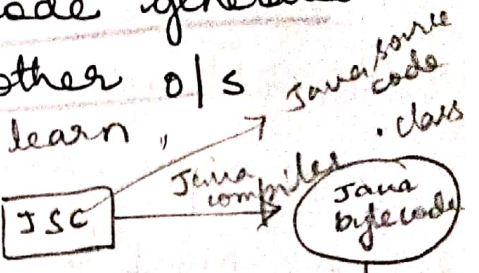
Machine level language

→ A M/C level language code is a code which contains instructions in the form of 0's and 1's

→ Machine level language code is a platform dependent code which means the code generated in one O/S cannot be used on another O/S for execution.



Advantages of Java [Technology works]

→ Java is a hybrid programming language [HLL → it is simple and easy to learn]

→ A Java compiler takes high level language stmts as the i/p and generates byte codes as the O/P.

→ Byte codes are such stmts which contains the instructions neither in HLL nor in MLL.

→ These stmts are not understandable by human brain because it would not be in human readable format.

→ Bytecodes are such stmts which would be understood by JVM

→ JVM is a s/w which is available in the internet specifically with respect to particular O/S. it means JVM s/w is platform independent.

→ Java pgming language support 2 features
* WORA - ~~it means write once~~
* Byte code

WORA → It means a java pgm can be written once and can be executed anywhere or run anywhere

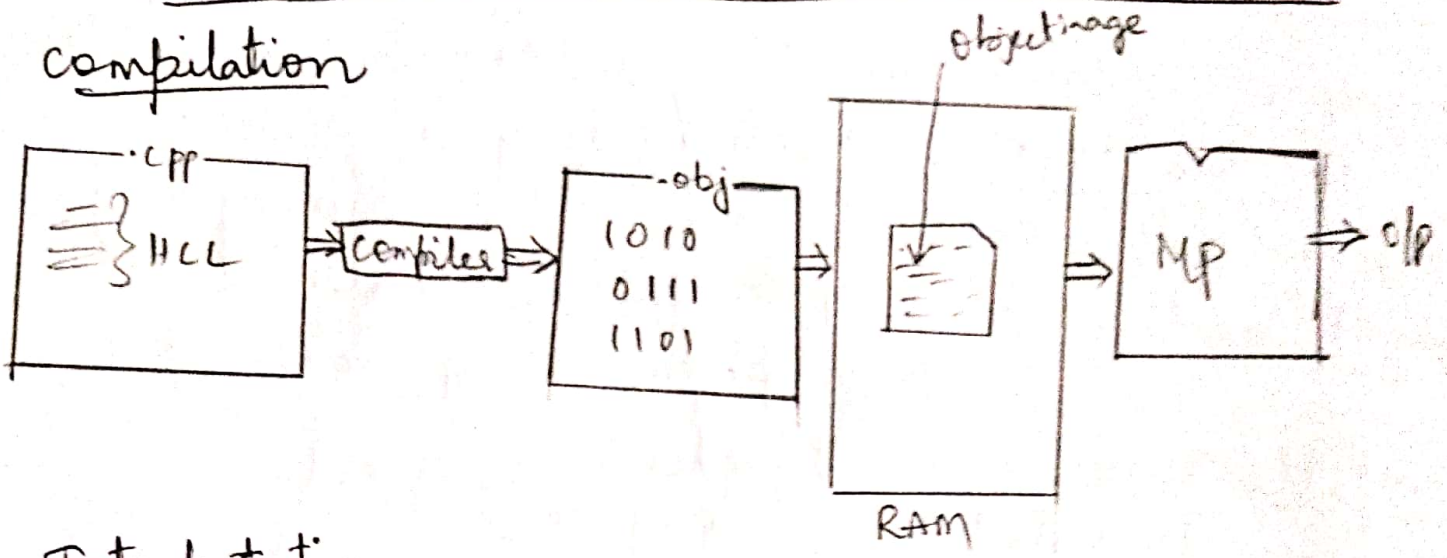
Bytecode → Java pgming lang is platform independent because of byte code facility

Difference b/w C++ and Java

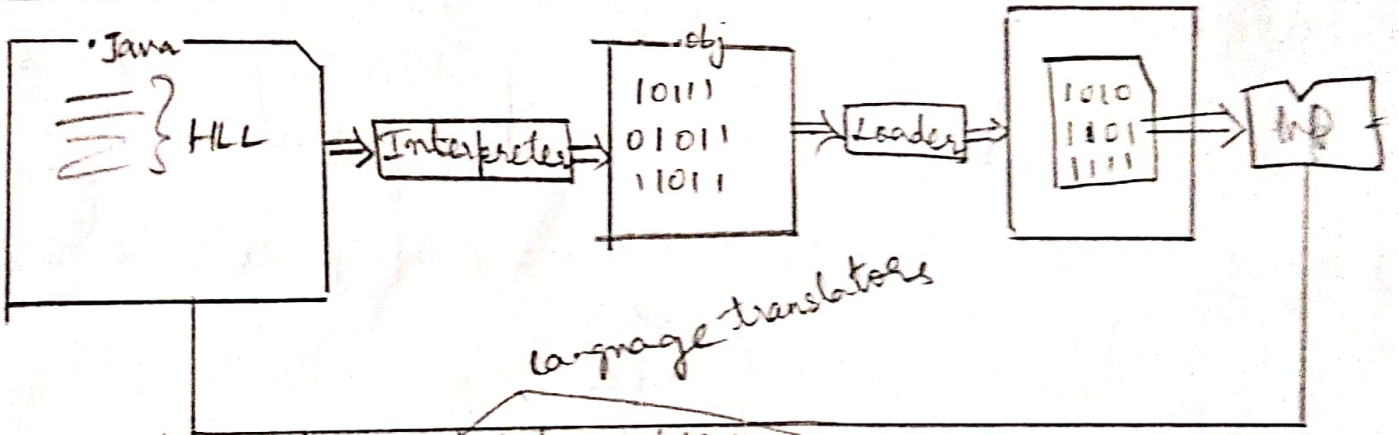
C++	Java
1. it is platform dependent pgming language	1. it is platform independent pgming language.
2. Upon compilation C++ generates object file	2. Upon compilation Java generates byte code.
3. C++ is a compiled and executed pgming language	3. Java is a interpreted and executed pgming language.
~~4. There is no header files in java~~	~~we have to use header file in C++~~
4. we have to use header file in C++	4. There is no header files in java
5. C++ support operator overloading	5. Java doesn't support operator overloading.

Difference b/w compilation and interpretation

Compilation



Interpretation



language translators

Both are translators but acts in way

Compilation	Interpretation
1. C++ pgming language are Compiled and executed	Java pgming language are interpreted and executed
2. They are faster in its execution	They are slower in its execution.
3. ~~Compiler~~ Converts the whole pgm in one go or in one stretch	it converts the pgm by taking a single line at a time
4. A compiler is a translator which transforms source lang (HLL) into object lang (MLL	its is a pgm initiates the execution of program written in a source lang.

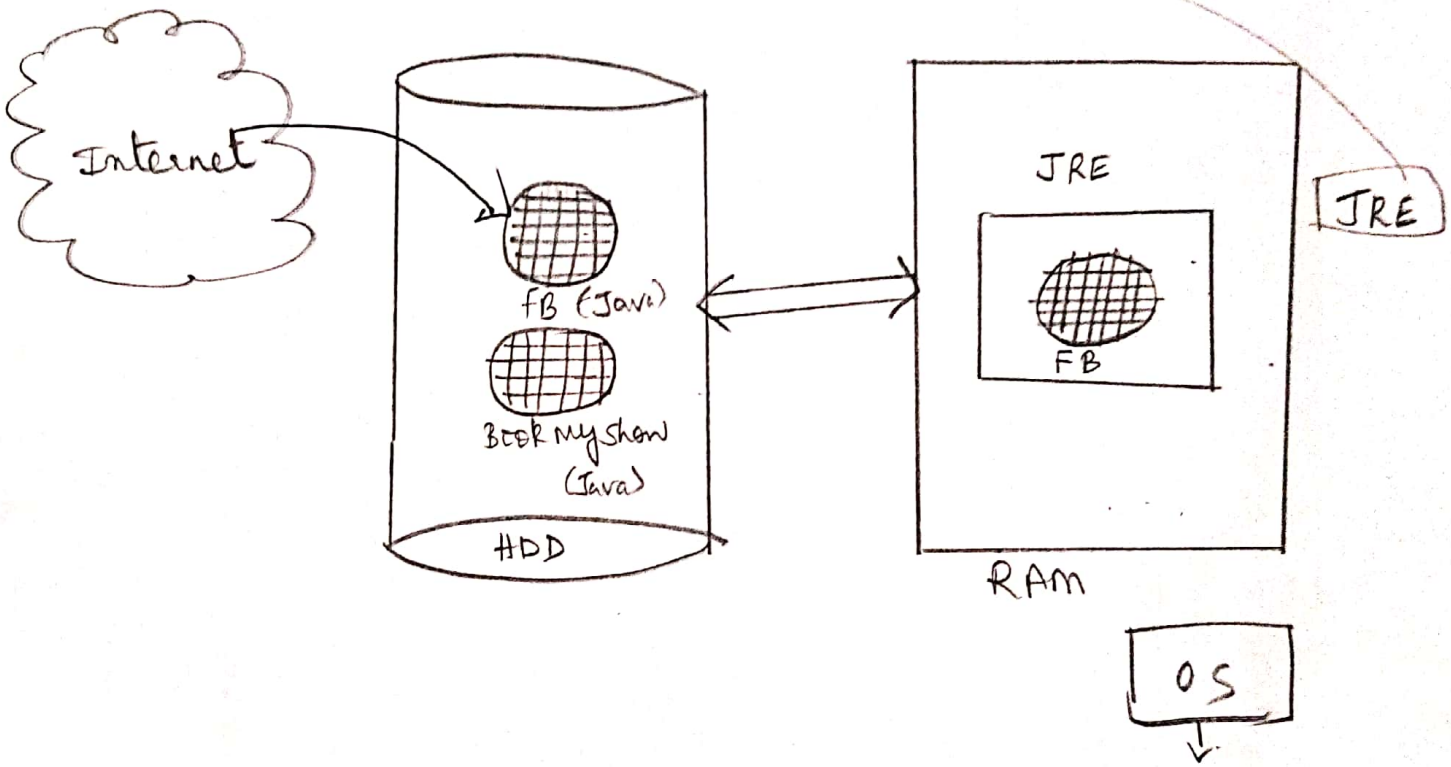
Features of Java

1. **Simple:** Syntax is based on C and C++. But no pointers, No goto, No operator overloading, No preprocessors, No global variables. Java is not a scripting lang.
2. **Object oriented:** Uses OOPs concepts (Inheritance, polymorphism, Encapsulation etc)
3. **Portable** – platform independent
   - "Write Once, Run anywhere" Runs on any platform that has JVM
4. **Secured** – Older s/m were never designed with the concepts of internet security when they were first conceived Java is intended to be used in n/w environment
5. **Robust** – Java uses the automatic garbage collection that prevents memory task.
   - Java is strictly typed language, Error free.
   - Runtime error of C & C++ is avoided.
6. **Dynamic** – Java loads in classes as they are needed
   - JVM is capable of linking dynamic new classes, methods and objects.
7. **Compiled and interpreted** – Java code is compiled to bytecode
   - Bytecode are interpreted on any platform by JVM.
   - Java pgms can be shared over the internet.
8. **High performance** – Bytecode are highly optimized → effective use of a resource
   - JVM executes Byte code much faster
9. **Distributed** – Java pgms can be shared over the internet
10. **Multithread** – Multithreading means handling more than one job at a time
    - The main advantage of multi-threading is that it shares the same memory
    - it is interactive and Networked program.

The Java development tool kit

JDK = JVM + JRE [JDK stands for Java development kit]

JRE: Java Runtime Environment



→ JDK is a combination of JVM and JRE it is used to give space in RAM

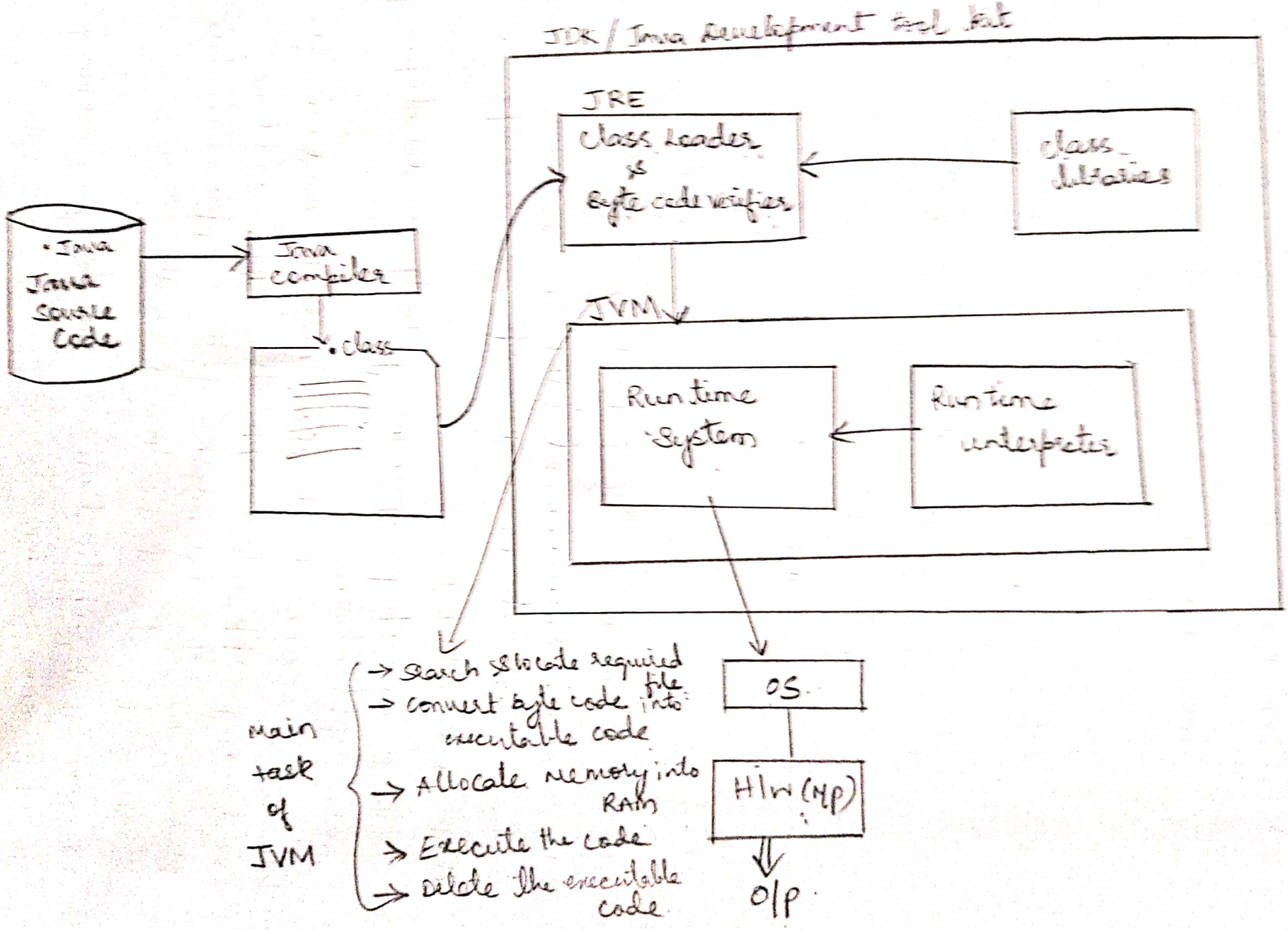
→ Java runtime Environment (JRE) is also known as Runtime, and it is a part of Java development ~~tool~~ kit (JDK),

→ it is a set of programming tools for developing java application.

→ When we connect to the internet and we down-load any application (app), the app would be stored in the harddisk. if the application has to run it should be loaded in the ram.

→ To run that application OS has to give some space on the RAM for the execution.

JDK Architecture



JDK Architecture :

→ JVM → [Java virtual machine]

→ Jvm is a part of JDK to execute java byte code (.*class)

** JDK has the following components.

→ JVM has Runtime interpreter and Runtime s/m.

** Runtime system - will check the code and gives the control to the o/s.

* Runtime interpreter - will check the code and gives the control to runtime System.

→ Class loader - The duty of the class loader is to load the class on to JRE and combines it with class libraries.

→ Bytecode verifier - The duty of the bytecode verifier is to check the underlying platform and also make sure that JRE is correctly loaded with the bytecodes

Note: when a Java program is compiled it would generate a .class file, the .class file will be picked by the class loader and generates a complete file by linking it with the class library. The bytecode verifier checks and handsover the control to Runtime s/m for the execution.

A First sample program

A simple java program

```
class Demo {
    public static void main (String[] args)
    {
        System.out.println ("Welccome to Java
                             programming language");
    }
}
```

class name and the starting letter should be Capital

if we write public then only os can see the main method

main method will not return anything

keyword

to access main method by o/s

jvm is sending string[] args and it will store it

where,

→ class → class keyword is used to declare classes in java.

→ public → it is an access specifier, it means this method is visible to all other classes.

→ static → it is again a keyword used to make a method static. To execute a static method you do not have to create an object of the class

→ void → it is the return type, means this method will not return anything.

→ main → main() method is most important method in java program. This is the method which is executed, hence all the logic must be inside the main() method. if a java class is not having a main() method, it causes compilation error. multiple main() methods in the same class not allowed.

→ String [] args → This represents an array whose type is string and name is args.

→ System.out.println → This is used to print anything on the console like printf in C language.

Note:

1) if o/s wants to see main() method to transfer the control then main() method should be made as public.

2) if the o/s wants to access the main() method then the main method should be declared as static.

Three steps to execute java program [on windows]

1. Enter the program - Open a text editor and write the code. Then save the file as class name, which contains method.
   Eg: Demo.java
   The class name first letter should be written always in capital letter.

2. Compile the program - Open the command prompt and go to the directory where you have saved java pgm. Then compile using javac

   javac Demo.java

   This command will call the java compiler and ask to compile the specified file. if no errors prompt will go to next line.

3. Run the program - To run the java code the syntax is java Filename
i.e java Demo

*write a java pgm to perform addition of 2 nos.

```
class Addition{
    public static void main(String[] args){
        int a=20;
        int b=30;
        int c=a+b;
        System.out.println("The sum is:"+c);
    }
}
```

O/P:

Identifiers in java

An identifiers is a name given to a method, variable and any other user-defined items that you want to identify in program

To access the memory location we use identifiers.

Rules

1) Can have one to several characters
2) variables should begin with a letter (A-Z, a-z), an underscore (_), a dollar sign ($)

→ A keyword cannot be used as an identifier
→ Cannot start with a digit but digits can be used after first character
→ Identifiers are case sensitive.

The java keywords

→ Fifty keywords are currently defined in the java language. These keywords, combined with the syntax of the operators and separators, form the foundation of the java language.
→ These keywords cannot be used as names for a variables, class, or methods.
→ The keywords const and goto are reserved but not used.

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	throws	super	Switch	synchronized
this	throw	break	transient	try	void
volatile	while				

D/w b/w Variable and Identifier
Variable is a name given to memory location
identifier is a name given to entity in a pgm.

The Key attributes of object oriented programming

→ Every entity is an object
→ No useless object exist (eg: dead body)
→ Every object has 2 parts
  * properties (has part)
  * behaviour (does part)

Eg: Dog

Colour breed No of legs cost	properties (noun or adjective)	bark eat sleep	behaviour (verbs)

Entity part

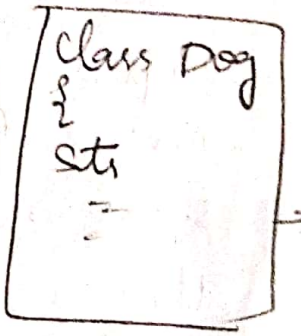
→ Every object representation is done by
  * instance variable
  * method

Programming part

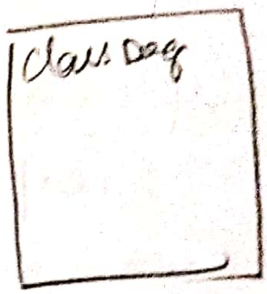
→ Information of an object is written with the help of class
→ class is just a blue print.

Eg:
```
class Dog {
    String colour;
    String breed;
    int noOfLegs;
    int cost;
    void bark()
    {
      System.out.println("Dog is barking");
    }
    void eat()
    {
      System.out.println("Dog is eating");
    }
```

instance variable
method



```
    void sleep ()
    { System.out.println ("Dog is sleeping");
    }
}
```



Class DogApp → class name

```
{
  Public static void main (String[] args)
  {
      Dog d1 = new Dog();
  System.out.println ("Dog d1 information is")
      d1. bark ();
      d1. eat ();
      d1. sleep();
  System.out.println();
  System.out.println ("Dog d2 information is")
        Dog d2 = new Dog();
          d2. eat ();
          d2. sleep();
  }
}
```

(Annotations on `Dog d1 = new Dog();`: Dog → classname; d1 → name of the object; d1 → reference variable; new → JVM is activate; new Dog() → create an object → instance variable)

output

Dog d1 information
Dog is barking
Dog is eating
Dog is sleeping

Dog d2 information
Dog is eating
Dog is sleeping

To support the principles of OOP, all OOP languages, including java have 3

1. Encapsulation 2. polymorphism 3. Inheritance.

Encapsulation

Encapsulation is a mechanism of wrapping the data (variables) and code acting on the data (methods) together as a single unit.

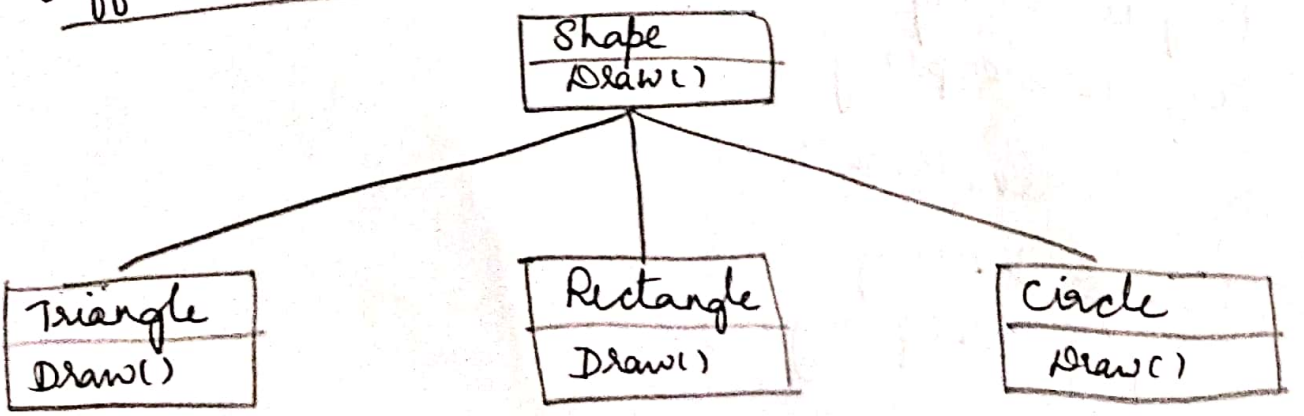
In encapsulation, the variables of a class will be hidden from other classes, and can be accessed only through the methods of their current class. Therefore, it is also known as data hiding.

Eg:
```
Class Employee {
    int eid;
    String name;
    float salary;
```

Polymorphism

it is derived from 2 greek words: "poly" and "morphs". The word "poly" means many and "morphs" means forms. So polymorphism means many forms.

it is defined as that allows one interface to access a general class of action or same name different operations.

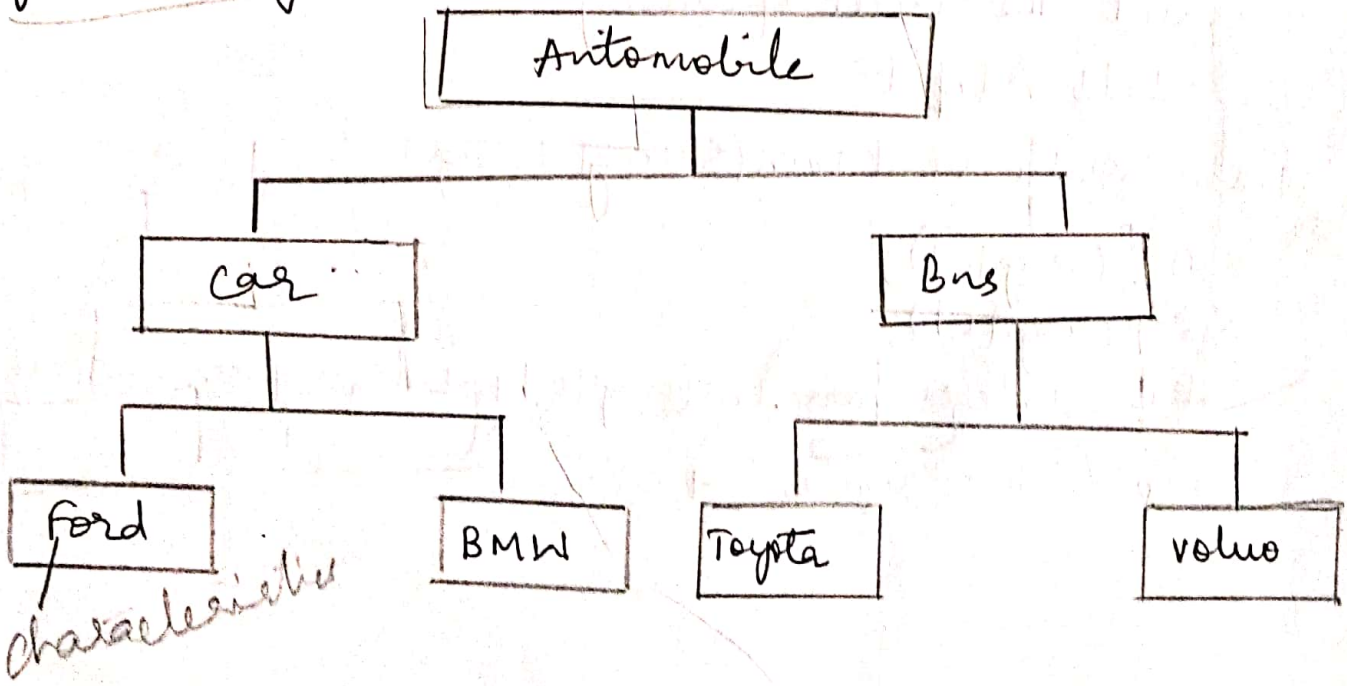


Inheritance

Inheritance can be defined as the process where one class acquires the properties (methods) of another. The idea behind inheritance in java is that you can create new classes that are built upon existing classes.

This is important because it supports the concepts of hierarchial classification. Without the use of hierarchies, each object would have to explicitly define all of its characteristics.

using inheritance, an object need only define those qualities that make it unique within the class. it can inherit its general attributes from its Parent. Thus, it is the inheritance mechanism that makes it possible for one object to be specific instance of a more general case.



Command Line argument

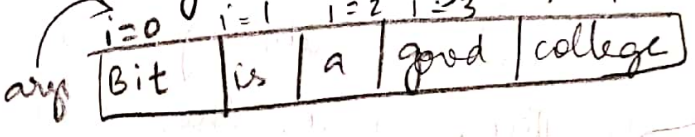
→ command line arguments are those arguments where the data is passed from the command line by the OS during the execution of the program

→ The data passed by the O/S should be collected by the main method String[]args

Eg: class CommandLine → joining two letters in class is called pascal convention

OS { it should collect data by main

```
Public static void main (String[] args)
{
  for (int i=0 ; i<=4 ;i=i+1)
  {
    System.out.println(args[i] + " ");
  }
}
}
```



O/P: BIT is good college

Eg: To print the result of adding

```
class AdditionInput
{ public static void main(String[] args)
{
  S.O.P (args[0]);
  S.O.P (args[1]);
  int c = Integer.parseInt(args[0] + Integer.parseInt(args[1]));
  S.O.p ("The sum is" + c);
}
}
```

Rule - First word first letter small second word first letter big is called camelconvention method

method representation

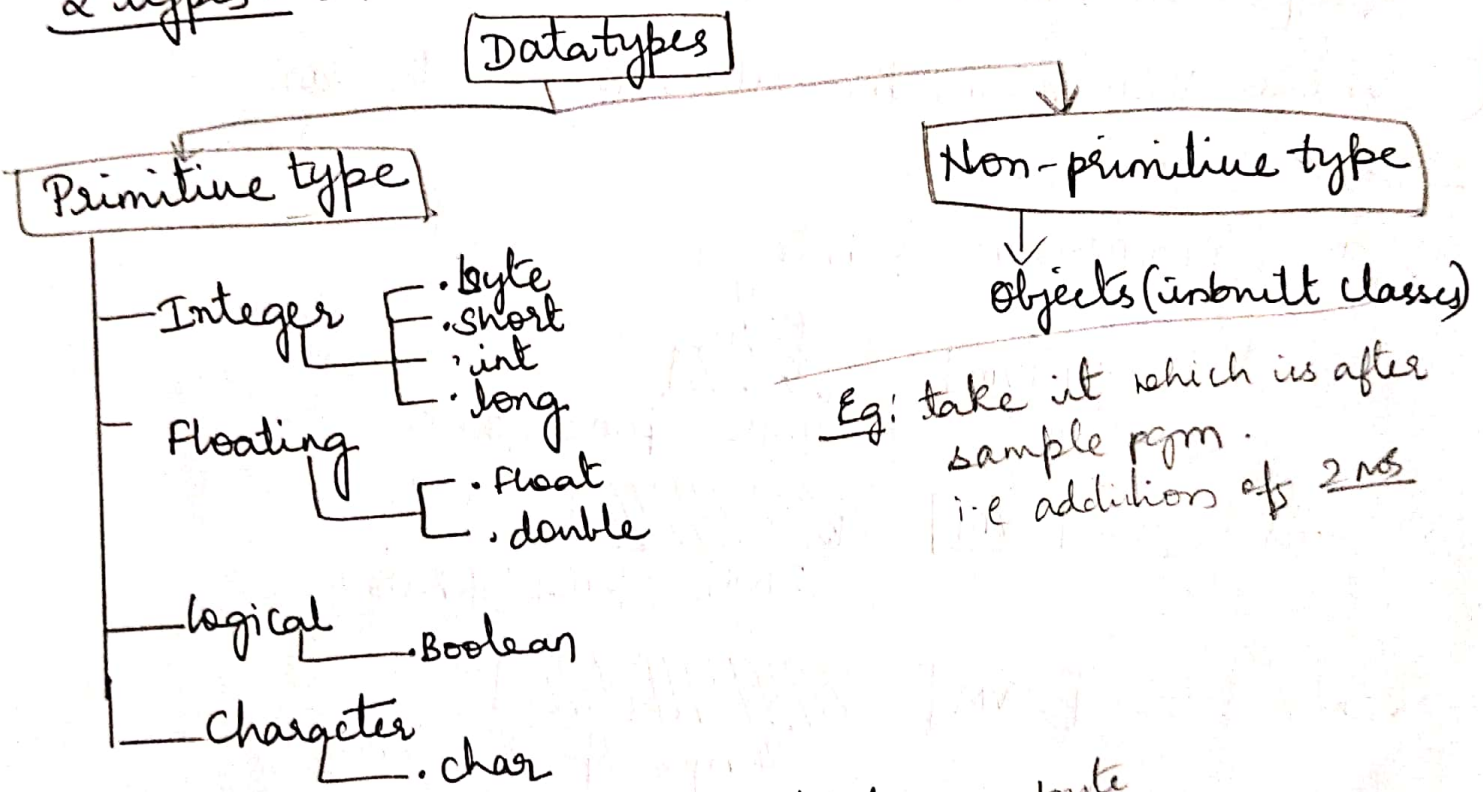
Introducing Datatypes and Operators

Datatypes

To store the information of a real world entity we use data types in realworld. we have data such as true, false, images, videos etc.

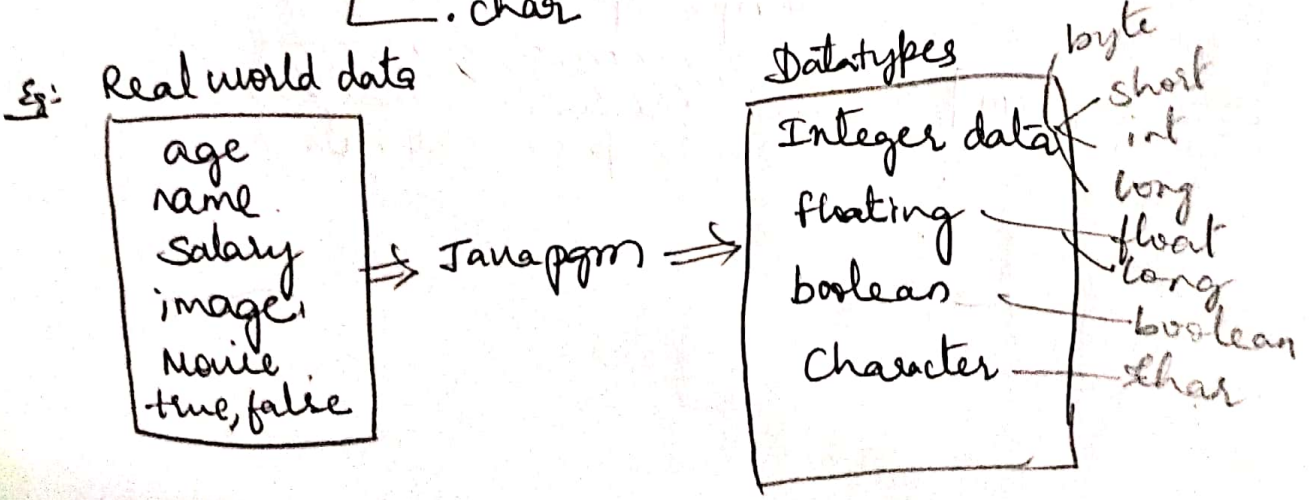
To store all these information through a java program, we use the concept of datatype.

Datatypes in java language is classified into 2 types: ① primitive ② Non-primitive



Eg: take it which is after sample pgm.
i.e addition of 2 nos

Eg: Real world data



A closer look at variables

Variables are declared using the form

<type var_name> : int count

where, type – datatype of the variable
var_name – name of the variable.

Initializing a variable

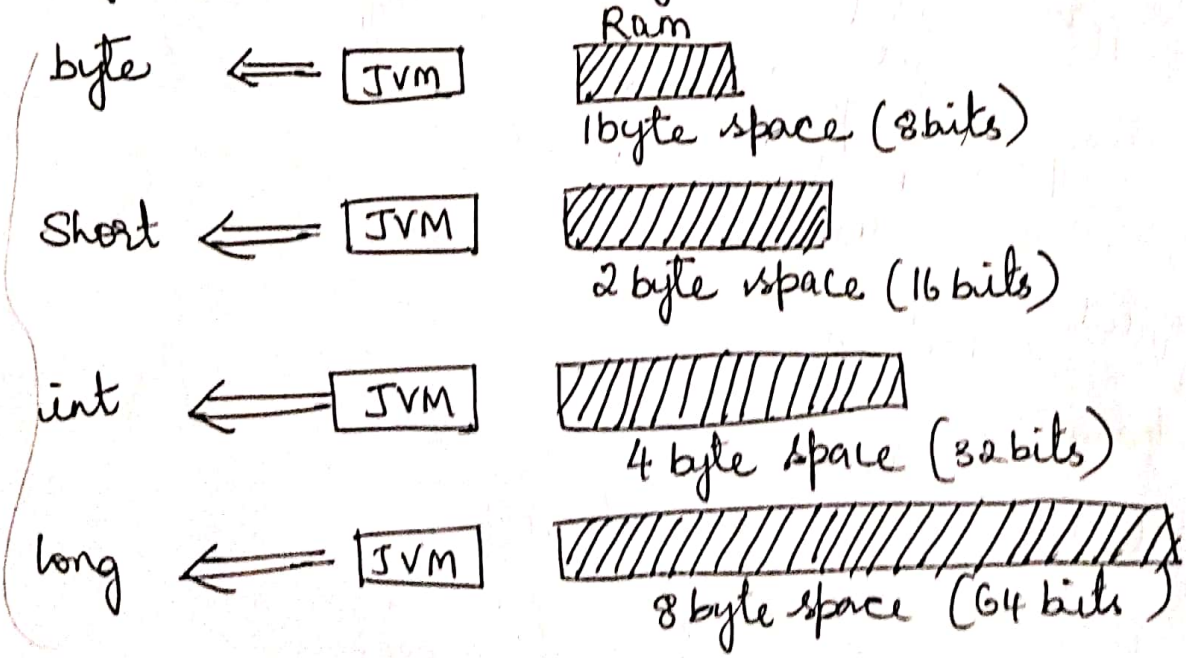
A variable must be declared before their used. The basic form of a variable declaration is:

<type var_name = value;>

Where, value – is the value that is to var.

Eg: int count = 10;

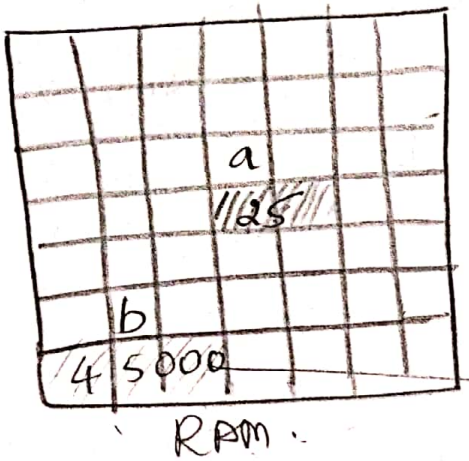
1) age of a person → integer



→ To store data we use datatype, to access the memory location we use identifiers or variables.

→ In order to access the memory we give name to the memory unit.

→ variables are nothing but identifiers



int b = 45000 → name for 4 bytes variable.

int a = 25

(when a value is stored in a ram we cant identifies the value in rows & columns, we have to give a variable name)

→ inside 4 bytes 45000 will go & sit.

Types of variables

→ Instance variable

→ static variable

→ local variable

* Any variable which is declared within a class are refered as instance variable.

* Any variable which is declared within a method are refered as local variable.

* Any variable which is declared with the help of static keyword is refered as static variable.

Eg:
```
class Bank {
    static int rateofInterest;      // static variable
    float pi;                       // instance variable
    float it;                       // instance variable
    public void getloan() {
        int amt;                    // local variable
    }
    public void calinterest() {
        int interest                // local variable
    }
}
```

Scope and life time of a variable

Any variable which is declared within a method can be accessed only within that method

Any variable which is declared within a block can be accessed only in that block.

Eg:
```
class Demo {
    public static void main(String[] args) {
        int x=10
        if (true)
        {
            int y=20;
            System.out.println(x);
        }
        System.out.println(y);
    }
}
```

o/p: 10

Error

If we put this stmt before closing the parentheses then no error

Operators

An operator is a symbol that tells the compiler to perform a specific mathematical, logical or other manipulation

Arithmetic	Relational	Logical
+	==	&
-	!=	|
*	>	^
/	<	||
%	>=	&&
++	<=	!
--		

Eg: class Operator [Example1]

```
{
  Public static void main (String[] args)
  {
     int a=100;
     int b=20;
     int c=a+b;
     int d=a-b;
     int e=a*b;
     int f=a/b;
     int g=a%b;
  System.out.println (c);
  System.out.println (d);
  System.out.println (e);
  System.out.println (f);
  System.out.println (g);
  }
}
```

Increment and decrement operator

Eg:

- Increment
  - postincrement (Print & increment)
  - Preincrement (increment & print)
- Decrement
  - Post Dec
  - Pre Dec

```
Class Sample1 {
Public static void main (String[] args)
{
    int a=5
    int b=6
    int c;
    int d;
        c= a++;
        d = b++ + ++a;
    System.out.println (a);
                       (b);
                       (c);
                       (d);
```

a 5 6 7

b 6 7

c 5

d 6 + 7 = 13

```
a=10
b=5
c=a--+--b+++b;
d=--a + ++b + ++c;
```

a 10 9 8

b 5 4 8 6

c 19 20

d 34

O/P : c = 10 + 4 + 5 = 19

d = 8 + 6 + 20 = 34

Shorthand Assignments operators

```
class shorthand
{
  Public static void main (String[] args)
  {
    int a=10;
    int b=5;
     b+=a;
    System.out.println(b);

     b-=a;
    System.out.println(b);

    b*=a;
    System.out.println(b);

    b/=a;
    System.out.println(b);

     b%=a;
    System.out.println(b);
  }
}
```

b+=a
= b+a
b = 5+10 = 15

b-=a
= b-a
= 15-10
= 5

b*=a
= b*a
= 5 x 10
= 50

b/=a
= b/a
= 50/10
= 5

b%=a
= b%10
= 5

Type conversion

The process of converting from one type to another type is called type casting

We have 2 types of type casting (conversion)

→ Implicit type casting

→ Explicit type casting

Implicit type casting

In this type of casting automatically compiler will type cast the data.

Eg:
```
class Implicit
{
  Public static void main(String[] args)
  { int
    byte a=10;
    double b;
      b= a;
      System.out.println(a);
      System.out.println(b);
  }
}
```

O/P: 10
10.0

Explicit type casting

In this type of casting compiler will not make the casting rather the programmer should explicitly make the data to go and sit in a variable, it is as shown below,

Eg:
```
class TypeConversionE
{
  Public static void main (String[] args)
  {
     double x=10.5
     int y= (int)x
     System.out.println(x);
     System.out.println(y);
  }
}
```

O/P:
```
10.5
10
```

Operator precedence

→ An operator's precedence determines at what point is evaluated in an expression.

→ An operator with a higher precedence will be evaluated before an operator with a lower Precedence.

Highest	Type	Categories	Precedence						
	unary	Postfix	expr++	expr--					
		Prefix	++expr	--expr	~	!	+	-	type cast
	Arithmetic		*	/	%				
			+	-					
	Shift		>>	>>>	< <				
	Relational	Comparison	>	>=	<	<=	instance of		
		Equality	==	!=					
	Bitwise	AND	&						
		Exclusive OR	^						
		Inclusive OR	|						
	Logical	AND	&&						
		OR	||						
	Ternary		? :						
Lowest	Assignment		=	op=					

Program control statements

Inputting characters from Keyboard

→ To read a character from the keyboard, we make use of System.in.read()

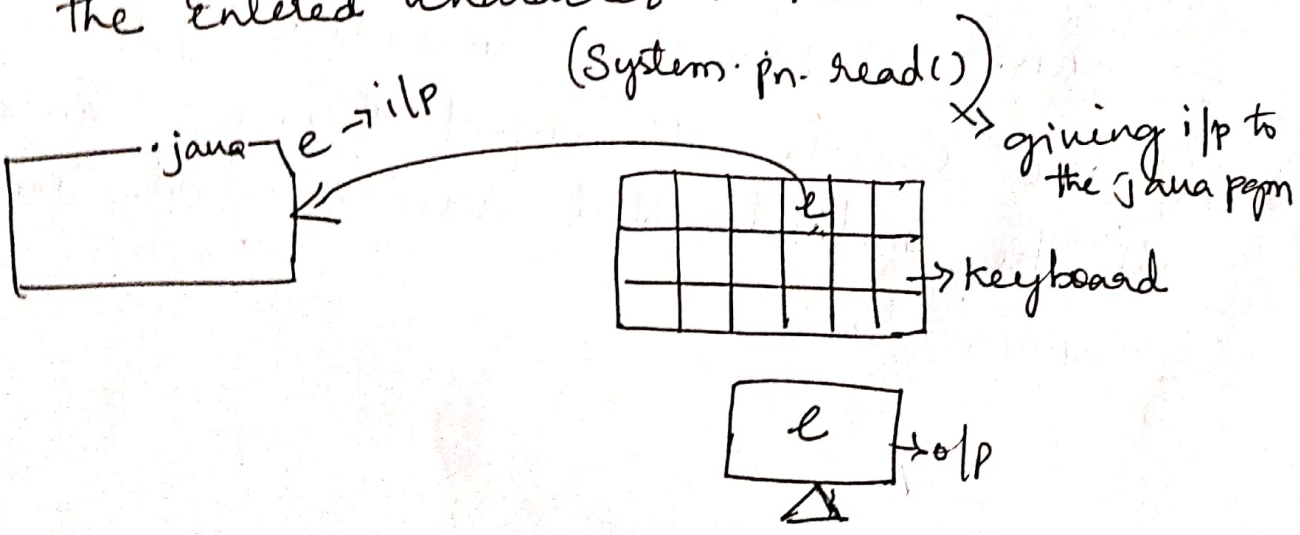
→ System.in is the complement to System.out

→ it is the input object attached to the keyboard.

→ The read() method waits until the user presses a key and then returns the result.

Eg:
```
class Input
{
    Public static void main (String[] args) throws Exception
    {
        System.out.println ("Please Enter a character from
                             the Keyboard")
        char c =(char) System.in.read();
        System.out.println ("The Entered character is :"+c);
    }
}
```

O/P : Plz Enter a character from the keyboard
e
The Entered character is : e



Reading an integer data from the keyboard

In order to read integer, float, int, string from the keyboard. we create an object of Scanner class.

```
class Scanner
{
  Public Int nextInt()
  {
    ==
  }
  Public float nextfloat()
  { ==
  }
  Public String nextLine()
  { ==
  }
  Public String next()
{ ==
}
```

This class is present inside java.util.Scanner.

Scanner → which is invented by James ghosling

if we create the object (Scanner) then we can access all the methods that are available in Scanner class.

Eg: nextInt (Same pgm can be written for float)

```
import java.util.Scanner;
class MultipleInput
{
  Public static void main (String[] args)
  {
    Scanner scan = new Scanner(System.in);
    System.out.println ("Enter the Integer number");
    int a= scan.nextInt();
    System.out.println ("Enter the Integer number");
    int b= scan.nextInt();
    int c= a+b;
    System.out.println ("The sum is: "+c);
  }
}
```

name of the object → scan; JVM is activated → creating an object scanner

O/p: Enter the Integer number
10
Enter the Integer number
20
The sum is 30.

String and Sentence

In order to read only a string we use next()

In order to read one entire line we use nextLine()

Eg: next()

```
import java.util.Scanner;
class StringInput
{
  Public static void main(String[] args)
  {
    Scanner Scan = new Scanner (System.in);
    System.out.println("Enter the String data");

    String data = Scan.next()
    System.out.println("The string data is :"+data);
  }}
```

O/P Enter the string data
Sachin
The string data is : Sachin

Eg: nextLine()

```
import java.util.Scanner;
Class StringInputLine
{
 Public static void Main(String() args)
 {
  Scanner Scan = new Scanner (System.in)
  System.out.println("Enter the Line data");
  String lineData=Scan.nextline()
  String.out.println("The sentence data is "+linedata);
}}
```

O/P: Enter the Line data
Sachin is a good batsman
The sentence data is " Sachin is a good batsman

Control flow statements

Break :

When break is encountered by JVM the control would come out of the loop.

Eg:
```
class Break
{
  Public static void main (String[] args)
  {
    System.out.println ("Break statement");
    for (int i=1; i<=5; i++)
    {
      if (i==3)
      {
        break;
      }
      System.out.println (i);
    }
  }
}
```

i = 1 → T
i = 2 → T
i = 1, i = 2, i = 3, i == 3 → break, come out of the loop.

O/P:
```
Break statement
1
2
```

Continue : →

When continue stmt is encountered by JVM it would skip the statements which is present below continue. it tells the loop to immediately jump to the next iteration.

Eg:
```
class Continue
{
  Public static void main (String[] args)
  {
```

```
System.out.println("Continue statement");
 for (int i=1 ;i<=5 ;i++)
 {
    if (i==3)
    {
      continue;
    }
    System.out.println(i);
 }
}
```

o/p: continue statement

1
2
4
5

Introduction to classes, Objects and Methods

Class Fundamentals

→ A class is a blue-print from which the individual objects are created.

→ A class can have any number of methods to access the value of various kind of methods.

→ Java uses a class specification to construct objects.

Syntax:

```
class <class name>
{
  type var1 ;
  ...
  type method (parameter)
  {
    ==
  }
}
```

Eg:

```
class Dog
{
  String breed;
  String colour;
  int cost;
  void bark()
  {
  }
  void sleep()
  {
  }
}
```

How Objects are created

→ Objects is an instance of a class.

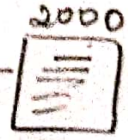
→ object is created by using keyword "new" that dynamically allocates memory for an object and returns a reference to it. The address in memory of the object allocated by new.

Syntax:

```
<classname objectname = new classname();>
```

Eg:

```
Dog d1 = new Dog();
```



→ The dot (.) operator is used to access both instance variable and methods.

Methods

→ A method is a collection of statements that are grouped together to perform an operation.

→ Each method has a name and performs only one task.

→ it is the same name used to call the method.

Syntax:

```
< access-specifiers ret-type name (Parameter list)
  {
     ==  // body of method
  }
>
```

Eg:

```
public static int methodName (int a, int b)
  {
     ==
  }
```

Here,

access-specifiers → provide a scope using public, private, protected, default

ret-type → type of data returned by the method

name → method is specified by name.

Parameter-list → identifier pairs separated by comma.

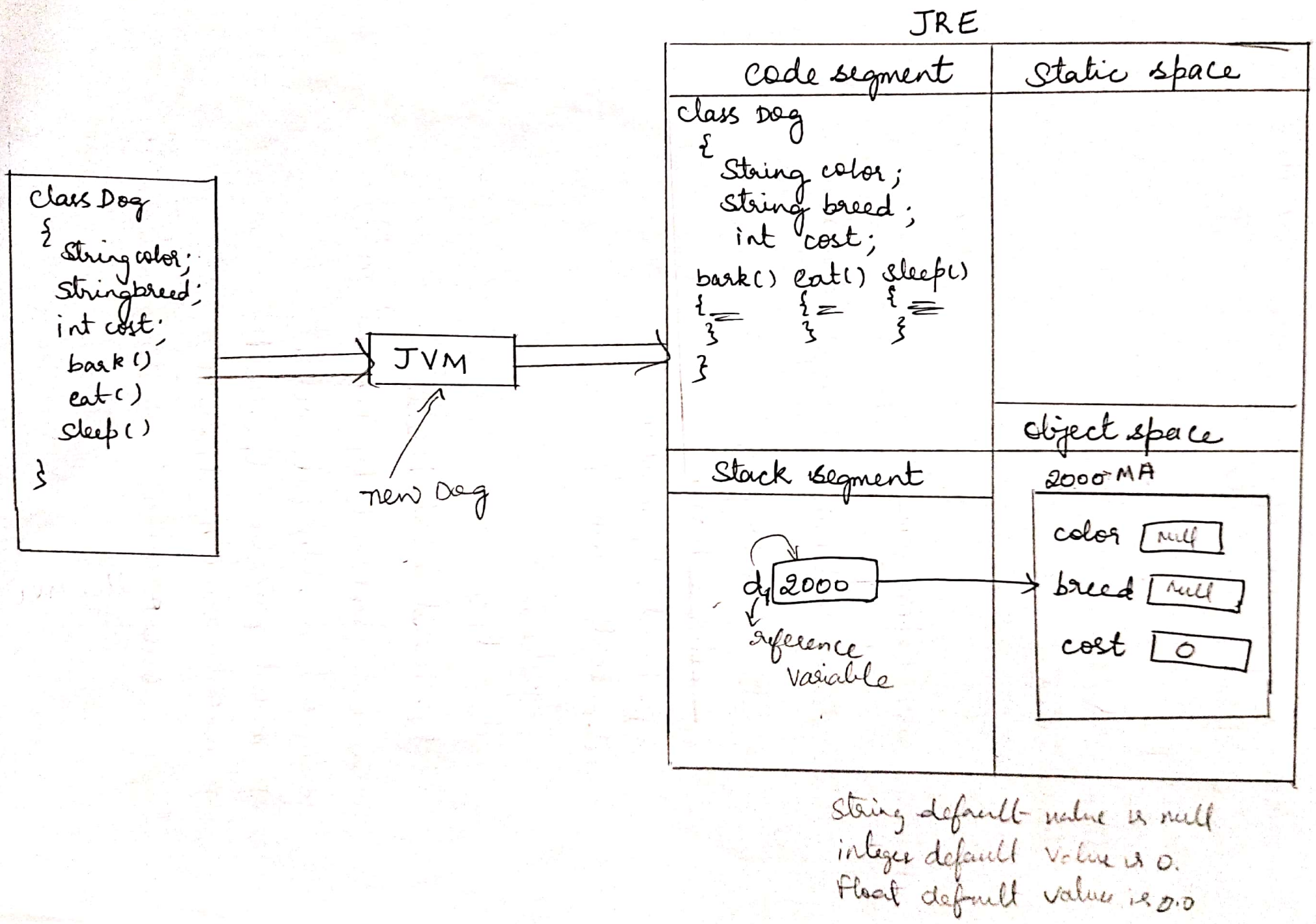
Eg: Classes, object and method

```
Class Dog
{
    String color;
    String breed;
    int cost;

    Public void eat()
    {
      System.out.println("Dog is eating");
    }
    Public void bark()
    {
      System.out.println("Dog is barking");
    }
    Public void sleep()
    {
      System.out.println("Dog is sleeping");
    }
}
```

Instance variable

```
class DogApp1
{
    Public static void main (String[] args)
    {
        Dog d1=new Dog(); -> object is created
        System.out.println ("Before Assigning");
        System.out.println (d1.color);
        System.out.println (d1.breed);
        System.out.println (d1.cost);

        d1.color = "brown";
        d1.breed = "Pomorean";
        d1.cost =10000;
        System.out.println ("After Assigning");
        System.out.println (d1.color);
        System.out.println (d1.breed);
        System.out.println (d1.cost);
    }
}
```

o/p: Before Assigning
null
null
0
After Assigning
brown
Pomorean
10000



Reference variable and Assignment

Assigning value of reference variable to another reference variable.

```
class Dog
{
  String breed;
  String color;
  int cost;
}
Public class RefAssignment
{
  Public static void main (String[] args)
  {
    Dog d1 = new Dog();
      d1.color = "brown"
      d1.breed = "Pomerian"
      d1.cost = "245";

      Dog d2;
      d2 = d1; → Assigned
     d2.breed = "Shepherd"
  System.out.println ("D1 Information");
  System.out.println (d1.color);
  System.out.println (d1.breed);
  System.out.println (d1.cost);
  System.out.println ("D2 Information");
```

```
        System.out.println (d2.color);
        System.out.println (d2.breed);
        System.out.println (d2.cost);
    }
}
```

o/p: D1 Information

brown
shephered
245
D2 Information
brown
shephered
245

Constructors

→ Constructors are the special type of method that is used to initialize the object and invoked at the time of object creation.

→ Constructors has the same name as that of the class name without any return type

→ These are the methods which would be called only during the creation of an object.

→ it allocate space for object

Types of constructor

→ Default constructor

→ parameterized constructor

Default Constructor

→ A constructor that have no parameter is known as default constructor.

→ if user doesn't supply the constructor for java class, the compiler builds an implicit default constructor for that class.

```
<class_name>()
{
}
```

Eg:

```
class Fan
{
  int cost;
  String color;
  String brand;
  Public void rotate()
  {
    System.out.println("fan is rotating");
  }
  Public void SwitchOn()
  {
    System.out.println("Fan is switched on..");
  }
  Public void switchOff()
  {
    System.out.println("fan is switched off");
  }
}
```

```
Public class constructorApp
{
  Public static void main (String[] args)
  {
    Fan f1 = new Fan();
    f1.rotate();
  }
}
```

we are calling constructor → default constructor

O/P: fan is rotating

Parameterized constructor

→ A constructor that have parameters is known as parameterized constructor.

→ if we want to initialize fields of the class with your own values, then use parameterized constructor.

Syntax:
```
<class name> (parameterlist)
{
  ==
}
```

Eg:
```
class Student
{
  String name;
  int id;
  String address;
  float height;
Public Student (String n, int i, String a,
                float h)
{
```

```
    name n;
     id  i;
    address a;
    height h;
  }
   Public void display()
  {
    System.out.println(name);
    System.out.println(id);
    System.out.println(address);
    System.out.println(height);
  }
}
 Public class StudentApp
{
 Public static void main(String[] args)
 {
   student S1 = new student("Sachin", 10, "Mumbai", 3.5f);
   S1.display();
 }
}
```

Here we are passing values i.e parameterized constructor

O/P: Sachin
10
Mumbai
3.5

The new operator Revisited

The new operator in java is responsible for the creation of new object.

Syntax: <class_name class_var = new class_name();>

classvar → variable of the class type being created

class name → name of the class that is being instantiated

new → The new can be used to create an Object of any class type.

→ The new operator returns the reference to the newly created object, which is assigned to class_var

The this Keyword

Eg:

```
class Cricket
{
   int runs;
   String name;
public Cricket (int runs, string name)
{
   this.runs = runs;
   this.name = name;
}
   public void display()
   {
    System.out.println("runs is :" + runs);
    System.out.println ("name is :" + name);
} }
```

```
public class thisApp
{
  public static void main (String[] args)
  {
    Cricket c = new Cricket (1000, "Sachin");
    c.display()
  }
}
```

o/p: run is :1000
name is : Sachin

Note:

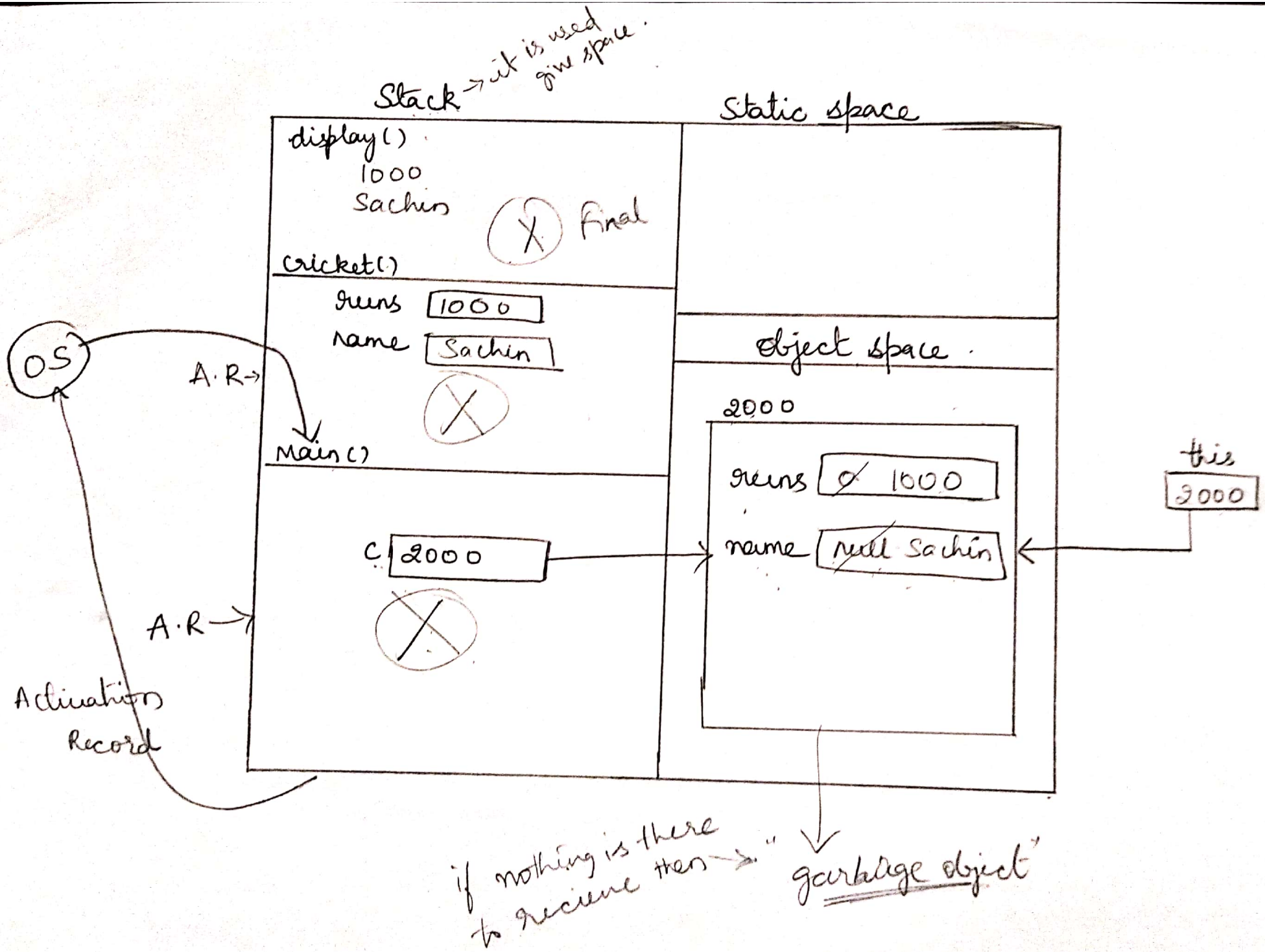
1) whenever a method gets called, a space would be given on the stack, this space is refered as activation record.

2) Once the method gets executed completely then the activation record would be completely deleted from the stack.

whenever local variable and instance variable have a same name it would result in a name clash. it is refered as shadowing.

To resolve the problem of shadowing, we use this keyword. (usage) ↓

① keyword THIS is a reference variable in java that refers to the current object ② this can be used to refer instance variable of current class ③ this can be used to invoke current class constructor ④ This can be passed as an argument in the method call ⑤ This can be passed as argument in constructor call ⑥ This can be used to return the current class instance from the method



Garbage collection and Finalizers

More Datatypes and Operators

Arrays

→ Arrays in java refers to collection of homogeneous data elements.

→ In java arrays are referred as objects that contains elements of similar datatypes.

→ it is a data structures where we store similar elements. we can store only fixed set of elements in java array.

→ Array is index based, first element of the array is stored at 0 index.

Types of arrays

→ Single dimensional Array

→ Multi-dimensional Array

Single dimensional array

Single dimensional array is a list of related variables.

Declare an array

The creation of array is two-step process

- First declare an array reference variable
- Second allocate memory for the array using new operator.

1) type[ ] array_name; 2) ~~type~~ array_name = new type [size]

- type[ ] array_name = new type [size];

Here, type - declare the element type
size - indicate the no. of elements that the array will hold.

Eg: int[ ] a; // creates array object
a = new int[5]; allocate memory for 5 elements

int [ ] a = new int [5]; // create and allocate memory for 5 integer elements has done in single line

---

Eg: Public class Array1
{
Public static void main (String[ ] args)
{
int [ ] a = {10, 20, 30, 40}
System.out.println("The array elements are:");
for (int

---

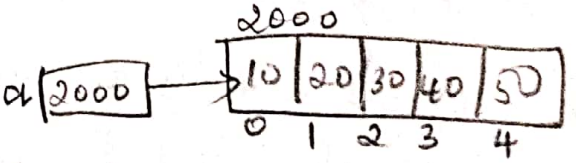
int a[5] = {10, 20, 30, 40, 50};

a

10	20	30	40	50

int [ ] a = new int [5];

a[0] = 10; a[3] = 40;
a[1] = 20; a[4] = 50;
a[2] = 30;



Eg: public class ArrayApp

```
{
  Public static void main (String[] args)
  {
    int [] a = new int [5];
     a[0] = 10;    a[2] = 30;    a[4] = 50;
     a[1] = 20;    a [3] = 40;
   System.out.println (a);
   for (int i=0 ; i<=4 ; i++)
   {
     System.out.println (a[i]);
   }
  }
}
```

O|P: [I@1db9742

10
20
30
40
50

* Pgm to take i|p from the keyboard.

```
import java.util.Scanner;
Public class ArrayApp
{
  Public static void main (String[] args)
  {
     int [ ] a= new int [5]; //Declaration
     Scanner Scan = new Scanner (System.in);
```

```
for (int i=0 ; i<=4 ; i++)
{
    System.out.println ("Enter the integer data");
    int data = Scan.nextInt(); // → reads the data  (camel case convention)
     a[i] = data ; // puts the data into an array
 }
 System.out.println ("The array elements are :");
  for (int i=0; i<=4 ; i++)
  { System.out.println (a[i]);
  }
  Scan.close();
 }
}
```

O/P:

```
Enter the integer data
10
Enter the integer data
20
Enter the integer data
30
Enter the integer data
40
Enter the integer data
50
The array elements are :
10
20
30
40
50
```

Multi dimensional array

Multi-dimensional arrays are arrays of arrays. Allows more than 2-dimensions.

Syntax

type[][]...[] arrayname = new type[size1][size2]..[sizeN];

Eg: int[][][] array = new int[2][3][4]

Initialization

type[][]..[] array_name = {{val1,, ... valn}, {val1,..valn},
. . . . . .
}

→ it refers to storing the information in the form of rows and columns. it is shown below

Eg:
```
import java.util.Scanner;
Public class TwoDimension
{
  Public static void main(String[] args)
  {
    int[][] a = new int[4][3]  ⟹ 12
    Scanner Scan = new Scanner(System.in);
    for(int i=0; i<=3; i++)
    {
      for(int j=0; j<=2; j++)
      {
        System.out.println("Enter the data:");
        a[i][j] = Scan.nextInt();
      }
    }
```

```
System.out.println("The array elements are:");
for(int i=0;i<=3;i++)
{
  for(int j=0;j<=2;j++)
  {
    System.out.println(a[i][j]+"\t");
  }
  System.out.println();
}
  Scan.close();
 }
}
```

o/p:

```
Enter the data
10
Enter the data
20
Enter the data
30
Enter the data
40
Enter the data
50
Enter the data
60
Enter the data
70
Enter the data
80
Enter the data
90
Enter the data
100
Enter the data
110
Enter the data
120
```

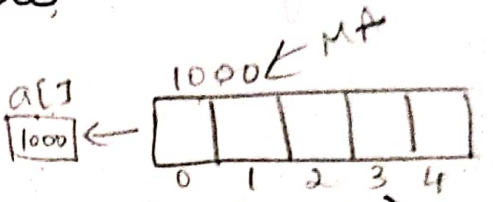
The array elements are

i → row / j → col	0	1	2
0	10 (0,0)	20 (0,1)	30 (0,2)
1	40 (1,0)	50 (1,1)	60 (1,2)
2	70 (2,0)	80 (2,1)	90 (2,2)
3	100 (3,0)	110 (3,1)	120 (3,2)

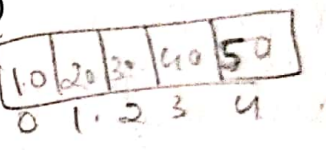
Using the Length member → The length is an instance variable of an array in java

write a pgm to print the array elements using the length member.

```
import java.util.Scanner;
Public class ArrayExample
{
   int a[] = new int [5];
   Scanner scan = new Scanner (System.in);
   System.out.println ("Enter the input :");
   for (int i=0; i<=a.length ;i++)
   {
       a[i]= scan.nextInt();
   }
   System.out.println ("The array elements are :");
   for (int i=0; i<a.length ;i++)
   {
     System.out.println (a[i]);
   }
   scan.close()
  }
}
```



inbuilt variable



10
20

[To find the element of an array, designer used length variable]

O/P: Enter the input

10
20
30
40
50

The array elements are:

10
20
30
40
50

Whenever we perform any operation with respect to an array, then the compiler automatically would set the size given by the user into an inbuilt variable called length. using which we can perform traversing operation in an array. [continued pg 38(A)]

The for-Each Style for loop

In case of for-each loop we would not perform

1) Initilization
2) condition
3) Incrementation

Instead of doing all these three things as a programmer we would only perform accessing operations.

Syntax:

```
for (datatype variable : arrayname)
{
   ========
}
```

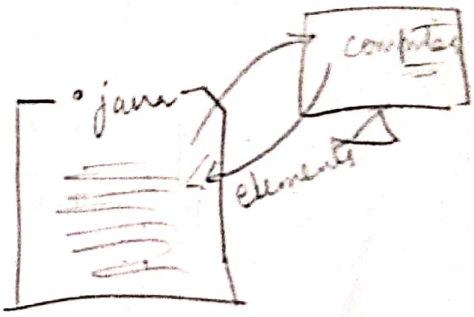
for-each - it is mainly used to traverse the array or collection elements. because it traverse each element one by one.

Eg:
```
for (int i: a)
{
  System.out.println(i);
}
```

For - general purpose control structure

foreach - is an enhanced for structure i.e applicable only to arrays and collections.

Eg:
```
Public class FOREach
{
 Public static void Main (String[] args)
 {
   int a[] = {10, 20, 30, 40};
   System.out.println ("The Array elements are:");
   for (int i : a)
   {
    System.out.println(i);
   }
 }
}
```



O/P:
```
The Array elements are:
10
20
30
40
```

length vs length() in java

Length member	Length()
1) The length variable is applicable to array but not for string objects	The length() method is applicable for string objects but not for arrays.
Eg: length int[] double[] String[]	Eg: length() String, String Builder.
2) To directly access field Member of array we can use .length	length() invokes a method to access a field member. length of a string can be modified because string class used this method.
3. Once array are initialized length cannot be changed	

Program to illustrate the concept of length and length()

```
Public class Test
{
Public static void main (String[] args)
 {

    int [ ] array = new int [4];
    S.o.p ("The size of the array is" + array.length);

    String str = "BITISGOODCOLLEGE"
    S.o.p (" The size of the string is" + str.length());

  }
}
```

O/P

```
4

16
```

Strings using array

Write a pgm to reverse a string

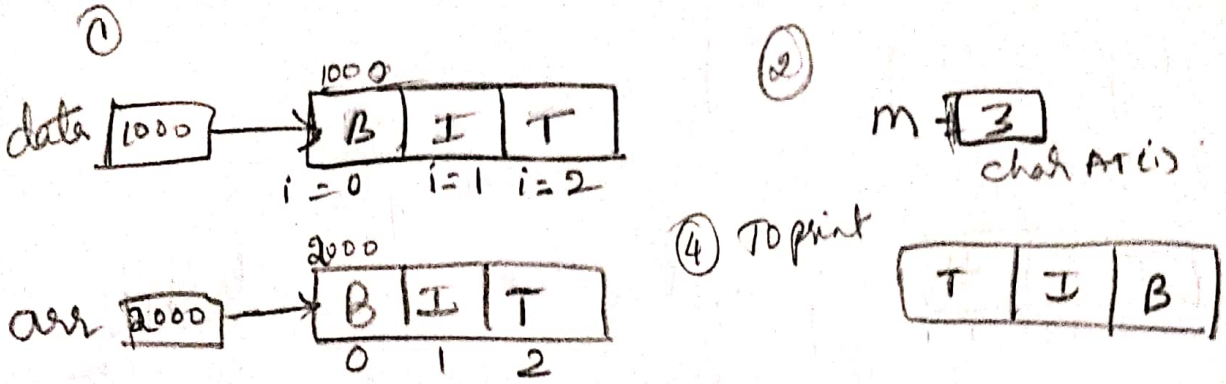
Step 1: Take string and calculate its length.

Step 2: Convert the given string into character Array.

Step 3: Print the character array in reverse order.

```
import java.util.Scanner;
class RevStr
{
  Public static void main(String[] args)
  {
    Scanner scan = new Scanner(System.in);
    System.out.println("Enter the string:");
    String data = Scan.next();
    int m = data.length(); // calculate its length
    char arr[] = new char[m]
    for (int i=0 ; i<=m-1 ; i=i+1)
    {
      arr[i] = data.charAt(i);
    }
    System.out.println("The Array elements are:");
    for (int i=m-1 ; i>=0; i=i-1)
    {
      System.out.println(arr[i]);
    }
  }
}
```

{ to print the string in reverse



O/P: Enter the String
bit
The Array elements are:
t
i
b

write a String pgm to reverse and check whether the string is palindrome or not.

Step 1: Take i/p from user, and check its length

Step 2: Convert the given string array to character array.

Step 3: Convert the string array and store the data in the rev order.

Step 4: Check whether the given character array are reversed array. if all character match print Palindrome otherwise print not palindrome.

Stop.

```
import java.util.Scanner;
class RevPalindrome
{
  Public static void Main (String[] args)
  {
    Scanner Scan = new Scanner (System.in);
    System.out.println ("Enter the String:");
    String data = Scan.next();
    int m = data.length();   // calculate length

    char org[] = new char [m];
    for (int i=0 ; i<=m-1 ; i=i+1)
    {
      org[i] = data.charAt(i);
    }
    char rev[] = new char[m];
    int i = m-1;
    int j=0;
    while (i>=0)
    {
      rev [j] = org[i]
      i = i-1;
      j = j+1;
    }
```

```
for (i=0; i<= m-1; i++)
{
  if (org[i] == rev[i])
    {
      continue;
    }
    else
    {
      System.out.println ("it is not a palindrome");
      System.exit(0);
    }
  }
  System.out.println ("it is a palindrome");
 }
}
```

O/P: Enter the string

Liril

it is a palindrome

Enter the string

Bangalore

it is not a palindrome