Object-Oriented Design

Chapter 7

Objectives

You will be able to

- Use the *this* reference in a Java program.
- Use the *static* modifier for member variables and methods of a Java class.

The `this` Reference

- The `this` reference allows the methods of an object to refer to the object's own members.
- The `this` reference, used inside a method, refers to the object through which the method is being executed.
- Usually not necessary, but makes the intent more clear.

The `this` Reference

- Suppose the **`this`** reference is used in a method called **`tryMe`**,

```
private int num;
public void tryMe(){
        this.num++;
}
```

- **`tryMe`** invoked as follows:

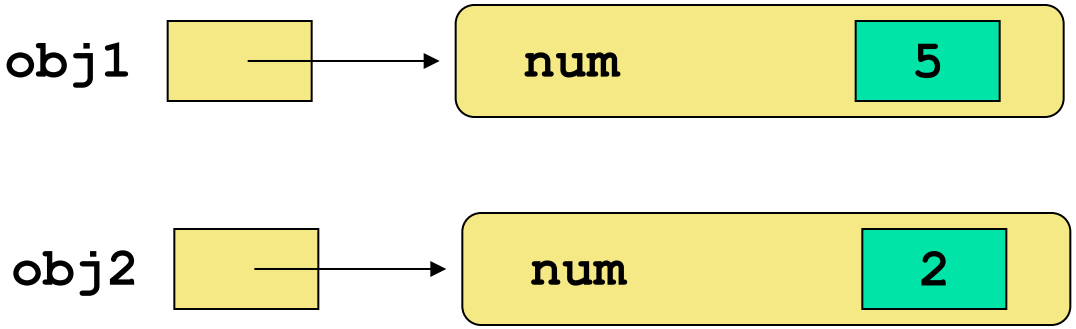
```
obj1.tryMe();
obj2.tryMe();
```

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

```
private int num;
public void tryMe(){
      this.num++;
}
```

- Before

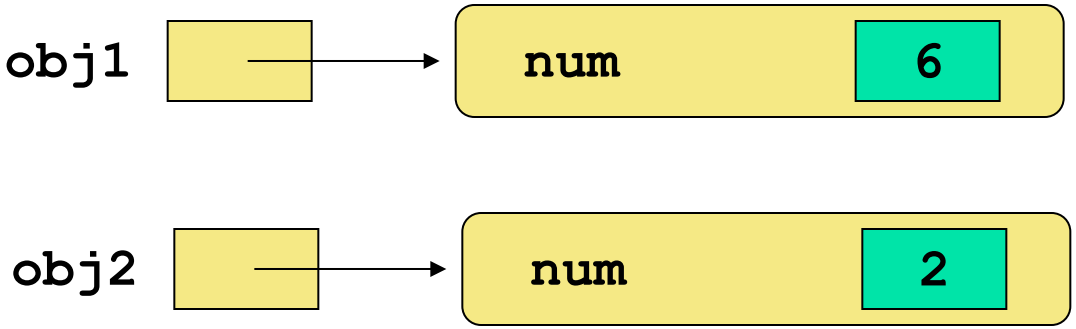
```
obj1.tryMe();
```



```
private int num;
public void tryMe(){
        this.num++;
}
```

- After

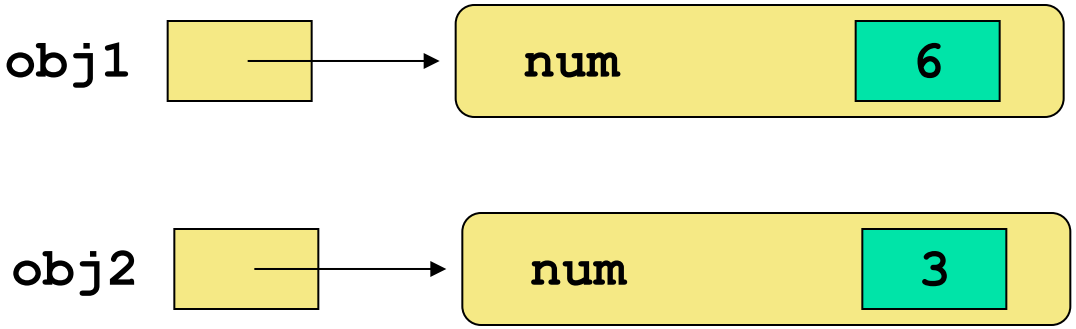
```
obj1.tryMe();
```



```
private int num;
public void tryMe(){
        this.num++;
}
```

- After

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



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.
- For example, the **`Account`** class (from Chapter 4) has three instance variables:

```
private String name;
private long acctNumber;
private double balance;
```

- The constructor was originally defined as:

```
public Account (String owner, long account,
                double initial)
{
   name = owner;
   acctNumber = account;
   balance = initial;
}
```

- The constructor can be modified using *this* reference

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

Exercise

- Modify the Dog class constructor using the **`this`** reference.

- Download to a test directory the two files in http://www.csee.usf.edu/~turnerr/Programming_Concepts/Downloads/2016_03_21_Writing_Classes/
  - Dog.java
  - Kennel.java

```
//*************************************************
//  Dog.java
//
//  Represents a dog.
//
//*************************************************

public class Dog
{
    // Instance variables
    private String name;
    private String breed;
    private int age;

    //-------------------------------------------------------
    // Constructor - sets up a dog object by initializing
    // the name, the breed, and the age.
    //-------------------------------------------------------
    public Dog(String newName, String newBreed, int newAge)
    {
        name = newName;
        breed = newBreed;
        age = newAge;
    }
```

Build and Run

```
Command Prompt
C:\test>
C:\test>javac Dog.java

C:\test>javac Kennel.java

C:\test>java Kennel
Enter the name of the dog: Spot
Enter the breed of the dog: Beagle
Enter the age of the dog: 4
Its age in person years is 28

Spot    Beagle  4  age in person years:28

C:\test>_
```

Revised Constructor

C:\test\Dog.java - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

Dog.java | Kennel.java

```
15      //-----------------------------------------------------
16      // Constructor - sets up a dog object by initializing
17      // the name, the breed, and the age.
18      //-----------------------------------------------------
19      public Dog(String name, String breed, int age)
20      {
21          name = name;
22          breed = breed;
23          age = age;
24      }
```

Java source file | length : 1408 lines : 48 | Ln : 25 Col : 4 Sel : 0 | 0 | Dos\Windows | ANSI | INS

Build and Run

```
Command Prompt
C:\test>
C:\test>javac Dog.java

C:\test>java Kennel
Enter the name of the dog: Spot
Enter the breed of the dog: Beagle
Enter the age of the dog: 4
Its age in person years is 0

null    null    0  age in person years:0

C:\test>
```

Whoa! What happened?

When a Method is Called

*C:\test\Kennel.java - Notepad++

Kennel.java

```
28
29 // Create a dog object
30 Dog dog1 = new Dog(dogName, dogBreed, dogAge);
31
```

Java s length : 1166 lines : 40 Ln : 33 Col : 32 Sel : 0 | 0 Dos\Windows ANSI INS

C:\test\Dog.java - Notepad++

Dog.java Kennel.java

```
15      //-----------------------------------------------
16      // Constructor - sets up a dog object by initiali
17      // the name, the breed, and the age.
18      //-----------------------------------------------
19      public Dog(String name, String breed, int age)
20      {
21          name = name;
22          breed = breed;
23          age = age;
24      }
```

Java source fil length : 1408 lines : 48 Ln : 25 Col : 4 Sel : 0 | 0 Dos\Windows ANSI INS

The *arguments* in the call are copied into the *parameters* of the method.

The method body is executed.

The parameters are local variables in the method.

In this case, the parameters *shadow* the member variables with the same names.

How to Avoid Shadowing

- The **`this`** reference permits us to refer to member variables having the same names as method parameters.

C:\test\Dog.java - Notepad++

File Edit Search View Encoding Language Settings Macro Run Plugins Window ?

Dog.java

```
15  //-----------------------------------------------------
16  // Constructor - sets up a dog object by initializing
17  // the name, the breed, and the age.
18  //-----------------------------------------------------
19  public Dog(String name, String breed, int age)
20  {
21      this.name = name;
22      this.breed = breed;
23      this.age = age;
24  }
```

Java source file | length : 1423 lines : 48 | Ln : 25 Col : 4 Sel : 0 | 0 | Dos\Windows | ANSI | INS

Build and Run

```
Command Prompt
C:\test>
C:\test>javac Dog.java

C:\test>java Kennel
Enter the name of the dog: Spot
Enter the breed of the dog: Beagle
Enter the age of the dog: 4
Its age in person years is 28

Spot    Beagle  4  age in person years:28

C:\test>
```

Now works as intended.

The `static` Modifier

Static Class Members

- Recall that a static method is one that can be invoked through its class name, instead of through an object of that class
- For example, the methods of the **`Math`** class are static:

```
result = Math.sqrt(25)
```

- Variables can be static as well

Static Class Members

- Determining if a method or variable should be static is an important design decision.
- The methods in the **`Math`** class perform computations based on values passed as parameters.
- There is no good reason to force us to create an object in order to request these services.

The static Modifier

- We declare static methods and variables using the **`static`** modifier.
- It associates the method or variable with the class rather than with an object of that class
- Recall that the **`main`** method is static
  - It is invoked by the Java interpreter without creating an object.

Static Methods

```
public class Helper
{
    public static int cube (int num)
    {
        return num * num * num;
    }
}
```

Because it is declared as static, the method can be invoked as

```
value = Helper.cube(3);
```

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.

Static Variables

- 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 order of the modifiers can be interchanged, but by convention visibility modifiers come first
- Static methods cannot reference instance variables because instance variables don't exist until an object exists
- A static method *can* reference static member variables and its own local variables.

Static Class Members

- Static methods and static variables often work together.
- 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.

Slogan.java

```
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++;
   }

   //-----------------------------------------------------------------
   //  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;
   }
}
```

SloganCounter.java

```
public class SloganCounter
{
   //-----------------------------------------------------------------
   //  Creates several Slogan objects and prints the number of
   //  objects that were created.
   //-----------------------------------------------------------------
   public static void main (String[] args)
   {
      Slogan obj1, obj2, obj3, obj4, obj5;

      obj1 = new Slogan ("Remember the Alamo.");
      System.out.println (obj1);

      obj2 = new Slogan ("Don't Worry. Be Happy.");
      System.out.println (obj2);

      obj3 = new Slogan ("Live Free or Die.");
      System.out.println (obj3);

      obj4 = new Slogan ("Talk is Cheap.");
      System.out.println (obj4);

      obj5 = new Slogan ("Write Once, Run Anywhere.");
      System.out.println (obj5);

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

Program Running

Command Prompt

```
C:\test>
C:\test>javac Slogan.java

C:\test>javac SloganCounter.java

C:\test>java SloganCounter
Remember the Alamo.
Don't Worry. Be Happy.
Live Free or Die.
Talk is Cheap.
Write Once, Run Anywhere.

Slogans created: 5

C:\test>
```

Question

- Can we call that static method getCount using an object rather than the class?

- Let's try it.
- Add at end of SloganCounter.java:

```
System.out.println("obj1 says Slogans created: " + obj1.getCount());
System.out.println("obj2 says Slogans created: " + obj2.getCount());
System.out.println("obj3 says Slogans created: " + obj3.getCount());
System.out.println("obj4 says Slogans created: " + obj4.getCount());
System.out.println("obj5 says Slogans created: " + obj5.getCount());
```

Test the Modified Program

Command Prompt

```
C:\test>
C:\test>javac SloganCounter.java

C:\test>java SloganCounter
Remember the Alamo.
Don't Worry. Be Happy.
Live Free or Die.
Talk is Cheap.
Write Once, Run Anywhere.

Slogans created: 5
obj1 says Slogans created: 5
obj2 says Slogans created: 5
obj3 says Slogans created: 5
obj4 says Slogans created: 5
obj5 says Slogans created: 5

C:\test>
```

Answer

- Yes, we can call static methods using an object as well as using the class name.

Exercise

- Modify the Dog class such that it contains static variable(s) and static method(s) to keep track of how many dog objects were created and make the information available in the driver (Kennel) program.

Dog.java

```
public class Dog
{
    private static int Dog_Count = 0;

    // Instance variables
    private String name;
    private String breed;

    private int age;
    ...


    public static int Get_Dog_Count()
    {
        return Dog_Count;
    }
```

Kennel.java

```
        // Print summary for the dog
        System.out.println(dog1);

        System.out.println("Number of dogs created: " +
                                Dog.Get_Dog_Count());
    }
}
```

Build and Run

```
Command Prompt
C:\test>
C:\test>javac Dog.java

C:\test>javac Kennel.java

C:\test>java Kennel
Enter the name of the dog: Spot
Enter the breed of the dog: Beagle
Enter the age of the dog: 4
Its age in person years is 28

Spot    Beagle  4  age in person years:28
Number of dogs created: 1

C:\test>_
```

Method Design: Overloading

Method Design: Overloading

- *Method overloading* is the process of giving a single method name multiple definitions
- It is useful when you need to perform similar methods on different types of data.

Method Overloading

- The **`println`** method is overloaded:

```
println (String s)
println (int i)
println (double d)
```

and so on...

Method Overloading

- The following lines invoke different versions of the **`println`** method:

```
System.out.println ("The total is:");
System.out.println (total);
```

Method Overloading

- If a method is overloaded, the method name is not sufficient to determine which method is being called.
- The *signature* of each overloaded method must be unique.
- The signature includes the number, type, and order of the parameters
  - But not the return type.

Method Signature

- The signature of the following method includes 2 parameters, *int* and *float*, and the order of them (*int* first and *float* second).

```
float tryMe(int x, float y)
{
   return x*y;
}
```

Method Overloading

The compiler determines which method is to invoke by analyzing the parameters

```
float tryMe(int x)
{
    return x + .375;
}

float tryMe(int x, float y)
{
    return x*y;
}
```

Invocation

```
result = tryMe(25, 4.32)
```

Overloading Methods

- The return type of the method is *not* part of the signature.
  - Overloaded methods cannot differ only by their return type.

```
float tryMe(int x, float y)
{
    return x*y;
}
```

Error!

```
int tryMe(int x, float y)
{
    return (int)x*y;
}
```

Invocation

```
result = tryMe(25, 4.32)
```

Overloading Constructors

- Constructors can be overloaded
  - We can have two more overloaded constructors in one class.
- Overloaded constructors provide multiple ways to initialize a new object of a given class.

Overloading Constructors

```
public Account (String name, long acctNumber)
{
    this.name = name;
    this.acctNumber = acctNumber;
    this.balance = 0;
}
```

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

Exercise

- Write a class *ComputeMax* with two overloaded methods *max*
  - One method takes two parameters and computes the maximum.
  - The other takes three parameters and computes the maximum.

Readings and Assignments

- Not submitted or graded
  - Reading: Chapter 7.1-7.4, 7.7-7.8
  - Exercises and Programming Projects:
    - After Chapter Exercises
      - EX 7.1, 7.2, 7.3, 7.4, 7.5
    - Programming Projects
      - PP 4.6, 4.7, 4.8
  - Exercise on previous slide.

- Assignment to be submitted and graded:
  - Project 13