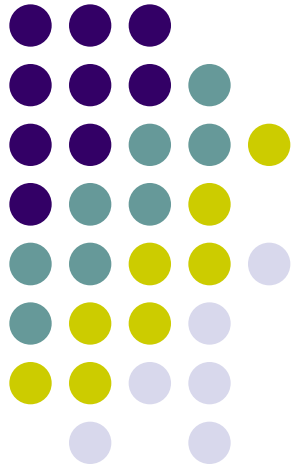
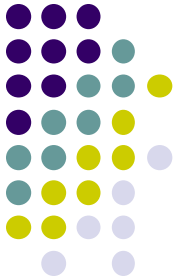
CSCI 2010
Principles of Computer Science

Basic Java Programming

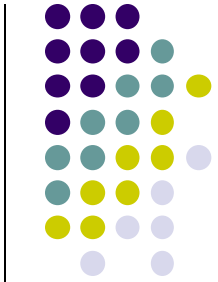


Today's Topics



- Using Classes and Objects
  - object creation and object references
  - the `String` class and its methods
  - the Java standard class library
  - the `Random` and `Math` classes
  - formatting output
  - Basic decision making if and switch
  - Review while, do, and for loops

Creating Objects



- Class name can be used as a type to declare an *object reference variable*

  String title;

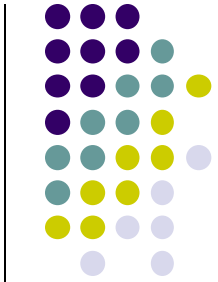
  --The object itself must be created separately

- Generally, we use the `new` operator to create an object

```
title = new String ("Java Software Solutions");
```

This calls the String *constructor*, which is a special method that sets up the object

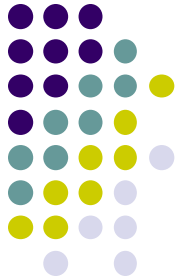
Invoking Methods



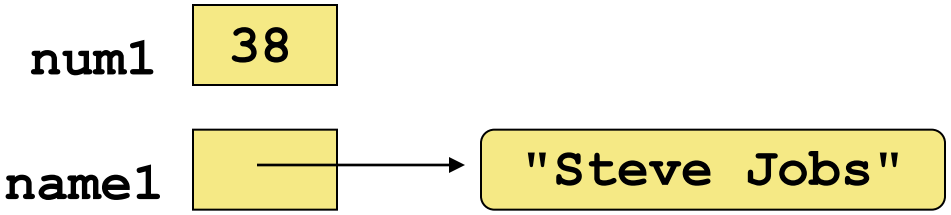
- Once an object has been instantiated, we can use the *dot operator* to invoke its methods

```
count = title.length()
```

Object References



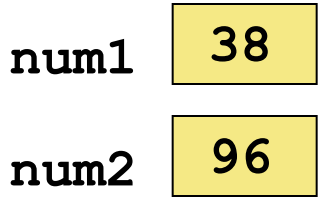
- An object reference can be thought of as a pointer to the location of the object



Object References- Assignment

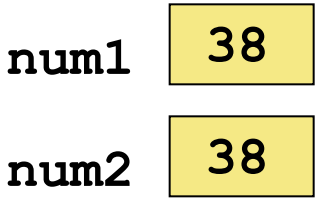
- For primitive types

Before:



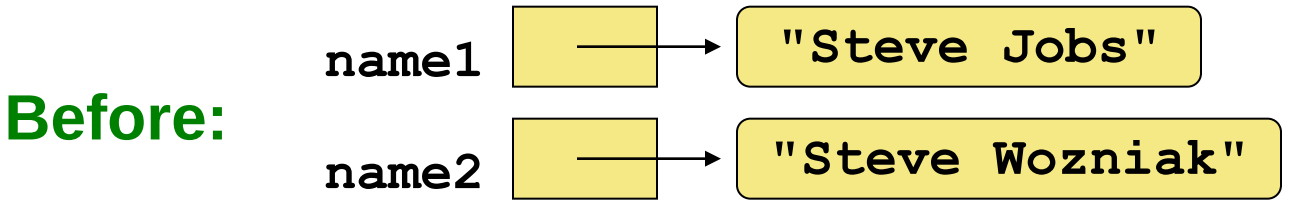

```
num2 = num1;
```

After:

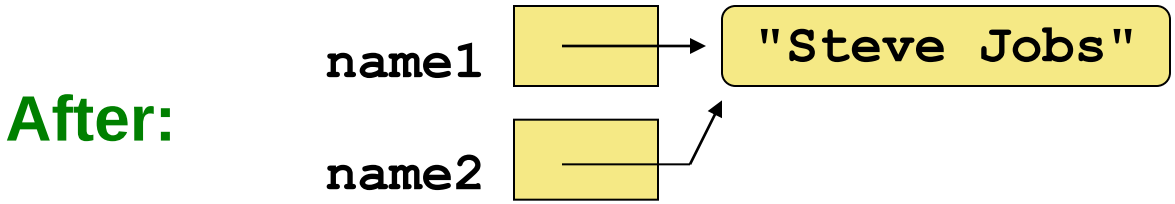


Object References- Assignment

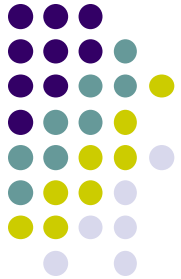
- For object references, assignment copies the address




```
name2 = name1;
```

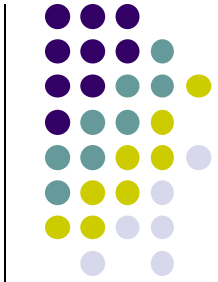


Garbage Collection



- When an object no longer has any valid references to it, it is called *garbage*
- Java performs *automatic garbage collection* periodically, returning an object's memory to the system for future use
- In other languages, the programmer is responsible for performing garbage collection

The `String` Class

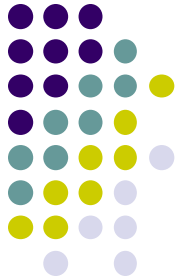


- We don't have to use the `new` operator to create a `String` object

```
title = "Java Software Solutions";
```

- Once a `String` object has been created, neither its value nor its length can be changed
- Thus we say that an object of the `String` class is *immutable*

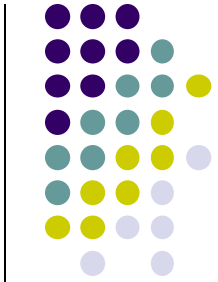
The `String` Class - Indexes



- How to refer to a particular character within a string?
- This can be done by specifying the character's numeric *index*
- The indexes begin at zero in each string
- In the string `"Hello"`, the character `'H'` is at index 0 and the `'o'` is at index 4

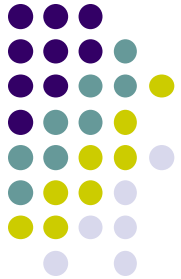
```
//********************************************************************
//  StringMutation.java       Java Foundations
//
//  Demonstrates the use of the String class and its methods.
//********************************************************************

public class StringMutation
{
   //-----------------------------------------------------------------
   //  Prints a string and various mutations of it.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      String phrase = "Change is inevitable";
      String mutation1, mutation2, mutation3, mutation4;

      System.out.println ("Original string: \"" + phrase + "\"");
      System.out.println ("Length of string: " + phrase.length());

      mutation1 = phrase.concat (", except from vending machines.");
      mutation2 = mutation1.toUpperCase();
      mutation3 = mutation2.replace ('E', 'X');
      mutation4 = mutation3.substring (3, 30);
 // Print each mutated string
      System.out.println ("Mutation #1: " + mutation1);
      System.out.println ("Mutation #2: " + mutation2);
      System.out.println ("Mutation #3: " + mutation3);
      System.out.println ("Mutation #4: " + mutation4);

      System.out.println ("Mutated length: " + mutation4.length());
   }
}
```

Class Libraries



- A *class library* is a collection of classes that we can use when developing programs
- The *Java standard class library* is part of any Java development environment

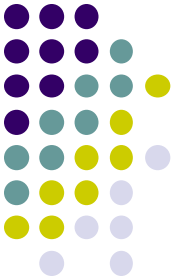
Packages



- The classes of the Java standard class library are organized into *packages*
- Some of the packages in the standard class library are

Package	Purpose
java.lang	General support
java.applet	Creating applets for the web
java.awt	Graphics and graphical user interfaces
javax.swing	Additional graphics capabilities
java.net	Network communication
java.util	Utilities
javax.xml.parsers	XML document processing

The `import` Declaration



- When you want to use a class from a package, you could use its *fully qualified name*

```
java.util.Scanner
```

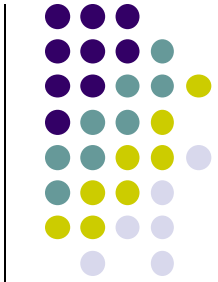
- Or you can *import* the class, and then use just the class name

```
import java.util.Scanner;
```

- To import all classes in a particular package, you can use the `*` wildcard character

```
import java.util.*;
```

The `import` Declaration



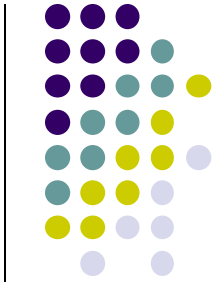
- All classes of the `java.lang` package are imported automatically into all programs
- It's as if all programs contain the following line

```
import java.lang.*;
```

- Why we must import the `Scanner` class?

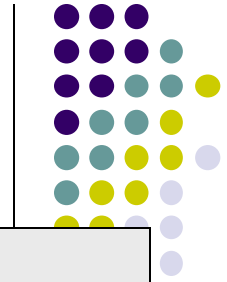
  The `Scanner` class, on the other hand, is part of the `java.util` package, and therefore must be imported

The `Random` Class



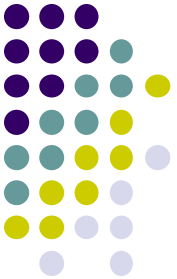
- The `Random` class is part of the `java.util` package
- It provides methods that generate pseudorandom numbers

RandomNumbers.java



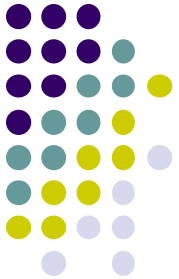
```
//********************************************************************
//  RandomNumbers.java       Java Foundations
//
//  Demonstrates the creation of pseudo-random numbers using the
//  Random class.
//********************************************************************

import java.util.Random;

public class RandomNumbers
{
   //-----------------------------------------------------------------
   //  Generates random numbers in various ranges.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Random generator = new Random();
      int num1;
      float num2;

      num1 = generator.nextInt();
      System.out.println ("A random integer: " + num1);

(more...)
```

RandomNumbers.java



```
        num1 = generator.nextInt(10);
        System.out.println ("From 0 to 9: " + num1);

        num1 = generator.nextInt(10) + 1;
        System.out.println ("From 1 to 10: " + num1);

        num1 = generator.nextInt(15) + 20;
        System.out.println ("From 20 to 34: " + num1);

        num1 = generator.nextInt(20) - 10;
        System.out.println ("From -10 to 9: " + num1);

        num2 = generator.nextFloat();
        System.out.println ("A random float (between 0-1): " + num2);

        num2 = generator.nextFloat() * 6;  // 0.0 to 5.999999
        num1 = (int)num2 + 1;
        System.out.println ("From 1 to 6: " + num1);
    }
}
```

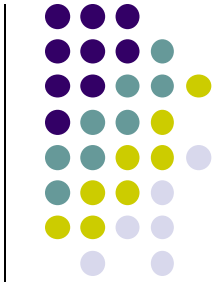
The `Math` Class



- The `Math` class is part of the `java.lang` package
- The `Math` class contains methods that perform various mathematical functions
- These include
  - absolute value
  - square root
  - exponentiation
  - trigonometric functions

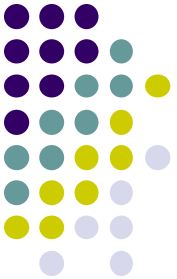
```
//********************************************************************
//  Quadratic.java       Java Foundations
//
//  Determines the roots of a quadratic equation.
//********************************************************************
import java.util.Scanner;

public class Quadratic
{
   public static void main (String[] args)
   {
      int a, b, c;  // ax^2 + bx + c
      double discriminant, root1, root2;

      Scanner scan = new Scanner (System.in);

      System.out.print ("Enter the coefficient of x squared: ");
      a = scan.nextInt();
 System.out.print ("Enter the coefficient of x: ");
      b = scan.nextInt();

      System.out.print ("Enter the constant: ");
      c = scan.nextInt();

      // Use the quadratic formula to compute the roots.
      // Assumes a positive discriminant.

      discriminant = Math.pow(b, 2) - (4 * a * c);
      root1 = ((-1 * b) + Math.sqrt(discriminant)) / (2 * a);
      root2 = ((-1 * b) - Math.sqrt(discriminant)) / (2 * a);

      System.out.println ("Root #1: " + root1);
      System.out.println ("Root #2: " + root2);
   }
}
```

Formatting Output



- The Java standard class library contains classes that provide formatting capabilities
- The `NumberFormat` class allows you to format values as currency or percentages
- The `DecimalFormat` class allows you to format values based on a pattern
- Both are part of the `java.text` package

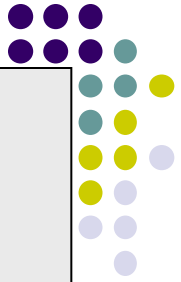
Formatting Output



- The `NumberFormat` class has static methods that return a formatter object

```
getCurrencyInstance()

getPercentInstance()
```

- Each formatter object has a method called `format` that returns a string with the specified information in the appropriate format



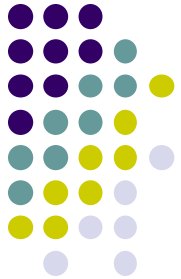
```
//********************************************************************
//  Purchase.java       Java Foundations
//
//  Demonstrates the use of the NumberFormat class to format output.
//********************************************************************

import java.util.Scanner;
import java.text.NumberFormat;

public class Purchase
{
   //-----------------------------------------------------------------
   //  Calculates the final price of a purchased item using values
   //  entered by the user.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      final double TAX_RATE = 0.06;  // 6% sales tax

      int quantity;
      double subtotal, tax, totalCost, unitPrice;

      Scanner scan = new Scanner (System.in);
```

(more…)

Purchase.java

```
        NumberFormat fmt1 = NumberFormat.getCurrencyInstance();
        NumberFormat fmt2 = NumberFormat.getPercentInstance();

        System.out.print ("Enter the quantity: ");
        quantity = scan.nextInt();

        System.out.print ("Enter the unit price: ");
        unitPrice = scan.nextDouble();

        subtotal = quantity * unitPrice;
        tax = subtotal * TAX_RATE;
        totalCost = subtotal + tax;

        // Print output with appropriate formatting
        System.out.println ("Subtotal: " + fmt1.format(subtotal));
        System.out.println ("Tax: " + fmt1.format(tax) + " at "
                            + fmt2.format(TAX_RATE));
        System.out.println ("Total: " + fmt1.format(totalCost));
    }
}
```

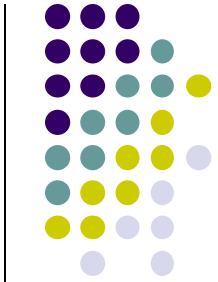
Formatting Output



- The `DecimalFormat` class can be used to format a floating point value in various ways
- For example, you can specify that the number should be truncated to three decimal places

```
//********************************************************************
//  CircleStats.java       Java Foundations
//
//  Calculates the area and circumference of a circle given its radius.
//********************************************************************

import java.util.Scanner;
import java.text.DecimalFormat;

public class CircleStats
{
   public static void main (String[] args)
   {
      int radius;
      double area, circumference;

      Scanner scan = new Scanner (System.in);

      System.out.print ("Enter the circle's radius: ");
      radius = scan.nextInt();

      area = Math.PI * Math.pow(radius, 2);
      circumference = 2 * Math.PI * radius;

      // Round the output to three decimal places
      DecimalFormat fmt = new DecimalFormat ("0.###");

      System.out.println ("The circle's area: " + fmt.format(area));
      System.out.println ("The circle's circumference: "
                          + fmt.format(circumference));
   }
```

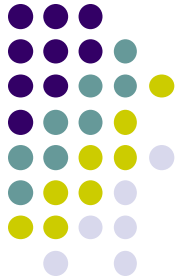
Enumerated Types



- An enumerated type establishes all possible values for a variable of that type
- The values are identifiers of your own choosing
- The following declaration creates an enumerated type called `Season`

```
enum Season {winter, spring, summer, fall};
```

Enumerated Types



- Once a type is defined, a variable of that type can be declared

```
Season time;
```

and it can be assigned a value

```
time = Season.fall;
```

- The values are specified through the name of the type

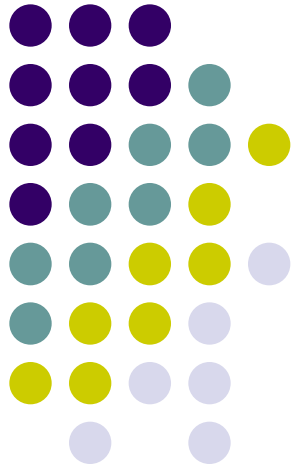
```
//********************************************************************
//  IceCream.java       Java Foundations
//
//  Demonstrates the use of enumerated types.
//********************************************************************


public class IceCream
{
   enum Flavor {vanilla, chocolate, strawberry, fudgeRipple, coffee,
                rockyRoad, mintChocolateChip, cookieDough}

   //-----------------------------------------------------------------
   //  Creates and uses variables of the Flavor type.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Flavor cone1, cone2, cone3;

      cone1 = Flavor.rockyRoad;
      cone2 = Flavor.chocolate;

(more…)
```

```
//********************************************************************
//  IceCream.java       Java Foundations
//  Creates and uses variables of the Flavor type.
//********************************************************************
public class IceCream
{
   enum Flavor {vanilla, chocolate, strawberry, fudgeRipple, coffee,
                rockyRoad, mintChocolateChip, cookieDough}

   public static void main (String[] args)
   {
      Flavor cone1, cone2, cone3;

      cone1 = Flavor.rockyRoad;
      cone2 = Flavor.chocolate;

      System.out.println ("cone1 value: " + cone1);
      System.out.println ("cone1 ordinal: " + cone1.ordinal());
      System.out.println ("cone1 name: " + cone1.name());

      System.out.println ();
      System.out.println ("cone2 value: " + cone2);
      System.out.println ("cone2 ordinal: " + cone2.ordinal());
      System.out.println ("cone2 name: " + cone2.name());

      cone3 = cone1;

      System.out.println ();
      System.out.println ("cone3 value: " + cone3);
      System.out.println ("cone3 ordinal: " + cone3.ordinal());
      System.out.println ("cone3 name: " + cone3.name());
   }
}
```

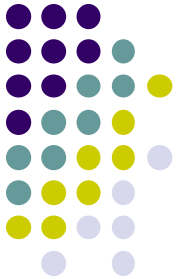
Procedure Programming in Java

Self Study:

See text for details on looping (for, while, do), branching (if-else, switch), etc.

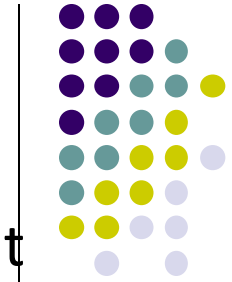


The `switch` Statement



- 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

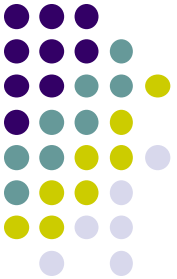
The `switch` Statement



A *break statement* is used as the last statement in each case. It causes control to transfer to the end of the switch statement. A switch statement can have an optional *default case*

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

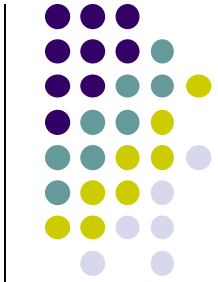
The `switch` Statement



- An example of a switch statement

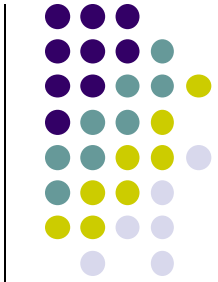
```
switch (option)
{
    case 'A':
        aCount++;
        break;
    case 'B':
        bCount++;
        break;
    case 'C':
        cCount++;
        break;
}
```

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



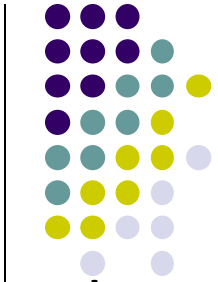
- The expression of a switch statement must result in an *integral type*, meaning an integer (byte, short, int, long) or a char
- It cannot be a boolean value or a floating point value (float or double)
- The implicit boolean condition in a switch statement is equality

```
//********************************************************************
//  GradeReport.java
//
//  Demonstrates the use of a switch statement.
//********************************************************************

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;
```

(more…)

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

      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.print ("below average. Please see the ");
            System.out.println ("instructor for assistance.");
            break;
         default:
            System.out.println ("not passing.");
      }
   }
}
```

Repetition Statements



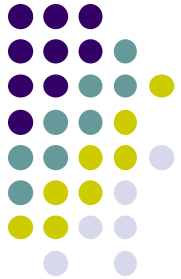
- *Repetition statements* allow us to execute a statement multiple times, aka *loops*
- Java has three kinds of repetition statements:
  - the *while loop*
  - the *do loop*
  - the *for loop*

```
//********************************************************************
//  Average.java       Java Foundations
//
//  Demonstrates the use of a while loop, a sentinel value, and a
//  running sum.
//********************************************************************

import java.text.DecimalFormat;
import java.util.Scanner;

public class Average
{
   //-----------------------------------------------------------------
   //  Computes the average of a set of values entered by the user.
   //  The running sum is printed as the numbers are entered.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      int sum = 0, value, count = 0;
      double average;

      Scanner scan = new Scanner (System.in);

      System.out.print ("Enter an integer (0 to quit): ");
```

(more…)

```
      value = scan.nextInt();

      while (value != 0)  // sentinel value of 0 to terminate loop
      {
         count++;

         sum += value;
         System.out.println ("The sum so far is " + sum);

         System.out.print ("Enter an integer (0 to quit): ");
         value = scan.nextInt();
      }

      System.out.println ();

      if (count == 0)
         System.out.println ("No values were entered.");
      else
      {
         average = (double)sum / count;

         DecimalFormat fmt = new DecimalFormat ("0.###");
         System.out.println ("The average is " + fmt.format(average));
      }
   }
}
```

The `do` Statement

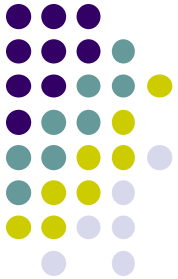


- A *do statement* has the following syntax

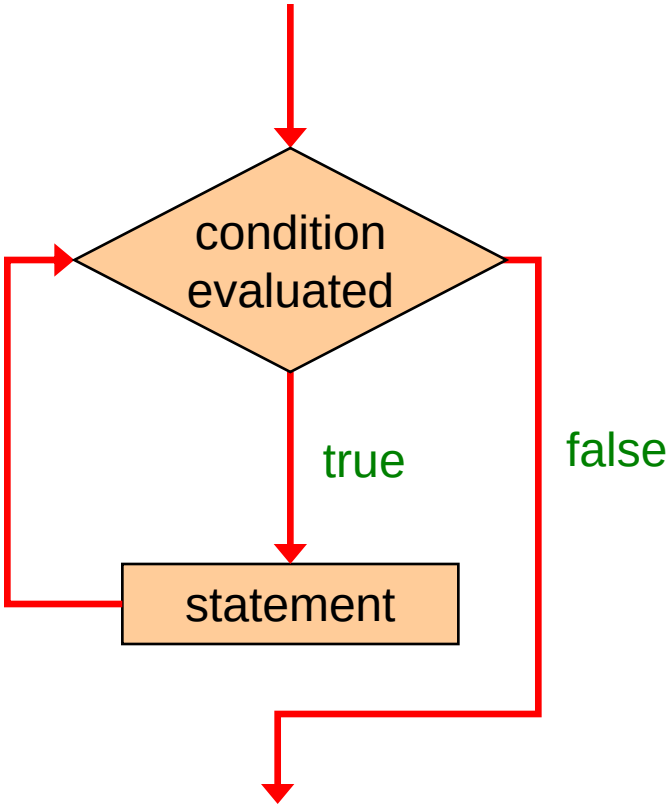
- The `statement` is executed once initially, and then the `condition` is evaluated
- The statement is executed repeatedly until the condition becomes false
- The body of a `do` loop executes at least once

```
//********************************************************************
//  ReverseNumber.java
//  This program will reverses the digits of an integer mathematically.
//  Demonstrates the use of a do loop.
//********************************************************************
import java.util.Scanner;

public class ReverseNumber
{
   public static void main (String[] args)
   {
      int number, lastDigit, reverse = 0;

      Scanner scan = new Scanner (System.in);
      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);
   }
}
```

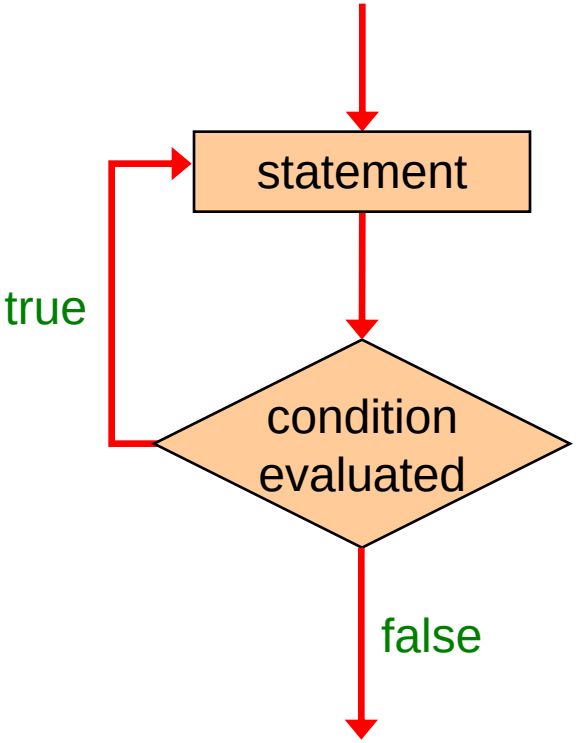
Comparing `while` and `do`



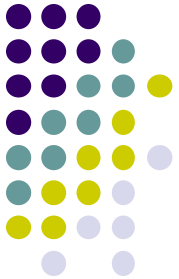
The while Loop



The do Loop



The `for` Statement



- A *for statement* has the following syntax

The *initialization* is executed once before the loop begins

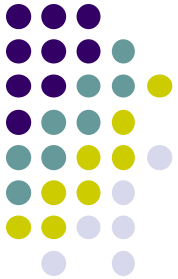
The *statement* is executed until the *condition* becomes false

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

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

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

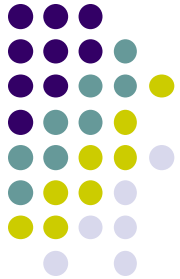
Summary



Today's focused on

- object creation and object references
- the `String` class and its methods
- the Java standard class library
- the `Random` and `Math` classes
- formatting output
- Basic decision making if and switch
- Review while, do, and for loops

READING MATERIAL



- TEXT
  - Chapter 3
    Classes and objects
  - Chapter 4
    Basic syntax of procedure programming using Java
  - Chapter 5
    Writing Java classes and simple design principles of writing Java classes.
- Online
  - http://java.sun.com/docs/books/tutorial/java/index.html
    - Language Basics
    - Classes and Objects
    - Numbers and Strings