The this Reference

- The `this` reference, used inside a method, refers to the object through which the method is being executed
- Suppose the `this` reference is used inside a method called `tryMe`, which is invoked as follows:

```
obj1.tryMe();
obj2.tryMe();
```

- In the first invocation, the `this` reference refers to `obj1`; in the second it refers to `obj2`

The this reference

- The `this` reference can be used to distinguish the instance variables of a class from corresponding method parameters with the same names

```
public Account (String name, long acctNumber,
                   double balance)
{
   this.name = name;
   this.acctNumber = acctNumber;
   this.balance = balance;
}
```

Dependency

- The following example defines a class called `RationalNumber`

- A rational number is a value that can be represented as the ratio of two integers

- Several methods of the `RationalNumber` class accept another `RationalNumber` object as a parameter

- See `RationalTester.java`
- See `RationalNumber.java`

```java
//********************************************************************
//  RationalTester.java       Author: Lewis/Loftus
//
//  Driver to exercise the use of multiple Rational objects.
//********************************************************************

public class RationalTester
{
   //-----------------------------------------------------------------
   //  Creates some rational number objects and performs various
   //  operations on them.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      RationalNumber r1 = new RationalNumber (6, 8);
      RationalNumber r2 = new RationalNumber (1, 3);
      RationalNumber r3, r4, r5, r6, r7;

      System.out.println ("First rational number: " + r1);
      System.out.println ("Second rational number: " + r2);
```

continue

continue

```
      if (r1.isLike(r2))
         System.out.println ("r1 and r2 are equal.");
      else
         System.out.println ("r1 and r2 are NOT equal.");

      r3 = r1.reciprocal();
      System.out.println ("The reciprocal of r1 is: " + r3);

      r4 = r1.add(r2);
      r5 = r1.subtract(r2);
      r6 = r1.multiply(r2);
      r7 = r1.divide(r2);

      System.out.println ("r1 + r2: " + r4);
      System.out.println ("r1 - r2: " + r5);
      System.out.println ("r1 * r2: " + r6);
      System.out.println ("r1 / r2: " + r7);
   }
}
```

continue

```
      if (r1.isLike
         System.out
      else
         System.out                                              ");

      r3 = r1.recip
      System.out.pr                                               r3);

      r4 = r1.add(r
      r5 = r1.subtr
      r6 = r1.multiply(r2);
      r7 = r1.divide(r2);

      System.out.println ("r1 + r2: " + r4);
      System.out.println ("r1 - r2: " + r5);
      System.out.println ("r1 * r2: " + r6);
      System.out.println ("r1 / r2: " + r7);
   }
}
```

Output

```
First rational number: 3/4
Second rational number: 1/3
r1 and r2 are NOT equal.
The reciprocal of r1 is: 4/3
r1 + r2: 13/12
r1 - r2: 5/12
r1 * r2: 1/4
r1 / r2: 9/4
```

```java
//********************************************************************
//  RationalNumber.java       Author: Lewis/Loftus
//
//  Represents one rational number with a numerator and denominator.
//********************************************************************

public class RationalNumber
{
   private int numerator, denominator;

   //-----------------------------------------------------------------
   //  Constructor: Sets up the rational number by ensuring a nonzero
   //  denominator and making only the numerator signed.
   //-----------------------------------------------------------------
   public RationalNumber (int numer, int denom)
   {
      if (denom == 0)
         denom = 1;

      // Make the numerator "store" the sign
      if (denom < 0)
      {
         numer = numer * -1;
         denom = denom * -1;
      }
```

continue

continue

```
      numerator = numer;
      denominator = denom;

      reduce();
   }

   //-----------------------------------------------------------------
   //  Returns the numerator of this rational number.
   //-----------------------------------------------------------------
   public int getNumerator ()
   {
      return numerator;
   }

   //-----------------------------------------------------------------
   //  Returns the denominator of this rational number.
   //-----------------------------------------------------------------
   public int getDenominator ()
   {
      return denominator;
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Returns the reciprocal of this rational number.
   //-----------------------------------------------------------------
   public RationalNumber reciprocal ()
   {
      return new RationalNumber (denominator, numerator);
   }

   //-----------------------------------------------------------------
   //  Adds this rational number to the one passed as a parameter.
   //  A common denominator is found by multiplying the individual
   //  denominators.
   //-----------------------------------------------------------------
   public RationalNumber add (RationalNumber op2)
   {
      int commonDenominator = denominator * op2.getDenominator();
      int numerator1 = numerator * op2.getDenominator();
      int numerator2 = op2.getNumerator() * denominator;
      int sum = numerator1 + numerator2;

      return new RationalNumber (sum, commonDenominator);
   }
```

continue

```
continue

   //-----------------------------------------------------------------
   //  Subtracts the rational number passed as a parameter from this
   //  rational number.
   //-----------------------------------------------------------------
   public RationalNumber subtract (RationalNumber op2)
   {
      int commonDenominator = denominator * op2.getDenominator();
      int numerator1 = numerator * op2.getDenominator();
      int numerator2 = op2.getNumerator() * denominator;
      int difference = numerator1 - numerator2;

      return new RationalNumber (difference, commonDenominator);
   }

   //-----------------------------------------------------------------
   //  Multiplies this rational number by the one passed as a
   //  parameter.
   //-----------------------------------------------------------------
   public RationalNumber multiply (RationalNumber op2)
   {
      int numer = numerator * op2.getNumerator();
      int denom = denominator * op2.getDenominator();

      return new RationalNumber (numer, denom);
   }

continue
```

continue

```
   //-----------------------------------------------------------------
   //  Divides this rational number by the one passed as a parameter
   //  by multiplying by the reciprocal of the second rational.
   //-----------------------------------------------------------------
   public RationalNumber divide (RationalNumber op2)
   {
      return multiply (op2.reciprocal());
   }

   //-----------------------------------------------------------------
   //  Determines if this rational number is equal to the one passed
   //  as a parameter. Assumes they are both reduced.
   //-----------------------------------------------------------------
   public boolean isLike (RationalNumber op2)
   {
      return ( numerator == op2.getNumerator() &&
               denominator == op2.getDenominator() );
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Returns this rational number as a string.
   //-----------------------------------------------------------------
   public String toString ()
   {
      String result;
      if (numerator == 0)
         result = "0";
      else
         if (denominator == 1)
            result = numerator + "";
         else
            result = numerator + "/" + denominator;
      return result;
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Reduces this rational number by dividing both the numerator
   //  and the denominator by their greatest common divisor.
   //-----------------------------------------------------------------
   private void reduce ()
   {
      if (numerator != 0)
      {
         int common = gcd (Math.abs(numerator), denominator);

         numerator = numerator / common;
         denominator = denominator / common;
      }
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Computes and returns the greatest common divisor of the two
   //  positive parameters. Uses Euclid's algorithm.
   //-----------------------------------------------------------------
   private int gcd (int num1, int num2)
   {
      while (num1 != num2)
         if (num1 > num2)
            num1 = num1 - num2;
         else
            num2 = num2 - num1;

      return num1;
   }
}
```

Aggregation

- In the following example, a `Student` object is composed, in part, of `Address` objects
- A student has an address (in fact each student has two addresses)
- See `Address.java`
- See `Student.java`
- See `StudentBody.java`

```java
//********************************************************************
//  StudentBody.java       Author: Lewis/Loftus
//
//  Demonstrates the use of an aggregate class.
//********************************************************************

public class StudentBody
{
   //-----------------------------------------------------------------
   //  Creates some Address and Student objects and prints them.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Address school = new Address ("800 Lancaster Ave.", "Villanova",
                                    "PA", 19085);
      Address jHome = new Address ("21 Jump Street", "Lynchburg",
                                   "VA", 24551);
      Student john = new Student ("John", "Smith", jHome, school);

      Address mHome = new Address ("123 Main Street", "Euclid", "OH",
                                   44132);
      Student marsha = new Student ("Marsha", "Jones", mHome, school);

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

```
//*********************                           **********************
//  StudentBody.java
//
//  Demonstrates the
//*********************                           **********************

public class StudentB
{
   //------------------                           ----------------------
   //  Creates some A                             and prints them.
   //------------------                           ----------------------
   public static void
   {
      Address school =                            er Ave.", "Villanova",
                                                  ;
      Address jHome =                             et", "Lynchburg",

      Student john =                              ", jHome, school);

      Address mHome =                             eet", "Euclid", "OH",
                                   44132);
      Student marsha = new Student ("Marsha", "Jones", mHome, school);

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

Output

```
John Smith
Home Address:
21 Jump Street
Lynchburg, VA  24551
School Address:
800 Lancaster Ave.
Villanova, PA  19085

Marsha Jones
Home Address:
123 Main Street
Euclid, OH  44132
School Address:
800 Lancaster Ave.
Villanova, PA  19085
```

```java
//********************************************************************
//  Student.java       Author: Lewis/Loftus
//
//  Represents a college student.
//********************************************************************

public class Student
{
   private String firstName, lastName;
   private Address homeAddress, schoolAddress;

   //-----------------------------------------------------------------
   //  Constructor: Sets up this student with the specified values.
   //-----------------------------------------------------------------
   public Student (String first, String last, Address home,
                   Address school)
   {
      firstName = first;
      lastName = last;
      homeAddress = home;
      schoolAddress = school;
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Returns a string description of this Student object.
   //-----------------------------------------------------------------
   public String toString()
   {
      String result;

      result = firstName + " " + lastName + "\n";
      result += "Home Address:\n" + homeAddress + "\n";
      result += "School Address:\n" + schoolAddress;

      return result;
   }
}
```

```
//********************************************************************
//  Address.java       Author: Lewis/Loftus
//
//  Represents a street address.
//********************************************************************

public class Address
{
   private String streetAddress, city, state;
   private long zipCode;

   //-----------------------------------------------------------------
   //  Constructor: Sets up this address with the specified data.
   //-----------------------------------------------------------------
   public Address (String street, String town, String st, long zip)
   {
      streetAddress = street;
      city = town;
      state = st;
      zipCode = zip;
   }
```

continue

continue

```
   //-----------------------------------------------------------------
   //  Returns a description of this Address object.
   //-----------------------------------------------------------------
   public String toString()
   {
      String result;

      result = streetAddress + "\n";
      result += city + ", " + state + "  " + zipCode;

      return result;
   }
}
```

Examples

- See `Client.java`
- See `Bus.java`
- See `BusApp.java`

Examples

- See Book.java
- See BookTest.java

- **See Library.java**
- **See TestLibrary.java**

Identifying Classes and Objects

- A partial requirements document:

> The user must be allowed to specify each product by its primary characteristics, including its name and product number. If the bar code does not match the product, then an error should be generated to the message window and entered into the error log. The summary report of all transactions must be structured as specified in section 7.A.

- Of course, not all nouns will correspond to a class or object in the final solution

Identifying Classes and Objects

- Sometimes it is challenging to decide whether something should be represented as a class

- For example, should an employee's address be represented as a set of instance variables or as an `Address` object

- The more you examine the problem and its details the more clear these issues become

- When a class becomes too complex, it often should be decomposed into multiple smaller classes to distribute the responsibilities

Identifying Classes and Objects

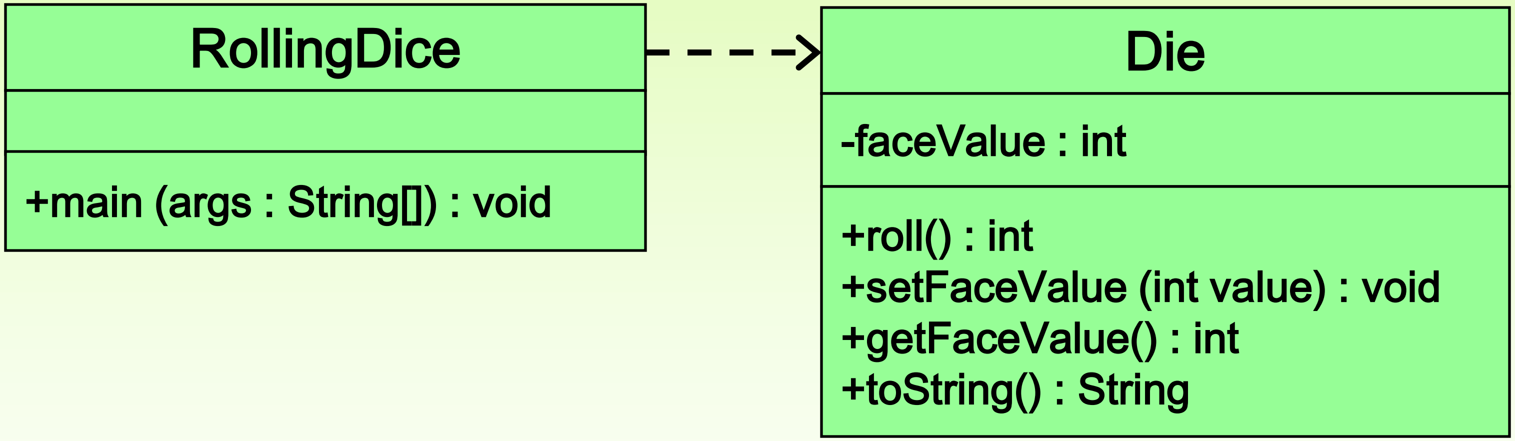
- Part of identifying the classes we need is the process of *assigning responsibilities* to each class
- Every activity that a program must accomplish must be represented by one or more methods in one or more classes
- We generally use verbs for the names of methods
- In early stages it is not necessary to determine every method of every class – begin with primary responsibilities and evolve the design

UML Diagrams

- UML stands for the *Unified Modeling Language*
- *UML diagrams* show relationships among classes and objects
- A UML *class diagram* consists of one or more classes, each with sections for the class name, attributes (data), and operations (methods)
- Lines between classes represent *associations*
- A dotted arrow shows that one class *uses* the other (calls its methods)

UML Class Diagrams

- A UML class diagram for the `RollingDice` program:



RollingDice

+main (args : String[]) : void

Die

-faceValue : int

+roll() : int
+setFaceValue (int value) : void
+getFaceValue() : int
+toString() : String

Static Variables

- Normally, each object has its own data space, but if a variable is declared as static, only one copy of the variable exists

```
private static float price;
```

- Memory space for a static variable is created when the class is first referenced
- All objects instantiated from the class share its static variables
- Changing the value of a static variable in one object changes it for all others

Static Class Members

- The following example keeps track of how many `Slogan` objects have been created using a static variable, and makes that information available using a static method

- See `SloganCounter.java`
- See `Slogan.java`

```java
//********************************************************************
//  SloganCounter.java       Author: Lewis/Loftus
//
//  Demonstrates the use of the static modifier.
//********************************************************************

public class SloganCounter
{
   //-----------------------------------------------------------------
   //  Creates several Slogan objects and prints the number of
   //  objects that were created.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Slogan obj;

      obj = new Slogan ("Remember the Alamo.");
      System.out.println (obj);

      obj = new Slogan ("Don't Worry. Be Happy.");
      System.out.println (obj);
```

continue

continue

```
      obj = new Slogan ("Live Free or Die.");
      System.out.println (obj);

      obj = new Slogan ("Talk is Cheap.");
      System.out.println (obj);

      obj = new Slogan ("Write Once, Run Anywhere.");
      System.out.println (obj);

      System.out.println();
      System.out.println ("Slogans created: " + Slogan.getCount());
   }
}
```

continue

```
      obj = new Slo
      System.out.pr

      obj = new Slo
      System.out.pr

      obj = new Slo                                  );
      System.out.pr

      System.out.println();
      System.out.println ("Slogans created: " + Slogan.getCount());
   }
}
```

Output

```
Remember the Alamo.
Don't Worry. Be Happy.
Live Free or Die.
Talk is Cheap.
Write Once, Run Anywhere.

Slogans created: 5
```

```java
//********************************************************************
//  Slogan.java       Author: Lewis/Loftus
//
//  Represents a single slogan string.
//********************************************************************

public class Slogan
{
   private String phrase;
   private static int count = 0;

   //-----------------------------------------------------------------
   //  Constructor: Sets up the slogan and counts the number of
   //  instances created.
   //-----------------------------------------------------------------
   public Slogan (String str)
   {
      phrase = str;
      count++;
   }
```

continue

continue

```
   //----------------------------------------------------------------
   //  Returns this slogan as a string.
   //----------------------------------------------------------------
   public String toString()
   {
      return phrase;
   }

   //----------------------------------------------------------------
   //  Returns the number of instances of this class that have been
   //  created.
   //----------------------------------------------------------------
   public static int getCount ()
   {
      return count;
   }
}
```

Examples

- See `Circle.java`
- See `TestCircle.java`