

OOps Concepts:

As the name suggests, Object-Oriented Programming or OOPs refers to languages that use objects in programming. Object-oriented programming aims to implement real-world entities like inheritance, hiding, polymorphism, etc in programming. The main aim of OOP is to bind together the data and the functions that operate on them so that no other part of the code can access this data except that function.

OOps Concepts:

- Class
- Objects
- Data Abstraction
- Encapsulation
- Inheritance
- Polymorphism
- Dynamic Binding
- Message Passing

1. Class:

A class is a user-defined data type. It consists of data members and member functions, which can be accessed and used by creating an instance of that class. It represents the set of properties or methods that are common to all objects of one type. A class is like a blueprint for an object.

For Example: Consider the Class of Cars. There may be many cars with different names and brands but all of them will share some common properties like all of them will have 4 wheels, Speed Limit, Mileage range, etc. So here, Car is the class, and wheels, speed limits, mileage are their properties.

2. Object:

It is a basic unit of Object-Oriented Programming and represents the real-life entities. An Object is an instance of a Class. When a class is defined, no memory is allocated but when it is instantiated (i.e. an object is created) memory is allocated. An object has an identity, state, and behavior. Each object contains data and code to manipulate the data. Objects can interact without having to know details of each other's data or code, it is sufficient to know the type of message accepted and type of response returned by the objects.

3. Data Abstraction:

Data abstraction is one of the most essential and important features of object-oriented programming. Data abstraction refers to providing only essential information about the data to the outside world, hiding the background details or implementation. Consider a real-life example of a man driving a car. The man only knows that pressing the accelerators will increase the speed of the car or applying brakes will stop the car, but he does not know about how on pressing the accelerator the speed is increasing, he does not know about the inner mechanism of the car or the implementation of the accelerator, brakes, etc in the car. This is what abstraction is.

4. Encapsulation:

Encapsulation is defined as the wrapping up of data under a single unit. It is the mechanism that binds together code and the data it manipulates. In Encapsulation, **the variables or data of a class are hidden from any other class** and can be accessed only through any member function of their class in which they are declared. As in encapsulation, the data in a class is hidden from other classes, so it is also known as data-hiding.

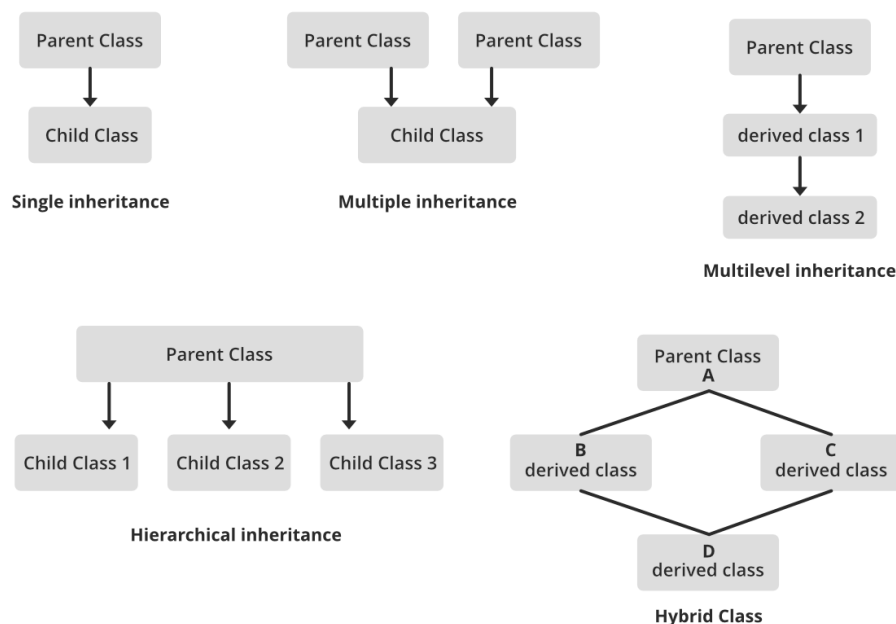
Encapsulation



Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section, etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps records of all the sales. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of the sales section and the employees that can manipulate them are wrapped under a single name “sales section”.

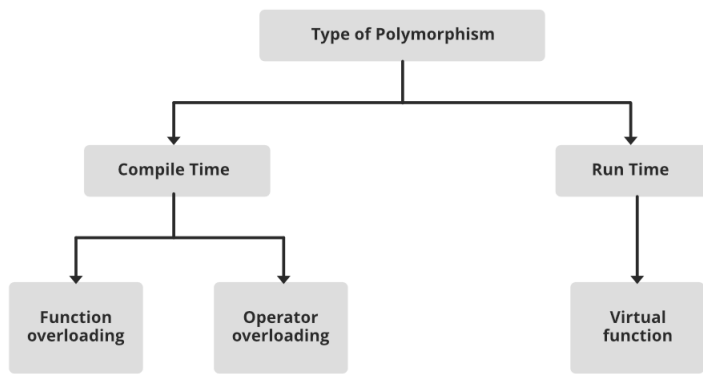
5. Inheritance:

Inheritance is an important pillar of OOP(Object-Oriented Programming). The capability of a class to derive properties and characteristics from another class is called Inheritance. When we write a class, we inherit properties from other classes. So when we create a class, we do not need to write all the properties and functions again and again, as these can be inherited from another class that possesses it. Inheritance allows the user to reuse the code whenever possible and reduce its redundancy.



6. Polymorphism:

The word polymorphism means having many forms. In simple words, we can define polymorphism as the ability of a message to be displayed in more than one form. For example, A person at the same time can have different characteristics. Like a man at the same time is a father, a husband, an employee. So the same person possesses different behavior in different situations. This is called polymorphism.



7. Dynamic Binding:

In dynamic binding, the code to be executed in response to the function call is decided at runtime. Dynamic binding means that the code associated with a given procedure call is not known until the time of the call at run time. Dynamic Method Binding One of the main advantages of inheritance is that some derived class D has all the members of its base class B. Once D is not hiding any of the public members of B, then an object of D can represent B in any context where a B could be used. This feature is known as subtype polymorphism.

8. Message Passing:

It is a form of communication used in object-oriented programming as well as parallel programming. Objects communicate with one another by sending and receiving information to each other. A message for an object is a request for execution of a procedure and therefore will invoke a function in the receiving object that generates the desired results. Message passing involves specifying the name of the object, the name of the function, and the information to be sent.

Classes and Objects

Classes and Objects are basic concepts of Object Oriented Programming which revolve around the real life entities.

Class

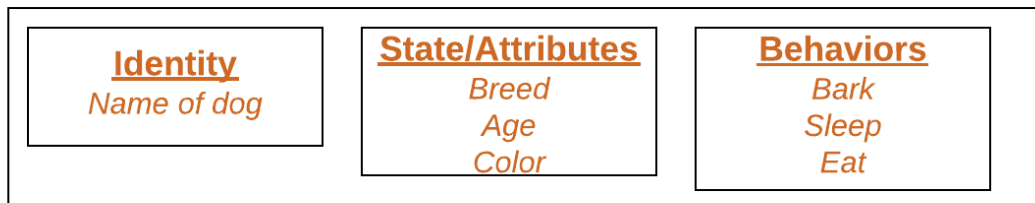
A class is a user defined blueprint or prototype from which objects are created. It represents the set of properties or methods that are common to all objects of one type. In general, class declarations can include these components, in order:

1. **Modifiers:** A class can be public or has default access
2. **class keyword:** class keyword is used to create a class.
3. **Class name:** The name should begin with an initial letter (capitalized by convention).
4. **Superclass(if any):** The name of the class's parent (superclass), if any, preceded by the keyword extends. A class can only extend (subclass) one parent.
5. **Interfaces(if any):** A comma-separated list of interfaces implemented by the class, if any, preceded by the keyword implements. A class can implement more than one interface.
6. **Body:** The class body surrounded by braces, { }.

Object

It is a basic unit of Object-Oriented Programming and represents the real life entities. A typical Java program creates many objects, which as you know, interact by invoking methods. An object consists of :

1. **State:** It is represented by attributes of an object. It also reflects the properties of an object.
 2. **Behavior:** It is represented by methods of an object. It also reflects the response of an object with other objects.
 3. **Identity:** It gives a unique name to an object and enables one object to interact with other objects.
- Example of an object: dog

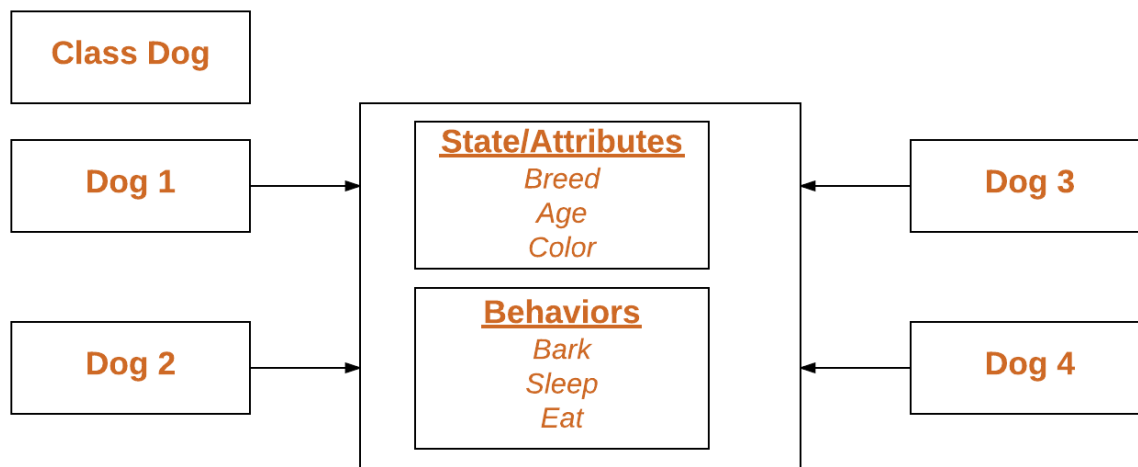


Objects correspond to things found in the real world. For example, a graphics program may have objects such as “circle”, “square”, “menu”. An online shopping system might have objects such as “shopping cart”, “customer”, and “product”.

Declaring Objects (Also called instantiating a class)

When an object of a class is created, the class is said to be **instantiated**. All the instances share the attributes and the behavior of the class. But the values of those attributes, i.e. the state are unique for each object. A single class may have any number of instances.

Example:



As we declare variables like (type name;). This notifies the compiler that we will use name to refer to data whose type is type. With a primitive variable, this declaration also reserves the proper amount of memory for the variable. So for reference variable, type must be strictly a concrete class name. In general, we **can't** create objects of an abstract class or an interface.

Dog **tuffy**;

If we declare reference variable(**tuffy**) like this, its value will be undetermined(null) until an object is actually created and assigned to it. Simply declaring a reference variable does not create an object.

Initializing an object

The new operator instantiates a class by allocating memory for a new object and returning a reference to that memory. The new operator also invokes the class constructor.

- Java

// Class Declaration

```
public class Dog
{
    // Instance Variables
    String name;
    String breed;
    int age;
    String color;

    // Constructor Declaration of Class
    public Dog(String name, String breed,
               int age, String color)
    {
        this.name = name;
        this.breed = breed;
        this.age = age;
        this.color = color;
    }

    // method 1
    public String getName()
    {
        return name;
    }

    // method 2
    public String getBreed()
    {
        return breed;
    }

    // method 3
    public int getAge()
    {
        return age;
    }

    // method 4
    public String getColor()
    {
        return color;
    }

    @Override
    public String toString()
    {
```

```

        return("Hi my name is "+ this.getName()+
            "\nMy breed,age and color are " +
            this.getBreed()+"," + this.getAge()+
            ","+ this.getColor());
    }

    public static void main(String[] args)
    {
        Dog tuffy = new Dog("tuffy","papillon", 5, "white");
        System.out.println(tuffy.toString());
    }
}

```

Output:

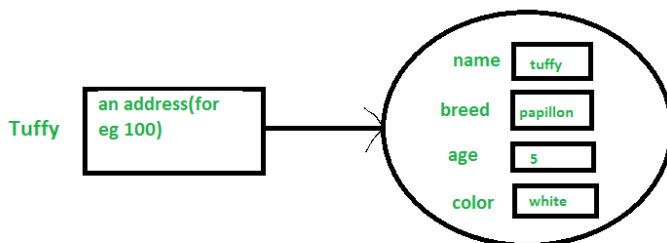
Hi my name is tuffy.

My breed,age and color are papillon,5,white

- This class contains a single constructor. We can recognize a constructor because its declaration uses the same name as the class and it has no return type. The Java compiler differentiates the constructors based on the number and the type of the arguments. The constructor in the *Dog* class takes four arguments. The following statement provides “tuffy”, ”papillon”,5,”white” as values for those arguments:

```
Dog tuffy = new Dog("tuffy","papillon",5, "white");
```

- The result of executing this statement can be illustrated as :



Note : All classes have at least **one** constructor. If a class does not explicitly declare any, the Java compiler automatically provides a no-argument constructor, also called the default constructor. This default constructor calls the class parent’s no-argument constructor (as it contain only one statement i.e `super();`), or the *Object* class constructor if the class has no other parent (as Object class is parent of all classes either directly or indirectly).

Ways to create object of a class

There are four ways to create objects in java. Strictly speaking there is only one way (by using *new* keyword), and the rest internally use *new* keyword.

- Using new keyword:** It is the most common and general way to create object in java. Example:
// creating object of class Test

```
Test t = new Test();
```

- **Using Class.forName(String className) method:** There is a pre-defined class in java.lang package with name Class. The forName(String className) method returns the Class object associated with the class with the given string name. We have to give the fully qualified name for a class. On calling new Instance() method on this Class object returns new instance of the class with the given string name.

// creating object of public class Test

// consider class Test present in com.pl package

```
Test obj = (Test)Class.forName("com.pl.Test").newInstance();
```

- **Using clone() method:** clone() method is present in Object class. It creates and returns a copy of the object.

// creating object of class Test

```
Test t1 = new Test();
```

// creating clone of above object

```
Test t2 = (Test)t1.clone();
```

Variables in Java

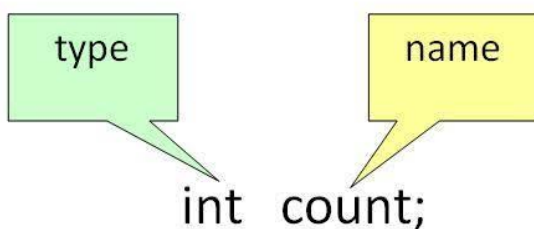
Variable in Java is a data container that saves the data values during Java program execution. Every variable is assigned a data type that designates the type and quantity of value it can hold. Variable is a memory location name of the data.

A variable is a name given to a memory location. It is the basic unit of storage in a program.

- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location, all the operations done on the variable effects that memory location.
- In Java, all the variables must be declared before use.

How to declare variables?

We can declare variables in java as pictorially depicted below as a visual aid.



From the image, it can be easily perceived that while declaring a variable, we need to take care of two things that are:

1. **Datatype:** Type of data that can be stored in this variable.
2. **Dataname:** Name was given to the variable.

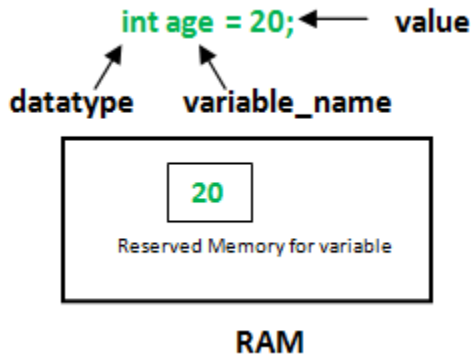
In this way, a name can only be given to a memory location. It can be assigned values in two ways:

- Variable Initialization
- Assigning value by taking input

How to initialize variables?

It can be perceived with the help of 3 components that are as follows:

- **datatype:** Type of data that can be stored in this variable.
- **variable_name:** Name given to the variable.
- **value:** It is the initial value stored in the variable.



Illustrations:

```
float simpleInterest;
```

```
// Declaring float variable
```

```
int time = 10, speed = 20;
```

```
// Declaring and Initializing integer variable
```

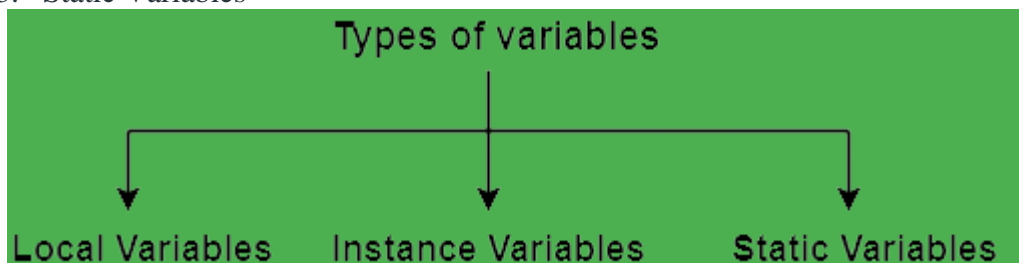
```
char var = 'h';
```

```
// Declaring and Initializing character variable
```

Types of Variables in Java

Now let us discuss different types of variables which are listed as follows:

1. Local Variables
2. Instance Variables
3. Static Variables



Let us discuss the traits of every variable been up here in detail.

1. Local Variables

A variable defined within a block or method or constructor is called a local variable.

- These variables are created when the block is entered, or the function is called and destroyed after exiting from the block or when the call returns from the function.
- The scope of these variables exists only within the block in which the variable is declared. i.e., we can access these variables only within that block.
- Initialization of the local variable is mandatory before using it in the defined scope.

- Java

```
/*package whatever //do not write package name here */
```

```
// Contributed by Shubham Jain
```

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

```
class GFG {
```



```

public static void main(String[] args)
{
    int var = 10; // Declared a Local Variable
    // This variable is local to this main method only
    System.out.println("Local Variable: " + var);
}
}

```

Output

Local Variable: 10

2. Instance Variables

Instance variables are non-static variables and are declared in a class outside any method, constructor, or block.

- As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
- Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier, then the default access specifier will be used.
- Initialization of Instance Variable is not Mandatory. Its default value is 0
- Instance Variable can be accessed only by creating objects.

Java

```

/*package whatever //do not write package name here */

import java.io.*;

class GFG {

    public String geek; // Declared Instance Variable

    public GFG()
    { // Default Constructor

        this.geek = "Shubham Jain"; // initializing Instance Variable
    }
//Main Method
    public static void main(String[] args)
    {

        // Object Creation
        GFG name = new GFG();
        // Displaying O/P
        System.out.println("Geek name is: " + name.geek);
    }
}

```

Output

Geek name is: Shubham Jain

3. Static Variables

Static variables are also known as Class variables.

- These variables are declared similarly as instance variables. The difference is that static variables are declared using the static keyword within a class outside any method constructor or block.

- Unlike instance variables, we can only have one copy of a static variable per class irrespective of how many objects we create.
- Static variables are created at the start of program execution and destroyed automatically when execution ends.
- Initialization of Static Variable is not Mandatory. Its default value is 0
- If we access the static variable like the Instance variable (through an object), the compiler will show the warning message, which won't halt the program. The compiler will replace the object name with the class name automatically.
- If we access the static variable without the class name, the compiler will automatically append the class name.

• Java

```
/*package whatever //do not write package name here */

import java.io.*;

class GFG {

    public static String geek = "Shubham Jain";    //Declared static variable

    public static void main (String[] args) {

        //geek variable can be accessed without object creation
        //Displaying O/P
        //GFG.geek --> using the static variable
        System.out.println("Geek Name is : "+GFG.geek);
    }
}
```

Output

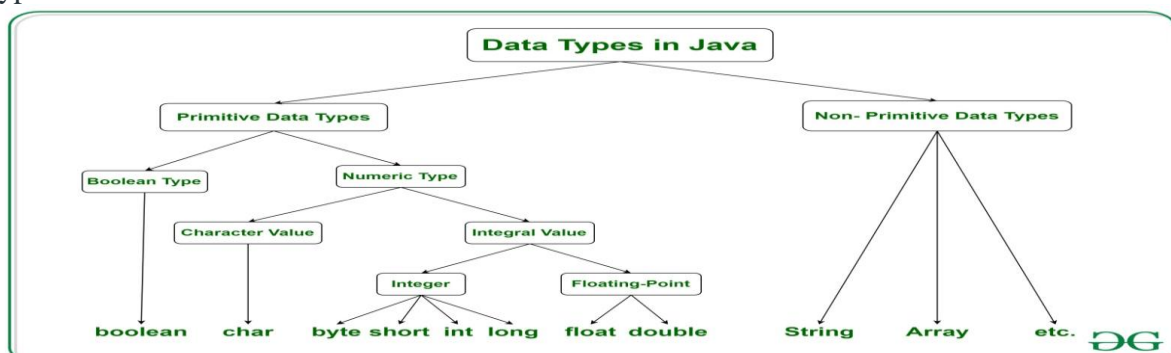
Geek Name is : Shubham Jain

Data types

Data types are different sizes and values that can be stored in the variable that is made as per convenience and circumstances to cover up all test cases. Also, let us cover up other important ailments that there are majorly two types of languages that are as follows:

1. First, one is a **Statically typed language** where each variable and expression type is already known at compile time. Once a variable is declared to be of a certain data type, it cannot hold values of other data types. For example C, C++, Java.
2. The other is **Dynamically typed languages**. These languages can receive different data types over time. For example Ruby, Python

Java is **statically typed and also a strongly typed language** because, in Java, each type of data (such as integer, character, hexadecimal, packed decimal, and so forth) is predefined as part of the programming language and all constants or variables defined for a given program must be described with one of the data types.



Java has two categories in which data types are segregated

- 1. **Primitive Data Type:** such as boolean, char, int, short, byte, long, float, and double
- 2. **Non-Primitive Data Type or Object Data type:** such as String, Array, etc.

Types Of Primitive Data Types

Primitive data are only single values and have no special capabilities. There are **8 primitive data types**. They are depicted below in tabular format below as follows:

TYPE	DESCRIPTION	DEFAULT	SIZE	EXAMPLE LITERALS	RANGE OF VALUES
boolean	true or false	false	1 bit	true, false	true, false
byte	twos complement integer	0	8 bits	(none)	-128 to 127
char	unicode character	\u0000	16 bits	'a', '\u0041', '\101', '\w', '\v', '\n', '\beta'	character representation of ASCII values 0 to 255
short	twos complement integer	0	16 bits	(none)	-32,768 to 32,767
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2	-2,147,483,648 to 2,147,483,647
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F	upto 7 decimal digits
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d	upto 16 decimal digits

Let us discuss and implement each one of the following data types that are as follows:

Type 1: boolean

Boolean data type represents only one bit of information **either true or false**, but the size of the boolean data type is **virtual machine-dependent**. Values of type boolean are not converted implicitly or explicitly (with casts) to any other type. But the programmer can easily write conversion code.

Syntax:

boolean booleanVar;

Size: Virtual machine dependent

Values: Boolean such as true, false

Default Value: false

Example:

- Java

// Java Program to Demonstrate Boolean Primitive DataType

// Class

class GFG {

// Main driver method

```

public static void main(String args[])
{

    // Setting boolean to true initially
    boolean b = true;

    // If condition holds
    if (b == true)

        // Print statement
        System.out.println("Hi Geek");
    }
}

```

Output

Hi Geek

Type 2: byte

The byte data type is an 8-bit signed two's complement integer. The byte data type is useful for saving memory in large arrays.

Syntax:

```
byte byteVar;
```

Size: 1 byte (8 bits)

Values: -128 to 127

Default Value: 0

Example:

- Java

// Java Program to demonstrate Byte Data Type

```
// Class
class GFG {
```

```
    // Main driver method
    public static void main(String args[]) {
```

```
        byte a = 126;
```

```
        // byte is 8 bit value
        System.out.println(a);
```

```
        a++;
        System.out.println(a);
```

```
        // It overflows here because
        // byte can hold values from -128 to 127
        a++;
        System.out.println(a);
```

```
        // Looping back within the range
        a++;
        System.out.println(a);
```

```
    }
}
```

Output

126

127

-128

-127

Type 3: short

The short data type is a 16-bit signed two's complement integer. Similar to byte, use a short to save memory in large arrays, in situations where the memory savings actually matters.

Syntax:

short shortVar;

Size: 2 byte (16 bits)

Values: -32, 768 to 32, 767 (inclusive)

Default Value: 0

Type 4: int

It is a 32-bit signed two's complement integer.

Syntax:

int intVar;

Size: 4 byte (32 bits)

Values: -2, 147, 483, 648 to 2, 147, 483, 647 (inclusive)

Note: The default value is '0'

Remember: In Java SE 8 and later, we can use the int data type to represent an unsigned 32-bit integer, which has a value in the range $[0, 2^{32}-1]$. Use the Integer class to use the int data type as an unsigned integer.

Type 5: long

The long data type is a 64-bit two's complement integer.

Syntax:

long longVar;

Size: 8 byte (64 bits)

Values: {-9, 223, 372, 036, 854, 775, 808} to {9, 223, 372, 036, 854, 775, 807} (inclusive)

Note: The default value is '0'.

Remember: In Java SE 8 and later, you can use the long data type to represent an unsigned 64-bit long, which has a minimum value of 0 and a maximum value of $2^{64}-1$. The Long class also contains methods like comparing Unsigned, divide Unsigned, etc to support arithmetic operations for unsigned long.

Type 6: float

The float data type is a single-precision 32-bit IEEE 754 floating-point. Use a float (instead of double) if you need to save memory in large arrays of floating-point numbers.

Syntax:

float floatVar;

Size: 4 byte (32 bits)

Values: upto 7 decimal digits

Note: The default value is '0.0'.

Example:

- Java

```
// Java Program to Illustrate Float Primitive Data Type
```

```
// Importing required classes
import java.io.*;

// Class
class GFG {

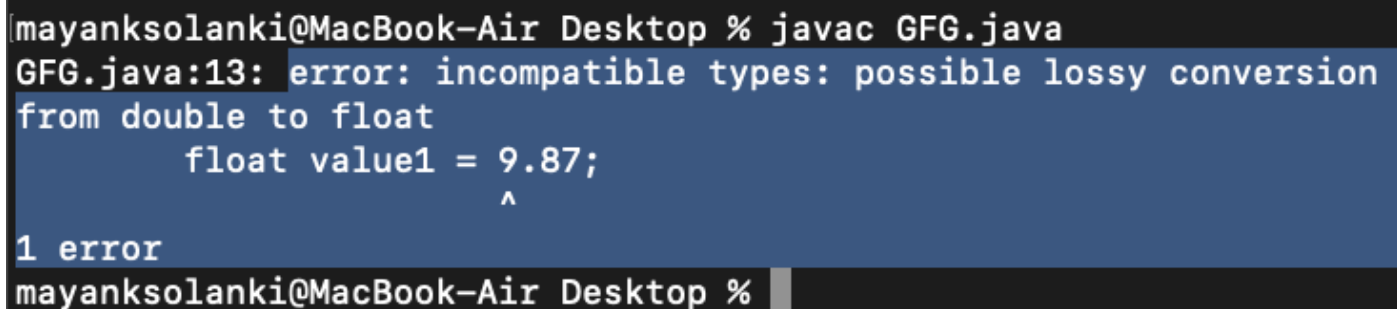
    // Main driver method
    public static void main(String[] args)
    {
        // Declaring and initializing float value

        // float value1 = 9.87;
        // Print statement
        // System.out.println(value1);
        float value2 = 9.87f;
        System.out.println(value2);
    }
}
```

Output

9.87

If we uncomment lines no 14,15,16 then the output would have been totally different as we would have faced an error.



```
mayanksolanki@MacBook-Air Desktop % javac GFG.java
GFG.java:13: error: incompatible types: possible lossy conversion
from double to float
    float value1 = 9.87;
                  ^
1 error
mayanksolanki@MacBook-Air Desktop %
```

Type 7: double

The double data type is a double-precision 64-bit IEEE 754 floating-point. For decimal values, this data type is generally the default choice.

Syntax:

double doubleVar;

Size: 8 bytes or 64 bits

Values: Upto 16 decimal digits

Note:

- The default value is taken as '0.0'.
- Both float and double data types were designed especially for scientific calculations, where approximation errors are acceptable. If accuracy is the most prior concern then, it is recommended not to use these data types and use BigDecimal class instead.

Type 8: char

The char data type is a single 16-bit Unicode character.

Syntax:

char charVar;

Size: 2 byte (16 bits)

Values: ‘\u0000’ (0) to ‘\uffff’ (65535)

Note: *The default value is ‘\u0000’*