Java

Lecture 1

The Java Development Kit – JDK

In order to get started in Java programming, one needs to get a recent copy of the Java JDK. This can be obtained for free by downloading it from the Sun Microsystems website, http://java.sun.com/

Once you download and install this JDK you are ready to get started. You need a text editor as well and Microsoft's Notepad (standard with all Windows versions) suits fine.

My first Java program

Open your text editor and type the following lines of code:

```
/*
My first program
Version 1
*/
public class Example1 {
public static void main (String args []) {
System.out.println ("My first Java program");
}

}
```

Variables and Data Types

Variables

A variable is a place where the program stores data temporarily. As the name implies the value stored in such a location can be changed while a program is executing (compare with constant).

```
class Example2 {
public static void main(String args[]) {
int var1; // this declares a variable
int var2; // this declares another variable
var1 = 1024; // this assigns 1024 to var1
System.out.println("var1 contains " + var1);
var2 = var1 / 2;
System.out.print("var2 contains var1 / 2: ");
System.out.println(var2);
}
}
```

Mathematical Operators

As we saw in the preceding example there are particular symbols used to represent operators when performing calculations:

Operator	Description	Example – given a is 15 and b is 6
+	Addition	a + b, would return 21
-	Subtraction	a - b, would return 9
*	Multiplication	a * b, would return 90
/	Division	a / b, would return 2
%	Modulus	a % b, would return 3 (the remainder)

```
class Example4 {
public static void main(String args[]) {
int iresult, irem;
double dresult, drem;
iresult = 10 / 3;
irem = 10 % 3;
dresult = 10.0 / 3.0;
drem = 10.0 % 3.0;
System.out.println("Result and remainder of 10 / 3: " +
iresult + " " + irem);
System.out.println("Result and remainder of 10.0 / 3.0: "
+ dresult + " " + drem);
}
}
```

```
Predicted Output:
Result and Remainder of 10/3: 3 1
Result and Remainder of 10.0/3.0: 3.3333333333333335 1
```

Logical Operators

These operators are used to evaluate an expression and depending on the operator used, a particular output is obtained. In this case the operands must be Boolean data types and the result is also Boolean. The following table shows the available logical operators:

Operator	Description
&	AND gate behaviour (0,0,0,1)
|	OR gate behaviour (0,1,1,1)
^	XOR – exclusive OR (0,1,1,0)
&&	Short-circuit AND
||	Short-circuit OR
!	Not

```
class Example5 {
public static void main(String args[]) {
int n, d;
n = 10;
d = 2;
if(d != 0 && (n % d) == 0)
System.out.println(d + " is a factor of " + n);
d = 0; // now, set d to zero
// Since d is zero, the second operand is not evaluated.
if(d != 0 && (n % d) == 0)
System.out.println(d + " is a factor of " + n);
/* Now, try same thing without short-circuit operator.
This will cause a divide-by-zero error.
*/

if(d != 0  ||  (n % d) == 0)
System.out.println(d + " is a factor of " + n);
}
}
```

Predicted Output:

```
2 is a factor of 10
```

Character Escape Codes

The following codes are used to represents codes or characters which cannot be directly accessible through a keyboard:

Code	Description
\n	New Line
\t	Tab
\b	Backspace
\r	Carriage Return
\\	Backslash
\'	Single Quotation Mark
\"	Double Quotation Mark
*	Octal - * represents a number or Hex digit
\x*	Hex
\u*	Unicode, e.g. \u2122 = ™ (trademark symbol)

```
class Example6 {
public static void main(String args[]) {
System.out.println("First line\nSecond line");
System.out.println("A \t B \t C");
System.out.println("D \t E\ t F") ;
}

}
```

Predicted Output:

```
First Line
Second Line
A B C
D E F
```

Data Types

The following is a list of Java's primitive data types:

Data Type	Description
int	Integer – 32bit ranging from -2,147,483,648 to 2,147,483,648
byte	8-bit integer ranging from -128 to 127
short	16-bit integer ranging from -32,768 to 32,768
long	64-bit integer from -9,223,372,036,854,775,808 to -9,223,372,036,854,775,808
float	Single-precision floating point, 32-bit
double	Double-precision floating point, 64-bit
char	Character , 16-bit unsigned ranging from 0 to 65,536 (Unicode)
boolean	Can be true or false only

The 'String' type has not been left out by mistake. It is not a primitive data type, but strings (a sequence of characters) in Java are treated as Objects.

```
class Example8 {
public static void main(String args[]) {
int var; // this declares an int variable
double x; // this declares a floating-point variable
var = 10; // assign var the value 10
x = 10.0; // assign x the value 10.0
System.out.println("Original value of var: " + var);
System.out.println("Original value of x: " + x);
System.out.println(); // print a blank line

// now, divide both by 4
var = var / 4;
x = x / 4;
System.out.println("var after division: " + var);
System.out.println("x after division: " + x);
}
}
```

Predicted output:

```
Original value of var: 10
Original value of x: 10.0
var after division: 2
x after division: 2.5
```

```
class Example9 {
public static void main(String args[]) {
char ch;
ch = 'X';
System.out.println("ch contains " + ch);
ch++; // increment ch
System.out.println("ch is now " + ch);
ch = 90; // give ch the value Z
System.out.println("ch is now " + ch);
}
}
```

Predicted Output:

```
ch is now X
ch is now Y
ch is now Z
```

ASCII printable characters

32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q
50	2	82	R	114	r
51	3	83	S	115	s
52	4	84	T	116	t
53	5	85	U	117	u
54	6	86	V	118	v
55	7	87	W	119	w
56	8	88	X	120	x
57	9	89	Y	121	y
58	:	90	Z	122	z
59	;	91	[	123	{
60	<	92	\	124	|
61	=	93	]	125	}
62	>	94	^	126	~
63	?	95	_		

```
class Example10 {
public static void main(String args[]) {
boolean b;
b = false;
System.out.println("b is " + b);
b = true;
System.out.println("b is " + b);
// a boolean value can control the if statement
if(b) System.out.println("This is executed.");
b = false;
if(b) System.out.println("This is not executed.");
// outcome of a relational operator is a boolean value
System.out.println("10 > 9 is " + (10 > 9));
}
}
```

Predicted output:

```
b is false
b is true
This is executed
10 > 9 is true
```

Introducing Control Statements

These statements will be dealt with in more detail further on in this booklet. For now we will learn about the ***if*** and the ***for loop***.

Operator	Description
<	Smaller than
>	Greater than
<=	Smaller or equal to, (a<=3) : if a is 2 or 3, then result of comparison is TRUE
>=	Greater or equal to, (a>=3) : if a is 3 or 4, then result of comparison is TRUE
==	Equal to
!=	Not equal

```
class Example11 {
public static void main(String args[]) {
int a,b,c;
a = 2;
b = 3;
c = a - b;
if (c >= 0) System.out.println("c is a positive number");
if (c < 0) System.out.println("c is a negative number");
System.out.println();
c = b - a;
if (c >= 0) System.out.println("c is a positive number");
if (c < 0) System.out.println("c is a negative number");
}
}
```

Predicted output:

```
c is a negative number
c is a positive number
```

```
class Example12 {
public static void main(String args[]) {
int count;
for(count = 0; count < 5; count = count+1)
System.out.println("This is count: " + count);
System.out.println("Done!");
}
}
```

Predicted Output:

```
This is count: 0
This is count: 1
This is count: 2
This is count: 3
This is count: 4
Done!
```

The following table shows all the available shortcut operators:

Operator	Description	Example	Description
++	Increment	a++	a = a + 1 (adds one from a)
--	Decrement	a--	a = a – 1 (subtract one from a)
+=	Add and assign	a+=2	a = a + 2
-=	Subtract and assign	a-=2	a = a – 2
*=	Multiply and assign	a*=3	a = a * 3
/=	Divide and assign	a/=4	a = a / 4
%=	Modulus and assign	a%=5	a = a mod 5

Blocks of Code

Whenever we write an IF statement or a loop, if there is more than one statement of code which has to be executed, this has to be enclosed in braces, i.e. ‘, …. -’

```
class Example13 {
public static void main(String args[]) {
double i, j, d;
i = 5;
j = 10;
if(i != 0) {
System.out.println("i does not equal zero");
d = j / i;
System.out.print("j / i is " + d);
}
System.out.println();
}
}
```

Predicted Output:

```
i does not equal to zero
j
```

Array

Declaring Array Variables

To use an array in a program, you must declare a variable to reference the array, and you must specify the type of array the variable can reference. Here is the syntax for declaring an array variable In Java, here is how we can declare an array.

dataType arrayName;[]

•dataType - it can be primitive data types like int, char, double, byte,etc.

•arrayName - it is an identifier

For example,double data;[]

Another example:int intArray[]; //declaring array

intArray = new int[20]; // allocating memory to array

OR

int intArray[] = new int[20]; // combining both statements in one

type[] name = new type[length]; int[] numbers = new int[5];

you can declare arrays of other types:

```
byte[] anArrayOfBytes;
short[] anArrayOfShorts;
long[] anArrayOfLongs;
float[] anArrayOfFloats;
double[] anArrayOfDoubles;
boolean[] anArrayOfBooleans;
char[] anArrayOfChars;
String[] anArrayOfStrings;
```

You can also place the brackets after the array's name:

```
// this form is discouraged
float anArrayOfFloats[];
```

++++++++++++++++++++++++

```
class Main {

 public static void main(String[] args) {

  // create an array

  int[] age = {12, 4, 5, 2, 5};
```

```
  // access each array elements
  System.out.println("Accessing Elements of Array:");
  System.out.println("First Element: " + age[0]);
  System.out.println("Second Element: " + age[1]);
  System.out.println("Third Element: " + age[2]);
  System.out.println("Fourth Element: " + age[3]);
  System.out.println("Fifth Element: " + age[4]);
 }
}
```

```
Accessing Elements of Array:
First Element: 12
Second Element: 4
Third Element: 5
Fourth Element: 2
Fifth Element: 5
```

++

```
class Main {
 public static void main(String[] args) {
  // create an array
  int[] age = {12, 4, 5};
  // loop through the array
  // using for loop
  System.out.println("Using for Loop:");
  for(int i = 0; i < age.length; i++) {
    System.out.println(age[i]);
  }
 }
```

```
}  Using for Loop:
12
4
5
```

++

```
class Main {
 public static void main(String[] args) {
  // create an array
  int[] age = {12, 4, 5};
  // loop through the array
  // using for loop
  System.out.println("Using for-each Loop:");
  for(int a : age) {
   System.out.println(a);
  }
 }
```

```
}Using for-each Loop:
12
4
```

```
class Main {
 public static void main(String[] args) {
  int[] numbers = {2, -9, 0, 5, 12, -25, 22, 9, 8, 12};
  int sum = 0;
  Double average;
  // access all elements using for each loop
  // add each element in sum
```

```
  for (int number: numbers) {
    sum += number;
  }
  // get the total number of elements
  int arrayLength = numbers.length;
  // calculate the average
  // convert the average from int to double
  average =  ((double)sum / (double)arrayLength);
  System.out.println("Sum = " + sum);
  System.out.println("Average = " + average);
}
```

```
}Sum = 36
Average = 3.6
```

Inside the loop, we are calculating the sum of each element. Notice the line,

```
int arrayLength = number.length;
```

Here, we are using the length attribute of the array to calculate the size of the array. We then calculate the average using:

```
average = ((double)sum / (double)arrayLength);
```

Multidimensional Arrays

Arrays we have mentioned till now are called one-dimensional arrays. However, we can declare multidimensional arrays in Java.

A multidimensional array is an array of arrays. That is, each element of a multidimensional array is an array itself. For example,

```
double[][] matrix = {{1.2, 4.3, 4.0},
```

```
        {4.1, -1.1}
};
```

```
import java.util.Scanner;
public class ArrayInputExample1
{
public static void main(String[] args)
{
int n;
Scanner sc=new Scanner(System.in);
System.out.print("Enter the number of elements you want to store: ");
n=sc.nextInt(); //reading the number of elements from the that we want to ente
int array[] = new int[10]; //creates an array in the memory of length 10
System.out.println("Enter the elements of the array: ");
for(int i=0; i<n; i++)
{

array[i]=sc.nextInt(); //reading array elements from the user
}
System.out.println("Array elements are: ");
// accessing array elements using the for loop
for (int i=0; i<n; i++)
{
System.out.println(array[i]);
}
}
}
```

++

What are the contents of numbers after executing this code?

```
int[] numbers = new int[8];
numbers[1] = 3;
numbers[4] = 7;
numbers[6] = 5;
int x = numbers[1];
numbers[x] = 2;
numbers[numbers[4]] = 9;
// 0 1 2 3 4 5 6 7
```

A. {0, 3, 0, 2, 7, 0, 5, 9}
B. {0, 3, 0, 0, 7, 0, 5, 0}
C. {3, 3, 5, 2, 7, 4, 5, 0}
D. {0, 3, 0, 2, 7, 6, 4, 4}

```
int[] numbers = new int[8];
for (int i = 0; i < numbers.length; i++) {
```

```
numbers[i] = 2 * i;
}
```
index 0 1 2 3 4 5 6 7
value **0 2 4 6 8 10 12 14**

```
++++++++++++++++++++++
```

```
public void run() {
int[] numbers = new int[7];
fillArray(numbers);
println(Arrays.toString(numbers));
}
private void fillArray(int[] arr) {
for (int i = 0; i < arr.length; i++) {
arr[i] = 2 * i;
}
}
```

```
++++++++++++++++++++++++++++++++++++++++++++++++
```

```
public void run() {
int[] array = new int[5];
...
swapElements(array[0], array[1]);
...
}
private void swapElements(int x, int y) {
int temp = x;
x = y;
y = temp;
}
```

```
++++++++++++++++++++++++++++++++++++++++++++++
```

```
for( i=0; i < temperature.length; i++)
{
temperature[i] = 0;
}


for( i=0; i < temperature.length; i++)
{
temperature[i] = Math.random()*100;
}


for( i=0; i < temperature.length; i++)
System.out.println(temperature[i]);
double total = 0;
for( i=0; i < temperature.length; i++)
{
```

```
total += temperature[i];
}


double max = temperature[0];
double min = temperature[0];
for( i=0; i < temperature.length; i++)
{
if( temperature[i] > max) max = temperature[i];
if( temperature[i] < min) min = temperature[i];
}


double max = temperature[0];
int indexOfMax = 0;
for( i=0; i < temperature.length; i++)
{
if( temperature[i] > max)
{
max = temperature[i];
indexOfMax = i;
}
}


double temp = temperature[0];
for( i=0; i < temperature.length; i++)
{
temperature[i - 1] = temperature[i];
}
```

Lecture 3

Using the Scanner Class

In Java 5 a particular class was added, the Scanner class. This class allows users to create an instance of this class and use its methods to perform input. Let us look at the following example which performs the same operation as the one above (works out the average of three numbers):

```
import java.util.Scanner;
Refrence_Variable = new Scanner(System.in);
Scanner  read = new scanner(system.in);
```

```
import java.util.Scanner;
public class ScannerInput {
public static void main(String[] args) {
//... Initialize Scanner to read from console.
Scanner input = new Scanner(System.in);
System.out.print("Enter first number : ");
int a = input.nextInt();
System.out.print("Enter second number: ");
int b = input.nextInt();
System.out.print("Enter last number : ");
int c = input.nextInt();
System.out.println("Average is " + (a+b+c)/3);
}
}
```

```
import java.util.Scanner;

class Main {
  public static void main(String[] args) {
```

```
    // creates an object of Scanner
    Scanner input = new Scanner(System.in);

    System.out.print("Enter your name: ");

    // takes input from the keyboard
    String name = input.nextLine();

    // prints the name
    System.out.println("My name is " + name);

    // closes the scanner
    input.close();
  }
}
```

The following simple example utilizes the Scanner4 for input:

```
//import package containing scanner
import java.util.*;
//read an integer and return it to user
public class Scan {
public static void main (String args[]){
//creating instance
Scanner kb = new Scanner(System.in);
System.out.println("Enter a number: ");
//read integer
int x = kb.nextInt();
System.out.println("Number: " + x);
}
}
```

Here, we have created an object of `Scanner` named `input`.

The `System.in` parameter is used to take input from the standard input. It works just like taking inputs from the keyboard.

We have then used the `nextLine()` method of the `Scanner` class to read a line of text from the user.

Now that you have some idea about `Scanner`, let's explore more about it.

Java Scanner Methods to Take Input

The `Scanner` class provides various methods that allow us to read inputs of different types.

Method	Description
`nextInt()`	reads an `int` value from the user
`nextFloat()`	reads a `float` value form the user
`nextBoolean()`	reads a `boolean` value from the user
`nextLine()`	reads a line of text from the user
`next()`	reads a word from the user
`nextByte()`	reads a `byte` value from the user
`nextDouble()`	reads a `double` value from the user
`nextShort()`	reads a `short` value from the user
`nextLong()`	reads a `long` value from the user

```
import java.util.Scanner;

class Main {
  public static void main(String[] args) {

    // creates a Scanner object
    Scanner input = new Scanner(System.in);

    System.out.println("Enter an integer: ");
```

```
    // reads an int value
    int data1 = input.nextInt();

    System.out.println("Using nextInt(): " + data1);

    input.close();
  }
}
```

_____

```
import java.util.Scanner;

class Main {
  public static void main(String[] args) {

    // creates an object of Scanner
    Scanner input = new Scanner(System.in);
    System.out.print("Enter Double value: ");

    // reads the double value
    double value = input.nextDouble();
    System.out.println("Using nextDouble(): " + value);

    input.close();
  }
}
```

_____

```
import java.util.Scanner;

class Main {
  public static void main(String[] args) {

    // creates an object of Scanner
    Scanner input = new Scanner(System.in);
    System.out.print("Enter your name: ");

    // reads the entire word
    String value = input.next();
    System.out.println("Using next(): " + value);

    input.close();
  }
```

```
}
```

```
Enter your name: Jonny Walker
Using next(): Jonny
```

```
import java.util.Scanner;

class Main {
  public static void main(String[] args) {

    // creates an object of Scanner
    Scanner input = new Scanner(System.in);
    System.out.print("Enter your name: ");

    // reads the entire line
    String value = input.nextLine();
    System.out.println("Using nextLine(): " + value);

    input.close();
  }
}
```

```
public class Powers
{
public static void main(String[] args)
{
// I will use println for simple fixed text
System.out.println("Table of powers");
for (int i=0; i<10; i++)
// .. and printf to incorporate values within a template
{ System.out.printf("%dˆ%d = %d%n", i, i,
power(i, i));
}
}
static int power(int x, int n)
{ if (n == 0) return 1;
int y = power(x, n/2);
if ((n % 2) != 0) return x*y*y;
else return y*y;
```

```
}
}
```

Control Statements - The if Statement

```
if(condition) statement;
else statement;
```

Note:

- **else** clause is optional
- targets of both the **if** and **else** can be blocks of statements.

The general form of the **if**, using blocks of statements, is:

```
if(condition)
{
statement sequence
}
else
{
statement sequence
}
```

```
class Guess2 {
public static void main(String args[])
throws java.io.IOException {
char ch, answer = 'K';
System.out.println("I'm thinking of a letter between A and Z.");
System.out.print("Can you guess it: ");
ch = (char) System.in.read(); // get a char
if(ch == answer) System.out.println("** Right **");
else System.out.println("...Sorry, you're wrong.");
}
}
```

```
class Guess3 {
public static void main(String args[])
throws java.io.IOException {
char ch, answer = 'K';
System.out.println("I'm thinking of a letter between A and Z.");
System.out.print("Can you guess it: ");
ch = (char) System.in.read(); // get a char
if(ch == answer) System.out.println("** Right **");
else {
System.out.print("...Sorry, you're ");
// a nested if
if(ch < answer) System.out.println("too low");
else System.out.println("too high");
```

```
}
}
}
```

A sample run is shown here:

```
I'm thinking of a letter between A and Z.
Can you guess it: Z
...Sorry, you're too high
```

if-else-if Ladder

```
if(condition)
statement;
else if(condition)
statement;
else if(condition)
statement;
...
else
statement;
```

```
// Demonstrate an if-else-if ladder.
class Ladder {
public static void main(String args[]) {
int x;
for(x=0; x<6; x++) {
if(x==1)
System.out.println("x is one");
else if(x==2)
System.out.println("x is two");

else if(x==3)
System.out.println("x is three");
else if(x==4)
System.out.println("x is four");
else
System.out.println("x is not between 1 and 4");
}
}
}
```

The program produces the following output:

```
x is not between 1 and 4
x is one
x is two
x is three
x is four
x is not between 1 and 4
```

Declared as follows:

```
Exp1 ? Exp2 : Exp3;
```

Exp1 would be a **boolean** expression, and Exp2 and Exp3 are expressions of any type other than void. The type of Exp2 and Exp3 must be the same, though. Notice the use and placement of the colon. Consider this example, which assigns absval the absolute value of val:

```
absval = val < 0 ? -val : val; // get absolute value of val
```

Here, absval will be assigned the value of val if val is zero or greater. If val is negative, then absval will be assigned the negative of that value (which yields a positive value)

The same code written using the **if-else** structure would look like this:

```
if(val < 0) absval = -val;
else absval = val;
```

e.g. 2 This program divides two numbers, but will not allow a division by zero.

```
// Prevent a division by zero using the ?.
class NoZeroDiv {
public static void main(String args[]) {
int result;
for(int i = -5; i < 6; i++) {
result = i != 0 ? 100 / i : 0;
if(i != 0)
System.out.println("100 / " + i + " is " + result);
}
}
}
```

The output from the program is shown here:

```
100 / -5 is -20
100 / -4 is -25
100 / -3 is -33
100 / -2 is -50
100 / -1 is -100
100 / 1 is 100
100 / 2 is 50
100 / 3 is 33
100 / 4 is 25
100 / 5 is 20
```

---

switch Statement (case of)

The **switch** provides for a multi-way branch. Thus, it enables a program to select among several alternatives. Although a series of nested **if** statements can perform multi-way tests, for many situations the **switch** is a more efficient approach.

```
switch(expression) {
case constant1:
statement sequence
break;
case constant2:
statement sequence
```

```
break;
case constant3:
statement sequence
break;
...
default:
statement sequence
}
```

```
// Demonstrate the switch.
class SwitchDemo {
public static void main(String args[]) {
int i;
for(i=0; i<10; i++)
switch(i) {
case 0:
System.out.println("i is zero");
break;
case 1:
System.out.println("i is one");
break;
case 2:
System.out.println("i is two");
break;
case 3:
System.out.println("i is three");
break;
case 4:
System.out.println("i is four");
break;
default:
System.out.println("i is five or more");
}
}
}
```

The output produced by this program is shown here:

```
i is zero
i is one
i is two
i is three
i is four
i is five or more
i is five or more
i is five or more
i is five or more
i is five or more
```

Execution will continue into the next **case** if no **break** statement is present.
You can have empty **case**s, as shown in this example:

```
switch(i) {
case 1:
case 2:
case 3: System.out.println("i is 1, 2 or 3");
break;
case 4: System.out.println("i is 4");
break;
}
```

Nested switch

```
switch(ch1) {
case 'A': System.out.println("This A is part of outer
switch.");
switch(ch2) {
case 'A':
System.out.println("This A is part of inner
switch");
break;
case 'B': // ...
} // end of inner switch
break;
case 'B': // ...
```

---

he for Loop

Loops are structures used to make the program repeat one or many instructions for 'n' times as specified in the declaration of the loop.
The for Loop can be used for just one statement:

```
for(initialization; condition; iteration) statement;
```

or to repeat a block of code:

```
for(initialization; condition; iteration)
{
statement sequence
}
```

•	*Initialization* = assignment statement that sets the initial value of the *loop control variable*, (counter) *Condition* = Boolean expression that determines whether or not the loop will repeat *Iteration* = amount by which the loop control variable will change each time the loop is repeated
• •	

```
// Show square roots of 1 to 99 and the rounding error.
class SqrRoot {
public static void main(String args[]) {
double num, sroot, rerr;
```

```
for(num = 1.0; num < 100.0; num++) {
sroot = Math.sqrt(num);
System.out.println("Square root of " + num +
" is " + sroot);
// compute rounding error
rerr = num - (sroot * sroot);
System.out.println("Rounding error is " + rerr);
System.out.println();
}
}
}
```

'For' loop counters (loop control variables) can either increment or decrement,

```
// A negatively running for loop.
class DecrFor {
public static void main(String args[]) {
int x;
for(x = 100; x > -100; x -= 5)
System.out.println(x);
}
}
```

```
// Loop until an S is typed.
class ForTest {
public static void main(String args[])
throws java.io.IOException {
int i;
System.out.println("Press S to stop.");
for(i = 0; (char) System.in.read() != 'S'; i++)
System.out.println("Pass #" + i);
}
}
```

Interesting For Loop Variations

It is possible to leave out parts of the loop declaration:

```
// Example 1 - Parts of the for can be empty.
class Empty {
public static void main(String args[]) {
int i;
for(i = 0; i < 10; ) {
System.out.println("Pass #" + i);
i++; // increment loop control var
}} }
```

```
// Example 2 - Parts of the for can be empty.
class Empty2 {
public static void main(String args[]) {
```

```
int i;
i = 0; // move initialization out of loop
for(; i < 10; ) {
System.out.println("Pass #" + i);
i++; // increment loop control var
} } }
```

Initialising the loop out of the 'for' statement is only required when the value needs to be a result of another complex process which cannot be written inside the declaration.

Infinite Loops

Sometimes one needs to create an infinite loop, i.e. a loop which never ends! (However it can be stopped using the break statement). An example of an infinite loop declaration is as follows:

```
for(;;)
{
// … statements
}
```

*N.B. Using **break** to terminate an infinite loop will be discussed later on in the course.*

No 'Body' Loops

Loops can be declared without a body. This can be useful in particular situations, consider the following example:

```
// Loop without body.
class Empty3 {
public static void main(String args[]) {
int i;
int sum = 0;
// sum the numbers through 5
for(i = 1; i <= 5; sum += i++) ;
System.out.println("Sum is " + sum);
} }
```

Predicted Output:

```
Sum is 15
```

```
// Declare loop variable inside the for.
class ForVar {
public static void main(String args[]) {
int sum = 0;
int fact = 1;
// compute the factorial of the numbers through 5
for(int i = 1; i <= 5; i++) {
sum += i; // i is known throughout the loop
fact *= i;
}
// but, i is not known here.
System.out.println("Sum is " + sum);
System.out.println("Factorial is " + fact);
}
}
```

Class Fundamentals

Definition

A class is a sort of template which has attributes and methods. An object is an instance of a class,

```
class classname {
// declare instance variables
type var1;
type var2;
// ...
type varN;
// declare methods
type method1(parameters) {
// body of method
}
type method2(parameters) {
// body of method
}
// ...
type methodN(parameters) {
// body of method
}
}
```

```
class Vehicle {
int passengers; //number of passengers
int fuelcap; //fuel capacity in gallons
int mpg; //fuel consumption
}
```

Please note that up to this point there is no OBJECT. By typing the above code a new data type is created which takes three parameters. To create an instance of the Vehicle class we use the following statement:

```
Vehicle minivan = new Vehicle ();
```

To set the values of the parameters we use the following syntax:

```
minivan.fuelcap = 16; //sets value of fuel capacity to 16
```

Note the general form of the previous statement: *object.member*

Using the *Vehicle* class

Having created the Vehicle class, let us create an instance of that class:

```
class VehicleDemo {
public static void main(String args[]) {
Vehicle minivan = new Vehicle();
int range;
// assign values to fields in minivan
minivan.passengers = 7;
 minivan.mpg = 21;
```

Till now we have created an instance of Vehicle called 'minivan' and assigned values to passengers, fuel capacity and fuel consumption. Let us add some statements to work out the distance that this vehicle can travel with a tank full of fuel:

```
// compute the range assuming a full tank of gas
range = minivan.fuelcap * minivan.mpg;
System.out.println("Minivan can carry " +
minivan.passengers + " with a range of " + range);
}
}
```

Creating more than one instance

It is possible to create more than one instance in the same program, and each instance would have its own parameters. The following program creates another instance, *sportscar*, which has different instance variables and finally display the range each vehicle can travel having a full tank.

```
class TwoVehicles {
public static void main(String args[]) {
Vehicle minivan = new Vehicle();
Vehicle sportscar = new Vehicle();
int range1, range2;
// assign values to fields in minivan
minivan.passengers = 7;
minivan.fuelcap = 16;
minivan.mpg = 21;

// assign values to fields in sportscar
sportscar.passengers = 2;
sportscar.fuelcap = 14;
sportscar.mpg = 12;
// compute the ranges assuming a full tank of gas
range1 = minivan.fuelcap * minivan.mpg;
range2 = sportscar.fuelcap * sportscar.mpg;
System.out.println("Minivan can carry " +
minivan.passengers +
" with a range of " + range1);
System.out.println("Sportscar can carry " +
sportscar.passengers +
" with a range of " + range2);
}
}
```

Creating Objects

In the previous code, an object was created from a class. Hence 'minivan' was an object which was created at run time from the 'Vehicle' class – vehicle minivan = **new** Vehicle( ) ; This statement allocates a space in memory for the object and it also creates a reference. We can create a reference first and then create an object:

```
Vehicle minivan; // reference to object only
minivan = new Vehicle ( ); // an object is created
```

Reference Variables and Assignment

Consider the following statements:

```
Vehicle car1 = new Vehicle ( );
Vehicle car2 = car 1;
```

We have created a new instance of type Vehicle named car1. However note that car2 is **NOT** another instance of type Vehicle. car2 is the same object as car1 and has been assigned the same properties,

```
car1.mpg = 26; // sets value of mpg to 26
```

If we had to enter the following statements:

```
System.out.println(car1.mpg);
System.out.println(car2.mpg);
```

The expected output would be 26 twice, each on a separate line

```
Vehicle car1 = new Vehicle();
Vehicle car2 = car1;
Vehicle car3 = new Vehicle();
car2 = car3; // now car2 and car3 refer to the same object.
```

Methods

Methods are the functions which a particular class possesses. These functions usually use the data defined by the class itself.

```
// adding a range() method
class Vehicle {
int passengers; // number of passengers
int fuelcap; // fuel capacity in gallons
int mpg; // fuel consumption in miles per gallon
// Display the range.
void range() {
System.out.println("Range is " + fuelcap * mpg);
}
}
```

Note that 'fuelcap' and 'mpg' are called directly without the dot (.) operator. Methods take the following general form:

```
ret-type name( parameter-list ) {
// body of method
}
```

'ret-type' specifies the type of data returned by the method. If it does not return any value we write **void**. 'name' is the method name while the 'parameter-list' would be the values assigned to the variables of a particular method (empty if no arguments are passed).

```
class AddMeth {
public static void main(String args[]) {
```

```
Vehicle minivan = new Vehicle();
Vehicle sportscar = new Vehicle();
int range1, range2;
// assign values to fields in minivan
minivan.passengers = 7;
minivan.fuelcap = 16;
minivan.mpg = 21;
// assign values to fields in sportscar
sportscar.passengers = 2;
sportscar.fuelcap = 14;
sportscar.mpg = 12;
System.out.print("Minivan can carry " +
minivan.passengers + ". ");
minivan.range(); // display range of minivan
System.out.print("Sportscar can carry " +
sportscar.passengers + ". ");
sportscar.range(); // display range of sportscar.
}
}
```

Main program:

```
class RetMeth {
public static void main(String args[]) {
Vehicle minivan = new Vehicle();
Vehicle sportscar = new Vehicle();
int range1, range2;
// assign values to fields in minivan
minivan.passengers = 7;
minivan.fuelcap = 16;
minivan.mpg = 21;
// assign values to fields in sportscar
sportscar.passengers = 2;
sportscar.fuelcap = 14;
sportscar.mpg = 12;
// get the ranges
range1 = minivan.range();
range2 = sportscar.range();
System.out.println("Minivan can carry " +
minivan.passengers + " with range of " + range1 + "
Miles");
System.out.println("Sportscar can carry " +
sportscar.passengers + " with range of " + range2 +
" miles"); }
}
```

Updated vehicle class:

```
class Vehicle {
int passengers; // number of passengers
int fuelcap; // fuel capacity in gallons
int mpg; // fuel consumption in miles per gallon
// Return the range.
int range() {
return mpg * fuelcap;
}
// Compute fuel needed for a given distance.
double fuelneeded(int miles) {
return (double) miles / mpg;
}  }
```

Main Program:

```
class CompFuel {
public static void main(String args[]) {
Vehicle minivan = new Vehicle();
Vehicle sportscar = new Vehicle();
double gallons;
int dist = 252;
// assign values to fields in minivan
minivan.passengers = 7;

minivan.fuelcap = 16;
minivan.mpg = 21;
// assign values to fields in sportscar
sportscar.passengers = 2;
sportscar.fuelcap = 14;
sportscar.mpg = 12;
gallons = minivan.fuelneeded(dist);
System.out.println("To go " + dist + " miles minivan
needs " +
gallons + " gallons of fuel.");
gallons = sportscar.fuelneeded(dist); //overwriting same
variable
System.out.println("To go " + dist + " miles sportscar
needs " +
gallons + " gallons of fuel.");
}
}
```

Predicted Output:

```
To go 252 miles minivan needs 12.0 gallons of fuel.
To go 252 miles sportscar needs 21.0 gallons of fuel.
```