UNIT 2 DATA TYPES, OPERATORS AND ARRAYS

Structure **Page Nos.**

2.0 INTRODUCTION

Data is one of the most important constituents of a computer program. Data to a computer can be numbers, characters or simply, values. Java has several different types of data that it can work with, and this chapter covers some of the most important of them.

Like any other programming language, Java supports its own set of data types. Implementation of the data types has been substantially cleaned up in the following ways:

1) Where many languages, like C and C++, leave a number of issues to be machine and compiler dependent (for instance the size of an int), Java specifies everything.

2) Java prevents casting between arbitrary variables. Only casts between numeric variables, and between sub and upper classes of the same object are allowed. In C programming language an arithmetic expression can combine different data types which produce result in higher data types.

3) All numeric variables in Java are signed.

4) The size of method is unnecessary in Java, because all sizes are precisely defined. For instance, an int is always 4 bytes. This may not seem to be adequate when dealing with objects that are not base data types. However, even if you did know the size of a particular object, you could not do anything with it, anyway. You cannot convert an arbitrary object into bytes, and back again.

2.1 OBJECTIVES

After going through this Unit, you will be able to:

- initialize variables of different data types;
- use datatypes – their usage and typecasting;
- use different types of operators and their hierarchy;
- understand decision construct conditional loops, and
- understand arrays.

2.2 DATA TYPES IN JAVA

In Java, there are two different categories into which data types have been divided:

primitive types
reference types

Primitive Data Types

Java has 8 primitive data types

Reserved Word	Data type	Size	Range of Values
byte	Byte-length Integer	1 byte	$-2^7$to 2^7-1
short	Short integer	2 bytes	$-2^{15}$to 2^{15}-1
int	Integer	4 bytes	-2^{31}to2^{31}-1
long	Long integers	8 bytes	$-2^{63}$to 2^{63}-1
float	Single precision number	4 bytes	-2^{31}to2^{31}-1
double	Real number with double precision	8 bytes	$-2^{63}$to 2^{63}-1
char	Character (16-bit Unicode)	2 bytes	0 to 2^{16}-1
boolean	Has value true or false	1 bytes	True or false

Identifiers in Java

Identifiers are the names of **variables, methods, classes, packages** and **interfaces.** Identifiers must be composed of **letters, numbers,** the **underscore_** and the **dollar sign $.** They cannot contain white spaces. Identifiers may only begin with a letter, the underscore or a dollar sign. You cannot begin a variable name with a number. All variable names are case sensitive, for example:

My **Variable** is not the same as my **Variable.**

There is no limit to the length of a Java variable name.

Character Variables

The syntax to create a character variable is:

Char ch = ‘b’;

In this example, the ‘ch’ variable has been assigned the value of the letter ‘b’. Notice the single quotes around the letter ‘b’: these tell the compiler that you want the literal value of ‘b’, rather than an identifier called ‘b’.

Integer Variables

The following lines show how to create variable of integer types:

```
byte x  = 10
short y = 15
int p   = 20
long q  = 25:
```

Boolean Variables

A boolean variable can have only one of two possible values, true or false. This is the type returned by all relational operators (>, < etc.).

Declaration:

boolean b = true:

Float Variables

The following lines show how to create variables of float types:

float f1 = 3.14:
float f2 = 1.402e-2

Literals

Literals are pieces of Java source code that indicate explicit values. For instance, Hello World! Is a String literal. Java has four kinds of literals: String, Character, Number, and Boolean.

String Literals

The string literal is always enclosed in double quotes. Java uses a **String** class to implement strings, whereas C and C++ use an array of characters. For example:

"Hello World!"

String message = "Hello World";

Character Literals

Character literals are similar to String literals except they are enclosed in single quotes and must have exactly one character. For example 'c' is a character literal. A backslash is used to denote the non-printing characters such as

Description	**Escape Sequence**
Line feed	\n
Carriage Return	\r
Horizontal tab	\t
Backslash	\\
Single quote	\'
Double quote	\"

Boolean Literal

A Boolean literal can have either of the values: **true or false**. They do not correspond to the numeric values, 1 and 0, as in C and C++.

Numeric Literals

There are two types of numeric literal: integers and floating point numbers. For example: 34 is an integer literal, and it means the number thirty four.

1.5 is a floating point literal.
45.6, 76.4E8 (76.4 times 10 to the 8th power) and 32.0 are also floating point literals.

2.3 OPERATORS

Operators are used to build expressions. They are described here in several related categories. Operators can be broadly categorised as:

a) **Relational and equality operators**

Relational operators are binary operators that require two operands to work on. The operands can be constants, variables, or expression. The operators are as follows:

>	greater than
>=	greater than, or equal to
<	less than
<=	less than, or equal to
!	Boolean NOT
!=	not equal to

b) **Assignment Operators**

=	assignment
^=	bitwise XOR and assign
&=	bitwise AND and assign
%=	take remainder and assign
=	subtract and assign
*=	multiply and assign
/=	divide and assign
|=	bitwise OR and assign
>>=	shift bits right with sign extension and assign
<<=	shift bits left and assign
>>>=	unsigned bit shift right and assign

c) **Arithmetic**

-	subtraction
x	multiplication
/	division
+	addition
%	modulo

d) **Bitwise**

|	bitwise OR
^	bitwise XOR
&	bitwise AND
>>	right shift
<<	left shift
~	bitwise NOT
>>>	unsigned bit shift right

e) **Increment and Decrement Operators**

++	increment by one
---	decrement by one

f) **Logical**

&&	Boolean AND
‖	boolean OR
═	Boolean equals
?:	conditional
!	not

Precedence of operators in Java

The following table lists all operators in Java, in order of their precedence. The first operator in the list has the highest precedence and the last operator has the least precedence.

Operator	Operator type	Description
[ ]	Array index	Used to access elements of an array
( )	Parameter list	Denotes a list of parameters
.	Method invocation	Used to specify a method within an object (or its hierarchy)
++,—	Arithmetic	Pre-or Post-increment/decrement
+. -	Arithmetic	Unary Plus, unary minus
~	Integral	Bitwise complement (unary)
!	Boolean	Logical Complement (unary)
(type)	Any	Cast
*,/,%	Arithmetic	Multiplication, Division, reminder
+,-	Arithmetic	Addition, Subtraction
+	String	String Concatenation
<<	Integral	Left shift
>>	Integral	Right shift with sign extension
>>>	Integral	Right shift with zero extension
<,<=	Arithmetic	Less than, Less than or equal
>,>=	Arithmetic	Greater than, Greater than or equal
Instance of	Object, type	Type comparison
==	Primitive	Equal (have identical values)
!=	Primitive	Not equal (have different values)
==	Object	Equal (refer to the same object)
!=	Object	Equal (refer to the different object)
&	Integral, Boolean	Bitwise AND
^	Integral, Boolean	Bitwise XOR
|	Integral, Boolean	Bitwise OR
&&	Boolean	Conditional AND
||	Boolean	Conditional OR

Sometimes the default order of evaluation is not what you want. For instance, the formula to change a Fahrenheit temperature to a Celsius temperature is C=(5/9) (F-32) where C is degrees Celsius, and F is degrees Fahrenheit. You must subtract 32 from the Fahrenheit temperature before you multiply by 5/9, not after. In such a case, you can use parentheses to adjust the order, much as they are used in the above formula.

☞ **Check Your Progress 1**

1) State True or False for the following statements:

	Reserved Word	Size	T / F
(i)	short	4 bytes	☐
(ii)	double	2 bytes	☐
(iii)	unicode	1 byte	☐
(iv)	int	8 bytes	☐
(v)	boolean	2 bytes	☐

2) List the important boolean operators.

…………………………………………………………………………...

…………………………………………………………………………...

…………………………………………………………………………...

3) Show the precedence of the following operators:

[ ] <> <> ~ !

…………………………………………………………………………...

…………………………………………………………………………...

…………………………………………………………………………...

……………………………………………………………………………

2.4 JAVA KEYWORDS

Keywords are identifiers, such as public, static and class that have a special meaning inside Java source code, and outside of comments and Strings. Keywords are reserved for their intended use and cannot be used by the programmer for variable, or method names. Some of the reserved keywords in Java are:

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

2.5 MIXING DATA TYPES

Besides combining different operations, you can mix and match different numeric data types on the same line. The program below uses both ints and doubles:

```
Class IntAndDouble
{
   public static void main (String args)
   {
         int I = 10;
         double x = 2.5;
         double k;
         System.out.printIn (“i is “ +i);
        System.out.printIn (“xis”+x)

         k= i + x
         System.out.printIn(I + x is “k);

         k = I + x:
         System.out.printIn(“I*x is “+k)

         k = i-x;
         System.out.printIn (“I-x is “+k);
        k=i/x;
        System.out.printIn("i/x is "+k);
    }
}
```

The output is as follows

```
i is 10
x is 2.5
i + x is 12.5
i * x is 25.0
i – x is 7.5
i/x is 4.0
```

2.6 TYPE CASTING

An int divided by an int is still an int, and a double divided by a double is still a double, but what about an int divided by a double or a double divided by an int? When doing arithmetic on different data types, Java tends to widen the types involved so as to avoid losing information.

The basic rule is that if any of the variables on the right hand side of an equal sign are doubles then Java treats all the values on the right hand side as doubles. If none of those values are doubles, but some are floats, then Java treats all the values as floats. If there are no floats or doubles, but there are longs, then Java treats all the values as longs.

If this is an assignment statement, i.e., if there is an equals sign, then Java compares the type of the left hand side to the final type of the right hand side. It will not change the type of the left hand side, but it will check to make sure that the value it has (double, float, int or long) can fit in the type on the left hand side. Anything can fit in a double. Anything except a double can fit in a float. Any integral type can fit in a long, but a float or a double cannot, and only on an int can fit inside an int. If the right hand side can fit inside the left hand side, the assignment takes place with no further ado.

However, if the right hand side may not be able to fit into the left hand side, then a series of operations take place to chop the right hand side down to size. For a conversion between a floating point number and an int or a long, the fractional part of the floating point number is truncated (rounded toward zero). This produces an integer. If the integer is small enough to fit in the left hand side, the process will stop here. The result will, thus, be a double, float, long or int, depending on what it was on the right hand side.

This can be a nasty bug in your code. It can also be hard to find since everything may work perfectly 99 times out of a hundred, and only on rare occasions will the rounding become a problem. However, when it does, there will be no warning or error message. You need to be very careful when assigning floating point values to integer types.

Assigning integer types to integer types, or double to floats can be equally troublesome when the right hand side is a lager number than the left hand side can hold. In fact, it is so troublesome that the compiler will not let you do it, unless you tell it that you really meant it with a cast. When it is necessary to force a value into a particular type, use a cast. To cast a variable or a literal or an expression to a different data type, just precede it with the type in parentheses. For instance:

```
int i = (int) 9.0/4.0
```

A cast lets the compiler know that you are serious about the conversion you plan to make.

Converting Strings to Numbers

When processing user input, it is often necessary to convert a String that the user enters into an int. The syntax is straightforward. It requires using the static Integer.valueOf (Strings) and int Value () methods from the java.lang.Integer class. To convert the string "22" into the int 22 you would write

```
int I = Integer.valueOf ("22").intValue();
```

Doubles, floats and long are converted similarly. To convert a String like "22" into the long value 22, you would write

```
long1 = Long.valueOf("22").longValue();
```

To convert "22.5" into a float or a double, you would write:

```
double x = Double.valueOf("22.5").doubleValue();

float y = Float.valueOf("22.5").floatValue();
```

The various **valueOf()** methods are relatively intelligent and can handle plus and minus signs, exponents, and most other common number formats. However, if you pass one of these methods, something completely non-numeric, like "hello world," it will throw a **NumberFormatException.** Exception related features will be discussed separately.

You can now write a program to accept radius in meters as user input from the command line.

```
Class area
{
    public static void main (String argsII)
    {
        double radius:
         double pi = 3.14;
         double A
          radius = Double.valueOf(args[0].doubleVaue ()
          A = pi*radius*radius:
          System.out.printin(A + "square meters")
    }

}
```

Here's the output

```
>javac are.java
>java area 2.5
24.649 square meters
```

☞ **Check Your Progress 2**

1) Where is the type casting operator used?

……………………………………………………………………………...

……………………………………………………………………………...

……………………………………………………………………………...

2) Write syntax to convert strings to numbers in Java, and explain.

……………………………………………………………………………...

……………………………………………………………………………...

……………………………………………………………………………...

3) Where is the system class defined?

……………………………………………………………………………...

……………………………………………………………………………...

……………………………………………………………………………...

2.7 PROGRAMMING CONSTRUCTS IN JAVA

In this section, we shall discuss conditional and looping constructs.

Java provides a set of standard conditional statement constructs. A conditional statement provides the program with the ability to alter its path of execution. The Java conditional statements can be built using the following:

if
else
else if
while
for
do while
switch case
break
continue
goto

The "if" statement in Java

All programming languages have some form of an "if" statement that tests conditions. In case the condition is true, the "if" statement passes a control to one set of statement otherwise it goes to another set of statement.

You can access the length of an array by using the length attribute of an array as arrayname.length. You can therefore, test the length of the args array as follows:

```
// This is a program with if statement

Class use of If statement
{
          public static void main (String args[ ])
         {
              if(args.length>0)
               {
                 System.out.printin ("Hello"+args[0])
              }

        }
}
```

System. out. printIn (args[0]) was wrapped in conditional test, if(args.length)>0) {}. The code inside the braces, **System.out.printin(args[0]**, now gets executed if, and only if, the length of the args array is greater than zero.

The argument to a conditional statement like if must be a Boolean value, that is something that evaluates to true or false. Integers are not permissible.

Testing for Equality

Testing for equality is a little trickier. You would expect to test if two numbers are equal by using the = sign. However, the = sign has already been used as an assignment operator that sets the value of a variable. Therefore, a new symbol is needed to test for equality. Java borrows C’s double equal sign, ==, for this purpose.

The else statement in Java

```
Class If Else statement
{
        public static void main (String args]})
        {
                if (args.length>0)
                {
                                System.out.println (“Hello”+args[0]};

                        }

                        else
                        {
                                System.out.println(“Hello whoever you are.”);
                        }
                }
        }
```

else if

The “if” statements are not limited to two cases. You can combine an “else” and an “if” to make an “else if” and test a whole range of mutually exclusive possibilities. For instance, here’s a version of the Hello program that handles up to four names on the command line.

```
//This is the Hello program in Java
class Hello
{
        public static void main (String.args[ ])
        {
                if (args.length ==0)
                {
                        System.out.printIn(“Hello whoever you are”);
                }

                else if (args.length ==1)

                {
                        System.out.printIn (“Hello” + args [0]);

                }

                else if (args.length ==2)
                {
                        System.out.printIn("Hello"+args[0]=" "+args[1]+"
                        "+args[2]);
                }

                else if (args.length ==4)
                {
                        System.out.printIn("Hello "+args[0]+" "+args[1]"
                        "args[2];

                }
                else if (args.length = =4)
                {
                System.out.printIn("Hello "+args[0]+" "+args[1]"
                        "args[2]+ " " + args[3];
                }
                else
                {
                        System.out.printIn("Hello "+args[0]+" "+args[1]+
                        " " args[2]+ " " +args[3]+ "and all the est! ");
                }
        }

}
```

You can see that this gets complicated very quickly. No experienced Java programmer would write code like this. There is a better solution and you will explore it in the next section.

The while loop in Java

```
//This is the Hello program in Java
class Hello
{
        public static void main (String args[ ])
        {
        int i;
        System.out.print ("Hello  ")" //say Hello
        i = 0;//Initialize loop counter
        while (I<args.length)
        {
                System.out.print(args[i]);
                System.out.print(" ");
                i=i+1; // Increment Loop Counter
        }

        System.out.printIn () " // Finish in the line
        }

}
```

The statements inside the while loop get executed repeatedly as long as the condition of the while loop (I<args.length)is true. When the condition become false, the control comes out of the while loop.

The for loop in Java

```
// this is the Hello program in Java

class Hello

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

        System.out.print (“Hello  “); //say Hello
        for (int i = 0; I,args.length; i = i + 1)
        {
                System.out.print(args[i]);
                System.out.print(“ “);
                i=i+1; // Increment Loop Centre
        }

        System.out.printIn () “ // Finish the line
        }

}
```

Multiple Initializers and Incrementers: Sometimes it is necessary to initialize several variables before beginning a for loop. Similarly, you may want to increment more than one variable. Java lets you do this by placing a comma between the different initializers and incrementers, like this:

```
for (int i = 1, j = 100; i,100; i+1, j=j-1)
{
System.out.printIn(i+j);
}
```

You cannot, however, include multiple test conditions, at least not with commas. The following line is illegal and will generate a compiler error.

```
for (int i = 1, j = 100; i,<=100, j>0; i=i-1, j=j-1) {
```

To include multiple tests, you need to use the Boolean logic operators && and 11

The Do While loop in Java

This is the Hello program in Java using the do while loop.

```
class Hello

{
        public static void main (String args[ ])
        {
        int i = -1
        do
        {
        (if i = -1)
        System.out.print (“Hello  “);
        else
        {
                System.out.print(args[i]);
                System.out.print(“ “);
        }
        i=i+1; // Increment Loop Centre
        } while (i,args.length);
        System.out.printIn () “ // Finish the line
        }
}
```

The do while loop works just like the while loop. The difference between

Break

A break statement exits a loop before an entry condition fails. In the example shown below, an error message is printed, and you break out of the for loop if j becomes negative.

```
Class breakExample
{
        public static void main (String args[])
        {
                int i, j, k
                j=1
                k=0

                for (i=1; <=64; i++)
                {
                        j=2
                        if(j<=0)
                        {
                                System.out.println("Error:Overflow");
                                break;
                        }
                        k+=j;
                        System.out.print(k+ "\t");
                        If(i%4=0)System.out.printin();
                }
                system.out.println("All done!");
        }
  }
```

Here's the output:

```
%javac break Example.java
%java break Example

2           6           14          30
62          126         254         510
1022        2046        4094        8190
16382       32766       65534       131070
262142 524286 1048574   2097150
4194302     8388606     16777214    33554430
67108862    134217726   268435454   536870910
1073741822  2147483646  Error:Overflow
All done!
%
```

The most common use of break is in switch statements:

Continue

A continue statement return to the beginning of the innermost enclosing loop without completing the rest of the statement s in the body of the loop. If you are in a for loop, the counter is incremented. For example, this code fragment skips even elements of an array:

```
for(int  i = 0; <m.length; i++){
if (m[I]%2==0) continue;
// process odd elements…
```

Labeled Loops

Normally, inside nested loops, break and continue exit the innermost enclosing loop. For example, consider the following loops:

```
for (int i = 1; j<10; i++){
for (int j = 1; j<4; j++)
{
if (j==2)
 break;
System.out.prinIn (I + ",”+j);
}
}
```

This code fragment prints

```
1, 1
2, 1
3, 1
4, 1
5, 1
6, 1
7, 1
8, 1
9, 1
```

This is because you break out of the innermost loop when j is two. However, the outermost loop continues. To break out of both loops, label the outermost loop and indicate that label in the break statement like this:

```
iloop for (int i = 1; i<3; i++){
for (int i = 1; j<4; j++) {
if (j==2) break iloop;
System.out.pringIn(I= ",”j)
}
}
```

This code fragment prints

```
1, 1
```

and then stops because j becomes equal to two, and the outermost loop is exited.

The switch statement in Java

Switch statements are shorthand for a certain kind of if statement. It is not uncommon to see a stack of if statements all related to the same quantity like this:

```
If (x==0) do something;
else if (x==1) do something else;
else if (x==2) do something else;
else if (x==3) do something else;
else if (x==4) do something else;
else do something else;
```

Java has a shorthand for these types of multiple if statements, the switch-case statement. Here is how you would write the above, using a switch-case.

```
Switch (x) {
Case 0:
```

```
do something;
 break;
case 1:
do something;
break;
case 2:
do something;
break;
case 3:
do something;
break:
default: do something else;
}
```

In this fragment, x must be a variable or expression that can be cast to an int without loss of precision. This means that the variable must be or the expression must return an int, byte, short or char. Variable x is compared with the value of each of the case statements in succession. When a match is found, the statements associated with the matched case are executed. Once the statements have been executed, the program flow executes all of the following case's statements as well, unless the optional **break** keyword is encountered, in which case, the program flow skips to the statement following the case selection construct, at which point the program's execution resumes.

Finally, if no cases are matched, the default action is triggered. If the break statement is not present in the individual cases, all values of x will trigger the default action.
It is important to remember that the switch statement does not end when one case is matched, and its action performed. The program continues to look for additional matches, unless specifically told to break.

The Ternary operator (?:) in Java

The value of a variable often depends on whether a particular Boolean expression is true or not, and on nothing else. For instance, one common operation is setting the value of a variable to the maximum of two quantities. In Java, you might write

```
If (a>b)
{
max = a;
}
else
{
max=b;
}
```

Using the conditional operator, you can rewrite the above example in a single line like this:

```
Max = (a> b) ? a: b;
```

This is an expression that returns one of two values, a or b. The condition, (a > b), is tested. If it is true, the first value, a , is returned. If it is false, the second value, b, is returned. Whichever value is returned dependes on the conditional test, a > b. The condition can be any expression which returns a Boolean value.

2.8 ARRAYS

There are three types of **reference variables:**

classes

interfaces
arrays

An array is simply a collection of similar items. If you have data that can be easily indexed, arrays are the perfect means to represent them. For instance, if you have five students in a class and you want to represent their marks, an array would work perfectly. An example of such an array is

int marks [] = {123,109,156,142,131);

The next line shows an example of accessing the marks of the third individual:

int student 3 = marks [2];

Arrays in Java are somewhat tricky. This is mostly because, unlike most other languages, there are three steps for filling out an array rather than one:

1) **Declare the array:** There are two ways to do this: Place a pair of brackets after the variable type, or place brackets after the identifier name. The following two lines produce the same result.

   Int MyIntArray[];
   Int[]MyIntArray;

2) Create space for the array and define the size. To do this, you must use the keyword new, followed by the variable type and size.

   My IntArray=new int [500];

3) Place data in the array. For arrays of native types, like those in this chapter, the array values are all set to 0 initally. The next line shows how to set the fifth element in the array;

   My IntArray[4]= 467

At this point, you may be asking yourself how we were able to create the five-element array and declare the values with the marks example. The marks example took advantage of a shortcut. For native types only, you can declare the initial values of the array be placing the values between braces ({, })on the initial declaration line.

There are several additional points about arrays you need to know:

Indexing of arrays starts with 0 (as in C and C++). In other words, the firsts element of an array is MYArray[0], not MYArray[1].

You can populate in an array on initialization. This only applies to native types, and allows you to define the values of the array elements.

Array indexes must either be type int (32-bit integer) or be able to be cast as an int. As a result, the largest possible array size is 2, 147,483,643.

Declaring arrays

Array declarations are composed of the following parts:

Array modifiers: (optional) the keywords public, protected, private

Type name: (required) the name of the type or class being arrayed.

Brackets (required) [ ]

Initialization : (optional)

Semicolon (required)

Like all other variables in Java, an array must have a specific type like byte, int, String, or double. Only variables of the appropriate type can be stored in an array. One array cannot store both ints and Strings.

Like all other variables in Java, an array must be declared. When you declare an array variable, you suffix the type with [ ] to indicate that this variable is an array. Here are some examples:

int [] k;
float []yt;
string[]names;

This says that k is an array of ints, yt is an array of floats and names is an array of strings. In other words, you declare an array just as you would declare any other variable, except that you append brackets to the end of the type.

You also have the option to append the brackets to the variables name instead of the type:

int k []'
float yt []'
String names []'

The choice is primarily one of personal preference. You can even use both at the same time like this:

int k []'
float yt []'
String names []

Creating Arrays

Declaring arrays merely says what kind of values the array will hold. It does not create them. Java arrays are objects, and like any other object, you use the new keyword to create them. When you create an array, you must tell the compiler how many components will be stored in it. Here is how you would create the variables declared on the previous page:

k=new int[3];
yt = new floats [7];
names = new String[50];

The numbers in the brackets specify the length of the array; that is, how many slots it has to hold values in. With the lengths above, k can hold three ints, and yt can hold seven floats and names can hold fifty Strings. This step is sometimes called allocating the array since it sets aside the memory that the array requires.

Initializing Array

Individual components of an array are referenced by the array name, and by an integer which represents their position in the array. The numbers that you use to identify them are called subscripts, or indexes, into the array.

Subscripts are consecutive integers beginning with 0. Thus the array k above has components k[0], k[1] and k[2]. Since you start counting at Zero there is no k[3], and trying to access it will throw an ArrayIndexOutOfBoundsException. The array elements will be initialized as:

```
k[0]=2;
k[1]=5;
k[2]=-2;
yt[17]=3.5f;
names[4] = "Fred";
```

For even medium sized arrays, it is unwieldy to specify each component individually. It is often helpful to use for loops to initialize the array. Here is a loop which fills an array with the squares of the numbers from 0 to 100:

```
Float []squares
Squares = new float [101];
For (int i = 0, i<=100; i++)
{
squares [1] = i*i;
}
```

Although i is an int, it is promoted to a float when it is stored in squares, since squares is declared to be an array of floats.

Examples of Declaring and Initializing Arrays

1) long Primes [] = new long [1000000];// Declare an array and assign
   long [] Even Primes = new long [1];// Either way, it is an array.
   Even Primes [0]=2;// populate the array.

2) // Now declare an array with an implied 'new' and populate.
   Long Fibonacci [] = {1, 1,2,3,5,8,13,21,34,55,89,144};
   Long Perfects [] = {6,28}; // Creates two element array.
   Long BlackFlyNum[];//Declare an array.

3) BlackFlyNum = new long [2147483647];// Array indexes must be type int.

4) //Declare a two dimensional array and populate it.

   Long TowerOfHanoi[][] = { {10, 9, 8, 7, 6, 5, 4, 3, 2, 1}, {}. {} };
   Long [][][] Three DticTacToe;// Uninitialized 3Darray.

☞ Check Your Progress 3

1) Define Unicode.

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

…………………………………………………………………………

2) Indicate True or False for the following statements.

(i) Every element is an array and has the same type. True ☐ False ☐

(ii) The array is fixed after it is declared. True ☐ False ☐

(iii) The index of an array always begins with 1. True ☐ False ☐

(iv) The array is fixed after it is created. True ☐ False ☐

3) What is an array index type?

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

………………………………………………………………………………………

2.9 SUMMARY

In this Unit you have learned about data types, operators, arrays and some programming constructs. These are the fundamental elements to write java programs.

Java provides four integer types: byte, short int and long. Two are floating types: float and double. Character type char represents a single character.

You also learned about how to declare and create arrays, and how to access individual elements in an array.

A Java array is an object. After an array is created its size becomes permanents. Arrays can be passed to a method as actual parameters. An array is an object, so arrays are passed using pass by reference.

2.10 SOLUTIONS / ANSWERS

Check Your Progress 1

1) (i) False
   (ii) False
   (iii) False
   (iv) False
   (v) False

2) The important Boolean operators supported in Java are:

   (i) AND (&&)
   (ii) OR (ll)
   (iii) Equals (=)

3) [ ] ~ ! << >>

Check Your Progress 2

1) The type casting operator is used to convert from one data type to another data type, for example, from string to integer according to certain rules.

2) The syntax is straightforward. It requires using the static Integer.valueOf (Strings) and int Value () methods from the java.lang.Integer class. To convert the string "22" into the int 22 you would write

   int I = Integer.valueOf ("22").intValue();

3) The System class is defined in Java.lang package which is automatically imported when Java programme starts running.

Check Your Progress 3

1) Java characters are Unicode: 16 bit encoding scheme established by the Unicode consortium to support the interchange, processing and display of the written text of the various languages of the world. Unicode takes 2 bytes, expressed in four hexadecimal numbers.

2) (i) True
   (ii) False
   (iii) False
   (iv) True

3) int data type.