OBJECT ORIENTED PROGRAMMING USING C++

Chapter 9 - Classes with Class Members

- Class Variables
- Class Methods
- How to Access Class Members
- When to Use Class Members
- Class Constants
- Example Program Using Class Members

Class Variables

- Based on what you've learned so far, when you're working on an object-oriented program, you should envision separate objects, each with their own set of data and behaviors (instance variables and instance methods, respectively).
- That's a valid picture, but you should be aware that in addition to data and behaviors that are specific to individual objects, you can also have data and behaviors that relate to an entire class. Since they relate to an entire class, such data and behaviors are referred to as *class variables* and *class methods*, respectively.

- For a particular class, each of the class's objects has its own copy of the class's instance variables.
- For a particular class, each of the class's objects shares a single copy of the class's class variables.

Class Variables

- If you'd like a variable to be shared throughout its class, make it a class variable by using the `static` modifier in its declaration:

  <private or public> `static` *<type> <variable-name>*`;`

- Example:

```
public class Mouse
{
  private static int mouseCount;
  private static double averageLifeSpan;
  ...
```

- Class variables are declared at the top of the class, above all the methods.

Class Variables

- Class variables use the same default values as instance variables:
  - integer types get `0`
  - floating point types get `0.0`
  - boolean types get `false`
  - reference types get `null`
- What are the default values for the class variables in this code fragment?

```
public class Mouse
{
  private static int mouseCount;
  private static double averageLifeSpan;
  private static String researcher;
  private static int simulationDuration = 730;
  ...
```

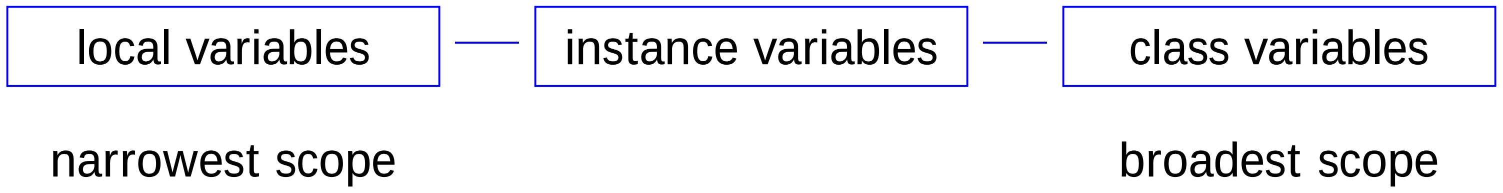
Initializations are allowed.

Scope

- You can access a class variable from anywhere within its class; i.e., you can access class variables from instance methods as well as from class methods.
- That contrasts with instance variables, which you can access only from instance methods.
- Thus, class variables have broader scope than instance variables. Local variables, on the other hand, have narrower scope than instance variables. They can be accessed only within one particular method.

Scope

- Here is the scope continuum:



- Narrower scope equates to more encapsulation, and encapsulation means you are less vulnerable to inappropriate changes.
- Class variables, with their broad scope and lack of encapsulation, can be accessed and updated from many different places, and that makes programs hard to understand and debug. Having broader scope is necessary at times, but in general you should try to avoid broader scope.
- Thus, you should prefer local variables over instance variables and instance variables over class variables.

Class Methods

- If you have a method that accesses class variables and not instance variables, then you should declare the method to be a *class method*. To do so, add `static` to the method's heading like this:

  `<private-or-public> static <return-type> <method-name>(<parameters>)`

- Example:

```
public class Mouse
{
  private static int mouseCount;
  private static double averageLifeSpan;

  public static void printMouseCount()
  {
    System.out.println("Total mice = " +
      Mouse.mouseCount);
  }
}
```

To access a class variable, prefix it with *<class-name>* dot.

How to Access Class Members

- Normally, to access a class member (a class variable or a class method), prefix it with <*class-name*> dot.
- Shortcut syntax for a class member:
  - In accessing a class member, you may omit the <*class-name*> dot prefix if the class member is in the same class as where you're trying to access it from. For example:

```
public class Mouse
{
  private static int mouseCount;
  private static void printMouseCount()
  {
    System.out.println("Total mice = " + Mouse.mouseCount);
  }
  public static void main(String[] args)
  {
    Mouse.printMouseCount();
  }
} // end Mouse class
```

OK to access `mouseCount` without <*class-name*> dot.

OK to access `printMouseCount` without <*class-name*> dot.

When to Use Class Methods

- You should make a method a class method:
  - If you have a method that uses class variables and/or calls class methods, then it's a good candidate for being a class method. Warning: If in addition to accessing class members, the method also accesses instance members, then the method must be an instance method, not a class method.
  - If you might need to call a method even when there are no objects from the method's class, then you should make it a class method.
  - The `main` method has to be a class method. If a main method uses helper methods that don't involve instance members, then the helper methods should be class methods as well.

Class Constants

- Sometimes, you'll want a class variable to be fixed/constant. That type of "variable" is called a *class constant*.
- To make a class constant, declare a variable with the `static` and `final` modifiers like this:

  <private or public> `static final` *<type> <variable-name>* `=` *<initial value>*`;`

- Note that class constants should be assigned as part of an initialization statement. If you attempt to assign a value to a class constant later on, that generates a compilation error.
- As with most variables, class constants usually use the `private` modifier. However, if the constant is so important that more than one class needs to use it, then you should use the `public` modifier.

Class Constants

- In the below Human class, we make NORMAL_TEMP a class constant (with the static and final modifiers) because all Human objects have the same normal temperature of 98.6° Fahrenheit.

class constant

```
public class Human
{
  private static final double NORMAL_TEMP = 98.6;
  ...
  public boolean isHealthy()
  {
    return Math.abs(currentTemp - NORMAL_TEMP) < 1;
  } // end isHealthy

  public void diagnose()
  {
    if ((currentTemp - NORMAL_TEMP) > 5)
    {
      System.out.println("Go to the emergency room now!");
      ...

} // end class Human
```

Class Constants

- Prior to this chapter, you used named constants that were declared locally within a method. For example:

```
public void calculateSignalDelay(double cableDistance)
{
  final double SPEED_OF_LIGHT = 299792458.0;
  double propagationDelay;  // time for electron to travel
                            // length of cable
  ...
  propagationDelay = cableDistance / SPEED_OF_LIGHT;
  ...
}
```

- So when should you use a local variable named constant and when should you use a class constant?
  - Use a local variable named constant if the constant is only needed within one method.
  - Use a class constant if the constant is a property of the collection of all the objects in the class or of the class in general.

Example Program Using Class Members

```
/*************************************************************
* PennyJar.java
* Dean & Dean
*
* This class counts pennies for individual penny jars and for
* all penny jars combined.
*************************************************************/

public class PennyJar
{
  public static final int GOAL = 10000;
  private static int allPennies = 0;
  private int pennies = 0;

  //**********************************************************

  public int getPennies()
  {
    return this.pennies;
  }

  //**********************************************************

  public void addPenny()
  {
    System.out.println("Clink!");
    this.pennies++;
    PennyJar.allPennies++;

    if (PennyJar.allPennies >= PennyJar.GOAL)
    {
      System.out.println("Time to spend!");
    }
  } // end addPenny
```

Example Program Using Class Members

```
  //************************************************************

  public static int getAllPennies()
  {
    return PennyJar.allPennies;
  }
} // end class PennyJar

/**********************************************************
* PennyJarDriver.java
* Dean & Dean
*
* This class drives the PennyJar class.
**********************************************************/

public class PennyJarDriver
{
  public static void main(String[] args)
  {
    PennyJar pennyJar1 = new PennyJar();
    PennyJar pennyJar2 = new PennyJar();

    pennyJar1.addPenny();
    pennyJar1.addPenny();
    pennyJar2.addPenny();
    System.out.println(pennyJar1.getPennies());
    System.out.println(PennyJar.getAllPennies());
  } // end main
} // end class PennyJarDriver
```