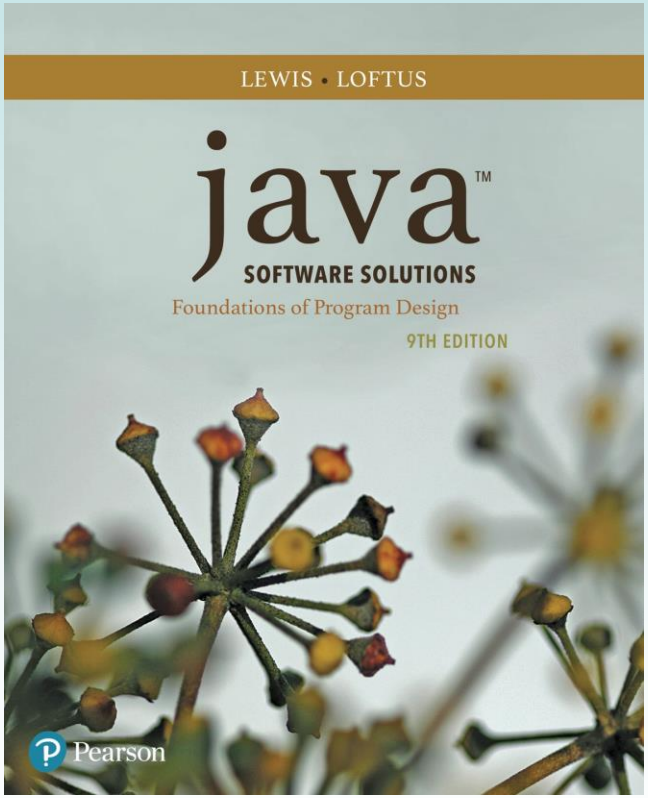
Chapter 6
More Conditionals and Loops



Java Software Solutions

Foundations of Program Design

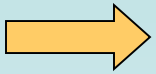
9th Edition

John Lewis
William Loftus

More Conditionals and Loops

- Now we can fill in some additional details regarding Java conditional and repetition statements
- Chapter 6 focuses on:
  - the `switch` statement
  - the conditional operator
  - the `do` loop
  - the `for` loop
  - using conditionals and loops with graphics
  - graphic transformations

Outline



The `switch` Statement

The Conditional Operator

The `do` Statement

The `for` Statement

Using Loops and Conditionals with Graphics

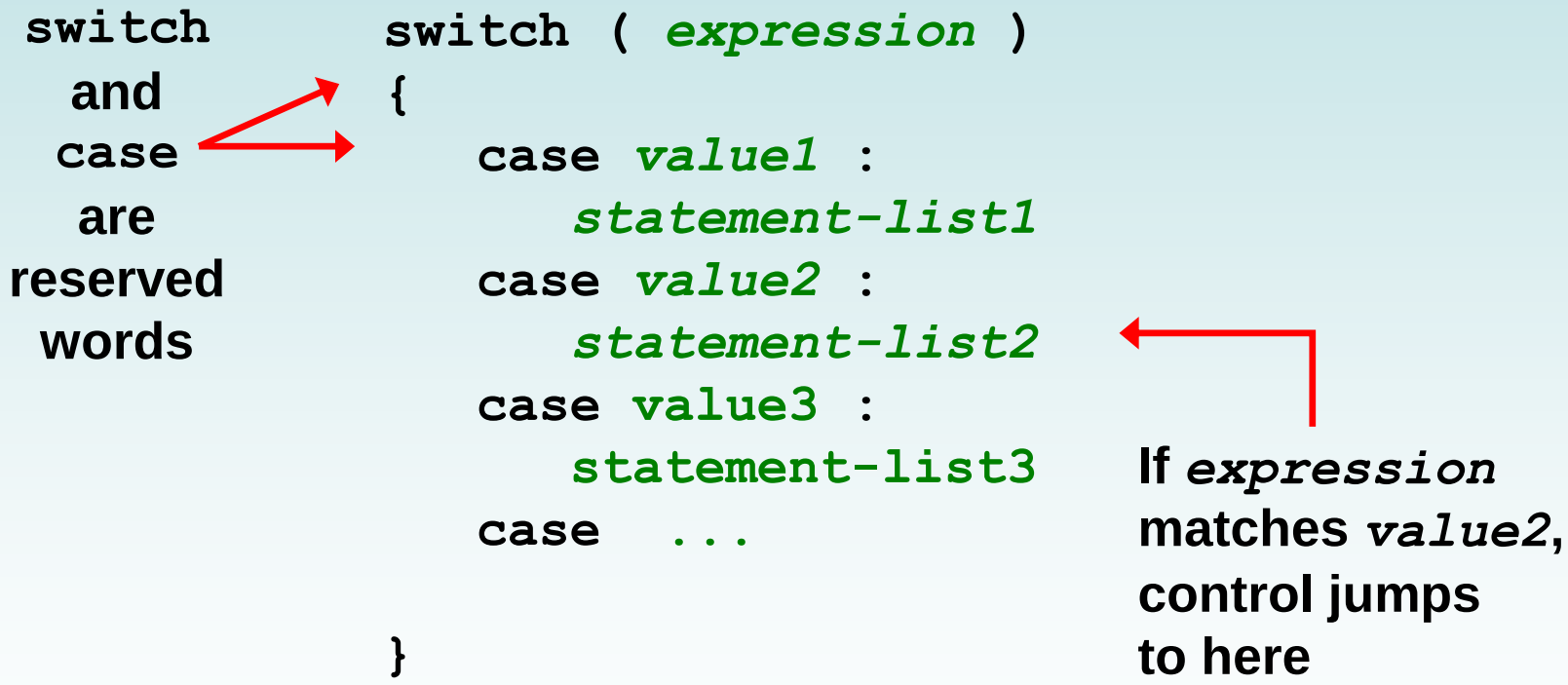
Graphic Transformations

The switch Statement

- The *switch statement* provides another way to decide which statement to execute next
- The `switch` statement evaluates an expression, then attempts to match the result to one of several possible *cases*
- Each case contains a value and a list of statements
- The flow of control transfers to statement associated with the first case value that matches

The switch Statement

- The general syntax of a `switch` statement is:



The switch Statement

- Often a *break statement* is used as the last statement in each case's statement list
- A `break` statement causes control to transfer to the end of the `switch` statement
- If a `break` statement is not used, the flow of control will continue into the next case
- Sometimes this may be appropriate, but often we want to execute only the statements associated with one case

The switch Statement

- An example of a `switch` statement:

```
switch (option)
{
   case 'A':
      aCount++;
      break;
   case 'B':
      bCount++;
      break;
   case 'C':
      cCount++;
      break;
}
```

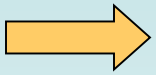
The switch Statement

- A `switch` statement can have an optional *default case*
- The default case has no associated value and simply uses the reserved word `default`
- If the default case is present, control will transfer to it if no other case value matches
- If there is no default case, and no other value matches, control falls through to the statement after the switch

The switch Statement

- The type of a `switch` expression can be integers, characters, enumerated types, or `String` objects
- You cannot use a `switch` with floating point values
- The implicit boolean condition in a `switch` statement is equality
- You cannot perform relational checks with a `switch` statement
- See `GradeReport.java`

Outline



The `switch` Statement

The Conditional Operator

The `do` Statement

The `for` Statement

Using Loops and Conditionals with Graphics

Graphic Transformations

The Conditional Operator

- The *conditional operator* evaluates to one of two expressions based on a boolean condition
- Its syntax is:

```
condition ? expression1 : expression2
```

- If the ***condition*** is true, ***expression1*** is evaluated; if it is false, ***expression2*** is evaluated
- The value of the entire conditional operator is the value of the selected expression

The Conditional Operator

- The conditional operator is similar to an `if-else` statement, except that it is an expression that returns a value
- For example:

```
larger = ((num1 > num2) ? num1 : num2);
```

- If `num1` is greater than `num2`, then `num1` is assigned to `larger`; otherwise, `num2` is assigned to `larger`
- The conditional operator is *ternary* because it requires three operands

The Conditional Operator

- Another example:

```
System.out.println("Your change is " + count +
   ((count == 1) ? "Dime" : "Dimes"));
```

- If `count` equals 1, then "Dime" is printed
- If `count` is anything other than 1, then "Dimes" is printed

Quick Check

Express the following logic in a succinct manner using the conditional operator.

```
if (val <= 10)
   System.out.println("It is not greater than 10.");
else
   System.out.println("It is greater than 10.");
```

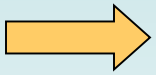
Quick Check

Express the following logic in a succinct manner using the conditional operator.

```
if (val <= 10)
   System.out.println("It is not greater than 10.");
else
   System.out.println("It is greater than 10.");


System.out.println("It is" +
   ((val <= 10) ? " not" : "") +
   " greater than 10.");
```

Outline



The `switch` Statement

The Conditional Operator

The `do` Statement

The `for` Statement

Using Loops and Conditionals with Graphics

Graphic Transformations

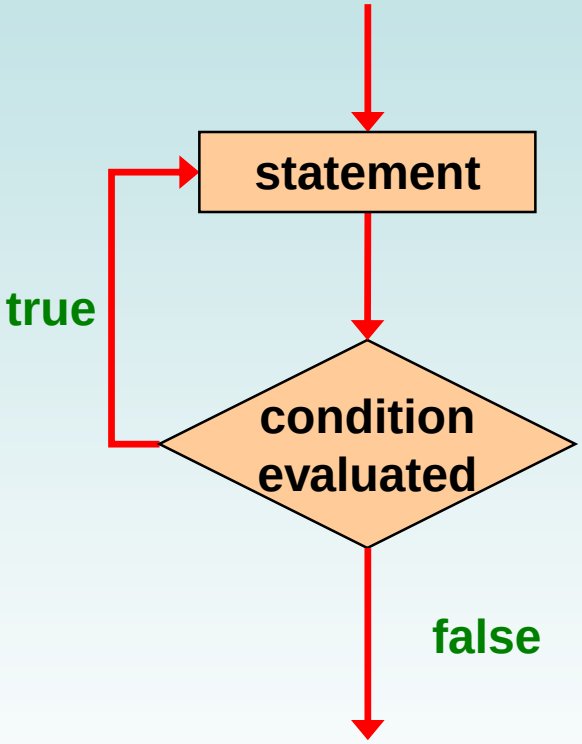
The do Statement

- A *do statement* has the following syntax:

```
do
{
    statement-list;
}
while (condition);
```

- The **`statement-list`** is executed once initially, and then the **`condition`** is evaluated
- The statement is executed repeatedly until the condition becomes false

Logic of a do Loop



The do Statement

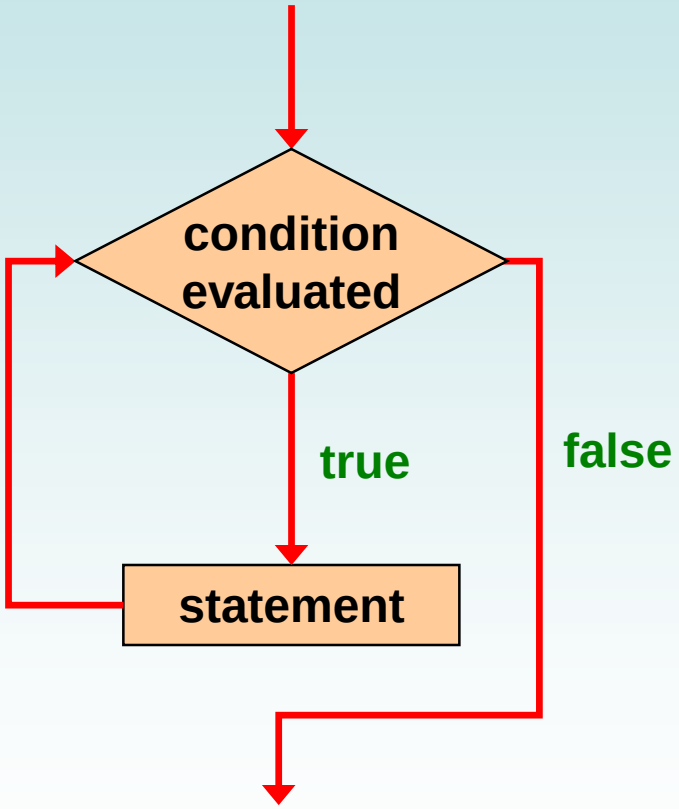
- An example of a `do` loop:

```
int count = 0;
do
{
   count++;
   System.out.println(count);
} while (count < 5);
```

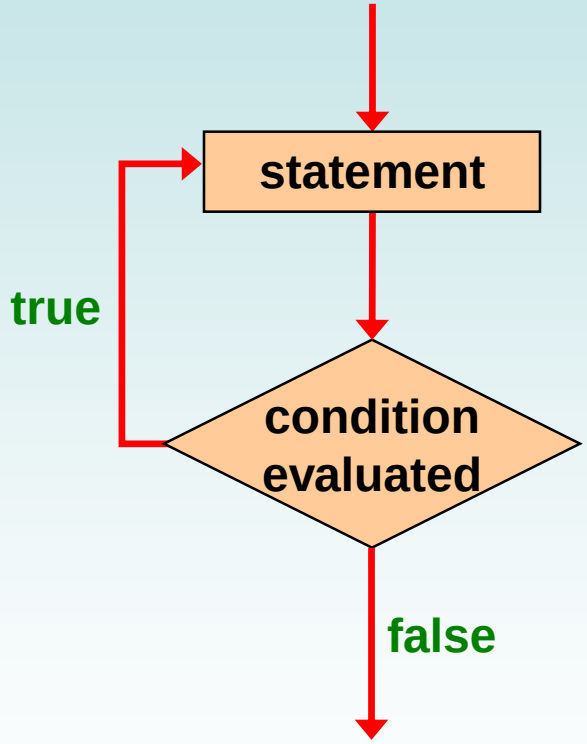
- The body of a `do` loop executes at least once
- See `ReverseNumber.java`

Comparing while and do

The while Loop



The do Loop

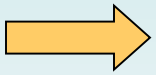


Outline

The `switch` Statement

The Conditional Operator

The `do` Statement



The `for` Statement

Using Loops and Conditionals with Graphics

Graphic Transformations

The for Statement

- A *for statement* has the following syntax:

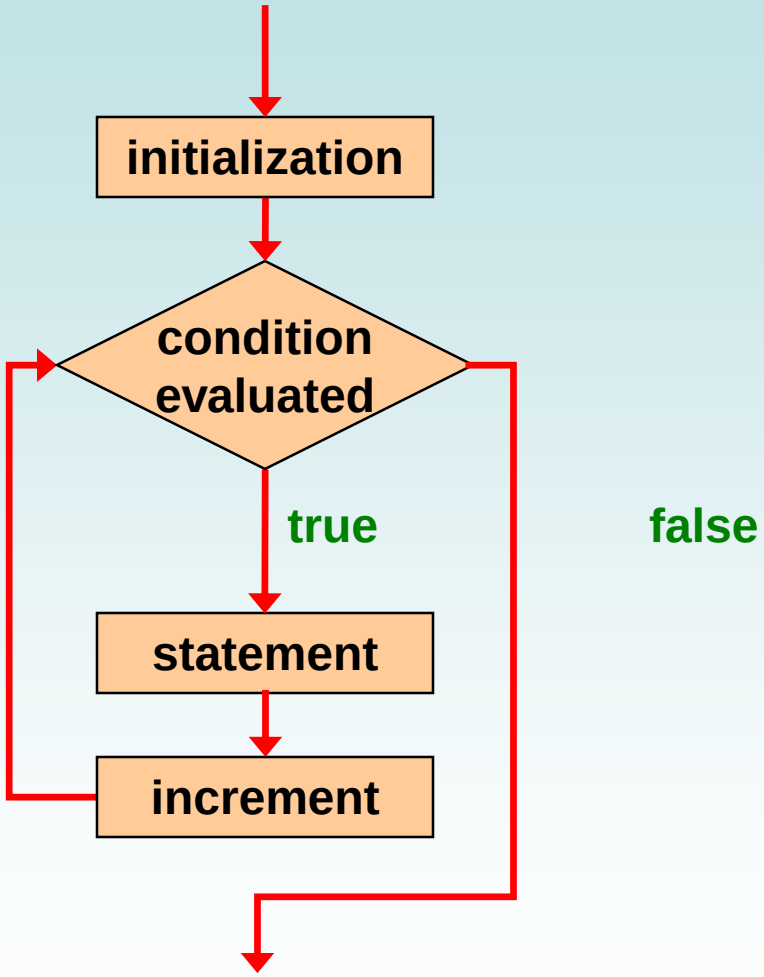
The *initialization* is executed once before the loop begins

The *statement* is executed until the *condition* becomes false

```
for ( initialization ; condition ; increment )
   statement;
```

The *increment* portion is executed at the end of each iteration

Logic of a for loop



The for Statement

- A `for` loop is functionally equivalent to the following `while` loop structure:

```
initialization;
while ( condition )
{
    statement;
    increment;
}
```

The for Statement

- An example of a `for` loop:

```
for (int count=1; count <= 5; count++)
   System.out.println(count);
```

- The initialization section can be used to declare a variable
- Like a `while` loop, the condition of a `for` loop is tested prior to executing the loop body
- Therefore, the body of a `for` loop will execute zero or more times

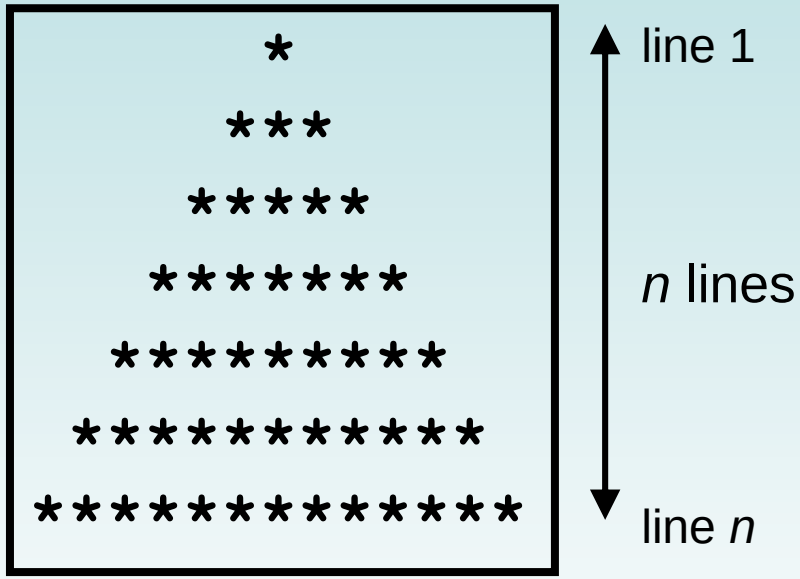
The for Statement

- The increment section can perform any calculation:

```
for (int num=100; num > 0; num -= 5)
   System.out.println(num);
```

- A `for` loop is well suited for executing statements a specific number of times that can be calculated or determined in advance
- See `Multiples.java`
- See `Stars.java`
- See also `Wedge.java` and `Tree.java`

Thought Process: Analyze Pattern



Program logic:
1. loop to print each line
2. inner loop print spaces
3. inner loop print stars

- Need to print ' ' and '*'
- How many ' ' for each line?
  - line *n* has 0 spaces
  - line *n*-1 has 1 space
  - (seems that line number plus number of spaces is *n*)
  - line 1 must have *n*-1 spaces
  - line *i* must have *n* - *i* spaces
- How many '*' for each line?
  - 1, 3, 5, … for lines 1, 2, 3, ---
  - line *i* seems to have 2**i*-1 '*'

Quick Check

Write a code fragment that rolls a die 100 times and counts the number of times a 3 comes up.

Quick Check

Write a code fragment that rolls a die 100 times and counts the number of times a 3 comes up.

```
Die die = new Die();
int count = 0;
for (int num=1; num <= 100; num++)
   if (die.roll() == 3)
      count++;
Sytem.out.println(count);
```

The for Statement

- Each expression in the header of a `for` loop is optional
- If the initialization is left out, no initialization is performed
- If the condition is left out, it is always considered to be true, and therefore creates an infinite loop
- If the increment is left out, no increment operation is performed

For-each Loops

- A variant of the `for` loop simplifies the repetitive processing of items in an iterator
- For example, suppose `bookList` is an `ArrayList<Book>` object
- The following loop will print each book:

```
for (Book myBook : bookList)
   System.out.println(myBook);
```

- This version of a for loop is often called a *for-each loop*

For-each Loops

- A for-each loop can be used on any object that implements the `Iterable` interface
- It eliminates the need to retrieve an iterator and call the `hasNext` and `next` methods explicitly
- It also will be helpful when processing arrays, which are discussed in Chapter 8

Quick Check

Write a for-each loop that prints all of the `Student` objects in an `ArrayList<Student>` object called `roster`.

Quick Check

Write a for-each loop that prints all of the `Student` objects in an `ArrayList<Student>` object called `roster`.

```
for (Student student : roster)
   System.out.println(student);
```

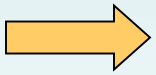
Outline

The `switch` Statement

The Conditional Operator

The `do` Statement

The `for` Statement



Using Loops and Conditionals with Graphics

Graphic Transformations

More Graphics

- Conditionals and loops enhance our ability to generate interesting graphics
- See `Bullseye.java`
- See `Boxes.java`

Outline

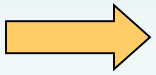
The `switch` Statement

The Conditional Operator

The `do` Statement

The `for` Statement

Using Loops and Conditionals with Graphics



Graphic Transformations

Graphic Transformations

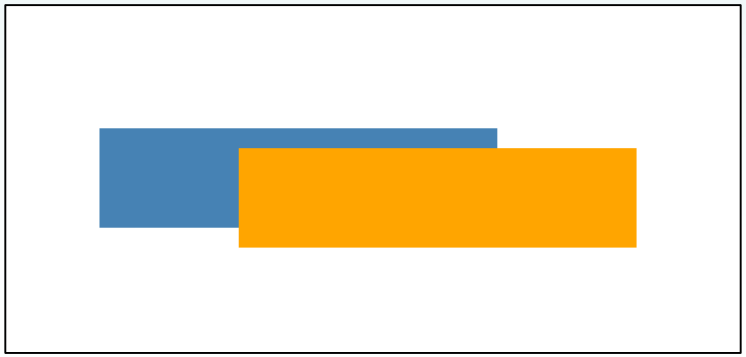
- A JavaFX *transformation* changes the way a node is presented visually
  - *translation* – shifts the position along the x or y axis
  - *scaling* – causes the node to appear larger or smaller
  - *rotation* – rotates the node around its center point
  - *shearing* – rotates one axis so that the x and y axes are no longer perpendicular

Translation

- The following creates two rectangles in the same position, then shifts the second one:

```
Rectangle rec1 = new Rectangle(100, 100, 200, 50);
rec1.setFill(Color.STEELBLUE);

Rectangle rec2 = new Rectangle(100, 100, 200, 50);
rec2.setFill(Color.ORANGE);
rec2.setTranslateX(70);
rec2.setTranslateY(10);
```



Scaling

- The following displays two `ImageView` objects, the second scaled to 70%:

```
Image img = new Image("water lily.jpg");
ImageView imgView1 = new ImageView(img);

ImageView imgView2 = new ImageView(img);
imgView2.setX(300);
imgView2.setScaleX(0.7);
imgView2.setScaleY(0.7);
```

Rotation

- The parameter to `setRotate` determines how many degrees the node is rotated
- If the parameter positive, the node is rotated clockwise
- If the parameter is negative, the node is rotated counterclockwise

Rotation

```
Rectangle rec = new Rectangle(50, 100, 200, 50);
rec.setFill(Color.STEELBLUE);
rec.setRotate(40);

Text text = new Text(270, 125, "Tilted Text!");
text.setFont(new Font("Courier", 24));
text.setRotate(-15);
```

Rotation

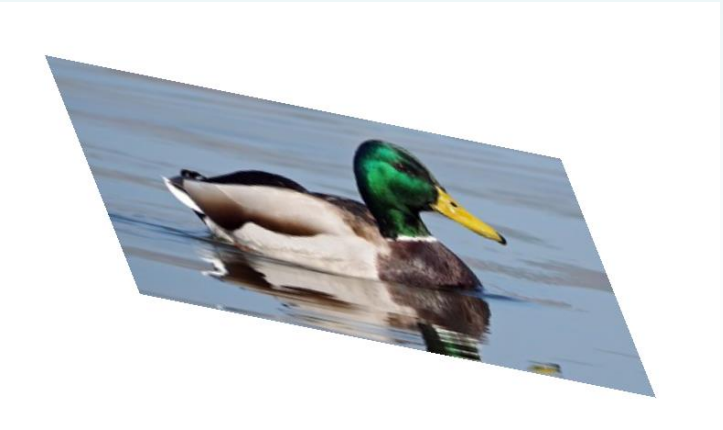
- To rotate a node around a point other than its center point, create a `Rotate` object and add it to the node's list of transformations
- The following rotates a node 45 degrees around the point (70, 150):

```
node.getTransforms().add(new Rotate(45, 70, 150));
```

Shearing

- Shearing is accomplished by creating a `Shear` object and adding it to this list of transformations
- The following applies a shear of 40% on the x axis and 20% on the y axis to an `ImageView` object:

```
Image img = new Image("duck.jpg");
ImageView imgView = new ImageView(img);
imgView.getTransforms().add(new Shear(0.4, 0.2));
```

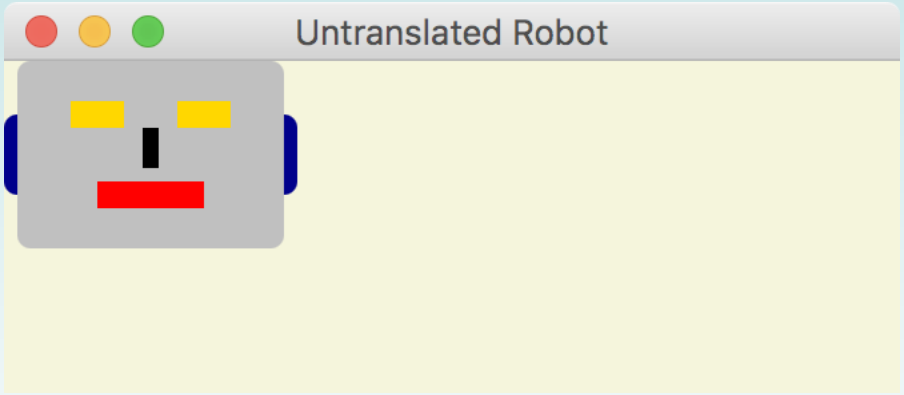


Transformations on Groups

- Transformations can be applied to any JavaFX nodes
    - shapes, images, controls
    - groups and panes
- When applied to a group or pane, the transformation is applied to each node it contains
- See `RobotFace.java`
- See `Robots.java`

Transformations on Groups

- If presented as defined, the robot face would be displayed in the upper left corner:



Summary

- Chapter 6 focused on:
  - the `switch` statement
  - the conditional operator
  - the `do` loop
  - the `for` loop
  - using conditionals and loops with graphics
  - graphic transformations