❖ Assume you have a Mouse object named “mouse” that is able to climb steps using the message:

```
mouse.step ();     // Note:  NO argument
```

❖ Write a code fragment to get the mouse to climb 5 steps

Loops

- Loops are used to make code more succinct by eliminating repeated lines of code.

- Write a code fragment that displays the numbers between 1 and 11 inclusive. Assume an **`OutputBox`** object named **`outBox`** has been declared, created, and sent the message **`outBox.show();`**

- These are examples of "count-controlled" loops. They must have steps to initialize, test, and increment (i.e., increase or decrease the counter).

- ❖ Write a code fragment to have a Mouse object named “mouse” climb to the next landing in a set of stairs. Assume the Mouse class has an instance method named **`atLanding()`** that returns a boolean value of **`true`** or **`false`**.

- ❖ How would your code change if you wanted the mouse to climb to the next Nth landing?

The while Statement

- Allows a program to execute a statement repeatedly while a condition is true

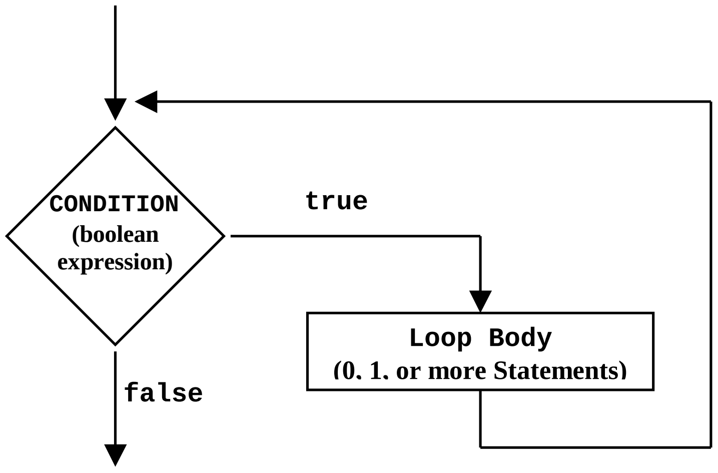
- Syntax:

```
while (<condition>) {
    <loop body>
}
```

`while` :	a Java reserved word
`<condition>`:	some boolean expression
`<loop body>`:	0, 1, or more Java statements

- Using curly braces {} is optional if only one statement is contained within the loop body. However, one should ALWAYS use curly braces regardless of the number of statements contained within the body (i.e., use "Defensive Programming").

- Control-Flow Diagram:



- As long as the **`<condition>`** evaluates to **`true`**, the **`<loop body>`** will continue to be executed.

The do...while Statement

❖ Allows a loop's condition to be tested AFTER it's loop body is executed

❖ Suitable when you want the loop body to execute at least once

❖ Syntax:

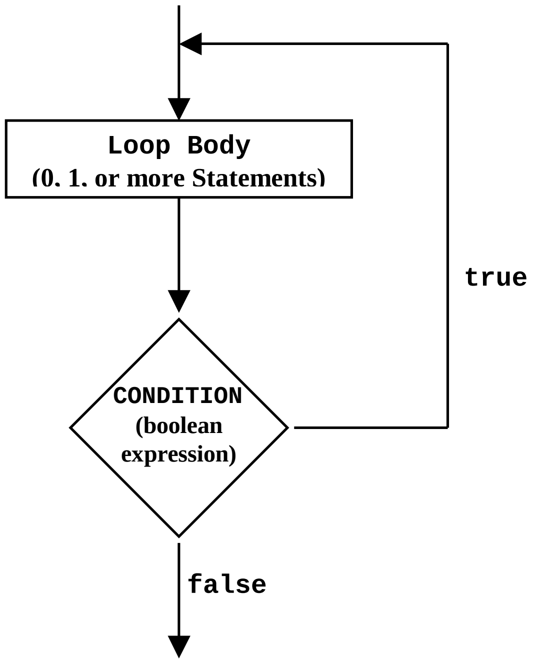
```
do {
    <loop body>

} while (<condition>);
```

`do`:	a Java reserved word
`<condition>`:	some boolean expression
`<loop body>`:	0, 1, or more Java statements

❖ Using curly braces {} is optional if only one statement is contained within the loop body. However, one should ALWAYS use curly braces regardless of the number of statements contained within the body (i.e., use "Defensive Programming").

❖ Control-Flow Diagram:



❖ The **`<loop body>`** will be executed at least once. Then, as long as the **`<condition>`** evaluates to **`true`**, the **`<loop body>`** will continue to be executed.

❖ A semi-colon ALWAYS terminates a do...while statement

The for Statement

❖ <u>Syntax</u>: **`for (<initialization>; <condition>; <increment>)`**

```
{
    <loop body>
}
```

`for`: a Java reserved word

`<loop body>`: 0, 1, or more Java statements

`<initialization>` (optional):

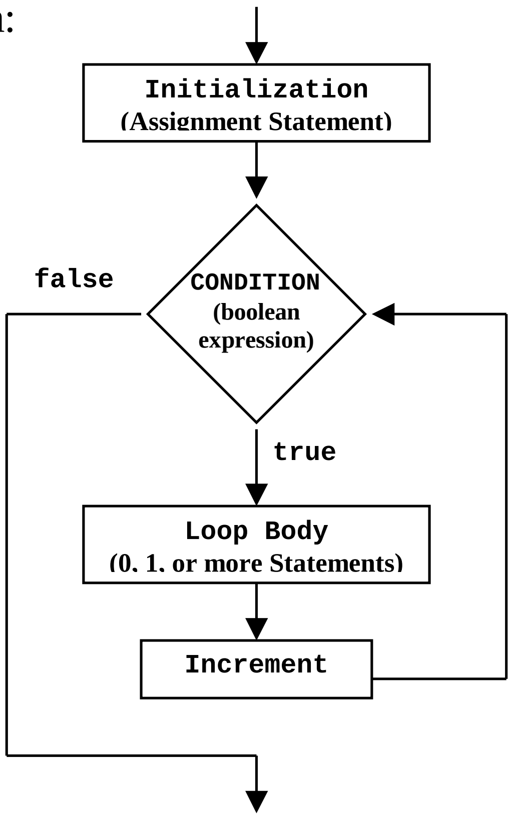
- an assignment statement (with or without declaration)
- the variable assigned a value is called the "<u>control variable</u>"
- example: **`int i = 0`**
  `count = 0` (count must be already declared)

`<condition>` (optional): some boolean expression

`<increment>` (optional):

- some arithmetic expression that increases or decreases a variable
- example **`i++`** **`j--`** **`i += 2`**

❖ Control-Flow Diagram:



The for Statement...continued

- ❖ Using curly braces {} is optional if only one statement is contained within the loop body. However, one should ALWAYS use curly braces regardless of the number of statements contained within the body (i.e., use "Defensive Programming").

- ❖ As long as the **`<condition>`** evaluates to **`true`**, the **`<loop body>`** will continue to be executed.

- ❖ Examples:

```
for (int i = 0; i < array.length; i++) {
    /*Do Something to each cell of array */
}
```

```
FoodCenterCollection fcc = tank.getFoodCenters ();

for (FoodCenter fc = fcc.nextFoodCenter();
      fcc.hasMoreFoodCenters();
      fc = fcc.nextFoodCenter()) {

  if (fc.getPosition().equals (fish1.getPostion()) {
      fish1.eat();
  }
}
```

- ❖ Syntactically legal for loops:

```
for ( ; ; ) {/* Do Something */}  // An infinite loop

for (int i = 0; ; ) { break;}     // Nothing more than
                                  // an assignment

int c = 0;
for ( ; c <= 10; c++) { outBox.printLine (c); }
```

The for Statement...continued

❖ Nested for loops: for loops can be nested within one another

```
for (int i = 1; i <= 4; i++) {
  for (int j = 1; j <= 3; j++) {
      outBox.print (“#”);
  }
outBox.skipLine ();
}
```

Rewrite the following for loop as a while loop:

```
int start = inBox.getInteger (“Start”);
int stop = inBox.getInteger (“Stop”);
int total = 0;

for (int count = start; count <= stop; count++)
{
    total += count;
}

outBox.printLine (“The total is: “ + total);
```

Loops...continued

- Infinite Loops: When the condition never evaluates to false

```
int product = 0;
while (product < 100) {
    product *= 10;
}

int count = 1;
while (count != 10) {
    count += 2;
}
// while loop eventually terminates due
// to an overflow error

double count = 0.0;
while (count < 1.0) {
    count += 0.3333333;
}
```

- Off-By-One Errors: When the loop repeats exactly one time more or less than the number of times it should repeat

```
int count = 1;
while (count < 10) {
    count++;
}
```

To have the loop repeat N times:

```
    counter = 0;
    while (counter < N) {}
```

OR

```
    counter = 1;
    while (counter <= N) {}
```

❖ <u>Pre-Test vs. Post-Test Loops</u>

Pre-Test (e.g., while, for): Test condition THEN execute loop body
Post-Test (e.g., do…while): Execute loop body THEN test condition

❖ <u>Priming</u>

Setting a variable to some value prior to the loop in order to ensure that the loop will be executed.

```
int n = 1, total = 0;
while (n != 0) {
    n = inBox.getInteger ();
    total += n;
}
```

❖ <u>Count-Controlled Loops</u>

Use of a counter in order to repeat a loop a precise number of times.

```
int count = 0, total = 0;
while (count <= 5) {
    total += count;
    count++;
}
```

❖ <u>Sentinel Controlled Loops</u>

Use a variable in order to repeat a loop until that variable reaches some set of "sentinel" values.

```
    /* See Priming for example */
```

❖ Flag Controlled Loops

Use a boolean variable (i.e., “flag”) in order to repeat a loop until the variable becomes false. The flag is set to true or false based on one or more conditions.

The following code sums the first 11 positive odd numbers entered by a user. The loop can end early if a number less than zero is entered.

```
boolean done =false;
int sum = 0, count = 0, num;

do {
    num = inBox.getInteger ();
    if (num % 2 == 0) { done = true; }
    else if (num < 0) { done = true; }
    else if (num == 0) { /* Do Nothing */ }
    else {
        sum += num;
        count++;
        if (count == 11) { done = true; }
    }
} while (!done);
```

class ResponseBox

- Defined as an instantiable class in the javabook package
- Requires an owner frame window (like a MainWindow object)
- Objects of the class provide up to three buttons (i.e., choices) that the user can select
- By default, there are two buttons: “yes” and “no”
- Can be used effectively with control statements to ask a user to repeat
- Example Code:

```
MainWindow mw = new MainWindow ();
ResponseBox rb = new ResponseBox (mw);

choice = rb.prompt (“Start Loop?”);

while (choice == ResponseBox.YES) {
  // code to do something
  choice = rb.prompt (“Continue Loop?”);
}
```

- To have >2 buttons, include an extra argument when creating a new ResponseBox object with ‘new’

```
ResponseBox threeButtonBox = new ResponseBox (mw, 3);
```

Write code to create a ResponseBox with three buttons labeled ONE, TWO, THREE. Using this ResponseBox, write code that displays a message indicating which button was clicked by the user.

```
MainWindow mw = new MainWindow ();
OutputBox outBox = new OutputBox (mw);
ResponseBox threeButtonBox = new ResponseBox (mw, 3);

threeButtonBox.setLabel (ResponseBox.BUTTON1, “ONE”);
threeButtonBox.setLabel (ResponseBox.BUTTON2, “TWO”);
threeButtonBox.setLabel (ResponseBox.BUTTON3, “THREE”);

int choice = threeButtonBox.prompt (“Click A Button”);

switch (choice) {

case ResponseBox.BUTTON1: outBox.print (“Button 1”);
                          break;

case ResponseBox.BUTTON2: outBox.print (“Button 2”);
                          break;

case ResponseBox.BUTTON3: outBox.print (“Button 3”);
                          break;

}
```

class Format

❖ Defined as a NON-instantiable class in the javabook package

❖ Can be used effectively with OutputBox objects to give the output certain formatting

❖ Example Code:

```
MainWindow mw = new MainWindow ();
OutputBox out = new OutputBox (mw);

int num1 = 6;
int num2 = 88;

out.printLine (“UNFORMATTED”);
out.printLine (“-----------”);

out.printLine (“num1: “ + num1);
out.printLine (“num2: “ + num2);

out.skipLine ();

out.printLine (“FORMATTED”);
out.printLine (“---------”);

out.printLine (“num1: “ + Format.rightAlign (3,
num1));
out.printLine (“num2: “ + Format.rightAlign (3,
num2));
```