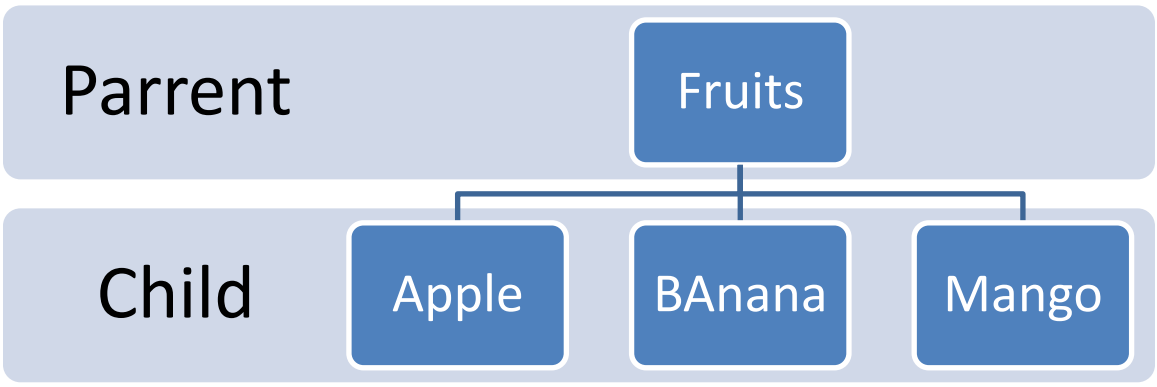
Java

Java was developed by a group of programmers at Sun Microsystems in 1991. Earlier it was named Oak. The intention behind the development of this language was to create a platform independent language that could be used in devices like microwave ovens. Now by using this characteristic of Java the language is widely used in mobiles for developing games, applications etc. The secret of the platform independent characteristic of Java lies in bytecode and JVM (Java Virtual machine).When we compile a java program it is converted to bytecode and then Java Virtual Machine executes the bytecode. Thus as a developer we don't need to make separate versions of our software for different platforms. A code in java will run perfectly in Windows, Linux, Macintosh or any other Operating system. Even a hardware upgrade does not affect the program.

OOP concept----

Java and C++ both are OOP (Object Oriented Programming) language. OOP is the crux of the two languages, especially Java. A curiosity arises in our mind, what is OOP and objects? We will now study OOP in some detail. Earlier we read that C++ was able to handle the increasing complexity in programs, this is all because Java is Object Oriented unlike C which is Structured Oriented. After we have studied OOP, students will be given a basic introduction of structured oriented approach. In OOP we take a program in terms of objects rather than methods or procedures. Characteristics of objects are determined by data and behaviour by methods or functions. OOP corresponds better to real life situations then any other paradigm.

Objects, also known as instances, are an independent unit with certain characteristics. Another important term associated with OOP is class. A class is group of objects. In C++ we can directly work on methods without creating a class, but in Java designing a class is binding on the programmer. A class acts as an encapsulating unit thus binding data and function in itself. A real life example of class can be fruits, which has instances like Mango, Apple and Banana.



In structured oriented approach the stress is on procedures, sometimes also known as methods or functions.
The three principles of OOP -
1. Encapsulation
2. Inheritance
3. Polymorphism

Encapsulation - As we studied earlier, Encapsulation is the mechanism of binding Data and functionin to class. This principle enables data abstraction i.e. hiding complexity and we can concentrate on relevant details. We can understand this concept better by taking a practical example of mobile phone which forwards hundreds of text messages everyday. Although we know the application but "how our mobile connects to network and how the network handles the traffic of thousands of such messages" is not known to most of us. Some access keywords like private and public help us to encapsulate our data or allow it to be accessed anywhere.
Public --- Data members declared public are available outside the class.
Private ---Data members declared private are available within the class.

Inheritance - The word '*Inheritance'* describes a lot about itself. We inherit traits from our parents and have our own added qualities, thus we are not clones but have an individual personality. The same principle applies in programming. After creating a parent class we can define its child class and hence reusing our code. The child class will have some added properties. Inheritance allows us to create an object oriented hierarchy.

Polymorphism - If you understand the meaning of the word, the whole concept will quickly become apparent. '*Polymorphism'* is a Greek word that means "many forms". Again taking a practical example of mobiles, we can compare polymorphism to the ringing of phone. If you get a message from Ankit the phone rings, the phone will even ring if you get a call from Nikhil. The ringing capability of phone is polymorphic. Take a look at '+' i.e. addition operator; it is used to add variables and constants. This operator can also be used to concatenate two '*strings*' which is just a group of characters like "k3b" or "splash". In the upcoming chapters we will deal with method and constructor overloading that too demonstrates polymorphism.

```
/*This the first java program hence named first.java*/
class first
{
public static void main (String args[])
{
System.out.println("This is a java program");
}
}
```

Understanding Java Programs

You have already seen the making and executing of a Java program. Understanding a program is the most important task. Have a closer look at "*first.java"*
Line 1::
/* Multiline Comment*/
The comments on program are enclosed within it. The compiler ignores these lines and hence no error is generated. Writing comments is not binding on the user.
Line 2::

class first This is the second line of our java program. Here we have created a class named "*first".* Creating a class is binding on the programmer as Java is strictly object oriented. Everything is encapsulated within a class.

Line 3::

{

Opening braces Braces play a vital role in programming. A group of statements are enclosed in braces called blocks. Brace, here depicts that the class has begin.

Line 4::

public static void main(String args[])

In this line we have created a method or function using a list of java "***keywords****".* A **keyword** is a special word that is recognized by the compiler and has a special use.

Keywords

1. public -- An access specifier, it declares that this method can be accessed by members who are even outside this class.

2. static -- We have created a static method named "*main".* In the context of methods "static" keyword means that this method is independent of any object. As main is executed before any other method, we need to make it static so that we can call other methods by creating their instances here.

Special Note—You will notice that when the same program was written in BlueJ, we did not mention the keyword "*static".* So at the time of executing we had to create an object. When we are not working with BlueJ we have to manually create an object. The format is --Class name objectname=new classname(); But as here we have a static method we don't need to create any h d

3. void -- You will learn in the upcoming chapters that a method can even return a value. The *"void"* keyword here means that the method does not return any value. If these things seem too complicated to you, have patience!! Soon each and every detail will be clear.

4. main (String args[]) -- We have now specified the main method. The brackets enclose within themselves 'the parameters'. Here we have an array of instances of String class. (Arrays are a group of one kind of data and String is a predefined class in Java that stores series of characters in itself).

Line 5::

{

Opening Brace

By this brace we come to know that our method has begun. It defines the starting point of method.

Line 6::

System.out.println("This is a java program");

This is an output statement. Well, there is good reason for this line working like magic and sincerely you don't need to mug it up. In Java we take input and output in forms of "*Streams".* Imagine it as a flow of data. System.out.println(); directs all the contents within the brackets as *"Output Streams"* to the console. Here System is a predefined class, out an object and println() a method. Once you have studied in detail about classes and objects, you will understand this scary business much better.

// Printing a statement

This is another way of writing comments. The difference being that this is a single line comment. It has a scope till the end of line. For commenting on the next line, we have to again insert //.

Remember — Writing comments are not binding on the programmer but it is a good practice to insert comments in your code. It makes the code clean, readable and understandable.

Line 7:
}
Closing Braces
These braces tell us that the method has ended. Now all other statements will not be considered as a part of that method.
Line 8::
}
These braces illustrate the ending of class.
Another way of writing the same program
class first { public static void main(String args[]) { System.out.println("This is a java program"); }}
You can even write the above mentioned program in this way. It is not recommended as it leads to a complex program that is difficult to understand and maintain.
AMAZING FACT— *Many softwares have even thousand thousands of lines of source code.*

Exercise

Q1. Discuss the evolution of Java.
Q2. How is the platform independent feature of Java implemented?
Q3. Explain Inheritance, Encapsulation and Polymorphism.
Q4. Write a short note on jdk and its tools.
Q5. What is an IDE?
Q6. Can we use Ms Word for writing java programs?
Q7. Write a program that prints the following two lines------
"Java is a powerful language"
"Executing a Java program"
Q8. Try writing and executing the *Q.7* in BlueJ.
Q9. What are Java "*keywords"*? Explain some of them.
Q10. Why are comments important in our program?

Data Types And Operators

In our daily life we receive information as stimulus from our surrounding. Then we process and analyze it within our minds. A computer too gets input from several sources which it processes to give an output. So data handling is a core issue of programming.

CONSTANTS

The word "*constant"* means anything that does not change. Constants that are also known as "literals" have a fixed value in programming. Let us assume a number '6', this digit represents a number. If I order two shopkeepers to give me six DVDs of a movie, both will supply me with the same number. The real world situation even applies here. In this way we can even call alphabets 'S''t' 'u' 'V' and Strings such as "Delhi" as constants.
A proper hierarchy will be ---
Constants ---
1. Numeric Constants----- (1). Integer Constants
(2). Real Constants
2. Character Constants---- (1).Characters
(2).String
If you are confused by the terms "Real" and "Integer" constants then here is the answer…
An integer constant contains no fractional part while Real Constants contains a decimal value.

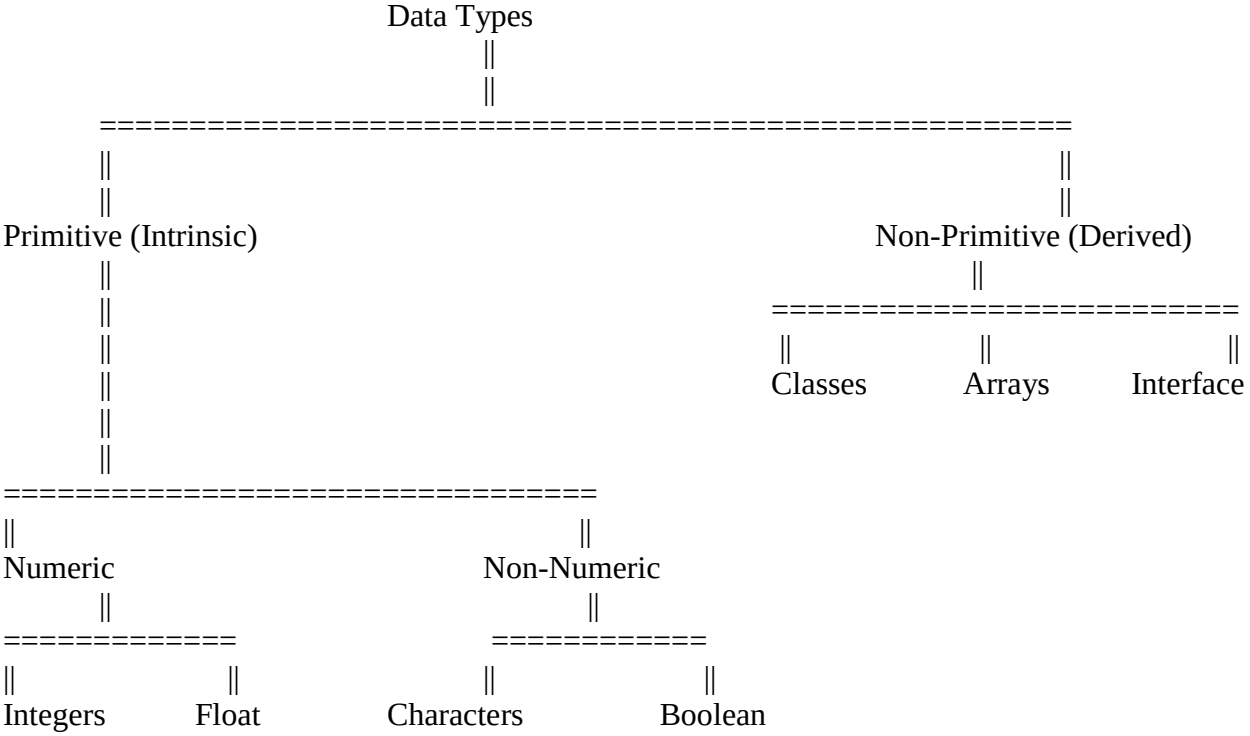
VARIABLES

A variable as the names suggest is an entity whose values changes from time to time. Imagine that you are solving an algebraic equation where you are calculating values of '*x'* and '*y'* (two variables). After solving the question, you might get the values 6, 17. Similarly in the next question the values of '*x'* and '*y'* may turn out to be 1, 14. This concept is also applicable in programming.
When we create a variable, a part of temporary memory is assigned to it. The value is stored till the execution of program and later the memory is freed. Just like constants a variable can also be of several types.

Data Types

Each and Every variable and constant holds a certain data. The data can be classified under any of these types. While declaring variables we have to specify its type.
Remember: Java is said to be strictly typed language. This is because of the fact that you can not easily assign a variable with a value of another data type

Integer Type
Integer (int) Holds digits without any fractional part.
Short (short) Holds digits without any fractional part. The range of value it holds is less if compared with "*int"*.
Long (long) Holds digits without any fractional part. The range of value it holds is more if compared with "*int"*.
Byte (byte) Holds digits without any fractional part. The range of value it holds is least.
Float Type
Float (float) Holds digits with fractional part.
Double (double) Holds digits with fractional part. The range of value it holds is higher in comparison with float.
Character Type
Char (char) Holds a single character.
Classification of Data Types



Reserved Words
There are certain predefined keywords in Java that help us in programming. Thus we cannot use them as variables name.
Special Printing Characters
Sometimes you want to format your output in a certain way. In such conditions you use these characters.
Study this Example which demonstrates the use of one of the special character, it also shows the use of variables and data types.
class demo

```
{
public static void main(String args[])
{
int no1,no2;                                    //Two variables of data type int
char name='N';                                  // Initialization at the tome of variable declaration
                                                //Post Initialization
no1=17;
no2=12;
System.out.println ("Character is "+name+"\n");       // Here "\n" is the special printing
//character
System.out.println("Numbers are "+no1+" "+no2);
}
}
```

Popular Printing Characters

Newline \n *Use--- Prints a blank line.*
Horizontal Tab \t *Use---Prints a horizontal space of 5*
Vertical Tab \v *Use---Prints a vertical space of 5*

OPERATORS

We widely use several operators in our daily language while working with values in Mathematics and even our day-today life.

Definition: Operators are certain symbols recognized by the compiler which when used with variables or constants (popularly also known as operands) do a pre-defined action.
Types of Operators are---
1. Arithmetical Operators
2. Increment Operators
3. Bitwise Operators
4. Logical Operators
5. Relational Operators
6. Ternary or Conditional Operator

Arithmetic Operators

The four operators '+', '-', '*' and '/' are known as arithmetic operators. They are used to perform simple arithmetic calculation in programming as in mathematics. Certain operators require only one operand (Example- '+, - ') and are hence known as Unary operator. Other operators that are used with two operands are known as Binary Operator ('*, / , +, -')
Here is an example

```
class arith_opr
{
public static void main(String args[])
{
float no1,no2;
float sum,product,difference;
float quotient;
no1=17;
no2=02;
```

```
sum=no1+no2;
product=no1*no2;
quotient=no1/no2;
difference=no1-no2;
System.out.println("Two Numbers are "+no1+" "+no2);
System.out.println("Sum is " +sum);
System.out.println("Product is "+product);
System.out.println("Quotient is "+quotient);
System.out.println("Difference is "+difference);
}
}
```

Increment and Decrement Operator

Like any other language such as C++ or C, Java too provides increment and decrement operator. The purpose of this operator is quite simple i.e. is to increment and decrement the value of operand by one.

Increment and Decrement operators can be in two forms.

1. Prefix
2. Postfix

Example ----

```
class inc_dcr
{
public static void main(String args[])
{
int no=3;
int no2=5;
no++; //Incrementing value by 1
++no; //Incrementing value by 1
no2--;
--no2;
System.out.println("Number 1-" +no);
System.out.println("Number 2-" +no2);
}
}
```

The difference between prefix and postfix increment and decrement operator is seen only when we use them in an expression. Have a look at this program.

```
class pre_post
{
public static void main(String args[])
{
int a,n,s;
a=5;
n=10;
s=++a*n;
System.out.println("After first calculation s= "+s);
a=5;
```

```
n=10;
s=0;
s=a++*n;
System.out.println("After final calculation s= "+s);
}
}
```

Prefix Increment or Decrement Operator – The prefix Increment or Decrement operator works on the principle "*First change then use."* The value of operand is first increased or decreased and then the value thus obtained is used in expression for calculation.

Postfix Increment or Decrement Operator – The postfix Increment or Decrement operator works on the principle "*First use then change".* The value of operand is directly used in the expression and the value is changed later.

Some examples of this Operator ---

```
class ex_inc_dcr
{
public static void main(String args[])
{
int a,b,c,d;
a=5;
b=6;
c=11;
d=++a*b---c;
System.out.println(+d);
a=5;
b=6;
c=11;
d=0;
d=a++*b-c--;
System.out.println(+d);
}
}
```

As you are now familiar with the theory behind the difference in increment and decrement operator, Let us see the working –

(1).d=++a*b---c;
a=5;
b=6;
c=11;
++a = 6;
b-- = 5(value 6 will be used in expression)
c = 11;
After final calculation--
6*6-11= 36-11=25

(2). a=5;
b=6;

c=11;
d=0;
d=a++*b-c--;
a++ =6(Value used in expression will be 5)
b =6
c-- =10(value used in expression will be 11)
5*6-11
30-11=19

Bitwise Operator
The Bitwise operator when used with any operand works at its individual bits. It does not work with float and double data type.
Logical Operator
Logical operators are generally used with conditional statements and are used to combine two conditions.
The three Logical Operators---
|| Logical Or
! Logical Not
&& Logical And
Example ---- (You will see their use in the upcoming chapters)
if(a= =1 && b==2)
statements…
If both the conditions are true then only && block is successfully executed
Even if one condition out of the many is true then also the block is successfully executed.
If the condition is wrong then only the statements inside the block are executed.
Relational Operators
As the name clear illustrates the meaning, the relational operators show the relations between the two variables.
Relational Operators are –
Greater Than >
Less Than <
Greater Than Equal To >=
Less Than Equal To <=
Equal To = =
Not Equal To !=
Ternary Operator
Ternary Operator –
Syntax---- **?:**
Conditional Operator is also known as Ternary Operator. It can be used to a substitute of '*if* ' statement. Observe this program carefully.

```
class cond_opr
{
public static void main(String args[])
{
int n1,n2,ans,ans2;
n1=5;
```

```
n2=10;
ans=(n1<n2)?n1:n2;
ans2=(n1>n2)?n1:n2;
System.out.println(+ans);
System.out.println(+ans2);
}
}
```

Taking Input as Command Line Argument

All the programs we have created till now have variables with there values initialized. (E.g. int x=4;) Now we will take input from the user using command line arguments. This is a simple calculator.

```
class sim_calc
{
public static void main(String args[])
{
int n1,n2,add,sub,mul,div;
n1=Integer.parseInt(args[0]);
n2=Integer.parseInt(args[1]);
add=n1+n2;
sub=n1-n2;
mul=n1*n2;
div=n1/n2;
System.out.println("Sum is "+add);
System.out.println("Diff is "+sub);
System.out.println("Product is "+mul);
System.out.println("Quotient is "+div);
}
```

Exercise

Q1. What is a literal?
Q2. Explain the hierarchy of Constants.
Q3. Why is Java called a Strictly "typed language*"* ?
Q4. Explain the term variable?
Q5. How can reserve words cause problem?
Q6. What is the main use of all special printing characters?
Q7. Explain Operators with their proper classification.
Q8. Explain the difference between prefix and postfix increment and decrement operator.
Q9. How is ternary operator useful?
Q10.Design a program that takes an input from the user

Understanding Classes and Functions

Introduction

In the first chapter, you studied about OOP, classes, functions and their use in Java language. As it was just a start you were introduced with certain basic concepts. The most powerful features were left untouched, e.g. - You had only seen the use of main method but are yet to discover the concept of using multiple methods in a class other then the "*main"*. Now you will explore them in depth. Before going to deep let us revise the things and start from basics.

A general class declaration with methods---

```
class name1
{
//public variable declaration
void methodname()
{
//body of method…
//Anything
}
}
```

Let us apply this concept and take an example of real world class.

```
class learn
{
private int x,y,z;
public void input()
{
x=10;
y=15;
}
public void sum()
{
z=x+y;
}
public void print_it()
{
System.out.println("Answer is =" +z);
}
}
```

This is a class 'learn' with three methods, i.e., input, sum and print. The three functions are designed to do a specific task. This you can easily figure out by seeing the code. The input () method initializes the two values, sum () adds them and finally print_it() shows the answer.

Will the class compile and execute?

Try it on your own, you will find that the class compiles in the DOS environment but fails to execute. If you will try it in the BlueJ environment, it will compile as well as execute successfully (This clearly shows the advantage of using BlueJ).

The class did not execute because of the absence of the "*main*" method. You already

know that the main method is the first one to execute. As soon as we run a java program the control is transferred to "*main*". So we will now add a main method to the "learn.java" program.

```
class learn
{
private int x,y,z;
public void input()
{
x=10;
y=15;
}
public void sum()
{
z=x+y;
}
public void print_it()
{
System.out.println("Answer is =" +z);
}
public static void main(String args[])
{
learn object=new learn();
object.input();
object.sum();
object.print_it();
}
}
```

Try to compile the program and voila!!! it runs successfully. Now let us see what to these additional lines mean. We have added the main method to our program. Look at the following code :

```
learn object=new learn();
object.input();
object.sum();
object.print_it();
```

In the first line we created an object (*A much talked subject in the first chapter!!)*. An object can be simply created by typing---

```
learn object;
```

The object thus created will be null and so we need to instantiate it. The new operator allocates memory for the object and instantiate it . Constructors are also called at this point (You will read about constructors in the next chapter).

The three methods are called by using the dot operator. When we call a method the code inside its block is executed.

The dot operator

The dot operator is used to call methods or access them. It has other use too, which we will discuss in this chapter later.

Creating "main" in a separate class

You can even create the main method in a separate class, but during compilation you need to make sure that you compile the class with the "main" method. The program will thus be like this….

```
class learn
{
private int x,y,z;
public void input() {
x=10;
y=15;
}
public void sum()
{
z=x+y;
}
public void print_it()
{
System.out.println("Answer is =" +z);
}
}
class apply
{
public static void main(String args[])
{
learn object=new learn();
object.input();
object.sum();
object.print_it();
}
}
```

Another use of dot operator

Read the next program carefully and try to figure out its working.

```
class dot1
{
int x,y,z;
public void sum(){
z=x+y;
}
public void show(){
System.out.print("The Answer is "+z);
}
}
class apply1
{
public static void main(String args[]){
dot1 object=new dot1();
```

```
dot1 object2=new dot1();
object.x=10;
object y=15;
object2.x=5;
object2.y=10;
object.sum();
object.show();
object2.sum();
object2.show();
}}
```

Instance Variable

All variables (Assume int num) are also known as instance variable. This is because of the fact that each instance or object has its own copy of values for the variable, as you can easily see in the program mentioned above.

Hence other use of the "*dot"* operator is to initialize the value of variable for that instance.

Static keyword --- Earlier you read about the use of "*static*" keyword when it was used before the method's name. This keyword can even be used with variables.

e.g. — static int num;

This will make the variable independent of any instance and it will have the same copy of value for each and every object.

Methods with parameters

Take an example of a method named 'sum()'. You can see 'sum()' with parenthesis. As they are empty, it shows that the method has no parameters. You can even add some parameters of your own to serve as an add-on to your method's functionality. Observe the following example —

```
class learn2
{
int n,n2,sum;
public void take(int x,int y)
{
n=x;
n2=y;
}
public void sum()
{
sum=n+n2;
}
public void print()
{
System.out.println("The Sum is"+sum);
}
}
```

```
class my_main
{
public static void main(String args[])
{
learn2 obj=new learn2();
obj.take(10,15);
obj.sum();
obj.print();
}
}
```

In this "*learn2*" program, the method "*take(int x,int y)*" has two parameters. These parameters are supplied with values from the main, when the method is called. The variables x, y are declared as parameters and you have transferred their values to two other variables n & n2 respectively.

Remember: Values supplied should match with the data type of parameters.

Methods with a Return Type

You might be still wondering about the use of void and the concept of returning a value. So here is a jargon buster you desperately need.

So far you have seen that methods can take parameters. Therefore, you can even expect them to return a value. Instead of using void, you can write something like 'int, char' thus specifying the type of value that you want your method to return.

Example—

```
public int process()
{
// body of function
// Other statements
return value;
}
```

Go through this example---

```
class learn3
{
int n,n2;
public void take( int x,int y)
{
n=x;
n=y;
}
public int process()
{
return (n+n2);
}
}
class apply3
{
public static void main(String args[])
{
```

```
int sum;
learn3 obj=new learn3();
obj.take(15,25);
sum=obj.process();
System.out.println("The sum is"+sum);
}
}
```

The method "*process"* returns the sum of the two variables. In the main method we have taken the sum in a variable "*sum"*. The method holds the value 40 like a variable and hence we don't have to use several variables. Declaring "*sum*" is optional, we could even print the answer directly.

```
System.out.println("The Sum is "+obj.process());
```

Method Overloading

You were earlier introduced to the concept of "*method overloading*" in the first chapter under the caption "polymorphism" (Turn back to refresh the topic once again, if you can't recall it!!). After that take a look at the example below—

```
class meth_over
{
int x=5,y=5,z=0;
public void sum()
{
z=x+y;
System.out.println("Sum is "+z);
}
public void sum(int a,int b)
{
x=a;
y=b;
z=x+y;
System.out.println("Sum is "+z);
}
public int sum(int a)
{
x=a;
z=x+y;
return z;
}
}
class apply4
{
public static void main(String args[])
{
meth_over obj=new meth_over();
obj.sum();
obj.sum(10,12);
System.out.println(+obj.sum(15));
```

```
}
}
```

REMEMBER :: The position of method in the program does not matter.

Passing Objects as Parameters

Objects can even be passed as parameters. Have a look at this example

```
class param
{
int n,n2,sum,mul;
public void take(int x,int y)
{
n=x;
n2=y;
}
public void sum()
{
sum=n+n2;
System.out.println("The Sum is"+sum);
}
public void take2(param obj)
{
n=obj.n;
n2=obj.n2;
}
public void multi()
{
mul=n*n2;
System.out.println("Product is"+mul);
}
}
class apply5
{
public static void main(String args[])
{
param ob=new param();
ob.take(3,7);
ob.sum();
ob.take2(ob);
ob.multi();
}
}
```

Passing Values to methods and Constructors

Now you are familiar with passing values to methods. We have passed integers(data types) as well as objects. These are two different ways of supplying values to methods. Classified under these two titles -

1. Pass by Value
2. Pass by Address or Reference

Pass by Value-When we pass a data type (like a variable "*num*" of int, float or any other type of data) to a method or some constant values like(15,10). They are all passed by value. A copy of variable's value is passed to the receiving method and hence any changes made to the values do not affect the actual variables.
Consider this example---

```
class pass_by_val
{
int n,n2;
public void get(int x,int y)
{
x=x*x; //Changing the values of passed arguments
y=y*y; //Changing the values of passed arguments
}
}
class apply6
{
public static void main(String args[])
{
int a,b;
a=1;
b=2;
System.out.println("Initial Values of a & b "+a+" "+b);
pass_by_val obj=new pass_by_val();
obj.get(a,b);
System.out.println("Final Values "+a+" "+b);
}
```

Pass by Reference
Objects are always passed by reference. When we pass a value by reference, the reference or the memory address of the variables is passed. Thus any changes made to the argument causes a change in the values which we pass.
Demonstrating Pass by Reference---

```
class pass_by_ref
{
int n,n2;
public void get(int a,int b)
{
n=a;
n2=b;
}
public void doubleit(pass_by_ref temp)
{
temp.n=temp.n*2;
temp.n2=temp.n2*2;
}
}
class apply7
```

```
{
public static void main(String args[])
{
int x=5,y=10;
pass_by_ref obj=new pass_by_ref();
obj.get(x,y); //Pass by Value
System.out.println("Initial Values are-- ");
System.out.println(+obj.n);
System.out.println(+obj.n2);
obj.doubleit(obj); //Pass by Reference
System.out.println("Final Values are");
System.out.println(+obj.n);
System.out.println(+obj.n2);
}
}
```

Pure & Impure Functions

Pure Functions -- Pure functions do not modify their arguments or output. The result of a pure function call is the return value.

Impure Functions—Just opposite to pure function.

Recursion

When a method calls itself, it is known as recursion. This phenomenon may look weird but it becomes really helpful in some conditions. The power of recursion can be demonstrated by taking an example of a class that calculates factorial of a number. If you are unfamiliar with the term "*factorial*", here is the description with an example.

No=5

Factorial=5*4*3*2*1

You can deduce from this example, the factorial of a number 'N' is the product of all natural numbers descending from 'N' till 1.

```
class fact
{
int ans;
public int take(int no)
{
if(no==1)
return 1;
ans=take(no-1)*no;
return ans;
}
}
class apply8
{
public static void main(String args[])
{
fact obj=new fact();
System.out.println(+obj.take(5));
}
}
```

The method calls itself repeatedly and calculates the factorial. If you want to see the stepwise calculation of the factorial, insert a print statement in the method. You will then understand its working better.

Use of "*this*" keyword

The "*this*" keyword refers to the object by which a method was called. Have a look at this piece of class.

```
// This is a wrong code…
class no_this
{
int a,b;
void take(int a,int b)
{
a=a;
b=b;
}
```

The mistake in this program can be easily pointed out. We cannot use the same name for the instance variables and the variables which are declared as parameters. The reason being the instance variables are hidden when we use parameters with the same name. If we are determined to use the same name, then the "*this*" keyword comes to rescue. Now look at a class named "*with_this*"

```
class with_this
{
int n,n2;
int sm;
public void get(int n,int n2)
{
this.n=n;
this.n2=n2;
}
public void sum()
{
sm=n+n2;
System.out.println("Sum is "+sm);
}
}
class apply9
{
public static void main(String args[])
{
with_this obj=new with_this();
obj.get(10,12);
obj.sum();
}
}
```

We can use the "*this*" keyword even if we have used different names for the parameters and the instance variable.

Exercise –

Q1. What is the importance of class in Java?
Q2. What is the role of "main" method in the execution of programs?
Q3. Explain the two uses of dot operator.
Q4. Why is the main method preceded by the keyword static?
Q5. A method can be written even without using a return type. What is the benefit if we use them?
Q6. Explain method overloading and demonstrate it with the help of an example.
Q7. State the difference between "pass by value" and "pass by reference".
Q8. State the use of 'this' keyword.
Q9. Explain recursion with the help of an example.
Q10.Write about pure and impure function.

Constructors

You have already studied in depth about methods, classes and Java. Now we will study about "*Constructors*", which are an important part of java programs.

Definition-A constructor of a class is a special function which is automatically called when the object of the class is called. A constructor makes our work easier by initializing objects at their creation. The most important question is how? We will now study it.

Special Features of Constructors—

These are some really important characteristics of Constructors. Read them very carefully.

1. Constructors have the same name as that of class.
2. They don't have any return type (void, int).
3. Constructors cannot be inherited. "*Super*" keyword is used to call the parent class constructor.

Declaration of Constructors---

```
class any
{
public any() //Constructor
{
/* Body of Constructor
Body continues ….
*/
}
}
```

Lets study a simple example of a class "Calc".

```
class calc
{
int n,n2,sum;
public calc() //Class Constructors
{
n=15;
n2=20;
}
public void sum()
{
sum=n+n2;
System.out.println("Sum is "+sum);
}
}
class apply10
{
public static void main(String args[])
{
calc obj=new calc(); //Constructor called....
obj.sum();
}
}
```

Have a look at the constructor's body, you can see two statements.

```
{
n=15;
n2=20;
}
```

Here when the constructor was called the two variables were automatically initialized. Thus we mentioned earlier that "*A constructor makes our work easier by initializing objects at their creation.*"

Two Types of Constructors---

1.Default Constructor.

2.Parameterized Constructor

Default Constructor---A "*default constructor*" is a constructor with no parameters.(You are already familiar with the concepts of parameters. The class constructed in the previous page contains a default constructor "*calc*".

Parameterized Constructor---A "*parameterized constructor*" is a constructor with parameters. The way of declaring and supplying parameters is same as that with methods. Study this example—

```
class cons_param
{
int n,n2;
int sum,sub,mul,div;
public cons_param(int x,int y)
{
n=x;
n2=y;
}
public void opr()
{
sum=n+n2;
sub=n-n2;
mul=n*n2;
div=n/n2;
System.out.printn("Sum is "+sum);
System.out.printn("Difference is "+sub);
System.out.printn("Product is "+mul);
System.out.printn("Quotient is "+div);
}
}
class apply11
{
public static void main(String args[])
{
cons_param obj=new cons_param(6,17);
obj.opr();
}
}
```

Constructor Overloading---

The basic concept underneath "*Constructor Overloading*" is same as that of method overloading. The difference lies on the concept that here we are working with constructors and not with methods.

Lets see our next example of the apply series.

```
class overload
{
int n=0,n2=0;
public overload()
{
n=1;
n2=2;
System.out.println("n & n2 "+n+" "+n2);
}
public overload(int x)
{
n=x;
System.out.println("n & n2 "+n+" "+n2);
}
public overload(int x,int y)
{
n=x;
n2=y;
System.out.println("n & n2 "+n+" "+n2);
}
}
class apply12
{
public static void main(String args[])
{
overload obj=new overload();
overload obj1=new overload(5);
overload obj2=new overload(10,15);
}
}
```

Exercises

Q1.What is a constructor?

Q2.How does a constructor differ from a method?

Q3.Explain the two types of constructors that can be created by Java.

Q4.What is the general use of constructor?

Q5.Differentiate between constructor and method overloading

\

Loops and Conditional Statements

Look at any program you have created till now. You can see that the execution begins with the main method. Other functions are called sequentially and after the execution of each and every statement, the program ultimately terminates. The "*Real World*" programs are entirely different. They respond to users actions. Explaining it in a better and technical way, they execute in a conditional manner.

Take an example our world, we interact with our environment and our behavior is determined by the conditions around us. You laugh at a clown's joke, but don't even dare to smile in front of your principal! This is what a conditional response. We can also design programs in such a way so that they take a step according to the conditions (Input).

Imagine that you have created a tax calculating program. According to the different pay scales, taxes which are to be paid differ. A man earning a meager Rs.4000 a month has a total annual pay which amounts negligible in comparison to the tax paid by a person earning Rs.300000 a month!! Thus your program should be capable enough to compute the tax according to the condition i.e. the basic monthly pay. Like many other languages Java too handles condition using if-else and switch statements.

Conditional Statements---

1. if-else statement
2. switch

If Statement

If statement's general format….

```
If (condition)
{
//Statements…...
}
```

Let's study this example---

```
class cond
{
pubic static void main(String args[])
{
int inp=10;
if(inp==06)
{
System.out.println("Hello Special Number");
}
System.out.println("Nothing Special");
}
}
```

If-Else Statements

Read this statement carefully "If the weather is bad I will not visit you *else* I will surely come". After reading this statement you can easily understand the context in which "*else*" is used.

```
class cond2
{
public static void main(String args[])
{
int x;
x=Integer.parseInt(args[0]);
if(x==1)
System.out.println("You are the no. one");
else
Sysem.out.println("You are not the no. one");
}
}
```

This program is unable to demonstrate the power of "*else*". The reason lies on the fact that this program we have not used else with if. The next example will do this.

If-else If

```
class cond3
{
public static void main(String args[])
{
int inp;
inp=Integer.parseInt(args[0]);
if(inp==1)
System.out.println("You the No. 1");
else
if(inp==2)
System.out.println("No.------ 2");
else
if(inp==3)
System.out.println("No.-------3");
else
System.out.println("You are not among the top three");
}
}
```

Working with "&' Operator

Imagine a garments store where you get discount and points for your patronage on your shopping. The more you purchase the more " discount and points" you are awarded. Look at a program developed to assist the Shopkeeper

```
class cond4
{
int money;
int dis;
int points;
```

```
public cond4(int temp)
{
money=temp;
}
public void calc()
{
if(money>0 && money<=1000)
{
dis=3*money/100;
points=money/10;
System.out.println("Discount Offered is "+dis);
System.out.println("Points Earned are "+points);
}
else
if(money>1000 && money<=3000)
{
dis=8*money/100;
points=money/10;
System.out.println("Discount Offered is "+dis);
System.out.println("Points Earned are "+points);
}
else
if(money>3000 && money<=5000)
{
dis=15*money/100;
points=money/10;
System.out.println("Discount Offered is "+dis);
System.out.println("Points Earned are "+points);
}
}
}
class store
{
public static void main(String args[])
{
int val;
val=Integer.parseInt(args[0]);
cond4 obj=new cond4(val);
obj.calc();
}
}
```

Menu Based Programs

You can use if-else to design menu based program. A menu based program has an ability to perform a variety of tasks but it asks the users input and carries out a particular job.
Example---Calculator

Nested If statements

A nested if statement contains another if statement inside its block. Look at this format.

```
if (condition)
{
if(condition2)
{
/* Statements…….
Ending */
}
// Some extra statements
}
```

Nested ifs can sometimes be used as a substitute to the "&&" operator. Example

```
if (x==1 && x== 2)
System.out.println ("Condition satisfied");
```

OR

```
if (x==)
{
if(x==2)
System.out.println("Condition satisfied");
```

An example demonstrating use of nested ifs.

```
/* This program gives you a compliment according to your marks and behavior*/
class nested
{
public static void main(String args[])
{
int marks=64;
int behm=10;
if(marks>=80 && marks<=100)
{
if(behm>=8 && behm<=10)
System.out.println("You are a Good Boy/Girl...KEEP IT UP!!!");
Else
System.out.println("You are Really good in studies but can work on you behavior");
else
System.out.println("Yu are good in studies...Need to work a lot on your behavior");
}
else
if(marks>=50 && marks<80)
{
if(behm>=8 && behm<=10)
System.out.println("Average. You are a Good Boy/Girl...KEEP IT UP!!!");
else
if(behm>=5 && behm<8)
System.out.println("You are average in studies but can work on you behavior");
else
System.out.println("Average in Studies...Need to work a lot on your behavior");
}
```

```
else
System.out.println("Improve in your studies..No behavior marks ");
}
}.
```

Switch Keyword

Just like the "*if-else"* statement, we ca even use "*switch*" for conditional execution of our program.

REMEMEBR:: We don't use switch very often as it is not as versatile as "if". The reason for the limitation of "*switch*" is *its inability to work with a range of values*.

General syntax---

```
switch (data type to check for a condition)
{
case cond1
{
Action;
break;
}
case cond2
{
Action;
break;
}
case n;
{
Action;
break;
}
default
{
Action;
}
```

A simple program that if a character is a vowel or consonant.

```
class swtch
{
public static void main(String args[])
{
int inp='i';
switch (inp)
{
case 'a':
System.out.println("A vowel");
break;
case 'e':
System.out.println("A vowel");
break;
case 'i':
```

```
System.out.println("A vowel");
break;
case 'o':
System.out.println("A vowel");
break;
case 'u':
System.out.println("A vowel");
break;
default:
System.out.println("A Consonant");
}
}
}
```

It is easy to judge the meaning of break. The "*break*" keyword is used to get out of a switch condition. The "*break*" keyword is widely used in other conditions too. We will discuss about it later in this chapter.

Loops ---

Sometimes we need to repeat an action again and again. Imagine writing a program that prints the line "HAPPY BIRTHDAY" thousand times. Until you use loops, writing such a program is fairly impossible. Java provides us with three kinds of Loops—

1. while
2. for
3. do-while

while

You will often use while loop to repeat statements in a Java program. After going through each and every type of loop you will find that it is best to use this loop when—

1.We are not sure about the number of times we have to repeat a statement.
2. We are not sure if our loop will even execute a single time

Syntax for while loop

```
while(condition)
{
// Statements;
every thing we ever wanted to execute
}
```

Example

```
/*A program to print a message 1000 times */
class whloop
{
public static void main(String args[])
{
int c=1;
while(c<=1000)
{
System.out.println("HAPPY B'DAY");
c++;
}
```

```
}
}
```

do-while

The do-while loop can also be used to repeat a set of statements.

Syntax---

```
do
{
Anything you wish!!!
}
while(condition);
```

If you will carefully examine the structure of do-while loop, you will find that the condition was tested after the execution was done once. Thus we come to a conclusion that the do-while executes the statement(s) at least once.

```
/* A program to print a message 1000 times. */
class dowhloop
{
public static void main(String args[])
{
int c=1;
do
{
System.out.println("HAPPY B'DAY");
c++;
}
while(c<=1000);
}
}
```

for loop

The "*for*" loop is most widely used for repeating a set of statements. The credit goes to the clear cut interface defined by it

Syntax—

```
for(initial value of counter; condition; update counter)
{
// Statements
}
```

For using for loop you should be clear about the fact-" How many times exactly will my program actually execute?"

```
/* A program to print a message 1000 times */
class forloop
{
public static void main(String args[])
{
int c;
for(c=1;c<=1000;c++)
System.out.println("HAPPY B'DAY");
}
```

```
}
```

Factorial of a number—

You have already studied about the concept of factorial, when we studied about recursion. So the definition of factorial is clear, now we just need to write a program using loops. Developing logic is the most important part which you will be able to do it yourself after some practice.

```
Factorialclass
fact
{
public static void main(String args[])
{
int no,c;
no=Integer.parseInt(args[0]);
for(c=no;c>1;c--)
{
no=no*(c-1);
}
System.out.println(" Factorial is "+no);
}
}
```

Writing the program using while loop.

```
class fact
{
public static void main(String args[])
{
int no,c;
no=Integer.parseInt(args[0]);
c=no;
while(c>1)
{
no=no*(c-1);
c--;
}
}
}
```

Writing the program using do-while loop

```
class fact
{
public static void main(String args[])
{
int no,c;
no=Integer.parseInt(args[0]);
c=no;
do
{
no=no*(c-1);
```

```
c--;
}
while(c<1);
}
}
```

Two keywords..

1. break
2. continue

(1).The “break” keyword is used to get out of a loop. Usually we break with an “*if*” statement, so the loop terminates only at a certain condition.

Read this program for a clearer understanding, this program uses the “io” package and gets input from user. I have used the InputStreamReader and BufferedReader classes. You are still unfamiliar with these concepts. So rather on concentrating on them and getting carried away better have an eye on the loop and break. The program is commented extensively to assist you.

```
import java.io.*; //Importing io package to help you get the input
class limit_no
{
public static void main(String args[]) throws IOException
{
int no; // Variable that will repeatedly take the input from user
int c=5;
InputStreamReader rd=new InputStreamReader(System.in); //Lines help us to take an
//input
BufferedReader inp=new BufferedReader(rd);// Lines help us to take an input
while(c==5) // Loop should not terminate until c=5
{
System.out.println("Enter your marks"); // Message that asks users for an input
String v1=inp.readLine();
no=Integer.parseInt(v1); //Variable "no" is finally initialized with the value entered by
//user
if(no<0 || no>100) //Loop will still terminate if this condition gets false.
break;
}
System.out.println("While Loop Terminated");
}
}
```

(2).continue: The “continue” keyword is used to force an execution of loop. Let's study an example similar to the previous one.

```
import java.io.*; //Importing io package to help you get the input
class limit_no2 // Variable that will repeatedly take the input from user
{
public static void main(String args[]) throws IOException
{
int no;
int c=1;
```

```
int x=1;
InputStreamReader rd=new InputStreamReader(System.in); //Lines to help us to take an
input
BufferedReader inp=new BufferedReader(rd);
while(x<=3) //while loop will cause the block of code to execute thrice
{
System.out.println("Enter a Number"); //Message to be printed
String v1=inp.readLine();
no=Integer.parseInt(v1);
x++; //Update value of x,thus regulating the times loop executes
if(no==5) // Force the execution of Loop
continue;
System.out.println("This message will not be printed in an exceptional case"); // This
//message will not be executed if number entered is equal to 5
}
}
}
```

Examples

To achieve perfection in loops and if statements, you need to practice lots of programs. The more problems you solve, the better your skills become. I recommend that you should first try them out independently and then look at the solution. It might be possible that your logic was even better than mine.

Note: All of the programs mentioned in this chapter were written by me initially in C++. As most of you would be unfamiliar with this language, I have changed the code to Java. You must know that the logic remains same in programming, just the way of implementation defer.

Example 1.

```
/* Print pattern
12345
23456
34567
45678
56789 */
class pat1
{
public static void main(String args[])
{
int c,c1,t;
for(c=1;c<=5;c++)
{
t=c;
for(c1=c;c1<=t+4;c1++)
System.out.print(+c1);
System.out.println();
}
}
```

```
}
```
Example 2
```
/* Printing a pattern
a
bb
bb
ccc
ccc
ccc
dddd
dddd
dddd /*
// Try this program after reading the chapter on Arrays
class pat2
{
public static void main(String args[])
{
char ch[]={' ','a','b','c','d'};
int c,c1,c2,n=1;
for(c=1;c<=4;c++)
{
for(c1=1;c1<=n;c1++)
{
for(c2=1;c2<=n;c2++)
System.out.print(ch[n]);
System.out.println();
}
n=n+1;
}
}
}
```
Example 3
```
/* A program that prints the following pattern
1234554321
1234 4321
123 321
12 21
1 1 */
class pat3
{
public static void main(String args[])
{
int c,c1,i=5,d=5,c2,s,g=0;
for(c=1;c<=5;c++)
{
for(c1=1;c1<=i;c1++)
```

```
System.out.print(+c1);
i=i-1;
for(s=1;s<=g;s++)
System.out.print(" ");
g=g+2;
for(c2=d;c2>=1;c2--)
System.out.print(+c2);
d=d-1;
System.out.println();
}
}
}
```

Example 4

/* A program that prints the following pattern


```
class pat4
{
public static void main(String args[])
{
int i,j,k,l,s=4;
char ch[]={' ','I','N','D','I','A'};
for(i=1;i<=5;i++)
{
for(j=1;j<=s;j++)
{
System.out.print(" ");
}
s=s-1;
for(k=1;k<=i;k++)
{
System.out.print(ch[k]);
}
for(l=i;l>1;l--)
{
System.out.print(ch[l-1]);
}
System.out.println("\n");
}
}
}
```

A program to print the following pattern

/* 7777777

55555

```
333
1
333
55555
7777777 /*
class pat5
{
public static void main(String args[])
{
int c,c1,s,n=7;
for(c=1;c<=4;c++)
{
for(s=1;s<c;s++)
System.out.print(" ");
for(c1=1;c1<=n;c1++)
System.out.print(+n);
n=n-2;
System.out.println();
}
n=3;
for(c=3;c>=1;c--)
{
for(s=1;s<c;s++)
System.out.print(" ");
for(c1=1;c1<=n;c1++)
System.out.print(+n);
n=n+2;
System.out.println();
}
}
}
```

Example 6.

```
// A program to print the following pattern
/*
54321
5432
543
54
5
/*
class pat6
{
public static void main(String args[])
{
int c,c1;
for(c=1;c<=5;c++)
```

```
{
for(c1=5;c1>=c;c1--)
{
System.out.print(+c1);
}
System.out.println();
}
} }
```

Example 7.

/* A program to print the following pattern

```
class pat7
{
public static void main(String args[])
{
int c,c1,c2,c3;
for(c=1;c<=5;c++)
{
for(c1=5;c1>=c;c1--)
{
System.out.print(" ");
}
for(c2=1;c2<=c;c2++)
{
System.out.print("*");
}
System.out.println();
}
}
}
```

Example 8

/* A program to print the following pattern

```
ZBBBBBBBB
AZBBBBBBB
AAZBBBBBB
AAAZBBBBB
AAAAZBBBB
AAAAAZBBB
AAAAAAZBB
AAAAAAAZB
AAAAAAAAZ */
class arr_pat
{
```

```
int c,c1,mid=1;
char arr[]={'A', 'Z','B'};
arr_pat()
{
for(c=1;c<=9;c++)
{
for(c1=1;c1<=9;c1++)
{
if(c==c1)
{
System.out.print(arr[1]);
mid=0;
if(mid==0)
System.out.print(arr[2]);
}
Else
System.out.print(arr[0]);
}
mid=1;
System.out.println();
}
}
}
class xyz
{
public static void main(String args[])
{
arr_pat obj=new arr_pat();
}
}
```

Exercises

Q1. What does conditional execution of a program mean?
Q2. Explain the use of ‘if’ statement with an example.
Q3. How is the “and” (&&) operator used with ‘if’ statement?
Q4. Design a menu based calculator using ‘if’ statement.
Q5. State the difference between ‘if’ and ‘switch’ statement.
Q6. What is the difference between while and do-while loop?
Q7. What advantage does ‘for’ loop holds over ‘while’ loop?
Q8. Describe ‘break’ and ‘continue’ statement.
Q9. State the use of counters in loop.
Q10. Write a program that prints the following patterns on the screen

Basic Exception Handling in Java

Java provides us an excellent exception management system, hence we can protect our programs from any run time errors. This robustness is achieved through keywords like "*try, catch, throw and throws*". The contents of this chapter are out of course if you compare them with the ICSE syllabus, but a basic knowledge will really help you in understanding the forthcoming chapters.

Exception

Exception is some unusual condition that sometimes generates during the execution of our code. The reason may be an invalid input, no input and even violation of some security features. If we don't handle the exception ourselves then the Java Run Time automatically handles and reports the exception, finally terminating the program. This can be embarrassing if you are writing a professional program (Take your own project as an example). Thus handling some exceptional condition is really important.

Using try and catch

The two keywords "*try and catch*" are extremely useful for handling run time exception. If we suspect a part of code to be prone to exceptions, we can enclose it within the try block. If an exception occurs, it is handled by the catch block as directed earlier by us. We can even manually throw an exception using "*throw and throws*" keyword.

Syntax—

```
try
{
// Statements
// Code
}
catch( ExcepionType obj)
{
//Catch block that handles exception
// Any suitable actions that are needed to be taken
}
```

Look at this program that will generate an error. Here we have not used try catch to protect our code from exceptions.

```
class airtexc2
{
public static void main(String args[])
{
int no,no2,ans;
no=10;
no2=0;
ans=no/no2;
System.out.println("Exception Occurred");
}
}
```

Now the same program is written using try and catch to handle exception.

```
class airtexc
{
public static void main(String args[])
{
int no,no2,ans;
no=10;
no2=0;
try
{
System.out.println("Try block begins");
ans=no/no2;
}
catch(ArithmeticException e)
{
System.out.println(" Sorry!!An Exception occured");
System.out.println(" Exception Details"+e);
}
System.out.println("EXCEPTION DETAIL PRINTED");
}
}
```

After looking at the code and the output of the program, some conclusions can be drawn
1. Program does not terminate in an exceptional condition if we use "*try-catch*" statements. This is proved by the fact that the line **"***System.out.println("EXCEPTION DETAIL PRINTED")*;**"** was executed after the exception occurred.
2. We can print the details of exception by simply printing the exception's object i.e. is "*e*".
Let us imagine that we are writing a program which manipulates digits in arrays (You will be learning about arrays in the last chapter). We suspect that there might be a condition where our program can generate.
(i). ArithmeticException
(ii). ArrayIndexOutOfBoundsException
How will be the two exceptional cases handled simultaneously? The answer to this problem is multiple catch blocks.
Syntaxtry
{
// Block of code to be inspected for errors
// Statements
}
catch(Exception1 obj)
{
// Handling exceptions
// All statements
}

```
catch(Exception2 obj)
{
// Handling exceptions
// All statements
}
```

throw and throws keywords---

throw-

Till now we have been handling exceptions automatically thrown by java runtime system due to certain lines in our code. The "*throw"* keyword is used to manually throw an exception. As soon as the interpreter encounters the line with the throw keywords, it searches for the catch statement of the try block in which the "*throw*" keyword is enclosed. If no catch block is found then the default exception handler handles the exception and terminates the program.

```
class thrw
{
public static void main(String args[])
{
try
{
System.out.println("Try block's execution begins");
throw new ArithmeticException ();
}
catch(ArithmeticException obj)
{
System.out.println("Inside the catch block");
System.out.println("Exception detail---"+obj);
}
}
}
```

throws

If we construct a method that throws an exception which it does not handle, the keyword "*throws*" warns the calling method of the exception.

```
return_type method( parameter list) throws exception
{
// Method's body
// Other statements
}
```

A straightforward and clear example will be a simple program that takes some input from the user.

```
import java.io.*;
class trws
{
public static void main(String args[]) throws IOException /. Use of "throws" keyword
{
int no;
```

```
InputStreamReader rd=new InputStreamReader(System.in);
BufferedReader inp=new BufferedReader(rd);
System.out.println("Enter a number");
String v1=inp.readLine();
no=Integer.parseInt(v1);
System.out.println("Multiplying a number");
no=no*no;
System.out.println("Square of the no. is "+no);
}
}
```

Exercise

Q1. What is an exception?
Q2. Why is exception handling a vital aspect of successful programming?
Q3. Explain the use of try-catch statements.
Q4. Which two other keywords other than ‘try-catch’ are used to handle exceptions?
Q5. When do these exceptional conditions occur?
(i). ArithmeticException
(ii). ArrayIndexOutOfBoundsException

Using Library Classes

Package

In Java, you can group a number of classes into a single unit known as packages which also act as a container for a group of classes.

Example---

There are several inbuilt java packages. They contain groups of classes that help us in writing our program.

java.io Java Input-Output package

java.lang Java language package

Defining a Package

A programmer, if desires can create his own package. This can be really helpful in certain conditions.

Syntax—

package name;

Remember: A class that defines itself inside a package "*pack*" should be placed inside a folder (directory) "*pack*".

In this way a class hierarchy can be built. A hierarchy of packages and even folders can thus be created.

Study this example for a better knowledge of packages.

```
package first;
class school
{
String name;
int marks;
public school()
{
name="Priyanka";
marks=94;
}
public void show()
{
System.out.println("Printing Student's name and marks");
System.out.println("Name is "+name);
System.out.println(name +"'s"+" marks are "+marks);
}
}
class sch
{
public static void main(String args[])
{
school obj=new school();
obj.show();
}
}
```

In this example we have constructed a package named first that contains two classes named school and sch. Save this program as "*sch.java*" under a folder "*first*". Use the cd

command in DOS to enter the directory "*first*". Compile the program in the usual way.
javac sch.java;
For executing the program, move above the "*first*" directory i.e. in the folder containing the "first" directory using the command "*cd..*". Then execute the program in this way----
java first.sch
As the class sch is in the package first, we can not compile the program using the normal way.
java sch // Wrong way

Classpath Woes

It becomes difficult for Java to search a package. As packages are as good as folders, it will be impossible for java to recognize the location of our package. In the example mentioned above we successfully created a package "*first*" and executed the code. This is because at the time of execution we were in a directory (*Java Prgs*) that was just above "*first*" in the folder's hierarchy. By default, java searches the specified package in the subdirectory of the folder in which we are working.
If you are not in the directory that is just above the "*package*" directory (Have a look at the screenshot that "*Java Prgs*" is the directory in this case), you have to specify a

Importing Package

Look at the program in the previous chapter that demonstrates the use of "*throws*" keyword. We have imported the "*io*" package in the program.
The Line goes like this…
import java.io.*;
This might give you some understanding of using a particular package. The keyword "*import*" helps us in this.
I have created three packages named highest, higher and high. Suppose that "*higher*" lies inside the package "*highest*". Higher too contains a package "*high*". The high package contains three classes. Let their names be first, second and third.
If you want to import the class "*second.java*", you will writeimport
highest.higher.high.second;
There is an alternative to this method—
import highest.higher.high.*;
If you use the "*import*" statement in this way, all the classes will be imported including "*second.java*" which we desire to import. Can you guess the reason behind it? It is due to the "*". Here "*" works as a wild card character that represents all the classes.
General Syntax—
import package1.package2.package'n'.classname;

A further study of java.io package

The "java.io" package contains classes that support java's input & output system. The *io* package holds immense value as it contains several classes like "*InputStreamReader*" & "*BufferedReader*". Thus the io package must be imported in each and every program for taking input from the console.
Once again have a look at an example that takes an input from its user.

```
import java.io.*; //importing all classes that are in the java.io package
class demoinp
{
public static void main(String args[]) throws IOException // "throws" keyword is used to
```

```
// handle exception
{
String name; //An object of class String that takes a series of characters as an input
int marks; // Variable that accepts the marks of user
InputStreamReader rd=new InputStreamReader(System.in); // Creating an object of the
//class InputStreamReader
BufferedReader inp=new BufferedReader(rd); // Creating an object of the class
// BufferedReader
System.out.println("Enter your Profile");
System.out.println("Enter your name"); // Message to the user
name=inp.readLine(); // "name" getting the input
System.out.println("Enter your marks");
String v1=inp.readLine(); // Readline method used
marks=Integer.parseInt(v1); // variable “mark” getting the input
if(marks>= 75 && marks<=100)
System.out.println("Well Done!!!");
else
if(marks>=50 && marks<75)
System.out.println("Good,You can improve");
else
if(marks>=0 && marks<50)
System.out.println("You need to really work hard!!");
else
System.out.println("Invalid Input");
}
}
```

Line 1:
Importing the io package.
Line 11 & 13:
We have created objects of two classes, InputStreamReader and BufferedReader. These classes help us to get an input as follows -
1. InputStreamReader – Input Stream that translates bytes to characters
2. BufferedReader ------BufferedReader wraps “System.in”, thus creating a character stream.
Line 19:
Input is always received by the Java “*Input Stream”* in the form of a String (Strings are a series of characters & digits, e-g. “Hello”, “agh1”, “12”. The variable “*name*” gets the input with the help of the readLine() function.
Line 22 & 23:
In the Line 22, we take the input in the form of String by the readLine() method. A variable “*v1*” gets the input. Then in the next line we have a statement that causes the variable “*marks*” to take the input as an Integer. A method Integer.parseInt(String name) accomplishes this task.
Remember:: There are two types of Streams in Java—
1. Byte Stream
2. Character Stream

The byte stream is used for reading & writing binary data while the character Streams handle input & output of characters. *Although at the basic level everything is in bytes.*

Access Protection

Using three access modifiers "*public, private & protected*" we can control the visibility of our members in and out of the package and class.

Public

Inside Class Visible
Outside Class Visible
Inside Package Visible
Outside Package Visible

Private

Inside Class Visible
Outside Class Not-Visible
Inside Package Not-Visible
Outside Package Not Visible

Protected

Inside Class Visible
Inside Package Visible
Outside Package(subclass) Visible
Outside Package(non-subclass) Non-Visible

Wrapper Class

Wrapper class is present in the "java.lang" package. Wrapper class was introduced to treat primitives like int, double, float, byte, short etc., as Objects.
For example. Consider this following code:

```
//This is correct
Vector vec = new Vector();
vec.addElement("Nitish");
//This is wrong, produces Compiler error!!
Vector vec = new Vector();
int num = 5;
vec.addElement(num);
```

Why the Compiler is complaining? This is because 'int' is not an object like String, it's a primitive.
So to make sure int, short, byte, long, float, double, char are treated as Objects , Wrapper classes where introduced.
If there is no Wrapper class , you can't use primitives when using Collections FrameWork.
Collections FrameWork (java.util)form the main part of Java.They have Classes like Vector, List, Date, Map, HashSet Iterator, etc., which are used for storing, sorting, and various functions.
What in C++ they call STL(Standard Template Library), they call it as 'Collections Framework' in Java.
So for each primitive there is a corresponding Wrapper class. For int ->Integer, float ->Float,double->Double
etc., A Wrapper class wraps around a primitive and makes it an Object.

For eg.,
int primInt = 23;
Integer wrapInt = new Integer(primInt);
Vector vec = new Vector();
vec.addElement(wrapInt);
Since Integer is an Object , it can be added to Vector. As you can see the Integer class wraps around the primitive 'int' variable, so it's called a Wrapper class.
Similarly you can change float, double, byte, etc., as Objects using Wrapper class.

Exercise

Q1. What is a package in Java?
Q2. Name some built in java packages with their uses.
Q3. How does a package help us in organization of our classes?
Q4. How does the Java Runtime System searches for a required package?
Q5. Explain the use of ‘import’ statement with the help of an example.
Q6. Write a program that takes the input from the user. Describe the classes you have used to get the input.
Q7 How access modifiers control the visibility in a package?
Q8. Explain wrapper class.
Q9. How is bytes stream different from Character stream?
Q10. How can we import all the classes of a package in our program?

String Handling in Java

In the second chapter you read about several data types. The concept of "*Strings*" was left unattended, as they are entirely different from any other conventional data type. In this chapter we shall study about each and every aspect of String handling.

What is a String?

You are already familiar with the name of "*java.lang*" package. This package contains many classes that support various runtime processes. This package is really important which you can understand after reading this---

"*The java.lang package is so vital for the execution of java programs that it is imported automatically in our code, thus we don't have to do it ourselves manually each & every time*".

String is a class contained by this package. Using the object of this class we can manipulate or work with series of characters.

A similar class-StringBuffer

It's not possible to edit the contents of String once initialized. Example —

String word="Matter"

The contents of String can be changed completely, i.e. String '*word*' can contain values such as "solid", "type" or "af12" etc. You might now wonder, why earlier it was mentioned to be '*non-editable*'? This is all because you can't remove the characters of a string by choice. Example - Removing 'M'& "ter" changing the sting value to "*at*". Hence sometimes we have to use StringBuffer as it is editable.

Remember:: *StringBuffer is a class that belongs to the java.lang package*.

Creating a String

To create a String we have to follow a procedure that is somewhat similar to creating an object.

General way of creating an object—

Classname obj=new Classname();

Syntax for creating a String ---

String name=new String();

But if you want to declare and initialize a String simultaneously then the procedure is a bit different. Imagine creating a string and initializing it with the word "India".

String name="India"

Concatenation of String

Sometimes we have two strings, we can then use '+' operators to join them, this is known as Concatenation.

Have a look at this program…

```
class concat
{
public static void main(String args[])
{
String name="Ankit";
int marks=93;
System.out.println("Printing Details"+ " of "+"the student" ); //Concatenating Strings
System.out.println(name+"'s"+" marks "+"are--"+marks);
}
}
```

Manipulating Strings

There are several methods associated with the "*String*" class that help us to manipulate it. You already know that call to a method is made by its object in this manner---
obj.method();
Thus in a similar way we will be using the methods.
Example- String name="India";
name.length(); // name is the object of String class and length() is the method.
Finding length of a String---
Length of a string is defined as the number of characters in a String. We use the "*length()*" method to get the length of a String.
Syntax—
name.length();
Let us look at a program using this method-----

```
import java.io.*;
class length
{
public static void main (String args[]) throws IOException
{
String a;
int m;
InputStreamReader reader=new InputStreamReader(System.in);
BufferedReader input=new BufferedReader(reader);
System.out.println("Enter a String ");
a=input.readLine();
System.out.println(a.length());
}
}
```

Remember: A space is also counted as a character.

Extracting Characters from a String

A String is made of several individual characters. A character can be an alphabet, a digit or even a space. We can easily extract a single or multiple character(s) from a String. The methods used are —
1. charAt();
2. getChars();

charAt()

The charAt() method is used to extract a single character from a String.
Example--

```
String xyz=" Perfect ";
char c=xyz.charAt(1);
```

The charAt() method will extract the first character of the String and will initialize the variable "*c*" with it. If you imagined the first character of the String to be 'P', think again! The characters of String start with 0, so the first character will be 'e'.
The complete program..

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

```
{
String xyz="Perfect";
char c=xyz.charAt(1);
System.out.println("The character extracted is "+c);
}
}
```

getChars()

The getChars() method extracts multiple characters from the String.

Syntax—

getChars(int StartPos, int EndPos, TargetArrayString, int StartArrayIndex);

Let us suppose that we have a string “Networking”. You have to extract four characters “work” from the string. As we have multiple characters to extract and store, a single character variable won’t be able to hold them all. Thus a character’s array is needed (An Array of characters can store as many characters as you wish!)

```
class extch2
{
public static void main(String args[])
{
String xyz="Networking";
char ch[]=new char[5];
xyz.getChars(3,7,ch,0);
System.out.println(ch);
}
}
```

getBytes()

The getBytes() method extracts String characters as bytes and stores it in a character array.

Syntax--

getBytes(int StartPos, int EndPos, TargetArrayString, int StartArrayIndex);

```
class extBytes
{
public static void main(String args[])
{
String abc="Networking";
byte b[]=new byte[5];
abc.getBytes(3,7,b,0);
System.out.println(b);
}
}
```

At the time of compilation you might get this error—

Note:: filename.java uses or overrides a deprecated API.

Note:: Recompile with -Xlint:deprecation for details.

This shows that this program uses an API(Application Programming Interface) that is deprecated (No longer popularly used), hence this error is flashed. You can ignore this and simply execute the program.

toCharArray()

The toCharArray() method is used to convert a Strings content entirely into an array.

```
class arrycon
{
public static void main(String args[])
{
String name="Mobiles";
char c[]=new char[8];
c=name.toCharArray();
System.out.println(c);
}
}
```

The substring method is used to extract some specified characters of a String and create a new String from these characters.
As an example, consider a String 'str' initialized with the word "Penguin".
If we write--
new1=str.substring(5);
If you will print the String "*new1*", the output will be "*in*".
What will you do to extract "*Pen*" as a String from Penguin. You will have to supply two parameters to the substring() method.

```
class substr
{
public static void main(String args[])
{
String stv="Penguin";
String new1;
new1=stv.substring(0,3);
System.out.println("Contents of the new String are "+new1);
}
}
```

trim()

The trim() method is used to remove spaces in a String from both of its ends.

```
class trm
{
public static void main(String args[])
{
String jkl=" Welcome to India";
jkl=jkl.trim();
System.out.println(jkl);
}
}
```

Changing the case of Alphabets

You can change the case of alphabets easily from lower to uppercase and vice-versa using toLowerCase() & toUpperCase().

```
class case1
```

```
{
public static void main(String args[])
{
String name="bill gates ";
name=name.toUpperCase();
System.out.println(name);
}
}
class case2
{
public static void main(String args[])
{
String name="COMPUTER APPLICATION";
System.out.println(name.toLowerCase());
}
}
```

equals()

The equals() method is used to check if true Strings are identical. You should keep in mind that “hello” and “HELLO” are not identical as their cases are different.

```
class eql
{
public static void main(String args[])
{
String str="name";
String str2="Name";
if(str.equals(str2))
System.out.println("The two Strings are equal");
else
System.out.println("The two Strings are unequal");
}
}
```

If you don’t want to consider the difference of Case, the method “*equalsIgnoreCase()*” should be used.

```
class eql
{
public static void main(String args[])
{
String str="name";
String str2="Name";
if(str.equalsIgnoreCase(str2))
System.out.println("The two Strings are equal");
else
System.out.println("The two Strings are unequal");
}
}
```

replace()

There may be a certain condition when you want to replace a certain character with another character.
Syntax----
name.replace('initial character', 'final character')

```
class rep
{
public static void main(String args[])
{
String str="mango";
String str2;
str2=str.replace('m','t');
System.out.println("The second String is "+str2);
}
}
```

reverse()
The reverse() method is used to reverse the contents of a String.

```
class rev
{
public static void main(String args[])
{
StringBuffer frm=new StringBuffer("Sun Microsystems");
frm=frm.reverse();
System.out.println("String in its reverse order is--"+frm);
}
}
```

compareTo()method
This method is used to sort strings in Dictionary order. The method returns '0',if two Strings are exactly equal. A value less than '0' is returned if the invoking string comes before the other string in the dictionary. Hence it is absolutely clear that if a value greater than '0' is returned, the invoking string will be next to the other string.
Example---

```
class dict
{
public static void main(String args[])
{
String name="Anjali";
String name2="Ankit";
if(name.compareTo(name2)<0)
System.out.println("1st");
else
System.out.println("2nd");
}
}
```

As "j" proceeds "k" in the dictionary, in this comparison a value smaller than zero is returned. You can add an IgnoreCase to this method making it unaffected by case

changes. Finally the method will be..
name.compareToIgnoreCase(name2)

Examples

As already discussed in chapter five, perfection is achieved by practice and persistence. Here are some examples, you should try to solve without looking at the solutions.
Note: In some programs arrays have been used, skip them until you read that chapter.

Example 1

```
/* Finding the largest word in a String
Example – "I am back"
Answer = Back */
class word
{
private String str;
private int c,len,l,u,c1,lar;
private int arr[]=new int[50];
//class constructor
word()
{
str="I am a proud Indian";
c=0;
l=str.length();
}
//Method to find largest word
public void store_lar()
{
u=0;
for(c=0;c<l;c++)
{
if(c==l-1 || str.charAt(c+1)==' ')
{
len=(c+1)-u;
arr[c1]=len;
c1++;
u=c+2;
}
}
}
//Calculate the length of Largest no
public void largest_no()
{
lar=arr[0];
for(c=0;c<arr.length;c++)
{
if(lar<arr[c])
lar=arr[c];
}
```

```
}
//Printing largest word
public void printlar()
{
u=0;
for(c=0;c<l;c++)
{
if(c==l-1 || str.charAt(c+1)==' ')
{
len=(c+1)-u;
if(len==lar)
{
for(int x=u;x<=c;x++)
System.out.print(str.charAt(x));
}
u=c+2;
}
}
}
}
class largest_word
{
public static void main(String args[])
{
word obj=new word();
obj.store_lar();
obj.largest_no();
obj.printlar();
}
}
```

Example 2

```
/* A program to sort names in Dictionary order */
class dictionary
{
static String arr[]={"Ankit","Dhanajay","Vinod","Priyanka","Nitish"};
public static void main(String args[])
{
for(int c=0;c<arr.length;c++)
{
for(int c1=c;c1<arr.length;c1++)
{
if(arr[c].compareToIgnoreCase(arr[c1])>0)
{
String temp=arr[c];
arr[c]=arr[c1];
arr[c1]=temp;
```

```
}
}
System.out.println("Sorted String is");
System.out.println(arr[c]);
}
}
}
/* A program to print the character which has largest ASCII value as well as the char
itself
E.G- education
Output- u 117
*/
class lar_ch_val
{
String str;
int c,lar,len,val=0,temp;
char ch;
public lar_ch_val()
{
str="BILL GATES";
c=0;
len=str.length();
}
public void lar()
{
for(c=0;c<len;c++)
{
ch=str.charAt(c);
temp=(int)ch; //explicit conversion of data types
if(temp>val) // finding max ASCII value
val=temp;
}
}
public void out()
{
ch=(char)val; //Getting no. back to char form
System.out.print(ch);
System.out.print("\t"+val);
}
public static void main(String sv[])
{
lar_ch_val obj=new lar_ch_val();
obj.lar();
obj.out();
}
}
```

Example 3

```
//reverse from last number...
/*Eg-"India is great";
Print-"great is India";*/
class work_on
{
private String x;
private int c,l,t,k,b;
// Class constructor
work_on()
{
x="India is great";
l=x.length();
k=l-1;
b=0;
}
public void rev()
{
for(c=l-1;c>=b;c--)
{
k--;
t=k;
while(t<l)
{
System.out.print(x.charAt(t));
}
}
}
}
class last_rev
{
public static void main(String args[])
{
work_on obj=new work_on();
obj.rev();
}
}
```

Example 4

```
/* A program to convert a word into pig latin.
E.G-- Input=King
Output=ngiKay*/
import java.io.*;
class piglatin
{
public static void main(String args[]) throws IOException
{
```

```
String word;
InputStreamReader rd=new InputStreamReader(System.in);
BufferedReader inp=new BufferedReader(rd);
System.out.println("Enter a word");
word=inp.readLine();
word=word.trim();
int c,l=word.length(),pos=0;
char ch;
for(c=l-1;c>=0;c--)
{
ch=word.charAt(c);
if(ch=='a'|| ch=='e'|| ch=='i'|| ch=='o'|| ch=='u')
pos=c;
}
for(c=pos+1;c<l;c++)
System.out.print(word.charAt(c));
for(c=pos;c>=0;c--)
System.out.print(word.charAt(c));
System.out.print("ay");
}
}
```

Example 5

```
/* Reversing a String /*
import java.io.*;
class rev
{
public static void main(String args[]) throws IOException
{
InputStreamReader reader=new InputStreamReader(System.in);
BufferedReader input=new BufferedReader (reader);
String inp;
int c,b=0,e;
System.out.println("Enter a String to reverse");
inp=input.readLine();
e=inp.length();
char arr[]=new char[1000];
inp.getChars(b,e,arr,0);
System.out.println("**********************");
for(c=e;c>=0;c--)
System.out.print(arr[c]);
}
}
```

Example 6

```
/* A program used to search a number /*
class search_word
{
```

```
public static void main(String args[])
{
String org="I am slowly improving my skills";
org=org.trim();
String find="my";
String temp=new String();
int c,len=org.length();
int beg=0;
boolean swt=false;
for(c=0;c<len;c++)
{
if(c==len-1 || org.charAt(c+1)==' ')
{
temp=org.substring(beg,c+1);
//System.out.println(temp);
if(temp.equals(find)) //Ignore Case Optional
swt=true;
beg=c+2;
}
}
if(swt==false)
System.out.println("Not Found!!");
else
System.out.println("Found!!!");
}
}
```

Example 7

```
/* A program to print name in short form
E.G- Mohandas Karamchandra Ganghi
output- M.K. Gandhi*/
class shname
{
String name=new String();
String namelast=new String();
int l,c,k=0,st,t;
char ch[]=new char[15];
shname()
{
name="Bhim Rao Ambedkar";
l=name.length();
c=l-1;
}
void workaround()
{
for(c=l-1;c>=0;c--)
{
```

```
if(name.charAt(c-1)==' ')
{
st=c;
break;
}
}
namelast=name.substring(st,l);
for(c=st-2;c>=0;c--)
{
if(c==0 || name.charAt(c-1)==' ')
{
ch[k]=name.charAt(c);
k++;
}
}
t=k;
}
void print_it()
{
for(c=t-1;c>=0;c--)
{
System.out.print(ch[c]+".");
}
System.out.print(" ");
System.out.print(namelast);
}
public static void main(String args[])
{
shname st=new shname();
st.workaround();
st.print_it();
}
}
```

Exercise

Q1. How does a String differ from other conventional data type?
Q2. State the difference between the classes “String” and “StringBuffer”.
Q3. What do you understand by the term concatenation of String?
Q4. Explain getChars and getBytes.
Q5. Why two same strings with different cases reported unequal by Java

Arrays - A collection of Similar Data Type

Introduction

The concept of several data type and String has already been taught to you. They provide us with a lot of flexibility to work with any type of data. You might be wondering if you have mastered everything, but surely this is a wrong supposition until you have worked with Arrays. Soon you will realize that Arrays are one of the key features of Java.
Create a program that finds the largest number between the three numbers entered. You will surely be able to do it. Now picture yourself in a condition where the numbers of variables entered are also unknown, now you will get stuck here. To write such a program you need to learn arrays.

Array

An Array is a collection of similar type of finite numbers of data values, stored in memory location. An array of integers, characters, Strings as well as objects can be created.
There are two types of Arrays —
1. One-Dimensional
2. Multi-dimensional

One-Dimensional

Have a look at the declaration of a one-dimensional array.
Syntax— int arr []=new int[2]; //Declaration of an Array that can hold 2 variables
OR
int []arr=new int[2];
An array is said to be single or multi-dimensional (two, three) by the number of subscripts. An array with a single subscript always has one square bracket. So how is array able to store multiple values of any kind of data?
Here is the answer…
int arr[0]=3;
int arr[1]=4
The different values are stored in indexes that begin with zero. Going through the internal working you will find that our temporary memory is allocated serial wise in a continuous way.

0 1 2 3 4 5 6 7 8

You can work over these individual values easily.
Remember: Index of arrays always begin with zero.

Multi-Dimensional

A multidimensional array has more than one subscript. Arrays with two and three subscripts are two and three dimensional arrays respectively.
A multidimensional array stores in rows and columns and not simply serial wise.
In this picture we can see a double-dimensional array (3*3). The values are stored in this way-
Row Column Value
0 0 3
0 1 9
1 2 6
2 0 7

Other indexes are empty.
Declaration for this multidimensional array is ---
int ar[][]=new int[3][3];

Working with Arrays

Consider an Array
int Arr[]=new int[5]; //Declaration

Initialization ---

We can initialize arrays by two ways-

```
Arr [0]=1;
Arr [1]=2;
Arr [2]=3;
Arr [3]=4;
Arr [4]=5;
```

A more convenient way ---

```
int Arr[]={1,2,3,4,5};
```

Input from the User

You will have to use InputStreamReader and BufferedReader classes to get input from the user. As an array might take hundreds of values, using loops (mostly 'for') to get the input is a good idea.

```
import java.io.*;
class arr
{
public static void main(String args[]) throws IOException
{
int ar[]=new int[5]; // Array Declaration
int c;
InputStreamReader rd=new InputStreamReader(System.in);
BufferedReader inp=new BufferedReader(rd);
for(c=0;c<5;c++) //a loop to take four inputs
{
System.out.println("Enter array elements");
String v1=inp.readLine();
ar[c]=Integer.parseInt(v1); //According to the value of 'c' the array gets initialized
}
System.out.println("Printing details of the Array");
for(c=0;c<5;c++)
{
System.out.println(+ar[c]);
}
}
}
```

Two Worked Out Examples

"Practice makes a student perfect!", this statement is the ultimate truth in programming. Have a look at these examples and thereafter you will definitely understand Arrays better. Majority of the programs will be covered in the "Examples" section.

Sum of Array Elements

```
import java.io.*;
class sum
{
public static void main(String args[]) throws IOException
{
int sum=0,lim,c;
int arr[]=new int[50];
InputStreamReader rd=new InputStreamReader(System.in);
BufferedReader inp=new BufferedReader(rd);
System.out.println("Enter the limit for numbers");
String v1=inp.readLine();
lim=Integer.parseInt(v1);
for(c=0;c<lim;c++)
{
System.out.println("Enter Array Elements");
String v2=inp.readLine();
arr[c]=Integer.parseInt(v2);
}
for(c=0;c<lim;c++)
{
sum=sum+arr[c];
}
System.out.println("Sum of Array Elements is" +sum);
}
}
```

FindingLargest and Smallest number in an Array

```
class larsml
{
public static void main(String args[])
{
int arr[]={8,5,4,3,9};
int c=0,lar=arr[c],sml=arr[c];
System.out.println("Processing.....");
for(c=0;c<5;c++)
{s
if(arr[c]>lar)
lar=arr[c];
if(arr[c]<sml)
sml=arr[c];
}
System.out.println("Largest No. is "+lar);
System.out.println("Smallest No.is "+sml);
}
}
```

String Array

A program demonstrating an array of “String” class

```
// Printing marks of Students
import java.io.*;
class record
{
public static void main(String args[]) throws IOException
{
String arr[]={"Nilay","Nitish","Nitin"};
int marks[]={97,92,90};
String inp;
boolean fnd=false; // Initially initialized with false to demonstrate that our search has
//not been successful
int c;
InputStreamReader rd=new InputStreamReader(System.in);
BufferedReader input=new BufferedReader(rd);
System.out.println("Enter a Name");
inp=input.readLine();
for(c=0;c<3;c++)
{
if(inp.equalsIgnoreCase(arr[c]))
{
System.out.println("Marks are " +marks[c]);
fnd=true; //The “Not Found!!!” message will not be printed in this case
}
}
if(fnd==false)
System.out.println("Not Found!!!");
}
}
```

Operations on Array

Many operations can be performed on elements in an array. According to the scope of syllabus we will discuss some of them here ---

1. Searching(Linear, Binary)
2. Sorting(Selection, Bubble)

Searching

We can easily search for any content in an array. Let us suppose that our array has four integers and we want to find out if one of them is ‘6’. Searching will serve the purpose.

1st Approach

The first approach is to compare each and every element of the array with the number being searched.

Say we have an array with elements ---

4, 5, 2, 0, 3

Searched Number- 6

We will start the searching from index zero which contains the value four, the process will continue till the limit of Array (4) is reached.

Actual Program – LINEAR SEARCH

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

```
class linsrch
{
public static void main(String args[]) throws IOException
{
int arr[]={5,2,7,8,1,0};
int c,no;
boolean fnd=false;
InputStreamReader rd=new InputStreamReader(System.in);
BufferedReader inp=new BufferedReader(rd);
System.out.println("Enter a number to Search");
String v1=inp.readLine();
no=Integer.parseInt(v1);
for(c=0;c<6;c++)
{
if(no==arr[c])
{
System.out.println("Successful Search");
fnd=true;
}
}
if(fnd==false)
System.out.println("Requested Number Not Found!!");
}
}
```

2nd Approach

The second approach of searching is a bit more complicated than the first one. This scheme of searching can only be applied on a sorted array (An array with elements arranged either in ascending or descending order). The basic concept is ---

An array num[] ={4, 9, 11, 13, 15, 19};

Imagine that you want to know if this array "*num*" contains the digit '9'. Follow these steps to search for the number ---

Have two variables, low and high. Variable low should be initialized with zero and high with the array's limit (Use the function *array_name.length*).

1. Sum up the lower and upper limit. (In this case sum=6).
2. Divide it by two. (Here the answer is 3).
3. Check the value in the array's index. (Here it is 13)
4. If the searched number is greater than the number in the index, set the low variable to the index number.

If the searched number is smaller (Just like our present example) set the upper bound to the index.

5. Repeat the process till the number is found.

Actual Program --- Binary Search

```
import java.io.*;
class binsrch
{
public static void main(String args[]) throws IOException
```

```
{
int arr[]={3,5,6,9,10};
int no;
int low=0,up=arr.length,mid;
boolean fnd=false;
int add=0;
InputStreamReader rd=new InputStreamReader(System.in);
BufferedReader inp=new BufferedReader(rd);
System.out.println("Enter a number");
String v1=inp.readLine();
no=Integer.parseInt(v1);
while(low<=up)
{
mid=(low+up)/2;
if(no==arr[mid])
{
fnd=true;
add=mid;
break;
}
else
if(no>arr[mid])
low=mid+1;
else
up=mid-1;
}
if(fnd==true)
{
System.out.println("Search Successful!!");
System.out.println("Index No. " +add);
}
else
System.out.println("Search Unsuccessful!!");
}
}
```

Sorting

The word “sorting” means to arrange systematically in a certain order, imagine a telephone directory that contains names in an alphabetically sorted way. Here we will study to sort numbers in increasing and decreasing order.

1st Approach

The first approach to sorting is the “*selection sort*”. In this technique the smallest number is checked and placed at the first index. The checking is again done in between the rest of the numbers and the smallest number is calculated. If we are sorting the array in descending order, the largest number is calculated.

Actual Program ---

```
import java.io.*;
class selsort
{
public static void main(String args[]) throws IOException
{
InputStreamReader reader=new InputStreamReader(System.in);
BufferedReader input=new BufferedReader (reader);
{
int k,n,temp=0,c,j;
int ever[]=new int[150];
System.out.println("Enter the Limit of Array");
String v1=input.readLine();
n=Integer.parseInt(v1);
for(c=0;c<n;c++)
{
System.out.println("Enter the Element to sort ");
String v2=input.readLine();
ever[c]=Integer.parseInt(v2);
}
for(k=0;k<n ;k++)
{
for(j=k+1;j<n;j++)
{
if(ever[j]<ever[k])
{
temp=ever[k];
ever[k]=ever[j];
ever[j]=temp;
}
}
}
for(c=0;c<n;c++)
{
System.out.println("Array after sorting is"+ever[c]);
}
}
}
}
```

```
//A program to sort an array using bubble the sort technique
import java.io.*;
class bub
{
public static void main(String args[]) throws IOException
{
```

```
InputStreamReader reader=new InputStreamReader(System.in);
BufferedReader input=new BufferedReader (reader);
{
int i,k=0,j,n,temp=0;
int a[]=new int[150];
System.out.println("Enter the Limit of Array");
String v1=input.readLine();
n=Integer.parseInt(v1);
for(i=0;i<n;i++)
{
System.out.println("Enter the Element to sort ");
String v2=input.readLine();
a[i]=Integer.parseInt(v2);
}
i=0;
for(k=0;k<n;k++)
{
for(j=0;j<n-1;j++)
{
if(a[j]>a[j+1])
{
temp=a[j];
a[j]=a[j+1];
a[j+1]=temp;
}
}
}
for(i=0;i<n;i++)
{
System.out.println(+a[i]);
}
}
}
}
```

Exercise

Q1. Explain arrays.
Q2. State the difference between Single or Multidimensional array.
Q3. Why loops are are used to take input?
Q4. State the difference between String and arrays of character.
Q5. Explain the two types of searching with their algorithm.
Q6. What do you understand by the term sorting?
Q7. State the significance of subscript in arrays.