5. High Level Languages Features

A high-level computer language is one which is problem-oriented and machine-independent. Machine independent means that the programmer of a high-level language need not know the architecture of the computer (unlike for example the assembly-language programmer). All one needs to know is the grammar of the language. Some languages, like Java, are called general purpose because by means of them one can program various programs in different fields. Others are written for particular areas for example FORTRAN is used for mathematical applications and COBOL is used for commercial applications.

5.1 Comments

High-level languages allow that the programmer insert comments in the program. These comments are ignored when the program is translated from 'source code' (the program as written by the programmer) to 'object code' (the program as understood by the computer). Comments make the program more readable.

5.2 Identifiers

Identifiers are 'names' we give to our variables, constants, classes, and methods.

Java identifiers must comply with these rules:

- The first character of an identifier must be a letter, an underscore (_), or a dollar sign ($).
- The rest of the characters in the identifier can be a letter, underscore, dollar sign, or digit. Note that spaces are NOT allowed in identifiers.
- Identifiers are case-sensitive. This means that 'age' and 'Age' are different identifiers.
- Identifiers cannot match any of Java's reserved words.

For example these identifiers are valid: 'MyClass', '$amount', 'totalGrades', 'TotalGrades', 'TAX_RATE' and 'one'.

These identifiers are NOT valid: 'My Class' (spaces are not allowed), 'numberOf*s' (the asterisk cannot be used), 'final' ('final' is a reserved word), '1Way' (identifiers cannot start with a digit).

5.3 Reserved Words

These words are part of the syntax of the high-level language. Each holds a specific meaning and cannot be used as an identifier.

The following table shows the reserved words of Java.

abstract	else	int	strictfp
assert	enum	interface	super
boolean	extends	long	switch
break	false	native	synchronized
byte	final	new	this
case	finally	null	throw
catch	float	package	throws
char	for	private	transient
class	goto	protected	true
const	if	public	try
continue	implements	return	void
default	import	short	volatile
do	instanceof	static	while
double			

Diagram 60: Java Reserved Words

5.4 ASCII and Unicode

Both ASCII and Unicode are codes that convert characters (like ‘n’, ‘N’, ‘@’, ‘+’ etc) and control codes (like ‘backspace’, ‘carriage return’, ‘shift in’, ‘shift out’ etc) into a number. As we know the computer expresses everything in numbers (in binary).

ASCII is an acronym for the ‘American Standard Code for Information Interchange’. ASCII is a code for representing English characters as numbers, with each letter assigned a number from 0 to 127 (using 7 bits). For example, the ASCII code for uppercase M is 77. Most computers use ASCII codes to represent text, which makes it possible to transfer data from one computer to another.

Text files stored in ASCII format are sometimes called ASCII files.

The ‘standard ASCII’ character set uses just 7 bits for each character. There are several larger character sets that use 8 bits, which gives them 128 additional characters. The extra characters are used to represent non-English characters (like ö, ū, etc), graphics symbols (like □, ◇, ●, etc), and mathematical symbols (like ⅛, √, ∫, etc). The DOS operating system uses a superset of ASCII called ‘extended ASCII’ or ‘high ASCII’. Extended ASCII uses 8 bits to represent a character.

A more universal standard is the ‘ISO Latin 1’ set of characters (8-bit), which is used by many operating systems, as well as Web browsers. ISO Latin-1 is a superset of the ASCII character set and is very similar to the ANSI character set used in Windows, though the two are not identical.

Another set of codes that is used on large IBM computers is EBCDIC (Extended Binary-Coded Decimal Interchange Code). The EBCDIC uses 8-bit representations.

Unicode, unlike ASCII, which uses 7 bits for each character, Unicode uses 16 bits, which means that it can represent more than 65,000 unique characters. This is necessary so that other languages, such as Greek, Chinese and Japanese can have their symbols represented. Many analysts believe that as the software industry becomes increasingly global, Unicode will eventually supplant ASCII as the standard character coding format.

5.5 Escape Sequences

An escape sequence is a sequence of characters that have a special meaning. The following list shows the Java escape sequences:

\t	(insert a tab in the text at this point)
\b	(insert a backspace in the text at this point)
\n	(insert a ‘newline’ in the text at this point)
\r	(insert a carriage return in the text at this point)
\f	(insert a ‘formfeed’ in the text at this point) (A formfeed is special character that causes the printer to advance one page length or to the top of the next page.)
\'	(insert a single quote character in the text at this point)
\"	(insert a double quote character in the text at this point)
\\	(insert a backslash character in the text at this point)

The character ‘\’ is called the escape character.

When an escape sequence is encountered in a print statement, the compiler interprets it accordingly. For example, if you want to put quotes within quotes you must use the escape sequence, \", on the interior quotes. To print the sentence

She said "Hello!" to me.

you would write:

```
System.out.println("She said \"Hello!\" to me.");
```

5.6 Literals

In computer science, a literal is a notation for representing a fixed value in source code. Examples are 4 (an integer literal), 56.7 (a floating point literal), “this is a string” (a string literal), True (a Boolean literal) etc.

5.7 Variables

The idea of a variable is that of having a value (with a given name and type) that can change during the execution of the program. The possibility of using variables in a program is a very powerful tool that makes most programs possible.

In the Java programming language (and in many others) all variables must first be declared before they can be used. This involves stating the variable's type and name as shown here:

```
int gear;
```

The above declaration tells your program that you will be using a variable of type integer (a whole number) whose name is gear. During the execution of the program gear can have values like 9, 7620, -98 but not 7.65. You can give an initial value to the variable during declaration as is shown in the following line:

```
int gear = 1;
```

5.7.1. Scope of Variables

A variable's ‘scope’ refers to those parts of the program in which the variable can be used. Another term for scope is ‘visibility’.

Look at the following program in Java.

```
class scope
{
   static int a = 7;

   public static void main (String args[])
   {
      int b = 2;

      System.out.println ( a );

      System.out.println ( b );

      // System.out.println ( c );
      // This gives an error because c is defined
      // within meth1 and is not known within this
      // method

      meth1();
      meth2();
   }

   static void meth1()
   {
```

```
            int c = 5;

            System.out.println ( a );
            // System.out.println ( b ); Error
            System.out.println ( c );
        }

        static void meth2()
        {
            int d = 4;

            for ( int i = 0; i<4; i++ )
                {
                    System.out.println ( d*i*a );
                }

            System.out.println ( a );
            // System.out.println ( b ); Error
            // System.out.println ( c ); Error
            System.out.println ( d );
            // System.out.println ( i ); Error because i
            // is defined for use in the 'for' loop only.
        }
    }
```

Diagram 61: A Program for studying the Scope of Variables

The following table gives the scopes of all the variables used in the above program.

Variable	Scope
a	The scope is the whole program because the variable is like a global variable.
b	The scope of ‘b’ is only the method ‘main’ since ‘b’ is defined within this method. Outside ‘main’ the variable ‘b’ cannot be accessed.
c	The variable ‘c’ is visible only inside method ‘meth1’ because it is inside this method that it is declared.
d	The variable ‘d’ is visible only inside method ‘meth2’ because it is inside this method that it is declared.
i	The scope of ‘i’ is only the ‘for’ loop within the method meth2.

Diagram 62: A Table that shows the Scope of Variables in the Program in diagram 61

It must be added that within a block variables can be declared at any point. Variables are created when their declaration is met and destroyed when the block in which they are created is exited. Therefore a variable is valid in a block from the line of its declaration till the end of the block.

In Java, although blocks can be nested, if two variables are created, one in an outer block and another in an inner block, these cannot have the same name.

In Java variables have 'modifiers'. These modifiers impose certain settings on the variables. One of these settings involves the scope of the variable. For example one modifier is 'public'. Here is an example:

```
public int b = 2;
```

This modifier indicates that the variable 'b' is accessible anywhere its class is. Some of the other modifiers are 'protected' (the variable is accessible only within its package and its subclasses), 'private' (variable is accessible only in the class which defins it) and 'final' (indicates that the variable cannot change its value). When there is no modifier this is considered as the default (or package) modifier. In this case the variable is accessible only in its package.

5.8 Constants

A constant is a value that does not change. In Java, as shown above, it is indicated by the modifier 'final'. The declaration

```
final int f = 6;
```

states that all occurances of 'f' in the program will be changed to 6 by the compiler.

5.9 Data Types

A 'type' is a set of values and operations on these values. For example the type 'int' has values {…, -2, -1, 0, 1, 2, …} and operations {+, -, *, /, MOD, DIV,…}. The type 'boolean' has values {true, false} and operations {AND, OR, NOT, …}.

Some data types are referred to as 'simple' data types (or 'atomic' or 'primitive'). A variable of a simple data type is made up of one value.

Some data types are called 'structured' because a variable declared of having such a type is made up of more than one value (i.e. it is a data structure). Examples are array, string or record.

Other types are called 'reference' types because a variable declared with such types will hold the 'reference' ('address' or 'pointer') to the (structured) value.

5.10 Java Primitive Types and Reference Types

The Java programming language is a ‘strongly typed’ language, which means that every variable and every expression has a type that is known at compile time. Strong typing helps detect errors at compile time.

The types of the Java programming language are divided into two categories:

- primitive types
- reference types.

5.10.1. Java’s Primitive Types

The eight primitive data types supported by the Java programming language are:

- **byte**: The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive).
- **short**: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive).
- **int**: The int data type is a 32-bit signed two's complement integer. It has a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive).
- **long**: The long data type is a 64-bit signed two's complement integer. It has a minimum value of -9,223,372,036,854,775,808 and a maximum value of 9,223,372,036,854,775,807 (inclusive).
- **float**: The float data type is a real number that ranges from approximately -3.40E+38 to 3.40E+38. It is a single-precision 32-bit floating point.
- **double**: The double data type is a double-precision 64-bit floating point. Its range of values is approximately from -1.80E+308 to 1.80E+308.
- **boolean**: The boolean data type has only two possible values: true and false. This data type represents one bit of information, but its "size" isn't something that's precisely defined.
- **char**: The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

5.10.2. Java’s Reference Type

There are three kinds of reference types:

- class types
- interface types
- array types

Basically, any type that is not a primitive is a reference. That means that arrays are references, as are instances of classes i.e. if you declare a variable ‘xyz’ as an array the value contained in xyz is not the array but a reference (pointer, memory address) to where the array is.

5.11 The Java String Type

Consider the following program in Java:

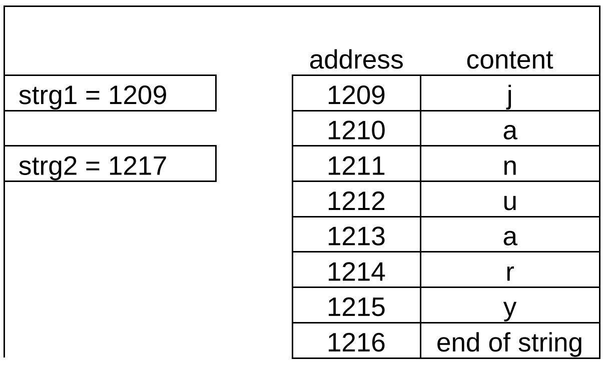
```
public class AboutStrings2
{
   public static void main (String args[])
   {
      String st1, st2;
      st1 = "november";
      st2 = "november";
      System.out.println (st1 == st2); //outputs 'true'
      System.out.println (st1.equals(st2)); //outputs 'true'

      String str1 = new String();
      String str2 = new String();
      str1 = "december";
      str2 = "december";
      System.out.println (str1 == str2); //outputs 'true'
      System.out.println (str1.equals(str2)); // outputs 'true'

      String strg1 = new String("january");
      String strg2 = new String("january");
      System.out.println (strg1 == strg2); //outputs 'false'
      System.out.println (strg1.equals(strg2)); // outputs 'true'
   }
}
```

Diagram 63: A Program about Equating String Variables

Note that when using the == operator st1, st2, str1 and str2 are treated like variables of primitive type while strg1 and strg2 are treated as variables of reference type. Consider the diagram below.



strg1 = 1209

strg2 = 1217

address	content
1209	j
1210	a
1211	n
1212	u
1213	a
1214	r
1215	y
1216	end of string

1217	j
1218	a
1219	n
1220	u
1221	a
1222	r
1223	y
1224	end of string

Diagram 64: How Two Strings are Memorised

The values of strg1 and strg2 are respectively 1209 and 1217. So strg1 is not equal to strg2. It is the content of their reference that is equal.

5.12 Enumerated Types

An enum type is a type whose fields consist of a fixed set of constants e.g. values of NORTH, SOUTH, EAST, and WEST. Because they are constants, the names of an enum type's fields are in uppercase letters.

In the Java programming language, you define an enum type by using the enum keyword as shown in the program below.

```
public class EnumTest
{
   public enum Day { SUNDAY, MONDAY, TUESDAY, WEDNESDAY,
                           THURSDAY, FRIDAY, SATURDAY }
   Day day; // day is a variable of the enumerated type Day

   public EnumTest(Day day) // constructor
   {
      this.day = day;
   }

   public void tellItLikeItIs()
   {
      switch (day)
      {
         case MONDAY: System.out.println("Mondays are bad.");
                  break;

         case FRIDAY: System.out.println("Fridays are better.");
                  break;

         case SATURDAY:
         case SUNDAY: System.out.println("Weekends are best.");
                  break;
```

```
        default:     System.out.println("Midweek days are so-so.");
                    break;
      }
   }

   public static void main(String[] args)
   {
     EnumTest firstDay = new EnumTest(Day.MONDAY);
     // An object called firstDay is created by means of the
     // constructor.  The variable 'day' in this object is equal to
     // MONDAY.
      firstDay.tellItLikeItIs();
     // The method 'tellItLikeItIs' for the object 'firstDay' is called.

      EnumTest thirdDay = new EnumTest(Day.WEDNESDAY);
      thirdDay.tellItLikeItIs();

      EnumTest fifthDay = new EnumTest(Day.FRIDAY);
      fifthDay.tellItLikeItIs();

      EnumTest sixthDay = new EnumTest(Day.SATURDAY);
      sixthDay.tellItLikeItIs();

      EnumTest seventhDay = new EnumTest(Day.SUNDAY);
      seventhDay.tellItLikeItIs();
   }
}
```

Diagram 65: A Program Showing Enumerated Types

The output is:

```
Mondays are bad.
Midweek days are so-so.
Fridays are better.
Weekends are best.
Weekends are best.
```

5.13 Other Classes as Types

Java itself is enriched with packages that hold classes that can be used by the user. There are hundreds of these classes. We mention only three. 'Scanner' is a class that can be used to enter information from the keyboard. 'LinkedList' is a class that implements the data structure Linked List. 'BigDecimal' is a class that is used when accurate arithmetic on very large numbers is required. A programmer can also define classes (called user-defined).

Most classes can be used as types where objects are created according to the definition of the class. This is analogous to creating a variable having a particular type. An object is a more complex comparison to a variable. The type of an object is a class.

Look at the following program that is made up of three classes.

```
class taxi
{
   //the taxi is paid at 1.2 euros per kilometre
   private final float RATE_PER_KM = 1.2f;

   private int kilom;

   //this is a constructor
   public taxi ()
   {
      kilom = 0;
   }

   //a second constructor
   public taxi (int k)
   {
      kilom = k;
   }

   //accessor
   public float getRate ()
   {
      return RATE_PER_KM;
   }

   //another accessor
   public int getKilom ()
   {
      return kilom;
   }

   public float payment ()
   {
      return kilom * RATE_PER_KM;
   }
}

class hiredCar
{
   //the hired car is paid 5c per kilometre
   private final float RATE_PER_KM = 0.05f;

   //the hired car is paid 5 euros per day
   private final float RATE_PER_DAY = 5.0f;
```

```
    private int kilom;

    private int days;

    //this is a constructor
    public hiredCar ()
    {
        kilom = 0;
        days = 0;
    }

    //a second constructor
    public hiredCar (int k, int d)
    {
        kilom = k;
        days = d;
    }

    //accessor
    public float getRatePerKm ()
    {
        return RATE_PER_KM;
    }

    //another accessor
    public float ratePerDay ()
    {
        return RATE_PER_DAY;
    }

    //another accessor
    public int getKilom ()
    {
        return kilom;
    }

    //another accessor
    public int getDays ()
    {
        return days;
    }

    public float payment ()
    {
        return (kilom * RATE_PER_KM + days * RATE_PER_DAY);
    }
}
```

```
import java.util.*;

class payments

{
   public static void main (String args[])
   {
      int choice;

      Scanner scan = new Scanner (System.in);
      scan.useDelimiter ("\n");

      do
      {
         System.out.println ();
         System.out.println ("PAYMENTS");
         System.out.println ();
         System.out.println ("1. Payment for Taxi");
         System.out.println ("2. Payment for Hired Car");
         System.out.println ("0. End Program");
         System.out.println ();
         System.out.print ("Enter your choice : ");
         choice = scan.nextInt();

         switch (choice)
         {
            case 1 : payForTaxi();
                   break;
            case 2 : payForHiredCar();
                   break;
         }
      }while (choice != 0);
   }

   static void payForTaxi ()
   {
      int km;
      float pay;
      taxi t;

      Scanner scan = new Scanner (System.in);
      scan.useDelimiter ("\n");

      System.out.println ();
      System.out.print ("How many kilometres? : ");
      km = scan.nextInt();

      t = new taxi (km);
      pay = t.payment();
```

```
            System.out.println ();
            System.out.println ("Payment = " + pay);
        }

        static void payForHiredCar ()
        {
            int km, dys;
            float pay;
            hiredCar hc;

            Scanner scan = new Scanner (System.in);
            scan.useDelimiter ("\n");

            System.out.println ();
            System.out.print ("How many kilometres? : ");
            km = scan.nextInt();

            System.out.println ();
            System.out.print ("How many days? : ");
            dys = scan.nextInt();

            hc = new hiredCar (km, dys);
            pay = hc.payment();

            System.out.println ();
            System.out.println ("Payment = " + pay);
        }
    }
```

Diagram 66: A Program in which Classes are created and then Objects are created from them

The three classes ‘taxi’, ‘hiredCar’ and ‘payments’ are all user defined. The class ‘payments’ defines 3 objects whose type is a class. For now look at a class as a complex data type and look at an object as a complex kind of variable whose type is a class. The following table shows which objects are used in the class ‘payments’

class	**object**
Scanner (this class is defined by Java)	scan
Taxi (this class is user-defined)	t
hiredCar (this class is user-defined)	hc

Diagram 67: The Objects created in the Program of diagram 66

5.14 Expressions and Operators

Expressions are phrases made up of operators and operands. An expression is evaluated to give an overall result. The most common expressions are the arithmetical ones. Look at the arithmetic expression 3.4 – 4.5 * (3 – 1.2)

The operators in this expression are minus (used twice) and multiplication. The operands are 3.4, 4.5, 3 and 1.2. This expression evaluates to -4.7. The following table shows some different types of expressions, their operators, operands and evaluation.

Expression	Type of Expression	Operators	Operands	Evaluation
3 + 53 mod 4	integer	+, mod	3, 53, 4	6 (int)
4.2 + 3.1 * 4.0	float	+, *	4.2, 3.1, 4.0	16.6 (float)
c++ (where c=’p’)	character	++	c	‘q’ (char)
“wind” + “mill”	string	+ (concatenation not plus)	+	“windmill”
True And False	Boolean	And	True, False	False (Boolean)
9>8	inequality	>	9, 8	True (Boolean)

Diagram 68: Expressions

5.15 Statements

A statement is an instruction that tells the computer what to do. Some statements are simple like a++ but others are more complex like an if-then-else statement or a loop statement. Statements can get more complex if one is nested inside another.

5.16 Precedence of Operators

Consider the arithmetic expression 2 + 3 * 4. This works out to 14. The multiplication must be worked out before the addition. If we switch the order of the operators and work out the addition first we arrive at the erroneous result of 20. So we say that multiplication has precedence over addition. In our secondary-school years this was called the BODMAS rule. Let us consider another example: 4>5-3. The minus operator must be worked out first. So we say that “-” has a precedence over “>”.

5.17 Associativity of Operators

Suppose we have the expression A AND B AND C. Which AND should be calculated first? The one on the left or the one on the right? Associativity tells us whether the one on the left should be worked out first (left-to-right) or the one on the right (right-to-left).

The following table gives the precedence and associativity rules of the Java operators.

Operator	Description	Associativity
`()`	Parentheses (grouping)	left-to-right
`++ --`	Unary postincrement/postdecrement	right-to-left
`++ --`	Unary preincrement/predecrement	right-to-left
`+ -`	Unary plus/minus	
`! ~`	Unary logical negation/bitwise complement	
`(type)`	Unary cast (change *type*)	
`* / %`	Multiplication/division/modulus	left-to-right
`+ -`	Addition/subtraction	left-to-right
`<< >>`	Bitwise shift left, Bitwise shift right	left-to-right
`< <=`	Relational less than/less than or equal to	left-to-right
`> >=`	Relational greater than/greater than or equal to	
`instanceof`	Type comparison	
`== !=`	Relational is equal to/is not equal to	left-to-right
`&`	Bitwise AND	left-to-right
`^`	Bitwise exclusive OR	left-to-right
`|`	Bitwise inclusive OR	left-to-right
`&&`	Logical AND	left-to-right
`||`	Logical OR	left-to-right
`?:`	Ternary conditional	right-to-left
`=`	Assignment	right-to-left
`+= -=`	Addition/subtraction assignment	
`*= /=`	Multiplication/division assignment	
`%= &=`	Modulus/bitwise AND assignment	
`^= |=`	Bitwise exclusive/inclusive OR assignment	
`<<= >>=`	Bitwise shift left/right assignment	

Diagram 69: All the Jave Operators Ordered according to Priority with their corresponding Associativity Rules

5.18 Exercise

Evaluate the following expressions:

(a) 4 + 9 / 3
(b) 5 - 12 MOD 5 + 17 DIV 3
(c) 3.4 + 12.4 / 4 * 2.2
(d) “Keep” + “ it ” + “simple.”
(e) kar++ (where kar = ‘B’)
(f) “after” > “before”
(g) (3 - 4 < 12) OR (15=22)
(h) (‘i’ = ‘I’) OR (12 < 27) AND (9 <= 14)