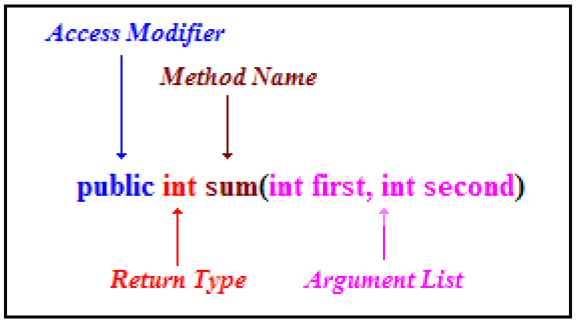
Method Signatures (Outsource: 6-6)

When you write a method you must first define the method signature. The signature of a method is made up of the following parts:

- Access modifier (public, private, protected)
- Return type
- Name of the method
- Argument list



```
public int sum(int first, int second)
```

Method Signature

Methods That Require a Single Argument (Outsource: 6-8 - 6-9)

When you write a method declaration for a method that can receive an argument, you need to include the following items within the method declaration parentheses:

- The Type of the argument
- A local name for the argument

For example, the declaration for a public method named **displayTax()** that displays the amount of sales tax on an item could have the declaration **public void displayTax(double saleAmount)**.

The argument *double saleAmount* within the parentheses indicates that the **displayTax()** method will receive a value of type double which can be used in the method **displayTax()**. The complete method could be written as follows:

```
public void displayTax(double saleAmount) {
   final double tax = .0825;                       // Amount of Sales Tax
   double taxAmount = saleAmount * tax;
   DecialFormat df = new DecimalFormat("#.00");    // Two decimal places
   System.out.println("Sales Tax: $") + df.format(taxAmount));
}
```

To make a call to **displayTax()** you would simply name the method and pass it a numeric value. For example, **displayTax(25.50)** would call **displayTax()** sending it the value *25.50*. The method **displayTax()** would receive the value 25.50 into the double saleAmount, calculate the tax amount and display the output: **Sales Tax $ 2.10**.

Note: The variable *taxAmount* is a local variable to the **displayTax()** method.

Methods That Require Multiple Arguments

A method can require more than one argument. You can pass multiple arguments to a method by listing the arguments within the call to the method and separating them with commas. For example, rather than creating a displayTax( ) method that calculates tax at a fixed 8.25%, you could create a displayTax( ) method that receives two arguments: salesAmount and taxAmount.

```
public void displayTax(double saleAmount, double taxPct) {
   double taxAmount = saleAmount * taxPct;
   DecialFormat df = new DecimalFormat("#.00");    // Two decimal places
   System.out.println("Sales Tax: $" + df.format(taxAmount));
}
```

To make a call to **displayTax()** you would simply name the method and pass it two numeric values - *salesTax* and *taxPct* in the order that they appear in the argument list. For example, **displayTax(25.50, .06)** would call **displayTax()** sending it the value *25.50* which would be stored in *salesAmount* and 8.0 which would be stored in *taxPct*. The method **displayTax()** would receive the value 25.50 and 8.0 respectively, calculate the tax amount and display the output: **Sales Tax $ 1.53**.

A declaration for a method that receives two or more arguments must list the type for each argument separately, even if the arguments have the same type.

The arguments in a method call often are referred to as **actual parameters**. The variables in the method declaration that accept the value from the actual parameters are the **formal parameters**.

Passing Arrays As Arguments

An array is an object and anytime you pass an object to a method it is passed by reference. Any change made to the array in the method will be reflected in the original array. The following program sample shows an example of passing an entire array to a method called **getNames** which changes the contents of the array.

```
public void someMethod() {
   String[ ] names = new String[10];
   getNames(names);                    // Passing the entire array
}

public void getNames(String[] n) {     // Receiving the entire array by reference
   Scanner reader = new Scanner(System.in);
   for (int i = 0; i < n.length; i++)
   {
      System.out.print("Enter a name: ");
      n[i] = reader.next ();           // or kybd.nextLine()
   }
}
```

When you pass an individual element of an array to a method it is passed by value, that is to say any changes you make in the method will not be reflected in the original array. The following program sample shows an example of passing an individual element of an array to a method:

```
public void someMethod( ) {
    String[] names = new String[10];
    for (int i = 0; i < names.length; i++)
        showGreeting(names[i]);                // Sending a single element
}

public void showGreeting(String name) {        // Receiving a single element by value
    System.out.println("Hello " + name + ", how are you today?");
}
```

Method Overloading (Polymorphism)

Polymorphism translates from Greek as **many forms** (*poly - many morph - forms*). **Method overloading** is a classic example of Polymorphism. Overloading involves using one term to indicate diverse meanings. When you overload a Java method, you write multiple methods with a shared name. The compiler understands your meaning based on the arguments you use with the method. An example of an overloaded method is **print()** in the java.io.PrintStream class

```
public void print(boolean b)
public void print(char c)
public void print(char[] s)
public void print(float f)
public void print(double d)
public void print(int i)
public void print(long l)
public void print(Object obj)
public void print(String s)
```

Suppose you create a class method to apply a simple interest rate to a bank balance. The method receives two double arguments – the balance and the interest rate – and calculates the interest earned.

```
import java.util.Scanner;

public class SimpleInterest
{
   private double interest; // Class member variable to hold the interest earned
   private double balance;   // Class member variable to hold the balance
   private double rate;      // Class member variable to hold the interest rate

   public static void main(String args[]) {
      SimpleInterest si = new SimpleInterest ();
      si.input();
      si.simpleInterest(balance, rate);
      si.output();
   }

   public void input() {
      Scanner reader = new Scanner(System.in);

      System.out.print("Enter the balance: ");
      balance = reader.nextDouble();
      System.out.print("Enter the interest rate: ");
      rate = reader.nextDouble();
   }

   public void simpleInterest(double bal, double rate) {
      interest  = bal * rate;
   }

   public void output() {
      DecialFormat df = new DecimalFormat("#.00");      // Two decimal places
      System.out.println("The interest earned is " +
                            df.format(interest));
   }
}
```

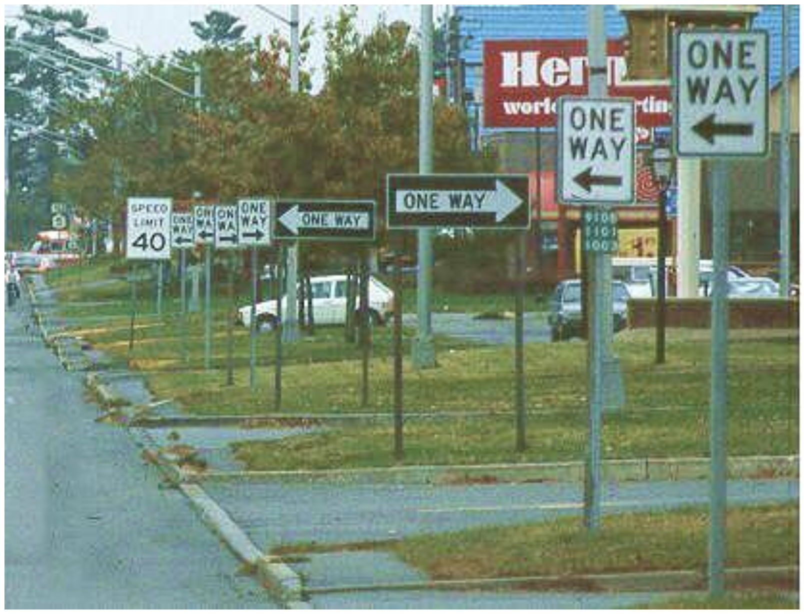
Normally the interest rate would be a double such as .04 (4 %). However, you might want to allow the user to work with interest as a fraction (.04) or as a percentage (4). A solution to this problem is to overload the **simpleInterest()** method. Overloading involves writing multiple methods with the same name, but with different arguments. For example, in addition to the **simpleInterest()** method shown above, you could add the following method:

```
public void simpleInterest(double balance, int rate) {
   interest = bal * rate / 100);
}
```

If the **simpleInterest()** method is called using two double arguments, as in **simpleInterest(1000.00, .04)**, the **simpleInterest()** method that receives two doubles will execute. However, if an integer is used as the second parameter in the call to **simpleInterest()**, as in **simpleInterest(1000.00, 4)**, then the **simpleInterest()** method that receives a double and an integer will execute.

Ambiguity

When you overload a method, you run the risk of creating an ambiguous situation – one in which the compiler cannot determine which method to use. For example, if you have overloaded **simpleInterest()** using the two methods shown above and you make the method call **int interest = simpleInterest(1000, 4);** you have created an ambiguous situation. The compiler does not know which method to call because neither method receives two integer arguments and, since an integer can be sent to a double, either method call is valid. The solution would be to have a third **simpleInterest()** method that receives two integer arguments.



Ambiguity results in confusion.

Methods That Return Values (Outsource: 6-10 – 6-11)

The return type for a method can be any type used in the Java programming language, which includes the primitive types int, double, char, and so on, as well as class types (including class types your create). Of course, a method also can return nothing, in which case the return type is *void*. For example, a method that returns an integer value could be declared as **public int someMethod()**.

The following example calculates the distance between to points on a line. The two points are received as arguments and the result of the calculation (distance) is returned as a double.

```
public double distance(double a, double b) {
   double distance = Math.abs(a - b);
   return distance;
}
```

Alternatively, we can simply return the results of the math process.

```
public double distance(double a, double b) {
   return Math.abs(a – b);
}
```

Notice the return type **double** in the method header. Also notice that a **return** statement is the last statement in the method. The return statement causes the value stored in *distance* to be sent back to any method that calls **distance**().

If a method returns a value, then when you call the method, you usually will want to use the return value, although you are not required to do so.

Alternatively, you can choose to display a method's return value directly without storing it in any variable, as in **System.out.println("Net Pay: " + calculatePay(hoursWorked, hourlyPay));**. In this last statement, the call to the **calculatePay**() method is made from within the **println**() method call. Because **calculatePay()** returns a double, you can use the method call **calculatePay**() in the same way you would use any simple double value. For example, besides printing the value of **calculatePay**(), you can perform math with it, assign it, and so on.

The following program demonstrates the use of passing arguments and returning values.

```
import java.util.Scanner;

public class CalculateHypotenuse
{
   private double sideA;                // Instance variable to hold side a
   private double sideB;                // Instance variable to hold side b

   public static void main(String args[]) {
      CalculateHypotenuse calc = new CalculateHypotenuse();
      calc.input();
      calc.output();
   }

   public void input() {
      Scanner reader = new Scanner(System.in);

      System.out.print("Enter the length of the first side: ");
      sideA = reader.nextDouble();
      System.out.print("Enter the length of the second side: ");
      sideB = reader.nextDouble();
   }

   public double hypotenuse(double a, double b) {
      return Math.sqrt(Math.pow(a,2) + Math.pow(b,2));
   }
   public void output() {
      System.out.println("The length of the Hypotenuse is " +
                         hypotenuse(sideA, sideB));
   }
}
```

Arrays As A Return Type

Arrays can be used as method return types. For example, to declare a method that returns an array of ints use the following declaration:

```
public int[ ] myMethod()
```

Of course, it is now necessary to end the method with a return statement that returns a matching array, in this case an array of ints. The following example demonstrates calling a method that returns an array of ints:

```
public int[ ] doSomething() {
    int[ ] numbers = new int[10];

    // do something with numbers
    return numbers;
}
```

Since doSomething returns an array we would want to call it from another method as an assignment statement:

```
int[ ] values = doSomething();
```

Lab 12 - ASSIGNMENT

Lab 12A - 60 points

OBJECTIVE

Speed signs in Europe and Asia (and pretty much every else in the world except the United States) are marked in Kilometers per hours. **One mile is equal to .6214 kilometers.** WAP that reads a speed in kilometers per hour and converts it into miles per hour.

FIELD SUMMARY

- **int kilometers** – the speed limit in kilometers per hour.

METHOD SUMMARY

- **main** – instantiate an object of this class. Make method calls to input and output.
- **input** – declare a Scanner object and read the speed in kilometers per hour.
- **kilometersToMiles** – **Arguments:** an int - the speed in kilometers. **Return:** a double. Calculate and return the speed in miles per hour. This method should be declared as ***static***.
- **output** – display the results on the console screen. Make appropriate calls to **`kilometersToMiles`** from the println statement. Format all doubles to show one decimal place with no trailing zeros.

SAMPLE KEYBOARD INPUT

Enter the speed limit in kilometers per hour: **60**

SAMPLE OUTPUT

60 kilometers per hour = 37.3 miles per hour.

Lab 12B - 70 points

OBJECTIVE

WAP that reads the radius of a circles from the keyboard. Calculate the area and circumference of the circle and display the results.

area $= \pi r^2$
circumference $= 2 \pi r$

FIELD SUMMARY

- **double radius** – the radius of the circle.

METHOD SUMMARY

- **main** – create an instance of this class. Make method calls to input and output.
- **input** – declare a Scanner object and read the radius of a circle using an appropriate prompt.
- **area** – **Arguments:** a double - the radius of the circle. **Return:** a double. Calculate and return the area of the circle. Use Math methods wherever applicable. This method should be declared as ***static***.
- **circumference** – **Arguments:** a double - the radius of the circle. **Return:** a double. Calculate and return the circumference of the circle. Use Math methods wherever applicable. This method should be declared as ***static***.
- **output** – display the results on the console screen. Make appropriate calls to `area` and `circumference` from the println statements. Format all doubles to show one decimal place with no trailing zeros.

SAMPLE KEYBOARD INPUT

Enter the radius of a circle: **12.5**

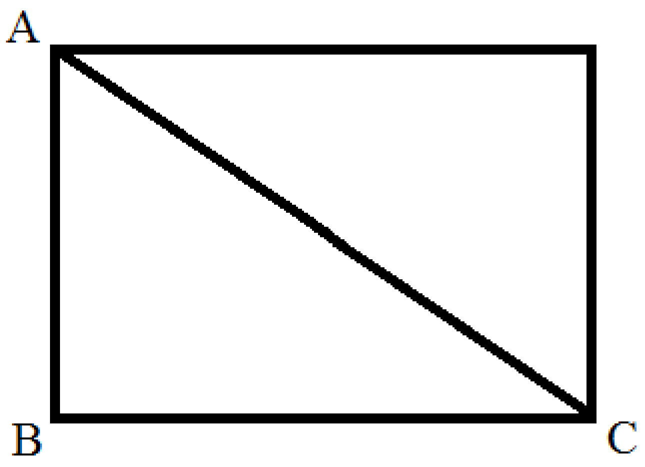
SAMPLE OUTPUT

```
The area of a circle with a radius of 12.5 is 490.9
The circumference of a circle with a radius of 12.5 is 78.5
```

Lab 12C - 80 points

OBJECTIVE

WAP that reads the distance from A to B and the distance from A to C for the rectangle shown below. Input will be from the keyboard. Calculate the area of the rectangle and display the results on the console screen.



FIELD SUMMARY

- **double ab** – the distance from A to B.
- **double ac** – the distance from A to C.

METHOD SUMMARY

- **main** – create an instance of this class. Make method calls to input and output.
- **input** – declare a Scanner object and read the distance from A to B and the distance from A to C using appropriate prompts.
- **area** – Arguments: two doubles - the distance from A to B and the distance from A to C. Return: a double. Calculate and return (as a double) the area of the rectangle. **Use Math methods wherever applicable.** This method should be declared as ***static***.
- **output** – display the results on the console screen. Make a call to `area` from the println statement. Format all doubles to show one decimal place with no trailing zeros. Whole values of more than three digits should be separated by commas.

SAMPLE KEYBOARD INPUT

Enter the distance from A to B: **12.5**
Enter the distance from A to C: **16.6**

SAMPLE OUTPUT

The area of the rectangle where AB is 12.5 and AC is 16.6 is 136.5.

Lab 12D - 90 points

OBJECTIVE

Debug Error Prone Ellie's latest program. Use the following chart format to describe the syntactical errors discovered and what you did to eliminate each error. To receive credit for this assignment you must turn in the chart and demonstrate a working program.

Line #	Error as reported by Java	What the error really was	What you did to fix the error.

Lab 12E - 100 points

OBJECTIVE

Free-falling objects are in a state of acceleration. Specifically, they are accelerating at a rate of 32.2 feet per second^2. In other words the velocity of a free-falling object is changing by 32.2 feet per second every second. The formula to calculate the speed of a falling object as:

$$s = a * t^2$$

where **`s`** is the speed of a falling object, **`a`** is the acceleration produced by the pull of gravity (32.2), and **`t`** is the time in seconds.

The formula for determining the distance an object falls in a specified time is:

$$d = \frac{1}{2} * a * t^2$$

where **`d`** is the distance traveled, **`a`** is the acceleration produced by the pull of gravity (32.2) and **`t`** is the time in seconds. From this, we can derive the time it takes a falling object to fall a specified distance:

$$t = \sqrt{d * 2 / a}$$

WAP that reads a distance in feet from the keyboard. Calculate the amount of time it would take for an object to reach the ground if it were dropped that distance from the ground. Since we are dropping the object you can assume the initial speed is 0. Ignore the effects of wind resistance. Then calculate the speed the object would be moving when it impacted with the ground.

FIELD SUMMARY

- **int distance** – the distance to the ground.

METHOD SUMMARY

- **main** – create an instance of this class. Make method calls to input and output.
- **input** – declare a Scanner object and read the distance to the ground using an appropriate prompt.
- **time** – Arguments: an int - the distance to the ground. Return: a double. Calculate and return the time it takes for the object to reach the ground. Use Math methods wherever applicable. This method should be declared as ***static***.
- **speed** – Arguments: an int - the distance to the ground. Return: a double. Calculate and return the speed the object is moving when it hits the ground. Use Math methods wherever applicable. You will need to make a call to **time**. This method should be declared as ***static***.
- **output** – display the results on the console screen. Make appropriate calls to `time` and `speed` from the println statements. Format all doubles to show one decimal place with no trailing zeros.

SAMPLE KEYBOARD INPUT

```
Enter the distance <in feet> to the ground: 1250
```

SAMPLE OUTPUT

```
It would take an object 8.8 seconds to fall 1250 feet.
The object would be moving 2500 feet per second when it hit the ground.
```