Loops
Chapter 5

Course: CPSC 1150

Instructor: Dr. Bita Shadgar

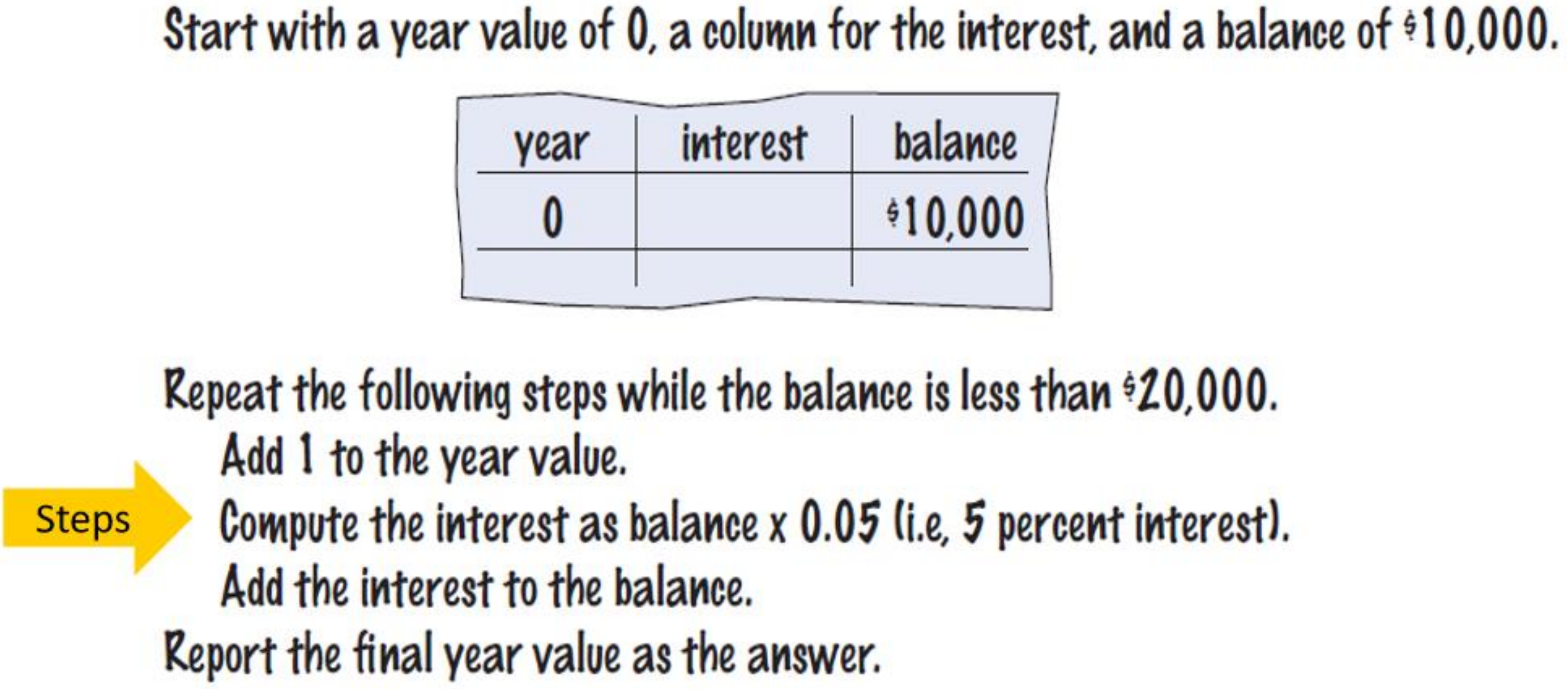
Lecture 10

Learning Outcomes

- Design algorithms with loops
- Recognize different types of loops; count-controlled and event-controlled loops.
- Problem solving using different java loops (for, while and do-while) and nested loops
- Choose the best loop
- Avoid pitfalls while working with loops
- Differentiate loops and decision structures
- Apply loops for validating data
- Apply sentinel value and loops to give user the control of execution

Problem statement

– How many years is needed for the balance to be doubled, if the interest rate is 5 percent?



Start with a year value of 0, a column for the interest, and a balance of $10,000.

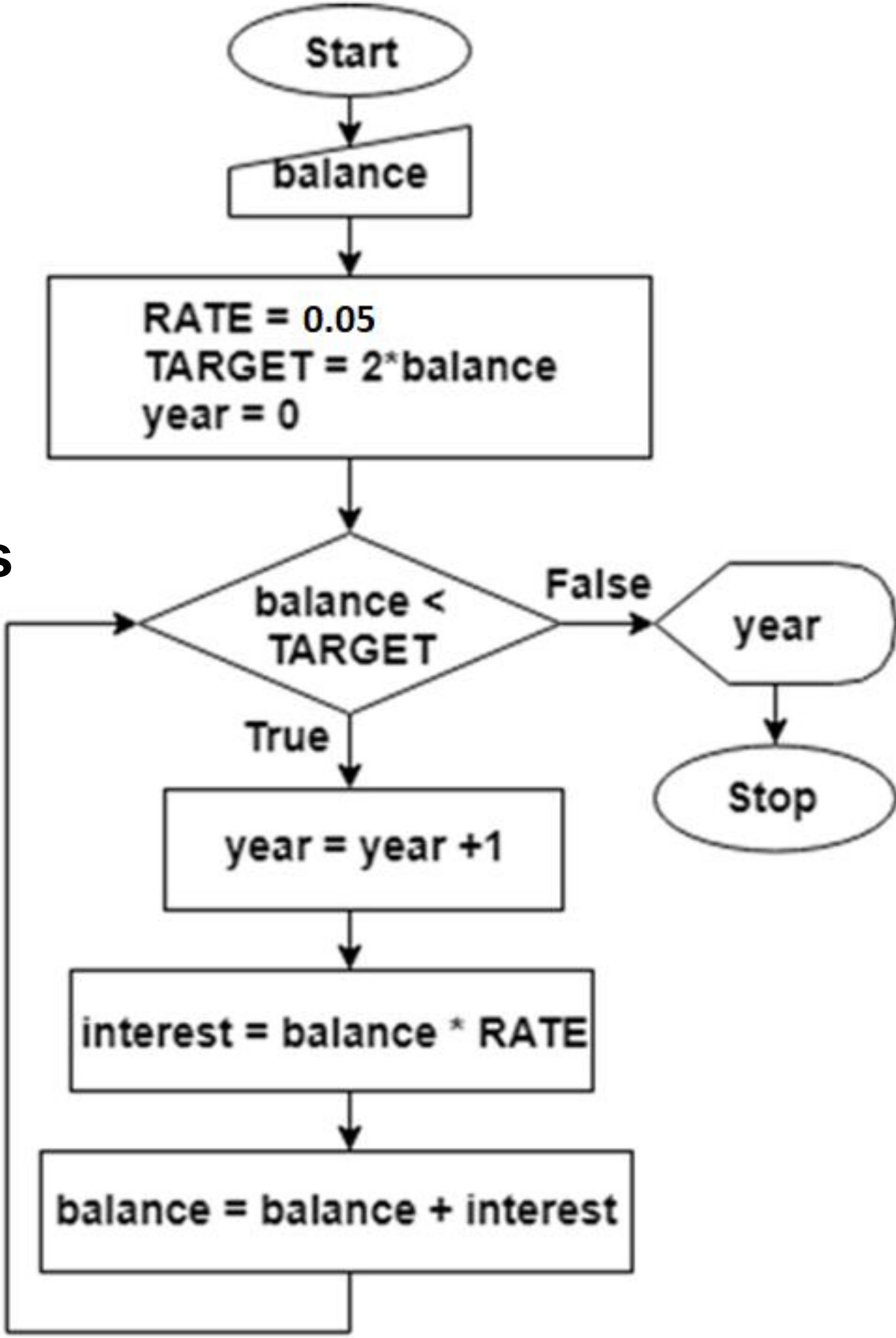
year	interest	balance
0		$10,000

Steps

```
Repeat the following steps while the balance is less than $20,000.
    Add 1 to the year value.
    Compute the interest as balance x 0.05 (i.e, 5 percent interest).
    Add the interest to the balance.
Report the final year value as the answer.
```

Plan the Solution

A loop executes instructions repeatedly while a condition is True.



Introducing while Loops

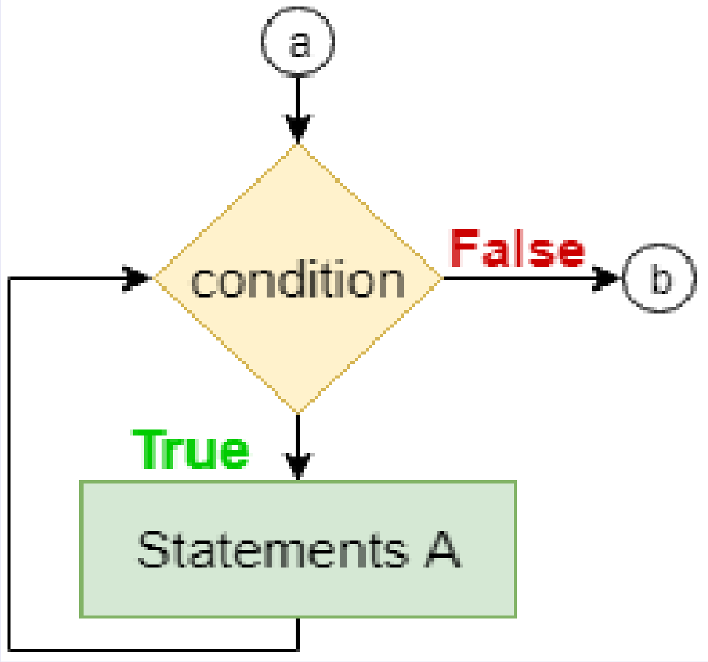
Java syntax

```
while ( condition ) {
      Statements A
}
```

Pseudocode

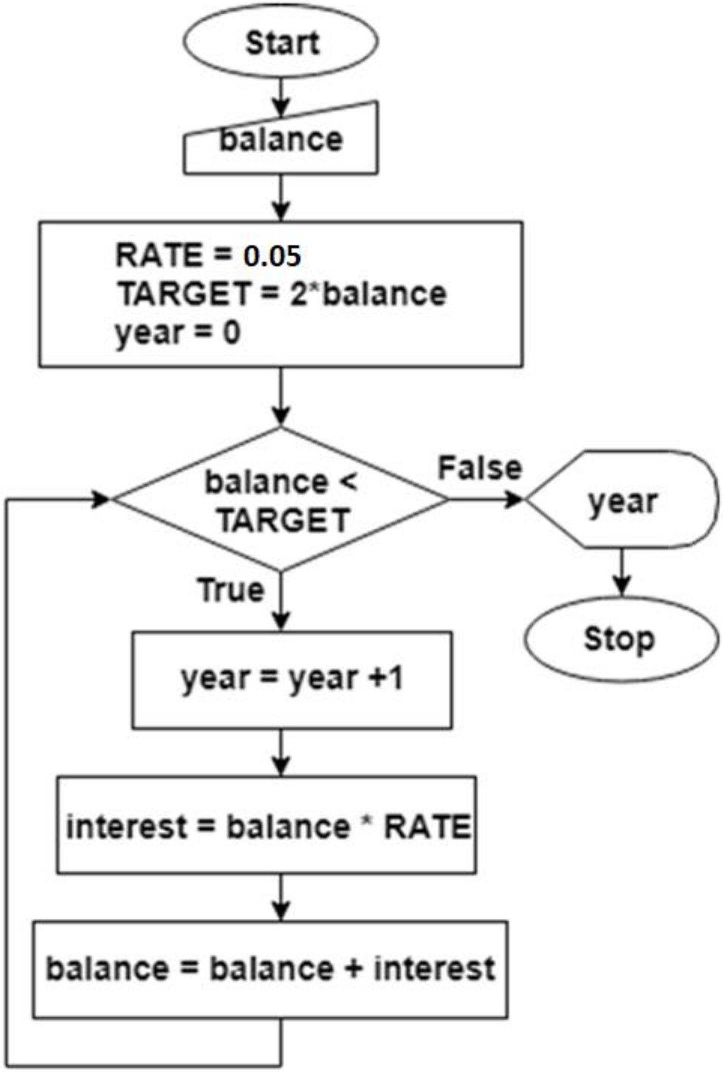
```
Repeat while condition is true
          Statements A
```

Flowchart



- A while loop repeats the statements inside the block
- Before entering the block, checks whether the condition is true
  - Only enters the block, if the condition evaluates to true
  - The first time that the condition is false, code execution continues after closing curly brace of the loop

Demo – BankBalance.java



Boolean variable as loop condition

```
boolean myBool = true;
while ( myBool ) {
  //statements
  /*somewhere in here, myBool must eventually
  become false (updating loop control variable)*/
}
```

– Trace your loop to make sure myBool eventually becomes false

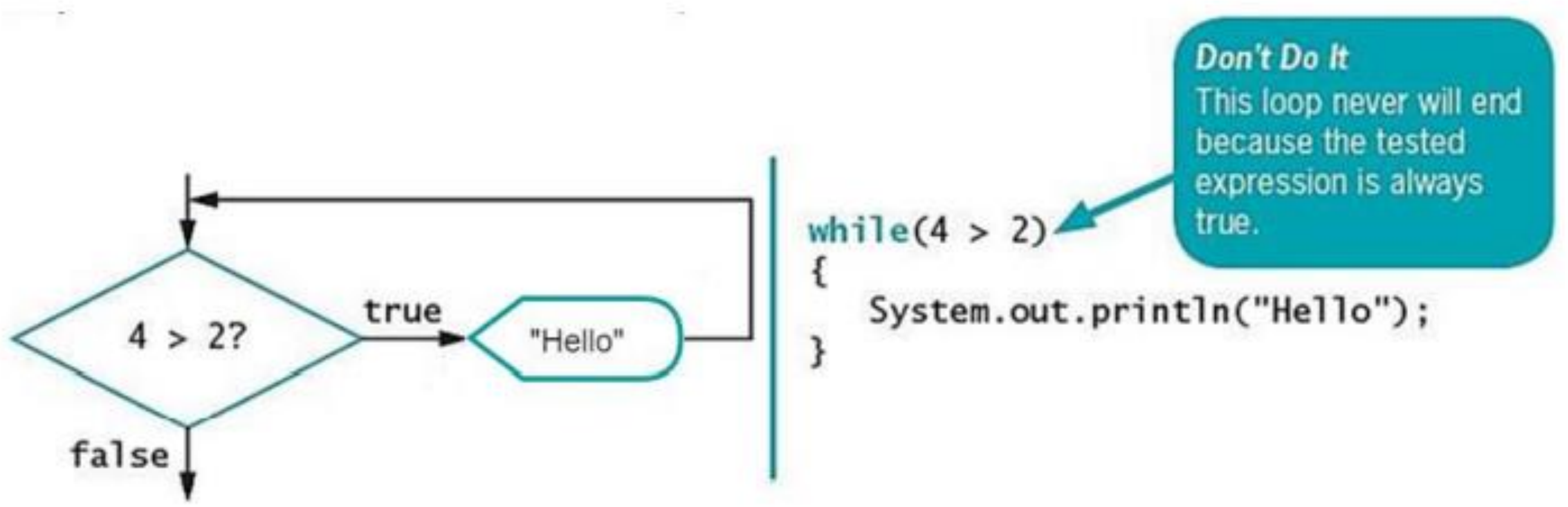
– Otherwise, you will have an **infinite loop**

  ◦ Your program will **never** terminate!

– Condition can be a complex condition

Infinite Loop

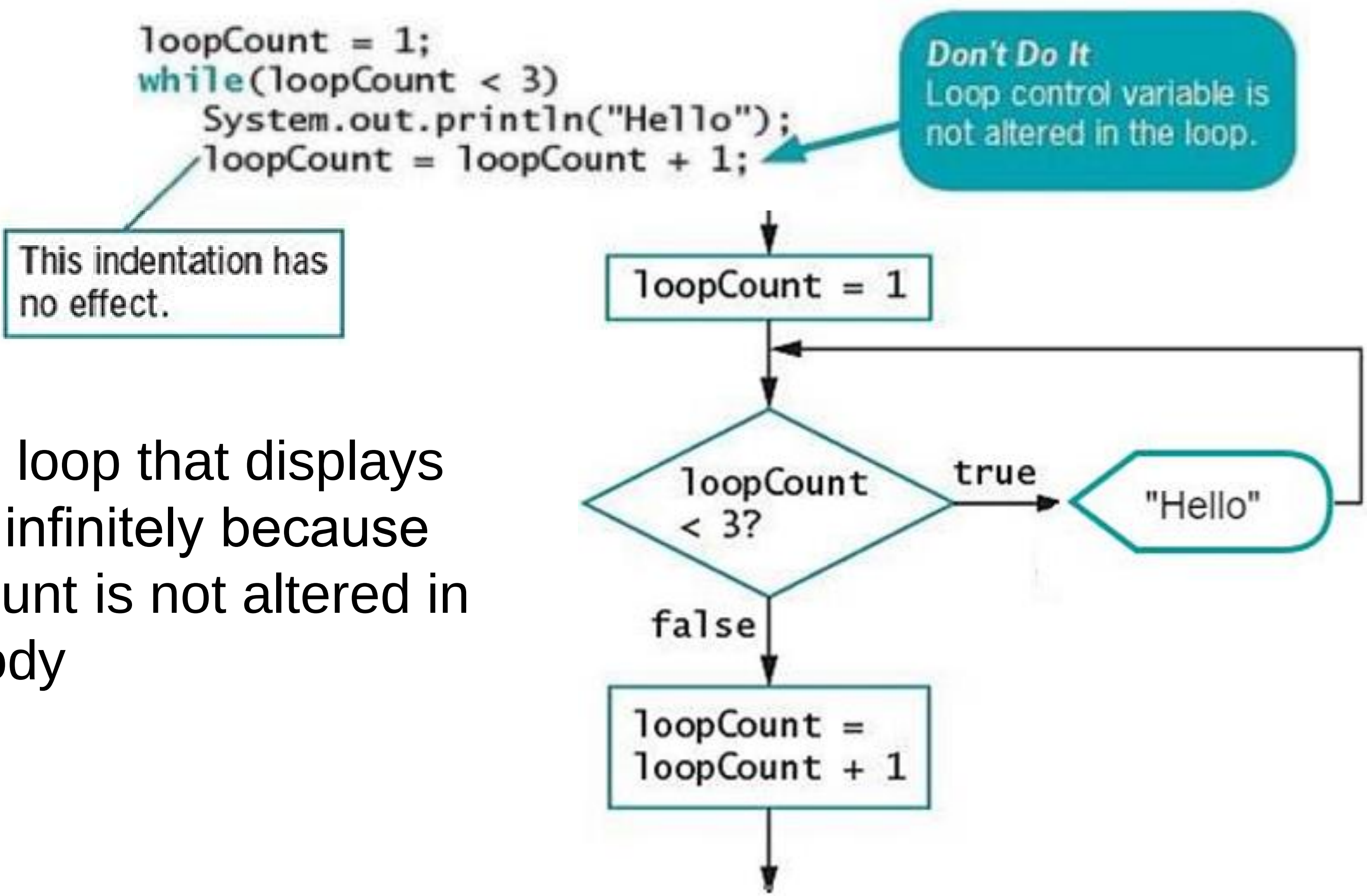
- A loop that never ends
- Can result from a mistake in the `while` loop
- Do not write intentionally



```
while(4 > 2)
{
   System.out.println("Hello");
}
```

Don't Do It
This loop never will end because the tested expression is always true.

Pitfall: Failing to alter the loop control variable within the loop body



```
loopCount = 1;
while(loopCount < 3)
   System.out.println("Hello");
   loopCount = loopCount + 1;
```

Don't Do It
Loop control variable is not altered in the loop.

This indentation has no effect.

loopCount = 1

loopCount < 3?

true

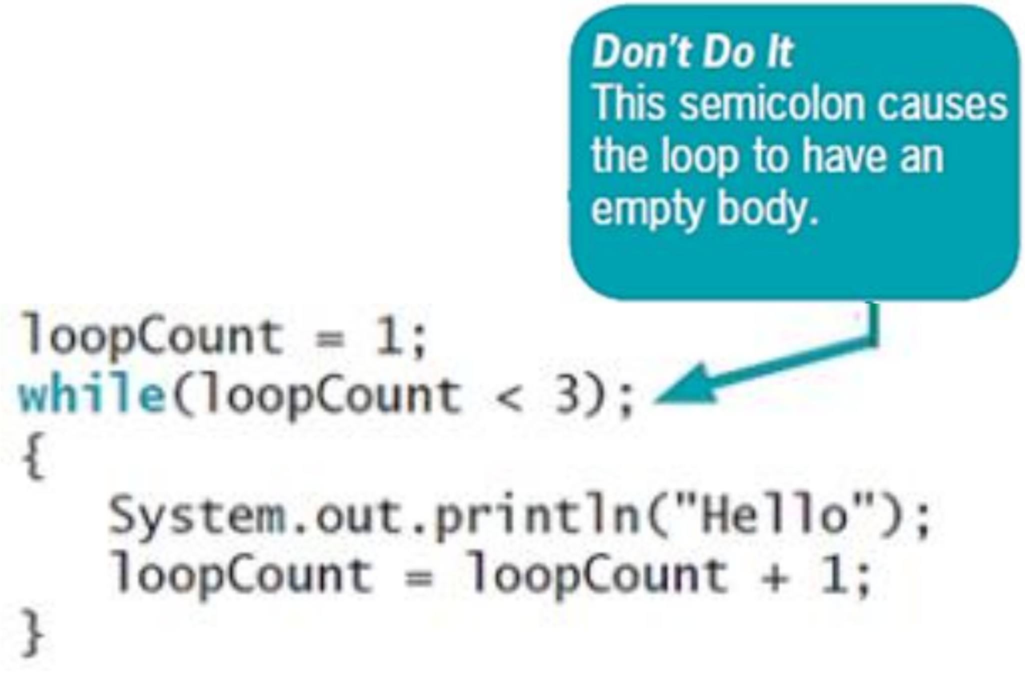
"Hello"

false

loopCount = loopCount + 1

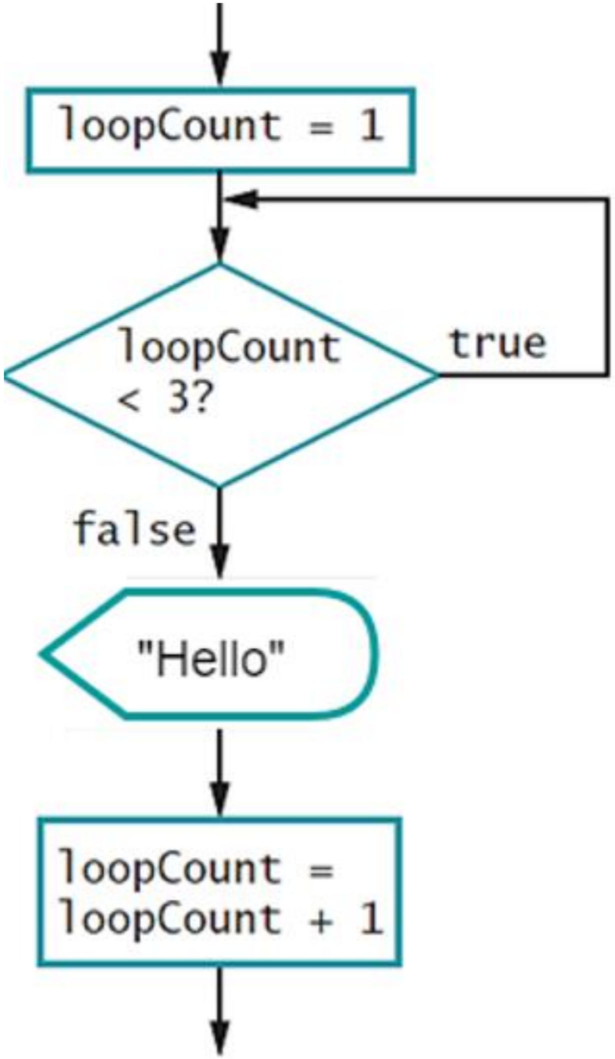
– A while loop that displays “Hello” infinitely because loopCount is not altered in loop body

Pitfall: a Loop with an Empty Body



Don't Do It
This semicolon causes the loop to have an empty body.

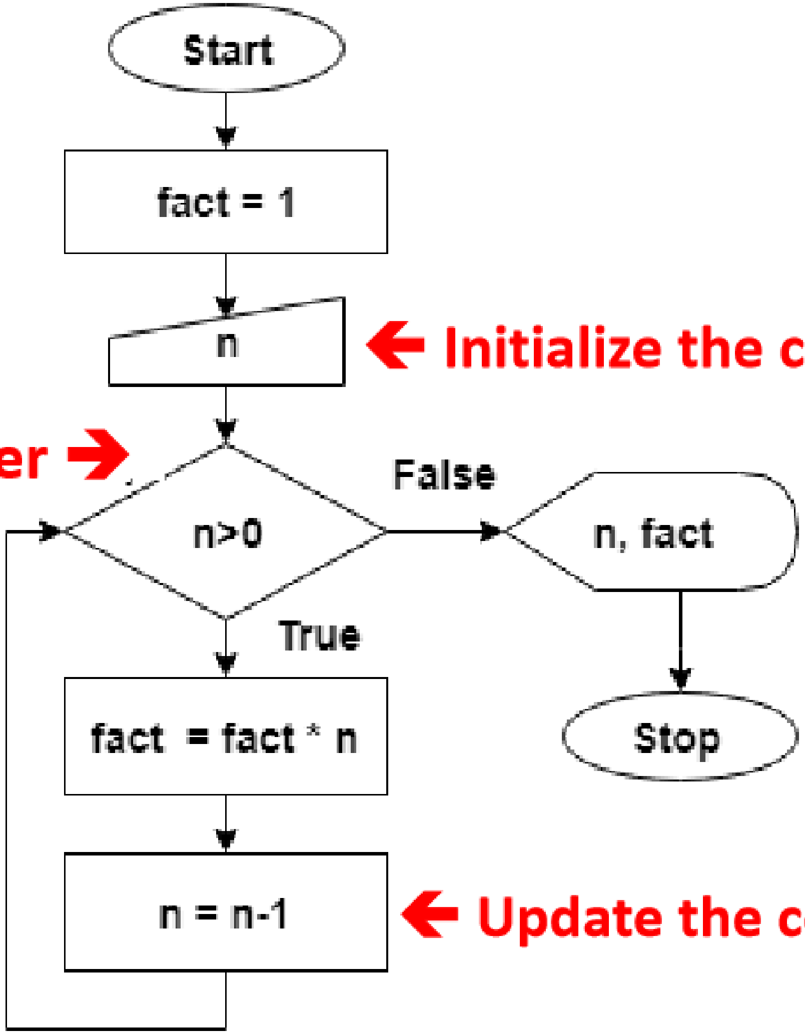
```
loopCount = 1;
while(loopCount < 3);
{
    System.out.println("Hello");
    loopCount = loopCount + 1;
}
```



– A while loop that loops infinitely with no output, because the loop body is empty

Count-controlled loop

- What is the main purpose of the sample code?
- What outputs are shown following the execution?



Check the counter →

← Initialize the counter

← Update the counter

Using a count-controlled loop

- Sometimes you know how many times your loop needs to run (N times)
  - You can use a loop counter variable
  - OK to use a one-letter name for this, especially i, j, k, n

```
int i = 0; // initializing the loop counter
while ( i < N ) {  // loop condition
   //statements go here
   i++; // update counter
}
```

- Don't need to worry about an infinite loop if you're always adding to a counter

for loop : Count-controlled Loops

Java syntax

```
for (initializing counter; loop condition; updating counter ) {
    //statements go here
    //called the loop body
}
```

Pseudocode

```
Repeat  n  times
        Statements A
```

- A for loop repeats the statements inside the block (like a while loop)
- Unlike a while loop, the control statements of a for loop are all in the loop header, which has three parts:
    - Initializing counter
    - Loop condition
    - Updating counter
- Note the two semicolons between the parts of the head

Count-Controlled Problem

– Suppose that you need to print a string (e.g., "Welcome to Java!") a hundred times.

100 times

```
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
…
…
…
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
System.out.println("Welcome to Java!");
```

Trace for Loop

Declare i

```
int i;
for (i = 0; i < 2; i++)
{
  System.out.println("Welcome to Java!");

}
```

Trace for Loop, cont.

Execute initializer
i is now 0

```
int i;
for (i = 0; i < 2; i++)
{
 System.out.println("Welcome to Java!");
}
```

animation

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++)
{
 System.out.println("Welcome to Java!");
}
```

(i < 2) is true
since i is 0

animation

Trace for Loop, cont.

Print Welcome to C++!

```
int i;
for (i = 0; i < 2; i++)
{
System.out.println("Welcome to Java!");
}
```

animation

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++)
{
 System.out.println("Welcome to Java!");
}
```

Execute adjustment statement
i now is 1

animation

Trace for Loop, cont.

(i < 2) is still true
since i is 1

```
int i;
for (i = 0; i < 2; i++)
{
 System.out.println("Welcome to Java!");
}
```

animation

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++)
{
 System.out.println("Welcome to Java!");
}
```

Print Welcome to C++

animation

Trace for Loop, cont.

```
int i;
for (i = 0; i < 2; i++)
{
 System.out.println("Welcome to Java!");
}
```

Execute adjustment statement
i now is 2

Trace for Loop, cont.

(i < 2) is false
since i is 2

```
int i;
for (i = 0; i < 2; i++)
{
 System.out.println("Welcome to Java!");
}
```

Trace for Loop, cont.

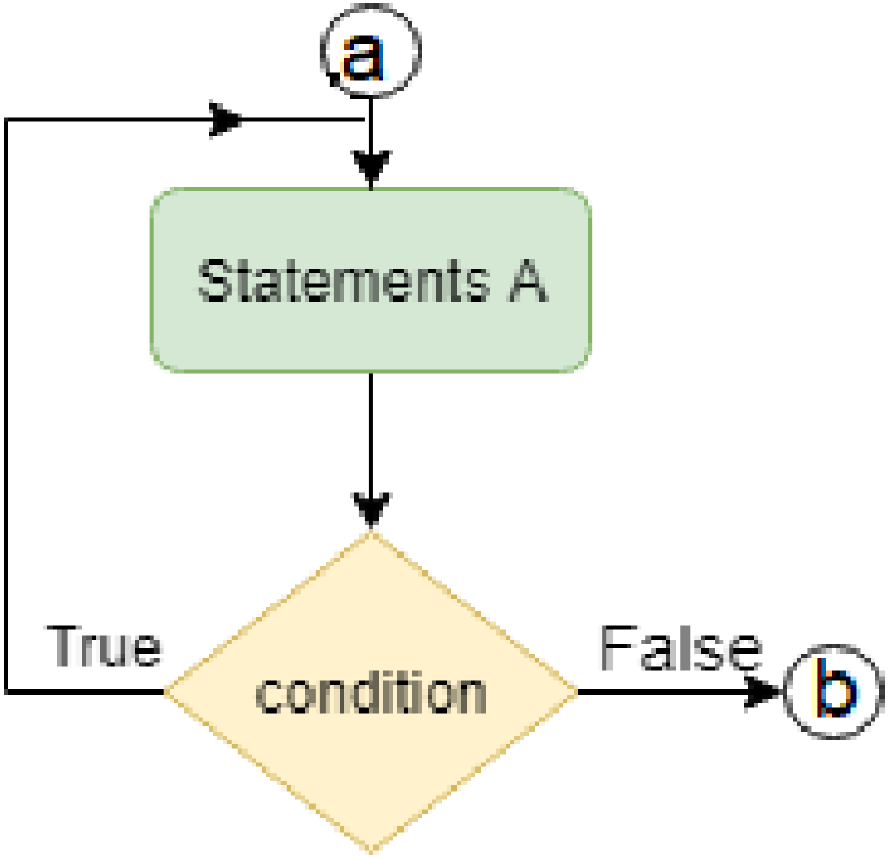
Exit the loop. Execute the next statement after the loop

```
int i;
for (i = 0; i < 2; i++)
{
 System.out.println("Welcome to Java!");
}
```

do-while loop

Java syntax

```
do {
    //statements A go here
} while ( condition );
```

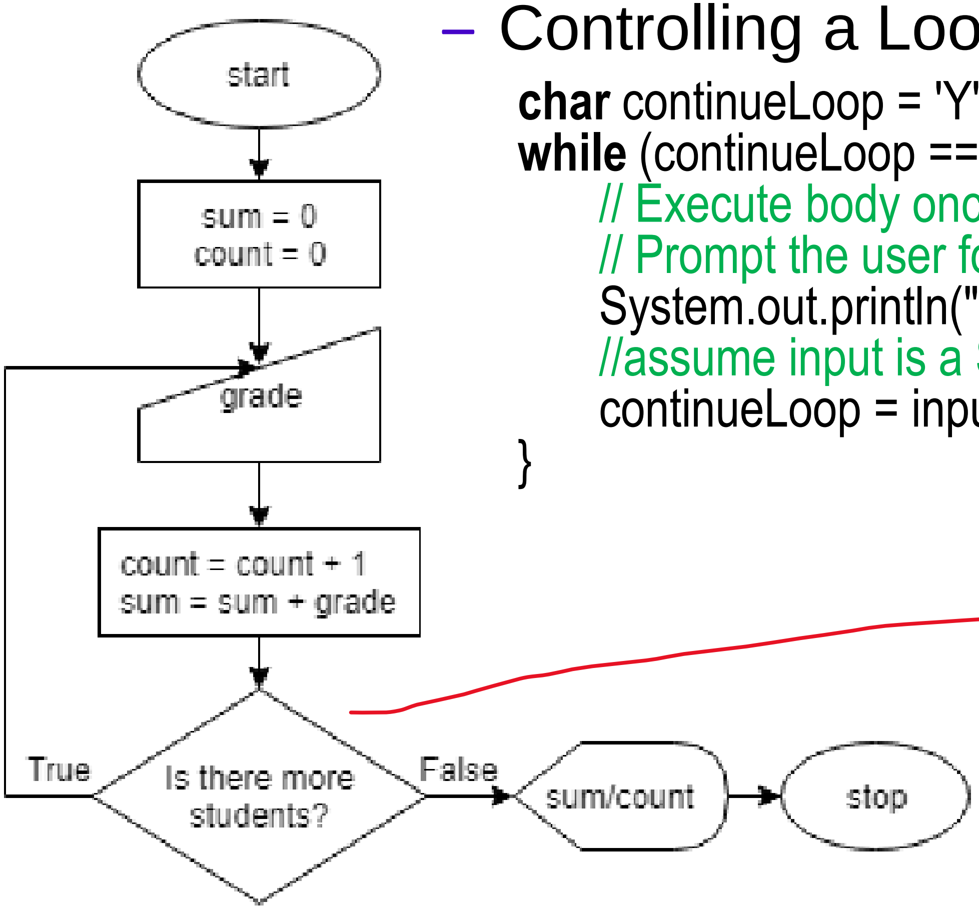


- do-while loop checks the condition after the loop A while loop checks before
- The do-while loop always executes at least once
- Can convert between types of loop

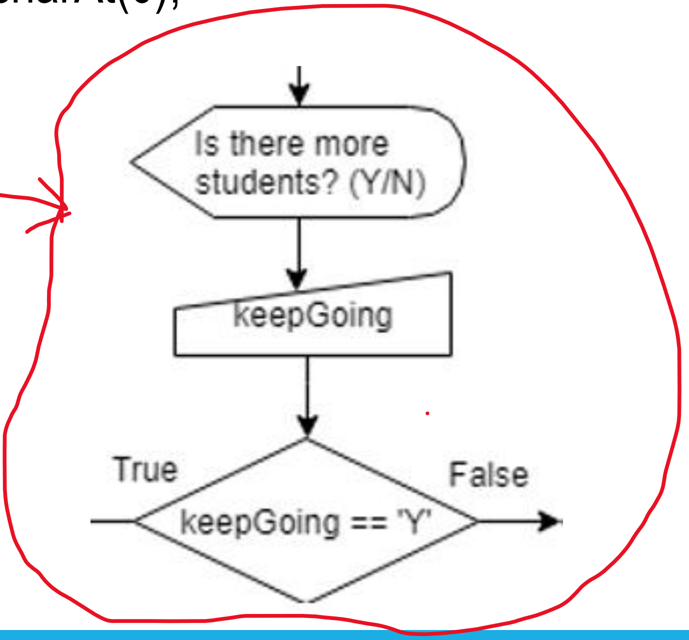
Practice – getting average of students grade

– Controlling a Loop with User Confirmation

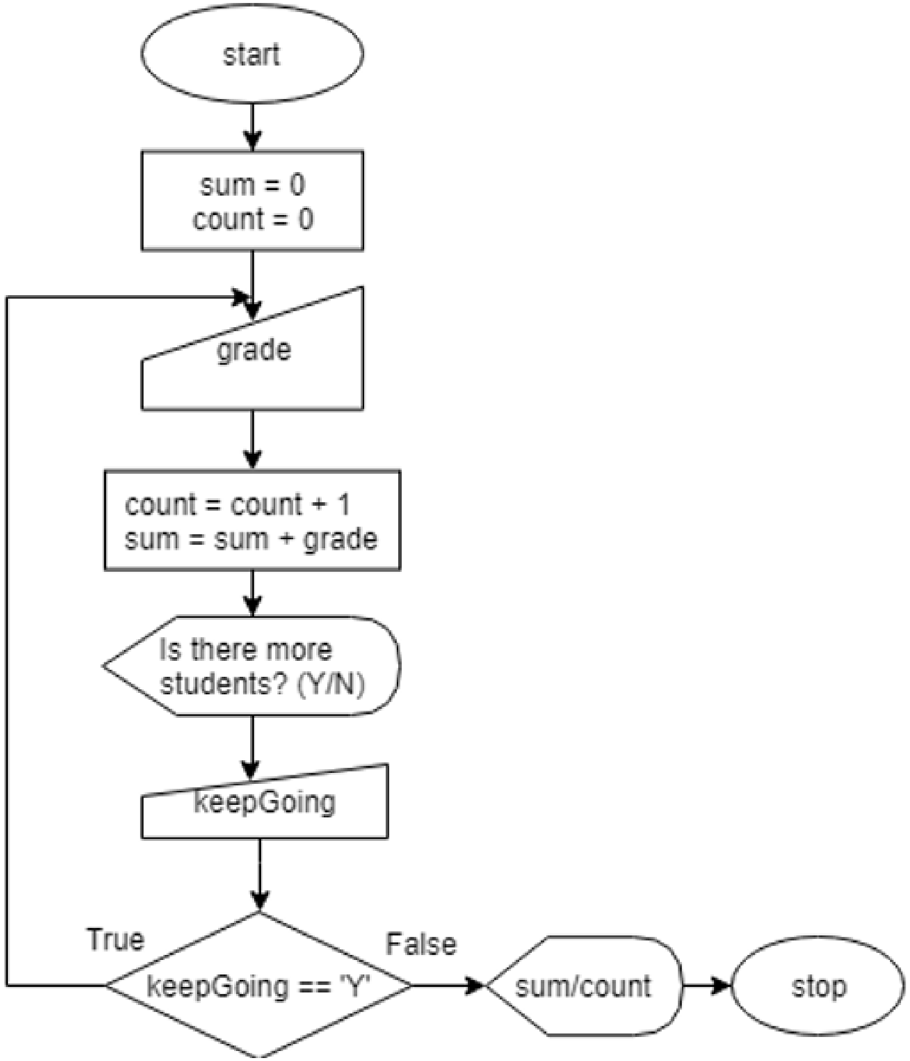
```
char continueLoop = 'Y';
while (continueLoop == 'Y') {
      // Execute body once
      // Prompt the user for confirmation
      System.out.println("Enter Y to continue or N to quit: ");
      //assume input is a Scanner object
      continueLoop = input.next().charAt(0);
}
```



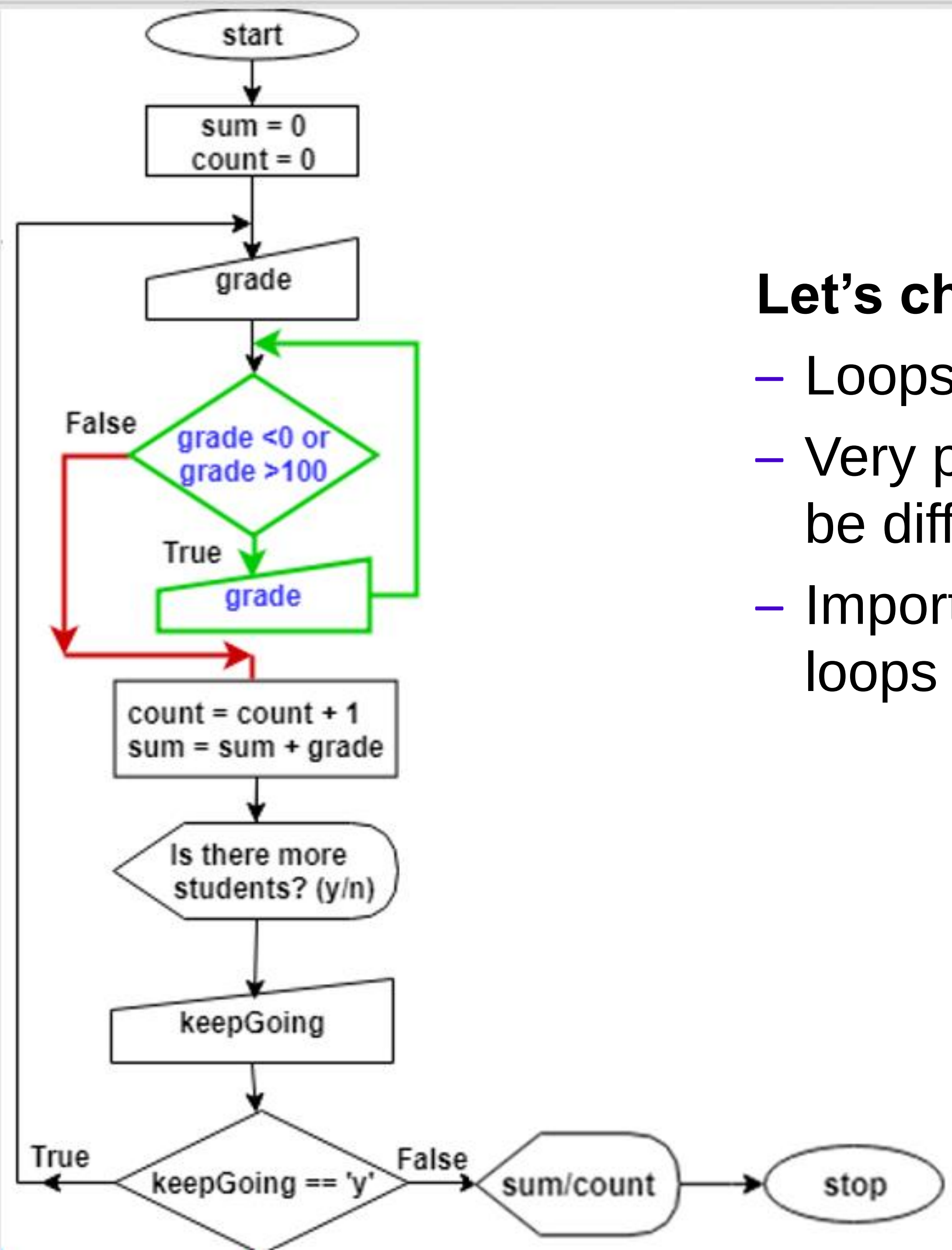
The condition is abstract and ambiguous.



Demo - AverageGrade.Java



Nested Loops



Let's check the validity of grades

– Loops are often inside other loops

– Very powerful tool, but also can be difficult to trace/use

– Important to practice using nested loops a lot

Which loop to pick?

- No wrong choice : it's largely a matter of personal preference
- Sometimes one feels more natural for the particular application
- If you can't decide:
  - If something must repeat a known number of times, or for a given list of values, a for loop is a good choice
  - If code must be executed at least once, no matter the condition, you may want to use a do-while loop
  - Otherwise, a while loop is generally a good choice

while versus do-while

Example

```
 int i = 1;
 while ( i <= N ) {
         System.out.println(i++);
 }
```

```
i = 1;
do {
         System.out.println(i++);
} while ( i <= N );
```

- – Question: Will the two above loops always have the same output?
- – Question: How can we ensure the same behavior, no matter the value of N?

For loop with more than one counter

- The **initial-action** in a **for** loop can be a list of zero or more comma-separated expressions.
- The **action-after-each-iteration** in a **for** loop can be a list of zero or more comma-separated statements.

```
for (int i = 0, j = 9; i + j < 10 ; i++, j-- ) {
              // Do something
}
```

For loops

- If the **condition** in a **for** loop is omitted, it is implicitly true.
- Thus the statement given below in (a), which is an infinite loop, is correct.
- Nevertheless, it is better to use the equivalent loop in (b) to avoid confusion:

```
for ( ; ; ) {
  // Do something
}
```

(a)

Equivalent

This is better

```
while (true) {
  // Do something
}
```

(b)

Using `break`

Usage of break in loops

When the **break** statement is encountered inside a loop, the loop is immediately terminated and program control resumes at the next statement following the loop.

```java
for (int i = 0; i < 10; i++){
    if (i == 4) {
        break;
    }
    System.out.println(i);
}
```

```
0
1
2
3
```

Using `continue`

Usage of continue in loops

The **continue** statement works somewhat like the break statement. Instead of forcing termination, however, **continue** forces the next iteration of the loop to take place, skipping any code in between.

```java
for (int i = 0; i < 10; i++){
    if (i == 4) {
        continue;
    }
    System.out.println(i);
}
```

```
0
1
2
3
5
6
7
8
9
```

Summary - we use loops because:

- Repeat the same steps many times
- Avoid copy-pasting
- All the code is in one place
- Execute based on a given logical condition
- Useful for:
    - String processing
    - Array processing
    - Multiple user inputs of the same type
    - Displaying patterns
    - Any repetitive process you can imagine

More Practice – Nested loops

- Write a program to display the following pattern, where the user selects the number of rows. Example if num rows is 5:

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
```

- Write a program to display a 2N*N upper right triangle of *s. Example if N = 5:

```
**********
  ********
    ******
      ****
        **
```