1/ Thinking Object Oriented:

Consider the two assertion (or two propositions illustrates)

(a) (i) Conflicting objectives

1) OOP is a revolutionary idea, totally unlike anything that has come before in programming.

2) OOP is an evolutionary step, following naturally on the heels of earlier programming abstractions.

Both are true

Note:

Fundamental features of OOPS were invented in the 1960's but OO languages came to the public attention only at 1980's. - 1st International Conference on OO Programming language & applications held at Portland in 1986, in that Smalltalk PL was described.

(2) Why OOP is so popular?

(a) 1) Proven record of success
2) Scales well from small problems to large ones

(b) The desire for an easy transition from procedural language (i.e C to C++, Pascal to Object Pascal, Fortran to Smalltalk, Basic to Visual Basic)

(c) Resonant: similarity to techniques for thinking about problems in other domains.

(b) (1) Language & Thought

→ Sapir - Whorf hypothesis, that says there are some thoughts you can express in one language that you can't express in another!

B) 1.1 → Eskimos & Snow → Inuit can think more intelligently about snow because their language contains more sophisticated words distinguishing various forms of it.

=) Eskimo language have many words to describe various types of snow
>50 words - wet, fluffy, heavy etc
=) Any community can find/develop vocabulary to discuss
But, language (English) what I speak does not force me into doing so & so it is not natural to me.

(cannot find/lack of equivalent words in other language, likely even in computer language)

=) ∴ a different language Inuit can lead one to view the world in a different fashion.

∴ Using 'OO' PL does not by itself force one to become an OO programmer. but 'OO' PL will simplify the development of object-oriented solution.

b) 1.2 An example from computer languages.
eg: Fortran vs APL

→ Fortran programmer was blinded by a culture that valued loops, simple programs

→ Sorting is a built in operation in APL, good programmers try to find novel uses for sorting

i.e language you speak leads you one direction or another

(that there are certain thoughts of an individual in one language that cannot be understood by those who live in another language)

Sapir-Whorf hypothesis =) our language (structure of a) influences or affects its speaker's world view or cognition or thoughts (the way people think is strongly affected by their native language)

Example to illustrate the Relationship between Computer language & problem solution

Fortran:

DNA Research student had the task of finding repeated sequences of 'M' values in a long sequence of values: IN FORTRAN

ACTCGGATCTTGCAT

write a simple (efficient?) program

(View data as array of data)

```
   DO 10 I = 1, N-M
   DO 20 J = 2, N-M
   found =. True
   DO 20 K = 1, M
20 If X[I+K-1] .NE. X[J+K-1]
   THEN found = .FALSE
   If found THEN ...
10 CONTINUE
```

Took a long time !! ie $O(M \times N^2)$

Final Conclusion: All thought is constrained by language, has been disproved; language does not influence thought at all - also false

Better solution:

APL Student found better solution by rearranging data & sorting

(View data as Matrix of rows & columns)

A C T C G A	1 to M
C T C G G A	2 to M+1
	3 to M+2
	6 to M+5

TAGACC
TAGACC

Ran quickly !! ie $O(M \times N \log N)$ computational power that permits problem

to be solved that cannot, in theory be solved by other means. But OO techniques do make it easier & more natural to address problems in a fashion that tends to favor the management of large S/w projects.

New paradigm:

OO programming referred to as a new programming paradigm.

Church Conjecture

In computation we have the following assertion:

Church Conjecture:

- Any computation for which there exists an effective procedure can be realized by a Turing M/c language.
- Anything can be done in any language, but it may be simply be easier or more efficient to use one language or another.

Willing to write Even-driven GUI interface in Turing M/c ?

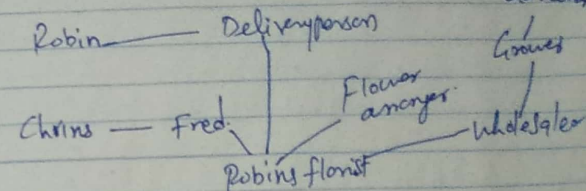
Imperative programming

→ Traditional model of Computation
- = State
- = Variable
- = Assignment
- = Loops

Processing unit is separate from memory & acts

Illustration of OOP Concepts - sending flowers to a friend (who lives in a different city).

Chris is sending flower to Robin.



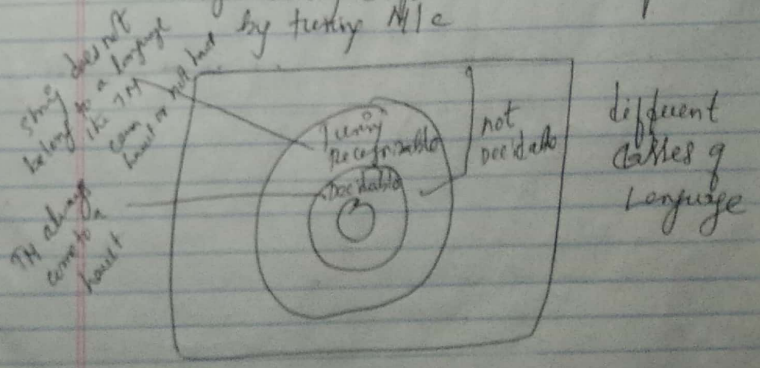
Each object has a part to play, a service they provide to the other Members of the Community.

Church-Turing Thesis

What are computable?

→ it there is a algorithm to solve a problem that can be implemented on a TM
A TM can compute anything that can be computed.

Algorithmically computable means computable by turing M/c



Java programming

Basic information

1) Comments
   - multiline Comments :) `/* */`
   - Single line ⇒ `//`

2) Statement
   - Each line terminated by ⇒ `;`

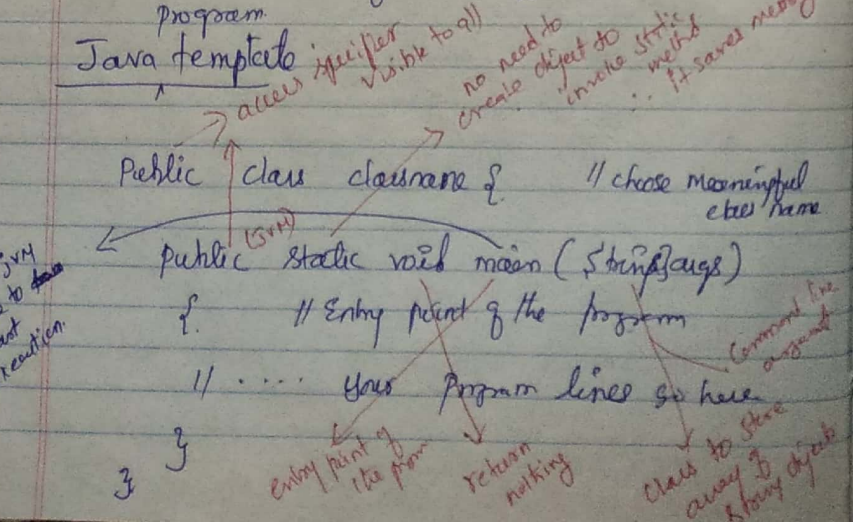
3) Block
   - Group of statement ⇒ { } (pair of braces)

   Note: No need to put semicolon after the braces
   Two blocks = one for class, other for main method

4) Case Sensitivity
   - Java is ⇒ Case Sensitive
   - ie filename ⇒ Case Sensitive
   - Variable ⇒ "
   - ie 'Jaya' is different from 'JAYA'

Program

Java template



5) Output ⇒ class in java.lang package ⇒ static member of the System class

   `System.out.println()` method

   ⇒ prints the given string and advances the cursor to the beginning of the next line.

   `System.out.print()`

   ⇒ prints the given string but places the cursor after the printed string

out is the static member of class System of type PrintStream & used to access println method of PrintStream class

```
class System
{
    static PrintStream out;
    static InputStream in;
}

class PrintStream
{
    public void println()
    {
    }
}
```

eg:

```
Public class Print {
    P.S.V.m (String[] args) {
        System.out.println ("Hello Jaya!");
        System.out.println(); // print empty line
        System.out.print ("Hello chitra");
        System.out.println ("welcome");
        System.out.print (" "); // print a space
        System.out.print ("CT");
    }
}
```

Output:

```
Hello Jaya!

Hello chitra welcome
CT
```

main: → When the program starts running, it has to start execution from Somewhere. That Somewhere is called main.
→ it is called once.

is main method Compulsory:
Yes, you can Compile java class without main but you can't execute without main method.

Can main method be private?
→ It will compile but 'java' interpreter looks for public main method signature.
∴ it's tell us where to start an application

Can we have more than one main method in a class:
A class can have only one main method.

Static:
⇒ access specifier
⇒ tells JVM to access main method without creating an instance of it.

Public:
→ To be found by JVM when the class is loaded

Can a class be private or protected ⇒

String:
→ String is a class
→ All class in java begins the first letter caps
→ Command-line arguments as an array of String objects.
ie java Sample one two

→ top level class can not be private or protected
it can have either public (or) no modifier
if no modifier it is default access.

Void ⇒ main method does not return any value

Can we override main method

No - because static method is a class method & the scope of this method is within the class itself
∴ override is only for instance method

Method Overloading:

i) Three ways to overload:-

a) no of parameters

add (int, int)
add (int, int, int)

b) data type of parameters

add (int, int)
add (int, float)

c) Sequence of Data type parameters

add (int, float)
add (float, int)

Invalid case of Method overloading

⇒ return type: ⇒ same parameters but different return type is not valid method overloading. This will throw compilation error.
ie int add (int, int)
float add (int, int)

Note:
→ Method overloading also known as static polymorphism.
→ ie compile time binding or early binding
→ static binding happens at Compile time.

input stream => System.in

output stream = System.out

Static vs non static method:

```
import java.io.*;
class StaticDemo
{
  p.static void method1(String s1, String s2)
  {
    s2 = s1;
    S.o.println("First string " + s1 +
                "second string " + s2);
  }
  p.s.v. main(String[] args)
  {
    method1("abc", "xyz");
  }
}
```

Static method

non static method

```
class nonStaticDemo
{
  Public void method1(String s1, String s2)
  {
    s2 = s1;
    S.o.p("First string" + s1 + "second string" +
          s2);
  }
  P.s.v. m(String[] args)
  {
    nonStaticDemo obj = new nonStaticDemo();
    obj.method1();
  }
}
```

java.lang => String class
System, Double, Float, Integer class etc.
Boolean, Byte, Character

Method overloading:

java.util.* => Scanner class

```
class Overload
{
  Public int method(int n1, int n2)
  {
    S.o.pn("method overload");
    return n1+n2;
  }
  Public int method(int n1, float n2)
  {
    S.o.pn("method overload");
    return n1+n2;
  }
  p.s.v. main(String args[])
  {
    Overload obj = new Overload();
    obj.method(2,2);
    obj.method(2,2.0f)
  }
}
```

Type promote byte → short → int → long
int → long → float → double
float → double
long → float → double

int m1 (int a, int b) ;
int m1 (int c, int d) ;

Compilati argument list are same type, same number & same sequence.

int m1 (int a)
float m1 (int a)

Compile time error — same method name, type, same no. of argument, same type, same difference
∴ → Return type is not considered in method overloading

Method overriding:

Method in subclass is similar to the method in parent class

Method in parent class ⇒ Overridden method

" child class ⇒ overriding method

```
class person
{
  public void speak ()
  {
    s.o.pn ( " person is speaking");
  }
}

class student extends person {
   public void speak () {
     s.o.pn ( " student is speaking");
   }
}

p.s.v.m ( String args[])
{
  student obj = new Student();
  obj.speak();
  Person obj1 = new student();
  obj.speak();
  Person obj2 = new Person();
  obj2.speak();
}
```

(Person obj1 = new student(); → parent class reference refers to the child class object. Then the overriding method (child class method) is called)

runtime polymorphism. i.e during runtime parent class method or child class method to be executed is determined by the object type

Output
Student is speaking
Student is speaking
Person is speaking

Object-Oriented Design

Object-Oriented program:

Key idea of OO Design

Community of objects

→ is like a community of interacting individuals, each having built, ned responsibilities, working together towards the attainment of a common goal.

Why start with Design?

a) Why start on Design?
b) Object-oriented thinking begins with Object-oriented design

obligation to perform a task or know information

Key Concepts

⇒ Responsibility Driven design

is a way to design using objects, responsibilities & collaborations

is a Design technique driven by the specification & delegation of responsibilities developed by practitioners of OO design.

interaction of objects

→ Major aspect in the design of such community interaction is: Determine the specific responsibilities for each member.

(i) Responsibility implies a degree of independence or non interference

ie tell a child that he/she is responsible for cleaning her room, having issued a directive you need not watch over her while the task is being performed.
having issued a directive in the correct fashion, the desired outcome will be produced.

Conventional program vs OO program

Conventional program	OO program
→ data-driven dependent-design ⇒ ie s/w intimately tied by control & data connections, to many other sections of the system. [Use data to determine the control flow]	→ Responsibility-driven design. [class define behaviours along with data.]
→ Centralized controlled system	→ delegated control

Programming in the Small & programming in the Large:

(S) Programming in the small	(L) Programming in the Large ("Community of objects collaborate in defined ways to fulfill the goals")
1) One programmer, understands everything from top to bottom	1) system is developed by large team of programmers
2) Major problem is the development of algorithms	2) Major problems are ⇒ management of details ⇒ Communication between programmers & their respective s/w subsystems

Basis for Design: → look for "noun" phrases in the problem statement

(i) Identify Candidate Objects that model a system as a set of abstractions

(ii) Determine the responsibility of each object. ⇓ verb, action

⇒ what an instance must be able to do & what each instance must know about itself

public responsibility ie an object offers service to others
private responsibility ie object takes in support of public responsibility

OO Design principle

Single responsibility principle.

→ Classes should have a single responsibility

Why?

→ Cohesion when high, reduces complexity, makes the system more understandable.

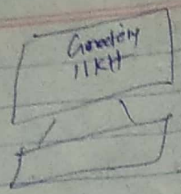
- Maintenance: Fixing or changing a module should not break other parts of the system

Cohesion
→ Strength of relationship / interdependence within module
→ High cohesion: single class with single function or task

Coupling
→ Strength of relation between module
→ Low coupling: Through interface

Example IIKH [Intelligent Interactive Kitchen helper]

Develop a S/w to implement IIKH that will replace the box of index cards of recipies in the kitchen



Abilities of the ISKH : User can do with ISKH.

1) Browse a database of recipies
2) Add a new recipe to the database
3) Edit an existing recipe
4) Plan a meal consisting of several courses
5) Scale a recipe for some number of users
6) Plan a longer period, say a week
7) Generate a grocery list that includes all the items in all the menus for a period

Characterising by behavior:

(i) Responsibility Driven Design will first characterize the application by behavior.

Software Components (Objects may become classes)

(ii) Identify the software component with associated Responsibilities

⇒ A component must have a small well defined set of responsibilities

⇒ A component should interact with other components to the minimal extent possible

(Collaborators)

(b) CRC cards (Candidates, Responsibility, Collaborators)

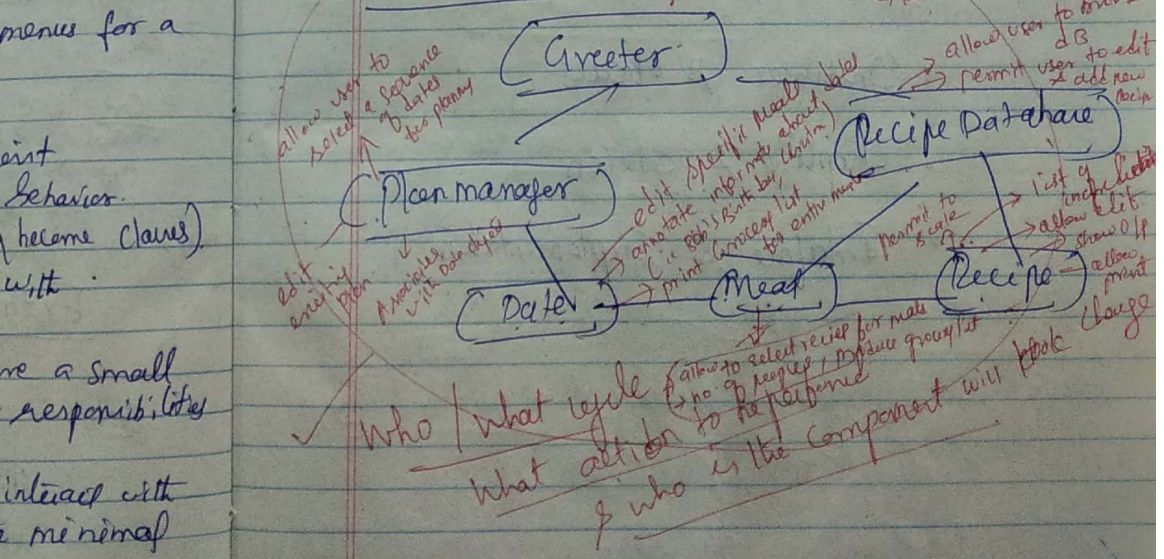
→ Components are most easily described using CRC card.

→ Prospective, Erasable, Physical.

- Component name
- Responsibility Description (Call sent to this Component)
- Collaborators (List of other Components)

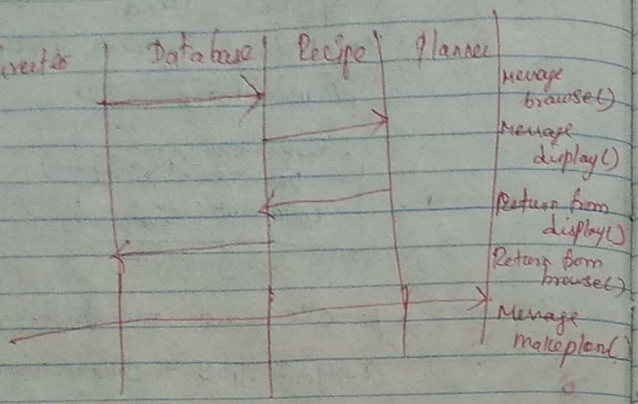
→ A CRC card records the name, responsibilities & collaborators of an Component.

Six Components:



Interaction:

- 'static' relationship



Characteristics of Components:

- Behavior & state
- Instances & classes
- Coupling & cohesion
- Interface & implementation

Behavior & State:

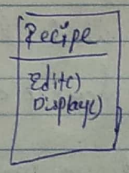
Behavior → Set of actions a Component can perform.

State → represents all the information (data value) held within a Component.

Behavior can change State
eg: edit Recipe()
↳ change preparation information
ie State

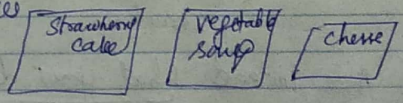
Instance & Classes:

Recipe class



Behavior is common to the class
ie all instance behave in same way

Recipe Instances



Cohesion & coupling:

Cohesion is the degree to which the tasks assigned to a Component seem to form meaningful unit

Maximize Cohesion

Coupling is the degree to which the ability to fulfill a certain responsibility depends upon the actions of another Component.

Minimize Coupling

Interface & Implementation

User of a S/w Component need only know what it does; not how it does it

"Ask not what you can do to a data structure

Ask instead what your datastructure can do for you

Parnas Principle:

→ Developed the concept of information hiding & Modular programming

Developer of a S/w Component must provide the intended user with all the information needed to make effective use of the services provided by the Component, & should Provide no other information

The implementor of a S/w Component must be provided with all the information necessary to carry out the given responsibilities assigned to the Component & should be provided with no other information

Primitive Data types:

Static type vs dynamic type

Java = Statically type language
⇒ type of variable is known at compile time
⇒ static-type checking
⇒ type checking performed during compile time
eg: Java, C, C++, C#, Haskell, Perl

for eg: (Python = Dynamically typed language)
⇒ type of variable is known at runtime
eg: Javascript, Python, php

Weak type vs Strong type

Strong type:
⇒ Each type of data (Int, Char, etc) is predefined as part of the PL.
⇒

Weak type:
C#, C,

Unicode: Java represents text in sequences of 16-bit code unit UTF-16 encoding

String.toUpperCase() eg: A ⇒ \u0041 a = \u0061
⇒ \u0B85 String S = "\u0B85";

Java PL is statically typed which means all variables must be first declared before they can be used

eg: int max=100;

Type: primitive type
non-primitive type

primitive type → numeric, Boolean
numeric ⇔ Integral, floatingpoint

Integral
1) byte size 1 byte, default value = 0
2) Short d.value = 0;
3) int d.value = 0
4) Long d.value = 0L; ⇒ 0
Unicode ⇒ 5) Char d.value = '\u0000'

Comparison
operators: <, <=, >, >=, ==, !=

numerical operators
Unary plus/minus +, −, *, /, %, ++, --
Shift <<, >>, >>>, ~, &, ^, |
?:, Cast, String (+) Concat,

Long: 64 bit precision

Float:
1) float d.value = 0.0f;
2) double d.value = 0.0d;
operators <, <=, >, >=, ==, !=
Unary minus, plus, +, −, *, /, %, ++, --, ?:, Cast, Shift

non-primitive type

String: java.lang.String (class)
default = null,
⇒ immutable ⇒ once created cannot be change

Array

class

Primitive

Boolean: default value = false
value: true & false

Operators ⇒ ==, !=,
!, &, ^, |,
&&, ||, ?:,
Shift.
new.

Underscore char in numeric literals } use as separator for understood

long creditcardnumber = 1234_5678_9012_3456L;
float pi = 3.14_15f.
long hex = 0XCAFE_BABE;
long bin = 0B1101_1001_1100;

not allowed;

Underscore is not allowed

⇒ At the begin / end of a number
⇒ Adjacent to a decimal pt in a float point literal.
⇒ prior to an F or L Suffix
⇒ In ::

eg: Invalids

float p = 3_.1415f; X adjacent to decimal point not allowed
float p = 3._145f; X
999_99_999_L; X ; prior to L suffix not allowed.
int x = 5_2; ✓ (cannot put underscore at the end of literal)
int x = 52_; X
int x = 5_____2; ✓
int x4 = 0_x52; X (cannot provide underscore in the 0x radix prefix or suffix)
int x4 = 0x_52; X
int x5 = 0x5_2; ✓
int x6 = 0x52_; X cannot put at end of number

Primitive

Boolean: default value = false
value: true & false

Operators => ==, !=,
!, &, ^, |,
&&, ||, ?:,
&=, |=, ^=,
+.
.

Underscore char in numeric literals } use as separate for understood

long creditCardNumber = 1234_5678_9012_3456L;
float pi = 3.14_15f;
long hex = 0XCAFE_BABE;
long bin = 0B1101_1100_1100_1100;

not allowed;

Underscore is not allowed

=> At the begin / end of a number
=> Adjacent to a decimal pt in a float point lit.
=> prior to an F or L Suffix
=> In :.

eg: Invalids

float p = 3_.1415f; X adjac to decimal point not allowed
float f = 3._145f; X
999_99_999_L; X ; prior to L Suffix not allowed.
int x = 5_2; ✓ (cannot put underscore at the end of literal)
int x = 52_; X
int x = 5_____2; ✓
int x4 = 0_X52; X (cannot provide underscore in the 0X radix prefix or suffix)
int x4 = 0X_52; X
int x5 = 0X5_2; ✓
int xf = 0X52_; X cannot put at end of num

1) Largest of three numbers using Ternary operator.

```
Scanner S = new Scanner(System.in)
num1 = S.nextInt();
num2 = S.nextInt();
num3 = S.nextInt();
temp = num1 > num2 ? num1 : num2;
result = num3 > temp ? num3 : temp;
S.o.pln ("Largest no" + result);
```

2) Smallest of three number using Ternary op.

3) number positive or negative (numb > 0.0)? "positive" : "not positive".

4)
```
class test {
    p.s.v.m ( )
    { ~~for (int i=0;~~ int t;
        S.o.pln(t);
    } }
```

O/p: compile time error, 0, Garbage value, runtime error

Answer: Compile time error: Because local var must be assigned a value before they are accessed.

Operators:

Why ternary?
=) Replace multiple line of code with single line of code
=) more readable

Arrays:

=) begin from '0'

Declare
int[] array; // declares an array of integers

- 'array' indicates var, holds an array
- size is not part of its type.

Creating
array = new int[10]; // allocates memory for 10 integers

Initialize
array[0] = 100; initialize first element

access
S.o.pn (array[0]);

```
int[] array = { 100, 200, 300 };
S.o.pn (array.length);
```

array of array - Multi dimension int[][]

```
String[][] names = { {"Mr.", "Mrs.", "Ms"},
                     {"Smith", "Jones"} };
= S.o.pn (names[0][0] + names[1][0]);
    // Mr. Smith
S.o.pn (names[0][2] + names[1][1]);
    // Ms. Jones
```

Operators:

Equality
precedence → left to Right [Binary]
Right to left [assignment]

Postfix a++, a--
Unary ++a, --b, +a, -a, ~a, !a
multiplicative * / %
addition + -
shift << >> >>>
relate <, >, <= >= instanceof
equality == !=
bitwise AND &
bitwise exclusive ^
" inclusive OR |
logical AND &&
" OR ||
ternary ?:
assignment =, +=, -=, *=, /=, %=,
&=, ^=, |=, <<=, >>=, >>>=

shifts a bit Pattern to left
right
unsigned Right shift operator zero into the left most position

Equality, Relation, Conditional

Equality & Relational
==, !=, >, >=, <, <=

Conditional:
&&
||

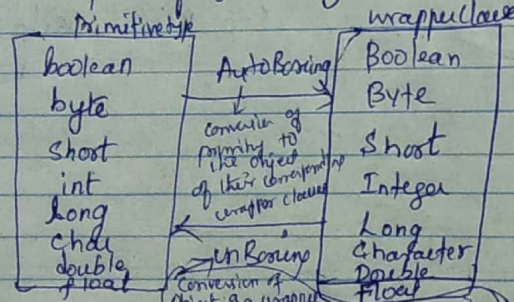
instanceof

class parent
{ }
class child extends parent { }
class test
{
p.s.v.m()
{
child childobj = new child();
if (childobj instanceof child)
{
S.o.p ("childobj is an instance of child");
}
}

WRAPPER CLASSES:

1. What are Wrapper classes?
2. Why do we need wrapper class?
3. How to create wrapper class instances?
4. What is autoboxing & unboxing?
5. What are the advantages of autoboxing?
6. Casting? Implicit Casting? Explicit Casting

Wrapper class:
→ is a class that wraps around a primitive data type.
→ is a class whose object wraps primitive data types.
→ it is part of java.lang. (:- no import is needed)



Need?
⇒ allow to create objects to represent a primitive
⇒ Wrapper classes are immutable - Cannot change the objects value.
⇒

→ Java provides 8 primitive data types
⇒ hence Java is not 100% object oriented
⇒ ∵ Because they are created from classes
→ Java platform provides wrapper classes for each of the primitive data type.
⇒ wraps the primitive in an object

⇒ Instance methods & static methods of wrapper classes.

Methods:
(1) Convert wrapper object to primitive
Convert wrapper to primitive
Integer ⇒ Parse: xxxValue() method

Instance Method

```
Integer i = new Integer(42);
byte b = i.byteValue();
short s = i.shortValue();
double d = i.doubleValue();
int i = i.intValue();
```

```
public byte byteValue();
public short shortValue();
public int intValue();
public long longValue();
public float floatValue();
public double doubleValue();
```

Program:

```
class Example
{
  p.s.v.m(String[] args)
  {
    Double d = new Double(10.5);
    s.o.println(d.byteValue() +
                d.shortValue() + d.doubleValue()
                d.longValue() + d.floatValue());
  }
}
```

O/p: 10 10 10 10.5 10 10.5

Character c = new Character ('*');
S.o.pln (c.charValue());

O/p: *

Boolean b = new Boolean ("true");
S.o.pln (b.booleanValue());

o/p: true

Convert string to primitive:

2) xxxParse Xxx (String) method:

Static method

ParseXxx ()
→ Convert string into Corresponding primitive
→ Returns the named primitive

Every wrapper class except character class contain ParseXxx() method to convert String to Corresponding primitive

public static primitivetype parseXxx (String s);

Class Example
{
Boolean bl = Boolean.parseBoolean ("true");
byte ba = Byte.parseByte ("10");
int i = Integer.parseInt ("10");
long l = Long.parseLong ("10");
float f = Float.parseFloat ("10.5f");
Double d = Double.parseDouble ("10.5");
S.o.pln (bl + ba + i + l + f + d);
}
o/p: true 10 10 10 10.5 10.5

public static primitive parseXxx (String s, int radix);

Autoboxing

Convert primitive to wrapper:

ValueOf()

Convert primitive to wrapper

Public static ValueOf (primitive);
= returns an instance of the wrapper with a value specified as argument.

Boolean bool = Boolean.ValueOf (true);
Byte b = Byte.valueOf (12);
Character c = Character.valueOf ('a');
Short s = Short.ValueOf (32);
Integer i = Integer.ValueOf (123);
Long l = Long.valueOf (9222345678L);
Float f = Float.valueof (12.45f);
Double b = Double.valueOf (12.35);
S.o.pln (bool + "" + b + "" + c + "" + s + "" + i + "" + l + "" + f + "" + b);
O/p: true, 12, a 32 123 9222345678 12.45 12.35

Convert String to wrapper:

Public static valueOf (String);
= Returns an instance of the wrapper with a value represented by the string argument

```
Byte b = Byte.valueOf("127");
Short s = Short.valueOf("327");
Integer i = Integer.valueOf("123");
Double d = Double.valueOf("123.45");
Float f = Float.valueOf("12.35F");
S.o.pln(b+" "+s+" "+i+" "+d+" "+f);
```

NumberformatException = it the value represented by the String parameter cannot be used to construct a new instance of the wrapper class.

eg: Byte.valueOf("128")

java.lang.NumberformatException: value out of range
Radix: 10

Integer.valueOf("one");

java.lang.NumberformatException: Input String one.

Long.valueOf("123.4");

NumberformatException:

Whereas: Boolean.valueOf(String) => does not throw a Exception - It returns true if the String argument is equal, ignoring case, else it returns false

```
Boolean b1 = Boolean.valueOf("true");
b2 = ("TRUE");
b3 = ("false");
b4 = ("yes");
S.o.pln(" "+b1+" "+b2+" "+b3+" "+b4);
```

O/P: true TRUE false false.

Convert Primitive to String:

Public static String toString(primitive)

eg: String s = Boolean.toString(98);

Integer & Long classes

```
int i = 13;
String binary = Integer.toString(i, 2);
```

=> toString(primitive, int radix)

```
String octal = Integer.toString(i, 8);
String decimal = Integer.toString(i, 10);
String Hexa = Integer.toString(i, 16);
S.o.pln(binary+" "+octal+" "+
        decimal+" "+Hexa);
```

O/P:
1101 15 13 d

Convert wrapper to String:

```
Public String toString();
String S = new Integer(87).toString();
S.o.pln(S);
```

Converting primitive to wrapper ~~X~~ Repeated

Public Static valueOf (primitive)

- returns an instance of the wrapper with value specified as primitive argument

eg: Boolean bool = Boolean.valueOf(true);

Integer i = Integer.valueOf(12345);

Short i = Short.valueOf(123);

convert Binary to Hex, Binary octal to Hex String

String s = Integer.toHexString(254) fe

Need of Wrapper Class:

* primitive types can't be null but wrapper classes can be null.
* To achive polymorphism.

```
i=15;
Integer in = new Integer(i);
in = null;
```

* To support Synchronization in multithreading
* To store Objects / Retrieve Objects in collection or Datastructures such as ArrayList, Vector, Stack, queue etc.
* Classes in java.util package handles only Classes.

Advantages of wrapper class:

* Readability
* no specific typecasting explicitly to be performed

Disadvantages of wrapper class:

String Manipulations:

=> String Class is immutable (constant). once created & initialized, cannot be changed

=> String is a final class, no other class can extend it.

=> String values use equals() method to compare

=> supports various methods

String initialization:

```
String s1 = "String is";
String s2 = new String("String is");
char[] c = {'a', 'b', 'c', 'd', 'e'};
String s3 = new String(c);
```

character array to String:

char ch[] = {'a', 'b', 'c', 'd'};

CopyValueOf (2)

String s1 = String.CopyValueOf(ch);
String s2 = String.CopyValueOf(ch, 3, 1);
S.o.pn(s1 + " " + s2
o/p => abcd de

Concatenation: [write a program to concatenate two strings]

two ways:

String s1 = "abc";
String s2 = "xyz";
String s = s1 + s2;
String ss = s1.concat(s2)
S.o.pn(s + " " ss);
abcxyz abcxyz

Compare:

(1) equals() =>
(2) equal to operator not used ie "==" because compares the object reference not string value

(2) equalsIgnoreCase()
=> ignore case

String x = "abc";
String y = "ABC";

if (x.equals(y))
{ S.o.pn ("equal");
else S.o.pn ("not equal");

if (x.equalsIgnoreCase(y))
{ S.o.pn ("equal");
else S.o.pn ("not equal");

o/p: not equal equals

indexOf()

→ to get the position of the specified string or char from the given string

String str = "Hello Jaya Welcome";

S.o.pn (Str.indexOf('e'));

S.o.p (Str.indexOf('e', 4)).
//first occurrence of e from 4th index onwards

S.o.pln (Str.indexOf ("jaya", 6))
//first occur of string jaya from 6th index onwards

o/p: 1
16
6

lastindexOf()

lastIndexOf() => last occurance of the reference string or char in the reverse order. (backward)

String str = "use strings";

(index marks over the string: 0 1 2 3 4 5 6 7 8 9)

s.o.p(str.lastIndexOf('s'));

o/p: 8

StartsWith(). String starts with the given String or not

String str = "This"

s.o.pn(str.startsWith("This"));

o/p: True.

str.startsWith("This", 5);

after index 5

endsWith()

String str = "Hello java";

s.o.p(str.endsWith("java"));

o/p true

String[] split (String s)

String[] split (String s, int limit) — Max. size of the returned array

Split:

String str = "Hello java";

String[] token = str.split(" ");

for (String s : tokens)

s.o.pn(s);

~~tokens = str.split(~~

o/p Hello

Java.

str.toUpperCase()

str.toLowerCase();

0	1	2	3	4	5	6	7	8
H	e	l	l	o	(space)	J	a	v	a

Substring(5, 9)

String Substring(int beginIndex, int endIndex)

(endIndex-1)

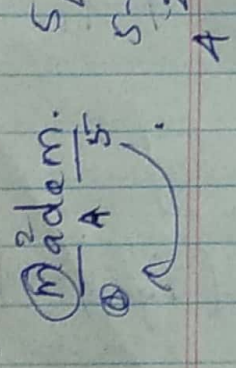
= Java.

String trim()

↳ leading & trailing white space removed.

boolean contains (charsequence s)
↳ Returns true if the String contains the specified charsequence.

String replace (char oldchar, char newchar)
String replace (charsequence target, charseq replace)
String replaceAll (String s, String Replace)
String replaceFirst (String s, String Replace)



endsWith()

```
String Str = "Hello java";
S.op(Str.endsWith("java"));
```

O/p true