Section 1- Basic Java

1. Variables & Data types
2. Keywords & Identifiers
3. Methods
4. Constructor
5. Control Statements
6. Loops

1. Variables & Data Types

Variables:-

- Variables are nothing but piece of memory use to store information. One variable can store 1 information at a time.
- Variables also used in information reusability.
- To utilize variables in java programing language we need to follow below steps:
    1. Variable declaration (Allocating/Reserving memory)
    2. Variable Initialization (Assigning or inserting value)
    3. Usage

Note: -

- According to all programming language dealing with information directly is not a good practice to overcome this variables are introduced.

Data Types:-

- Data type are used to represent type of data or information which we are going to use in our java program.
- In java programming it is mandatory to declare data type before declaration of variable.
- In java data types are classified into two types :

    1. Primitive data type.

    2. Non-primitive data type.

1. Primitive data type:-

- There are 8 type of primitive data types.
- All the primitive data types are keywords.
- Memory size of primitive data type are fix.
- The types of primitive data type are:

Note: - keyword starts with lower case

syntax: datatype variablename;

1. **(Numeric + Non-decimal):-**

Ex: 80,85,10,..etc

	Data Type	Size	Description
1.	byte	1 byte	Stores whole numbers from -128 to 127
2.	short	2 bytes	Stores whole numbers from -32,768 to 32,767
3.	int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
4.	long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807

2. **(Numeric + decimal):-**

Ex: 22.5,22.8,6.4....

5.	float	4 byte
6.	double	8 byte

3. **Character:-**

Ex: A,B,X,Z.

7.	char	2 byte

4. **Conditional:-**

Ex: true,false.

8. boolean 1 bit

2. Non-primitive datatype:-

- There are 2 types of non primitive datatypes .
- Non primitive datatypes are identifiers.
- Memory size of non primitive datatype is not defined.

Note: - Identifier starts with capital letter.

e.g. String, class

2. Methods:-

- A method is a block of code which only runs when it is called.
- You can pass data, known as parameters, into a method.
- Methods are used to perform certain actions, and they are also known as functions.
- Why use methods? To reuse code: define the code once, and use it many times.

1. Main method

- In any Java program, the main() method is the starting point from where compiler starts program execution.
- So, the compiler needs to call the main() method.
- Without main method we can't run any java program.

2. Regular method

1. static regular method

1. static method call from same class

   //methodname();

2. static method call from diffrent/another class

   //classname.methodname();

2. non- static regular method

3. non-static method call from same class

```
//classname objectname/variablename=new classname();
//objectname.methodname();
//sample2-->classname-->datatype
//s4-->objectname/variablename
//new-->operator/keyword-->to create blank object/instance
//sample2()-->classname()-->constructor-->to copy or load non static
   non static member in to object
```

4. non-static method call from diffrent/another class

```
//classname objectname/variablename = new classname();
//objectname.methodname();
```

Note:-

- At the time of program execution main method is going to get executed automatically, whereas regular methods are not going to get executed automatically.
- At the time of program execution priority is scheduled for main method only.
- To call a regular method we need to make call method call from main method, until unless if the method call is not made regular method will not get executed.
- Regular methods can be called multiple times.

5. method without/zero parameter

```
//methodname();
```

6. method with parameter.

```
//classname objectname/variablename = new classname();
//objectname.methodname();
```

7. method with return type

3. Java Keywords & Identifiers

Java Keywords:-

- Keywords are predefined, reserved words used in Java programming that have special meanings to the compiler. For example:

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

Java identifiers:-

- Identifiers are the name given to variables, classes, methods, etc.
- Rules for Naming an Identifier
  1. Identifiers cannot be a keyword.
  2. Identifiers are case-sensitive.
  3. It can have a sequence of letters and digits. However, it must begin with a letter, $ or _. The first letter of an identifier cannot be a digit.
  4. It's a convention to start an identifier with a letter rather and $ or _.
  5. Whitespaces are not allowed. Similarly, you cannot use symbols such as @, #, and so on.

4. Control Statements-

1. **if**
2. **if else**
3. **else if**
4. **nested if**
5. **switch**

1. **if**

```
if(conditon)
{
 }
```

2. **if else**

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

3. **else if**

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

4. **nested if**

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

5. **Switch**

```
 switch (key)
{
        case value: break;
        default:break;
  }
```

5. loops

1. **for loop**
2. **while**
3. **do while**
1. **for loop**

Syntax for for loop-

```
for (int i = 0; i < args.length; i++)
 {

 }
```

2. **while**

Syntax for while loop-

```
while (condition)
{
        Syso (“   ”);
        i++;
}
```

3. **do while**

Syntax of do while loop

```
do
{
        Syso (“   ”);
                i++;
}
while (condition);
```

6. Constructor

- A constructor in Java is a special method that is used to initialize objects/variables.
- The constructor is called when an object of a class is created.
- At the time of constructor declaration below points need to follow:
    1. Constructor name should be same as class name
    2. You should not declare any return type for the constructor (like void).
    3. any no of constructor can be declared in a java class but constructor name should be same as class name,but arguments/parameter should be different.

- Use of Constructor
    1. To initialize data member/variable
    2. To copy/load non-static members of class into object --> when we create object of class
- Types of Constructor
    1. Default Constructor
    2. User defined Constructor

1. Default Constructor

- If Constructor is not declared in java class then at the time of compilation compiler will provide Constructor for the class
- If programmer has declared the constructor in the class then compiler will not provide default Constructor.
- The Constructor provided by compiler at the time of compilation is known as Default Constructor

2. User defined Constructor

- If programmer is declaring constructor in java class then it is considered to be as User defined constructor.

User defined Constructor are classified into 2 types

1. Without/zero parameter constructor
2. With parameter constructor

Section-2 OOPs (Object Oriented Programming System/Structure)

1. Inheritance
2. Types of Variable
3. Polymorphism
4. Access Specifies
5. Abstract class & Concrete class
6. Interface & Implementation class
7. Generalization
8. Casting
9. **Encapsulation**

1. Inheritance

- It is one of the Oops principle where one class **acquires properties** of another class with the help of '**extends**' keywords is called Inheritance.
- The class from where **properties** are **acquiring/**inheriting is called **super class.**
- The class too where **properties** are inherited/**delivered** is called **sub class.**
- Inheritance **takes place between** 2 or more than 2 classes.
- Inheritance is classified into 4 types:

    1. Singlelevel Inheritance
    2. Multilevel Inheritance
    3. Multiple Inheritance
    4. hirarchicle

1. Singlelevel Inheritance:

- It is an operation where inheritance **takes place between 2 classes**.
- To perform singlelevel inheritance **only 2 classes are mandatory**.

2. Multilevel Inheritance:

- Multilevel Inheritance **takes place between 3 or more than 3 classes.**
- In Multilevel Inheritance 1 sub class acquires properties of another super class & that class acquires properties of another super class & phenomenon continuous.

3. Multiple Inheritance:

- **1 subclass** acquiring properties of **2 super classes** at the **same time** is known as Multiple Inheritance.
- **Java doesn't support** Multiple Inheritance using class because of **diamond ambiguity problem.**
- By using **interface** we can **achieve Multiple Inheritance**.

Note: object class is the super most class in java

4. Hirarchicle Inheritance:

- **Multiple sub classes** can acquire properties of **1 super class is known as hirarchicle Inheritance.**

2. Types of variable:

1. local variable

- Creating **variable inside method/block** is known as local variable.
- Scope of local variable **remains only within the method** & they are **temporary**.

2. global variable

- Creating variable **outside method/block** is known as global variable.
- Scope of global variable **remains throught the class** & they are **permanent**.

3. class/static variable

- **Declaring** the variable using **static keyword** is known as class/static variable because to access to static variable class name is used.
- To access static variable from diff class we need to make use of below statement:

```
classname.variablename;
```

4. Instance/non-static variable

- All the non-static variables are known as instance variable because to access non-static variable instance (object) need to be created.
- To access non-static variable we need to make use of below statement:

  -Create object of class.

  -objectname.variablename;

this keyword-

- this keyword is use to access global variable from the same class

super keyword-

- this keyword is use to access global variable from the super/different class

3. Access specifiers:

- Access specifiers are use to represent scope of members of class.
- In java Access specifiers are classified into 4 types

  1. private
  2. default
  3. protected
  4. public

1. private:

- If you declare any member of class as private then scope of that member remains only within the class.
- It can't be access from other classes.

2. default:

- If you declare any member of class as default then scope of that member remains only within the package.
- It can't be access from other packages.
- There is no keyword to represent default access specifiers.

3. protected:

- If you declare any member of class as protected then scope of that member remains only within the package that class which is present outside the package can access it by one condition ie. inheritance operation

4. public:

- If you declare any member of class as public then scope of that member remains throught the project.

4. Polymorphism:

- It is one of the OOPs principle where **one object showing different behavior** at different stages of life cycle.
- Polymorphism is an latin word where **poly** stand for **many** & **morphism** stands for **forms**.
- In java Polymorphism is classified into 2 types:

    1. Compiletime Polymorphism
    2. Runtime Polymorphism

1. Compiletime Polymorphism:

- In Compiletime Polymorphism **method declaration** is going to get **binded** to its **defination** at compilation time, **based on argument** is known as compiletime Polymorphism.
- As binding takes **during compilation time only**, so it is also known as **early binding**.
- Once binding is done, **again rebinding can't be done**, so it is called **static binding.**
- **Method overloading** is an example of compiletime Polymorphism.

2. Runtime Polymorphism:

- In Runtime Polymorphism **method declaration** is going to get **binded** to its **defination** at Runtime, **based on object creation** is known as runtime Polymorphism.
- As binding takes **during Runtime time**, so it is also known as **late binding**.
- Once binding is done, **again rebinding can be done**, so it is called **dynamic binding**.
- **Method overriding** is an example of Runtime Polymorphism.

Method overloading:

- **Declaring** multiple method with same method name **but with diffrent argument** in a **same class** is called method overloading

Method overriding:

- **Acquiring** super class method into sub class with the **help of extends keyword** & **changing implementaion/definition** according to **subclass specification** is called method overriding.

5. Abstract and Concrete class:-

Abstract Class:

- A class declared with **"abstract"** keyword is called **abstract class**.
- An Abstract class is nothing but an **incomplete class** where **programmer can declare** complete as well as incomplete methods in it.
- Programmer can declare **incomplete methods as abstract method**, by declaring keyword called **"abstract"** in front of method**.**
- We **can't create object of abstract class**, to **create object of abstract class** we **need** to make **use** of **concrete class**.

Concrete class:

- A class which **provides definations** for all **the incomplete method**s which **are present in abstract class** with the **help of extends** keywords is called **concrete class**.

6. Interface & Implementation class

Interface:

- It is one of the OOPs principle.
- It is pure **100% abstract** in nature.
- Interface is use to **declare** only **incomplete methods** in it.

Features of Interface:

1. **D.M** (data member)/**variable** declared inside Interface are by **default static and final**.
2. **Methods declared** inside Interface are by **default public & abstract**.
3. **Constructor** concept in **not present** inside Interface.
4. **Object** of Interface **can't be created**.
5. Interface **support multiple inheritance**.
6. To **create object** of Interface programmer **need** to make use of **Implementation** class.

Implementation class:

A class which **provides definations** for all the **incomplete methods** which are **present in interface** with **the help of implements** keyword is called **Implementation class**.

7. Casting:

- Converting **one type** of **information** into **another type** is called casting
- In java casting is classified into 2 types:
    1. Primitive casting
    2. Non-primitive casting

1. Primitive-casting:

- Converting **one data type** of information into **another data type** is called casting
- Primitive-casting is classified into 3 types:
    1. implicit casting
    2. explicit casting
    3. boolean casting

1. implicit casting:

- Converting **lower data type** info into **higher data type** info is called implicit casting.
- implicit casting is also called **widning casting**, where **memory size** goes on **increasing**,

eg.

```
int a=5     // (memory size of int is 4 byte)
sop(a)      // 5
double b = a    //(memory size of double is 8 byte)
sop(b)         //5.0
```

2. explicit casting:

- Converting **higher data type** info into **lower data type** info is called explicit casting.
- explicit casting is also called **narrowing casting**, where memory size goes on decresing.

  In explicit casting data loss takes place

  eg.

```
double b = 2.5    //(memory size of double is 8 byte)
sop(b)         //2.5
int a= (int)b    // (memory size of int 4 byte)
sop(a)          //2
```

3. boolean casting:

- boolean casting is considered to be **incompatible casting type**, because boolean data type is **unique type** of data type where **information** is already **predeclared inside** it.

  boolean str=true

2. Non-primitive-casting

- Converting **one type of class** into **another type of class** is called non-primitive.

  Non-primitive is classified into 2 types:

  * 1. up casting
    2. down casting

1. up casting:

- **Assigning** subclass **property** into **superclass** is called up casting.
- **Before performing** upcasting 1st we **need to perform inheritance** operation.
- After performing inheritance, the **property** which are **present inside** superclass **comes into** subclass.
- In the subclass **programmer** can **declare new properties**.

- At the time of upcasting the properties which are **inherited from superclass** are **only eligible** for the upcasting operation.
- The **new property** which were **declared inside subclass** are **not eligible** for upcasting operation.

2. down casting:

- **Assigning superclass** property into **subclass** is called downcasting.
- **Before** performing **downcasting** 1st we **need to perform upcasting.**

8. Generalization:

- Generalization is the process of **extracting shared characteristics** from two or more classes, and **combining** them into a **generalized superclass**.
- Generalization file can be normal java class or abstract class or Interface, but only **Interface is recommended**.

9. Abstraction:

- Abstraction is one of the OOPs principle in java.
- **Hiding** the implementation code and **providing** only functionality to the **end user** is called abstraction.
- The scenario of Abstraction is "if customer is visiting or making use of any application, then he should utilize functionality only & he should not feel any backend code processing**"**