AP Computer Science A

Summer Packet 2021

Herndon High School

Welcome to Computer Science! In AP Computer Science A, you will learn how to think logically, problem solve, and program using the Java language. This packet has many examples of working programs and describes some of the key features of Java.

There are many online resources for java programming. One of the best is the CS Awesome site because it is interactive and corresponds to the content of this class. You can access the site at CS Awesome or by typing course.csawesome.org into your browser.

This course uses an integrated development environment (IDE) called jGRASP. This application makes developing and testing java programs easier. If you would like to install jGRASP on your computer so that you can practice with the environment, then go to the jGRASP site https://www.jgrasp.org/index.html and download and install the bundled version of jGRASP. "Bundled" means that the install package includes a version of Java so that you can install one program and be ready to go.

If you install JGRASP and want to test it out. You can try the following:

Double-click on jGrasp.
In the upper-left hand corner, click on **File->new->Java**. A large, blank editing window will open in the center-right side of the program. In that editing window, type in the following:

```
public class HelloWorld
{
   public static void main(String[] arg)
   {
      System.out.println("Hello World");
   }
}
```

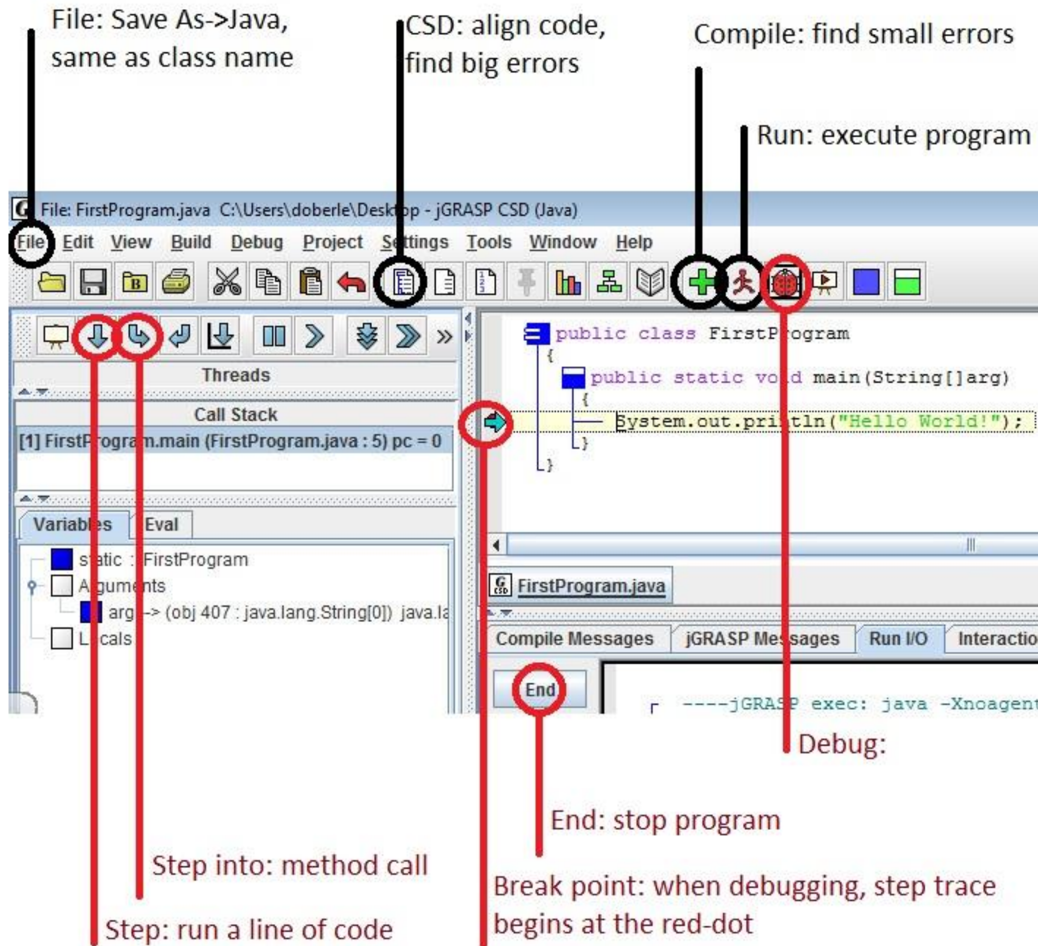
Now go to **File->Save As**. If the program is typed in correctly, it will save as **HelloWorld.java**.
Click on the Compile button (the button with the +). If there are no errors, click on the run button (next to compile).

Avoiding common errors:
The file must be the same as the title on the first line after public class, including case sensitivity.
If the program is defined within **public class HelloWorld**, the file must be saved as **HelloWorld.java**.

The vertically aligned brackets are the “squiggly-brackets”, to the right of the ‘p’ key on the keyboard.
The parenthesis after main and println are the round-parenthesis SHIFT-9 and SHIFT-0 keys.
The System.out.println call is followed by a semi-colon, to the right of the ‘L’ key.

The following pages contain a quick reference guide on jGRASP. This is followed by Java Basics which include many examples. Read as far as you choose to learn about Java.

Try creating your own program to accomplish a task. Try asking the user for some input and then providing them back some output. Your program should be of your own design, not one of the examples from Java Basics, from a book or online. Think of an idea that will require if-statements, if-else statements, and loops. Have fun and experiment. See you in August!



File: save as a Java file with the same name as the public class name.

CSD: finds big errors, like unmatched parenthesis, braces and brackets. Aligns your code to make it easy to read.

Compile: finds small errors, like missing semicolon, spelling and case-sensitivity issues.

Run: executes last successful compile.

Debug: if your program compiles but does not execute correctly, you can run the program one line at a time and watch the variables change with the debugger. Set up a break point at the part of the program you want to debug, hit the Debug button and step through the code to find the logical error.

Break Point: along the left border of the code window, toggle red-dot for a break point where you want to start debugging.

Step: when debugging, step will execute a single line of code

Step into: when debugging, step into will enter a method call to allow you to debug within.

End: halt a program if you have an infinite loop.

Java Basics:

Java programs must be titled with a class name that is exactly the same as its file name, with case sensitivity. When you run a java program, it executes the code found in the main function.

```
public class FirstProgram
{
    public static void main(String [] arg)
    {
        System.out.println("Hello World.");
    }
}
```

- This must be in a file called FirstProgram.java, since that is the name of the public class.

- The main function always appears as: public static void main(String [] arg),
and the code within the braces contains the commands that you want to run.

- System.out.println will write text and information out to the screen. Upon running the program, it will write the message **Hello World**. to the screen.

Java is an object-oriented programming language, which means you can define and use objects.
An **Object** is an entity that can store multiple pieces of data (data fields) as well as perform actions (methods). Consider that you might have a Tank object called sherman that stores information, like its location, speed, and amount of ammo, but can also perform actions like sherman .move(5), which might move forward 5 meters, or sherman .shoot(), which fires a shell and decreases the ammo count by one.

Primitive variables are simple data types that can store a single value. There are many different types, but we will primarily use the following:
- int stores a single whole number (integer value)
- double stores a single real number (may have a decimal)
- boolean stores the value true or false (used for conditions)

And we will also use a common object called the String:
- String stores a collection of characters (a word or sentence)

Declaring a variable requires assigning a type, and giving it a name to identify it. Identifier names need only follow these simple rules:
- only comprised of letters, numbers and the underscore
- must start with a letter, and should be a name that makes sense for its use
- can't be a reserved word, which is already taken by Java (public, static, void, String, int, etc).

```
int num = 5;              //creates an integer called num that stores the state 5.
double accel = 9.81;      //creates a real number called accel that stores the state 9.81
boolean done = false;     //creates a boolean called done that stores the state false
String word = “hello”;    //creates a String called word that stores the state hello.
```

Note that literal strings are placed inside of quotes “”.

```java
//in SecondProgram.java
public class SecondProgram
{
   public static void main(String [] arg)
   {
      int num = 5;
      double x = 4.5;
      System.out.println("Our integer is " + num + " and our real is " + x );
   }
}
```

The program will output the following when run: **Our integer is 5 and our real is 4.5**
Literal text is placed within quotes: "Our integer is ". We can write the state of a variable by adding the variable name to the literal text. Upon reaching the num, it writes the state of the variable which is 5.

Consider that our println command looked like the following:
System.out.println("Our integer is num and our real is x”);
In this case, the output would be **Our integer is num and our real is x**, because the entire contents would be considered literal text.

Any text after a double slash // will be ignored by the compiler and is used to comment code.
This is called an end-of-line comment, because everything after it on that line is ignored.
You can comment out several lines of code by placing comments between /* and */.

```java
//in SecondProgramB.java
public class SecondProgramB
{
   public static void main(String [] arg)
   {
      /*
         NONE OF THIS TEXT WILL AFFECT THE CODE.
         It is just a comment block, and used to communicate information about the program.
         You will learn what is worthy to comment later.
      */
      int num = 5;
      double x = 4.5;

      //this is also a comment and will be ignored by the compiler

      System.out.println("Our integer is " + num + " and our real is " + x );
   }
}
```

Basic math operators include:
+ (addition), - (subtraction), * (multiplication), / (divide), / (div), and % (modulous)

/ (division), when given at least one number with a decimal.

```
System.out.println(3 / 6.0);       //would output 0.5
System.out.println(3.0 / 6);       //would output 0.5
System.out.println(3.0 / 6.0);     //would output 0.5
int x = 3;
double y = 6.0;
System.out.println(x / y);         //would output 0.5
```

/ (div), when given two integers, performs whole number division.

```
System.out.println(3 / 6);         //would output 0, because 6 goes into 3 zero times
int a = 3;
int b = 6;
System.out.println(a / b);         //would output 0
a = 7;
b = 3;
System.out.println(a / b);         //would output 2, because 3 goes into 7 two times
```

% (mod), when given two integers, mod returns the remainder after whole number division.

```
System.out.println(3 % 6);         //would output 3, because 6 goes into 3 zero times
                                   //with a  remainder of three
a = 7;
b = 3;
System.out.println(a % b);         //would output 1, because 3 goes into 7 two times
                                   //with a remainder of one
System.out.println(6 % 3);         //would output 0, because 3 goes into 6 two times
                                   //with a  remainder of zero
```

The assignment operator = is used to assign a value to a variable.
Only the variable to the left of the assignment operator will change.

```
        a = 7;                          //a is assigned the state 7
/*      7 = a;                          //WILL NOT COMPILE                */
        int num = 5;                    //an integer num is assigned the state 5
        num = num + 10;                 //num is assigned to its old state + 10, so it changes to 15
        System.out.println("Num is " + num);     //will output Num is 15
        int value = 3;                  //value is assigned to 3
        value = value + num;            //value is assigned to its old state + num's state
                                        //note:  num does not change here
        System.out.println("Value is " + value);   //will output Value is 18
```

Arithmetic shortcuts:

The increment operators ++ and -- will add or subtract 1 to the previous state of any numeric variable.

```
a = 8;
a++;                    //x now stores the state 9.  Equivalent to a = a + 1;
num = 12;
num--;                  //num now stores the state 11.  Equivalent to num = num - 1;
```

The operators +=, -=, *=, /= and %= will perform an operation on the previous state of a variable with a value.

```
a = 3;
a += 10;        //equivalent to a = a + 10;
                //a is assigned to its old state 3 plus 10, so now a is 13.
b = 3;
b *= 2;         //equivalent to b = b * 2;
                //b is assigned its old state 3 times 2, so now it is 6.

num = 7;
num %= 2;       //equivalent to num = num % 2;
                //num is assigned to its old state 7 % 2, so now it is 1.
                //2 goes into 7 three times with a remainder of 1.
```

Getting input from the keyboard:

```
//in ThirdProgram.java
import java.util.*;     //the import statement makes useful tools available for your program
import java.io.*;
public class ThirdProgram
{
   public static void main(String [] arg)
   {
        Scanner input = new Scanner(System.in);         //used to get input from the keyboard
        double x, y, ans;                               //3 primitive variables of type double (real #)
        System.out.println("Enter a number");
        x = input.nextDouble();                         //waits for input and stores it in x
        System.out.println("Enter another number");
        y = input.nextDouble();                         //waits for input and stores it in y
        ans = x * y;
        System.out.println(x + " times " + y + " is " + ans);
   }
}
```

If the user enters in the value 2 for x, then enters 4 for y. The program will output: **<u>2 times 4 is 8</u>**

Given a Scanner object called input, the methods used for reading values include:

```
input.nextInt();        //for reading in whole numbers
input.nextDouble();     //for reading in numbers that might have a decimal
input.next();           //for reading in a String (any characters up to the first space or enter key)
input.nextLine();       //for reading in an entire line of text (a String that might include spaces)
```

We will get to the String objects later on...

```
import java.util.*;                                         //makes useful tools available
import java.io.*;
public class ThirdProgramB                                  //in ThirdProgramB.java
{
    public static void main(String [] arg)
    {
        Scanner input = new Scanner(System.in);             //used to get input
        String name;
        int age;
        System.out.println("Enter your 1st name");
        name = input.next ();                               //waits for input, stores it in name
        System.out.println("Enter your age");
        age = input.nextInt();                              //waits for input, stores it in age
        System.out.println(“Little " + name + " is already " + age + " years old!" );
    }
}
```

If the user enters in the value Doug for name, then enters 45 for age.

The program will output: **Little Doug is already 45 years old!**

Writing a program:
1) define any needed variables (one for each piece of input, one for each solution that must be found)
2) ask for the input and read it in (use System.out.println and the Scanner input object)
3) find the answer using assignment statements
4) show the answer (using System.out.println statements)

Consider that we want a program that will compute the mpg rating for a car after we take a trip. The user will need to enter the number of miles they drove and the number of gallons they used. That will require two variables of type double, since they might have decimals. The program will then take the user input and compute the mpg rating for the car, which will require another variable, also of type double.

So we need to define 3 variables of type double, ask for and read in the miles driven and gallons used, compute the solution and then show the solution to the user:

```
//in a file called FourthProgramA.java
import java.util.*;
import java.io.*;
public class FourthProgramA
{
   public static void main(String [] arg)
   {
      //1 - define any needed variables
      Scanner input = new Scanner(System.in);
      double miles, gallons, mpg;

      //2 - ask for the input and read it in
      System.out.println("How many miles driven?");
      miles = input.nextDouble();
      System.out.println("How many gallons used?");
      gallons = input.nextDouble();

      //3 - find the answer
      mpg = miles / gallons;

      //4 - show the answer
      System.out.println("Your car gets " + mpg + " miles per gallon.");
   }
}
```

When the program runs, if the user enters 250 for miles and 15 for gallons, the output would be
<u>Your car gets 17.2 miles per gallon.</u>

```
//in FourthProgramB.java, which will find the distance between two points
import java.util.*;
import java.io.*;
public class FourthProgramB
{
   public static void main(String [] arg)
   {     //1 - define any needed variables
         Scanner input = new Scanner(System.in);
         double x1, y1, x2, y2, dist;

         //2 - ask for the input and read it in
         System.out.println("Enter the first point x-coordinate");
         x1 = input.nextDouble();
         System.out.println("Enter the first point y-coordinate");
         y1 = input.nextDouble();
         System.out.println("Enter the second point x-coordinate");
         x2 = input.nextDouble();
         System.out.println("Enter the second point y-coordinate");
         y2 = input.nextDouble();

         //3 - find the answer
         dist = Math.sqrt(((x2-x1)*(x2-x1)) + ((y2-y1)*(y2-y1)));

         //4 - show the answer
         System.out.println("the distance is " + dist);
   }
}
```

The Math library has many useful functions, like:

```
Math.sqrt(x);        //returns the square root of x
Math.abs(x);         //returns the absolute value of x
Math.random();       //returns a random double between 0 inclusive and 1 exclusive
Math.sin(x);         //returns the sine of x (assumed that x is in radians)
Math.cos(x);
Math.tan(x);
Math.PI              //a constant value that returns PI (or a number really close to PI)
Math.min(x,y);       //returns the smaller of the two between x and y
Math.max(x,y);       //returns the larger of the two between x and y
```

Just like with a calculator, you will want to try to avoid the square root of a negative or dividing by zero.

Equality and inequality operators:

==	checks equality between two primitives
!=	checks to see if two primitives are not equal
<	checks to see if one primitive is less than another
>
<=	checks to see if one primitive is less than or equal to another
>=

Control structures - The if statement:

Used to make a block of code execute if a certain condition is true

```
if( /* a condition is true */)
{
   //execute the code here
}
```

The if-else statement:

Used to run one of two blocks of code depending on if a condition is true

```
if(/* a condition is true */)
{
   //runs the code here when the condition is true
}
else
{
   //runs the code here when the condition is false
}
```

The nested if statement - Used to run one of many blocks of code depending on many conditions

```
if(/* condition 1 is true */)
{
   //runs the code here when the first condition is true
}
else
   if(/* condition 2 is true */)
   {
      //runs the code here when the second condition is true
   }
   else
      if(/* condition 3 is true */)
      {
         //runs the code here when the third condition is true
      }
      else
      {
         //runs the code here when none of the conditions are true
      }
```

Note that if there is not a last-else statement at the end, there is a possibility that no code will run in the event that all the conditions are false.

```
//in FifthProgram.java, which will find the distance between two points
import java.util.*;
import java.io.*;
public class FifthProgram
{
    public static void main(String [] arg)
    {       //1 - define any needed variables
            Scanner input = new Scanner(System.in);
            double temp;

            //2 - ask for the input and read it in
            System.out.println("Enter the temperature in Fahrenheit");
            temp = input.nextDouble();

            //3 - find the answer (and in this case, that includes showing it at the same time)
            if(temp >= 90)
                System.out.println("Dress light - it is hot");
            else
                if(temp >= 70)
                    System.out.println("Dress light - it is warm");
                else
                    if(temp >= 50)
                        System.out.println("Dress regular");
                    else
                        System.out.println("Dress in layers - it is cold");
    }
}
```

If the user enters in the value **95**, the first condition will be true, and the program will output:

Dress light - it is hot

after which, it will skip the else (which contains the other if statements) and the program will end

If the user enters in the value **60**, the first and second conditions will be false, but the third condition will be true, and the program will output: **Dress regular**
after which, it will skip the else (which contains the last statement) and the program will end

If the user enters 28, all of the conditions will be false, and the program will default to the last else statement and output: **Dress in layers - it is cold**

Again, note that if the program did not have the last else statement and the user entered 28, the program would not have any output.

Commonly, begin and end braces are required to mark which block of code is to be executed for the if or else part of the structure. But if the code only consists of a single statement, the braces are not required. The code above could have been written as:

```
if(temp >= 90)
{
    System.out.println("Dress light - it is hot");
}
```

Control structures - the for loop:

A for loop is used to repeat a block of code a known number of times. A loop control variable is used to count how many times the loop repeats until a loop condition is no longer true.

```
for(int i=0; i < 5; i++)            //loop will repeat the loop body five times
{
    System.out.print("*");          //the System.out.print command will keep output on the same line
}
```

The variable i starts at zero, then it checks the condition. Since zero is < five, the loop body executes. Then i increments to the value one, and it checks the condition again. The loop will end once the condition is false. If i starts at zero, continues while i is < five and increases one at a time, then the loop body will repeat five times, and the output will be *****.

```
import java.util.*;
import java.io.*;
public class SixthProgram
{
    public static void main(String [] arg)
    {
        Scanner input = new Scanner(System.in);
        int num;

        System.out.println("Enter the number of stars you want");
        num = input.nextInt();

        for(int i=0; i<num; i++)
        {
            System.out.print("*");
        }
    }
}
```

If the user enters the value 8 when prompted, the program will output ********.

A for loop can go either direction, and the variable can increment in any way such that the condition eventually becomes false:

```
for(int i=0; i<=5; i++)      //will repeat 6 times because it starts at zero and ends at (and including) five.

for(int i=10; i > 4; i--)    //will repeat 6 times because it starts at ten and ends before reaching four.

for(int i=1; i<14; i+=2)     //will repeat 7 times because it starts at 1, ends before fourteen
                             //and i increases 2 at a time.

for(int i=0; i < 10; i--)    //an infinite loop, because i starts at zero, continues while i is < 10 and
                             //decreases by one each time, which will ALWASY be < 10.

for(int i=0; i > 10; i--)    //a dead loop, because i starts at zero and zero is not greater than ten.
```

Control structures - the while loop:
The while loop is a loop that is used to repeat a number of times that is unknown at compile time. It repeats a block of code while a condition is true.

```
while(/* condition is true */)
{
    //repeat the body of code here
}

int count = 0;
int num = 7842;
while(num > 0)          //while num has digits
{
    num /= 10;          //effectively knocks the least significant digit off of num, i.e. num = num / 10
    count++;            //add one to count
}
```

Since num starts at 7842, and 3842 is > 0, the loop body will execute.
num changes to 784 and count goes to 1.
Since 384 is > 0, the loop body executes again.
num changes to 78 and count goes to 2.
Since 38 is > 0, the loop body executes again.
num changes to 7 and count increases to 3.
Since 3 > 0, the loop body will execute one more time.
num changes to 0 and count increases to 4.
Zero is not > zero, so the loop ends.
count stores the number of digits in num.

```
public class SeventhProgram                     //assume import statements are added above
{
    public static void main(String [] arg)
    {
        Scanner input = new Scanner(System.in);
        double x, y, ans;
        int  again = 1;
        while (again == 1)                      //while the user wants to run the program again...
        {
            System.out.println("Enter a number");
            x = input.nextDouble();
            System.out.println("Enter another number");
            y = input.nextDouble();
            ans = x * y;
            System.out.println(x + " times " + y + " is " + ans);
            System.out.println("Enter 1 to run again, or 0 to quit");
            again = input.nextInt();            //allow the user to quit the program
        }
    }
}
```

Any for loop can be written as a while loop, but the compact nature of the for loop makes it more efficient when you know how many times a loop should repeat:

```
for(int i=0; i < 5; i++)          //first initialize i to its starting value (zero)
{                                 //while the condition is true (i < 5)
   System.out.print("*");         //complete the loop body and increment i (i++)
}
```

this loop is equivalent to...

```
int i = 0;                        //first initialize i to its starting value (zero)
while(i < 5)                      //while the condition is true (i < 5)
{
   System.out.println("*");       //complete the loop body
   i++;                           //increment i
}
```

Loops can also be called from inside of another loop. These are called nested loops.
The outside loop will repeat an inside loop a certain number of times.

```
for(int r=0; r<3; r++)            //the outside loop repeats the following three times...
{
   for(int c=0; c<4; c++)         //this loop writes four stars on the same line
   {
      System.out.print("*");
   }
   System.out.println();          //then goes down to the next line
}
```

A total of twelve stars will be written (4 stars on a line, repeated three times).
The output will be:

```
****
****
****
```

```
for(int r=3; r>0; r--)            //the outside loop repeats the following three times...
{
   for(int c=0; c<r; c++)         //this loop writes as many stars as is the state of the variable r
   {
      System.out.print("*");
   }
   System.out.println();          //then goes down to the next line
}
```

A total of six stars will be written (3 stars on a line, then 2 stars, then 1 star).
The output will be:

```
***
**
*
```

Nested loops are used to execute a lot of instructions with relatively little code.

Boolean operators: && (AND), || (OR) and ! (NOT)
Conditions can be composed using the three logical operators, &&, || and !.

The && operator (AND) only yields true when all conditions are true.

```
if(x >= 0 && y >= 0)
   System.out.println("DONE");
```

This will only output **DONE** in the case that x is positive AND y is positive.
If either or both conditions are false, the whole condition is false.

The || operator (OR) yields true if any condition is true.

```
if(x >= 0 || y >= 0)
   System.out.println("DONE");
```

This will output **DONE** in the case that x is positive or y is positive or both are positive.
Only if both conditions are false, the whole condition is false.

The ! operator (NOT) makes any boolean condition its opposite.

```
if(!word.equals("yes"))
   System.out.println("DONE");
```

This reads as: if it is **not** true that word equals "yes".
It will output **DONE** in the event that word is equal to anything except "yes".

&& has higher order of operation precedence over ||.
(A || B && C) is equivalent to (A || (B && C)).

Java will short circuit a boolean expression by ending a check once the result is known:

Given (A && B && C), it examines the conditions from left to right.
If A is false, it stops checking and returns false (because false && anything will yield false).
It will only examine the condition B if A is true.
If B is false, it stops checking and returns false.
It will only examine condition C if A is true and B is true. The result will then depend on C.

Given (A || B || C), it examines the conditions from left to right.
If A is true, it stops checking and returns true (because true || anything will yield true).
It will only examine the condition B if A is false.
If B is true, it stops checking and returns true.
It will only examine condition C if A is false and B is false. The result will then depend on C.

Demorgan's Theorem:
!(A && B) == !A || !B,   !(A || B) == !A && !B
Think of this like distribution of a negative into a polynomial: every term becomes its opposite, but with Demorgan's theorem, an && will switch to an ||, or the || will switch to an &&.

!(x > 0 && y<=14) is equivalent to (x<=0 || y>14).
Both conditions become their opposite, and the && operator changed to an ||.
Remember that the opposite of (greater-than) is (less-than-or-equal-to), not (less-than).

!(num >= 0 || word.equals("hello")) is equivalent to (num < 0 && !word.equals("hello"))

Error checking user input:

You can use a while loop to trap invalid user input until it is received as valid.
Consider that invalid input would be nonsensical values, like a negative age or a height of zero, or values that could cause the program to throw an exception, like division by zero or square root of a negative.

The model is:

1) ask for and read in user input
2) while(the input is bad)
```
 {
    tell the user the input is bad
    ask for and read in user input again
 }
```

```
public class EighthProgram                          //assume import statements are added above
{
   public static void main(String [] arg)
   {
        Scanner input = new Scanner(System.in);
        double x, ans;
        int  again = 1;
        while (again == 1)                          //while the user wants to run the program again...
        {
           System.out.println("Enter a number to find the square root of:");
           x = input.nextDouble();
           while( x < 0)                            //we will stay in the loop until the input is valid
           {
              System.out.println("Negative values are invalid.");
              System.out.println("Enter a number to find the square root of:");
              x = input.nextDouble();
           }
           ans = Math.sqrt(x);
           System.out.println("The square root of " + x + " is " + ans);
           System.out.println("Enter 1 to run again, or 0 to quit");
           again = input.nextInt();                 //allow the user to quit the program
        }
   }
}
```

Consider you are asking the user to pick one of many options:

```
int option = 0;
System.out.println("Pick an option (1, 2, 3, 4, or 5):");
option = input.nextInt();
while(option < 1 || option > 5)             //we will stay in the loop until the input is valid
{
   System.out.println("That option is invalid.");
   System.out.println("Pick an option (1, 2, 3, 4, or 5):");
   option = input.nextInt();
}
```

Methodizing:
When a complex task can be broken down into logical subtasks, they are written as methods that can streamline the code, be called many times, or reused in multiple applications. Static methods are subtasks in which there is only one version that does not require creating an instance of an object. They may return a value, like the way square root returns a number with a decimal. Or they may be a void method, which performs a task, but does not return a value. They may require parameters (arguments) to work , like the way square root requires that you send it a number, or they may not need any arguments sent.

```
public class NinthProgram                      //assume import statements are added above
{
   public static void showMenu()               //this method only shows options on the screen
   {                                           //so it does not need arguments or return a value
      System.out.println("type 1 to find the area of a right triangle");
      System.out.println("type 2 to find the perimeter of a right triangle");
      System.out.println("type 3 to find the area of a rectangle");
      System.out.println("type 4 to find the perimeter of a rectangle");
   }

   public static double findTriangleArea(double base, double height)
   {                                           //this method finds the area of a right triangle
      return 0.5*base*height;                  //so it needs information to do its job (parameters)
   }                                           //and returns a value as a number with a decimal

   //assume similar methods called findTrianlgePerim, findRectangleArea and others are defined here
   public static void main(String [] arg)
   {
      Scanner input = new Scanner(System.in);
      double base, height, ans=0;
      int option;
      System.out.println("Enter the base:");
      base = input.nextDouble();               //assume error checking is done for the base here
      System.out.println("Enter the height:");
      height = input.nextDouble();             //assume similar error checking is done for the height
      showMenu();                              //calls the showMenu method defined above
      option = input.nextInt();                //assume error checking is done for the option here
      if(option == 1)
         ans = findTriangleArea(base, height); //calls the methods defined above, and the returned
      else if(option == 2)                     //value is stored in the variable ans.
         ans = findTrianglePerim(base, height);
      else if(option == 3)
         ans = findRectangleArea(base, height);
      else if(option == 4)
         ans = findRectanglePerim(base, height);
      System.out.println("The solution you seek is: " + ans);
   }
}
```

Exercise: complete this as a working program with all methods and error checking defined, and test it.

Let's look more closely at some of these methods:

```
public static void showMenu()                       //this line is called the method header
{
   System.out.println("type 1 to find the area of a right triangle");
   System.out.println("type 2 to find the perimeter of a right triangle");
   System.out.println("type 3 to find the area of a rectangle");
}
```

This method merely writes information out to the screen. Therefore, it does not require that we send it any information as arguments in the parenthesis.
The result of performing the commands to show the user the available options, there is no part of the process that requires finding a particular solution, so it is defined as a void method. The term void means that the method will not return a value (like the main function).

If the main objective of a subtask is to find a solution of some kind, then it will be a return method. In the method header, the return type is declared before the method name.

```
public static double findTriangleArea(double base, double height)          //method header
{
   return 0.5*base*height;
}
```

Note that in order to find the area of a right triangle, we need to know some information to complete the task: the dimensions of the triangle. So we have parameters declared in the parenthesis to state what information is required to complete the task. The base and height might have decimals, so they are declared as type double (seen to the right of the method name). The solution that is returned will also have a decimal, so the return type is also declared as double (seen to the left of the method name).

Calling a void method only requires stating the method name, and sending it any needed arguments. Since showMenu does not require any arguments, all we have to do is type:

```
showMenu();
```

Calling a return method should usually be done in such a way that we use or can retrieve the value that is returned. Send it any needed arguments, and call it in an assignment statement, a println command (to write out the returned value) or in a condition.

For the method findTriangleArea, we need to send it two numbers in order for it to complete its task. Here are different ways of calling it such that we use the returned value in some way:

```
//calling a return method in a println
System.out.println( "Area is " + findTriangleArea(3, 4));            //will output Area is 6.0

//calling a return method in an assignment statement
double ans = findTriangleArea(3, 4);                                 //returned value is saved in ans

//calling a return method in a condition
if (findTriangleArea(3, 4) < 10)
   System.out.println("Small triangle");
else
   System.out.println("Big triangle");                               //will output Small Triangle
```

Variable scope:

Variables only exist within the construct in which they are defined. If a variable is defined inside of a method, then it only exists within the method. If a variable is defined inside of a loop or if statement, then the variable no longer exists once the body of code for that construct ends. Consider this:

```
        //within a method...
        int x = 5;
     /* int x = 3;          //WILL NOT COMPILE, because x is already defined!      */

        //BUT, now consider the following:
        if(/* condition is true */)
        {
           int x = 5;                     //x begins its existence here
           System.out.println(x);
        }                                 //x no longer exists when the if-body ends
        else
        {
           int x = 3;                     //we can define another variable called x,
           System.out.println(x);         //because the old one is gone
        }                                 //x no longer exists when the else-body ends
```

It is because of variable scope that you can have disjoint for loops with the same loop variable name:

```
        for(int r=0; r<3; r++)            //r begins to exist here
        {
      /*   for(int r=0; r<4; r++)         //WILL NOT COMPILE - r is already defined for the loop body*/
           {
              //some code here
           }
        }
        //BUT, the following would work
        for(int r=0; r<3; r++)            //variable r begins to exist here
        {
           //loop body
        }                                 //variable r ends existing here
        for(int r=0; r<4; r++)            //a new variable r begins existing here
        {
           //loop body
        }                                 //the new variable r ends existing here
```

If you define multiple variables in different methods with the same name, it is important to note that they are not the same variable.

```
        public static void method1(int x)
        {
           String word = "hello";
        }
        public static void method2(int x)          //this x is entirely different from the x in method1
        {
           String word = "world";                  //this word has nothing to do with the word in method1
        }
```

The String object:
The String is a data type that can store a collection of characters, like a word or sentence, and can perform actions that either reveal information about the string or return new strings.

Creating a string and assigning it to a literal value requires quotation marks.

```
String word = "hello";
System.out.println(word);        //will output hello
```

The Scanner object can read in a string in one of two ways:

```
Scanner input = new Scanner(System.in);
System.out.println("Enter a word:");
String word = input.next();
```

The input.next() method will wait for the user to enter text and press enter. It will return a string comprised of the characters the user typed in up to the first occurrence of a space or carriage return.

```
System.out.println("Enter a sentence:");
String word = input.nextLine();
```

There is another Scanner method called input.nextLine(), which will return a String that can include spaces within it. This is what you would use if you wanted the user to type in a sentence, or someone's full name.

String methods:

```
int length()                          //returns the number of characters in a String
int indexOf(String part)              //returns position of part in the string, returns -1 if not found
String substring(int start)           //returns a new String from index start to the end of the String
String substring(int start, int end)  //returns a new String from index start to index (end-1)
String toUpperCase()                  //returns a new String of all upper-case characters
String toLowerCase()                  //returns a new String of all lower-case characters
```

Calling a method from a string requires using dot-notation. It is important to note that once a string has been created, it can't be changed unless you assign it to a different String. For example:

```
String word = "hello";
System.out.println(word.toUpperCase());     //will output HELLO
System.out.println(word);                   //will output hello
```

The method .toUpperCase() merely returns a new String that is then sent to a println statement. The original String word is unchanged. If you wanted to change word to its upper-case version, you would need to reassign it to a new String:

```
String word = "hello";
word = word.toUpperCase();                  //word changes to its upper-case version
System.out.println(word);                   //will output HELLO
```

.indexOf(String part) //returns the position of part within the string, returns -1 if not found

Each character in a String is stored at a unique location called an index. The first character is stored at index 0 and the last character is stored at the String's length - 1.

String word = "hello";

index:	0	1	2	3	4
name:	h	e	l	l	o

word.indexOf("h")	returns 0, since the "h" is the first character in the String
word.indexOf("e")	returns 1, since the "e" is the second character in the String
word.indexOf("o")	returns 4, since "o" is the fifth character in the String
word.indexOf("z")	returns -1, since "z" is not found in the String
word.indexOf("H")	returns -1, since there is no capital "H" in the String (case sensitivity)
word.indexOf("hell")	returns 0 because the substring "hell" is found starting at index 0 of the String
word.indexOf("llo")	returns 2 because the substring "llo" is found starting at index 2 of the String

The indexOf method can be used as a search engine for a String, since it returns -1 if the argument is not found.

.substring(int start) //returns a new String from index start to the end of the String
.substring(int start, int end) //returns a new String from index start to index (end-1)

Consider the following String and each character's index

String name = "Jefferson";

index:	0	1	2	3	4	5	6	7	8
name:	J	e	f	f	e	r	s	o	n

name.substring(3)	returns "ferson"	- the substring from index 3 all the way to the end
name.substring(3,7)	returns "fers"	-the substring from index 3 to index 6 (7-1)
name.substring(6)	returns "son"	-the substring from index 6 all the way to the end
name.substring(6,8)	returns "so"	-the substring from index 6 to index 7 (8-1)
name.substring(0)	returns "Jefferson"	-the substring from index 0 all the way to the end
name.substring(0,4)	returns "Jeff"	-the substring from index 0 to index 3 (4-1)

Note that for the two argument version of substring, we are going from index start (inclusive) to index end (exclusive), so effectively from index (start) to index (end-1).

name.substring(0, word.length() / 2) returns "Jeff" because name.length() is 9, and 9/2 is 4.
So we take the substring from index 0 to index 3.

name.substring(3, word.length() - 3) returns "fer", since name.length() is 9, and 9-3 is 6.
So we will have the substring from index 3 to index 5 (6-1).

int index = name.indexOf("er"); //"er" is found at index 4
name.substring(index) returns "erson" because index will store the state 4.
So we take the substring from index 4 all the way to the end of the String

name.substring(0, index) returns "Jeff", the substring from index 0 to index 3 (4-1).

Strings can be added together using the + operator. This is called concatination.

```
String word1 = "ABC";
String word2 = "123";
String combined = word1 + word2;
System.out.println(combined);                  //will output ABC123
combined = combined.toLowerCase() + "!!!";     //change combined to its lower case version plus "!!!"
System.out.println(combined);                  //will output abc123!!!
```

```
//here is a program that will ask the user to enter their name in the format <first-name last-name>
//and output it in the form <last-name, first-name>
import java.util.*;
import java.io.*;
public class TenthProgram
{
   //pre:  name is in the format <first-name last-name>
   //post: returns the name in the format <last-name, first-name>
    public static String formatName(String name)
    {
        int index = name.indexOf(" ");                //search for position of the space within the name
        String first = name.substring(0, index);   //from the 1st character to the last one before the space
        String last = name.substring(index+1);    //from the character after the space all the way to the end
        return last + "," + first;
    }

    public static void main(String [] arg)
    {
            Scanner input = new Scanner(System.in);
            String name, ans;
            System.out.println("Enter your first and last name separated by a space");
            name = input.nextLine();
            while (name.indexOf(" ") == -1)  //can't find a space in the name, so error check
            {
                System.out.println("Invalid input - no space found:");
                System.out.println("Enter your first and last name separated by a space");
                name = input.nextLine();
            }
            ans = formatName(name);
            System.out.println(ans);             //shows the name in the format of <last-name, first-name>
    }
}
```

A common error with the Scanner object occurs when you call the nextLine() method call after reading in a number with nextInt() or nextDouble(): it would appear to skip the nextLine() command. If this happens, just add an extra nextLine() command right before the nextLine() that is being skipped.   It will clear the keyboard buffer and allow the String to be read in.

Casting primitives:
You can take most primitive types and return them in a different form, such as dropping the decimal from a double to get the whole number part, or forcing an int to be a double so that you can use a division operator as opposed to div (whole number division):

```
double num = 4.8;
System.out.println( (int)(num));          //casts 4.8 into an integer, which will output 4
```

Note that the example above did not round up:
Casting a double into an int will truncate the number, which effectively knocks off the decimal part.

```
int value = 9;
System.out.println( (double)(value));     //casts 9 into a double, which will output 9.0
```

```
int x = 3, y=6;
System.out.println( x / y);               //will output 0, because 6 goes into 3 zero times
System.out.println( (double)(x) / y);     //casts x into 3.0, so now this will divide and output 0.5
```

Using Math.random() :
Math.random() returns a double between 0 inclusive and 1 exclusive. So it can be as low as zero and as large as almost one, or 0.999999999

You can easily make a program flip a coin by checking to see what Math.random returns.

```
if(Math.random() < .5)                    //this will be true 50% of the time
   System.out.println("HEADS");
else
   System.out.println("TAILS");
```

You can shape a random value to get an integer between a range of values, say from min to max:
The algorithm is:

1) get a random number between 0 and almost 1.
2) multiply it by the range of values, or the number of random values you want.
   this will be (max - min + 1)
3) cast the resulting product into an integer to drop the decimal
4) add the min value to the result

This will give you a random integer between min and max inclusive.
As a single assignment statement, it would be:

```
int ran = (int)(Math.random() * (max - min + 1)) + min;
```

Examples:
the result of a single die roll would be: (int)(Math.random()*6) + 1
1 is our min and 6 is our max.

a random number between 0 and 9 would be: (int)(Math.random()*10)

a random number between 15 and 20 would be: (int)(Math.random()*6) + 15
Note that there are six numbers between 15 and 20 inclusive (20 - 15 + 1).

Arrays:
an array is a single entity that can store many values, each at its own unique index.
We have seen this before - a String object contains an array of characters.
Creating an array requires stating the type of values it will store, giving it a valid identifier name and stating how many elements we want to store.

```
int[] numbers = new int[5];       //creates an array called numbers that can store five integers.
double[]vals = new double[8];     //an array called vals that can store eight real numbers.
String[]words = new String[50];   //an array called words that can store 50 String objects.
```

It would be expected that we fill the arrays somewhere after assigning it.

You can also create an array and immediately assign values to it.

```
int[] nums = {5, 3, 7, 0, 9};     //nums stores six integers:  5, 3, 7, 0 and 9
String[] names = {"Bob", "Otto", "Anna"};
                                  //names stores three String objects:  Bob, Otto and Anna
```

Each element of the array can be accessed or changed by its index. Like a String, the first element is stored at index zero, and the last element is stored at the array's length - 1.

Each array has a data field that stores the number of elements called length.

```
int[] nums = new int[5];
nums[0] = 5;
nums[1] = 3;
nums[2] = 7;
nums[3] = 0;                      // note that the array could have been created this way:
nums[4] = 9                       // int[] nums = {5, 3, 7, 0, 9};
```

The array is visualized as:

index:	0	1	2	3	4
value:	5	3	7	0	9

System.out.println(nums[2]); //would output **7**, because the seven is stored at index two
System.out.println(nums[2] - nums[1]); //would output **4**, because 7 - 3 is four.
System.out.println(nums.length); //would output **5**, because there are five elements in the array

Note that calling the length data field for an array does not need parenthesis afterwards, where it does for a String object. The length for an array is a data field, where the length() of a String calls a method.

Traversing through every element of an array can be easily done with a for loop to access every index.

```
for(int i=0; i < nums.length; i++)       //i will traverse through each valid index of nums
   System.out.print(nums[i] + " ");      //will output 5 3 7 0 9
```

The print statement differs from the println statement in the following way:
A print statement will keep the cursor on the same line after it has completed its command, so multiple consecutive calls to the print statement will all be output on the same line.
A println statement will send the cursor to the next line after it has completed its command, so multiple consecutive calls to the println statement will result in each output being printed on its own line.

```
//here is a modified version of TenthProgram that will work for five names
public class TenthProgramB                          //assume needed import statements are included
{
   //pre:  name is formatted  <first-name last-name>
   //post: returns the name in the format <last-name, first-name>
   public static String formatName(String name)
   {
      int index = name.indexOf(" ");                //search for position of the space within the name
      String first = name.substring(0, index);   //from the 1st character to the last one before the space
      String last = name.substring(index+1);    //from the character after the space all the way to the end
      return last + "," + first;
   }

   //post:  fills array with user chosen names in the format <first-name last-name>
   public static void fillArray(String [] names)
   {
      Scanner input = new Scanner(System.in);
      for(int i=0; i<names.length; i++)                //note that this will work with an array of any size
      {
         System.out.println("Enter your first and last name separated by a space");
         names[i] = input.nextLine();
         while (names[i].indexOf(" ") == -1)         //can't find a space in the name, so error check
         {
            System.out.println("Invalid input - no space found:");
            System.out.println("Enter your first and last name separated by a space");
            names[i] = input.nextLine();
         }
      }
   }

   //post:  displays each element of array, one element per line
   public static void showArray(String [] array)
   {
      for(int i=0; i<array.length; i++)                //note that this method will work with an array of
         System.out.println(array[i]);                 //any size, not just the one defined in the main function
   }

   public static void main(String [] arg)
   {
        String [] names = new String[5];             //An array of five Strings.
        fillArray(names);                            //Calls the method above to fill with user input.
        for(int i=0; i<names.length; i++)            //Traverse through each index of the names array and
           names[i] = formatName(names[i]);    //have each name change to its formatted version.
        showArray(names);                            //Calls the method above to show all array elements.
   }
}
//NOTE: if we wanted to change the program to work for an array of 8 names, we would only have to
//change one character in the main function (in defining the array size).
```

Two Dimensional Arrays:
You can store data in a row-column oriented chart using a two-dimensional array. Creating one is similar to a regular array, but you must specify the number of rows and number of columns.

```
int[][] nums = new int[3][5];             //creates a 3 x 5 array called nums that can store 15 integers.
double[][]vals = new double[8][8];        //vals  can store 64 real numbers in 8 rows and 8 columns.
String[][]words = new String[10][5];      //words that can store 50 Strings in 10 rows and 5 columns.
```

Accessing any 2-D array element requires first specifying the row index, then the column index.
Given a 2-D array called chart, the number of rows is specified by chart.length and the number of columns is returned by chart[0].length. Why? Because compositionally, a 2-D array is built as an array of arrays.

```
int [][] chart = new int[2][3];               //6 integers arranged in 2 rows and 3 columns
chart[0][0] = 5;
chart[0][1] = 9;                              //chart at row 0, col 1 is assigned the state nine
chart[0][2] = 4;
chart[1][0] = 7;                              //chart at row 1, col 0 is assigned the state seven
chart[1][1] = 0;
chart[1][2] = 6;
for(int r = 0; r < chart.length; r++)         //r traverses through each row index (chart.length will be 2)
{
   for(int c = 0; c < chart[0].length; c++)  //c traverses through each column index (chart[0].length is 3)
   {
      System.out.print(chart[r][c] + " ");
   }                                          //would output:        5 9 4
   System.out.println();                      //                     7 0 6
}
```

Remember that the print statement keeps the output on the same line, so each row element will be displayed on a single line. After the inner loop completes showing every row element, the println statement is used to drop the cursor to the next line for the next row.

Note that we use a nested for loop to traverse through each row and each column.
Let's say we wanted to find the sum of all of the values in chart:

```
int sum = 0;
for(int r = 0; r < chart.length; r++)         //r traverses through each row index
{
   for(int c = 0; c < chart[0].length; c++)  //c traverses through each column index
   {
      sum = sum + chart[r][c];                //add each array element to the previous state of sum
   }
}
System.out.println(sum);                      //would output 31, the sum of 5+9+4+7+0+6
```

Notice that the nested for loop used to show all of the array elements is the same as the one used to find the sum. The only difference is how each array element is handled.

Creating objects:
When you want to define a new kind of object, you need to consider what information you want it to store (data fields) and what abilities you want it to have (methods). Consider that we want a Car object that stores the car's name and price tag:

```
//in Car.java
public class Car
{
   private String name;                  //data fields
   private double price;

   public Car(String n, double p)        //METHOD: constructor
   {
      name = n;
      price = p;
   }

   public String toString()              //METHOD
   {
      return "MODEL:" + name + "     PRICE TAG: $" + price;
   }

   public String getName()               //METHOD: accessor
   {
      return name;
   }

   public double getPrice()              //METHOD: accessor
   {
      return price;
   }

   public void setPrice(double p)        //METHOD: mutator
   {
      price = p;
   }
}
```

The visibility modifier called **private** means that the data fields name and price are only directly accessible from within Car.java. No other program has access to them.

The **constructor** method is called when we create an instance of a Car, and it assigns starting values to the data fields. For example, in some other program within the same folder, we might do this:

```
Car coup = new Car("Civic", 18000);
```

This will call the constructor, sending the String "Civic" in for the name, and the number 18000 in for the data field price.

The **toString** method is used to return information we want to see about the Car object as a String. It is called automatically when we send an instance of a Car to a print or println statement.

```
System.out.println(coup);          //will output MODEL: Civic     PRICE TAG: $18000
```

Accessor methods are used to return a data field that is private in the object's definition. The methods getPrice() and getName() are accessor methods for the Car object. If we want to see a Car's name or price from some driver program outside of Car.java, we can't access them directly outside of the class definition. But we can call an accessor method that returns the value we want:

```
        Car suv = new Car("CRV", 24000);
        System.out.println(suv);                //will output MODEL: CRV     PRICE TAG: $24000
/*      System.out.println(suv.price);    WILL NOT COMPILE, because price is private to Car.java */
        System.out.println(suv.getPrice());     //will output 24000
```

Mutator methods are used to change a data field that is private for the object. The method setPrice(x) is a mutator method for the Car object. Let's say we want to put a car on sale and drop its price.

```
        System.out.println(suv);                //will output MODEL: CRV     PRICE TAG: $24000
/*      suv.price = 22800;                WILL NOT COMPILE, because price is private to Car.java */
        suv.setPrice(22800);
        System.out.println(suv);                //will output MODEL: CRV     PRICE TAG: $22800
```

Note that there is not a mutator method for the name of the car. Why? It is a simple design decision: it seems more likely that we might change the price of a car after it has been created, but certainly not likely that we would change its name.

```
public class EleventhProgram     //assume this is in the same folder as Car.java
{
   public static void main(String [] arg)
   {
        //create three instances of Car objects
        Car pickup = new Car("F150", 26000);
        Car suv = new Car("CRV", 24000);
        Car coup = new Car("Civic", 18000);
                                                    //will output
        System.out.println("INVENTORY:");           //INVENTORY
        System.out.println(pickup);                 //MODEL: F150    PRICE TAG: $26000
        System.out.println(suv);                    //MODEL: CRV     PRICE TAG: $24000
        System.out.println(coup);                   //MODEL: Civic   PRICE TAG: $18000

        System.out.println("All SUVs price drop 5%");   //All SUVs price drop 5%
        suv.setPrice(suv.getPrice() * 0.95);
        System.out.println(suv);                    //MODEL: CRV     PRICE TAG: $22800
   }
}
```

Notice the line that says: suv.setPrice(suv.getPrice() * 0.95);
We go to the inner most parenthesis first and call suv.getPrice(). This returns 24000, which is then multiplied by 0.95. The result which is 22800 is then sent to the setPrice method to change the price of the suv.

Subclasses and inheritance:
From any base class, you may define a subclass that inherits all data fields and concrete methods from the base class by using the reserved word **extends**. The only items that are not inherited from a base class are constructors and any abstract methods (which we will get to later).

```
//in the same folder as Car.java
public class RentalCar extends Car          //a RentalCar now has all features and abilities of a Car
{
   private double pricePerDay;              //a new data field in addition to the ones inherited

   public RentalCar(String n, double p, double ppd)
   {
      super(n, p);                          //calls the constructor for Car and sets the name and price
      pricePerDay = ppd;                    //initializes our extra data field
   }

   public String toString()                 //override the toString that is inherited with a new version
   {
      return super.toString()+ "     RENTAL FEE: $" + pricePerDay;
   }

   public double getPricePerDay()           //a new accessor method in addition to the ones inherited
   {
      return pricePerDay;
   }

   public void setPricePerDay(double ppd)
   {                                        //a new mutator method in addition to the ones inherited
      pricePerDay = ppd;
   }
}
```

When we say that a RentalCar extends Car, it means that a RentalCar is a Car. It now inherits the data fields name and price, as well as the methods getPrice, setPrice, getName and toString (which the RentalCar will override with its own version).

In the constructor for the RentalCar, you see the reserved word **super**, which is a call to the base class Car. When invoked in the RentalCar's constructor, it calls the constructor for the Car and sets its name and price data fields to the values of the first two arguments sent into the RentalCar's constructor.

We also see the term super used in the RentalCar's version of toString. When a subclass has a method with the exact same header as one that it inherits from the base class, the subclass version will **override** the one that it inherits. The command super.toString() calls the toString method from the base class Car, which returns the Cars name and price. Then we add onto that String the cost of our rental car's price per day.

We do not inherit constructors. It is expected that when you create a new type of object that you define a constructor for it.

If we wanted, we could make a subclass of RentalCar, which would inherit all properties and abilities from the RentalCar and the Car.

```
public class EleventhProgramB   //assume this is in the same folder as Car.java and RentalCar.java
{
   public static void main(String [] arg)
   {
        Car suv = new Car("CRV", 24000);
        RentalCar coup = new RentalCar("Civic", 18000, 24.5);
                                              //will output
        System.out.println(suv);              //MODEL: F150    PRICE TAG: $26000
        System.out.println(coup);             //MODEL: Civic    PRICE TAG: $18000    RENTAL FEE: $24.5

        System.out.println(suv.getPrice());                  //will output 24000
/*      System.out.println(suv.getPricePerDay());            WILL NOT COMPILE:  suv is not a RentalCar */
        System.out.println(coup.getPrice());                 //will output 18000
        System.out.println(coup.getPricePerDay());           //will output 24.5
   }
}
```

Note that the coup is a Car AND a RentalCar, but the suv is just a Car object. So we may call the RentalCar method getPricePerDay() from the coup, but not from the suv.