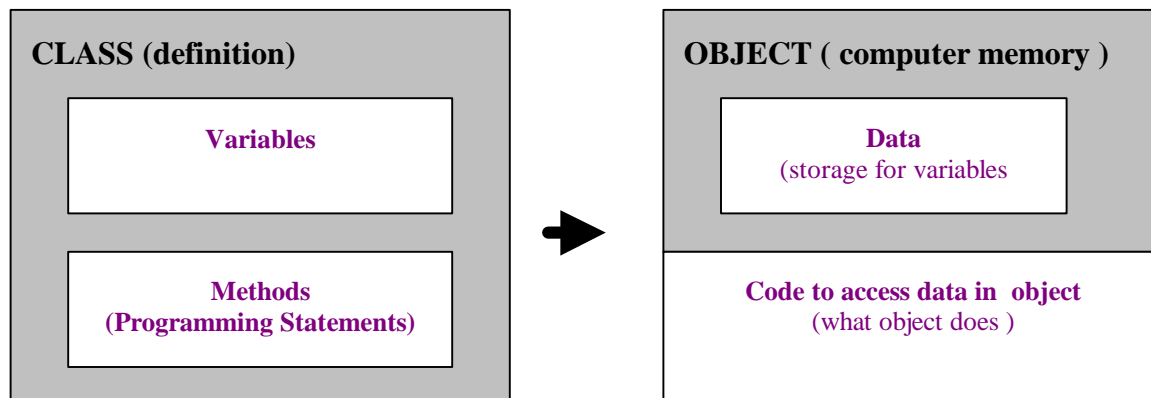


## JAVA PROGRAMMERS GUIDE LESSON 1

File:	JavaL1.doc
Date Started:	July 12,1998
Last Update:	Dec 28, 2005
Java Version:	1.5

### INTRODUCTION

This manual introduces the Java programming language techniques and concepts. Java is a new language designed for Internet programming. When you connect to a WEB page a Java **applet** may be downloaded to your browser to start a special effect on your screen. An **applet** is a program written in Java. The Java language has built in interfaces to the Internet. The Java language has now found it way into the University environment. It is a great teaching language, it is rich in graphics and good language constructs. Java uses the **Object Oriented Programming (OOP)** approach. This approach uses a programming language to describe an object. What is an Object? An Object is an everyday thing, a car, house, table or even a person. Every object has an appearance and does something. A car has a shape, a color and moves. The object oriented programming approach, wants to represent everyday objects in a programming language. How can it do this? A programming language can store the appearance of an object as a value in the computer memory. A programming language also has **programming statements** that are instructions telling the computer what the object does. When a computer stores information about an object, the stored information is known as **data**. The data is information that describes the objects and is stored in the computer memory. The location where the data is stored is represented by a **variable** having a name. A programming statement is used to describe what an object does and is translated into **code**. Code is used to run on the computer to perform some action or operation for the object. The code is the translated instructions telling the computer what to do. The programming language groups programming statements together into what is known as a **method**. A method is a collection of programming statements that get translated into code and runs on your computer. Before you can have an object you need a definition of the object. A **class** is used to define an object. The class lists all the methods and variables needed by the **object**. A **class** defines the variables for the object and **the code used** to access the **data** in the object. The same code can be used for many of the same objects. Think that a class is the drawings or a plan to build a house. When the house is made then it is an object. When you class runs on the computer it is an object. An object is allocated memory for the data variables defined in the class.



Objects are created in the computer memory from the class definitions. The class definitions tell what data and methods your object needs. Memory in the computer is allocated for the variables defined in the class definition. It is these created objects that use computer memory for **data** when you run your Java program. The Object Oriented Programming approach will enable you to organize your programs so that your programming tasks will be much easier to accomplish. Answer this question: What is the difference between a class and an object?

This manual teaches by using the **analogy** approach. This approach makes it easier for you to understand new concepts by comparing the operation to common known principles. We also use the **seeing** approach, the words we use, are carefully chosen so that you get a picture of what is happening. **Visualization** is a very powerful concept to understanding programming. Once you get the picture of what is happening then the programming concept is much easier to understand and accomplish. It is very important to visualize and see things as a picture rather than trying to understand by reading and memorizing textbooks. Pictures are made up of words. This document does not cover all details of Java programming but acts as a guide so that you can get started right away with Java programming. The main purpose of this document is to introduce, and teach you Java and Object Oriented Programming techniques. You should have a textbook as a companion to look up important terms for clarification or for more in depth study. Good textbooks are:

Title	Author(s)	Publisher	Web page
Core Java Volume 1	Gary Cornell, Cay s. Horstmahn	SunSoft Press	<a href="http://www.horstmann.com/corejava.html">http://www.horstmann.com/corejava.html</a>

## PURPOSE OF PROGRAMMING LANGUAGES

The purpose of a programming language is to direct a computer to do what you want it to do. In most cases of beginners the computer tells the programmer what to do. Good programmers are in control of the computer operation. A computer program lets the user enter data from the keyboard, perform some calculation to solve some problem and then displays the results on the computer screen. The user can store input and output data on a data file for future use.

## LESSONS AND ASSIGNMENTS

You may use these lessons as a study guide or to learn java programming. There are 14 lessons in the Java course, where the last lesson is a java project. Each lesson consists of exercises that you can send in for marking. You should understand all the material and do all the exercises in each lesson before proceeding to the next lesson. There is a marking charge of \$15 for each lesson. Additional java tutoring is available from \$15 and up. Grading is based on the assignments. (P) Pass, (G) Good and (E) Excellent. Excellent is awarded to students with outstanding programs that involve creativity and works of genius. Good is awarded to students that have exceptional working programs. Pass is awarded to students that have minimal working programs. Important concepts and Java keywords are introduced in **bold**. Keywords are the words a programming language is made up of. Language definitions are in *purple italics* and statements are in **blue** and comments in **green**.

Send all solutions to assignments to [students@cstutoring.com](mailto:students@cstutoring.com) (\$15 marking charge)

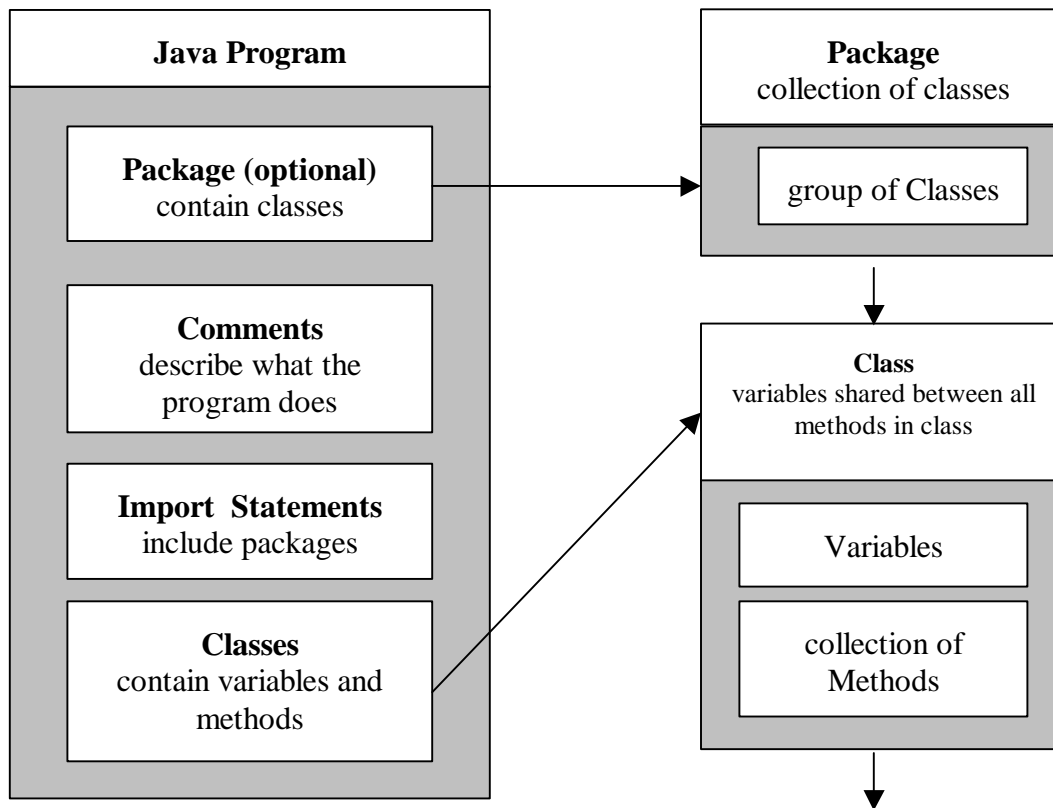
Java tutoring by email or telephone is also available from \$15 per hour.

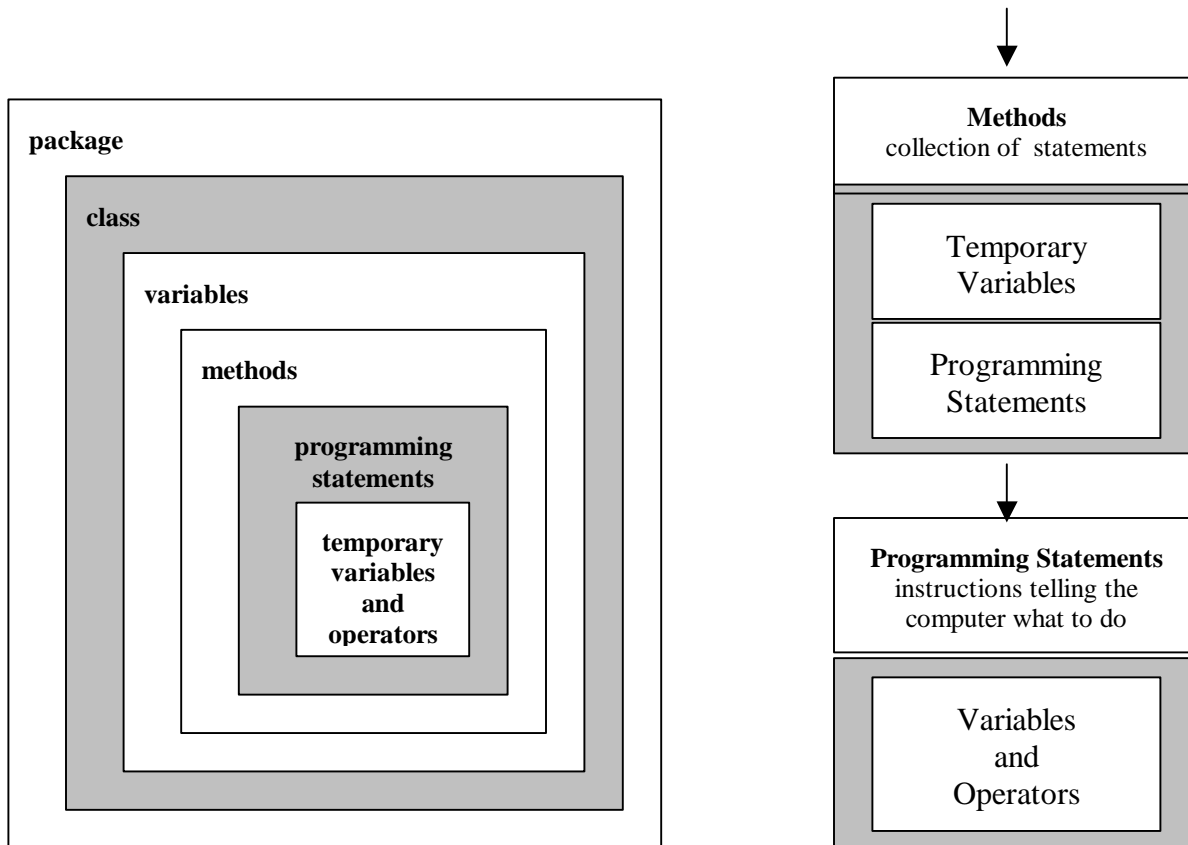
Send all programming questions and the Java code you have started to [students@cstutoring.com](mailto:students@cstutoring.com) for a quote (tutoring starts from \$15)

## LESSON 1 JAVA PROGRAM COMPONENTS AND VARIABLES

### JAVA PROGRAM FORMAT AND COMPONENTS

A Java program is made up programming components usually arranged in a predefined format. By following a proven format, your programs will be more organized and easier to read. A Java program contains **packages, import statements, classes, and methods, programming statements, variables** and **operators**. **Packages** are used to groups **classes** together that have something in common. **Import statements** are used to include system or user packages that contain classes that you may need for your program. **A class** declares and defines the **variables** and **methods** needed by an **object**. **A class** defines an **object**. **Variables** are used to **store** and **retrieve** data values. **Methods** are what the class object uses to perform the tasks it needs to do. **Methods** are made up of **variables** and programming **statements**. Programming **statements** are instructions used to direct the flow of program execution. **Statements** are made up of **variables** and **operators**. **Operators** perform operations on variables like addition and subtraction. Think that a Java program has many different layers each depending on each other: A Java program is an organized collection of components.

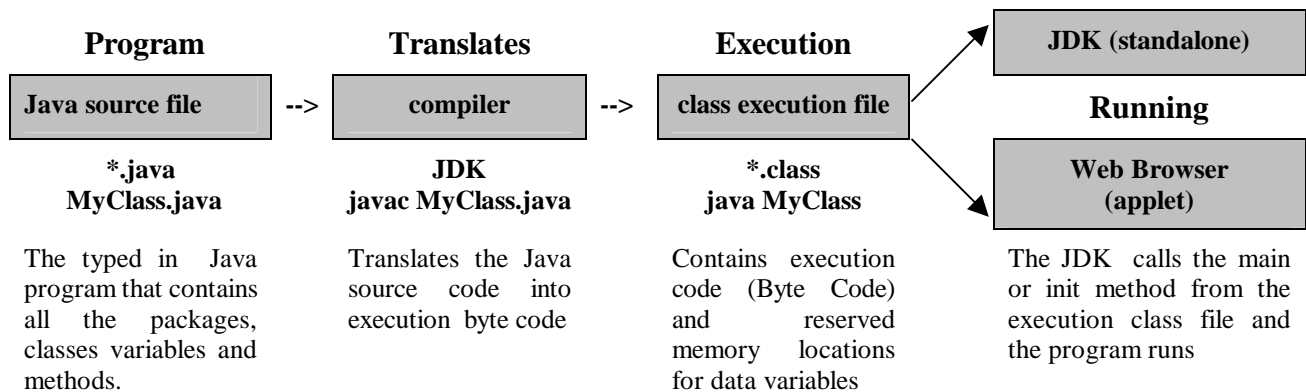




We will study each component individually. You do not need to understand every thing right now. The most important thing is that you are introduced to all the **terminology** and the **components** that make up a Java program. Analogy to a Java program is what cities and towns are made up of. Towns are made up of neighborhoods, neighborhoods are made up of streets, streets are made up of homes, homes are made up of people etc. A minimal Java program has only one class. A Java program may be written as an **applet** or a **non-applet**. Applets are Java programs that run on your web browser. Non-applet programs are known as **standalone** applications that run directly on your computer. In the first few lessons, we will deal with non-applet programs. We will deal with applets in future lessons. For a non-applet Java program a minimal class has one method called the **main method**. The **main method** is the first method to run in a Java standalone program. When a Java program **runs** it means it is executing programming statements contained in a method. You run a Java program by executing a class file that contains a main method. I call this class the "**running class**", the class that starts your Java program. The Java program that you type in is stored in a file having the **java** extension. The file that has the class with the main program in it should have the same name as the class name. Each class is usually kept in a separate file having the same name as the class. The file name and class name is case sensitive meaning upper and lower case characters are considered different. Before a Java program can run it must be compiled to an intermediate execution file made up of byte codes having the extension "**\*.class**". If your Java program has been written as an applet then it is this class file that runs on your internet browser. Java programs not written as an applet are run on your computer using the Sun Microsystems **JDK**. The JDK interprets the byte code. The interpreter is known as a virtual machine (**VM**). This means Java can run on any computer machine that has the JDK.

## JDK (Java Development Kit)

The **JDK** is used to compile a Java source program into an execution class file so that your program can run. The class file is run directly on your computer using the JDK. You can run applets on your computer without a web browser by using the JDK **applet viewer**. When your program is running the byte codes are interpreted by the JDK. An Integrated Development Environment (**IDE**) is a complete Java development system. Most IDE'S have the JDK built into them. There are many Java Integrated Development Environments on the market from IBM, Sun Microsystems, JCreator, and Borland.



## Downloading JDK

To run java programs you should download the JDK from Sun Microsystems. They named the JDK J2SE 5.0 this is the newest version.. It is best to use the newest version. You can download the J2SE 5.0 from

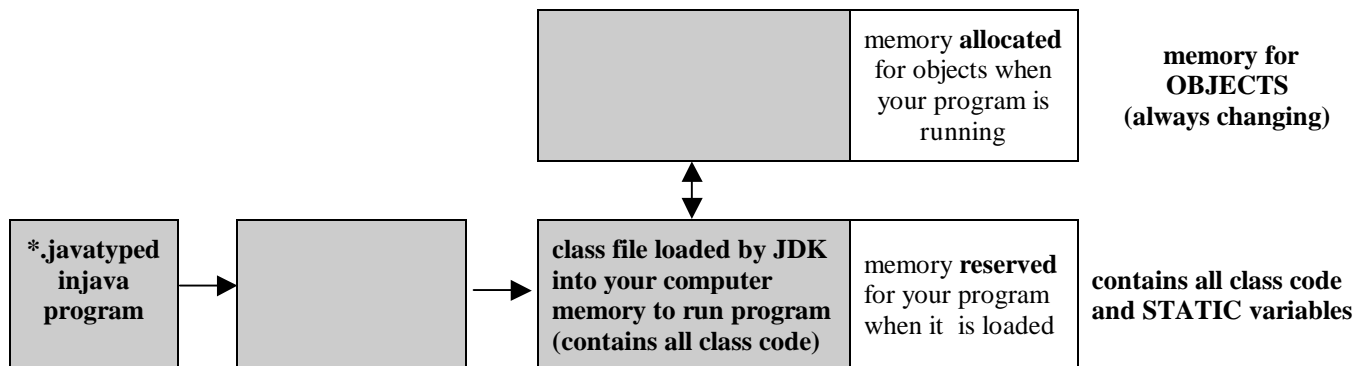
<http://java.sun.com/j2se/1.5.0/download.jsp>

You have many choices here just select JDK 5.0 Update 6. To install down load the exe file then run. Java will be installed and automatically set up for you.

We will write, compile and run our first java program soon. But before we do this you will need the following background information.

## Compile time and run time

**Compile time** is the time when the compiler is compiling your program into an execution file. During compile time the compiler is checking your program for errors. If you have errors in your program you have to fix them and then re-compile. During compile time the compiler will convert your Java programming statements into execution code and **reserve memory** locations for your data variables and compiled code. When the compiler reserves memory locations for your data variables and compiled code and places them inside the class execution file. When your program runs, the class execution file is loaded into your computer running the JDK. The JDK gives you some memory to run your program. The size of the memory depends on the size of your class execution file. We call this memory **program** or **class memory** and is said to be **reserved** memory. It is at a fixed size and never changes. Some methods are static meaning they can be used directly from the class execution code. When your program is running it may need additional memory for objects called **object memory**. The additional memory comes from your computer memory in run time and is **allocated** memory. Objects are created in computer memory during run time from the class definitions defined in your program. What is the difference between compile time and run time ? What is the difference between a class and an object ?



Why do we need additional memory when a program is running ? A good example is a telephone book program. When someone writes the program they do not know how many telephones entries the book is going to have. For every person needed in the telephone book there must be enough memory for the persons name, address and telephone number. When the program first runs it will ask the person how many entries the telephone book will have. As soon as the person enters the number the program must go to the computer memory and get enough memory for 100 entries. Alternatively you could just add additional memory when you add entries to your telephone book. You allocate memory in run time when you do not know how much memory you need. You reserve memory in compile time when you know how much memory you need. If you knew how many people you have in your telephone book then you could of reserve memory in compile time. You will soon learn how to reserve memory in compile time and allocate memory in run time.

## JAVA PROGRAM COMPONENTS

We will now discuss in depth all the components of a Java program. In Java you will see curly brackets like these { } all the time. They are used to group statements together in a method or methods in a class. Classes are made up of methods, methods are made up of statements, and statements are made up of variables and operators. Operators perform operations on variables like adding or equating.

## Comments

All programs need comments to explain to the user how the program works, what the variables represent and to define the purpose of classes and methods etc. You may place comments anywhere in your Java program. The compiler ignores comments. They are just there for the people who read your Java program. A comment starts with a `/*` and ends with a `*/`.

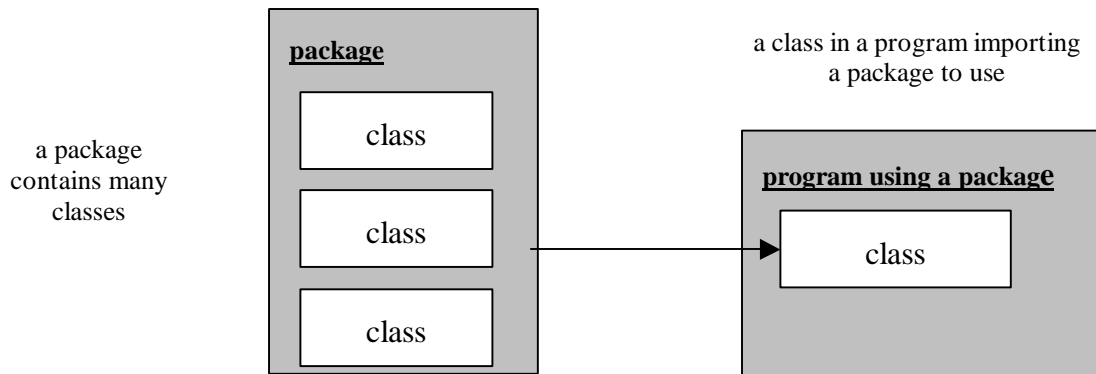
```
/* My first Java coment */
```

A comment may also start with a `"/"` . The end of the comment is assumed to be the end of the line.

```
// Lesson 1 Program 1
```

## Packages

Packages are optional and used mostly in large programs involving many classes. Packages allow you to group a collection of classes together that have a common purpose. A Package includes many classes. A class may import classes from a package.



## Importing Java pre-built classes

**Import statements** are used to include Java classes contained in a package. Each package has a class that performs Input/Output, math functions, language implementation etc. Each class has many methods so that you can easily get your Java program up and running quickly. The purpose of pre-defined classes is to relieve the programmer of all burdens. System source code files include input/output classes that allows you to get data from a keyboard or file system, send information to the computer screen, store data on an output file, make a connection to the internet etc. All of the Java system classes you **use** are imported from packages. You will get used to all the packages in Java because you are always forced to use them. The **compiler** needs to know what packages you want to use when it **compiles** your program. You use the import statement to use packages.

```
import package_name.*;
```

```
import java.io.*;
```

When you import a package you get to use all the classes in that package. In our example we have imported the **java.io** package and all of its classes. This package contains all classes needed for I/O. I/O means all the input and output streams classes needed to read data from a keyboard, send information to a file etc. We will study all the Java I/O streams in this course. The \* means to import all classes in the **java.io** package. If you just want one class in this package then you can specify it by name.

```
import package_name.class_name;
```

```
import java.io.BufferedReader;
```

Most people don't know all the names they want in a package so they use \* to get all classes belonging to that package. Java automatically imports the **java.lang.\*** package for you. This package contains all the basic Java classes to compile and run your programs like the System and String classes. Don't forget the semicolon at the end of the package name in the import statement.

### writing your own package

You can also write your own Java packages, just to store your very own classes. To make your classes belong to a package you just put the name of the package at the beginning of your Java source code file. All classes in this file will now belong to your package.

```
package package_name;
```

```
package student;
```

You must be careful in using packages of the same name. Some Java development tools will overwrite classes having the same name with the same named package without warning. You will lose all your original code. Always make sure all your package names are unique.

### importing your own package

You can also import your own packages. To use your own package you must also use the import statement.

```
import user_package_name.*;
```

```
import student.*;
```

The "\*" means use all classes in your package. If you just want to use a specific class of a package then you just specify which class you want:

```
import package_name . class_name;
```

```
import student.Calculate;
```

## CLASSES

A **class** defines all the **variables** and **methods** that your **object** will need. A class is defined by a **level of visibility**, the **class** keyword, the **class name** and the required **variables** and **methods**. The open curly bracket **{** means to **begin** something the closed curl bracket **}** means to **end** something.

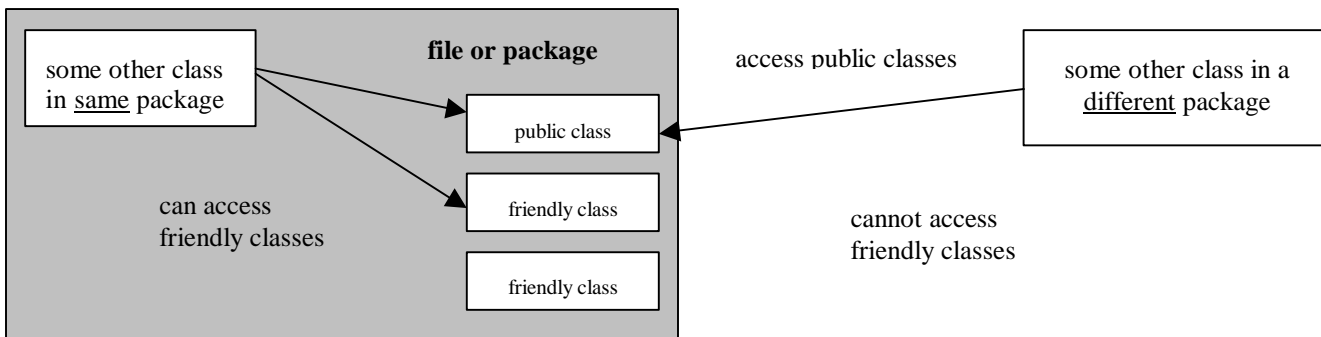
<pre style="margin: 0;">visibility class class_name {   variables   methods }</pre>	<pre style="margin: 0;">// example class public class L1p1 {   public static void main ( Strings[ ] args )   {     System.out.println("hello");   } }</pre>	<div style="border: 1px solid black; padding: 5px; margin-bottom: 5px;">class</div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 5px; background-color: #fff;">variables</div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 5px; background-color: #fff;">method</div> <div style="border: 1px solid black; padding: 5px; margin-bottom: 5px; background-color: #fff;">method</div> <div style="border: 1px solid black; padding: 5px; background-color: #fff;">main method</div>
---	---	---

### class visibility

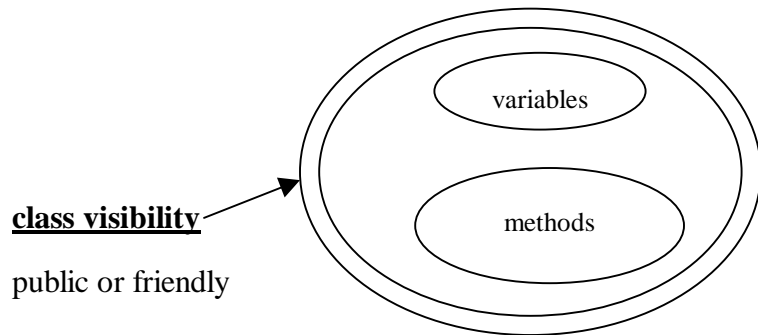
The **level of visibility** states who can use this class. Visibility for classes can be **public** or **friendly**. If the visibility is not specified the default visibility is assumed to be **friendly**. The visibility of our example class is public. Visibility is like a protection mechanism where the programmer can choose who can access the class.

class visibility	description
<b>public</b>	means all other classes can access this class
<b>friendly (default)</b>	means only classes in the package or file can access this class (classes defined in the same file are considered to be in the same package)

There is no friendly keyword. Friendly visibility is the default visibility when you do not specify a visibility.



Visibility is like a protection mechanism allowing only certain objects to enter



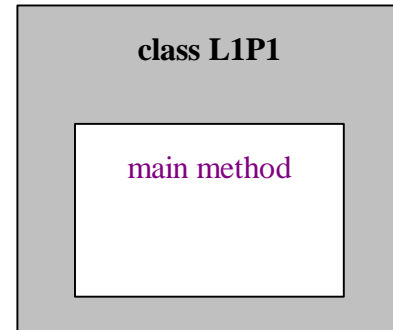
### class name, variables and methods

The **class** keyword indicates the following block of code is to be a class definition. All classes need a name so that the programmer and compiler can identify it. The name of our example class is L1P1. The variables and methods belonging to a class are enclosed by curly brackets { }. The variables in a class are shared between all methods in the class. This is why classes were invented so that we could have a collection of methods all sharing the same variables. This is a highly desirable thing to do, it allows your programs to be highly organized and stops data corruption. Data corruption is avoided because we now know which methods access the data. All classes start with a CAPITAL letter where methods start with a lower case letter. You need to define a class before you can make an object of it. A class definition is like the recipe to bake a cake. You need a recipe before you can bake a cake. When a cake is baked it is ready to be eaten. Obviously you cannot eat a recipe! When your program is running **objects** are constructed in the computer memory from your class definitions defined in your program. A class contains **variables** and **methods**. Just like a cake needs **ingredients** and the needed **instructions** to bake the cake. Variables are used to store and retrieve data. The variables in a class are shared between all methods in the class and retain their value. This is the main reason we have classes is that we want a bunch of methods to share the same variables! A class also defines all the methods needed by the object to access the data. Methods contain **variables** and **programming statements**. Variables in a method are used for temporary storage where the programming statements are the instructions telling the computer what to do. The variables in a methods do not retain their values and are said to be temporary. Only when a methods is used will the variables contain a value. After a method is finished running the variables values disappear. We will study methods in future lessons. In this lesson we will concentrate on variables.

### LESSON 1 PROGRAM 1

A simple Java program only needs a **class** and a **main method**. The main method is the first method to be executed when the Java program first runs. Each class can have its own main method. All class names in Java start with a CAPITAL letter, where method names start with a lower case letter. The following Java program has a class called L1p1 and is **public** meaning everybody can use it. Our example class just has a main method and prints out the message "Hello" to the screen.

```
// example class
public class L1p1
{
    // main method
    public static void main ( String[ ] args )
    {
        System.out.println("hello");
    }
}
```



**Program output:**

hello

The L1p1 class is known as the "running class" because it is the class we call to run the program. It makes sense that the running class is public. Why ? Our main method just puts the word "hello" on the screen using the following programming statement:

```
System.out.println("hello"); // print "hello" on computer screen
```

The System class is defined in **java.lang** and is automatically imported for you by the Java compiler. We will discuss the System class in more detail shortly.

**LESSON 1 EXERCISE 1**

Type the above program into your Java compiler and run it. You can obtain the JDK from Sun Microsystems at [www.java.sun.com](http://www.java.sun.com). For those people using the JDK, type the program into an editor, the file will be called L1p1.java. From the command line of your computer type:

```
javac L1p1.java
```

```
java L1p1
```

Your program should now run. The command line is case sensitive. The file name must be identical to the class name. Change the above program so that it print's out "Hello" and your name on the computer screen. Call your class L1ex1 and your Java file L1ex1.java. You will now need the following commands to compile and run your program.

```
javac L1ex1.java
```

```
java L1ex1
```

## Compiling and running classes defined in packages

The next challenge is to compile and run java programs defined in packages. This always seems a difficult thing to do. Just follow the following steps and then it should work.

**Step 1:** Type in the following program in a file called **L1p2.java**, but now define in a package called student.

```
package student;

// example class using packages
public class L1p2
{
    // main method
    public static void main ( String[ ] args )
    {
        System.out.println("hello");
    }
}
```

**step 2:** put the file L1p2.java in a subdirectory called student

**step 3:** compile the program in the parent directory

```
javac student\L1p2.java
```

**step 4:** run the file specifying the class path. The class path tells the Java loader where the student package is.

```
java -classpath . student/L1p2
```

Notice the '.' (dot) after -classpath this means use current directory. Notice the / after student this means the L1p2 class file is located in the sub directory student. Notice the slash leans towards the right, this is a Unix format for specifying subdirectories and files. You must use this format.

## importing your own package

We want to run a class defined in a another package. This is another difficult thing to do. We write a class called L1p3 in another file called L1p3.java. In this file we the **import** student package and call the main method from the L1p2 class.

**Step 1:** Type in the following program in a file called L1p3.java that imports the student package

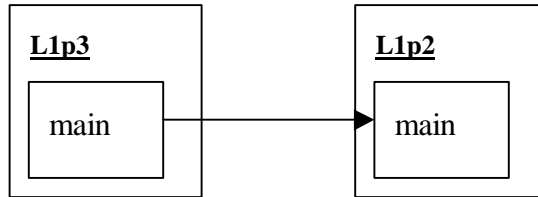
```
import student.*

// example class
public class L1p3

{
    // main method
    public static void main ( String[ ] args )

        {
            L1p2.main(args);
        }

}
```



**step 2:** compile the class in the parent directory but specify where the student package is

```
javac -classpath \student; L1p3.java
```

Since the student package is in the student subdirectory the compiler can find it so you can omit the -classpath directive

```
javac L1p3.java
```

**step 3:** run the file specifying the specifying the class path. The class path tells the Java loader where the student package is.

```
java -classpath \student; L1p3
```

Since the student package is in the student subdirectory the class loader can find it so can omit the -classpath directive

```
java L1p3
```

The L1p3 class main method calls the main method from the L1p2 class.

## MAIN METHOD

When you run your program, the **main** method is the first method to be called from the running class. Every class you want to run must have a main method.

```
public class L1p1
{
// main method
public static void main ( String[ ] args )

        {
System.out.println("hello");
}
}
```

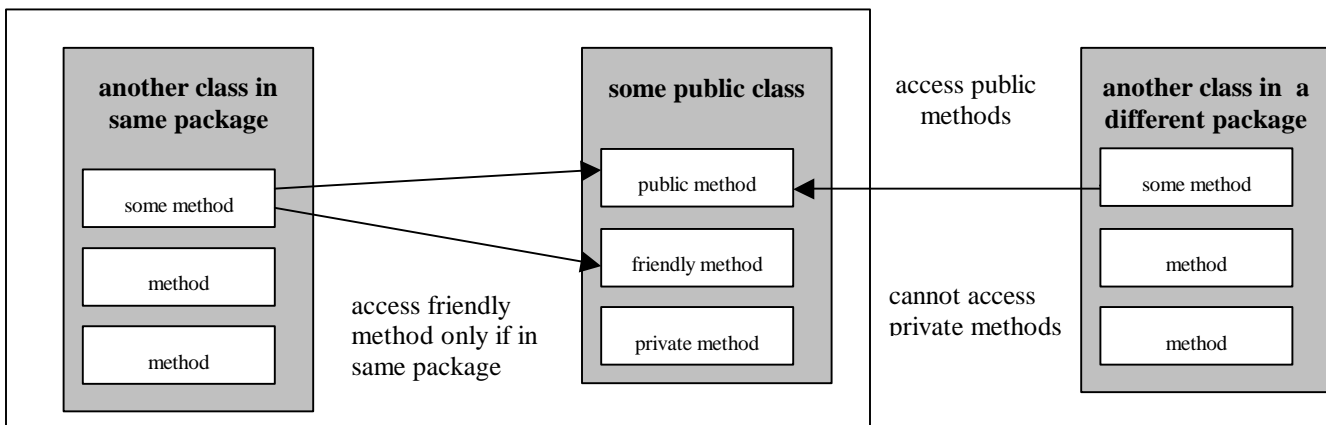
We will now discuss and analyze the main method in detail . Every method must state who can access it, what data type value it returns, its name what data type values it receives.

<i>method visibility</i>	<i>modifier</i>	<i>return data _type</i>	<i>method name</i>	<i>parameter(s)</i>	
				<i>parameter_type</i>	<i>parameter_name</i>
public	static	void	main	( String[ ]	args )

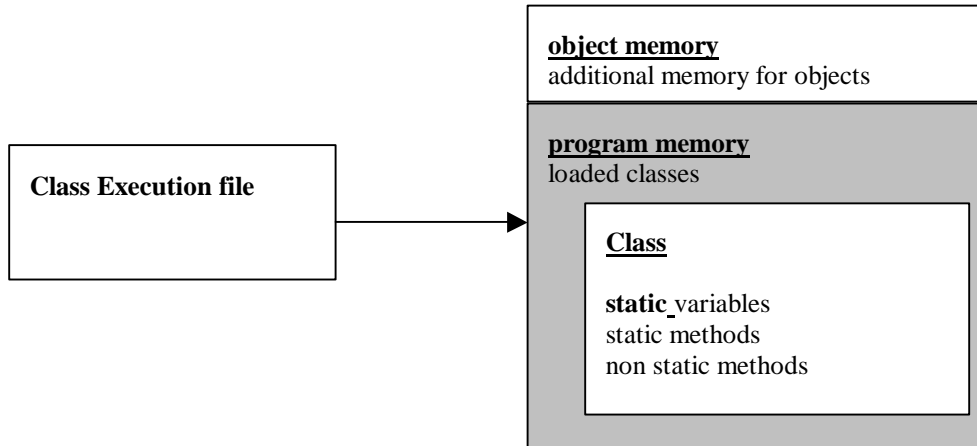
## method visibility

Methods have visibility **public**, **friendly** and **private**.

method visibility	description
<b>public</b>	means all other classes can access this method
<b>friendly (default)</b>	can be accessed by classes in the same package or file(classes defined in the same file are considered to be in the same package) (There is no keyword called friendly)
<b>private</b>	visibility means the method can only be accessed by the class it was defined in.



All classes are placed in program memory when a Java program is loaded. **Static** means the code of the main method is directly available to be used. A static method is not associated with an object and can be used standalone. A static variable is also not associated with any object and can also be used standalone. For non-static variables and methods, you need an object before you can use them. Why? Because the methods are accessing variables of an object. If you do not have an object then the methods cannot access the variables of the object. Right?

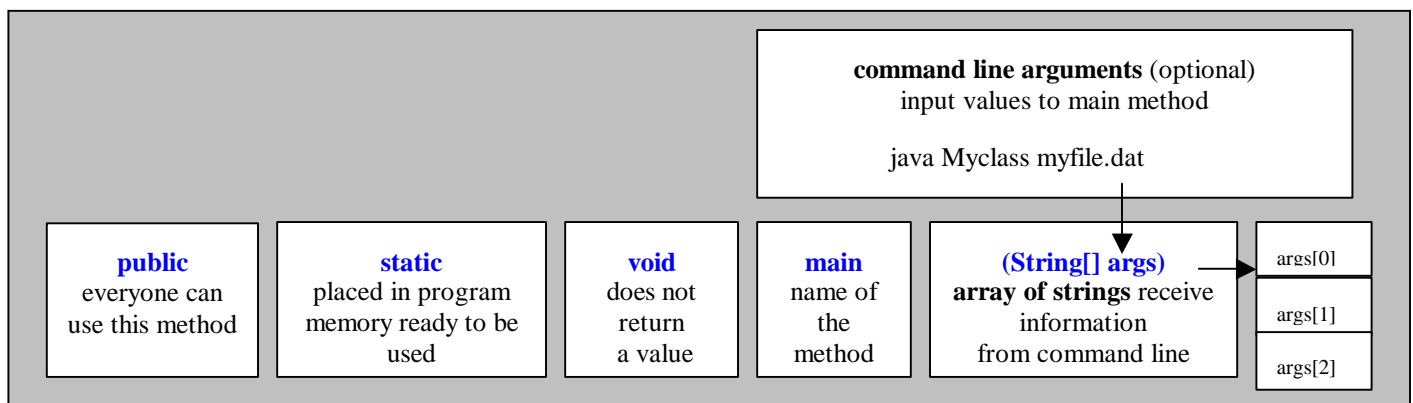


The purpose of a method is to receive a value, do a calculation and return a result. A method may or may not return a value. The **return data type** is the type of data the method will return. The return data type of **void** is used to indicate a method does not return a value. Notice the return data type of the main method is **void**. We will study data types very shortly. All method names end with round brackets () to distinguish the method name from an variable name. Values are sent to a method through a **parameters**. Parameters are declared inside the round brackets of the method name declaration to pass values to the method. The values sent to the main method are known as **arguments**.

**Arg** is an array of String objects that will let you send data values to the main method. You send the data values to the main method through the parameter list by including extra information on the command line when you run your Java program. An example is you may want to specify an additional data file name when you want to run your Java program on the computer.

**java myprogram myfile.dat**

The argument "myfile.dat" will be placed into an array of Strings referred to by the parameter args.



Do not be too worried about arguments and parameters now, you just need to know that you can send information to the main method from the command line when you run your Java program. The String object type will be explained in this lesson. Arrays and methods will be explained in detail in future lessons. Without the main method the program can not run. Review questions: Why must the main method be made public ? Why is the main method static ?

## LESSON 1 EXERCISE 2

Write a program that gets a name from the keyboard and greets the user of your program with the name they entered. Call your java program and class L1Ex2. The following is an example compiling and running the program.

```
javac L1Ex2
```

```
java L1Ex2 tom
```

```

Hello Tom

```

## Constants

Numbers like (0-9) and letters like ('a'- 'Z') are known as constants. Constants are **hard coded** values assigned to variables. There are many different types of constants.

**character:** 'A'    **numeric:** 9    **decimal:** 10.5    **character string:** "hello"

Numbers represent numeric value whereas letters are assigned numeric values for identification from a chart known as the ASCII table. The letter 'A' is assigned the numeric value 65. Letters constants have **single quotes** 'a' around them and are known as characters. What is the difference between '9' and 9 ? Numbers may be grouped together to make larger numbers like 1234. Characters maybe grouped together to make messages and enclosed by **double quotes**: "hello". Groups of letters are known as character strings. What is the difference between "1234" and 1234 ? What is the difference between a character and a character string ?

## backslash characters

There are special character that are called back slash codes. The backslash codes allow you to specify new line or tab in your character strings like: "hello\n"

code	description	code	description
\b	backspace	\0	end of string terminator
\n	new line	\\	backslash
\r	carriage return	\v	vertical tab
\t	horizontal tab	\a	bell
\"	double quotation mark	\o	octal constant
\'	single quotation mark	\x	hexadecimal constant

**range of a number**

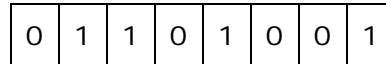
All memory is made up of **bits** having values 0 and 1. Eight bits are make up a byte.

1 byte = 8 bits



Data types use 1 to many bytes. Data types representing small numbers use few bytes of memory where data types representing large numbers use many bytes of memory. All numbers in Java are **signed** meaning they can represent negative and positive numbers. The first bit Most Significant Bit (MSB) in a **signed** number is called the sign bit. It indicates a signed number if it is positive or negative.

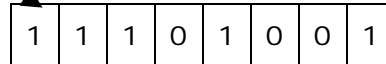
Signed **Positive** numbers start with a 0.



sign bit



Signed **Negative** numbers start with a 1



**range of a signed number**

The smallest negative value to the largest positive value a number can represent is known as the **range** of a number. We need to know the range of a number before we can use it or we will get inaccurate results. We use the following formula to calculate the range of the number from the number of bits N the number has.

$$- 2^{(N-1)} \quad \text{to} \quad (2^{(N-1)} - 1)$$

N is the number of bits in the number.

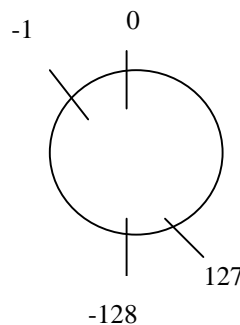
N=8

The range of a number is calculated for an 8 bit signed number where:

$$- 2^{(8-1)} \quad \text{to} \quad 2^{(8-1)} - 1 = - 2^7 \quad \text{to} \quad 2^7 - 1 = -128 \quad \text{to} \quad 127$$

This means a data type that has 8 bits can represent values between -128 and 127. Why is the negative number magnitude greater than the positive number magnitude ?

**The positive numbers start at 0 and end at 127**  
**The negative numbers start at -1 and end at -128**  
**Each has 128 numbers**



## VARIABLES AND DATA TYPES

**Variables** let you store and retrieve data in a computer memory represented by a variable name like **x**. The **variable name** lets you identify a particular memory location. The compiler reserves the memory address automatically for you when it compiles your program. A variable is like a bank account. You can put money in and take money out. The particular place (location) where your money is located is identified by the bank account number. Variables have different **data types**, just as a bank account would represent many different **currencies**. Each variable must be declared with a **data type**. The data type indicates what kind of data the variable is to represent. The following chart lists the Java data types and an example of the data it is to represent.

Java data types				signed range
type	bytes	N bits	example	$- 2^{\text{exp}(\text{N}-1)}$ to $2^{\text{exp}(\text{N}-1)}-1$
byte	1	8	45	-128 to 127
short	2	16	1234	-32,768 to 32,767
int	4	32	123467	-2147483648 to 2147483647
long	8	64	123456788	$-2^{63}$ to $2^{63}-1$
float	4	32	34.56f	$\pm 3.40282347 \times 10^{38}$
double	8	64	23.23333333	$\pm 1.79769313486231570 \times 10^{308}$
char	2	16	'A' (unicode)	16 bit character set
boolean	1	8	true or false	false true

These are known as **primitive data types** and are not considered objects. Data types are not objects because they just store data, and cannot do any operations on the stored data. Where objects store data and can do operations on the stored data, using its methods. We need different data types because we want to **minimize** the amount of data memory space we use. The data type also indicates the type of data the memory allocated for the variable is to represent

### float and double number representation

Float and double data types are used to represent large numbers known as floating point numbers stored in a small memory space. Floating point numbers have a **fraction (mantissa)** part and an **exponent part** and represented in decimal point notation 24.56f or exponent notation 3.456E04f. You will notice that float constants have a 'f' at the end of them. In Java you need to distinguish a float constant from a double constant. It is the exponent that allows the floating point number to represent a large value in a small memory space. The following is a 32 bit floating point format known as a **float**, where the sign, exponent and fractional parts are stored separately.

1	8	23	32 bit floating point number ( <b>float</b> )
sign bit	exponent	fraction (mantissa)	$(-1)^{\text{sign}} * 2^{\text{exp} - 127} * (\text{mantissa})$
-	12	.12365	$1.2365 * 10^{12}$

The following is a 64 bit floating point format known as a **double**, where the sign, exponent and fractional parts are stored separately. Double has greater precision over float because it stores more bits in their exponent and fraction.

1	16	47	64 bit floating point number ( <b>double</b> )
sign bit	exponent	fraction (mantissa)	$(-1)^{\text{sign}} * 2^{\text{exp} - 127} * (\text{mantissa})$
-	12	.12365	$1.2365 * 10^{12}$

Variables store data

## Declaring variables

When you declare a variable you are telling the compiler to **reserve** a memory space location in the computer memory for your data when your program runs. The variable name is used to represent the location in the computer memory where the data value is stored. Declaring a variable is like opening a bank account. The name of the variable is like the bank account number. The bank account number tells you where your money is in the bank. The bank account number identifies the location where your money is in the bank. It is just like the variable in your program identifying the locations in the computer memory where your data value is stored. To declare a variable you specify the data or object type and the variable name. A data type is like the currency of the money stored in the bank account. When you declare a variable in a method the data value is **uninitialized** U meaning a known value has not been placed at that memory location yet. When you declare a variable in a class definition it gets a default value of 0. You declare a variable by specifying the data type and an identification name. Variable names should be short and meaningful. Every variable must be declared with a data type and a name. The data type tells your program what kind of data the computer memory is suppose to represent. The variable name tells your program the location in the computer memory where your data value is stored. Data type indicated what kind of data the variable represents.

*data\_type*      *variable\_name;*

**int** y;      // declare variable y of data type int

**char** ch;      // declare variable c of data type char.

**double** d;      // declare variable d of data type double

**uninitialized variables**

<b>y</b>	<b>U</b>	<b>0</b>
<b>ch</b>	<b>U</b>	<b>'\0'</b>
<b>d</b>	<b>U</b>	<b>0.0</b>

You may declare more than one variable at a time if it is the **same data type**, by separating the variable names by commas.

*data\_type*      *variable\_name\_list*;

**int i, j, k;**      // declare a variable's i, j and k having data type integer

uninitialized variables

<b>i</b>	<b>U</b>	<b>0</b>
<b>j</b>	<b>U</b>	<b>0</b>
<b>k</b>	<b>U</b>	<b>0</b>

## declaring and initializing variables

You may assign values to variables when they are declared. This is called **initializing**. It is like opening a bank account with an initial deposit.

*data\_type* *variable\_name* = *value*;

**int x = 5;**      // declare x and initialize to the value 5

**double d = 10.5;**      // declare d and initialize to the value 10.5

<b>x</b>	<b>5</b>
<b>d</b>	<b>10.5</b>

A string variable s1 refers to a String object rather than contains the character string itself. The string object contains the character string "hello". Refers means s1 is not the String object itself but contains the **location** where the String object is located in the computer memory.

You can declare many variables and initialize them all at once if they are of the **same data type**, by separating the variable names and initializations by commas.

*data\_type* *variable\_name\_and\_initialization\_list*;

**int m=1, n=2, p=3;**

<b>m</b>	<b>1</b>
<b>n</b>	<b>2</b>
<b>p</b>	<b>3</b>

## String Objects

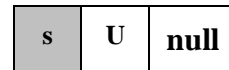
Character strings are groups of letters enclosed by double quotes:

"message"

It is easy to use character strings in Java because Java has the built in String class located in **java.lang**. Character strings in Java are represented by the String class. A String is considered an **object** in Java and is not a primitive data type. String is known as a **object type** rather than a **data type**. The String class lets you do operations on strings. These operations may be assigning character strings or joining two character strings together. A character string is made up of characters. A character is a single letter in the string. A character is denoted by single quotes, like the letter 'a'. There is a big difference between 'a' and "a". 'a' is a character where as "a" is a character string with only 1 character.

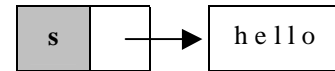
You declare a String variable as follows:

```
String s; // declare variable s having a object type of String
```

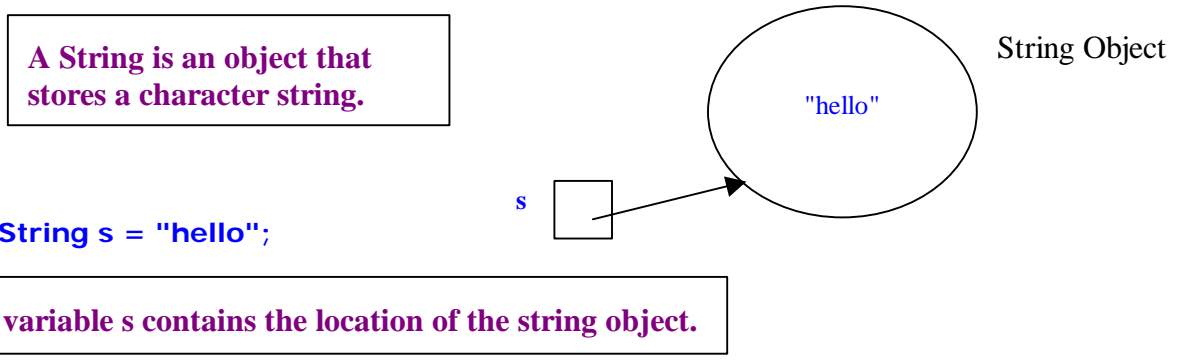


The String variable `s` will store the location of a String object. `s` is known as a reference to a string variable. Right now variable `s` does not refer to any String object and is **undefined** or **null**. A string variable in a method is undefined a string variable in a class definition gets the default value **null**. `null` means no known object location. The best thing to do is to declare a String variable and assign a String object to it. The String object will store the character string.

```
String s = "hello"; // declare and initialize s to "hello"
```



The String variable `s` does not contain the character string "hello" but contains the **location** where the string object is.



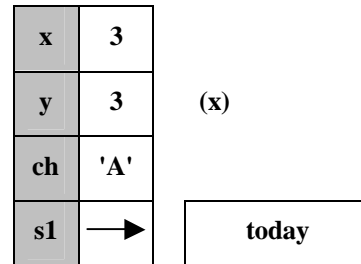
**The variable `s` contains the location of the string object.**

### Assigning values to variables

Assignment statements using the "=" assignment operator allow you to assign values to existing declared variables. The expression on the right hand side is assigned to the variable on the left hand side of the assignment "=" operator. The variable gets the new value and the old value is over written. Values are expressions. An expression may be a constant, numeric, String data values or other variable with an assigned value. Once a variable is declared it does not need to be re-declared to assign a new value to it. Its like you do not need to re-open a bank account when you want to make deposits and withdraws. Before you can assign a value to a variable, the variable must be previously declared. When you assign a variable to another variable the value of the variable (not the variable name) on the right hand side is assigned to the variable on the left hand side. Assignment statements have the form:

```
variable_name = expression;
```

```
x = 3; // variable x gets the value of 3
y = x; // variable y gets the value of x which happens to be 3
ch = 'A'; // variable ch gets the character constant 'A'
s1 = "today"; // the String object s gets the character string "hello"
```



## Printing out variable values

The `System.out.println()` statement is used to print values and messages to the computer screen. The `System` class is defined in **java.lang** and is automatically imported for you when you compile your program. The `System` class has a variable called **out** that is a reference to a **PrintStream** class object defined in `java.io`. (we will study reference's to objects shortly) **println()** is a method of the `PrintStream` class. The `println()` method causes your printed message to end with a new line **after** the message is printed not before. If you do not want a new line after your message use the `print()` method instead.

```
System.out.print("hello"); // print a message to the screen, with out a new line
System.out.println("hello"); // print a message to the screen
System.out.println("hello"); // print a message to the screen
```

```
hellohello
hello
```

To print out a variable value you just include the variable name.

```
System.out.println(x); // print a variable value to the screen
```

```
3
```

To print a message and a variable value to the screen you list the message in double quotes use the "+" **catenation** operator and list the variable name. Messages are surrounded with double quotes and variables are not. The "+" catenation operator is used to join messages and variables together.

```
System.out.println("the value of x is: " + x); // print a message and value to screen
```

```
join messages together
```

```
the value of x is: 3
```

### LESSON 1 EXERCISE 3

Write a simple Java program that has only one class and a main method. In the main method declare **five** variables of different data types. and initialize these variables with values. Use the `System.out.println()` statement to print the values of the variables on the screen. Next assign values to your variables and print out the new values. Call your class `L1ex2` and your Java file `L1ex3.java`.

**IMPORTANT**

You should use all the material in all the lessons to do the questions and exercises. If you do not know how to do something or have to use additional books or references to do the questions or exercises, please let us know immediately. We want to have all the required information in our lessons. By letting us know we can add the required information to the lessons. The lessons are updated on a daily bases. We call our lessons the "living lessons". Please help us keep our lessons alive.

**E-Mail all typos, unclear test, and additional information required to:**

**[courses@cstutoring.com](mailto:courses@cstutoring.com)**

**E-Mail all attached files of your completed exercises to:**

**[students@cstutoring.com](mailto:students@cstutoring.com)**

**Order your next lesson from**

**[www.cscourses.com/java.htm](http://www.cscourses.com/java.htm)**

This lesson is copyright (C) 1998-2006 by The Computer Science Tutoring Center "cstutoring"  
This document is not to be copied or reproduced in any form. For use of student only