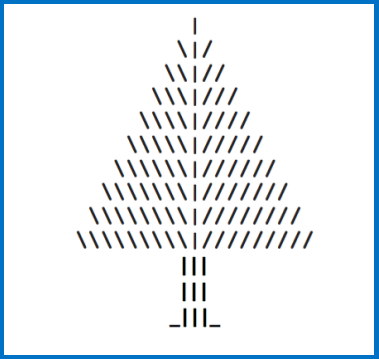
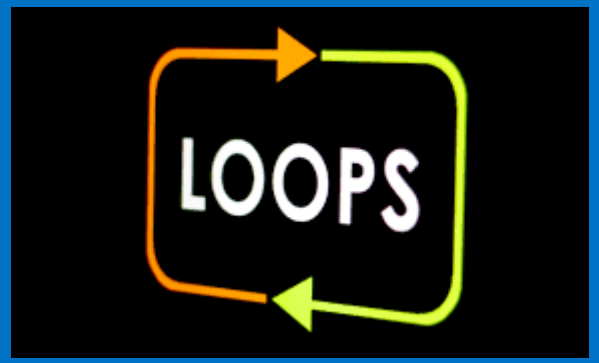
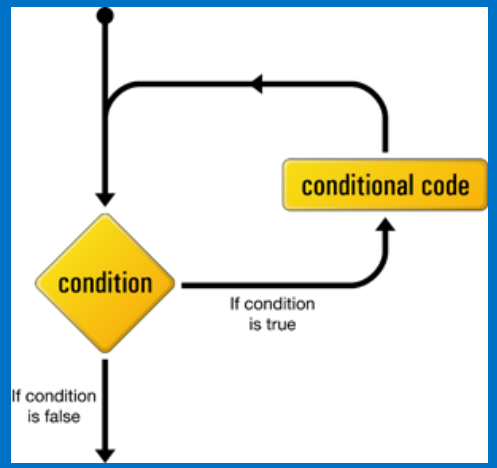




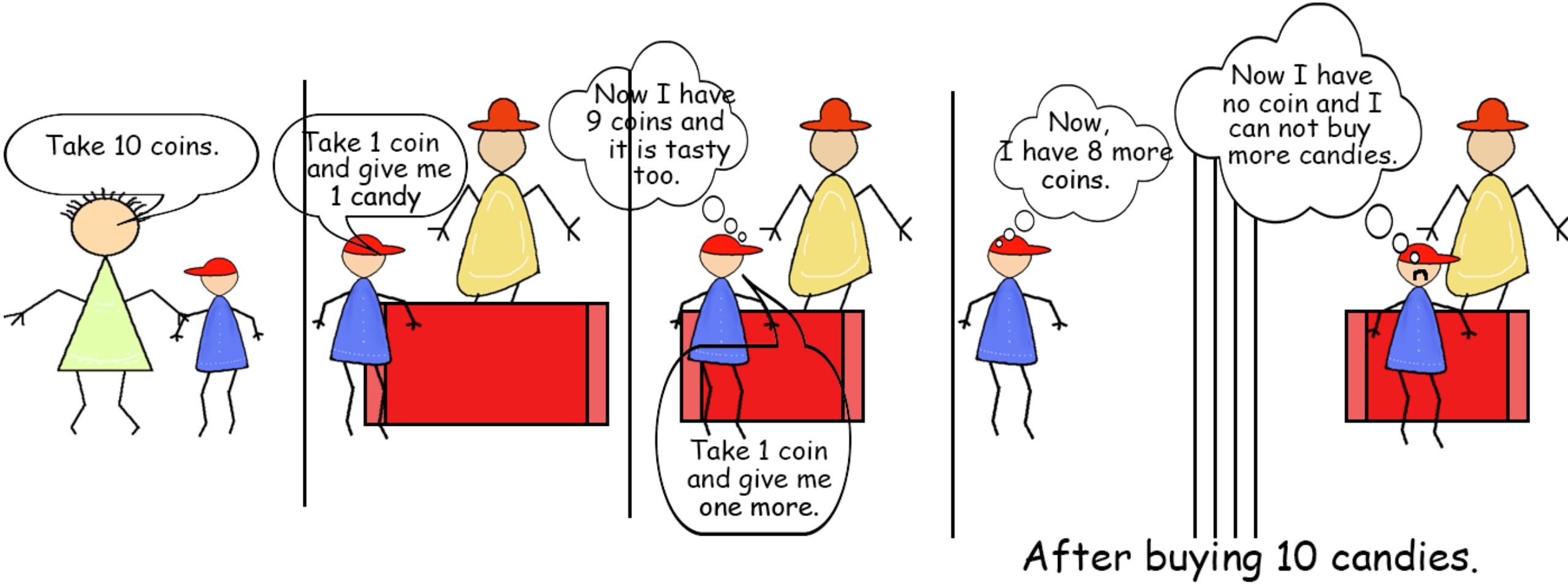


CSS 142

Lecture 7

Arkady Retik aretik@uw.edu

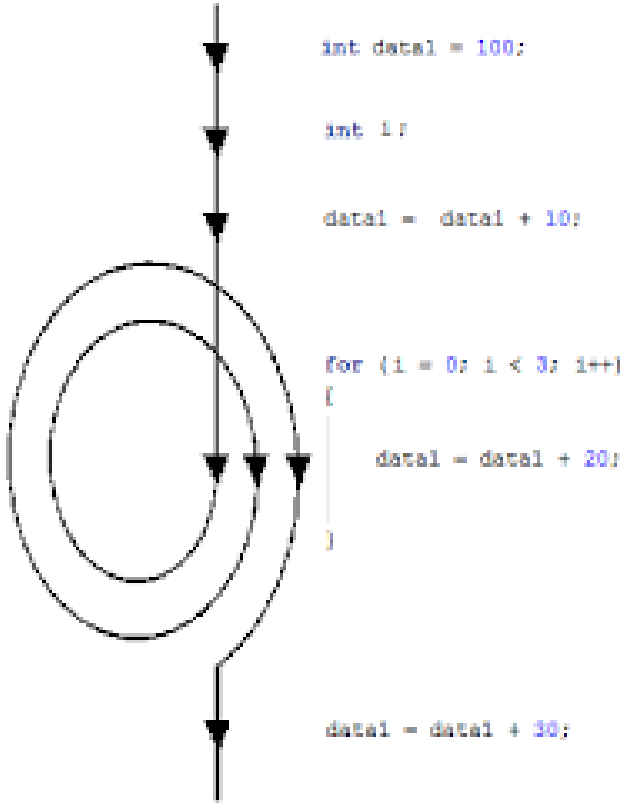
TODAY'S JOKE



Remember – always to watch two things in loops: loop counter and loop calculation results – coins and candy

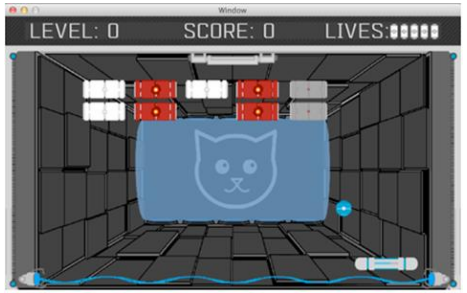
TODAY'S CONTENT

1. **Q&A: Midterm /HW**
2. **Loops**
3. **Using Loops**
4. **HoA3b – see end of the slide deck**



NEXT:

- **Wednesday – Midterm: Good Luck!!**
- **HW3**



No students hours tomorrow – Tue. 3-4
You can see me Mon 4-5 pm or Tue 8-8.30 pm

Homework 2

Average Score:	**29.11**
High Score:	**30**
Low Score:	**26**
Median:	**30**
Total Graded:	**42 submissions**

Common mistakes:
Incorrect output for H2P1. Logic error
Incorrect output for 10$/hr and 45hr = 475
Some had 450. I guess, they didn't read the instruction carefully.

Midterm 1: content

- 4 weeks content:
  - Savitch Chapters 1,2,3
  - Lectures 1-7
  - HW1,2 && HoA 1,2,3 && Labs
- Logistics
  - Wedn , Apr 18th 11.00-1.00
  - Closed books; paper based
  - Combination of Q&As, writing code; reading code and writing output
  - Bring pencils and erasers

Key Topics

- Assignments
- Primitives and String
- Statements
- Methods; Printing
- IO, Scanner
- Branching: IFs, CASE
- Booleans
- Loops
  - while; do while; for

HoA 3a

1. **90% of the people skipped or didn't attempt to do this question.**
   5% people who attempted didn't get the problem correct. (wrong interpretation or incorrect)
   5% people did the problem right.

2. **50% attempted this problem.**
   25% got the problem right.
   25% had incorrect syntax or incorrect interpretation of the problem.

3. **80% attempted this problem.**
   50% got the problem correct.
   30% had some error regarding to switch statement.

errors like.

```
int number;
switch(number){
case "1": System.out.println("one");
//string incompatibility.
case(number): System.out.println("one");
//not sure how a number can also be a case()
}
```

4. **70% attempted this problem**
   40% got this problem correct.
   30% had some logic error and the return case should be a boolean. Also, different interpretation of the prompt.

5. **85% attempted this problem.**
   40% got this problem correct. both A and B should be printed.
   45% got only the output (incorrect) "Some kind of A" or "Some kind of B" not both.

FoR

Here are some of most common to least common question for the reading.

1. The for-statement and loops are confusing and hard to understand.

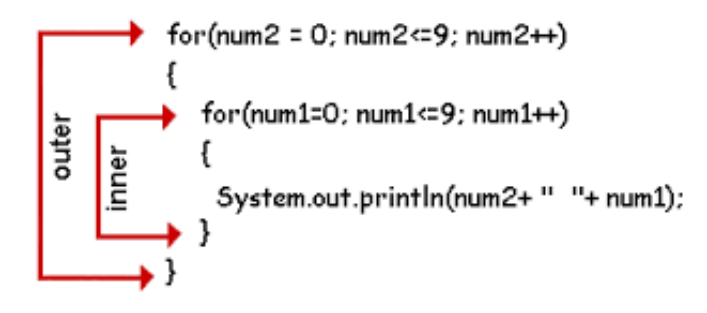
2. When do we use While-loops vs. Do-While loops?

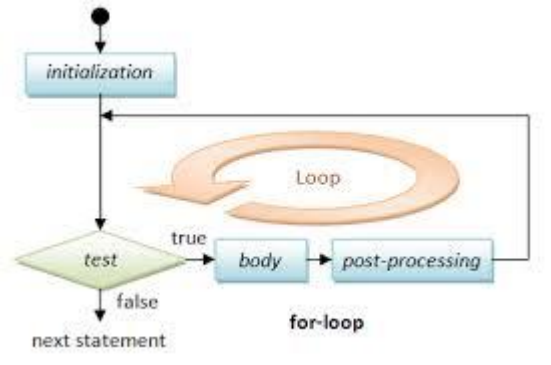
2. When do we use While-loops vs. For-loops?

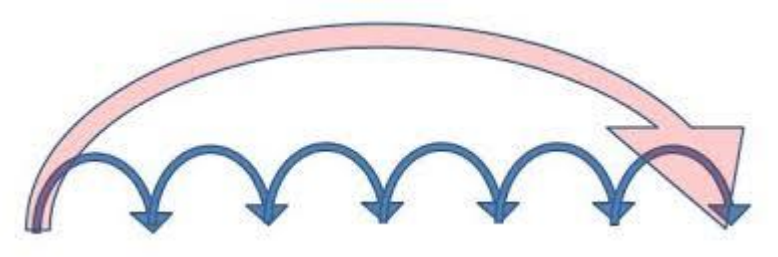
3. When or how should be implement break statement or continue statement.

3. Nested loops are confusing. How did the book get this output? etc.

LOOPS


for(num2 = 0; num2<=9; num2++)
{
for(num1=0; num1<=9; num1++)
{
System.out.println(num2+ " "+ num1);
}
}
outer
inner

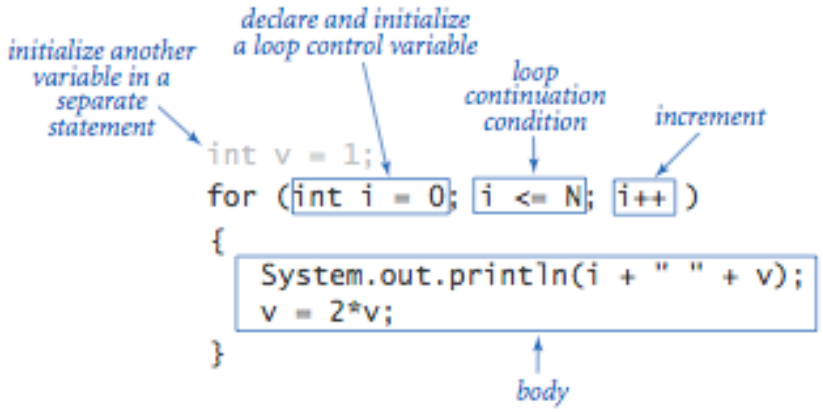

initialization
Loop
test
true
body
post-processing
false
next statement
for-loop



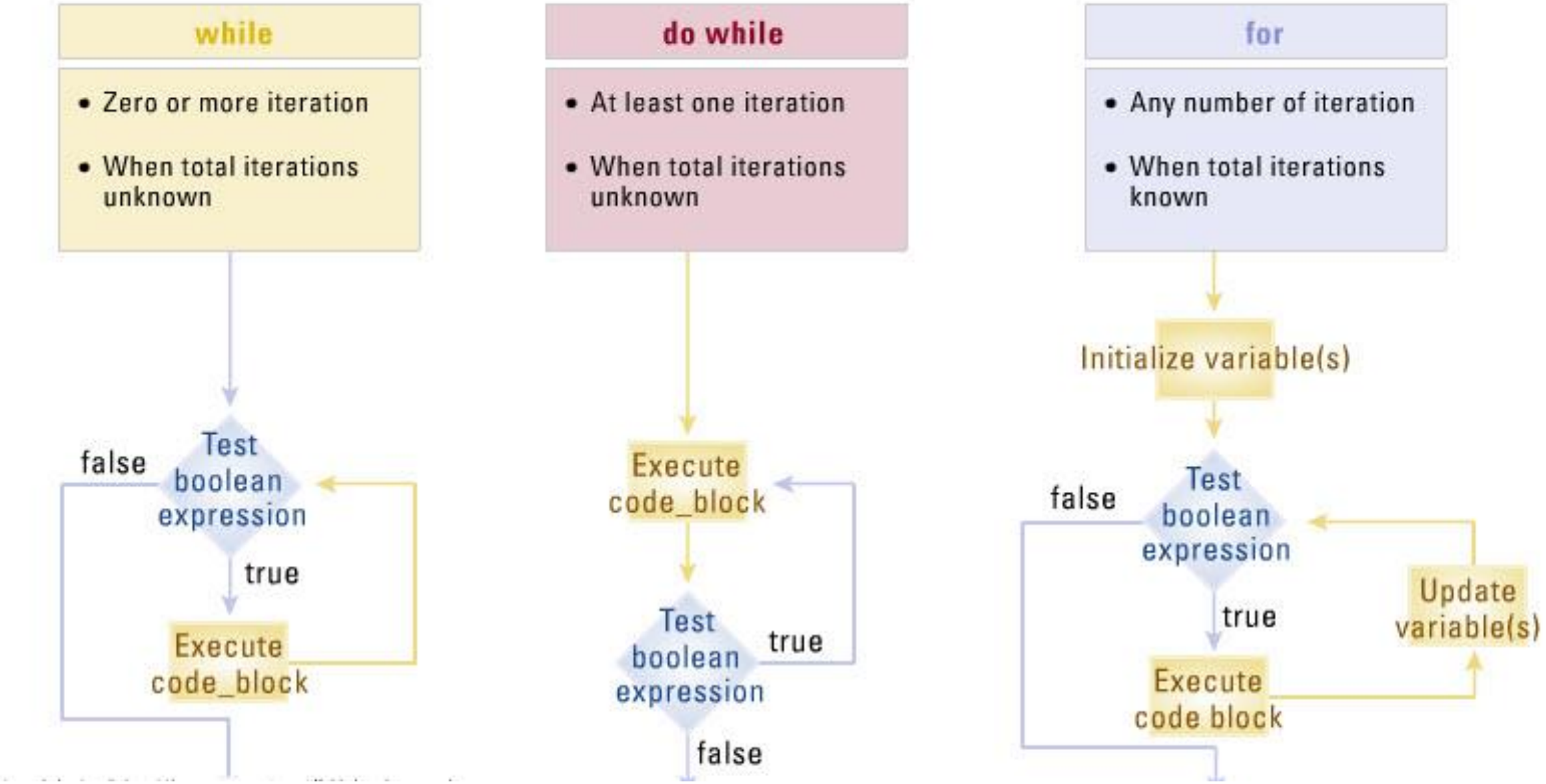
Loops

- *Loops* in Java are similar to those in other high-level languages
- Java has **three types of loop statements**:

  1. **`while`**,
  2. **`do-while`**,
  3. **`for`**



- The code that is repeated in a loop is called the *body* of the loop
- Each repetition of the loop body is called an *iteration* of the loop


while
• Zero or more iteration
• When total iterations unknown
Test boolean expression
false
true
Execute code_block
do while
• At least one iteration
• When total iterations unknown
Execute code_block
Test boolean expression
true
false
for
• Any number of iteration
• When total iterations known
Initialize variable(s)
Test boolean expression
false
true
Execute code block
Update variable(s)

print largest power of two less than or equal to N	`int v = 1;`
	`while (v <= N/2)`
	`  v = 2*v;`
	`System.out.println(v);`

compute a finite sum $(1 + 2 + \ldots + N)$	`int sum = 0;`
	`for (int i = 1; i <= N; i++)`
	`  sum += i;`
	`System.out.println(sum);`

compute a finite product $(N! = 1 \times 2 \times \ \ldots \ \times N)$	`int product = 1;`
	`for (int i = 1; i <= N; i++)`
	`  product *= i;`
	`System.out.println(product);`

print a table of function values	`for (int i = 0; i <= N; i++)` `  System.out.println(i + " " + 2*Math.PI*i/N);`

Repeated operations

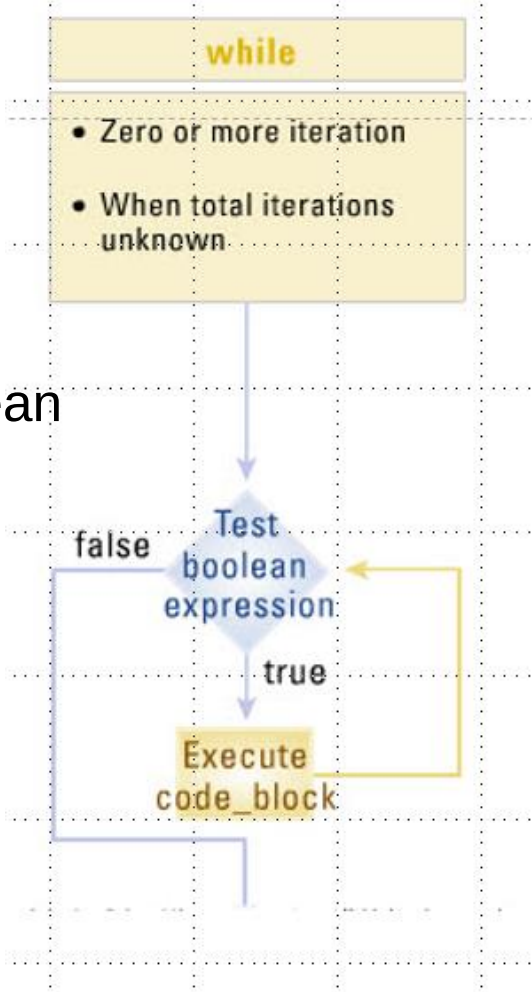
Concise code

Table: EXAMPLES

`while` statement

```
while (Boolean_Expression)
  Statement
```

- A `while` statement is used to repeat a portion of code (i.e., the loop body) based on the evaluation of a Boolean expression
  - The Boolean expression is checked ***before*** the loop body is executed
    - When **false**, the loop body is **not executed at all**
  - Before the execution of each following iteration of the loop body, the Boolean expression is checked again
    - If true, the loop body is executed again
    - If false, the loop statement ends
  - The loop body can consist of a single statement, or multiple statements enclosed in a pair of braces (`{ }`)



`while` Syntax

```
while (Boolean_Expression)
    Statement
```

Or

```
while (Boolean_Expression)
{
    Statement_1
    Statement_2
        ⋮
    Statement_Last
}
```

`while` statement: examples

- A `while` statement example

```
int count = 0;
while (count < 100) {  // boolean test within (…)
  System.out.println("count:" + count);  // body: statements within {...}
  count = count + 1; // or can be:   count++;
}
System.out.println("all done!");
```

Is there a difference b/w
count = count+1; and count++

```
int count2s(int num)
{
    int count = 0;   // count how many divisions done
    while (num >= 1)
    {
        num = num / 2;
        count++;
     }
  return count;
}
```

Write down an example of a method that uses a loop to count number of 2s in a given number

`do-while` Statement

- A `do-while` statement is used to execute a portion of code (i.e., the loop body), and then repeat it based on the evaluation of a Boolean expression
  - **The loop body is executed at least once**
    - The Boolean expression is checked ***after*** the loop body is executed
  - The Boolean expression is checked after each iteration of the loop body
    - If true, the loop body is executed again
    - If false, the loop statement ends
    - Don't forget to put a semicolon after the Boolean expression
  - Like the while statement, the loop body can consist of a single statement, or multiple statements enclosed in a pair of braces (`{ }`)

do-while Syntax

```
do
    Statement
while (Boolean_Expression);
```

or

```
do
{
    Statement_1
    Statement_2
        ⋮
    Statement_Last
  } while (Boolean_Expression);
```

do-while statement: examples

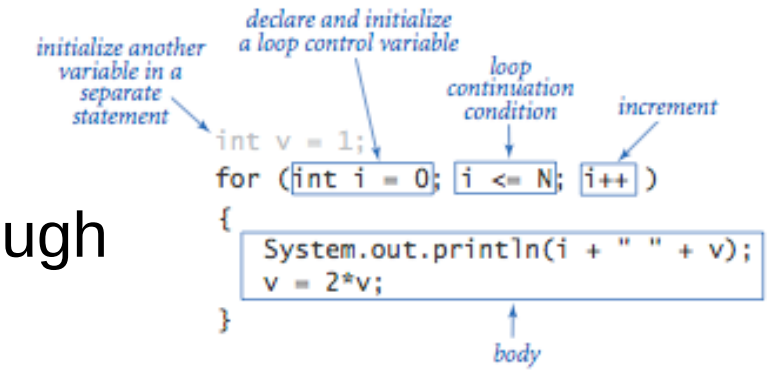
- A do-while statement example

```
int i =0;

  do
  {
    System.out.println("i is : " + i);
    i++;

  }while(i < 5);

int count =i;

System.out.println("count is : " + count);
```

Write down the output

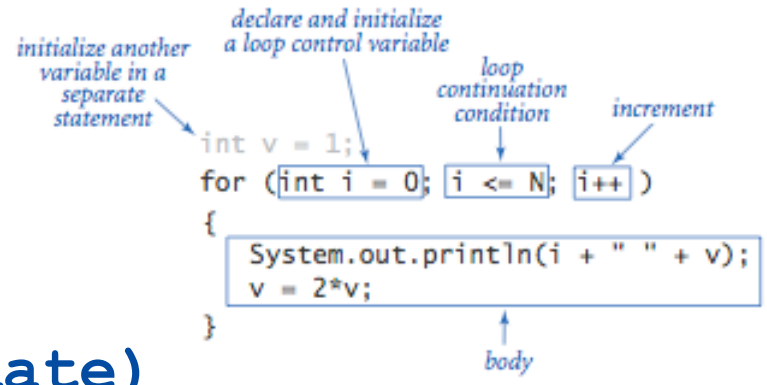
```
/*
Output would be
i is : 0
i is : 1
i is : 2
i is : 3
i is : 4
count is : 5
*/
```

The `for` Statement



- The `for` statement is most commonly used to step through an integer variable in equal increments
- It begins with the keyword `for`, followed by **three expressions** in parentheses that describe what to do with one or more *controlling variables*
  - The first expression tells how the control variable or variables are *initialized* or *declared* and *initialized* before the first iteration
  - The second expression determines when the loop should *end*, based on the evaluation of a Boolean expression *before* each iteration
  - The third expression tells how the control variable or variables are *updated after* each iteration of the loop body

The `for` Statement Syntax



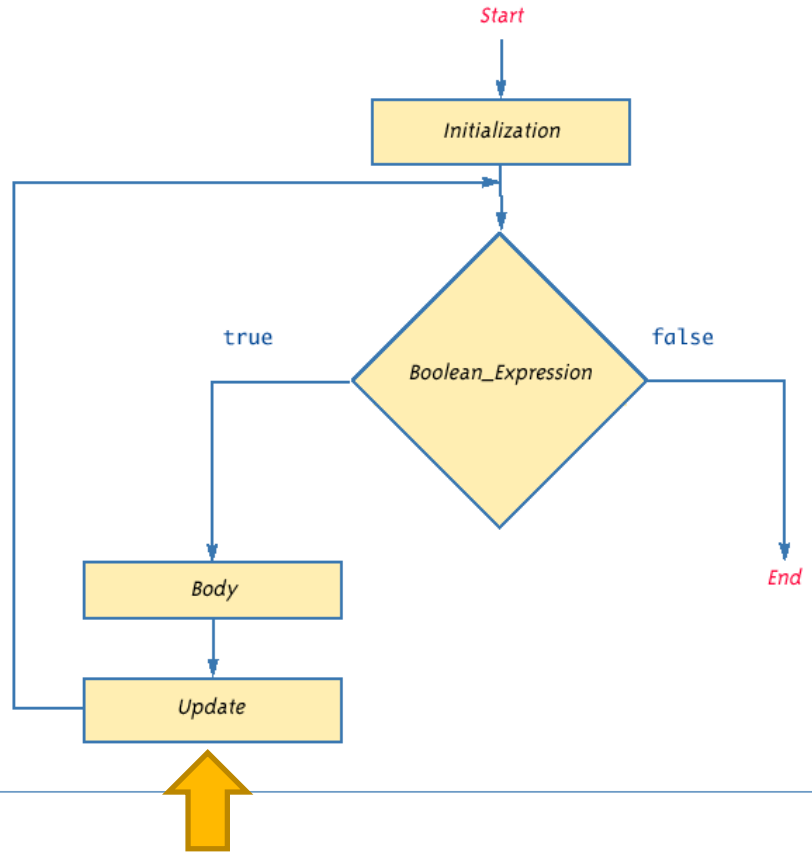

```
for (Initializing; Boolean_Expression; Update)
  Body
```

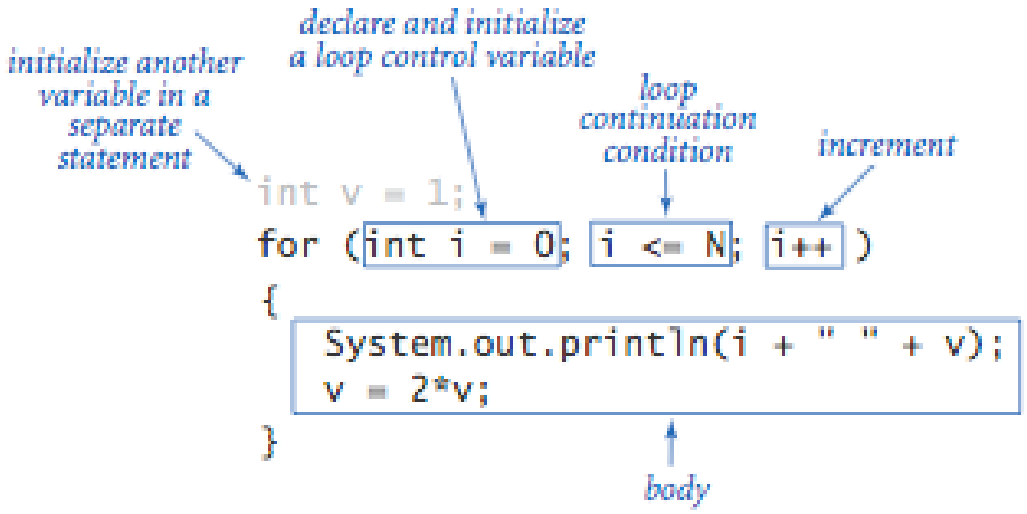
- The **`Body`** may consist of a single statement or a list of statements enclosed in a pair of braces (`{ }`)
- Note that the three control expressions are separated by **two semicolons**
- Note that there is **no semicolon after the closing** parenthesis at the beginning of the loop

Semantics of the `for` Statement

Display 3.9 Semantics of the for Statement

```
for (Initialization; Boolean_Expression; Update )
        Body
```





Three Parts of a `for` Loop

```
for ( initialization ; boolean condition ; update)
{
        //code that will loop
}
```

- **Initialization**
  - Used to set up a counter variable that will help keep track how many time the loop needs to execute
- **Boolean condition**
  - Code inside for loop will be executed as long as boolean condition is met
- **Update**
  - Will update the counter variable

Typical `for` Loop Example

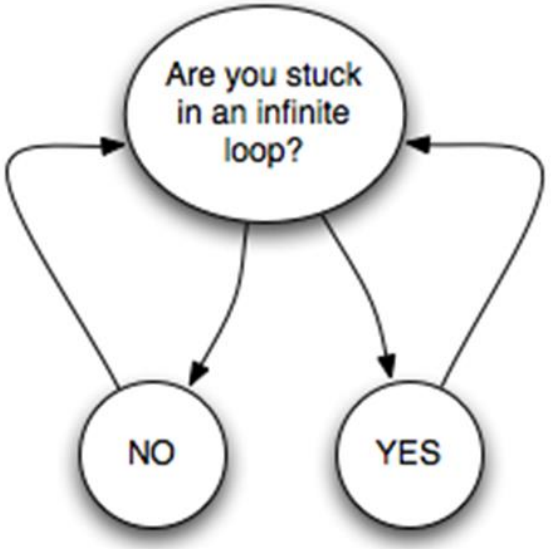
```
        1              2  5    4
for (int i = 0; i < 10; i++)
{
 3  System.out.print(i);
}
```

- Repeat steps 2 – 5 until boolean condition is not met.
- What does this code output?
- How many times is it executed?

VERY IMPORTANT

The boolean condition must eventually test to `false` or else the loop will never stop

- This is called an **infinite loop**



Typical `for` Loop Example

```
for (int i = 0; i < 10; i++)
{
    System.out.print(i);
}
```

- What will be difference, if any, if we say

  `i = 11  instead of  i < 10`

PRACTICE

Self Test

```
for (int i = 3; i < 10; i++){
    System.out.print(i + “, ”);
}
for (int i = 10; i >= 0; i--){
    System.out.print(i + “, ”);
}
for (int i = 0; i < 10; i = i + 2){
    System.out.print(i + “, ”);
}
```

3,4,5,6,7,8,9,10,9,8,7,6,5,4,3,2,1,0,0,2,4,6,8,

Nested `for` Loops

You can have **loops inside of loops**

```
for(int i = 0; i < 2; i++)
{
    for(int j = 0; j < 2; j++)
    {
        System.out.print("*");
    }
    System.out.println();
}
```

```
**
**
```

The `for` Statement: examples

- The `for` statement example

```
int num1, num2;

for (num2=0; num2<=9; num2++)
{
        for (num1=0; num1<=9; num1++)
        {
            System.out.println(num2+""+num1);
        }
}
```

Write down the values and output for first 3 iteration1
+
whiteboard

```
//part 2
for (num2=0; num2<=9; num2++)
{
    System.out.println("loop " +num2);
        for (num1=0; num1<=9; num1++)
        {
            System.out.println(num1);
        }
}
System.out.println("last num2 is " +num2);
```

loop 0
0
1
2
3
4
5
6
7
8
9
Loop 1
0
1
2

.....
Last num2 is 10

Write down the values and output for first 3 iteration1 for the second case

The `for` Statement: examples

- The `for` statement example

```
int num1, num2;

for (num2=0; num2<=9; num2++)
{
        for (num1=0; num1<=9; num1++)
        {
            System.out.println(num2+""+num1);
        }
}
```

Write down the values and output
for first 3 iteration1
+
whiteboard

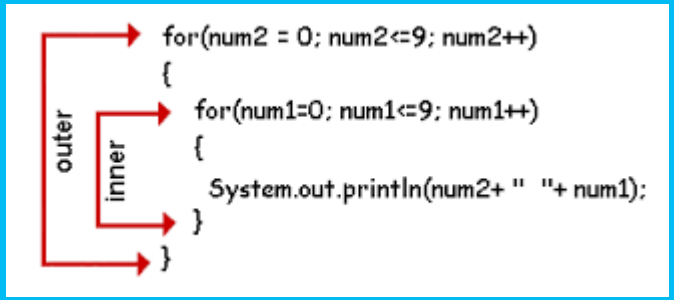
1) **< in a column>** 00 01 02 03 04 05 06 07 08 09 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99

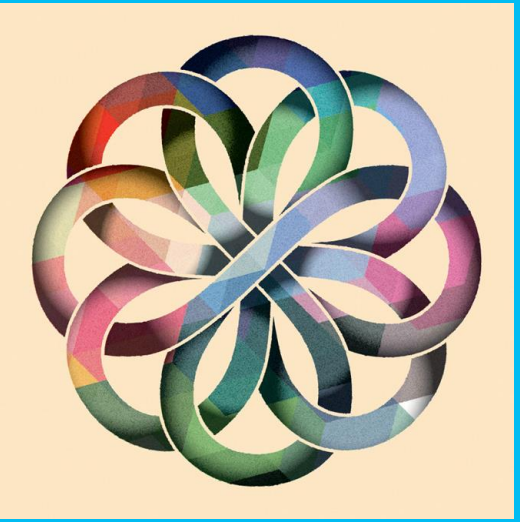
2) See previous page

The Comma in `for` Statements

- A `for` loop can contain multiple initialization actions separated with commas
  - Caution must be used when combining a declaration with multiple actions
  - It is illegal to combine multiple type declarations with multiple actions, for example
  - To avoid possible problems, it is best to declare all variables outside the `for` statement
- A `for` loop can contain multiple update actions, separated with commas, also
  - It is even possible to eliminate the loop body in this way
- However, a `for` loop can contain **only one Boolean expression** to test for ending the loop

USING LOOPS


for(num2 = 0; num2<=9; num2++)
{
for(num1=0; num1<=9; num1++)
{
System.out.println(num2+ " "+ num1);
}
}
outer
inner




INFINITE LOOP

`for` Statement Syntax and Alternate Semantics

Display 3.10 for Statement Syntax and Alternate Semantics *(Part 1 of 2)*

for Statement Syntax:

Syntax:

```
for (Initialization; Boolean_Expression; Update)
    Body
```

Example:

```
for (number = 100; number >= 0; number--)
    System.out.println(number
         + " bottles of beer on the shelf.");
```

Display 3.10 for Statement Syntax and Alternate Semantics *(Part 2 of 2)*

Equivalent while Loop:

Equivalent Syntax:

```
Initialization;
while (Boolean_Expression)
{
    Body
    Update;
}
```

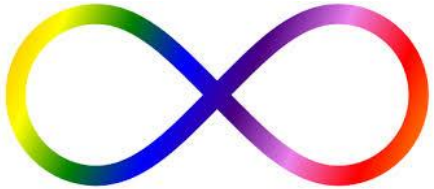
Equivalent Example:

```
number = 100;
while (number >= 0)
{
    System.out.println(number
         + " bottles of beer on the shelf.");

    number--;
}
```

Sample Dialogue

```
100 bottles of beer on the shelf.
99 bottles of beer on the shelf.
              .
                .
                .
0 bottles of beer on the shelf.
```

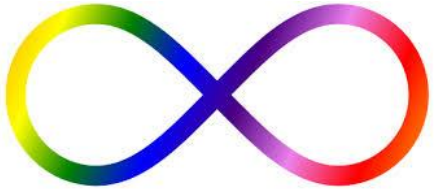
Infinite Loops



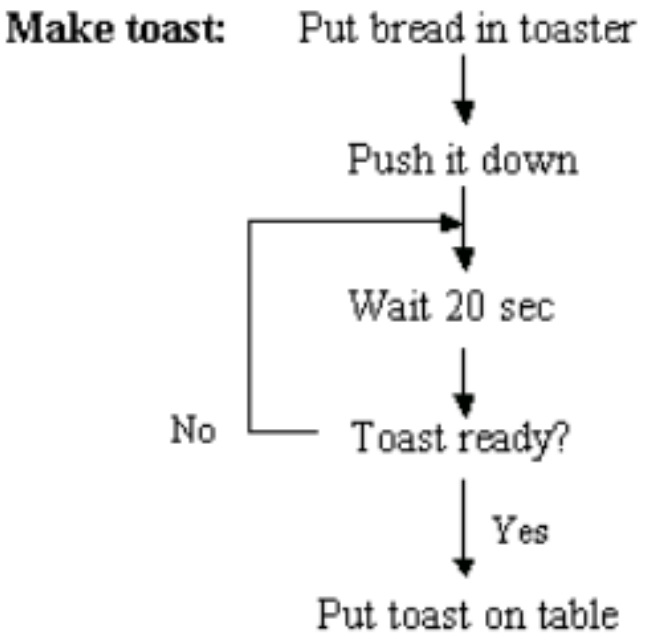
- A **`while`**, **`do-while`**, or **`for`** loop should be designed so that the value tested in the Boolean expression is changed in a way that eventually makes it false, and terminates the loop
- If the Boolean expression remains true, then the loop will run forever, resulting in an *infinite loop*
  - Loops that check for equality or inequality (**`==`** or **`!=`**) are especially prone to this error and should be avoided if possible

Why?

Infinite Loops

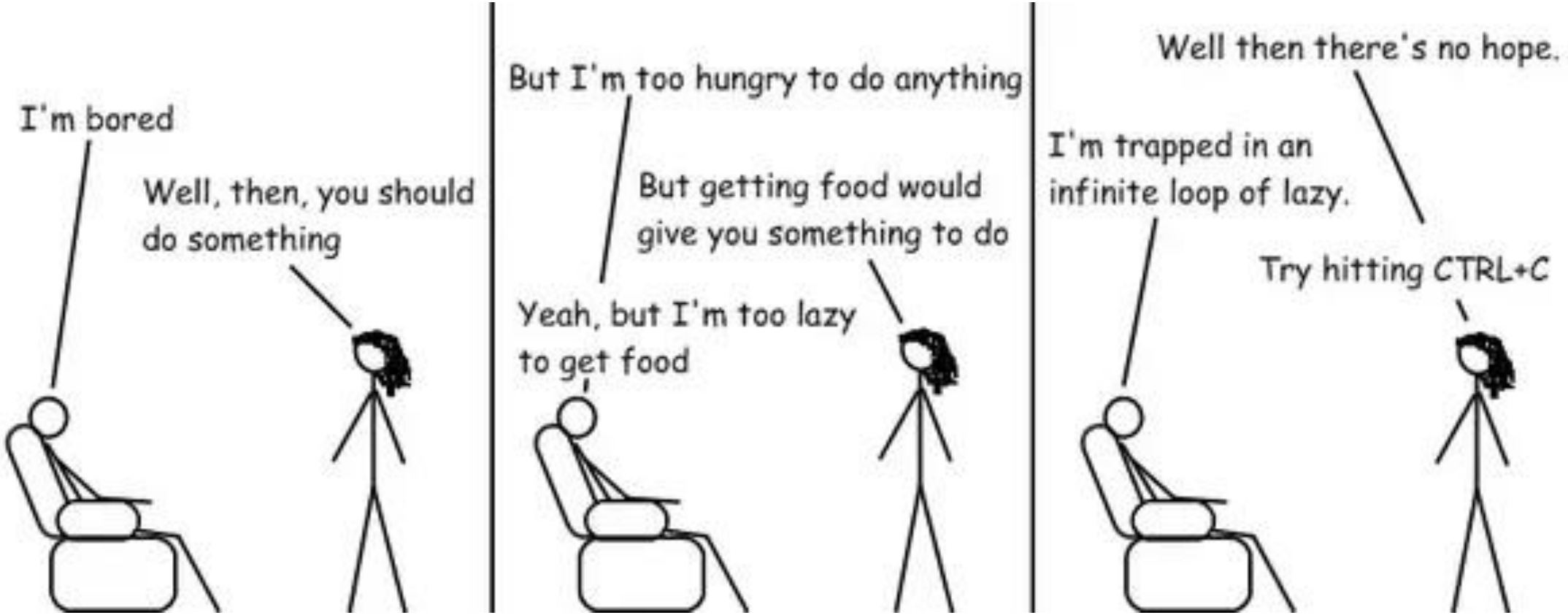


- Suppose your "Eat breakfast" routine contains the "Make toast" procedure:

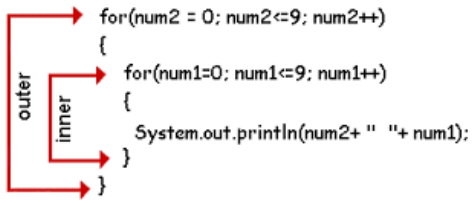


Suppose there was a power outage. Or the toaster is just simply broken. Both events would to an **infinite loop** (or at least a very long lasting loop) when you check every 20 sec whether the toast is ready or not. A well-behaved algorithm has to include a **timeout** provision to avoid infinite loops.

Infinite Loops



Nested Loops



- Loops can be *nested*, just like other Java structures
  - When nested, the **inner** loop iterates from beginning to end for each single iteration of the **outer** loop

```
int rowNum, columnNum;
for (rowNum = 1; rowNum <=3; rowNum++)
{
  for (columnNum = 1; columnNum <=2; columnNum++)
    System.out.print(" row " + rowNum + " column " + columnNum);
    System.out.println();
}
```

How many rows and columns?

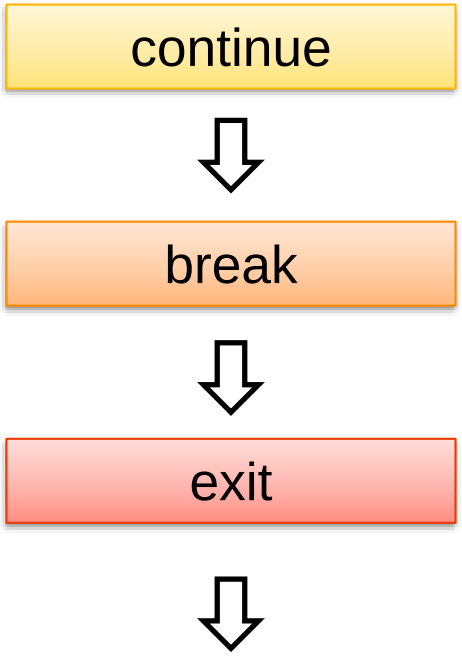
See output Savitch p.145

The `break` and `continue` Statements

- The `break` statement consists of the keyword `break` followed by a semicolon
  - When executed, the `break` statement ends the nearest enclosing switch or loop statement
- The `continue` statement consists of the keyword `continue` followed by a semicolon
  - When executed, the `continue` statement ends the current loop body iteration of the nearest enclosing loop statement
  - Note that in a `for` loop, the `continue` statement transfers control to the *update* expression
- When loop statements are nested, remember that any `break` or continue statement applies to the **innermost,** containing loop statement

The `exit` Statement

- A `break` statement will end a loop or switch statement, but will not end the program
- The `exit` statement will immediately end the program as soon as it is invoked:
  - `System.exit(0);`
- The `exit` statement takes one integer argument
  - By tradition, a zero argument is used to indicate a normal ending of the program



The Labeled **`break`** Statement

- There is a type of **`break`** statement that, when used in nested loops, can end any containing loop, not just the innermost loop
- If an enclosing loop statement is labeled with an *Identifier,* then the following version of the break statement will exit the labeled loop, even if it is not the innermost enclosing loop:

```
break someIdentifier;
```

- To label a loop, precede it with an *Identifier* and a colon:

```
someIdentifier:
```

```
topLoop:
 do
    {….
  while …..
       if ….
              for….
                 if …

                    break topLoop;
                   ……
       }
```

- The looping mechanism that always executes at least once is the _____ statement.
  1. if…else
  2. do…while
  3. while
  4. for

     A 2

- When the number of repetitions are known in advance, you should use a _____ statement.
  1. while
  2. do…while
  3. for
  4. None of the above

     A 3

- A _____ statement terminates the current iteration of a loop.
  1. Break
  2. Continue
  3. Switch
  4. Assert

     A 2

Hands-on: Class Activity (HoA)

Declaring and Initializing loop control variable

Checking condition

Incrementing loop control variable

```
for (int i =0; i<10 ; i++) {

// Loop statements to be executed

}
```

Hands-on: Activity 3b

Name/s:

Date:

Instructions: In pairs, work on the following problems **using pencil and paper**.

HoA 3b:

Problems 1-2: submit your solutions/output from today's class exercises

Problem 3. Write a program using one for loop and one if statement, that prints the integers from 1000 to 2000 with five integers per line. ***Hint:*** use the % operator.

Problem 4.
Write a program that calculate value of n!
Value **n** to be provided by user.

Problem 5: Print output for the following loop:

```
int i,j,k;
for (int i=0; i < 3; i++) {
    for (int j=0; j < 4; j++) {
        System.out.println("hi");
    }
    for (int k=0; k < 5; k++) {
        for (int l=0; l < 2; l++) {
            System.out.println("bye");
        }
    }
}
```