JAVA NOTES-BETA

Revision of Class IX Syllabus

Programming paradigms are a way to classify programming languages based on their features. Paradigm means organizing principle of a program. It is an approach to programming.

Procedure Oriented Programming: A Procedure oriented programming approach allows the users to develop their logic by using a number of functions that would enhance the program's productivity.

Example BASIC, COBOL, C

Object Oriented Programming: An Object Oriented Programming is a modular approach, which allows the data to be applied with a stipulated program area. It also provides the reusability feature to develop productive logic, which means to give more emphasis on data.

Basic Principles of OOP:

1. **Abstraction**: The act of representing essential features, without including the background details.
2. **Inheritance**: Capability of one class of things to inherit capabilities or properties from another class.
3. **Encapsulation**: Wrapping up of data and functions into a single unit.
4. **Polymorphism**: Polymorphism is the ability for a message or data to be processed in more than one form.

Java compilation process:

1. Java programs are written in ".java" file. (source code) and then compiled by Java compiler.
2. **Byte code**: Java compiler converts the source code into an intermediate binary form called the byte code.

3. **Java Virtual Machine (JVM)**: It a java interpreter that converts byte code into machine to various platforms.
4. **Just In Time(JIT)**: It is part of the JVM and it compiles byte code into executable code in real time, one piece-by-piece, demand basis.

Characteristics of Java:

1. Write Once Run Anywhere (WORA): The Java programs need to be written just once, which can run on different platforms without making changes in the Java program.
2. Light Weight Code: With Java, no huge coding is required.
3. Security: Java offers many enhanced security features.
4. Object Oriented Language: Java is Object Oriented language, thereby, very near to real world.
5. Platform Independent: Java is essentially platform independent. Change of platform does not affect the original Java program.

Types of Java program:

1. Internet Applets: The programs executed inside the Java based web browser.
2. Java Applications: The programs developed by the users.

Java libraries:

A package is a collection of various classes. Each class contains different functions.

A package can be inclued in the program by using a keyword 'import'.

ex) import java.io.*;

```
import java.util.*;
```

The (*) sign denotes that all the classes of the concerning package will be made available for use in your program.

Keywords or Reserved words:

Java reserved words or the keywords are the words which carry special meaning to the system compiler. Such words cannot be used for naming a variable in the program.

case , switch, else, break , static, do, const, throws, float , char, try, int, double, void, goto, for, while, new, import , boolean, long, if, byte , package, private, catch, short, public, class , default.

Output statement: System.out.println( );

Comment line: The comment statement is a non executable statement in Java.

These are the remarks given by the user.

Types: single line comment	//----------
Multiline comment	/* ---------- */
Document comment	/** --------- **/

ASCII

A - Z	65 - 90
a-z	97 - 122
0-9	48 - 57
white space	32

Token: The smallest individual unit in a program is known as a Token

Types:

1. **Keywords**: Keywords are the reserved words that convey a special meaning to the language compiler. These are reserved for special purpose.
   Example) class, int, void, float

2. **Literals:** Literals or constants are data items that are fixed data values (do not during the execution of the program).
   Types:
   a) Integer Literals are whole numbers without any fractional part.
   Decimal, Octal, Hexa decimal.
   Ex)   a= 505,   b=-15

   b) Real literals are numbers having fractional parts.

   Ex)   p=16.79   , q=-1.005

   c) character literal is one character enclosed in single quotes.

   Ex) 'x', '9','*''

   d) String literals are multiple character enclosed with double quotes.

   Ex) name= "aravind"
   e) Boolean literal: the Boolean type has two values, true or false.

   f) Null literal has one value, the null reference. A null literal is always of the null type.

3. **Separators:** The following nine ASCII characters are the separators(punctuators).
   (   )   {   } [   ]   ;   ,   .

4. **Identifier**: Identifiers are fundamental building block of a program such as a variable, class, method etc.

It's used as the general terminology for the names given to different parts of the program. double area=15.6;

Rules for forming identifiers:

- Identifiers can have alphabets, digits and underscore and doller sign characters.
- They must not be a keyword or Boolean literal or null literal
- They must not begin with a digit
- They can be of any length

5. **Operators:** Operators are special symbols that cause an action to take place.
   Arithmetic, relational, logical, conditional operators.
   A=b*c;
   Escape sequences: Nongraphic characters are those characters that cannot be typed directly from keyboard. E.g. backspace, tabs etc.,
   An escape sequence is represented by a backslash (\ ) followed by one or more characters.

Escape sequence	Nongraphic character
\b	Backspace
\f	Formfeed
\n	Newline or linefeed
\r	Carriage return
\t	Tab space
\	Backslash
\'	Single quotes
\"	Double quotes
\?	Question mark
\0	null

Data types:

Data Types are means to identify the type of data and associated operations of handling it.

Two types:

1. Primitive or fundamental or Instrinsic (predefined)

int, long, float, double, char, short, byte , boolean

2. Reference or composite (user defined)
   class, string, array

<PRIMITIVE DATA TYPE WITH SIZE CHART>

Variable:

A variable is a named temporary memory location to store the values.
eg) float b = 15.5; <data type> <var name> = <value>;
char ch = 'G';

Initializing a variable:

Initializing a **variable** means specifying an initial value to assign to it (i.e., before it is used at all). Notice that a **variable** that is not **initialized** does not have a defined value, hence it cannot be used until it is assigned such a value.

Static initialization: Initialize the variable at the time of declaration

int b=5,a=0; float c=0.0

Data type	Value
Int	0
Long	0
Float	0.0
Double	0.0
Char	'\u000'
String	" "
Boolean	False

Dynamic initialization : Inilialize the variable at program execution time.(at run time)
int b=5, c=6;
int A= b+c;

<u>Arithmetic expression:</u>

A set of variables, constants and arithmetical operators used together to yield
a meaningful result is known as Arithmetical Expression.
d= a*b+c/4;

Types:

1. ***Pure Expression***: An arithmetical expression that uses all its components of same data type.
int a,b;
int c=a+b*4;

2. ***Impure expression:*** An arithmetical expression in which one or more components are of different data types.
int a; float f; double d;
double s= a*f/d;

Type conversion: In a mixed expression, the result can be obtained in any one form of its datatypes. Hence, it is needed to convert the various datatypes into a single type. Such conversion is termed as Type conversion.
[Converting one form of data type into another form of data type]

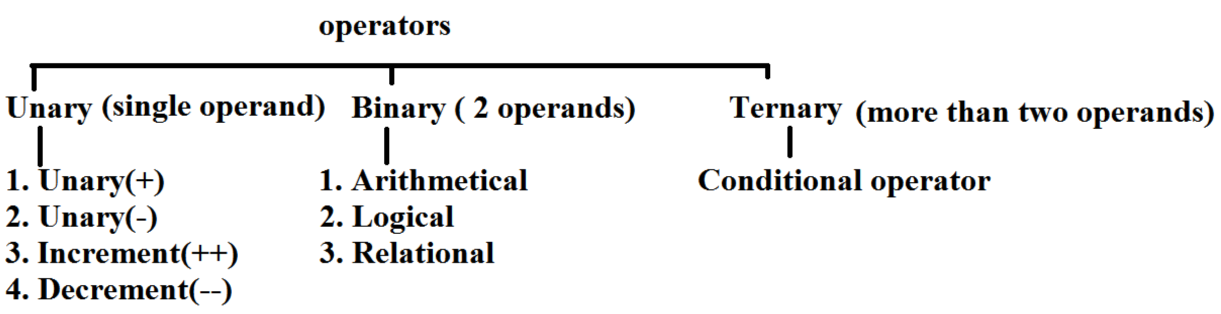
Types:

1. Implicit type conversion (Coercion)
The data type of the result gets automatically converted into the highest data type available into the expression without any intervention of the user.
Hierarchy of Data types (DFL ISC B)
double -> float-> long-> int -> short -> char -> byte
int a; float b; double c;
d= a+b*c;
The data type of d is double.

2. Explicit conversion (Type casting)

When the data type gets converted to another data type after the user's intervention.
int a; float b; double c;
d= (float) a+b*c;
The data type of d is float.

Note: Explicit conversion of literals is allowed in Java programming.
float a= 14.67 F;
double b= 14.67 D;
int c= 59;
long f= 59 L;
Operators: Operators are special symbols that cause an action to take place.
Types:



Unary operators:

1. Unary(+): int a=5;
   int b= +a; //5
2. Unary(-): int a=5;
   int b= -a; //-5
   int a=-5; int b=-a ; //5
3. Increment operator(++): (increased by 1)
   a) Post increment a++
   b) Pre increment ++a
4. Decrement operator(--): (decreased by 1)
   a) Post decrement a--
   b) Pre decrement --a

Binary operators: (Two operands and one operator)

1. Arithmetic operators: **+ , - , * , / , %**

a=5, b=2

a+b, a - b, a*b, a/b, a%b

2. Relational operators:(comparing the values)

>, >= ,< , <= , == , !=

a>b, a>=b, a<b, a<=b, a==b, a!=b

3. Logical operators: AND (&&) , OR (| |) , NOT (!)

Ternary operator (or) Conditional operator: (? :)

Syntax: Condition ? Expression 1 : Expression 2 ;

Example;

1) a= 19 , b= 15

max = (a > b) ? a : b ;

2) salary = 15000

bonus = (salary> 10000) ? salary*15/100 : salary* 5/100 ;

Special operators:

new: dynamically allocate the memory for the object.

Example: example e = new example( );

Dot (.) operator: invoking members of class

Example: System.out.println(); (System.in)

Operator precedence:

Operator precedence determines the order in which the operators in an expression are evaluated.

<OPERATOR PRECEDENCE CHART>

Object is an instance of a class. It is an identifiable entity with some characteristics and behavior**.**

Class represents a set of objects that share common characteristics and behavior.

Abstraction : it is the act of representing essential features and hiding the background details.

Encapsulation: it is an act of wrapping of the data and function as one single unit. Provides insulation of the data from the direct access by the program.

Inheritance : it is the capability of one class to inherit the properties from another class

Polymorphism: it is the ability of data to be processed in more than one form

Message passing – when the objects need to interact with one another, they pass /request information to one another. This interaction is known as message passing.

Compiler : It is a software which converts high level language program to machine language and viz. as a whole. It is faster but hard to debug.

Interpreter: It is a software which converts high level language program to machine language and viz line by line. It is slow but easy to debug.

Bytecode : When a java source code is compiled the resultant got is called a bytecode.

Keywords: These are words have a specific meaning to a language compiler.

Tokens: It is the smallest unit of a program

Literals : is a token whose value does not change during the execution of a program.

Comments: Giving remarks for a statement is called as a comment. There are 3 different types of giving a comment.

- // single line comment
- /* ……
    - …….. */ multi line comment
- /** …………….. */ documentation

Jdk – Java development kit.

Constant: a data item that never changes its value during a program runs.

Final: The keyword final makes a variable as constant. i.e., whose value cannot be changed during the program execution

Operators: a symbol used to depict a mathematical or logical operation.

Arithmetic operators: they provide facilitation to the mathematical calculations within a program: they are + - * / %

Assignment operators: These operators are used to assign the value of an expression to a variable : =

Shorthand operators : allows ease while programming instructions and feature with all the available operators with two operands : += -= *= /= %=

Relational operators: provide facilitation for comparing numbers and characters for calculation and do not function with strings.: > < <= >= == !=

Increment operator : is used to increment the variable by 1: ++

Logical operators: used to conduct the logical combination of Boolean values within a program.: && || !

Expression : is a combination of operators, constants and variables.

Type conversion: The process of converting one predefined type into another.

Explicit type conversion: The process of converting one predefined type into another.

It is user defined conversion that forces an expression to be of specific type (also called as **type casting)** e.g. int x= 65; char c = (char) x; it is called type casting.

Implicit type conversion: The process of converting one predefined type into another is called type conversion. It is performed invisible without user intervention and hence known as automatic conversion.

When two operands of different types are encountered in the same expression the lower type variable is converted to the type of the higher type variable automatically and is also known as **type promotion**.:

byte x=8; int y=x; it is called **coercion**

Variables

A variable is defined as a location in the memory of the computer where the values are stored.

Static initialization: it is an expression that initializes a variable during compile time.

E.g., int a=10;

Dynamic initialization: it is an expression that initializes a variable during runtime;

E.g.: int a= c * b;

Package : is a collection of classes. Each package includes related built-in functions.

Statement : A set of instructions which terminate with a colon are called a statement.

E.g., int a =4;

Compound statement / Block statement: Multiline statements are called compound/block statement. They are enclosed in curly brackets { }

E.g., if(a>b)

```
  {
      a=4;
      b=0;
   }
```

Selection statement: These are statements which allows to choose a set of instructions for execution depending upon an expression.

if-else statement tests an expression and depending upon its truth value one of the two sets of action is executed.

Dangling else: In Nested if statement the number of it is more than the number of else.(unmatched if and else)

Switch is a multiple branching statement, this statement tests the value of an expression against a list of integer or character constants for equality.

Default statement gets executed when no match is found in the switch cases.

Fall through: the fall of control to the following cases of matching case (or)

execution of multiple cases after matching takes place in a switch statement.

Iteration statements : executing a set of statements repeatedly until a given condition is met.

Types: for, while and do- while statements.

for is the easiest to understand of the Java loops. All its loop control elements are gathered in one place(on the top of the loop)

While statement is another looping statement and it is entry controlled. The statement will work only if the condition is true.

Do-while: It is another looping statement and it is exit controlled. The statement will work once even if the condition is false.

Jump statements: unconditionally transfer program control within a function.

Break: it is used as a terminator from the enclosed loop and transfers the control to the statement after the loop.(terminate the current loop)

(or)

Break : When break statement is executed it comes out of the inner most loop(if it is a loop statement) and it comes out of a switch-case statement.

continue: Causes the control to transfer from within the block to the next iteration of the loop. (Terminates the current iteration)

POP and OOPs

POP	OOP
Emphasis is given to functions	Emphasis is given to functions and data

= & = =

=	= =
= is an assignment operator, used to assign values to a variable	= = is a relational operator, used to check for equality
e.g., a=4;	if(a = = 5)

Variables & constant

Variable	Constant
A variable is defined as a location in the memory of the computer where the value is stored.	A constant is the value which is stored in a variable

E.g.: a, amount	"Ida" "37 P.S.M. st"

Pre increment and post increment both increment the value by 1.

Pre increment	Post increment
Prefix the variable gets incremented first and then use.	post increment is incremented after the operation of function of its associate operator,
Whereas e.g., a=4; a+=++a; =9	a+=a++; = 8

& and &&

&	&&
& is a bitwise operator (for the manipulation of data at the bit level)	&& is a logical operator(the logical operator evaluates to true value if either of the 2 expressions is true.

Implicit Type conversion & Explicit type conversion

Implicit	Explicit
Is a conversion of one data type into another by the compiler without the user's intervention	It is user defined that forces an expression to be of specific type
Implicit –E.g. float a=4.5f; double x=987.9998; If any operation is done using the above data types then the resultant will be double.	int x=66; char c=(char)x;

Expression and operators

Expression	Operators

Expression is a combination of operators, constants and variables. It can be arithmetic, relational etc. E.g. : u+1/2 at^2	Operators : it is a symbol used to represent logical or arithmetic operations E.g. : + - * / %

Unary and Binary operators

Unary operator	Binary operator
1 operator 1 operand	1 operator 2 operand
E.g. -4	E.g. 2+4

if else- switch case

If else	Switch case
All the different data types can be used	Only char and int type can be used
Does not have a default operation	Has a default operation
Can be tested for logical and relational fn	Only for equality

for and while

For	While
Can be used for only fixed iteration	Can be used when the no of times for a loop to be done is not known
Can use only char or int type	Any data type

do-while and while statement

Do while	While
Exit controlled	Entry controlled
If the condition is false then also the loop will be done once	If the condition is false then the loop will not be done at all

E.g.: int a=1;
While (a>=5)
{
a++;
}
	int a=1;
	do
	{
	a++;
	}while(a>=5);

loop will not be done even once	loop will be done at least once

/ and %

/	%
Returns the quotient	Returns the remainder
E.g.: int c=7/2 answer =3 float c=7/2 answer = 3.5	E.g.: int c=7%2 Answer =1

Compiler and interpreter

Compiler	Interpreter
It is a software which converts high level language into machine language as a whole	It is a software which converts high level language into machine language line by line
It is faster	Slower
Debugging is harder	Debugging is easier

character and string literal

Character literal	String literal
A single character enclosed within single quotes	Zero or more characters enclosed within double quotation.

primitive and user defined datatype

Primitive data type	User defined data type
These are built in data types	Created by user
The sizes of these objects are fixed	The sizes of these data types depend upon their constituent members

These data types are available in all parts of a java program	The availability of these data types depends upon their scope

print() and println()

Print()	Println()
The successive output will come in the same line	The successive output will come in a new line

Break & continue

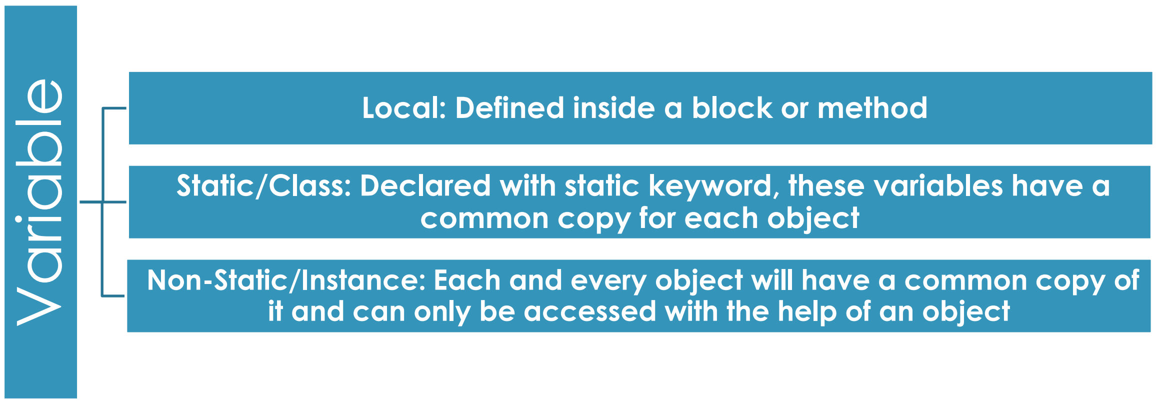
Break	Continue
it serves as a terminator from the enclosed loop,(i.e.) it comes out of an inner loop. And transfers the control to the statement after the loop.	Continue-causes the control to transfer from within the block to the next iteration of the loop.
Break –E.g. While(amount<100) { if(amount = = 11) break; } if amount is 11 then the control will be transferred to the statement following the while loop.	E.g. for(i=0;i<5;i++) { if(a= =0) continue; System.out.println(a*a);

Objects & Classes

Object An identifiable entity with certain characteristics, behavior, and attributes.

Classes A group or collection of similar objects with similar characteristics and behavior. (An instance of a class).

Variable A named memory location to store values.


Variable
Local: Defined inside a block or method
Static/Class: Declared with static keyword, these variables have a common copy for each object
Non-Static/Instance: Each and every object will have a common copy of it and can only be accessed with the help of an object

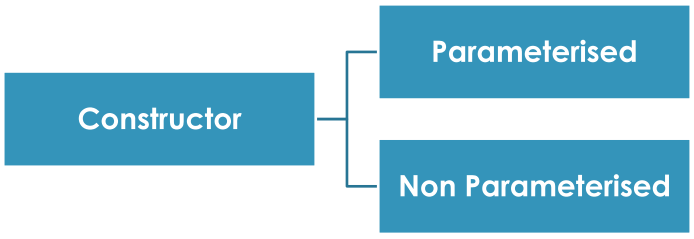
Constructor

Constructor is a special method which is used to initialize instance/non-static variable.

Properties of constructors

- A constructor has the same name as the class
- It does not have a return type, not even void
- It doesn't have a calling statement
- The constructor gets called when an object is created

Types of constructors



Examples:

Parameterized:

```
Int num1, num2, sum;
Sum(int num1, num2){
  num1=num1;
  num2=num2;
}
```

Non-Parameterized:

```
Int num1, num2, sum;
Sum(){
  Sum=0;
}
```

Constructor Overloading When there are more than one constructor in a program, it's called constructor overloading.

"this" keyword "this" is a keyword which refers to the current object.

Example:

```
String flavor; float cost;
Fruitjuice(String flavor, float cost){
  this.flavor=flavor;
  this.cost=cost;
}

```

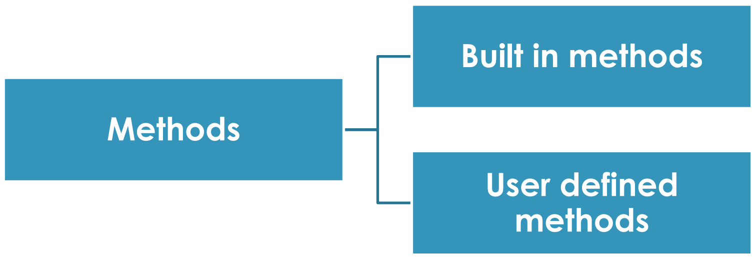
Return types indicate the type of outcome of a method to be returned to its caller. Ex: int, double, float, char, short, byte, Boolean, void

Example:

```
Public int sum(int a, int b){
  c=a+b;
  Return c;
}
```

Function/Method

Function is a set of Java executable statements enclosed in a function definition.



Advantages of methods:

- Reusability
- Manage complex programs into smaller parts
- Hide details

Method definition/prototype refers to the first line of a method which contains the access specifier, modifier, return type, method name and the method signature.

Syntax

```
<access specifier> <modifier> <return type> <method name> (list of
parameters)
{
  <statements>
}
//eg:
public static int(int a, int b){
int c=a+b;
return c;
}
```

Method call syntax:

```
<object name>.<method name>(arguments);
//Example:
a.example();
```

new keyword is used to dynamically allocate memory for the object

Static keyword is a modifier which identifies the method as a class method. It means that the method is not called through an object

Access-specifier: Keywords (except friendly) that control the visibility of a data member

Private: the data members or member methods which are specified as private can be used only within the scope of the class. These members are not accessed outside the class.

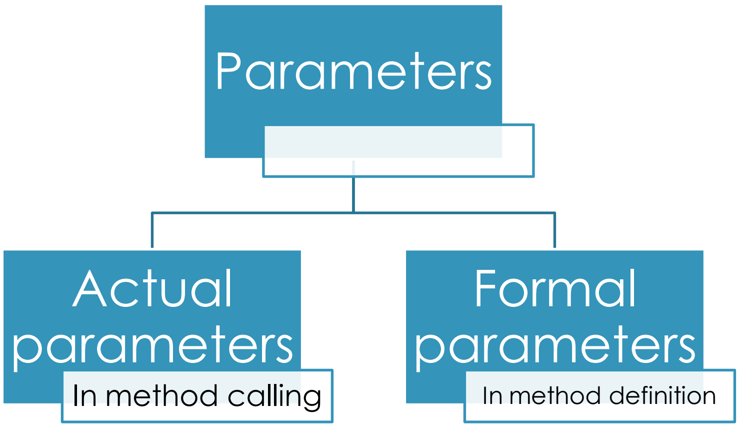
Public: The class members specified as public can be used even outside the visibility of a class

Protected: the protected members are used in the class as private members which can only be applied within the class but can be inherited to another class

Return types indicate the type of outcome of a method to be returned to its caller. Ex: int, double, float, char, short, byte, Boolean, void

Method signature: Collection of data type and variable names written inside a function definition.

Parameters: The value which is passed into the function to instantiate



Actual parameters: The parameters that appear in method calling

Formal parameters: The parameters that appear in method definition

Actual Parameters	Formal Parameters
Appear in method calling	Appear in method definition
Original value	Copied value of actual parameters

Ways of passing values to a function

Pass/call by value:

Any change made in the formal parameters will not reflect in the actual parameters

Pass/call by reference:

Any change made in the formal parameters will reflect in the actual parameters

Call by value	Call by reference

Changes made in the formal parameters will not reflect in the actual parameters	Changes made in the formal parameters will reflect in the actual parameters
Primitive type data are passed to the method using pass by value	Reference type data are passed to the method using pass by value
It is a pure function	It is an impure function

Pure function/accessor method: A method that returns a value but does not change the state of an object

Impure function/mutator: A method that may not return a value but change the state of an object

Recursive Function: A function that calls itself / A function that refers to itself for execution

Method Overloading

Method overloading: Multiple functions sharing the same name with different parameters/ method signature.

Example:

```
void Area(int side){
area=side*side;
}
void Area(int l, int b){
area=l*b; }
```

Arrays

Array: An Array is a set of like variables which are referred to by a common name. A number called subscript/index is used to distinguish one from the other.

Syntax

```
<data type> <array name>[]=new <data type>[<n>];
```

'n' denotes the maximum number of elements that the array can hold.

Assigning values for an Array

Ex:

```
int arr[]={1, 2, 3, 4}
```

.lenght function – Tells the number of elements in an array

Ex:

```
int len=arr.length;
```

Linear search: The search element is checked with all the elements of the array. If the element is found, the flag variable is made to '1' otherwise, it remains '0'.

Linear Search	Binary Search
Can work with both sorted and unsorted arrays.	Works only with sorted arrays.
Reads and compares each element of the array one by one.	Works by dividing the array in two segments out of which only one needs to be searched.

String Handling & Library Classes

In Java, String is an object which contains a sequence of characters. String class is used to create and manipulate strings. The String class is available in java.lang package.

Declaration and Assigning a String

```
//Declaration:
String <variable>;
//Ex:
String str;
//Assigning:
<variable>=<String literal>;
//Ex:
str=”Hello World!”;
```

Input a String

```
//For a string without any space (For a single word):
<variable>=<Scanner object>.next();
Str=sc.next();
//For a String with spaces (For a sentence):
<variable>=<Scanner object>.nextLine();
Str=sc.nextLine();
```

String Functions

For all the below examples, **str=”COMPUTER”; Output** will be displayed as a **single line comment (//)**.

Quick Tip

1. **Function names start with lowercase and then the second word starts with uppercase letters.** Eg: indexOf();
2. **Topics asked in board questions are marked with** ❗

.length() (int)

This function is used to return the **length** of the string.

Syntax with example:

```
<int variable>=<string var>.length();
int Len=str.length();
//8
```

.charAt() (char)

This function returns the **character** from the given index.

Syntax with example:

```
<char variable>=<string var>.charAt(<index>);
char ch=str.charAt(2);
//O
```

.indexOf() (int)

This function returns the **index** of **first occurrence** of a character.

Syntax with example:

```
<int variable>=<string var>.indexOf(<character>);
int idx=str.indexOf(‘M’);
//2
```

.indexOf(char ch, int start_index) (int)

This function returns the **index** of a given **character** from the given **index**.

Syntax with example:

```
<int var>=<String var>.indexOf(<char var>,<int var>);
char ch=’M’;
int ind=str.indexOf(ch, 1);
//2
```

.lastIndexOf(char ch) (int)

This function returns the **index** of the **last occurrence** of a given character.

Syntax with example:

```
<int var>=<String var>.lastIndexOf(char ch);
int ind=str.lastIndexOf('E');
//6
```

.substring(int start_index, int last_index) (String)

This function is used to extract a **set of characters** simultaneously from a given index upto the end of the String or till a given index.

Syntax with example:

```
<String var>=<String var>.substring(<int var>,<int var>);
String ext=str.substring(3);
//PUTER
```

.toLowerCase() (String)

This function is used to convert a given String to **lowercase** letters (entire string).

Syntax with example:

```
<String var>=<String var>.toLowerCase();
String lc=str.toLowerCase();
//computer
```

.toUpperCase() (String)

This function is used to convert a given String to **uppercase** letters (entire string).

Syntax with example:

```
<String var>=<String var>.toUpperCase();
String uc=str.toUpperCase(ind);
//COMPUTER
```

.replace(char old, char new) (String)

This function is used to **replace** a **character or a sequence of characters** in a String with a new character or sequence of characters. (**NOTE**: This does not work with int values)

Syntax with example:

```
<String var>=<String var>.replace(<char var>,<char var>);
String rep=str.replace(“PUTER”,”PUTE”);
//COMPUTE
```

.concat(String second) (String)

This function is used to **concatenate/join** two Strings together. (**NOTE**: This does not add any spaces in-between)

Syntax with example:

```
<String var>=<String var>.concat(s);
String s=”STUDENT”;
String con=str.(s);
//COMPUTERSTUDENT
```

.equals(String srt) (boolean)

This function is used to check for **equality** between two Strings. (**NOTE**: This function returns a **boolean** value. This function cannot be used for characters. //You can simply use == for characters. This can be used in if statements)

Syntax with example:

```
<boolean var>=<String var>.equals(<String var>);
String s=”COMPUTER”;
boolean chk=str.equals(s);
//true
```

. equalsIgnoreCase(String str) (boolean)

This function does the same function of .equals() function. The only difference is that it does not care about the case (It **ignores the case**).

Syntax with example:

```
<boolean var>=<String var>.equalsIgnoreCase(<String var>);
boolean chk=str.equalsIgnoreCase(“cOmPuTeR”); //true
```

.compareTo(String str) (int)

This function is used to **compare** two Strings. It also checks whether a String is **bigger or smaller** than the other and returns a suitable **int value**. It returns **0** if both are **equal**. A **positive** value when the **first is bigger** than the second and a **negative** value when the **second String is bigger** than the first. It returns the **no. of additional characters** when both the Strings' **first sequence of characters are equal** but the other has additional characters.

Syntax with example:

```
<int var>=<String var>.compareTo(<String var>);
String s=”SCIENCE”;
int cmp=str.compareTo(s);
//A, B, C, (C is the 3rd letter in the Alphabet and S is the 19th)
//the value of cmp will be-16 because (3-19=-16)
```

.compareToIgnoreCase(String str) (int)

This function does the same function as .compareTo but it **ignores the case**.

Syntax with example:

```
<int var>=<String var>.compareToIgnoreCase(<String var>);
int cmp=str.compareToIgnoreCase(“cOmPuTeR”);
//0
```

.trim() (String)

This function removes **spaces** at the **start and end** of the String. (**NOTE:** This function does not remove spaces in-between characters)

Syntax with example:

```
<String var>=<String var>.trim();
Str=”    He llo World!  “;
String trm=str.trim();
//He llo World!
```

.startsWith(String str) (boolean)

This function is used to check if the given String is a **prefix** to the other.

Syntax with example:

```
<boolean var>=<String var>.startsWith(<String var>);
pfx=”COM”
boolean chk=str.startsWith(pfx);
//true
```

.endsWith(String str) (boolean)

This function is used to check if a given String has a specified **suffix**.

Syntax with example:

```
<boolean var>=<String var>.ends with(<String var>);
boolean chk=str.endsWith(“TER”);
```

.equals()	.compareTo()
Returns a Boolean value	Returns a an int value
It checks for equality between two Strings	It checks if a String is equal, bigger or smaller than the other.

Difference Between equals() and compareTo() functions

Library Classes & Wrapper Classes

For better understanding:

Before we get into Library Classes & Wrapper Classes, it’s important to know what is a primitive and composite data types.

Primitive Data Type: These are **fundamental built-in data types** of **fixed sizes**. **Ex:** int, long, float

- **Composite/Reference/User-Defined Data Type:** These are data types **created by the user**. The availability of these data types depends upon their **scope and sizes** depend upon their **constituent members**. **Ex:** array, class, object

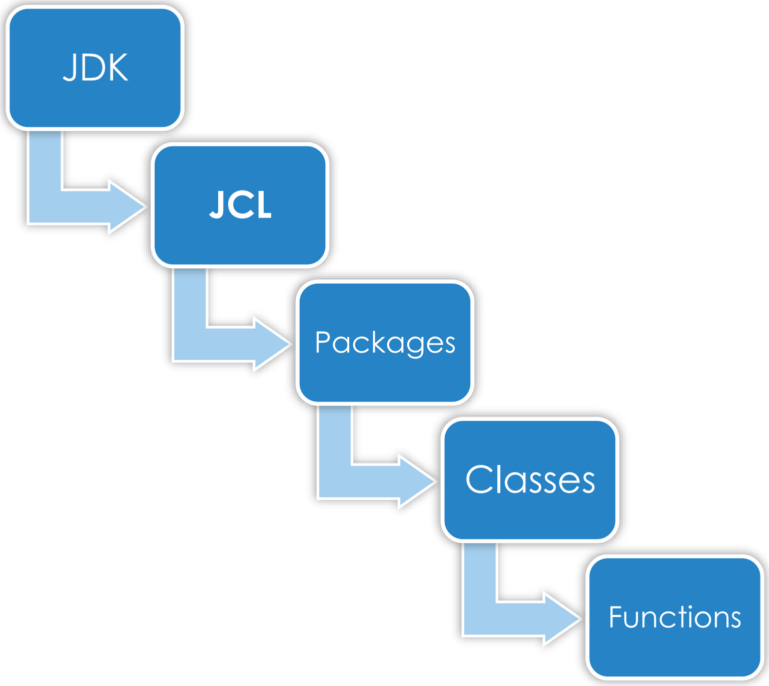
Primitive data type	Composite data type
These are built in data types	Created by user

The sizes of these objects are fixed	The sizes of these data types depend upon their constituent members
These data types are available in all parts of a java program	The availability of these data types depends upon their scope

Difference between primitive and composite data type.

Library Classes

JDK (Java Development Kit) V1.5 and above contains Java Class Library (JCL) which contains various packages. Each package contains various classes containing different built-in functions.



Ex: *java.lang, java.math*

Wrapper Class

Wrapper Classes are a part of java.lang (A Library Class Package). Wrapper classes **contain primitive data values in terms of objects**/ Wrapper Class **wraps a primitive data type to an object**. There are 8 wrapper classes in Java. **Ex:** Integer, Byte, Double

(**NOTE:** Wrapper Classes always start with an uppercase letter
Ex: Integer, Boolean, Float)

Need for Wrapper Classes

- To store primitive values in the objects
- To convert a string data into other primitive types and vice-versa

Wrapper Class	Primitive Type
Byte	Byte
Short	short
Integer	int
Long	long
Float	float
Double	double
Character	char
Boolean	boolean

Wrapper Classes and their primitive types

Functions/Methods of Wrapper Classes

Conversion from String to Primitive types

For converting String to any primitive data type, Wrapper Class functions can be used. For any primitive data Wrapper Class, the **parse**<prm data type>(<String arg>) (or) **valueOf**(<String arg>) functions can be used.

Eg: int i=Integer.parseInt(s); int j=Integer.valueOf(s);

For better understanding:

```
<prm data type var>=<prm data type Wrapper Class>.parse<prm data
type name>(<String arg>);
<prm data type var>=<prm data type wrapper class>.valueOf(<String
arg>);

//Examples:
int a=Integer.parseInt("238");
doubleb=Double.parseDouble("23.45");
int c=Integer.valueOf("37");
float d=Float.valueOf("42.87");
```

Examples of each <> (In the above syntax):

prm data type: int a | double b
prm data wrapper class: Integer | Double
prm data type name: Int | Long | Double
String arg: "38.743" | "1874293856"

Conversion from primitive type data to String

For converting a primitive type data to a String, the **toString()** Wrapper Class function can be used.

Ex: Integer.toString() | Double.toString()

```
<String var>=<Wrapper Class>.toString(<prm data arg>);
String cnv=Integer.toString(38);
String dbl=Double.toString(94.53);
```

Boxing, Unboxing & Autoboxing

Boxing

Conversion of primitive type data **to an object**.

Syntax with example:

```
<wrapper class> <object name>=new <wrapper class>(<prm type arg>);
int a=239;
Integer x=new Integer(a);
```

Unboxing

Conversion of an object **to primitive type data**.

Syntax with example:

```
<int var>=<wrapper class obj>
int b=x;
```

Autoboxing

Boxing is the mechanism and autoboxing is the feature of the compiler which generates the boxing code.

Syntax with example:

```
<wrapper class> <object name>=new <wrapper class>(<prm type arg>);
int a=239;
Integer x=new Integer(a);
```

Character

Character is defined as a letter, a digit or any special symbol/UNICODE enclosed within single quotes. **Ex:** '@', 's', '5'

Assigning a character

A Character is declared under char data type.

Syntax with example:

```
char <var name>='<char literal>';
char ch='a';
```

Input a character

A Character is declared under char data type.

Syntax with example:

```
<char var>=<Scanner obj>.next().charAt(0);
ch=sc.next().charAt(0);
```

Character Functions

Character.isLetter() (boolean)

This function is used to check if a given argument is a letter or not.

Syntax with example:

```
<boolean var>=Character.isLetter(<char arg>);
boolean chk=Character.is('A'); //true
```

Character.isDigit() (boolean)

This function is used to check if a given argument is a digit or not.

Syntax with example:

```
<boolean var>=Character.isDigit(<char arg>);
boolean chk=Character.is('7'); //true
```

Character.isLetterOrDigit() (boolean)

This function is used to check if a given argument is either a letter or a digit or none of these.

Syntax with example:

```
<boolean var>=Character.is(<char arg>);
boolean chk=Character.is(‘A’); //true
```

Character.isWhitespace() (boolean)

This function is used to check if a given argument is a blank/gap/space or not.

Syntax with example:

```
<boolean var>=Character.is(<char arg>);
boolean chk=Character.is(‘A’); //false
```

Character.isUpperCase() (boolean)

This function is used to check if a given argument is an uppercase letter or not.

Syntax with example:

```
<boolean var>=Character.is(<char arg>);
boolean chk=Character.is(‘A’); //true
```

Character.isLowerCase() (boolean)

This function is used to check if a given argument is a or not.

Syntax with example:

```
<boolean var>=Character.is(<char arg>);
boolean chk=Character.is(‘A’); //false
```

Character.toUpperCase() (char)

This function is used to convert/returns a given argument/character/letter to/in uppercase character/letter.

Syntax with example:

```
<char var>=Character.toUpperCase(<char arg>);
char uc=Character.toUpperCase(‘a’); //A
```

Character.toLowerCase() (char)

This function is used to convert/returns a given argument/character/letter to/in lowercase character/letter.

Syntax with example:

```
<char var>=Character.toLowerCase(<char arg>);
char lc=Character.toLowerCase(‘A’); //a
```

Revision/Review Questions

Revision of Class IX Syllabus

1. What is a token? Give examples
2. What are non-graphic characters? Explain their usage in Java programming language.
3. Name any two OOPs principles. **[ICSE 2015]**
4. What is a class?
5. Define encapsulation. **[ICSE 2016]**
6. What is inheritance? **[ICSE 2017]**
7. What are keywords? Give an example. **[ICSE 2016]**
8. Define byte code. **[ICSE 2010]**
9. Name any two types of programs. **[ICSE 2007]**
10. What is a literal? **[ICSE 2013]**
11. Rewrite the following using ternary operator **[ICSE 2018]**

```java
if(bill>10000)
dis=bill*10.0/100;
else
dis=bill*5.0/100;
```

12. Differentiate between
    - if-else & switch **[ICSE 2014]**
    - = , == **[ICSE 2007]**
    - while & do-while **[ICSE 2018]**
    - break & continue **[ICSE 2013]**
    - class & object **[ICSE 2011]**
    - Primitive & Non-Primitive data types **[ICSE 2016]**
13. Name any two access specifiers. **[ICSE 2016]**

14. Why is a class called a factory of objects? **[ICSE 2009]**
    (Ans.) A class contains all the statements needed to create an object, as well as statements to describe the operations that the object will be able to perform.
15. Name the primitive data type in Java that is **[ICSE 2014]**
    i) A 64-bit integer and is used when you need a range of values wider than those provided by int
    (or)
    What is the size of long data type?
    ii) A single 16-bit Unicode character whose default value is '\u0000' (**Ans.** char)

Objects and Classes, Constructor, Functions, Arrays

1. Define
   - Class
   - Object
   - Function
   - Function prototype
   - Array
2. Differentiate between
   - Actual & Formal parameters
   - Call by value & call by reference
   - Pure & Impure functions
   - Static & Non-Static variables
   - Linear & Binary search
3. What is the role of the keyword void in declaring functions? **[ICSE 2007]**
4. What is the OOP principle implements function overloading? **[ICSE 2007]**
5. Write the prototype of a function check() which takes an integer value as an argument and returns a character.

[ICSE 2018]

String Handling & Library Classes

MCQ

1. What does compareTo() return? **[ICSE 2014]**

a) boolean
b) double
c) int
d) A detailed list of analysis of the values of the Strings

2. What does equals() return? **[ICSE 2014]**
   a) int
   b) float
   c) String
   d) boolean
3. Name a function that removes the blank spaces at the start and at the end of the String. **[ICSE 2015]**
   a) removeSpace()
   b) trim()
   c) delSpace()
   d) Dear Examiner, there is no such function which does that. -Candidate
4. What is the value of ind if ind=str.indexOf('a'); while str="Tata, bye bye!";
   a) 2
   b) 2,4
   c) 1
   d) ERROR. Please check the question!

Answer the following

1. Name any two wrapper classes **[ICSE 2013]**
2. What is the return type of the following library functions
   e) isWhiteSpace() b) Math.random **[ICSE 2013]**
3. What are library classes? Give an example. **[ICSE 2011]**
4. Why is class known as composite data type? **[ICSE 2009]**
5. A method that converts a String to an Integer primitive data type **[ICSE 2009]**

Programs (Includes questions out of board papers)

1. Write a program to replace a particular index with a given character in a String.
   Ex:
   input: FrozenNotes | 5 | d (3 separate inputs)
   output: FrozedNotes
2. Write a program to display a word in reverse.
3. Write a program to delete alt letters in a word.
   Ex:
   Input: FrozenNotes
   Output: F o e n o e s

Answer Key 🗝

<TO BE UPDATED>