Java Class 15

Java: an island in Indonesia to the south of Borneo; one of the world's most densely populated regions in Asia made up of an archipelago including more than 13,000 islands; achieved independence from the Netherlands in 1945; the principal oil producer in the Far East and Pacific regions; the capital of Indonesia, Jakarta is located on the island of Java.

The switch Statement

The general syntax of a `switch` statement is:

switch and case are reserved words

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

}
```

If *expression* matches *value2*, control jumps to here

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

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

An example of a switch statement:

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

The switch Statement

The type of a `switch` expression must be integers, characters, enumerated types, or , as of Java 7, 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

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 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, the "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.");
```

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

The do Statement

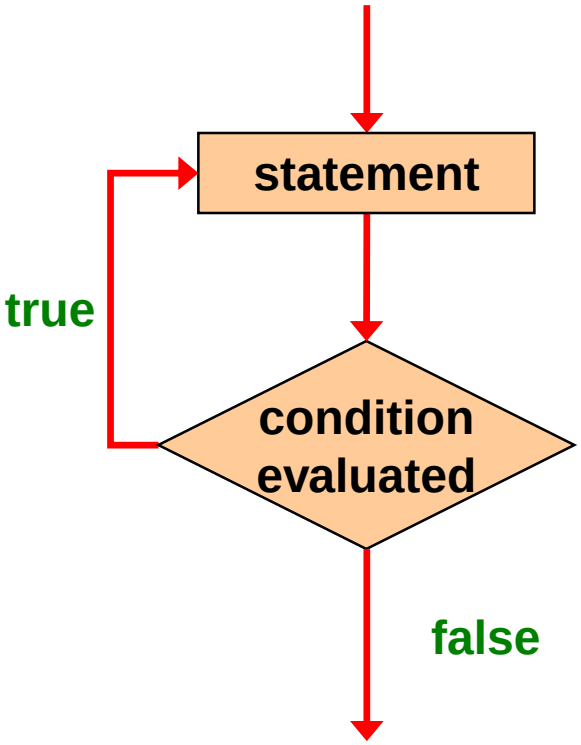
A *do statement* has the following syntax:

```
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

```java
//*******************************************************************
//  ReverseNumber.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a do loop.
//*******************************************************************

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

continue

```
        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 " + reverse);
    }
}
```

continue

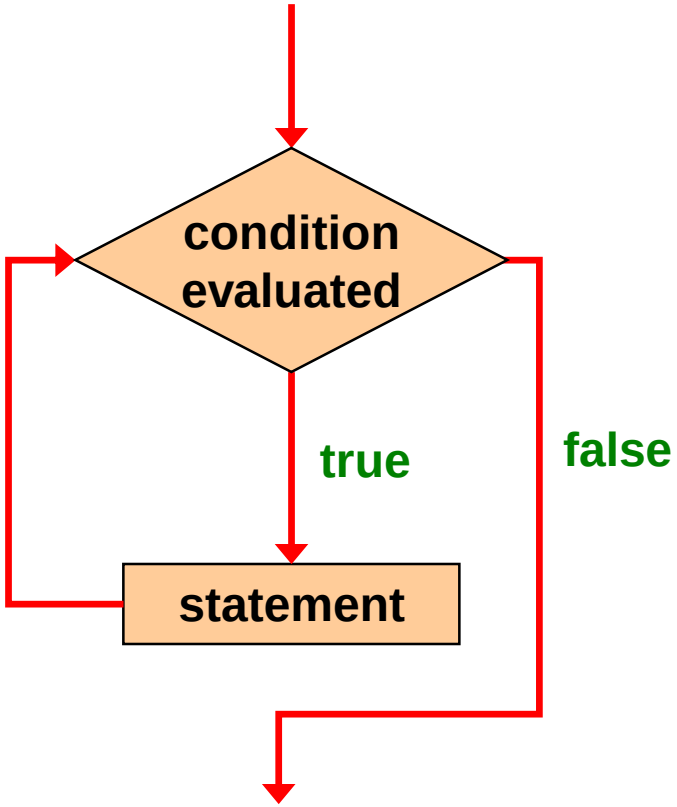
```
        System.out.
        number = sc

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

        System.out.println ("That number reversed is " + reverse);
    }
}
```

Sample Run

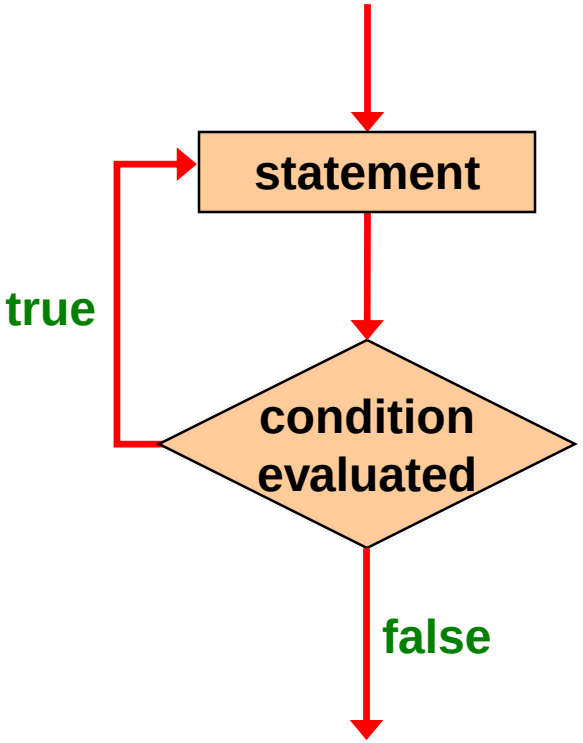
```
Enter a positive integer: 2896
That number reversed is 6982
```

Comparing while and do Loops

The while Loop



The do Loop



The for Statement

A *for statement* has the following syntax:

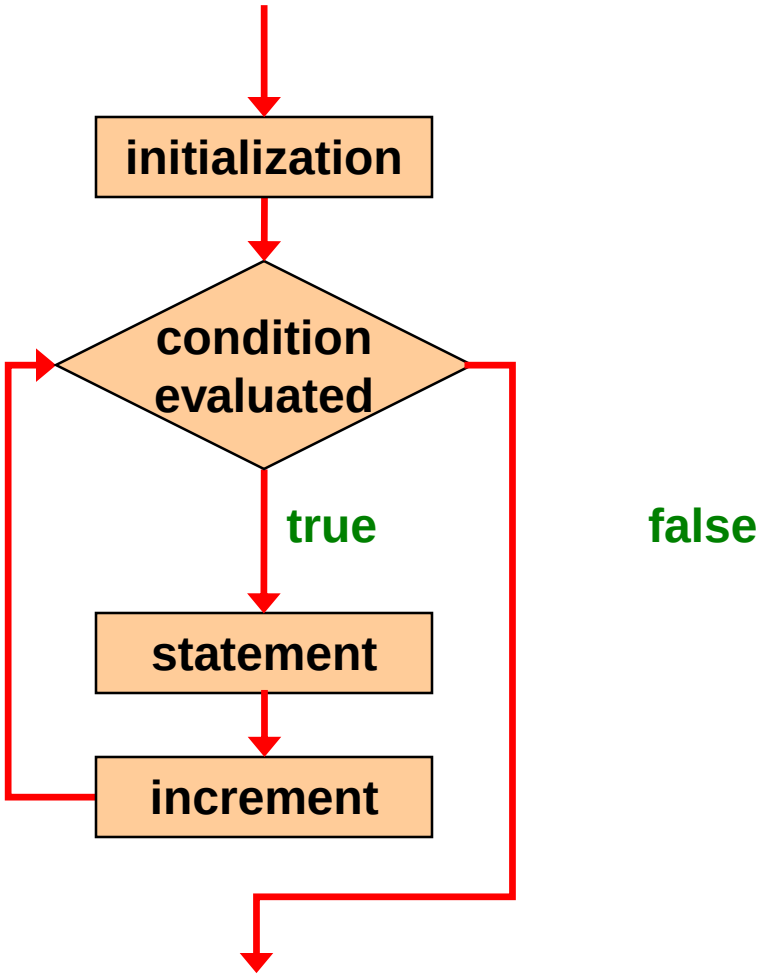
The *initialization* is executed once before the loop begins

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

```java
//*****************************************************************
//  Multiples.java       Author: Lewis/Loftus
//
//  Demonstrates the use of a for loop.
//*****************************************************************

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

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();
        }
    }
}
```

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
```

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++;
System.out.println (count);
```

```java
//*******************************************************************
//   Stars.java        Author: Lewis/Loftus
//
//   Demonstrates the use of nested for loops.
//*******************************************************************

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();
      }
   }
}
```

```
//*********************************************************************
//   Stars.java        Auth                                   s
//
//   Demonstrates the use                                    oops.
//*********************************************************************

public class Stars
{
   //-----------------------------------------------------------------
   //  Prints a triangle                                    erisk (star) characters.
   //-----------------------------------------------------------------
   public static void mai                                   s)
   {
      final int MAX_ROWS

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

         System.out.println();
      }
   }
}
```

Output

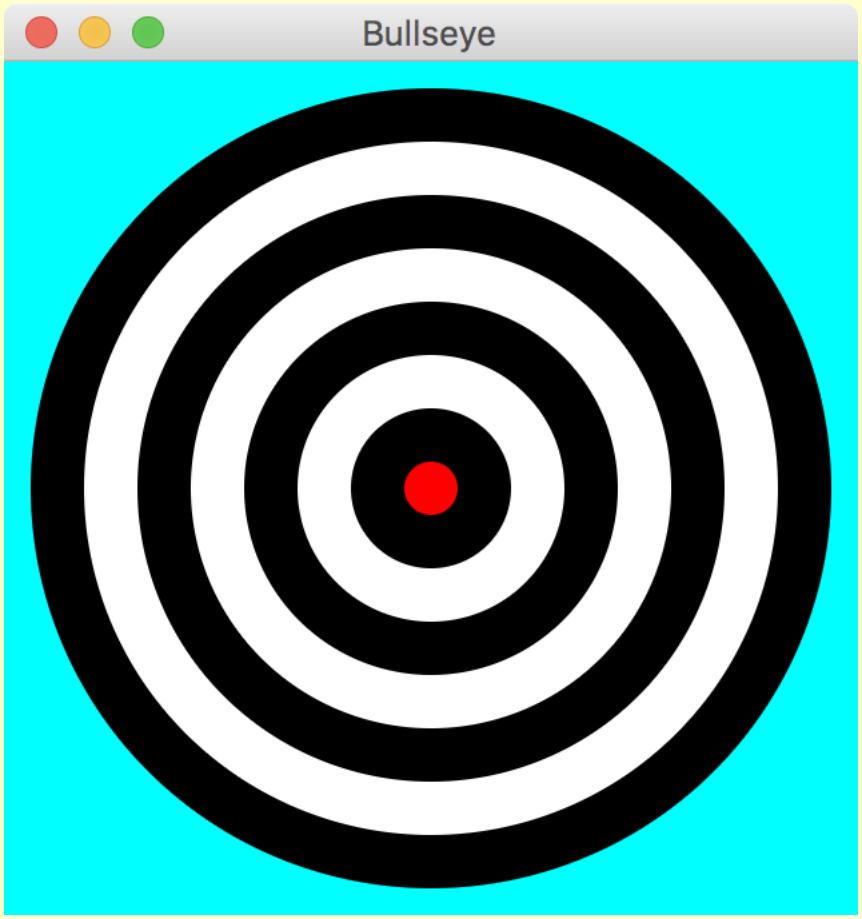
```
*
**
***
****
*****
******
*******
********
*********
**********
```

More Graphics

Conditionals and loops enhance our ability to generate interesting graphics

continue

```
        for (int
        {
            ring
            ring
            root

            if (

            else


            radi
        }

        ring.set

        Scene sc                                                    ;

        primaryS
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```



```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.
//*******************************************************************

public 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;
```

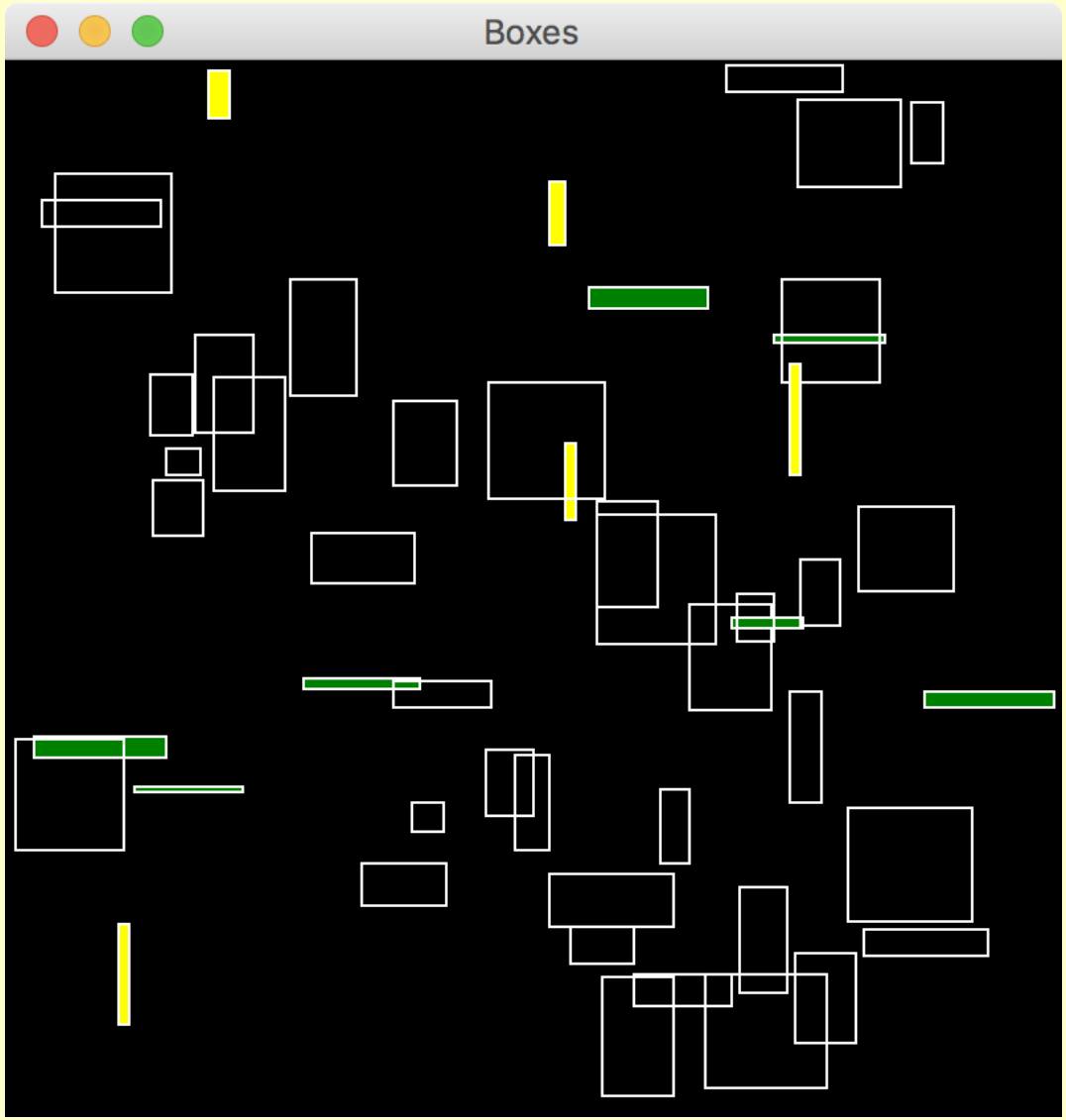
continue

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

```
        Scen

        prim
        prim
        prim
    }
}
```



```java
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.
//*******************************************************************

public 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

continue

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

        primaryStage.setTitle("Boxes");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
```

Group Exercises

Ex: 6.1

Ex: 6.2

Ex: 6.5

Ex: 6.7

Assignment for Class 16

Review GradeReport, ReverseNumber, Multiples, Stars, Bullseye, Boxes

Read 5.8, 5.9, 5.10. 6.6