DATE: 02/07/18

Object Oriented Programming

IOT → Internet OF Things ⇒ Applications of Java
Cloud computing
Bit coin, Block chain } ⇒ Application of Java

Bit data ⇒ storage of massive amount of data
Artificial Intelligence (AI) ⇒ analyses the available data and predicts the future consequence. JDK 1.8

JAVA

* JAVA SE (Software Edition)
* ENTERPRISE EDITION (JAVA X - online shopping)
* MICRO EDITION (for mobile applications)
* KOTLIN ⇒ Heavy competitor for JAVA (Extensively used for mobile → android)
* .net is the product of MICROSOFT
* Now JAVA was initially developed by SUN and Later it was bought by ORACLE
* JAMES GAUSLING was the one who developed JAVA in 1995
* It was earlier named as OAK

Class
objects
state
Behaviour
Inheritance
Passing message
Access specifiers
Overriding
Methods
Attributes
Static
system

Data encapsulation
constructor → copy, default
Polymorphisms
Garbage collectors
Members
scanners
Data Abstraction
public and protected
String

Encapsulation :
Hiding data
Eg : class,
public,
protected

Inheritance :
Reusability

Polymorphism :
Objects taking many forms.

Data Abstraction :
Interface is available
Implementation independant

Methods :
Programs

Attributes : parameters

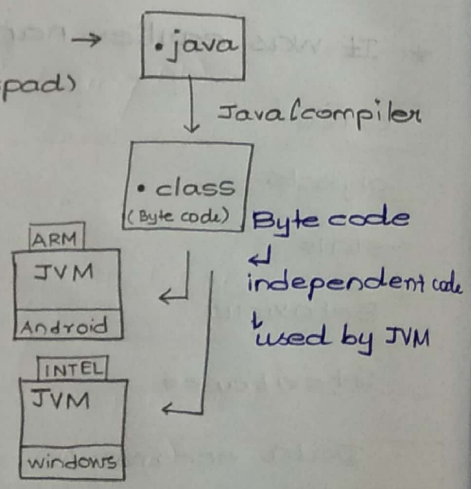
Garbage collector :
It is an inbuilt feature. It acts as a destructor and destroys the unwanted data.

Method overloading is allowed in JAVA but operator overloading is not allowed.

Write Once Run Anywhere
(WORA)

KEY POINTS :-

* It is platform independant
* JVM (Java Virtual Machine) - Software
* JIT
* JDK (Java Development kit)
* JRE
* Interpreter
* Byte code



JVM - platform dependant (contains interpreter)
JDK - contains the compiler. It has both JRE and JVM.

JRE: Only required to access the java files, webpages etc.

JIT → compiler.

www.oracle.com

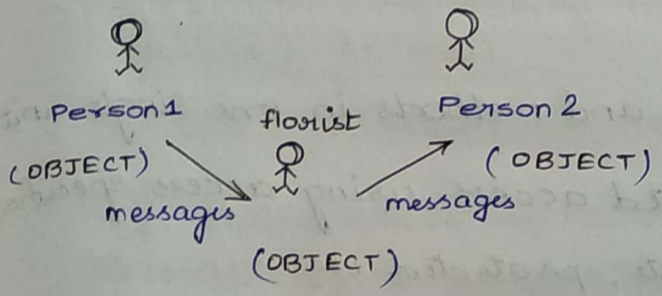
Deitel & Deitel

05/07/18

THINKING OBJECT ORIENTED

Why OOP is important?

* To solve real-life problems and scenario
* Here every real life character is an object. They have an instance / method of their own with some classes. Within the classes the Behaviour varies
* Methods ⇒ action to be performed.



Method: To pass flowers from person 1 to person 2

* Based on the language and its nature, the views (actions) and thoughts vary.
* CLASS: It is a blueprint. It is the collection of objects.

Eg:-

```
class House          → class
{                    → object
   String Door;
   String address;   } state

   public void open door() → Behaviour / methods
   {
print statement ← System.out.println("Door open");
in JAVA
   }
}
```

class example
{
public static void main (string args [])
{
House h1 = new House (); // h1 → object belonging to class "House"
House h2 = new House ();
h1.opendoor (); // public void opendoor [It is a method]
h2.opendoor (); // Here on seeing the method "opendoor" → opendoor of house h1 is first accessed and then h2 is accessed.
}

new → to create the new instance of the class.

Doubt

O/P :-
Door open
Door open

ENCAPSULATION :-

* Binding all states and methods in one single unit.
* To provide restricted access using access specifiers say: public, private, protected etc..

Eg:-

```
class person
{
    private string Name;
    public void setname (string N)
    {
        Name = N;
    }
    public string getname ()
    {
        System.out.println (Name);
    }
}
```

```
class example
{public static void main (String args[])
  {
    person s = newperson();
    System.out.println (s.name);
    s..
```

INHERITANCE:

Keyword :- extends

No need to create the common methods again and again.

Reusability is a key feature.

A class (child) can inherit the information of the parent methods. At the same time it can create and access its own methods.

Common methods can be grouped in a class

ABSTRACTION:

Only name of method is given whereas implementation is hidden.

POLYMORPHISM:

Method overloading → same method name with different parameters
→ It happens only within the class

Overriding.

Method overloading:-

```
class person
{
    string name;
    string fname;
    string lname;
    public getname (string)        → same method name
     {                                  → argument 1
     }
    public getname (string, string)    → same method name
     {                                  → argument 2
     }                          // so arguments vary.
```

Method Overriding:

Method name is same as well as arguments are the same.

~~class teacher~~ Same methods across different classes

Dynamic Polymorphism: (Method overriding)

* During Runtime we decide what class will be ~~Same metho used during~~ used.
* Static: user

JAVA → multilevel inheritance ✓

multiple inheritance ✗ → one single child inherits from different parent classes.

HW

Simple programs for 4 OOP concepts, Defn.

~~Main → Jc~~

C:> javac C:/java/student.java

C:> java-cp C:/java student

Original meaning of the word Paradigm?

→ Is OOP a paradigm? Yes

→ C, PASCAL → Imperative paradigm (procedural)

Paradigm :- A model with its own steps to solve

How do objects interact with each other? Messages

How do messages differ from method calls?

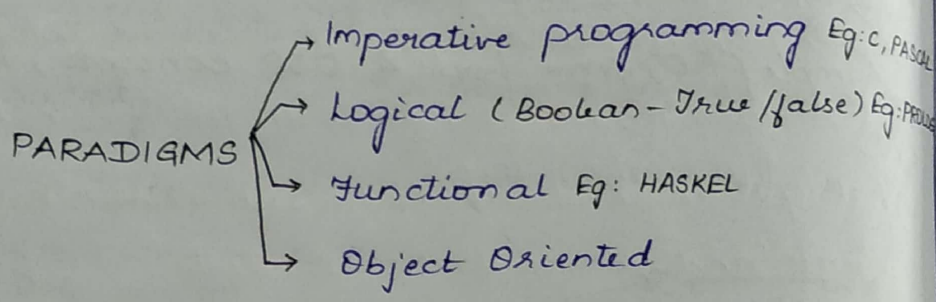
Intended receiver can be specified in JAVA. The receiver of the info can be mentioned.

What is the name applied to describe an algo. an object uses to respond to a request?

By invoking the methods

Why OO approach naturally imply a high degree of info. hiding? (Ans for why OOP is a new paradigm)

Class, its states and Behaviour



JAVA is a multi-paradigm. As versions were added JAVA transformed from OO paradigm to multi paradigm. Higher order functions are also accessible in JAVA.

What is class?

How classes are linked to behaviour?

By defining methods for every class. By creating Objects.

What is a class inheritance heirarchy? How is it linked to class and Behaviour?

common Behaviours can be shared or reused

What does it means for one method to override another method

What are the basic elements of process state model computation?

Turing machine,

A problem with effective algo. is computable by a turing machines which is abstract

Disadv:- "It is not common that any language is chosen to solve any problem" → statement is disproven.

Units :- I/P, O/P, Processing.

How does object oriented model differ from process model?

Solving real world problems naturally

In what way is OOP a simulation?

We can simulate and imitate real life objects

THINKING OBJECT ORIENTED

① ASSERTION 1:

* OOP is a revolutionary idea, totally unlike anyth that has come before in programming.

ASSERTION 2:

* OOP is an evolutionary step following naturall on the heads of earlier programming abstractions

Ans:- Assertion 1 and 2 are true

② Why OOP is so popular?

* Quickly and easily led to increased productivity and improved reliability.

* The desire for an easy transition from existing languages

* Similarity to techniques of thinking about problems in other domains.

③ What is Sapir-Whorf hypothesis? and Church-Turing Conjecture?

JAVA is case sensitive

file name and class name should be the same.

string is a class in JAVA.

c:/Java → Sample

c:/java> java Sample

jdk → bin → javac.exe

c:>cd Java

c:/java > Java C Sample.java

```
import java.io.*;   → will have the println statement
import java.util.*; → for strings inclusion
class Sample
{
  public static void main (String [] args)
  {
    System.out.println ("Welcome to Java Course");
  }
}
```

access from anywhere; no return; retain value; entry pt for execution; member; class; method

```
class Person
{
  private string name;
  public int aadharno;
  public void
```

to access a variable we need to have methods to have method getno() setno()

set number

get number

12/07/18

Object Oriented Design:

* OOP comes under a new paradigm.
* Objections under evolutionary & revolutionary concep
* Languages used : Simple / Efficient.

OOD :-

1. Earlier planning is a pivotal feature (Design plan)
2. What is the key idea driving OOD ?
   What is the idea of Responsibility tie to hiding info?
3. OOD ⇒ Responsibility driven design.
   C ⇒ Data driven design.
4. Implementation hiding → Interface
   Information hiding →
5. Small program ⇒ 1 programmer is responsible for develop
   of entire program. He is the sole responsibility
   for algorithm development.
6. Large program → The entire project is divided into several modules. Problem lies in knowing the path of recombining the modules and interact with other modules.
7. Objects interact with each other By Passing Messages
   (invoking method)
8. Problems in Large programming:
   → Distribution of task to ppl in various modules
   → Interaction b/w modules.

9. characterisation of Behaviour:

* A Behaviour has the ability to change the class.
* Use of getter and setter methods
* Figure out the scenario where the behaviour comes into play.

* Tasks in OOD
  - Identify software components
  - Responsibilities held by components (follows certain design principles)

* Design Principle
  - Cohesion (within the class) (methods that are high interrelated)
  - Coupling (among the classes) (interrelationship among classes)

* Acc. to Design principle, Cohesion rate should be higher than the coupling rate
* To reduce interrelationship, we opt for Interface concept
* Low coupling rate enhances the Software maintenence

SOFTWARE COMPONENTS OF INTELLIGENT KITCHEN

→ Greeter
→ Recipe database
→ Recipe class (Add/edit recipe)
→ Meals course (interact b/w recipes)
→ Planning (plan for the week/no. of people)

10. Scenarios help in identification of Behaviour

CRC (Class Responsibility Collaboration) Card

↳ holds the name of class
↳ information of that particular class
↳ collaboration b/w the classes.

11. Use of Physical Index Card over CRC card :

Assignment of 1 software program to a particular programmer.

12. Q:- D/b instance & class :

Q:- D/b coupling & cohesion :

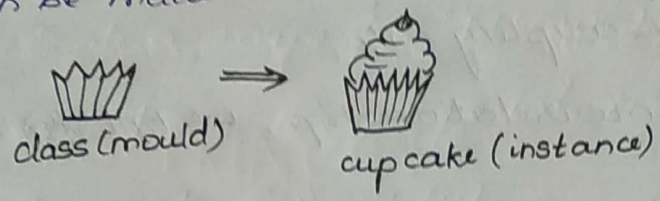
Q:- D/b Interface & Implementation :

13. PARNAS PRINCIPLE - DAVID PARNAS → concept of information hiding.

14. Instance: Has a common characteristics

Class : It must support unique characteristics.

15. Class is a mould from which "n" no. of instances can be made



HINTS (OOD):

* It is the community of objects each having assigned responsibilities, and working together towards the attainment of a common goal.

★ START WITH DESIGN ?

- Determine the software components
- Determine the specific responsibilities for each component.

★ OBJECT ORIENTED DESIGN:

- Responsibity Driven design
- Delegated control (Independant monitoring)

★ CONVENTIONAL PROGRAM?

- Data driven design
- Centralised controlled system (Overall monitoring of all Process)

SMALL PROGRAM	LARGE PROGRAM
One programmer	Many programmers
Problems involved: Development of entire algorithm	Problems involved: Management of details communication (Programmers / subsystem)

* BASIS FOR DESIGN:-

Identify the candidate objects (i.e) noun, phrases (they become class) in the problem statement.

Determine the responsibility for each object (i.e) Verb → Responsibilities for the class

* OBJECT ORIENTED DESIGN PRINCIPLE:

Single Responsibity principle
High cohesion
Low coupling.

* COHESION :- Strength of relationship within the module or class

COUPLING :- Between the modules / classes

* DESIGN:

- IIKH : Intelligent Interactive Kitchen Helper
- GOAL :- Develop a software to implement IIKH that will replace the card box recipes in the kitchen.
- HOW?

1. Characterise by Behaviour:

Software components
Responsibility.

2. Use CRC cards:

Candidate Responsibility Collaborators

3. Advantages of CRC:

Inexpensive

Erasable.

▲ ABILITIES OF IIKH :-

(what user can do with IIKH)

Browse database recipes

Add a new recipe

Edit an existing recipe

Plan a meal of several courses

Scale a recipe for 'N' number of users

Plan for a longer period (say a week)

Generate Grocery list.

* What does CRC card contain?

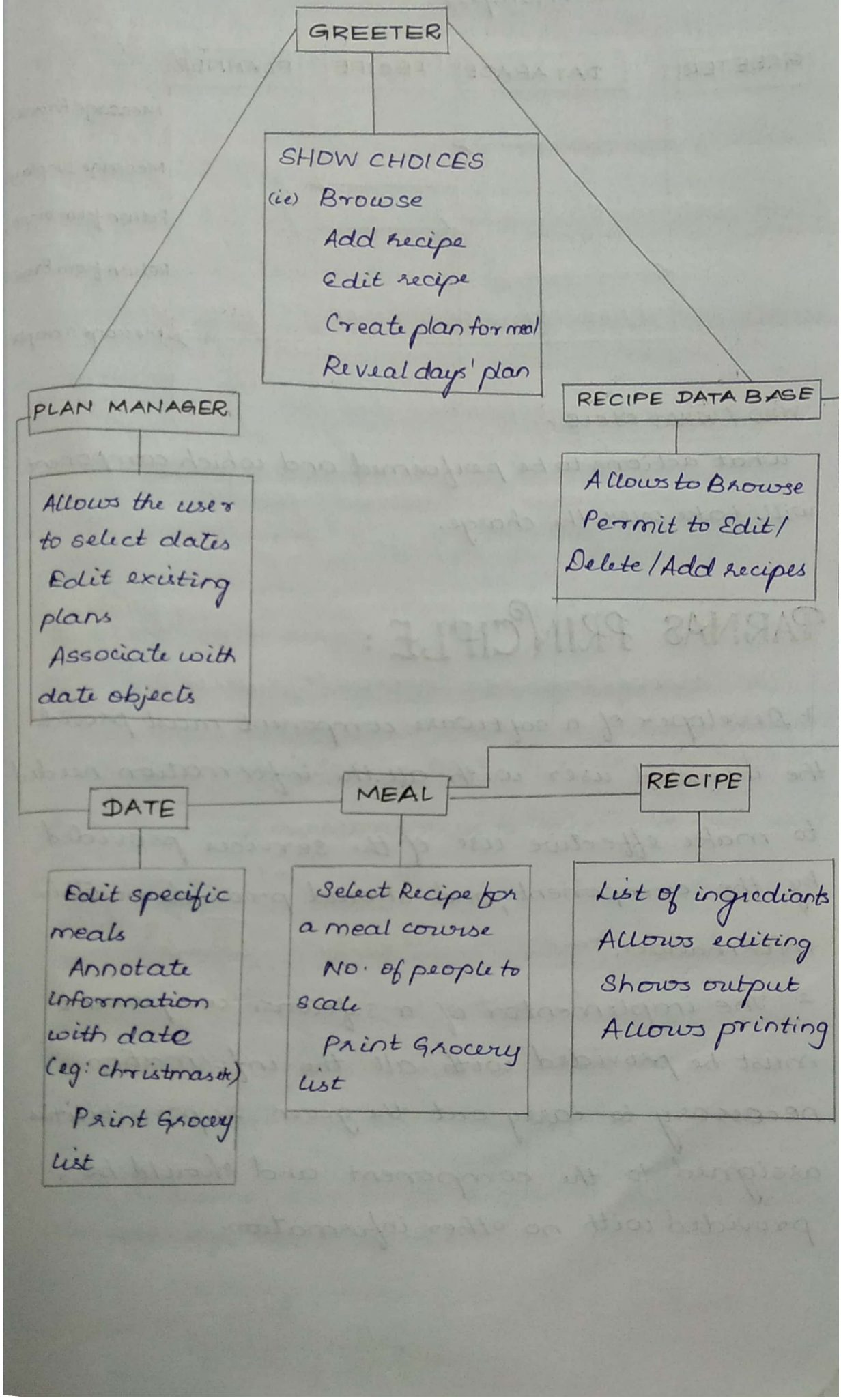
Component name Eg: Recipe

Responsibilities Eg: Edit recipe ()

collaborators Eg: List of other components.

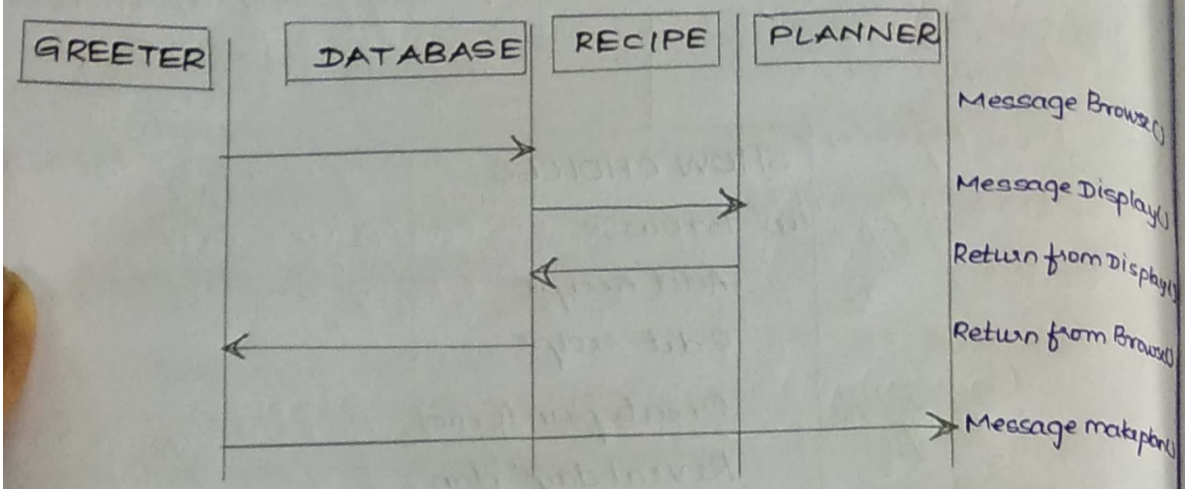
To summarise, CRC cards hold/record name, responsibilities and collaborators of a component.

IIKH



INTERACTION DIAGRAM (STATIC):

(static Relationships)



WHO / WHAT CYCLE:

What actions to be performed and which component will take over the charge.

PARNAS PRINCIPLE:

1. Developer of a software component must provide the intended user with all the information needed to make effective use of the services provided by the component, and should provide no other information.
2. The implementor of a software component must be provided with all the information necessary to carry out the given responsibilities assigned to the component and should be provided with no other information.

HW:-

1) How to read inputs from the user:-
   - Name
   - Gender
   - Age
   - Height and Weight
2) Fibonacci series (I/p "n" nos and find the fibonacci)
3) Largest of 3 numbers (using 'If else')
4) Factorial of a given no. (For loop / while / dowhile loop)
5) Calculator (using switch case)

GETTING USER INPUTS:

The class "Scanner" is used for getting user inputs in JAVA.

```
/* public class Sample
   {
      public static void main (String args[])
      {
         Scanner scan = new Scanner (System.in);
// the above statement enables us to take input from user.
      }
   } */
```

The above program compiled as such throws an error saying "Scanner not resolved".

To debug:- incorporate the header file:

```
import java.util.Scanner;
```

Adds the Scanner class to the program. Scanner class is already defined in JAVA.

System.out.print("..."); ⇒ to print o/p without breaking the line

Answers :-

1)
```
import java.io.*;
import java.util.Scanner; // inclusion of header for user
class p1
{
  public static void main (String args[])
  {
/*  Scanner scan = new Scanner (System.in)
    <keyword>          <variable name>   <keyword>
    System.out.println ("Enter the age");
    int age = scan.nextInt();
  <datatype><variable>   < reserved keyword>
    System.out.println ("The age is : ");
    System.out.print (age); */ - for getting age as

/* Scanner scan1 = new Scanner (System.in)
   System.out.println ("Enter Height");
   float height = scan1.nextFloat();
   System.out.println ("The height is :");
   System.out.print (height); */ - for getting height

/* Scanner scan2 = new Scanner (System.in)
   System.out.println ("Enter the name");
   String name = scan2.nextLine();
   System.out.println ("The entered name is :");
   System.out.print ("name"); */ for getting nam

  }
}
```

Quick Facts:

nextInt(); ⇒ Reserved keyword for getting integer i/p.

System.out.print ⇒ The print statement without 'ln' is used to print without breaking the line.

Using print statements, the outputs are printed on the same line continuously

println is the same as "\n" in C. O/p is printed on the next line.

Depending on the datatype of the input; the next statements can be accordingly used say

i/p	next statement	function
integer	nextInt();	Gets an integer input
float	nextFloat();	Gets a float input
String	nextLine();	Gets input as alphabets / sentences
True/False	nextBoolean();	Get an input which is supposed to be either True or False
double	nextDouble();	Gets an input of double datatype
	simplenext();	returns String

Getting an i/p from user, assigning it to a variable and displaying the i/p received as O/p ⇒

Getting User Input.

16/0/18

Oracle.com ⇒ JAVA tutorials.

JAVA Variables :-

* Class
* Instance
* Local
* Parameters

Variables defined in the class are of Both class and Instance types.

If the keyword "static" is used before the variable it is an class variable.

Instance variables are unique for each and every thing.

Class variables are common to certain entities.

Local variable : Scope of the variable lies only within the method.
Access specifier : "final" (only)

Instance variable : Access specifiers : public, private, protected, final etc. NOT : Static & Abstract.

Class variable : Access specifier : "Static" (only)

Fields :- Class variables & Instance variables (only)

VARIABLES :

Object stores its state in fields or variables.

KINDS OF VARIABLES :-

Instance / non-static
class / static
Local
Parameters

Global (not directly supported in JAVA)

INSTANCE :-

Associated with Objects

It should be defined inside the class but outside any method.

Memory is allocated only when the object is created

Unique value for each instance of the class.

Even if uninitialised, default value is printed.

ACCESS LEVELS :

Can use : • final • private • public • protected • default • transient

Cannot use : • abstract • static

Sample Eg :- (Instance Variables)

```
class Person
{
  private int aadharno; // instance variable.
  public void setAadhar (int aadharno)
    {
      this.aadharno : aadharno;
    }
  public int getAadhar ()
    {
      return aadharno;
    }
}
```

// when instance variable and parameter variable have same name, use 'this' pointer operator

```
class InstanceVariable
{ private int show = 10;
  public static void main (String [] args)
    {
```

obj. creation // InstanceVariable IS = new InstanceVariable();

code works // IS.show = 50;

obj. creation // Person Jayashree = new Person();

" // Person Billu = new Person();

```
        Jayashree.setAadhar(123456);
        System.out.println("Jayashree's Aadhar no is " +
                                Jayashree.getAadhar());
    }
}
```

Class variable :-

Associated with class

Every instance of the class shares a class variable

Any instance / object can change the value of class variable

can be accessed using instance (or) class name.

Even without the instance, we can use the class variable.

Use :- • static • public • private • protected • final
• transient

Sample Eg :-

```
Class classVariable
{                                   > if this keyword is removed code doesn't work
    private static int Max = 100;
    public static void main(String args[])
    {
        System.out.println("Max = " + max);
        System.out.println("Max = " + ClassVariable.max);
        ClassVariable CV = new ClassVariable();
        System.out.println("Max = " + CV.max);
        System.out.println("Min " + min);
    } System.out.println("Min" + Sample.min);
```

Static methods only allow access to static variables

To access non static variable, we use methods to access them

(To save the pgm the class where p.s.v.m. is there, we should use that as filename.java)

```
class sample
{
  private static int min = 100;
}
```

Local Variables :

Scope is inside a method or a block

Store its temporary state

Not accessible from rest of the class.

All local variables should be initialised (else errormsg)

Parameters :-

Are always variables not fields.

Eg:-

```
class LocalVariable
 {
  public void print()
    { int value = 123456;
      System.out.println(" value is " + value);
    }
  public static void main(String args[])
    {
      new LocalVariable().print(); // to print
```

equi to global variable

(or) — the object which is just created but no specific name is given

```
    LocalVariable L = new LocalVariable();
      L.Print();
    }
```

Name rules for variables:

Variable names are case sensitive

It uses unicode letters and digits (UTF-16 ⇒ default unicode used by JAVA) - std rep of letters & digits

Begins with letters, $, (or) _ (underscore)

No key words / reserved words can be used for naming

If there is a single word, the first letter should be a small one.

If there are two / more words, the first letter of the consequent words should be capitalised (underscores may be used in b/w)

In case of constant variable, we need to capitalise all the letters of the word (Eg: MAX)

Subsequent characters can be a letter, digit or underscore

JAVA:

JAVA is a language or technology that enables secure high performance, distributed and highly robust applications on multiple platforms in heterogeneous distributed networks.

(different hardware & s/w able to run together)

Design Goals of JAVA:

1) simple, object-oriented and familiar:

* simple: need not require extensive training
* Object-Oriented: Distributed, cls Systems, encapsulated message passing.
* familiar: Many of them Use.

2) Robust and Secure:

* to create reliable software
* extensive compile time check followed by run time check
* Memory management using "new" operator
* Automatic garbage collectors
* No pointers so no chance for memory leak
* It is secured from intrusion of virus

3) Architecture neutral and Portable:

* Byte code:
  Intermediate code designed to run on multiple hardware and software platforms.
* No datatype incompatability across s/w and h/w architecture
* JVM: platform dependent
  JAVA: platform independant

4) High Performance:

* Interpretor (default)
* JIT compiler (uses cache codes)

5) Threaded and Dynamic:

* Threaded: concurrent execution
* Dynamic: Dynamic classes are linked.
  using Class.forName(String ClassName)
  ↳ method

6) JAVA: static type checking
  Better readability (Boolean says True/False rather than 0s' and 1s' in C & C++)

VIRTUAL MACHINE:

It is like a software implementation of a physical computer which works like a real physical computer. (i.e) compile and run just like a physical computer.

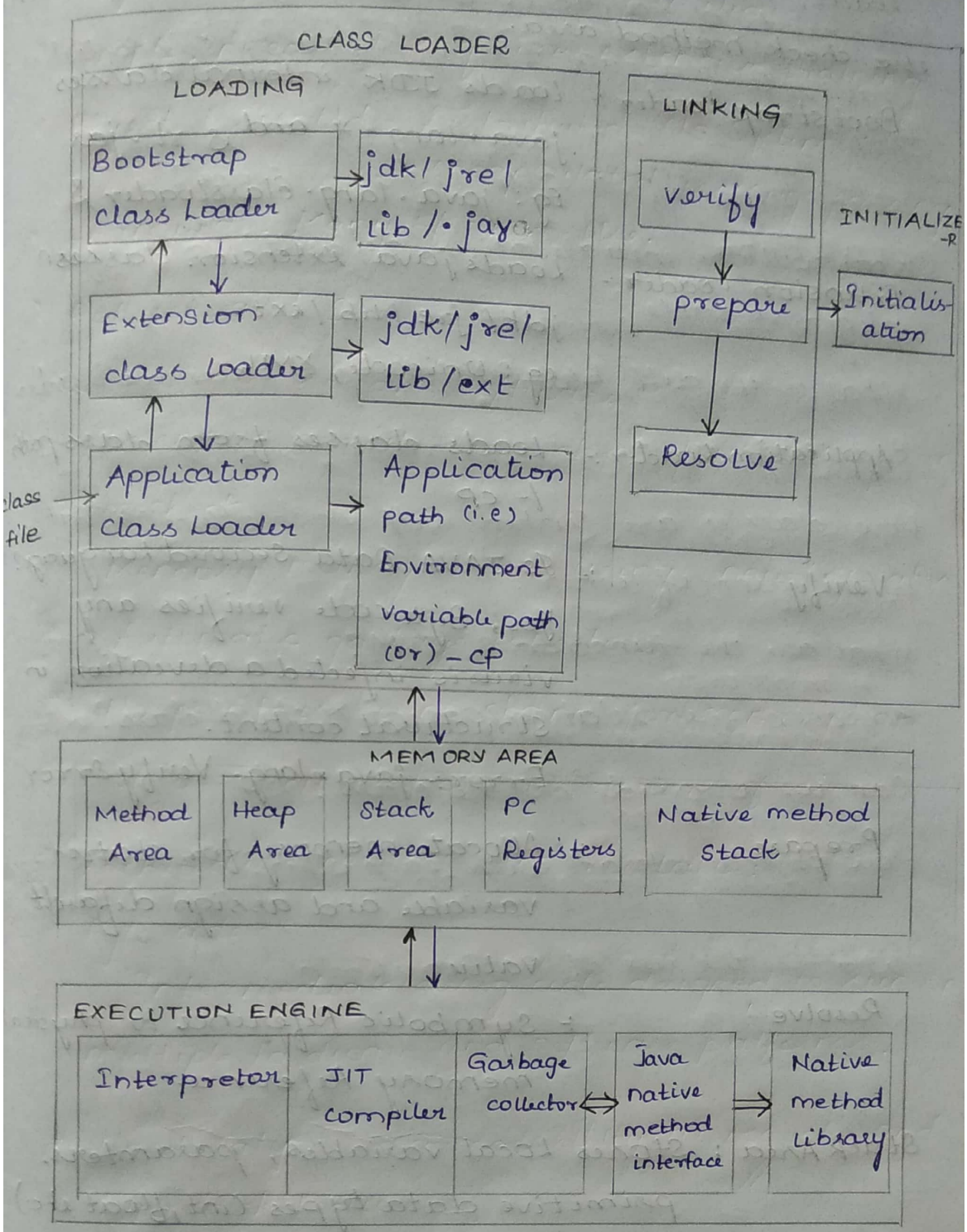
JVM: Java Virtual Machine

It is an abstract machine that provides run-time environment in which byte code can be executed.

JVM Operations:

* Loads the code
* Verifies the code
* Executes the code
* Provides runtime environment.

JVM ARCHITECTURE :



Native Method Stack :- Holds answers for who loads the class ? ans :- java.class.lang loade-r
↓
loads the Bootstrap loader.

Garbage collector :- Removes the unreferenced operators.

CLASS LOADER:

Loads, links and initialises the class for first time else check method area.

Bootstrap Loader :- Loads JDK internal classes java.lang.* & rt.jar
Eg: java.lang.ClassLoader &

Extension Loader :- Loads java extension classes jdk/jre/lib/ext.
eg: Unicode, XML

Application Loader :- Loads classes from class path /-cp

Verify :- Is JAVA Data secured Language?
Yes, Byte code verifies any virus is injected a deviation in structural content.
Error :- java.lang.VerifyError

Prepare :- Allocate memory for static variable and assign default value

Resolve :- Symbolic Reference to physical memory Reference.

Stack Area : Stores Local variables, paramters, primitive data types (int, float etc)

Method Area : Holds Class Level Information :-
* Class name
* Parent class name
* Static variables
* Methods name
* Instance variable.

Heap area : Stores Objects :- (creation by "new" operator)

PC registers :- holds current executing Instruction address

Initialiser :- All static variables are arranged with the original values.

Error msg :- (for :- lack of initialisation sym)
" java.lang. Linkage error "

Why use JAVA?

Is JAVA an 100% OOL? If yes state the reason.
If No, State the reason :-

What is JVM?

What is API?

What is the diff b/w C++ & JAVA?

What do you mean by class loaders in JVM?

Why main method in JAVA is static?

Can static public void main be used in code instead of public static void main? Will it throws error?

What is responsibility driven? data driven?

D/b cohesion & coupling :-

Design principles of OOP :-

What component does CRC contain?

Parnas principle :-

Can a static method access non static variables?

Can a static method invoke non static variable

Four access level of Instance variables :-

How do you implement global variables in JAVA :-

Access levels of local variables?

Can Local variables be used without initialisation?

Diff kinds of variables in JAVA :

What is the package that is defaulty supported in JAVA? java.lang

Which type of variable requires scope?

What will the initialisation (default) value of an integer variable?

What is the diff b/w Bootstrap & Extensive loader

Byte code is in binary form.

How classes are linked with each other? What is the func. of linker?

What are the function of JVM?

keyword for relating parent & child class? extends

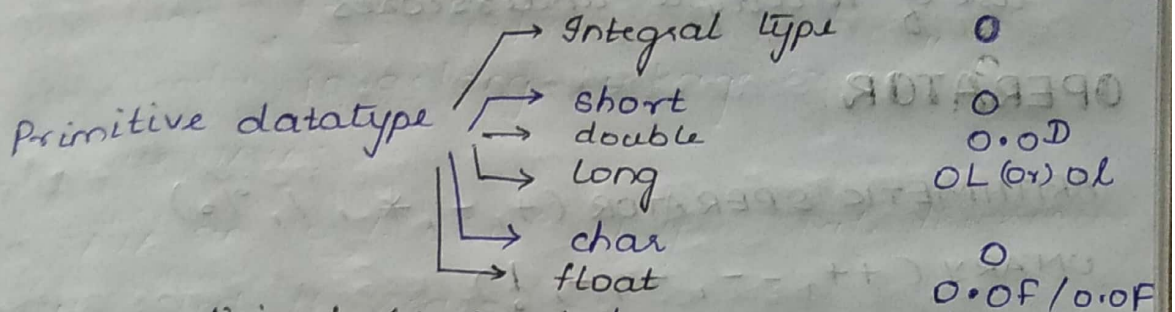
Can we override private class? No

How objects interact with each other? Passing Messages (Using Methods)

What is the use of PC Registers?

23/07/18

* Primitive datatype
  - integral type
    - bytes
    - short (2 bytes)
    - long
  - floating pt type
* Reference datatype



* JAVA uses Unicode (UTF-16) for representing the language letters

Byte code: machine independant; intermediate code that JVM can interpret

Apple's own virtual Machine ⇒ DVM (Dalvic Virtual Machine).

DVM uses AOT compilation (Ahead Of Time)

Android OS ⇒ Google.
IOS ⇒ Apple

(Oracle) (Google)
↑ ↑

// Identify the difference b/w JIT & AOT compilation and CLR (common Language Runtime) and DVM
↓ ↓
(Microsoft) (Apple)

Reference datatype:

- String is also an object
- Array also needs to be initialised as an obj:
- class → its reference is object

Underscores:- (DON'TS)

Before and after decimal pt

At the beginning and end of a no.

DO's:-

use of multiple underscores

OPERATOR

ARITHMETIC OPERATOR (+, -, *, /, %)

UNARY (++, --, +, -, !)

EQUALITY AND RELATIONAL (== ; >, <, >=, <= !=)

CONDITIONAL (&&, ||)

TERNARY (if (cond) ? ___ : ___)

↳ small prog

↳ enhanced readability

TYPE COMPARISON

Instance of : compares an object of specified type

Eg:

BIT-WISE AND AND BIT SHIFT :-

<< - shift left

>> - shift right

>>> - unsigned Rshift

~ - negation / compliment

& - bit wise "AND"

^ - EXOR

LOOPS :-

If .... then .... else

for (;;) - infinite loop

```
{
// unconditional stmt
}
```

"for each" loop :-

```
// declaration    int marks [];
// creation       marks = new int [10];
// Initialisation marks = {1,2,3,4,5,6,7,8,9,10}
// Initialisation marks [0] = 1;
// 'for each' loop : for (int item : marks)
// print stmt     System.out.println ("Marks | item" +
                                        item);
```

For eg :-

```
int marks []
marks = {1,2,3,4,5}
// Here the pgm. defaultly assumes the size as 5.
```

EXERCISE :-

1. Odd or even checking
2. Pass or fail checking
3. To print days of week (Switch case)
4. Fibonacci

(1)

```
import java.io.*;
import java.util.Scanner;
```

```
class oddeven
{ public static void main (String args [])
  {
  Scanner scan = new Scanner (System.in)
  System.out.println ("Enter no.");
  int a = scan.nextInt();
  System.out.println ("The no. is " +a);
   if (a %2==0)
    {
      System.out.println ("Even");
    }
   else
     System.out.println ("Odd");
  }
}
```

(2) To check Pass/Fail :-

```
import java.io.*;
import java.util.Scanner;
```

30/7/2018

1. What is autoboxing and unboxing in JAVA?
2. What are wrapper classes in JAVA?
3. Why do we need wrapper classes in JAVA?
4. Spot the difference b/w the statements:

   int i = 20;

   Integer iobject = new Integer (20);

5. What do you mean by immutable objects?
6. Are wrapper class instances mutable or immutable objects?
7. What is the diff. b/w equals() and == operators?
8. Where is the primitive data values stored?
9. Give the package name that contains wrapper classes: java.lang
10. Does void come under wrapper class?

30/7 WRAPPER CLASSES IN JAVA

int i=20; i ⇒ primitive datatype (primitive type)

Integer iobject = new Integer (20); ⇒ (object type)

↳ wrapper class

Wrapper class: Wraps primitive datatype as object

CONVERSIONS:

primitive → wrapper

wrapper → primitive

primitive → String

string → primitive

AUTOBOXING :

Conversion of primitive datatype ⟶ Wrapper class

AUTOUNBOXING:

conversion of Wrapper class ⟶ primitive datatype

WRAPPER CLASS:-

* Wrapper Class is a class that "Wraps around" primitive data type.
* Wrapper class is a part of java.lang package.

Why do we need Wrapper Class?

* To create objects, store as objects in collections (or) data structures such as Array List, Stack, Queue etc.

To assign null because primitive types can't be Null.

To achieve polymorphism, Synchronisation

To enhance the readability.

PRIMITIVE TYPES		WRAPPER CLASS
byte	Autoboxing	Byte
short		Short
int		Integer
long	Unboxing	Long
char		Character
float		Float
double		Double
boolean		Boolean

instance.method

AUTOBOXING:

converts the primitive datatype to the object of the corresponding wrapper classes.

Eg: int i = 20;
Integer iobject = new Integer (i);

UNBOXING:

converts object of the wrapper class to its corresponding primitive type.

Eg: Integer iobject = new Integer (20);
int x = iobject.intValue ();
instance.method();

class names → Always should begin with Capital letter

CONVERT

WRAPPER TO PRIMITIVE

Instance Method:

public primitive xxxValue() method.

(i.e)
public byte byteValue();
public short shortValue();
public long longValue();
public float floatValue();
public double doubleValue();
public boolean booleanValue();

Eg:
Character c = new Character('a'); ↳ object type
char cc = c.charValue(); ↳ primitive type
Boolean b = new Boolean(true);
boolean bb = b.booleanValue();
Integer i = new Integer(20);
byte b = i.byteValue();
short s = i.shortValue();
int a = i.intValue();

(using the object of wrapper class)

PRIMITIVE TO WRAPPER

Static Method:

public static valueOf(primitive) method.

⇒ returns an instance of wrapper of specified argument primitive type

Eg:
Boolean b = Boolean.valueOf(true);
Byte bb = Byte.valueOf(128);
(Not possible)
Number format Exception
↳ value exceeds 127
(Byte range is only till 127)
Byte b = Byte.valueOf(127);
Integer i = Integer.valueOf(1245);

(using the class name by itself)

128 - can't be stored in Byte range
Byte range → till 127
Number format exception

STRING TO PRIMITIVE

Static Method:

public static primitive parseXxx(String) method

⇒ convert string into the respective primitive type

boolean b = Boolean.parseBoolean("true");
o/p ⇒ true ↵
boolean b = Boolean.parseBoolean("TRUE");
o/p ⇒ TRUE ↵
boolean b = Boolean.parseBoolean("Abc");
o/p ⇒ False ↵
Float f = Float.parseFloat("10.25f");

PRIMITIVE TO STRING

STRING MANIPULATION

java.lang.String

Immutable

METHODS: Examine individual characters, words,
- Compares String
- Search String
- Extract
- Create
- Case mapping (upper case, lower case)

char charAt (int index)

Returns char at specified index.

int compareTo (String another String)

compares 2 strings lexographically.

int compareToIgnoreCase (String str)

String concat (String str)

boolean contains (CharSequence s)

String copyValueOf (char [] data)

String copyValueOf (char[] data, int count)

boolean equals (Object O)

boolean equalsIgnoreCase (String S)

boolean isEmpty()

String join (char sequence delimited, char separate elements)

int lastIndexOf (int ch) character

int lastIndexOf (String str)

int lastIndexOf (int ch, int fromIndex)

int lastIndexOf (String str, int fromIndex)

String replace (char old char, char new ch)

String replace (char sequence target, Char Sequence replace)

String replaceAll (String regex, String replace)

String replaceFirst (String regex, String replace)

String[] split (String regex)

boolean Starts With (String prefix)

boolean Starts With (String prefix, String offset)

String Substring (int beg index)

String Substring (int beg index, int end index)

char[] to charArray()

String toLowerCase()

String toUpperCase()

int indexOf (int ch)

int indexOf (int ch, int fromIndex)

int indexOf (String S)

Index of : checks the first occurance of the character in the string.

split : used for counting the length of the string.

s.split("....") ⇒ The word is split into individual characters and stored in an array.

Write a JAVA program for :-

1) Reversing a string
2) Concatenating two strings
3) Check whether the given string is a palindrome or not :
4) No. of words present in a given word.
5) To remove the duplicated words
6) E-mails validation.
7) Date Validation.
8) To remove white spaces
9) To sort the given words in a string.
10) To find the occurence of word in a given string
11) Write a program to replace the given character / string
12) Write a program to remove a particular char from a string

13) To search for a given word in a string

14) Replace the occurence in a string

15) Capitalise the first character of each word in a str

16) To compare 2 strings using (i) compare method
(ii) equals to method
and check the result.

17) To count no words in the given string.

18) To calculate no. of characters in a given string.

19) To validate the given date formatters.

20) To match the phone numbers to a given pattern
(Landline -044- ; Cell -+91....; etc)

02/08/2018

CLASS, METHODS, CONSTRUCTORS, STATIC:

CLASS:

Declare Class:

```
class myclass
 {
    // fields, constructors,
    // Method declaration
 }

class    class extends    class implements class
 {
    // fields, constructors
    // Method declaration
 }
```

MODIFIERS : doubt

public, private, protected.

CLASS NAME :

Noun

first letter is capitalized

MEMBER VARIABLES :

member variables ⇒ Fields

Local variables —used in→ (Methods)

Parameter → (Methods)

STATE OF THE CLASS : member variables

BEHAVIOUR OF THE CLASS : methods of a class

TYPES :

All variables have their own data type

Primitive type :- int, float...

Reference type :- Integer, String, ...

VARIABLE NAMES :

follow naming conventions.

METHODS :

Modifiers (all methods must posses modifiers)

Return type (all methods must have return type)

Method name — Verb

For two words — first word - verb

second word - Adjective (or) Noun.

First letter - capitalised.

use super() key word to call default constructor of parent class

Con :-
→ Overloading ✓
→ Overriding ✓
→ Inherited by sub classes

A class can't create an instance of itself

Constructors :
→ can be private
→ can't be static, final, abstract

default :- constructor returns an object of class type

no return type ⇒ constructor cannot return any value

'new' keyword invokes constructor but with return type

class name

Methods can also have same names as class

CONS → initialise the numbers of initialise instance var. of a class (objects)

CONSTRUCTORS:

To create objects

It is a block of code that initialises the newly created object.

It has the same name as the class name.

Eg1 public class Myclass
```
{
    Myclass()
    {
    }
}
```

Eg2:- public class Student
```
{ int rollno;
  String college name;
  Student()
  {
    this.college name = "MIT";
  }
}
public static void main (String args[])
{
  Student S = new Student();
  System.out.println (S.collegename);
}
```

Key points regarding Constructors :-

Types of constructors :

No arguments
Default
Parameterised

Default constructors:

JAVA compiler inserts the default constructor

No arg constructors:

Constructor with no arguments
Signature is same as the default constructor

Parameterised:

Constructor with parameters

```
public class Student
{
    int regno;
    String name;
    static int count = 0;
    Student ()  ⇒ default constructor
    {
        count++;
        System.out.println (count + "objects created");
    }
    Student (int regno) ⇒ single arg. constructor
    { count++;
      this.regno = regno;
      System.out.println(count + "object created");
    }
    Student (int regno, String name)
    { count++;
      this.regno = regno;
      this.name = name;
      System.out.println (count + "object created");
    }
    public static void main (String args [])
    {
        Student S1 = new Student ();
        Student S2 = new Student (101);
        Student S3 = new Student (101, "xyz");
        System.out.println (S1.regno);
```

invokes

invokes

No arg constructor is mandatory

If no arg. constructor] param is used ⇒ default will be used

(one more constructor of the same type is not possible.

System.out.println(s2.regno);
System.out.println(s3.regno);
}

Key pts:-

Defaultly:-
Compiler specifies the first line of child class with "super()"; by mod

this("one")
↓
single para. const

this(2, "two")
↓
2 para constn

this - oper
↓
specifies current operator value

parameterised:-

CONSTRUCTOR CHAINING: to be called explicitly
constructor calls another constructor

⇒ If parameter & instance have same name

```
public class MyClass
{
    int value;
    String variable;
    MyClass()
    {
        this("one");
    }
    MyClass(String s)
    {
        this(2, s);
    }
    MyClass(int value, String ss)
    {
        this.value = value;
        variable = s;
    }
    public static void main()
```

2

```
MyClass Obj = new Myclass();
}
```

Keypts:-

Singleton class => only one object can be created in a class.
→ Utilises private constructor
→ 2nd time it returns all the created obj.

Super :-

```
class Parent
 {
   parent (ints)
    {
      System.out.println ("Parent Class");
    }
 }
 class child extends Parent
   {
     child ()
      {
        super ();
        System.out.println ("Child class");
      }
   }
```

Keypts:-

Private constructor :

permits us to create the objects inside the class but outside the particular ~~ob~~ class, creating an object of that class is restricted

Creates Singleton class :-

```
class Myclass
```

STATIC:

Static methods are invoked by class names.

Used to initialise all the static variables of a particular class.

No static constructor exists

CONSTRUCTORS (QUICK FACTS):

Every class has a constructor

The Abstract class can also have a constructors

Constructors don't have return type

Constructor name should be the same as classname

Constructor can use any access specifier (public/ private)

Methods can also have same name as the class name but they have a return type.

The constructor within the class can be invoked by using this() operator

The parent class constructed can be invoked by using super() operator

Default constructors are provided by the compiler itself.

Constructor overloading is possible while overriding is not possible

Constructors cannot be inherited but can be invoked by using "super" keyword

Interfaces don't have constructors

constructors can't be Abstract, Final & Static

Sample code snippet for "Static" :-

```
public class Example
{
  static
    {
      System.out.println ("First, static Block");
    }
  public static String staticString = "Static variable";
  static
    {
    System.out.println ("Second static block" +
                        "and" + Static String);
    }
  public static void main ( )
  {
    Example e = new Example ( );
    Example.static method ( );
  }
public static void static method ( )
  {
    System.out.println ("this static method is
       invoked using class ");
  }
static
```

of type ← variable name

modifier

calling fn.

```
{
"no space staticmethod2();
System.out.println("Third Static Block");
}
public static void staticmethod2()
{
system.out.println("Static Method");
}
}
```

→ object of a class not needed for calling static methods

STATIC BLOCKS

* They are used to initialise static fields of a class

When used?:

* It is used to execute the static blocks once when the class is first loaded.

When not used?:

* Static blocks cannot exceed 64 KB (lines of code)
* Cannot create exceptions
* Very difficult to debug.
* "this" keyword is not permitted for usage
* No return
* Testing is a nightmare.

COPY CONSTRUCTOR :-

Constructor having the class name as a parameter.

Eg:

```
public class Example
{   int w;
    :
}
Example ( Example e2)        // It is of type class Example.
{   this.w = e.w;
}
public static void main()
{
   Example e = new Example();
     e.w = 10;
   Example e1 = new Example (e);
}
```

// the above process is similar to assigning e1 = e;

// It utilises the use of constructors rather than assigning directly.

9/8/18

Dynamic memory allocation ⇒ Heap

Static memory ⇒ Stack

INTERFACE IN JAVA :-

WHAT?

It is a blueprint. It contains fields and methods

fields : By default - public, static, final
                ↳ constant

methods : By default - public, abstract.
                ↳ abstract.

Interface is like to class

JAVA 8 Supports ↳ default methods (has contents)
                ↳ static methods (has contents)

WHY we need an INTERFACE?

Interface contains only the name of the method. Implementation can be done in various classes based on the user's will.

(+) [Abstract method - empty method - having no coding part
    [Implementation method - must possess the keyword "default".

↓

Abstract class ← (Atleast one)

Implementation Over Inheritance :

JAVA doesnot support Multiple Inheritance
This can be made possible using Interface; its Ambiguity is overcome

WHEN interface is NEEDED?

To acheive : Abstraction
             Multiple inheritance
             loose coupling.

An Interface decreases the tight coupling, dependancy.

HOW to implement interface?

```
interface <name>
 {
    fields
    abstract methods();
    default methods();
    static methods();
 }
```

Sample Examples:-

SIMPLE INTERFACE:-

```
interface Example
{
   int Max = 100;        ⟶ It is a constant value; cannot
   void display();          change the value.
}
class Sample implements Example
{
  public void display()
    {
      System.out.println("Interface Method in Class
                                    Sample");
      System.out.println("Max" + max);
    }
}
class Test
{
   public static void main (String args[])
    {
/*  Example e = new Example()      We cannot create objects for interface
      e.display(); */ — will not compile error
```

```
Example e = new Sample();
    e.display();
  }
}
```

methods → can be called without instance
static → no instance is required
default → instance is a must

Interface can't be instantiated

STATIC METHODS : (default)

```
interface Example
{
  int Max = 100;
  static void display()
   {
    System.out.println("I am an Interface Method");
   }
  default void show()
   {
    System.out.println("Show is interface");
   }
}
class Sample implements Example
 {
  :
 }
 class Test
  {
   public static void main()
    {
```

Like higher datatype → lower datatype

Assignment ✗ `Example e = new sample();`

`Example.display();` → calling the static function. Function

called by the class name. Static methods → is not necessary to create instances (say objects)

```
Sample s = new Sample();
    s.show();  → access the default method
    e.show();
}
```

it is essential to create an instance for accessing the method.

Access the method show by using the (Example (e)) variable (concept III law to type cast

Example :-

Class → class - inheritance (extends)
class → interf - interface (implements)
inter → inter - extends

```
import java.io.*;
interface Vehicle
{   int speed = 140;
    void changeGear(int a);
    void speedUp(int a);
    void applyBrakes(int a);
}
```

Abstract methods

```
class Bicycle implements Vehicle
{
    int speed1;
    int gear;
    public void changeGear(int newGear)
    {
        gear = newGear;
    }
    public void speedUp(int increment)
    {
```

```
    speed1 = speed + increment;
  }

public void applyBrakes(int decrement)
  {
    speed1 = speed - decrement;
  }

public void printStates()
{
  System.out.println("Speed" + speed1 + "Gear" +
                                          gear);
  }
}

class Bike implements Vehicle
{
  int speed1;
  int gear;

  public void changeGear(int newGear)
  {
    gear = newGear;
  }

  public void speedUp(int increment)
  {
    speed1 = speed + increment;
  }
public void applyBrakes(int decrement)
 {
   speed1 = speed - decrement;
 }
```

```
public void printstates()
{
  System.out.println("Speed"+speed1+"Gear"+
                                    gear);
}
}
class Sample
{
public static void main(String[]args)
  {
     Bicycle bicycle = new Bicycle();
     bicycle changeGear(2);
     bicycle SpeedUp(3);
     bicycle applyBrakes(1);
     System.out.println("Present state of bicycle");
     bicycle.printstates();
     Bike bike = new bike();
     bike changeGear(2);
     bike speedUp(3);
     bike applyBrakes(1);
     System.out.println("Present state of Bike");
     bike.printstates();
  }
}
```

Sample Program 2: (Loose C

```
class Teacher
{
    Smart Student SS;
    Lazy Student LS;

public Teacher (Smart Student S, Lazy Student L)
    {
        this.SS = S;
        this.LS = L;
    }
    public void test ()
    {
        SS.exam();
        LS.exam();
    }
}
class Smart Student
{
    public void exam()
    {
        System.out.println("Smartie");
    }
}
class Lazy Student
{
    public void exam()
    {
        System.out.println("Lazy Student");
    }
}
```

S is of type smart student

Constructor // takes in object of the class members SS, LS

IStudent - reference (can hold EOS / SS / LS objects

The above program is an example for tight coupling. To incorporate another class called Extraordinary student we need to alter the "Teacher class", "Test class". To overcome this we go for "Interface" → facilitates "Loose Coupling"

```
class Test
{
  p.s.v.m (String args[])
  {
    Smart Student S = new student();
    Lazy Student L = new student();
    Teacher T = new Teacher(S);
    Teacher T = new Teacher(L);
    To
```

13/08/2018

INHERITANCE : is-a relationship

Single level inheritance

Why Inheritance?

Enhance the reusability of the modules.

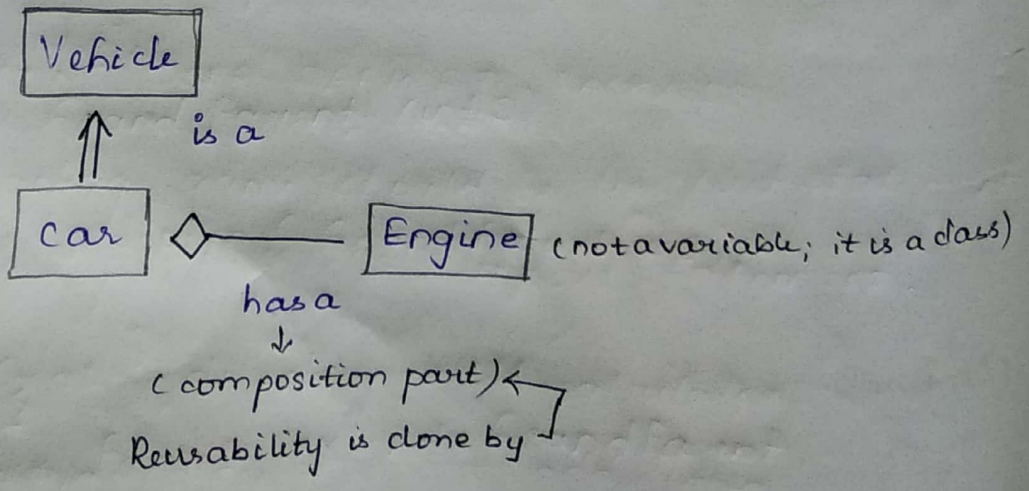
Involves an is-a relationship.

Inheritance can involve its new methods and methods inherited from parent class

Final:- It cannot be overridden

Fields:- can be inherited.

Static members:- cannot be inherited.



```
class vehicle
 { void startup()
    { System.out.println("Vehicle Started");
    }
   void stop()
    {
     System.out.println("Vehicle stop");
    }
 }
class car extends vehicle
  {
    Engine E;
    int speed;
    void start()
     {
       System.out.println("Car");
       E.new Engine();
       E.start();
     }
   void stop()
    {
      System.out.println("Car stopped");
    }
   void speedup(int s)
    {
      speed = speed+s;
      System.out.println("Speed of a car "+speed);
  }  }
```

```
class Engine
  {
    void start ()
      {
        System.out.println ("Engine Started");
      }
  }

class Test
  {
    Vehicle V = new Car();
    Car c = new car ();
    c.start ();
    c.stop ();
  }
```

Child class can't refer to parent but parent can refer to child.

Eg. for Multi level Inheritance:

new operator ↓ encountered
Searches for constructor
constructor ↓ checks for default constructor.

```
class GrandParent
  {
    GrandParent ()
      {
        S.O.pn ("GrandParent");
      }
    GrandParent (int s)
      {
        S.O.pn ("GrandParent" + s);
      }
  }
class Parent extends GrandParent
  {
```

```
Parent()
{ super();
  S.o.pln("Parent");
}
Parent(int s)
{ s.o.pln("Parent"+s);
}
}
class child extends Parent
{
  child()
  {
    super(10);
    S.o.pln("child");
  }
  public static void main(String args[])
  {
    child c = new child();
  }
}
```

16/08/2018 **PACKAGES**

→ JAVA doesn't support "FRIENDS" concept
→ This is made possible by using packages.
→ package ⇒ for namespace management
→ Advantage:
  ° Easy to access.
  ° To overcome name conflicts (confusions with same names)
  ° To enhance the protection
  ° To restrict modifiers

PACKAGE:

packages
interface

Package is a grouping of related types (classes, Interfaces, packages) providing access protection and namespace management.

WHY?

Organise related classes into groups.

Eg:- Similar to directories; and folders in harddisk

Prevent name conflicts.

(Reduce problems with conflicts in names)

Allows protection to classes, variables and methods

Easy to access or identify classes as all those classes have unique identifiers. (Package name)

Existing packages: java.lang, java.io

PACKAGE NAMING CONVENTIONS:

First letter should not be special symbols, numbers, underscores etc.

All the letters in the name of the package should be in lowercase

CREATING USER-DEFINED PACKAGES:

Package name should be the first statement (i.e) even before the import statement.

Create a sub directory called calculator.

c:\java\calculator ⟶ add.java

```
package calculator;
import java.io.*;
public class Add
{
    int a=10;
    public int b=20;
    private int c=30;
    protected int sum;
    public void addition()
    {
        sum = a+b+c;
        System.out.println("Sum= "+sum);
    }
}
```

Provide Access b/w different packages

// for importing the package :

```
class TestPack
{
    public static void main(String args[])
    {
        calculator.Add add0 = new calculator.Add();
        add0.addition();
    }
}
```

← no package = default package

package name ←

// We can access the package by using qualified name.

```
class Test extends add
{
    ⋮
}
```

↳ to inherit we use this

↑ If class is declared as protected

• * - to access all the classes in the package.

```
class TestPack
{
    psvm(String args[])
    {
        Add add0 = new Add();
        X add0.a = 20; → Error (both are 2 diff. classes)
    }
}
```

default package can use the protected only when inherited

ACCESS MECHANISM

	PRIVATE	DEFAULT	PUBLIC	PROTECTED
Same class	Yes	Yes	Yes	Yes
Same package subclass	No	Yes	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non subclass	No	No	Yes	NO
Same package non subclass	No	Yes	Yes	Yes

JAVA	C++
no switching all are .java files	switching b/w .h and <.cpp> extensions
use of packages	use of friends

```
package calculator;
import java.io.*;
public class Add
{
 int a=10;
 public int b=20;
 privateint c=30;
 protected int sum;
 public void addition()
 {
  sum = a+b+c;
  System.out.println("Sum = "+sum);
 }
}
package calculator;
import java.io.*;
public class Sub
{
  int a=10;
  public int b=20;
  private intc =30;
  protected int dif;
  public void subtraction()
  {
    dif= a-b-c;
   System.out.println("Diff = "+dif);
  }
}
```

```
package calculator;
import java.io.*;
public class Mul
{
    int a=10;
    public int b=20;
    private int c=30;
    protected int mul;
    public void multiply()
    {
     mul = a*b*c;
     System.out.println("Mul = " + mul);
    }
}
```

```
Package mypackage;
import calculator.*;
class Testpack
{
  public System void main(String args[])
    {
      Add.add0 = new Add();
        add0.addition();
      Sub.sub0 = new Sub();
        sub0.subtraction();
      Mul.mul0 = new Mul();
        mul0.Multiply();

    }
}
```

Use of FINAL :

final variable ⇒ prevent changes
final method ⇒ prevent overriding
final class ⇒ prevent inheritance

Eg:

```
class Parent
{
protected int a = 10;
final String str = "Parent";
final void display()
  {
    S.o.pln (str);
  }
}
class Child extends Parent
{ final String str = "Child";
  void display()
    {
      S.o.pln (Str);
    }
public static void main (String args[])
  {
    Parent pobj = new Parent();
    Child cobj = new Child();
    Parent Ref; → create a reference variable
    pobj.str = "not parent";
    Cobj.str = "not child";
    S.o.pln ("a from parent access is " + (obj.a));
    Ref = pobj;
    S.o.pln (Ref.Str);
  } Ref = cobj; → Run time polymorphism
```

Cannot override

final → method overriding

Hiding / Shadowing → Variable having the same (i.e) constant name as the parent

Pref : given to child : variable

Default modifiers can be accessed within package

final → NC cannot change the value.

child can access the parent variable.

RTP → implicitly caste over

Sto CTP - Method overload

HW

class employee
class director
class actor
inherit the properties

Error stmts Because str is declared as "final" (its constant)

Diff b/w JAVA and C++

JAVA	C++
Memory management: • No explicit garbage collector • No pointers in JAVA. • All objects make use of Heap (stack only for variables)	
Inheritance: • No multiple inheritance in JAVA • Instead supported by Interfaces (No multiple inheritance - avoids ambiguity)	
File / Project Management: • No friends operator • Instead it has Packages • No .h files • Instead all files are stored as .java extensions	
Environment Issue :- • No machine dependant. • No machine codes instead we have byte code (architechud neutral) • No weak type in JAVA instead it has strong data type	
Interpretations: Supports Interpretationst Compilation (No separate compilers)	