT said:**

```
There are several reasons this "doesn't work", but one of them is the  
**main killer**, and the rest explain why even fixing that won't behave  
how you expect.
```

I'll go from most important to least.

```
=====  
1. YOUR CODE ONLY RUNS ONCE  
=====
```

This is the biggest problem.

You do NOT have a loop that increases brightness.

This part:

```
redLED.setDutyCycle(cycle);  
cycle += 0.1;
```

```
Thread.sleep(50);
```

runs **\*\*exactly one time\*\***, then the program continues.

So the LED:

- Starts at 0.1
- Immediately becomes 0.2
- Stops forever

There **is NO** loop that keeps increasing the duty cycle.

You **\*meant\*** something like:

"keep increasing brightness over time"

but your code does:

"increase once, then stop".

```
=====
2. `if (cycle == 1.0)` WILL NEVER BE TRUE
=====
```

This **is** a classic floating-point problem.

You **are** doing:

```
cycle += 0.1;
```

Binary floating-point numbers cannot represent 0.1 exactly.

So instead of reaching exactly 1.0, you **get values like**:

0.999999999999

or

1.0000000000002

So this **condition**:

```
if (cycle == 1.0)
```

almost never becomes true.

That means the dimming loop never runs.

```
=====
3. YOUR DIMMING LOOP USES THE WRONG CONDITION
=====
```

Inside:

```
while (redLED.getDutyCycle() > 0)
```

```
you never update the LED's duty cycle **before checking again** in a  
meaningful way.
```

```
You also rely on `getDutyCycle()` instead of the variable you are actually  
controlling.
```

```
You should always base logic on your own variable, not hardware feedback,  
unless explicitly required.
```

```
=====  
4. NO BOUNDS CHECKING  
=====
```

```
You allow `cycle` to go:
```

- above 1.0
- below 0.0

```
Phidgets duty cycle must stay between:  
0.0 and 1.0
```

```
Going outside that range can cause undefined behavior or no visible  
change.
```

```
=====  
WHAT YOU WERE TRYING TO DO  
=====
```

```
You wanted:
```

- Gradually brighten the LED
- Then gradually dim it

```
That requires:
```

- A loop
- Direction control
- Proper bounds

```
=====  
MINIMAL FIX (STILL SIMPLE, STILL THREAD.SLEEP)  
=====
```

```
This version does exactly what you want and uses ONLY what you've learned.
```



```

double cycle = 0.0;
boolean increasing = true;

while (true)
{
    redLED.setDutyCycle(cycle);

    if (increasing)
    {
        cycle += 0.05;
        if (cycle >= 1.0)
        {
            cycle = 1.0;
            increasing = false;
        }
    }
    else
    {
        cycle -= 0.05;
        if (cycle <= 0.0)
        {
            cycle = 0.0;
            increasing = true;
        }
    }
}

Thread.sleep(50);
}

```

```
=====
WHY THIS WORKS
=====
```

- Continuous loop
  - No floating-point equality checks
  - Duty cycle stays in valid range
  - Smooth brighten and dim
  - Uses only `while`, `if`, variables, and `Thread.sleep`
- ```
=====
```

```
IMPORTANT LESSON (THIS IS BIG)
```

```
Never compare doubles using `==`.
```

```
Never expect code to "keep running" without a loop.
```

```
Never use hardware state as your control variable when you already have one.
```

```
=====
```

## SUMMARY

```
=====
```

```
Your code doesn't work because:
```

- It only runs once
- `cycle == 1.0` is never true
- There is no real brightness loop
- Floating-point math betrayed you

```
You were *very close* conceptually – this is exactly the kind of bug teachers expect students to hit at this stage.
```

```
If you want, I can rewrite this in the **smallest possible change** to match your original structure exactly.
```

```
You said:
```

```
Why won't this work?
```

```
//Add Phidgets Library
```

```
import com.phidget22.*;
```

```
public class LEDBrightness {  
    //Handle Exceptions  
    public static void main(String[] args) throws Exception{
```

```
        //Create
```

```
        DigitalOutput redLED = new DigitalOutput();
```

```
        //Address
```

```
        redLED.setHubPort(1);
```

```
        redLED.setIsHubPortDevice(true);
```

```
        //Open
```

```
redLED.open(1000);
```

//Use your Phidgets with Duty Cycle | Duty Cycle controls the power to your Phidget (Digital Output object). Duty Cycle values range from 0.0 - 1.0.

```
//Create cycle variable  
double cycle = 0.1;
```

```
//Loop for increase and decrease in brightness
```

```
while (redLED.getDutyCycle() > 0)
```

```
{
```

```
    redLED.setDutyCycle(cycle);
```

```
    cycle += 0.1;
```

```
    Thread.sleep(50);
```

```
    if (cycle >= 1.0)
```

```
{
```

```
        while (redLED.getDutyCycle() > 0)
```

```
{
```

```
            redLED.setDutyCycle(cycle);
```

```
            cycle -= 0.1;
```

```
            Thread.sleep(50);
```

```
}
```

```
}
```

```
}
```

```
}
```

**ChatGPT said:**

There are THREE separate logic problems here. Any ONE of them is enough to make this "not work", but together they completely break the program.

I'll explain them in the exact order the program runs, so you can \*see\* why it fails.

=====

PROBLEM 1: Your OUTER while condition is false immediately

=====

Your outer loop is:

```
while (redLED.getDutyCycle() > 0)
```

But at this point `in` the program, you have NEVER `set` the duty cycle yet.

Phidgets DigitalOutput duty cycle defaults `to **0.0**.`

So the condition becomes:

```
while (0.0 > 0)
```

That `is false.`

**Result:**

The outer `while` loop NEVER runs.

Nothing inside it executes.

The LED never changes.

This alone completely stops the program.

---

Correct thinking:

---

You should NEVER base a `loop` condition `on` a value that hasn't been `set` yet.

---

=====

PROBLEM 2: You are `using` the HARDWARE value `as` your control variable

---

You are repeatedly checking:

```
redLED.getDutyCycle()
```

But the value you are actually changing `and` controlling `is:`

```
double cycle
```

Your logic should be based `on` `cycle`, `not on` reading the hardware back.

Hardware reads are:

- slower
- `not guaranteed to` update instantly
- unnecessary here

This causes unreliable logic `and` infinite loops.

```
=====
```

PROBLEM 3: NESTED `while` LOOP TRAPS EXECUTION

```
=====
```

When this condition becomes `true`:

```
if (cycle >= 1.0)
```

You enter this loop:

```
while (redLED.getDutyCycle() > 0)
```

That inner `while` loop has NO WAY to exit properly, because:

- The condition depends on hardware
- You decrease `cycle`, but you never test `cycle`
- Floating-point math may never hit exactly 0

So even if the program reached this point, it could get stuck forever.

```
=====
```

THE CORE CONCEPT YOU ARE MISSING

```
=====
```

You need:

- ONE loop
- ONE control variable
- ONE direction flag

Not nested `while` loops based on hardware state.

```
=====
```

MINIMAL FIX (USES ONLY WHAT YOU'VE LEARNED)

```
=====
```

This is the \*\*simplest correct version\*\*, very close to your intent:

```
double cycle = 0.0;
```

```
boolean increasing = true;
```

```

while (true)
{
    redLED.setDutyCycle(cycle);

    if (increasing)
    {
        cycle += 0.1;
        if (cycle >= 1.0)
        {
            cycle = 1.0;
            increasing = false;
        }
    }
    else
    {
        cycle -= 0.1;
        if (cycle <= 0.0)
        {
            cycle = 0.0;
            increasing = true;
        }
    }

    Thread.sleep(50);
}


```

=====

WHY THIS VERSION WORKS

=====

- Loop always runs
- Duty cycle stays between 0.0 and 1.0
- No floating-point equality checks
- No hardware-based loop conditions
- Smooth brighten and dim

=====

IMPORTANT RULES TO REMEMBER (WRITE THESE DOWN)

=====

- 1) Never use `getDutyCycle()` as a loop condition
- 2) Never compare doubles with `==`

- ```
3) Never expect a loop to run without guaranteeing its condition is true  
4) Use ONE variable to control behavior
```
- 
- 

SUMMARY

---

---

Your code fails because:

- The outer loop never starts
- Hardware state is used incorrectly
- Nested loops trap execution
- Floating-point math breaks equality logic

You are VERY close conceptually – you just need to stop thinking in terms of “hardware tells me what to do” and instead think “my variable controls the hardware”.

If you want, I can rewrite this using a \*\*for-loop version\*\* or explain how your teacher expects this assignment to be structured.

---

You said:

This program is printing that player 1 wins when the green wins and player 2 wins when the red wins, but i want it the other way around. Why is the program doing this?

```
//Add Phidgets Library  
import com.phidget22.*;
```

```
public class Btn_LedEvents_TOW {  
    //Turn on/off LEDs with Global Variables  
    static boolean turnRedLEDOn = false;  
    static boolean turnGreenLEDOn = false;  
  
    //Handle Exceptions
```

```
    public static void main(String[] args) throws Exception {
```

```
        //Variables  
        int timesBtnPressed = 0;  
        int player1Press = 0;  
        int player2Press = 0;  
        boolean currentGreenState = false;  
        boolean currentRedState = false;  
        boolean previousRedState = false;  
        boolean previousGreenState = false;
```

```
//Create
DigitalInput redButton = new DigitalInput();
DigitalInput greenButton = new DigitalInput();
DigitalOutput redLED = new DigitalOutput();
DigitalOutput greenLED = new DigitalOutput();
```

```
//Address
redButton.setHubPort(0);
redButton.setIsHubPortDevice(true);
greenButton.setHubPort(5);
greenButton.setIsHubPortDevice(true);
redLED.setHubPort(1);
redLED.setIsHubPortDevice(true);
greenLED.setHubPort(4);
greenLED.setIsHubPortDevice(true);
```

//Event | Event code runs when data input from the sensor changes. The following event is a state change event. The code will listen to the button (Digital Input Object) and only run the contain code when the button is pressed or released (state changes).

```
redButton.addStateChangeListener(new DigitalInputStateChangeListener() {
    public void onStateChange(DigitalInputStateChangeEvent e) {
        //Record button state to turn on/off the red LED
        turnGreenLEDOn = e.getState();
    }
});
```

//Event | Event code runs when data input from the sensor changes. The following event is a state change event. It will listen to the button (Digital Input Object) and only run the contain code when the button is pressed or released (state changes).

```
greenButton.addStateChangeListener(new DigitalInputStateChangeListener() {
    public void onStateChange(DigitalInputStateChangeEvent e) {
        //Record button state to turn on/off the green LED
        turnRedLEDOn = e.getState();
    }
});
```

```
//Open
redLED.open(1000);
greenLED.open(1000);
redButton.open(1000);
```

```
greenButton.open(1000);

//Use your Phidgets | In the button events you recorded the Button State. Here we
will use that data to turn on/off the LEDs
while(player1Press < 10 && player2Press < 10) {
    //Set current states
    currentGreenState = turnGreenLEDOn;
    currentRedState = turnRedLEDOn;

    //Count player presses
    if (currentGreenState && !previousGreenState)
    {
        player2Press++;
    }

    if (currentRedState && !previousRedState)
    {
        player1Press++;
    }

    previousGreenState = currentGreenState;
    previousRedState = currentRedState;

    //sleep for 150 milliseconds
    Thread.sleep(50);
}

//Flash LEDs when game ends
redLED.setState(true);
greenLED.setState(true);
Thread.sleep(1500);
redLED.setState(false);
greenLED.setState(false);
Thread.sleep(500);

//Print winner
if (player1Press == 10)
{
    System.out.println("Player 1 wins!");
}
```

```
        else if (player2Press == 10)
    {
        System.out.println("Player 2 wins!");
    }
}
```

**ChatGPT said:**

This is happening because you have \*\*accidentally swapped the buttons in your event handlers\*\*, so:

- the RED button is updating the GREEN state
- the GREEN button is updating the RED state

Your counting logic is actually correct – it's the \*\*event wiring logic\*\* that is flipped.

Let's walk through it carefully.

```
=====
THE CORE PROBLEM (THIS IS THE BUG)
=====
```

Look at your event listeners.

```
RED BUTTON EVENT:
-----
redButton.addStateChangeListener(...)
{
    turnGreenLEDon = e.getState();
}
```

```
GREEN BUTTON EVENT:
-----
greenButton.addStateChangeListener(...)
{
    turnRedLEDon = e.getState();
}
```

This means:

- Pressing the RED button sets \*\*turnGreenLEDon\*\*
- Pressing the GREEN button sets \*\*turnRedLEDon\*\*

So from the program's point of view:

```
RED button → GREEN state  
GREEN button → RED state
```

That is why the winner appears reversed.

```
=====  
WHY THE WINNER PRINTS BACKWARDS  
=====
```

Later, in your loop, you do:

```
currentGreenState = turnGreenLEDOn;  
currentRedState = turnRedLEDOn;
```

Then you count:

```
if (currentGreenState && !previousGreenState)  
{  
    player2Press++;  
}  
  
if (currentRedState && !previousRedState)  
{  
    player1Press++;  
}
```

Because the states are swapped:

- RED button increments player 2
- GREEN button increments player 1

So when GREEN physically wins, the program thinks PLAYER 1 won.  
When RED physically wins, the program thinks PLAYER 2 won.

The program is doing exactly what you told it to do – just not what you meant.

```
=====  
THE FIX (ONE-LINE LOGIC FIX)  
=====
```

Each button must update its OWN color state.

Change your event handlers to this:

```
RED BUTTON EVENT:  
-----
```

```
redButton.addStateChangeListener(new DigitalInputStateChangeListener() {
    public void onStateChange(DigitalInputStateChangeEvent e) {
        turnRedLEDon = e.getState();
    }
});
```

#### GREEN BUTTON EVENT:

```
-----  
greenButton.addStateChangeListener(new DigitalInputStateChangeListener() {
    public void onStateChange(DigitalInputStateChangeEvent e) {
        turnGreenLEDon = e.getState();
    }
});
```

Nothing else in your program needs to change.

```
=====  
WHY THIS FIX WORKS  
=====
```

Now the data flow is correct:

```
RED button → red state → player 1  
GREEN button → green state → player 2
```

```
Your edge detection logic is already solid.  
Your loop logic is already solid.  
Your win condition logic is already solid.
```

Only the **\*\*wiring between input and variable\*\*** was crossed.

```
=====  
IMPORTANT CONCEPT (VERY IMPORTANT)  
=====
```

When using event-driven code:

```
If the wrong variable is updated,  
EVERYTHING downstream breaks logically  
even if the math and conditions are perfect.
```

This is a classic real-world bug and not a beginner mistake.

```
=====  
SUMMARY  
=====
```

