Any fool can know. The point is to understand.
A. Enistein
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:

```java
switch (expression) {
    case value1:
        statement-list1
    case value2:
        statement-list2
    case value3:
        statement-list3
    case ...
}
```

`switch` and `case` are reserved words. If `expression` matches `value2`, control jumps to here.
The switch Statement

• An example of a switch statement:

```java
switch (option)
{
    case 'A':
        aCount++;
        break;
    case 'B':
        bCount++;
        break;
    case 'C':
        cCount++;
        break;
}
```
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 (more) often we want to execute only the statements associated with one case.
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 must be integers, characters, or enumerated types

• As of Java 7, a switch can also be used with strings

• 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
import java.util.Scanner;

public class GradeReport
{
    // -----------------------------------------------------------------------------------
    //  Reads a grade from the user and prints comments accordingly.
    // -----------------------------------------------------------------------------------
    public static void main (String[] args)
    {
        int grade, category;

        Scanner scan = new Scanner (System.in);

        System.out.print ("Enter a numeric grade (0 to 100): ");
        grade = scan.nextInt();

        category = grade / 10;

        System.out.print ("That grade is ");

        continue
continue

    switch (category)
    {
    case 10:
        System.out.println("a perfect score. Well done.");
        break;
    case 9:
        System.out.println("well above average. Excellent.");
        break;
    case 8:
        System.out.println("above average. Nice job.");
        break;
    case 7:
        System.out.println("average.");
        break;
    case 6:
        System.out.println("below average. You should see the");
        System.out.println("instructor to clarify the material " + "presented in class.");
        break;
    default:
        System.out.println("not passing.");
    }
    }
switch (category) {
    case 10:
        System.out.println("a perfect score. Well done.");
        break;
    case 9:
        System.out.println("well above average. Excellent.");
        break;
    case 8:
        System.out.println("above average. Nice job.");
        break;
    case 7:
        System.out.println("average.");
        break;
    case 6:
        System.out.println("below average. You should see the");
        System.out.println("instructor to clarify the material "+"presented in class.");
        break;
    default:
        System.out.println("not passing.");
    }
}
public class SwitchExample {

    private enum Color {red, green, blue}

    public static void main(String[] args) {
        Color col = Color.blue;
        switch (col) {
            // note that you do not write "case Color.blue:
            case blue:
                System.out.println("It is blue");
                break;
            case red:
                System.out.println("It is red");
                break;
            case green:
                System.out.println("It is green");
                break;
            default:
                System.out.println("No color");
        }
    }
}
Switch and String

col = Color.green;
switch (col.name()) {
  case "blue":
    System.out.println("It is blue");
    break;
  case "red":
    System.out.println("It is red");
    break;
  case "green":
    System.out.println("It is green");
    break;
}
Quiz

You might choose to use a switch statement instead of nested if-else statements if

A) the variable being tested might equal one of several hundred int values
B) the variable being tested might equal one of only a few int values
C) there are two or more int variables being tested, each of which could be one of several hundred values
D) there are two or more int variables being tested, each of which could be one of only a few values
E) none of the above, you would never choose to use a switch statement in place of nested if-else statements under any circumstance
Quiz

You might choose to use a switch statement instead of nested if-else statements if

A) the variable being tested might equal one of several hundred int values

B) the variable being tested might equal one of only a few int values

C) there are two or more int variables being tested, each of which could be one of several hundred values

D) there are two or more int variables being tested, each of which could be one of only a few values

E) none of the above, you would never choose to use a switch statement in place of nested if-else statements under any circumstance

The switch statement can only be used if there is a single variable being tested and it is an integral type (an int or a char in Java). Further, because you have to enumerate each possible value being tested, the switch statement only makes sense if the number of values being tested is a small number.
Exercise

• Write a new toString() method of the class Die that returns a string with the English name of the faceValue of the die – use switch
public String toString() {
    String result = "";
    switch (faceValue) {
    case 1:
        result = "One";
        break;
    case 2:
        result = "Two";
        break;
    case 3:
        result = "Three";
        break;
    case 4:
        result = "Four";
        break;
    case 5:
        result = "Five";
        break;
    case 6:
        result = "Six";
        break;
    }
    return result;
}
Quiz

If $x$ is currently equal to 5, what will the value of $x$ be after the switch statement executes?

```java
switch (x) {
    case 3 : x += 1;
    case 4 : x += 2;
    case 5 : x += 3;
    case 6 : x++;
    case 7 : x += 2;
    case 8 : x--;
    case 9 : x++;
}
```

A) 8  B) 6  C) 11  D) 5  E) 10
Quiz

If x is currently equal to 5, what will the value of x be after the switch statement executes?

```
switch (x) {
    case 3 : x += 1;
    case 4 : x += 2;
    case 5 : x += 3;
    case 6 : x++;  
    case 7 : x += 2;
    case 8 : x--;  
    case 9 : x++;  
}
```

A) 8   B) 6   C) 11   D) 5   E) 10
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:

  \[
  \text{condition} \ ? \ \text{expression1} : \ \text{expression2}
  \]

• 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.

• It is an operator not a statement.
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:

\[
larger = ((\text{num1} > \text{num2}) \ ? \ \text{num1} \ : \ \text{num2});
\]

• 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 the value of the conditional expression

• If `count` is anything other than 1, then the conditional operator evaluates to "Dimes"
Express the following logic in a succinct manner using the conditional operator.

```java
if (val <= 10)
    System.out.println("It is not greater than 10.");
else
    System.out.println("It is greater than 10.");
```
Express the following logic in a succinct manner using the conditional operator.

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

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

The statement:

\[
\text{if (} x < 0 \text{) } y = x; \text{ else } y = 0;
\]

can be rewritten using a conditional operator as:

A) \( y = (x < 0) \ ? \ x : 0; \)
B) \( x = (x < 0) \ ? \ y : 0; \)
C) \( (x < 0) \ ? \ y = x : y = 0; \)
D) \( y = (x < 0); \)
E) \( y = \text{if (} x < 0 \text{) } x : 0; \)
Quiz

• The statement if \((x < 0)\) \(y = x;\) else \(y = 0;\) can be rewritten using a conditional operator as:

A) \(y = (x < 0) \ ? \ x \ : \ 0;\)
B) \(x = (x < 0) \ ? \ y \ : \ 0;\)
C) \((x < 0) \ ? \ y = x : y = 0;\) \(\leftarrow\) Remember the conditional operator is not a statement
D) \(y = (x < 0);\)
E) \(y = \text{if} \ (x < 0) \ x : 0;\)
Quiz

• Is this code syntactically correct? If yes, what is printing?

```java
int y = 1, x = 2, z;
z = (x < 0) ? (y = x) : (y = 0);
System.out.println("y = " + y);
```
Quiz

• What is the difference between a conditional operator and a conditional statement?
Quiz

• What is the difference between a conditional operator and a conditional statement?

• A conditional operator is a ternary operator that evaluates a condition and produces one of two possible values

• A conditional statement (e.g., if) is a category of statements that allow conditions to be evaluated and the appropriate statements executed as a result.
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:

```plaintext
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

true

condition evaluated

false

statement
The do Statement

- An example of a `do` loop:

```java
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`

What is this loop printing?

36
import java.util.Scanner;

public class ReverseNumber
{
    // Reverses the digits of an integer mathematically.
    public static void main(String[] args)
    {
        int number, lastDigit, reverse = 0;

        Scanner scan = new Scanner(System.in);

        continue
System.out.print("Enter a positive integer: ");
number = scan.nextInt();

\{
    lastDigit = number % 10;
    reverse = (reverse * 10) + lastDigit;
    number = number / 10;
\}
while (number > 0);

System.out.println("That number reversed is "+ reverse);
}
System.out.print( "Enter a positive integer: ");
number = scan.nextInt();
do {
    lastDigit = number % 10;
    reverse = (reverse * 10) + lastDigit;
    number = number / 10;
} while (number > 0);
System.out.println("That number reversed is ");
Comparing while and do

The **while** Loop

- Condition evaluated
- True: Statement
- False: Loop continues

The **do** Loop

- Statement
- True: Condition evaluated
- False: Loop continues
Quiz

How many times will the following loop iterate?

```java
int x = 10;
do {
    System.out.println(x);
    x--;
} while (x > 0);
```
Quiz

How many times will the following loop iterate?

```java
int x = 10;
do {
    System.out.println(x);
    x--;
} while (x > 0);
```

10 times
Exercise

• Write a code fragment (using do loop) that prints the characters stored in a String object called str backwards. Hint: use the method of String charAt(int).

Solution

```java
String str = "arbatax";
int index = str.length() - 1;
do {
    System.out.print(str.charAt(index));
    index--;
} while (index >= 0);
```

• But if the string is "", this code produces a run time error
• We must do a test before entering the loop
• Better use a while loop here
Alternative solution

String str = "arbatax";
int index = str.length() - 1;
while (index >=0) {
    System.out.print(str.charAt(index));
    index--;
}

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:

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

The *initialization* is executed once before the loop begins.

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

The *increment* portion is executed at the end of each iteration.
Logic of a for loop

- **Initialization**: The loop starts with an initialization step.
- **Condition Evaluated**: The condition is evaluated.
  - If **true**, the statement is executed.
  - If **false**, the increment step is taken and the condition is re-evaluated.
- **Statement**: The statement inside the loop is executed.
- **Increment**: The loop variable is incremented.

This flowchart illustrates the logic of a for loop, where the loop executes as long as the condition is true.
Example of a for loop

• i is initialized to 0
• i is compared to 5
  – if this is true x is incremented with i
• i is incremented by 1
• i is compared again with 5
  – ...

```cpp
int x = 0;
for (int i=0; i<5; i++)
    x += i;
```
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:

```java
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:

```java
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
import java.util.Scanner;

public class Multiples {
    // Prints multiples of a user-specified number up to a user-specified limit.
    public static void main (String[] args) {
        final int PER_LINE = 5;
        int value, limit, mult, count = 0;

        Scanner scan = new Scanner (System.in);

        System.out.print ("Enter a positive value: ");
        value = scan.nextInt();
        continue
    }
}
System.out.print ("Enter an upper limit: ");
limit = scan.nextInt();

System.out.println ();
System.out.println ("The multiples of "+ value + ", between "+ value + ", and "+ limit + ", (inclusive) are:");

for (mult = value; mult <= limit; mult += value)
{
    System.out.print (mult + \\
                       "t");

    // Print a specific number of values per line of output
    count++;
    if (count % PER_LINE == 0)
        System.out.println();
}
}
System.out.print("Enter an upper limit: ");
limit = scan.nextInt();
System.out.println();
System.out.println("The multiples of "+value+" between "+value+" and "+limit+" (inclusive) are:");
for(mult = value; mult <= limit; mult += value){
    System.out.print(mult+" ");
    // Print a specific number of values per line of output
    count++;
    if(count % PER_LINE == 0)
        System.out.println();
}

Sample Run
Enter a positive value: 7
Enter an upper limit: 400
The multiples of 7 between 7 and 400 (inclusive) are:
7 14 21 28 35
42 49 56 63 70
77 84 91 98 105
112 119 126 133 140
147 154 161 168 175
182 189 196 203 210
217 224 231 238 245
252 259 266 273 280
287 294 301 308 315
322 329 336 343 350
357 364 371 378 385
392 399
public class Stars {
    // Prints a triangle shape using asterisk (star) characters.
    public static void main (String[] args) {
        final int MAX_ROWS = 10;

        for (int row = 1; row <= MAX_ROWS; row++) {
            for (int star = 1; star <= row; star++)
                System.out.print("*");

            System.out.println();
        }
    }
}
public class Stars {

    // Prints a triangle shape using asterisk (star) characters.

    public static void main(String[] args) {
        final int MAX_ROWS = 10;
        for (int row = 1; row <= MAX_ROWS; row++) {
            for (int star = 1; star <= row; star++)
                System.out.print("*");
            System.out.println();
        }
    }
}
Exercise

How many iterations will the following for loops execute?

a for (int i = 0; i < 20; i++) {}

b for (int i = 0; i <= 20; i++) {}

c for (int i = 5; i < 20; i++) {}

d for (int i = 20; i > 0; i--) {}

e for (int i = 1; i < 20; i=i+2) {}

f for (int i = 1; i < 20; i*=2) {}
Exercise

How many iterations will the following for loops execute?

<table>
<thead>
<tr>
<th>Loop</th>
<th>Description</th>
<th>Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>for (int i = 0; i &lt; 20; i++) {}</td>
<td>20</td>
</tr>
<tr>
<td>b</td>
<td>for (int i = 0; i &lt;= 20; i++) {}</td>
<td>21</td>
</tr>
<tr>
<td>c</td>
<td>for (int i = 5; i &lt; 20; i++) {}</td>
<td>15</td>
</tr>
<tr>
<td>d</td>
<td>for (int i = 20; i &gt; 0; i--) {}</td>
<td>20</td>
</tr>
<tr>
<td>e</td>
<td>for (int i = 1; i &lt; 20; i = i+2) {}</td>
<td>10</td>
</tr>
<tr>
<td>f</td>
<td>for (int i = 1; i &lt; 20; i* = 2) {}</td>
<td>5</td>
</tr>
</tbody>
</table>
Quiz

Given that s is a String, what does the following loop do?

```java
for (int j = s.length(); j > 0; j--)
    System.out.print(s.charAt(j-1));
```

A) it prints s out backwards
B) it prints s out forwards
C) it prints s out backwards after skipping the last character
D) it prints s out backwards but does not print the 0th character
E) it yields a run-time error because there is no character at s.charAt(j-1) for j = 0
Given that $s$ is a String, what does the following loop do?

```java
for (int j = s.length(); j > 0; j--)
    System.out.print(s.charAt(j-1));
```

A) it prints $s$ out backwards
B) it prints $s$ out forwards
C) it prints $s$ out backwards after skipping the last character
D) it prints $s$ out backwards but does not print the 0th character
E) it yields a run-time error because there is no character at $s$.charAt(j-1) for $j = 0$
Quick Check

Write a code fragment that rolls a die 100 times and counts the number of times a 3 comes up and finally prints the count. Use the for loop!
Use the roll() methods of the Die class (it returns the new int faceValue of the Die object on which is called).
Quick Check

Write a code fragment that rolls a die 100 times and counts the number of times a 3 comes up. Use the `for` loop!

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

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:

```java
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` (from the `Iterable` by calling the `iterator()` method) 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.

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

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

• Transform the following `while` loop in a program with the same semantic but using `for`

```java
while (x > 2) {
    System.out.println(x);
    x++;  
}
```
Quiz

• Transform the following `while` loop in a program with the same semantic but using `for`:

```java
while (x > 2) {
    system.out.println(x);
    x++;  
}

for(;x > 2; x++) {
    system.out.println(x);
}
```
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
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;

//************************************************************************
// Bullseye.java Author: Lewis/Loftus
//
// Demonstrates the use of loops and conditionals to draw.
//************************************************************************

class Bullseye extends Application
{
    //--------------------------------------------------------------------
    // Displays a target using concentric black and white circles
    // and a red center.
    //--------------------------------------------------------------------
    public void start(Stage primaryStage)
    {
        Group root = new Group();
        Color ringColor = Color.BLACK;
        Circle ring = null;
        int radius = 150;
        continue
for (int count = 1; count <= 8; count++)
{
    ring = new Circle(160, 160, radius);
    ring.setFill(ringColor);
    root.getChildren().add(ring);

    if (ringColor.equals(Color.BLACK))
        ringColor = Color.WHITE;
    else
        ringColor = Color.BLACK;

    radius = radius - 20;
}

ring.setFill(Color.RED);

Scene scene = new Scene(root, 320, 320, Color.CYAN);

primaryStage.setTitle("Bullseye");
primaryStage.setScene(scene);
primaryStage.show();
continue

for (int count = 1; count <= 8; count++)
{
    ring = new Circle(160, 160, radius);
    ring.setFill(ringColor);
    root.getChildren().add(ring);

    if (ringColor.equals(Color.BLACK))
        ringColor = Color.WHITE;
    else
        ringColor = Color.BLACK;

    radius = radius - 20;
}

ring.setFill(Color.RED);

Scene scene = new Scene(root, 320, 320, Color.CYAN);

primaryStage.setTitle("Bullseye");
primaryStage.setScene(scene);
primaryStage.show();
import java.util.Random;
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;
import javafx.stage.Stage;

//****************************************************************************
// Boxes.java        Author: Lewis/Loftus
//
// Demonstrates the use of loops and conditionals to draw.
//****************************************************************************

class Boxes extends Application
{
    // Displays multiple rectangles with random width and height in
    // random locations. Narrow and short boxes are highlighted with
    // a fill color.
    public void start(Stage primaryStage)
    {
        Group root = new Group();
        Random gen = new Random();

        continue
    }
}
continue

for (int count = 1; count <= 50; count++)
{
    int x = gen.nextInt(350) + 1;
    int y = gen.nextInt(350) + 1;

    int width = gen.nextInt(50) + 1;
    int height = gen.nextInt(50) + 1;

    Color fill = null;
    if (width < 10)
        fill = Color.YELLOW;
    else if (height < 10)
        fill = Color.GREEN;

    Rectangle box = new Rectangle(x, y, width, height);
    box.setStroke(Color.WHITE);
    box.setFill(fill);

    root.getChildren().add(box);
}

continue
Scene scene = new Scene(root, 400, 400, Color.BLACK);

primaryStage.setTitle("Boxes");
primaryStage.setScene(scene);
primaryStage.show();
}
```java
Scene scene = new Scene(root, 400, 400, Color.BLACK);
primaryStage.setTitle("Boxes");
primaryStage.setScene(scene);
primaryStage.show();
```
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:

```java
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%:

```java
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):

  ```java
  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:

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

  ![Sheared Image](image.png)
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
import javafx.scene.Group;
import javafx.scene.paint.Color;
import javafx.scene.shape.Rectangle;

public class RobotFace extends Group
{
    public RobotFace()
    {
        Rectangle head = new Rectangle(5, 0, 100, 70);
        head.setFill(Color.SILVER);
        head.setArcHeight(10);
        head.setArcWidth(10);

        Rectangle ears = new Rectangle(0, 20, 110, 30);
        ears.setFill(Color.DARKBLUE);
        ears.setArcHeight(10);
        ears.setArcWidth(10);
    }
}
Rectangle eye1 = new Rectangle(25, 15, 20, 10);
    eye1.setFill(Color.GOLD);

Rectangle eye2 = new Rectangle(65, 15, 20, 10);
    eye2.setFill(Color.GOLD);

Rectangle nose = new Rectangle(52, 25, 6, 15);
    nose.setFill(Color.BLACK);

Rectangle mouth = new Rectangle(35, 45, 40, 10);
    mouth.setFill(Color.RED);

getchildren().addAll(ears, head, eye1, eye2, nose, mouth);
Transformations on Groups

• If presented as defined, the robot face would be displayed in the upper left corner:
import javafx.application.Application;
import javafx.scene.Group;
import javafx.scene.Scene;
import javafx.scene.paint.Color;
import javafx.stage.Stage;

public class Robots extends Application {
    // Displays three robot faces, applying various transformations.
    public void start(Stage primaryStage) {
        RobotFace robot1 = new RobotFace();
        robot1.setTranslateX(70);
        robot1.setTranslateY(40);

        RobotFace robot2 = new RobotFace();
        robot2.setTranslateX(300);
        robot2.setTranslateY(40);
        robot2.setRotate(20);
    }
}

continue
continue

RobotFace robot3 = new RobotFace();
robot3.setTranslateX(200);
robot3.setTranslateY(200);
robot3.setScaleX(2.5);
robot3.setScaleY(2.5);

Group root = new Group(robot1, robot2, robot3);

Scene scene = new Scene(root, 500, 380, Color.WHITE);

primaryStage.setTitle("Robots");
primaryStage.setScene(scene);
primaryStage.show();
}
continue

RobotFace
robot3 = new RobotFace();
robot3.setTranslateX(200);
robot3.setTranslateY(200);
robot3.setScaleX(2.5);
robot3.setScaleY(2.5);

Group root = new Group(robot1, robot2, robot3);

Scene scene = new Scene(root, 500, 380, Color.WHITE);

primaryStage.setTitle("Robots");
primaryStage.setScene(scene);
primaryStage.show();
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
Exercise

• What output is produced by the following code fragment?

```java
for (int val = 200; val >= 0; val -= 1)
    if (val % 4 != 0)
        System.out.println(val);
```
Solution

• What output is produced by the following code fragment?
  ```java
  for (int val = 200; val >= 0; val -- = 1)
      if (val % 4 != 0)
          System.out.println(val);
  ```

• The output produced is all values from 200 down to 1, except those that are evenly divisible by 4:
  • 199
  • 198
  • 197
  • 195
  • and so on until...
  • 5
  • 3
  • 2
  • 1
Exercise

• When the following loop stops?

```java
for (double x = 0.0; x != 10.0 && x < 11; x += 0.1)
    System.out.println(x);
```
Exercise

• In mathematics, the $n$-th **harmonic number** is the sum of the reciprocals of the first $n$ natural numbers:

$$ H_n = 1 + \frac{1}{2} + \frac{1}{3} + \frac{1}{4} \ldots + \frac{1}{n} $$

• Implement a method (of the class Harmonic) that computes the $n$-th harmonic number ($n$ is the int parameter of the method)
public class Harmonic {

    public double harmonic(int N) {
        double sum = 0.0;
        for (int i = 1; i <= N; i++)
            sum += 1.0 / i;
        return sum;
    }

    public static void main(String[] args) {

        Harmonic h = new Harmonic();
        for (int i = 1; i <= 20; ++i)
            System.out.println(h.harmonic(i));
    }
}
Exercise

• $x_1 = y$
• $x_{n+1} = 1/2*(x_n + y/x_n)$
• $\lim_{n \to \infty} x_n = \sqrt{y}$

• Using the information above write a (static) method that computes the square root of a double
• Hint: implement a loop where a variable $x$ is updated with the formula above. The loop is terminated when $x_n * x_n$ is enough close to $y$
Exercise

```java
public static double sqrt(double y) {
    if (y < 0)
        return Double.NaN;
    double err = 1e-15;
    double x = y;
    while (Math.abs(x*x - y) > err)
        x = (y / x + x) / 2.0;
    return x;
}
```
Exercise

• Implement a simple primality test that, given an integer, returns true if this number is larger than 1 and is not divisible by any number smaller than itself and larger than 1 (and false otherwise).
public static boolean isPrime(int N) {
    if (N < 2)
        return false;
    for (int i = 2; i * i <= N; i++)
        if (N % i == 0)
            return false;
    return true;
}

for (int i = 2; i < N; i++) is also OK
Exercise

• Write a method that convert a positive integer \( n \) into a string containing the binary representation of the integer \( n \)
• The method has an \( \text{int} \) parameter and return a \( \text{String} \)

Convert \( 14_{10} \) to Base 2

\[
\begin{array}{c|c}
2 & 14 \\
2 & 7 & \text{with remainder 0} & \leftarrow \text{Least significant digit} \\
2 & 3 & \text{with remainder 1} \\
2 & 1 & \text{with remainder 1} \\
0 & & \text{with remainder 1} & \leftarrow \text{Most significant digit}
\end{array}
\]

Read remainders in reverse order to obtain

\[14_{10} = 1110_2\]
Solution

```java
public class Integer2BinaryString {

    public static String convert(int n) {
        String s = "";

        for (int m = n; m > 0; m /= 2) {
            s = m%2 + s;
        }

        return s;
    }

    public static void main (String[] args) {
        System.out.println(convert(10));
    }
}
```