Java BASIC

Diary
2016

For :

Leads Java SE_Android Batch 2

Graphic Arts Institute

05 Tuesday Time Januray 2016

Core Java → Language Fundamentals

(I) Identifiers
(II) Reserved Words
(III) Data Types
(IV) Literals
(V) Arrays
(VI) Types of variables
(VII) var-arg methods

06 Wednesday Time Januray 2016

(VIII) main method
(IX) Command line arguments
(X) Java coding standards

January 2016

Sat	Sun	Mon	Tue	Wed	Thu	Fri
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

07 Thursday Time

Identifiers:

A name in java program is called identifier. Which can be used for identification purpose. It can be method name, class name, variable name or lable name.

Example:

08 Friday Time Januray 2016

```
class Test
{
public static void main(String[] args)
{
int x = 1;
}
```

(1) (2) (3) (4) (5)

How many identifiers are here: 5

09 Saturday Time Januray 2016

① Rules for defining Java Identifiers:

① The only allowed characters in Java identifiers are:

a to z
A to Z
0 to 9
$
_

total_number ✓ (valid)
total # ✗ (Invalid)
total123 ✓ (valid)
123total ✗ (Invalid)

If we will use any other character

10 Sunday Time Januray 2016

then we will get compile time error.

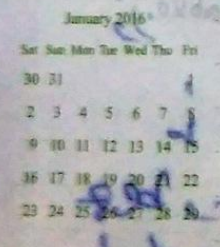
② Identifiers can't start with digit.

total123 ✓
123total ✗

③ Java Identifiers are case sensitive

int valid = 1;
int Valid = 1;
int VALID = 1;

} all are different variable.



11 Monday Time

4) There is no length limit for java identifiers. But it is not recommended to take too lengthy identifiers.

(5) We can't use reserved words as identifiers.

```
int x = 10;   ✓
int if = 10;  ✗
```

↖ this is reserved word

(6) All predefined java class names and Interface names, we can use as identifiers

```
class Test {
    public static void main(String[] args) {
        int String = 88;
        S.O.println(String);
    }
}
```

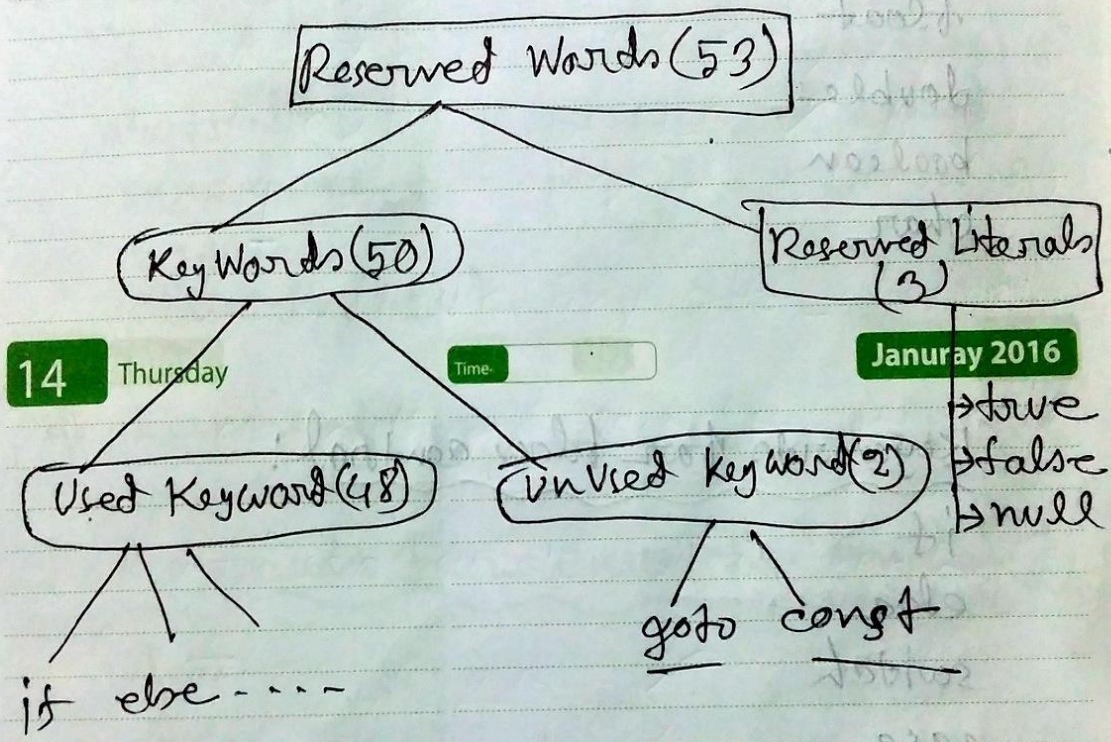
```
int Runnable = 99;
S.O.println(Runnable);
```

Even though it is valid but it is not a good programming practice, bcz it reduces readability and create confusion.

13 Wednesday Time Januray 2016

Reserved Words:

In java some words are reserved words, to represent some meaning and functionality. Such type of words are called, reserved words.



14 Thursday Time Januray 2016

Januray 2016

15 Friday Time

Keyawords for Data types:(8)

byte
short
int
long
float
double
boolean
char

16 Saturday Time

Januray 2016

Keyawords for flow control:

if
else
switch
case
default
while
do
for
break
continue
return

January 2016

17 Sunday Time Januray 2016

Keywords for modifiers: (11) with default it is (12)

public
private
protected
static
final
abstract
synchronized
native
strictfp (1.2v)

18 Monday Time Januray 2016

transient
volatile

Keywords for exception Handling (6)

try
catch
finally
throw
throws
assert (1.4 v)

19 Tuesday Time Januray 2016

Class Related Keywords: (6)

class
Interface
Extends
implements
Package
import

20 Wednesday Time Januray 2016

Object Related keywords: (4)

new
instanceof
super
this

Return type: (1)

void

In java return type is mandatory. If a method won't return anything then we have to declare that method void return type.

21 Thursday Time Januray 2016

Unused keywords: (2)

goto: Uses of goto, create a several problems, in old languages and hence some people ban this keyword in java.

const: Use of final instead of "const".

N.B: goto and const are unused

22 Friday Time Januray 2016

keywords and if we are trying to use, we will get compile time error.

Reserved Literals: (3) true, false, null

true and false are values for boolean data types.

null is default value for object reference,

January 2016
Sat Sun Mon Tue Wed Thu
30 31 1
2 3 4 5 6 7 8
9 10 11 12 13
16 17 18 19 20 21 22
23 24 25 26 27 28 29

23 Saturday Time Januray 2016

Enum keyword : (1)

enum (1.5 v)

We can use enum to define a group named constant.

Example :

```
enum month
{
jan, feb, mar, .... Dec;
}
```

24 Sunday Time Januray 2016

```
enum Beer {
KF, KO, RC ...;
}
```

January 2016

Sun	Mon	Tue	Wed	Thu	Fri
31					1
3	4	5	6	7	8
10	11	12	13	14	15
17	18	19	20	21	22
24	25	26	27	28	29

25 Monday Time Januray 2016

Conclusions:

① All 53 reserved words in java contains only lower case alphabet simbles.

② In java we have only new keyword, and there is no delete keywards, bez distruction of useless object is the responsibility of garbage collector.

26 Tuesday Time Januray 2016

③ The following of new keywords in java :

strictfp 1.2 v

assert 1.4 v

enum 1.5 v

27 Wednesday Time Januray 2016

(4) storicttp but not strictFP
instanceof but not instanceOf
synaronized but not synaronize
extends but not extend
implements but not implement
inport but not imports
const but not constant

28 Thursday Time Januray 2016

January 2016
Sun Mon Tue Wed Thu Fri
31 1
3 4 5 6 7 8
0 11 12 13 14 15
7 18 19 20 21 22
25 26 27 28 29

29 Friday Time Januray 2016

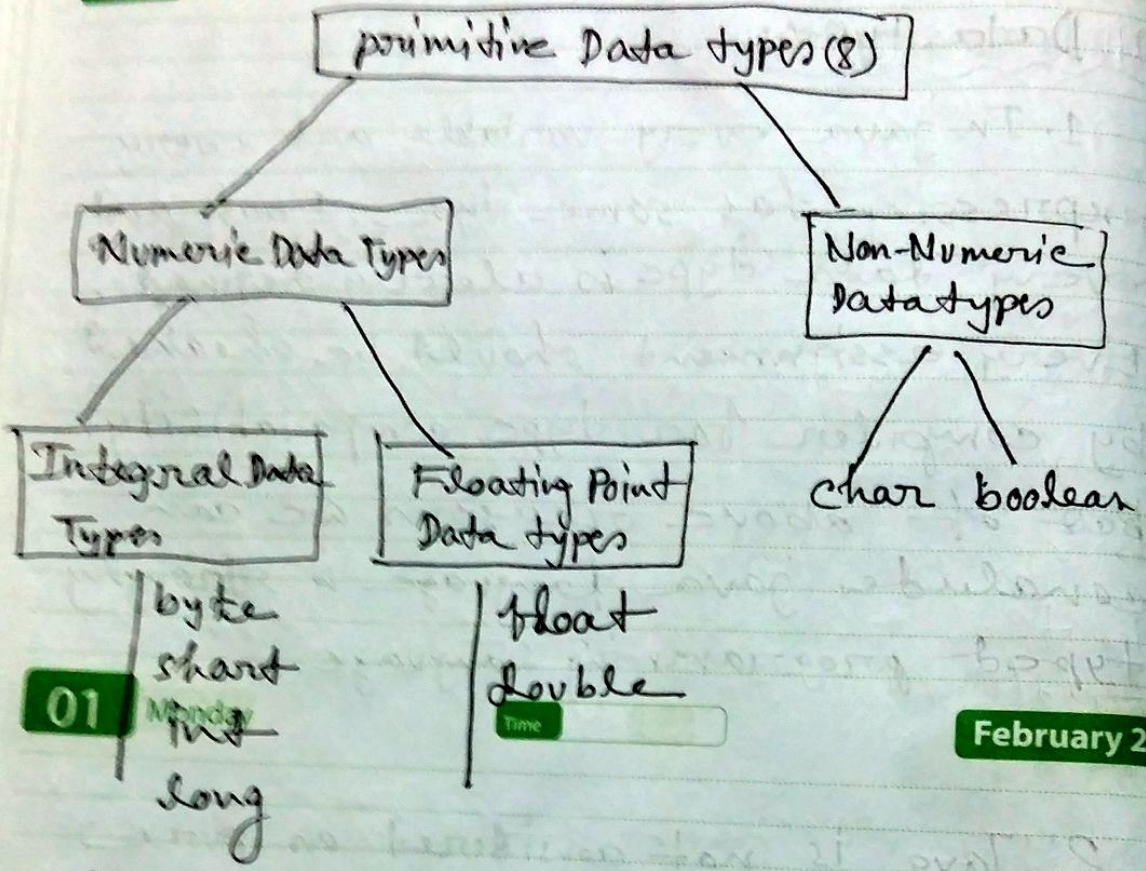
Data types:

1. In java every variable and every expression has some type. Each and every data type is clearly defined. Every assignment should be cheeked by compiler for type compatability. Bez of above reasons we can conclude java language is strongly typed programming language.

30 Saturday Time Januray 2016

2. Java is not considered as pure object oriented programming language bez several opp features are not satisfied by java. (Like: operator overloading, multiple Inheritance etc.)

More over we are depending on premitive data types, which are non objects.



Except boolean and char, remaining data types ~~are~~ are considered as signed data types, bcz we can represent both positive and negative numbers.

```
int a = -10;
int b = +5;
float c = -10.5;
```

```
char d = -'a'; X
boolean b = -true; X
```

02 Tuesday Time February 2016

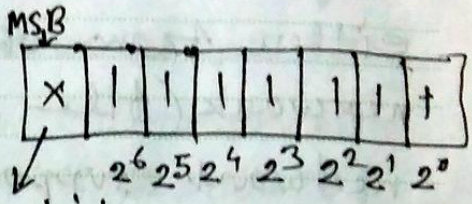
① byte: Size 1 byte (8 bit)

Max. value → +127

Min. value → -128

Range: -128 to +127

Negative value calculate always with seconds complement.



signbit

0 → +ve

1 → -ve

64+32+16+8+4+2+1 = 127

03 Wednesday Time February 2016

✓ byte b = 10;

✓ byte b = 127;

✗ byte b = 128;

↓

CE: Possible loss of precision
found: int
required: byte

byte b = true;

CE: incompatible types
found: boolean
required: byte

byte b = "stab";

CE: incompatible types
found: String
required: byte

✗ byte b = 10.5;

↓

CE: Possible loss of precision
found: double
required: byte

04 Thursday Time February 2016

Byte is the best choice, if we want to handle data in terms of streams either from the file or from the network (file supported form or network supported form is byte).

(II) Short:

Size: 2 bytes (16 bits)

Range: -2^{15} to $2^{15}-1$

05 Friday Time February 2016

[-32768 to 32767]

This is the most rarely used data types in Java.

✓ short s = 32267;

✗ short s = 32268;

CE: PLP
found: int
required: short

short s = true; ✗

CE: incompatible type
found: boolean
required: short

06 Saturday

February 2016

short data types is best suitable for 16 bit processor like 8085. but these processors are completely out dated and hence corresponding short data type is also out dated data types.

07 Sunday

February 2016

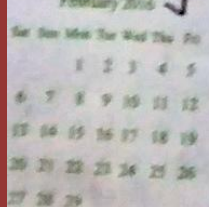
(III) int:

size → 4 byte (32 bit)

Range → -2^{31} to $2^{31}-1$

[-2147483648 to 2147483647]

The most commonly used data types in java is "int".



08 Monday

✓ int x = 2147,48,3647;

✗ int x = 2147483648;
CE: integer number too large

✗ int x = 2147483648L;
CE: PLP
found: long
required: int

09 Tuesday

✗ int x = true;
CE: incompatible type
found: boolean
required: int

February 2016

10 Wednesday Time February 2016

(iv) Long:

Sometimes int may not enough to hold big values then we should go for long type:

Ex: The amount of distance travelled by lights in 1000 days, to hold this value int may not enough, we should go for long data type.

long l = 126,000 × 60 × 60 × 24 × 1000 miles

11 Thursday Time February 2016

Ex2: The number of characters present in a big file may exceeds int range, hence the return type of length method is long but not int.:

long l = f.length();

size: 8 bytes (64 bit)

Range: -2^{63} to $2^{63}-1$

12 Friday Time

All the above data types (byte, short, int, long) represent integral values. If we want to represent floating point values, then we should go for floating point data types.

13 Saturday Time

Floating Point data types:

float	double
(I) 5 to 6 decimal point	(I) 15 to 20 decimal point
(II) Single precision (less accuracy)	(II) Double precision (more accuracy)
(III) 4 bytes	(III) 8 bytes
(IV) Range: ~~1.7e-38 to 1.7e38~~ ~~1.7 x 10^38~~ -3.4e38 to 3.4e38	(IV) ~~-3.4e308 to 3.4e308~~ -1.7e308 to 1.7e308

14 Sunday Time

February 2016

boolean:

Size: N/A (Not applicable) VM dependen

Range: N/A (Not applicable)

but allowed values are: true/false

✓ boolean b = true;

✗ boolean b = 0;

CE: incompatible types
found: int
required: boolean

15 Monday Time

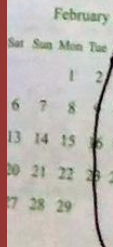
February 2016

✗ boolean b = True;

CE: can not find symble
symble: variable True
location: class Test

✗ boolean b = "True";

CE: incompatible type
found: java.lang.String
required: boolean



16 Tuesday Time

1.
```
int x=0;
if (x){
  S.O.P("Hello");
}else{
  S.O.P("Hi");
}
```

CE: incompatible types
found: int
required: boolean

2.
```
while(1){
  S.O.P("Hello");
}
```

17 Wednesday Time

Both example are valid in C ~~(OS)~~ languages), but not in java bcz java is strongly typed language.

18 Thursday Time February 2016

char:

Old languages (like c, c++) are ASCII code based. and the number of allowed different ASCII code characters are ≤ 256. To represent these 256 characters 8 bits are enough. Hence the size of char in old language is 1 byte (8 bit)

19 Friday Time February 2016

But java is UNICode based and the number of different unicode characters are > 256 and ≤ 65536. To represent these many characters 8 bits may not enough. compulsory we should go for 16 bit (2 byte). Hence the size of char in java is 2 bytes.

Range: 0 - 65535

20 Saturday Time

Summary of Java Primitives Data Types

Data Type	Size	Range	Wrapper Class
byte	1 byte	-2^7 to 2^7-1	Byte
short	2 bytes	-2^{15} to $2^{15}-1$	Short
int	4 bytes	-2^{31} to $2^{31}-1$	Integer
long	8 bytes	-2^{63} to $2^{63}-1$	Long
float	4 bytes	$-3.4e^{38}$ to $3.4e^{38}$	Float
double	8 bytes	$-1.7e^{308}$ to $1.7e^{308}$	Double
boolean	N/A	N/A (but allowed values are true, false)	Boolean
char	2 bytes	0 to 65535	character

21 Sunday Time

22 Monday Time February 2016

Default value

0

0

0

0

0.0

23 Tuesday Time February 2016

0.0

false

0 [represents space character]

February 2016

Sat	Sun	Mon	Tue	Wed	Thu	Fri
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29				

24 Wednesday Time

null

char ch = null;

CE: incompatible types
found: null type
required: char

null is the default value for object reference. And we can't

25 Thursday Time

apply for primitives. If we are trying to use for primitive, then we will get compile time error.

February 2016

Sat	Sun	Mon	Tue	Wed	Thu	Fri
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29				

26 Friday Time February 2016

Literals:

```
int   x   =   10;
```

- `int` → Data type/Keywords
- `x` → Name of Variable or Identifier
- `10` → constant value or Literals

A constant value which can be assigned to the variable is called a literal.

27 Saturday Time February 2016

Integral Data type Literals value:

byte
short
int
long

① decimal literals (Base-10) 0-9

```
int x = 10;
```

② Octal literals (Base-8) 0-7

```
int x = 010;
```

prefix will be 0 for octal

③ Hexadecimal literals (Base-16) 0-9 a-f

```
int x = 0x10;
```

prefix will be 0x for hexadecimal

28 Sunday Time:

For Integral literals:

For integral data types (byte, short, int, long) we can specify literal value in the following ways.

Decimal literals: Base-10, allowed digits are 0-9.

For extra digit in Hexadecimal (a to f) we can use both lower or upper case letters. There is one of very few areas where Java is not case sensitive

29 Monday Time:

The literal value should be prefixed with 0x or 0X.

✓ int x = 10;

✗ int x = 0786; CE: integer number too large
↳ In octal 8 is not allowed.

✓ int x = 0XFace;

✓ int x = 0777;

01 Tuesday Date March 2016

```
class Test {
    public static void main(String[] args) {
        int x = 10;
        int y = 010;
        int z = 0x10;
        S.O.P(x+"..."+y+"..."+z);
    }
}
```

output: 10 ... 8 ... 16

$(10)_8 = (?)_{10}$

$0\times8^0 + 1\times8^1$
$= 0 + 8 = 8$

$(10)_{16} = (?)_{10}$

$0\times16^0 + 1\times16^1$
$= 0 + 16 = 16$

02 Wednesday Date March 2016

10, 010, 0x10 → int

10l, 010L, 0x10L → long

By default every integral literals are int type. But we can specify explicitly as long type by sufix with small "l" or "L".

03 Thursday Time

March 2016

✓ int x = 10;

✓ long l = 10L;

✗ int x = 10L; → CE: PLP found: long required: int

✓ long l = 10;

04 Friday Time

March 2016

There is no direct way to specify byte and short explicitly. But Indirectly we can specify. Whenever we are assigning integral literals to the byte variable and if the value with in the range of byte, then compiler treats it automatically as byte literals. Similarly, short literals also.

05 Saturday Time March 2016

✓ byte b = 10;

✓ byte b = 127;

✗ byte b = 128; → CE : PLP
found : int
required : byte

✓ short s = 32767;

✗ short s = 32768; → CE : PLP
found : int
required : short

06 Sunday Time March 2016

07 Monday Time March 2016

Floating Point literals: By default every floating point literals are double type and hence we can't assign to the float variable. But we can specify floating point literals as float type by sufix with "f" or "F".

08 Tuesday Time March 2016

X float f = 123.456; CE: PLP found: double required: float

✓ float f = 123.45f;

✓ double d = 123.56;

We can specify explicitly floating point literals as double type by sufix with "d" or "D". Of course this convention is not required.

March 2016

Graphic Arts Institute

09 Wednesday

Time

March 2016

✓ double d = 123.456D;

✗ float f = 123.456f; → CE: PLP
found: double
required: float

We can specify floating point literals only in decimal form. and we can't specify in octal or hexadecimal forms:

10 Thursday

Time

March 2016

✓ double d = 123.456;

✓ double d = 0123.456; → This will treated as decimal

✗ double d = 0X123.456;

CE: malformed decimal point literals.

March 2016

Sun	Mon	Tue	Wed	Thu	Fri
		1	2	3	4
6	7	8	9	10	11
13	14	15	16	17	18
20	21	22	23	24	25
27	28	29	30	31	

11 Friday March 2016

We can assign integral literal directly to floating point variables and that integral literals can be specified, either in decimal or octal or hexadecimal forms.

✗ double d = 0786; => CE: integer number too large

✓ double d = 0xFace;

✓ double d = 0786.0;

12 Saturday March 2016

✗ double d = 0xface.0;

✓ double d = 10;

✓ double d = 0277;

double d = 10;
S.O.println(d);
output: 10.0

[bec, integral decimal number converted into floating point number;]

13 Sunday Time March 2016

We can't assign floating point literal to integral types;

✓ double d = 10;

✗ int x = 10.0; | CE: PLP
found: double
required: int

14 Monday Time March 2016

We can specify floating point literals, even in exponential form (e; 1.2e3)

✓ double d = 12e3; → 12e3 = 12×10^3 = 1200.0
Its a double number.

✗ float f = 1.2e3; → we can not assign double to float variable
✓CE: P2P
found: double
required: float

✓ float f = 1.2e3F;

15 Tuesday Time

boolean literals: the only allowed values for boolean data types are true/false.

✓ boolean b = true;

✗ boolean b = 0; → CE: incompatable types
found: int
required: boolean

16 Wednesday Time

March 2016

✗ boolean b = True; → CE: can not find symbol
symbol: variable True
location: class Test

✗ boolean b = "true"; → CE: incompatible types
found: String
required: boolean

March 2016

Sat	Sun	Mon	Tue	Wed	Thu	Fri
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

17 Thursday Time March 2016

```
int x=0;
if(x){
s.o.pln("Hello");
}else{
s.o.pln("Hi");
}
```

CE: incompatible type
found: int
required: boolean

18 Friday Time March 2016

```
while(1){
    s.o.pln("Hello");
}
```

But both are valid in C language.

March 2016

Sat	Sun	Mon	Tue	Wed	Thu	Fri
			1	2	3	4
5	6	7	8	9	10	11
12	13	14	15	16	17	18
19	20	21	22	23	24	25
26	27	28	29	30	31	

19 Saturday Time March 2016

Character literal:

We can specify char literal as single character within single cotes.

✓ char ch = 'a';

✗ char ch = a; → CE: can not find symbol
symbol: variable a
location: class Test

20 Sunday Time March 2016

✗ char ch = "a"; → CE: incompatible type
found: java.lang.String
required: char

✗ char ch = 'ab'; →
CE1: unclosed char literal
CE2: unclosed char literal
CE3: not a statement

March 2016
Sun Mon Tue Wed Thu Fri
1 2 3 4
6 7 8 9 10 11
13 14 15 16 17 18
20 21 22 23 24 25
27 28 29 30 31

21 Monday Time March 2016

We can specify char literal as integral literal which represents unicode value of the character and the integral literal can be specified in either decimal, octal or hexadecimal form. But allowed range is 0 to 65535

22 Tuesday Time March 2016

✓ char ch = 97;
S.O.p (ch); output: a

✓ char ch = 0XFace;

✓ char ch = 0777;

✓ char ch = 65535;

✗ char ch = 65536; → CE: PLP found: int required: char

March 2016
Sat Sun Mon Tue Wed Thu Fri
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31

23 Wednesday Time March 2016

We can represent char literal in unicode representation, which is nothing but ~~single quote~~

'\uXXXX'

4-digit hexadecimal Number

char ch = '\u0061';
S.O.println(ch);
output: a

if 97
6 -1

24 Thursday Time March 2016

Every escape character is a valid char literal.

✓ char ch = '\n';
✓ char ch = '\t';
✗ char ch = '\m'; {CE: illegal escape character}

25 Friday Time March 2016

In Java there are eight escape character:

escape charater	Description
\n	new line
\t	Horizontal tab
\r	carriage return
\b	Back space
\f	Form feed

26 Saturday Time March 2016

\'	Single quote
\"	double quote
\\	Back slash

27 Sunday Time

Which of the following are valid:

✗ char ch = 65536;

✗ char ch = 0XBeer;

✓ char ch = '\uFace';

28 Monday Time

March 2016

Sun	Mon	Tue	Wed	Thu	Fri
		1	2	3	4
6	7	8	9	10	11
13	14	15	16	17	18
20	21	22	23	24	25
27	28	29	30	31	

29 Tuesday March 2016

String Literal:

Any sequence of characters with in double quote is treated as string literal.

Ex: String s = "durga";

30 Wednesday March 2016

1.7 v Enhancement with respect two literals:

(1) Binary literals.

For integral data types (byte, short, int, long) until 1.6 v, we can specify literal value in the following ways:

(i) decimal (ii) octal (iii) Hexadecimal

but 1.7 v onwards we can specify literal value even in Binary form also.

31 Thursday Time

Allowed digits are: 0, 1

literal value should be prefixed with 0b or 0B.

Ex: int x = 0B1111;

s.o.p(x);

output: 15

01 Friday Time April 2016

(11) Uses of underscore symbol in numeric literals:

From 1.7v onwards we can use underscore symbol between digits of numeric literals.

Ex: double d = 123456.786;

double d = 1_23_456.7_8_6;

double d = 123_456.7_8_6;

March 2016
Sat Sun Mon Tue Wed Thu Fri
1 2 3 4
5 6 7 8 9 10 11
12 13 14 15 16 17 18
19 20 21 22 23 24 25
26 27 28 29 30 31

02 Saturday Time April 2016

The main advantage of this approach is readability of the code will be improve.

At the time of compilation these underscore sign will be remove automatically. Hence after compilation the above line will become:

double d = 123456.786;

03 Sunday Time April 2016

We can use more than one underscore symbol between the digits.

double d = 1__23___4_5_6.7_8_6;

dobble d = 1__23__4_5_6.7_8_6;

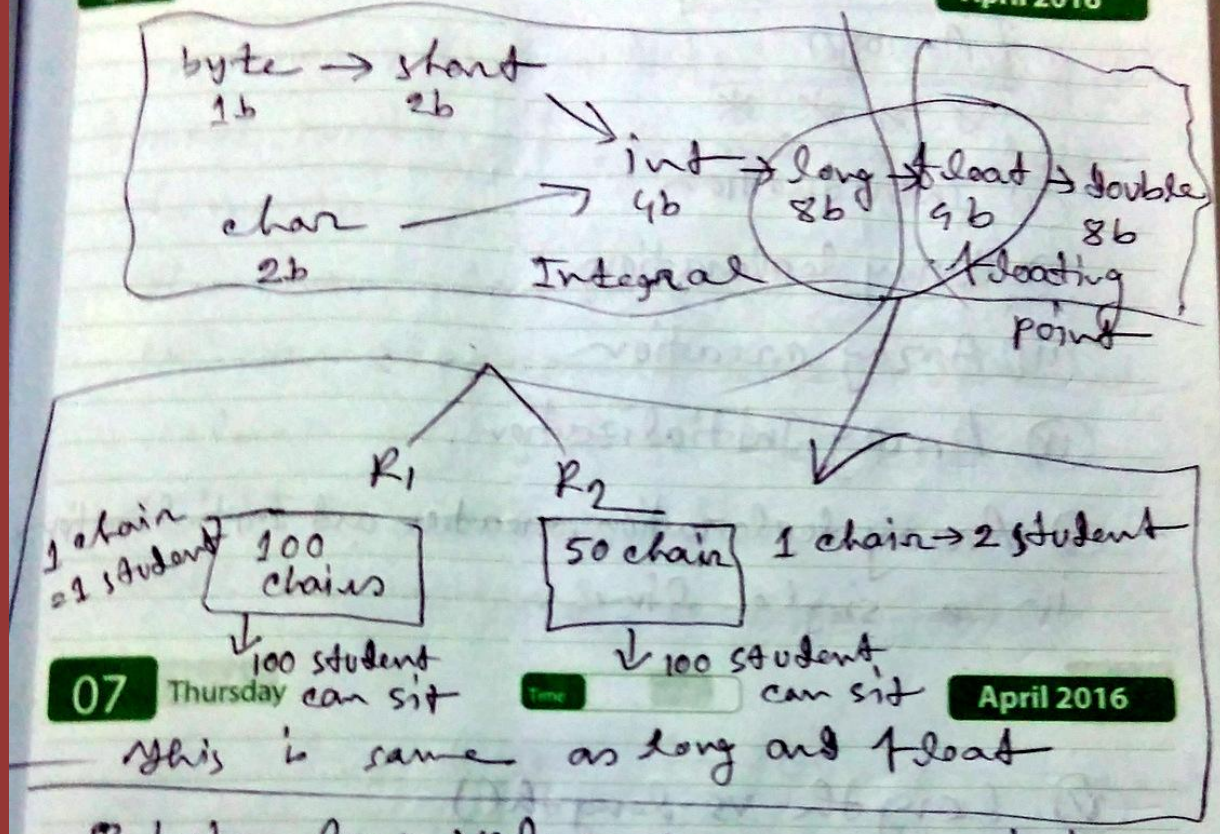
04 Monday Time April 2016

We can use underscore symbol only between the digits. If we are using any other areas, we will get compiletime error.

X
```
double d = 1_23_456.7_8_9;
double d = 1_23_456.7_8_9;
doble d = 1_23_456.7_8_9;
```

05 Tuesday Time April 2016

06 Wednesday Time April 2016



07 Thursday Time April 2016

8 byte long value we can assign 4 byte float variable bcz both are following different memory representation internally.

```
float f = 10l;
S.O.P(f);
```

output: 10.0f

08 Friday Time

Arrays:

* * * *

(i) Introduction

(ii) Array declaration

(iii) Array creation

(iv) Array Initialization.

(v) Array declaration, creation and Initialization in a single line.

09 Saturday Time

(vi) length vs length()

(vii) Anonymous Arrays.

(viii) Array element assignments

(ix) Array variable assignment.

10 Sunday Time April 2016

An array is an indexed collection of fixed number of, homogeneous data elements.

The main advantage of array is, we can represent huge number of values by using single variable, so that readability of the code will be improved, but the main

11 Monday Time April 2016

dis advantage of array is fixed in size, that is once we create an array there is no chance of increasing or decreasing the size, based of our requirements. Hence to use arrays concept, compalsury we should know the size in advance, which may not possible always.

April 2016

12 Tuesday Time April 2016

Array declaration:

One dimentional array declaration

```
int[] x;
int [] x;
int x[];
```

13 Wednesday Time April 2016

int[] x; is recomended bcz name is clearly separated from type.

At the time of declaration we can't specify the size, otherwise we will get compile time error.

```
int[6] x; X
int[] x; ✓
```

April 2016

Sun	Mon	Tue	Wed	Thu	Fri
					1
3	4	5	6	7	8
9	11	12	13	14	15
	18	19	20	21	22
25	26	27	28	29	

April 2016

Graphic Arts Institute

14 Thursday

April 2016

Two dimentional Array declaration:

```
int[][]  x;
int      [][]x;
int      x[][];
int[]    []x;
int[]    x[];
int      []x[];
```

15 Friday

April 2016

```
int[] a, b;   a→1, b→1
int[] a[], b;  a→2, b→1
int[] a[], b[];  a→2, b→2
int[] []a, b;  a→2, b→2
int[] []a, b[]  a→2, b→3
```

April 2016

Sat	Sun	Mon	Tue	Wed	Thu	Fri
30						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25				

```
int[] []a, []b; → CE.
```

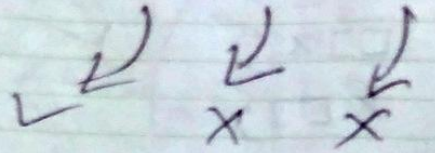
If we want to specify dimention before the variable, that facility is applicable only for first variable in declaration.

April 2016 — Graphic Arts Institute

16 Saturday — April 2016

If we are trying to apply for remaining variable, we will get compile time error.

```
int[] []a,[]b,[]c;
```



17 Sunday — April 2016

Three dimentional Array declaration:

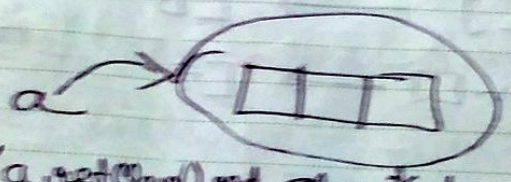
```
int[][][] a;
int   [][][]a;
int[]   [][]a;
int[] a[][];
int[] [] a[];
int[][] []a;
int[][] a[];
int   [][]a[];
int   []a[][];
```

18 Monday Time April 2016

Array Creation: One dimentional:

Every array in java is object only. Hence we can create arrays by using new operator.

```
int[] a = new int[3];
```




```
S.O.p(a.getClass().getName());
```

19 Tuesday Time April 2016

out: [I

For every array type corresponding classes are available and these classes are part of java language and not available to the programmer level.

20 Wednesday Time April 2016

Array type	corresponding class name
int[]	[I
int[][]	[[I
double[]	[D
short[]	[S
byte[]	[B
boolean[]	[Z

21 Thursday Time April 2016

① At the time of array creation compulsory we should specify the size, otherwise we will get compile time error.

int[] x = new int[]; ✗

int[] x = new int[6]; ✓

April 2016

22 Friday Time April 2016

② It is legal to have an array with size "0" in java.

int[] a = new int[0]; ✓

③ If we are trying to specify array size with some negative int value, then we will get ~~exception~~ runtime exception saying:

23 Saturday Time April 2016

negativearraysize exception.

int[] a = new int[-6];

RE: negativearraysize

24 Sunday Time April 2016

④ To specify array size that allowed data types are: byte, short, char, int.

If we are trying to specify any other type then we will get compile time error.

✓ int[] x = new int[10];

✓ int[] x = new int['a'];

25 Monday Time April 2016

byte b = 10;

✓ int[] x = new int[b];

short s = 20;

✓ int[] x = new int[s];

✗ int[] x = new int[10L];

CE: PLP
found: long
required: int

April 2016 — Graphic Arts Institute

26 Tuesday — April 2016

byte → short
char → int → long → float → double

⑤ The maximum allowed array size in Java is 2147483649, which is the maximum value of int data type.

27 Wednesday — April 2016

✓ int[] x = new int[2147483647];

✗ int[] x = new int[2147483648];

CE: integer number too large

Even in the first case we make a run time exception, if sufficient heap memory not available.

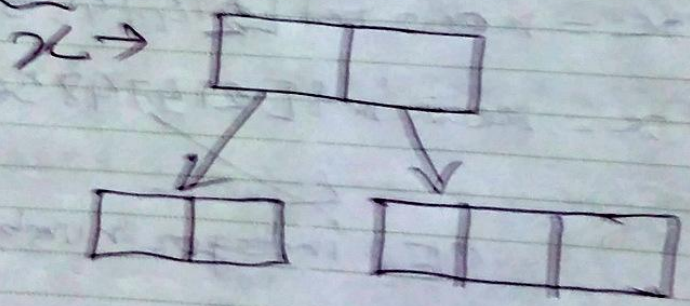
28 Thursday

Two dimentional Array creation:

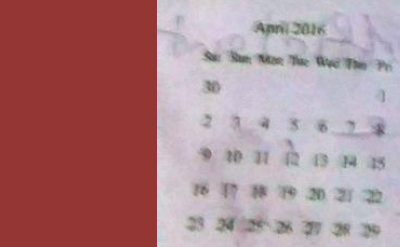
In java two dimentional array not implemented by using matrix style. Some people followed array of arrays approach for multidimentional array creation. The main advantage of this approach is memory utilization will be improved.

29 Friday

Ex: 1:



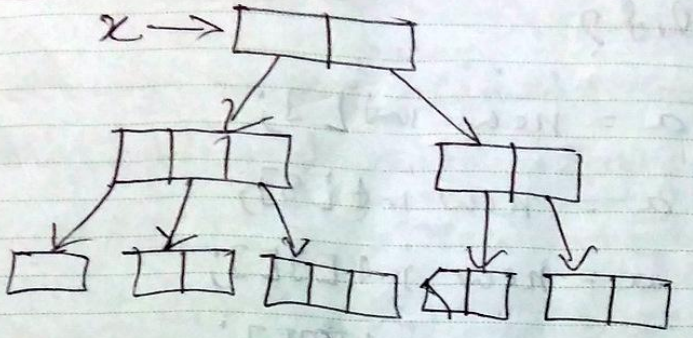

```
int[][] x = new int[2][];
x[0] = new int[2];
x[1] = new int[5];
```



30 Saturday Time

April 2016

Ex 2:



01 Sunday Time

May 2016

```
int[][][] x = new int[2][][];
x[0] = new int[3][];
x[0][0] = new int[1];
x[0][1] = new int[2];
x[0][2] = new int[3];
x[1] = new int[3][2]
```

May 2016

Sat	Sun	Mon	Tue	Wed	Thu	Fri
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

02 Monday Time May 2016

1. Which of the array declarations are valid?

```
int[] a = new int[];
int[] a = new int[3];
int[][] a = new int[][];
int[][] a = new int[3][];
int[][] a = new int[][4];
int[][] a = new int[3][4];
```

03 Tuesday Time May 2016

```
int[][][] a = new int[3][4][5];
int[][][] a = new int[3][4][];
int[][][] a = new int[3][][5];
int[][][] a = new int[][4][5];
```

May 2016

Graphic Arts Institute

04 Wednesday

Time

May 2016

Array initialization:

Once we create an array every array element by default, initialized with default values.

```
int[] x = new int[2];
S.O.println(x); [I@122
S.O.println(x[0]);
output: 0
```

default value initialization

x → [0 | 0]

05 Thursday

Time

May 2016

Whenever we are trying to print any reference variable, internally toString() will be called, which is implemented by default to return the String in the following form.

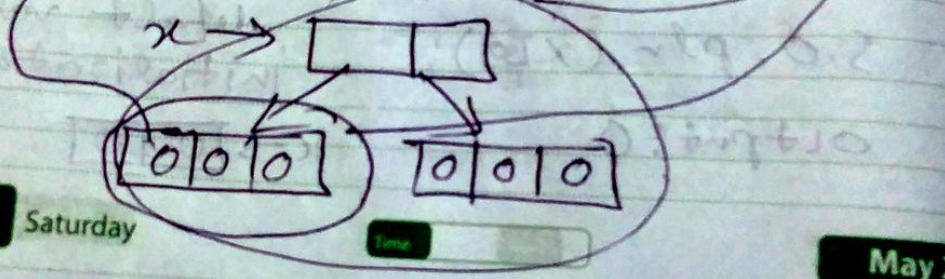
May 2016

Sat	Sun	Mon	Tue	Wed	Thu	Fri
	1	2	3	4	5	6
7	8	9	10	11	12	13
14	15	16	17	18	19	20
21	22	23	24	25	26	27
28	29	30	31			

06 Friday

Time

```
int[][] x = new int[2][3];
S.O.P (x); [[I@32ef23
S.O.P(x[0]); [I@22ef24
S.O.P(x[0][0]); 0
```



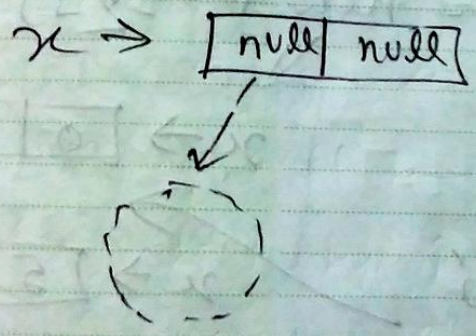
07 Saturday

Time

08 Sunday Time May 2016

```
int [][] x = new int[2][];
S.O.P(x); [[I@2fec12
S.O.P(x[0]); null
S.O.P(x[0][0]); R.E: null pointer exception
```

09 Monday Time May 2016



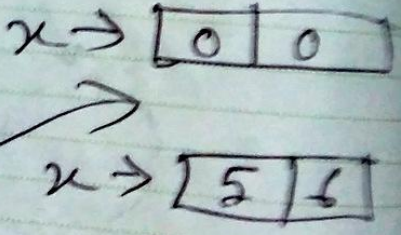
If we are trying to perform any operation on null, then we will get Runtime exception saying null pointer exception.

10 Tuesday Time

Once we create an array, every array ~~elements~~ elements by default ~~initialized~~ initialized with default values (0). If we are not satisfied with default values, then we can override ~~with~~ this values with our customized values.

11 Wednesday Time

```
int[] x = new int[2];
```




```
x[0] = 5;
x[1] = 6;
```



```
x[2] = 10;
```

RE: ArrayIndexOutOfBound exception

```
x[-1] = 10;
```

May 2016
Sun Mon Tue Wed Thu
1 2 3 4 5 6
8 9 10 11 12 13
15 16 17 18 19 20
22 23 24 25 26 27
29 30 31

12 Thursday Time May 2016

x[1.2] = 5; CE: PLP
found: double
required: int

* We can declare, create and initialize an array in a single line (Shortcut Representation)

13 Friday Time May 2016

int[] x;
x = new int[3];
x[0] = 10;
x[1] = 20;
x[2] = 30;

→ int[] x = {10, 20, 30};

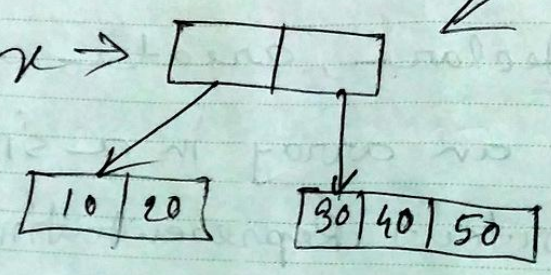
char[] ch = {'a', e', 'b'}

String[] s = {"a", "aa", "ab"}

14 Saturday Time

* We can use this shortcut for multidimensional arrays also.

```
int[][] x = {{10,20},{30,40,50}};
```



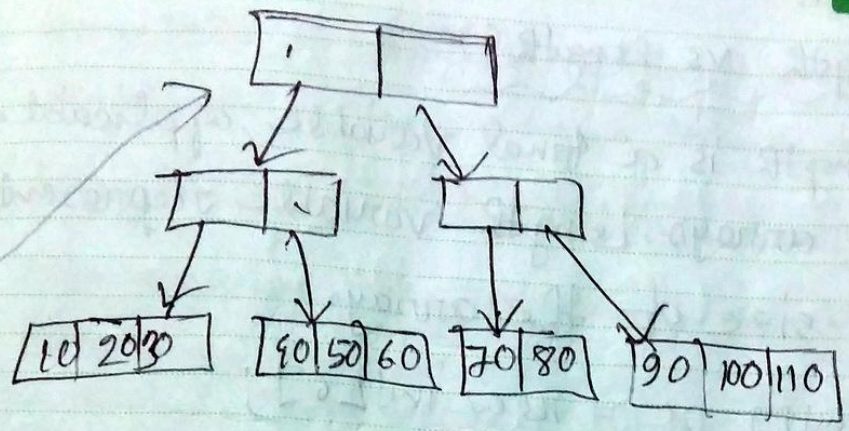
15 Sunday Time

```
int[][][] x = {{{10,20,30},{40,50,60}},{{70,80,
                {90,100,110}}};
S.o.p(x[0][1][2]); o.p: 60
sop(x[1][0][1]); op: 80
S.o.p(x[2][0][0]); RE: AIOOBE
S.o.p(x[1][2][0]); RE: AIOOBE
S.o.p(x[1][1][1]); 100
S.o.p(x[2][1][0]); RE: AIOOBE
```

16 Monday

Time

May 2016



* If we want to use this shortcut, compulsory we should perform all activities in a single line. If we are trying to divide into multiple line then, we will get compile time error.

17 Tuesday

Time

May 2016

```
int [] x = {10,20,30};
```

```
int[] x;
x = {10,20,30};
```

→ CE: illegal start of Expression

18 Wednesday Time May 2016

length vs length()

length is a final variable, applicable for arrays. Length variable represents the size of the array.

```
int[] x = new int[6];
S.O.P(x.length());
S.O.P(x.length);
```

o.p: 6

19 Thursday Time May 2016

CE: cannot find symbol
symbol: method length()
location: class int[]

20 Friday Time May 2016

length() method is a final method, Applicable for String objects. length() method returns number of character present in the string.

```
String s = "Hello";
s.o.p(s.length);
```

21 Saturday Time May 2016

```
s.o.p(s.length());
```

o.p: 5

CE: can not find symbol
symbol: variable length
location: class Java.lang.String

May 2016
Sat Sun Mon Tue Wed Thu Fri

length variable applicable for arrays but not for String objects, whereas length() method applicable for String object but not for arrays.

22 Sunday

String[] s = {"A", "AA", "AAA"};

✓ (I) S.O.P(s.length); 3

✗ (II) S.O.P(s.length()); → CE: can not find symbol
symbol: method length()
location: class String[]

✗ (III) S.O.P(s[0].length); → CE: can not find symbol
symbol: variable length
location: class I.R.String

✓ (IV) S.O.P(s[0].length()); 1

23 Monday

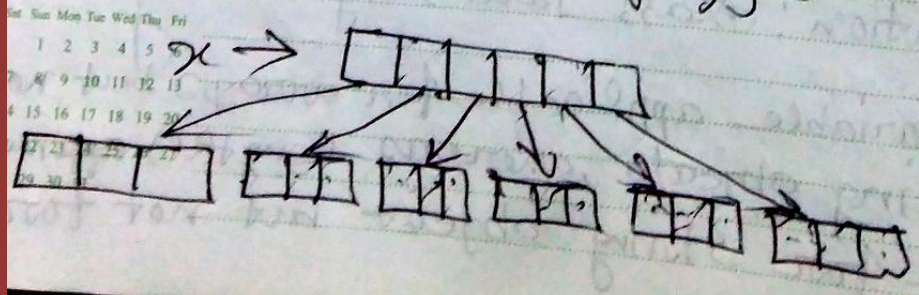
May 2016

* In multidimentional arrays, length variable represent only base size but not total size.

int[][] x = new int[6][3];

S.O.P(x.length); 6

S.O.P(x[0].length); 3



24 Tuesday Time

May 2016

There is no direct way to specify total length of multidimentional array. But indirectly we can find as follows.

x[0].length + x[1].length + --- + x[5].length

* Annonymous Array: Sometimes we can declare an array without name, such type of nameless arrays are called annonymous arrays.

25 Wednesday Time

May 2016

The main purpose of annonymous array is just for instant use (one time use).

We can create annonymous arrays as follows:

new int[] {10, 20, 30, 40} ✓

May 2016
Sun Mon Tue Wed Thu Fri
1 2 3 4 5
8 9 10 11 12 13
15 16 17 18
22 23 24 25 26 27
29 30 31

while creating annonymous arrays we can't specify the size, otherwise we will get compile time error:

new int[3] {10, 20, 30} ✗, new int[] {10, 20, 30} ✓

26 Thursday Time

we can create multidimensional annonymous arrays also:

new int[][]{{10,20},{30,40,50}};

Based on our requirement we can give the name for annonymous array, then it is no longer annonymous.

int[] x = new int[]{10,20,30};

27 Friday Time

Array element assignments:

Case 1: In the case of primitive type arrays, as array elements we can provide any type which can be implicitly promoted to declain these type:

int[] x = new int[5];

x[0] = 10;

x[1] = 'a';

byte b = 20;

x[2] = b;

short s = 30;

x[3] = s;

x[4] = 10L; → CE: PLP found: long required: int

28 Saturday Time May 2016

Case 2: In the case of object type arrays, as array elements, we can provide, either declaired type object or its child class object.

Ex1:

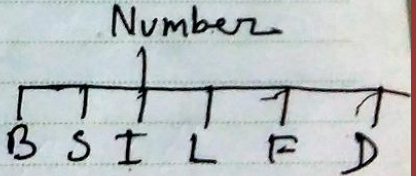
```
Object[] a = new Object[10];
a[0] = new Object();
a[1] = new String("durga");
a[2] = new Integer(10);
```

29 Sunday Time May 2016

Ex 2:

```
Number[] n = new Number[10];
n[0] = new Integer(10);
n[1] = new Double(10.5);
n[2] = new String("durga");
```

CE: incompatible types
found: j.l.String
required: j.l.Number



30 Monday Time May 2016

Case-3: For interface type arrays, as array elements, its implementation class objects are allowed:

```
Runable[] r = new Runable[10];
✓ r[0] = new Thread();
✗ r[1] = new String("durga");
```

Runable(I) ↓ Thread

CE: incompatible types
found: java.lang.String
required: Runable

31 Tuesday Time May 2016

June 2016 — Graphic Arts Institute

01 Wednesday — Time — June 2016

Array type	Allowed element type
primitive type Arrays	Any types which can be implicitly promoted to declared type.
Object type Arrays	Either declared type or its child class objects.
Abstract class type arrays	Its child class object.

02 Thursday — Time — June 2016

interface type arrays	Its implementation class objects are allowed.

03 Friday Time June 2016

Array variable assigment:

Case 1: Element level promotions are not applicable at array level. For example char element can be promoted to int type where as char array can not be promoted to int array.

```
int[] x = {10, 20, 30};
char[] ch = {'a', 'b', 'c'};
```

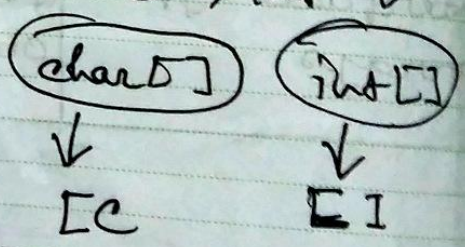
04 Saturday Time June 2016

✓ `int[] b = x;`

✗ `int[] c = ch;`

CE: incompatible types
found: char[]
required: int[]

char → int ✓



05 Sunday

Time

June 2016

Which of the following promotions are perform automatically:

✓ char → int

✗ char[] → int[]

✓ int → double

✗ int[] → double[]

✗ float → int

✗ float[] → int[]

06 Monday

Time

June 2016

✓ String → Object

✓ String[] → Object[]

But in the case of Object type arrays, child class type array can be promote to parent class type array.

Ex:

```
String s = {"A", "AA", "AAA"};
Object[] o = s;
```

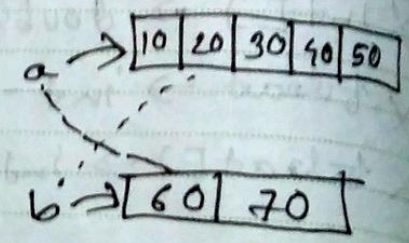
June 2016
Sat Sun Mon Tue Wed Thu Fri
1 2
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30

07 Tuesday Time June 2016

Case-2: Whenever we assigning to one array to another array internal element can't be copied, just reference variable will be reassigned.

```
int[] a = {10,20,30,40,50}
int[] b = {60,70}
```

1. a = b
2. b = a



08 Wednesday Time June 2016

Case-3: Whenever we are assigning one array to another array, ~~internal element~~ the dimention must be matched.

Ex: In the place of one dimentional int[] we should provide one dimention array only.

If we are trying to provide any other dimention, then we will get compile time error.

Graphic Arts Institute

09 Thursday

Time

June 2016

```
int[][] a = new int[3][];
```

✗ `a[0] = new int[4][2];` → CE: incompatible types
found: int[][]
required: int[]

✗ `a[0] = 10;` → CE: incompatible type
found: int
required: int[]

10 Friday

Time

June 2016

✓ `a[0] = new int[2];`

** Whenever we assigning one array to another array, both dimention and type must be matched, but sizes are not required to match.

June 2016

Sat	Sun	Mon	Tue	Wed	Thu	Fri
				1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	

11 Saturday Time June 2016

*Types of Variables: Based on type of value represented by a variable, all variables are divided into two types:

1. Primitive Variables: Can be used to represent primitive values.
Ex: int x = 10;

12 Sunday Time June 2016

2. Reference Variables: Can be used to reffer objects.
Ex: Student x = new Student();

x → Object (new Student())

13 Monday Time June 2016

Division Two: Based on position of declaration and behaviour all variables are divided into three types:

(I) Instance Variable

(II) Static Variable

(III) Local variables

14 Tuesday Time June 2016

Instance Variable: If the value of a variable is varied from object to object, such type of variables are called Instance variables.

For every object a separate copy of instance variable will be created.

15 Wednesday June 2016

* Instance variable should be declared with in the class directly but outside of any method or block or constructor.

* Instance variable will be created at the time of object creation and destroyed at the time of object distruction. Hence the scope of Instance variable is

16 Thursday June 2016

exactly same as scope of object.

* Instance variable will be stored in the heap memory as the part of object.

17 Friday

Time

June 2016

* We can't access instance variable directly from static area, but we can access by using object reference.

But we can access instance variable directly from instance area.

* For instance variable JVM will always provide default values and we are not required to perform initialization explicitly.

18 Saturday

Time

June 2016

```
class Test {
  int x;
  double d;
  boolean b;
  String s;
  p.s.v. main(String[] args){
    Test t = new Test();
    S.o.p(t.x); → 0
    S.o.p(t.d); → 0.0
    S.o.p(t.b); → false
    S.o.p(t.s); → null
  }
}
```

19 Sunday Time June 2016

* Instance variable also known as Object level variables or attributes.

Static Variable:

If the value of a variable is not varied from object to object then it is not recommended to declair variable as instance variable.

20 Monday Time June 2016

We have to declair, such type of variable as class level, by using static modifier.

* In the case of instance variables for every object a separate copy will be created but in the case of static variable a single copy will be created at class level and shared by every object of the class.

June 2016

Graphic Arts Institute

21 Tuesday

Time:

June 2016

* Static variable should be declared with in the class directly but outside of any method or block or constructor with static keyword.

* Static variable will be created at the time of class loading and destroyed at the time of class unloading, hence the scope of static variable is exactly same as scope of .class file.

22 Wednesday

Time:

June 2016

23 Thursday Time

Java Test

1. Start JVM
2. Create and Start Main Thread
3. Locate Test.class file
4. Load Test.class (Static Variable creation)
5. Execute Main method
6. Unload Test.class (Static Variable destruction)
7. Terminate main Thread
8. Shut Down JVM

24 Friday Time

Static variable will be stored in method area.

We can access static variable either by object variable or class name. But recommended to use by class Name. Within the same class it is not required to use class name and we can access directly.

25 Saturday Time June 2016

```
class Test {
  static int x = 10;
  public static void main(String[] args){
    Test t = new Test();
    S.O.P (t.x);
    S.O.P (Test.x);
    S.O.P (x);
  }
}
```

26 Sunday Time June 2016

We can acces static variable directly from both instance and static areas.

```
class Test{
  static int x = 10;
  P.S.V. main(String[] args){
    S.O.P(x);
  }
  public void m1(){
    S.O.P(x);
  }
}
```

June 2016
Sat Sun Mon Tue Wed Thu Fri
1 2 3
4 5 6 7 8 9 10
11 12 13 14 15 16 17
18 19 20 21 22 23 24
25 26 27 28 29 30

27 Monday June 2016

* For static variables JVM will provide default values and we are not required to perform initialization explicitly.

```
class Test {
    static int a;
    static double d;
    static String s;
```

28 Tuesday June 2016

```
    P. S. V. main(String[] args){
        S.O.P(a); → 0
        S.O.P(d); → 0.0
        S.O.P(s); → null
    }
}
```

output

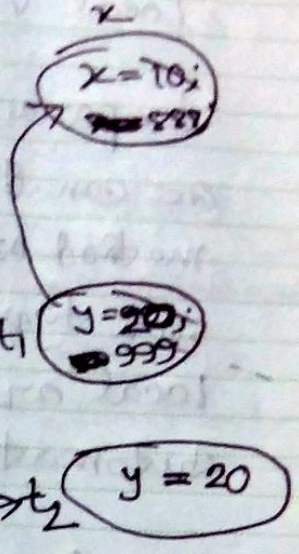
* Static variable also known as class level variable or fields

29 Wednesday

Time

June 2016

```
class Test{
  static int x=10;
    int y=20;
  p.s.v. main(String[] args){
  Test t1 = new Test();
    t1.x = 888;
    t1.y = 999;
  Test t2 = new Test();
  S.O.P(t2.x+"..."+t2.y);
  }
}
```



30 Thursday

Time

June 2016

O/P : 888 ... 20

June 2016

Sun	Mon	Tue	Wed	Thu	Fri
			1	2	3
5	6	7	8	9	10
12	13	14	15	16	17
19	20	21	22	23	24
26	27	28	29	30	

01 Friday Time

Local Variables: Sometimes to meet temporary requirements of the programmer we can declair variable inside a method or block or constructor, such type of variables are called Local or Temparary or Stack or Automatic variables.

Local variable will be stored inside

02 Saturday Time

stack memory.

Local variable will be created, while executing the block in which we declaired it, Once block execution completes, automatically local variable will be destroy. Hence the scope of local variable is the block in which we declaired it.

July 2016 — Graphic Arts Institute

03 Sunday — Time — July 2016

```
class Test{
  P.S.V. main(String[] args){
    int i=0;
    for(int j=0; j<3; j++){
      i=i+j;
    }
    S.O.pln(i+"..."+j);
  }
}
```

CE: Can not find symbol
symbol: variable j
location: class Test

04 Monday — Time — July 2016

```
class Test{
  P.S.V. main(String[] args){
    try{
      int j = Integer.parseInt("ten");
    }catch(NumberFormatException e){
      j=10;
    }
    S.O.P(j);
  }
}
```

July 2016
Sat Sun Mon Tue Wed Thu Fri
30 31 1
2 3 4 5 8 14 15
9 10 11 12 16 17 18 19 20 21 22
23 24 25 26 27 28 29

05 Tuesday Time July 2016

* For local variables JVM won't provide default values, compalsury we should perform initialization explicitly, before using that variable. That is if we are not using, then it's not required to perform initialization.

06 Wednesday Time July 2016

```
class Test{
P.S.V.main(String[] args){
  int x;
  S.O.P("Hello");
 }
}
```

✓

```
class Test{
P.S.V main(String[] args){
int x;
S.O.P(x);
}
}
```

→ CE: variable x might not have been initialization

07 Thursday Time July 2016

The only applicable modifier for local variable is final. By mistake if we are trying to apply any other modifier then we will get compile time error.

If we are not declaring with any modifier, then by default it is default but this rule,

08 Friday Time July 2016

is applicable only for instance and static variables but not for local variables.

Instance and static variables can be accessed by multiple threads simultaniously and hence these are not thread safe. But for local variables, for every thread separate copy will be created, that's why local variables are thread safe.

July 2016
Sat Sun Mon Tue Wed Thu Fri
30 31 1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28

09 Saturday Time

Local variable:

① int[] x;

S.O.p(x);

S.O.p(x[0]);

} → CE: variable x might not have been initialized

② int[] x = new int[3];

S.O.p(x); [I@1256 ✓

S.O.p(x[0]); 0 ✓

10 Sunday Time July 2016

Once we creates an array, every array element by default initialized with default values. Whether it is instance, static, or local array.

11 Monday Time July 2016

Var-Arg Methods (Variable number of argument methods):

Until 1.4v we can not declair a method with variable number of arguments. If there is a change in number of arguments compalsury we should go for new method. It increasaes length of the code and reduces readability.

12 Tuesday Time July 2016

To over come this problem SUN people introduces var-arg methods in 1.5 v. According to this we can declair a method, which can take variable number of arguments such type of methods are call var-arg methods.

July 2016

Sat	Sun	Mon	Tue	Wed	Thu	Fri
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

13 Wednesday Time July 2016

m1(int ... x)

we can declair a var-arg method as above.

We can call this method by passing any number of int values, including 0 number.

m1(); ✓ m1(10, 20); ✓

m1(10); ✓

14 Thursday Time July 2016

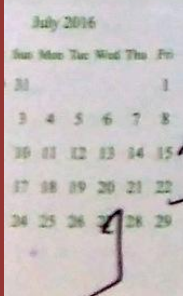
```
public static void m1(int ... x){
    S.o.p("var-arg");
}

public static void main(String[] args){
    m1();  ✓
    m1(10); ✓
    m1(10,20); ✓
}
```

15 Friday Time July 2016

Internally var-arg parameter will be converted into one dimentional array. Hence within var-arg method we can differentiate values by using index.

```
class Test {
  public static void main(String[] args){
    sum();
    sum(10,20);
    sum(10,20,30);
    sum(10,20,30,40);
  }
  public static void sum(int ... x) {
    int total = 0;
    for(int x1 : x) {
      total = total + x1;
    }
    System.out.println("total : "+total);
  }
}
```

16 Saturday Time July 2016



17 Sunday Time July 2016

m1(int x, int... y) ✓
m1(String s, double... y)

m1(double... d, String s) ✗

m1(char ch, String... s) ✓

var-arg parameter should be last parameter.

18 Monday Time July 2016

m1(int... x, double... d) ✗

inside var-arg method we can take only one var-arg parameter and we can't take more than one var-arg parameter.

July 2016

Sat	Sun	Mon	Tue	Wed	Thu	Fri
30	31					1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29

July 2016

Graphic Arts Institute

19 Tuesday

Time

July 2016

main() method:

Whether class contain main method or not, and whether main method is declaired according to requirement or not, these things won't be checked by compiler. At runtime JVM is responsible to check these things.

20 Wednesday

Time

July 2016

If JVM unable to find main method, then we will get runtime exception said:

No such method error: main

class Test{

}

javac Test.java ✓

Java Test → RE: NoSuchMethodError: main

July 2016

Sat Sun Mon Tue Wed Thu Fri

30 31 1

2 3 4 5 6 7 8

9 10 11 12 13 14 15

16 17 18 19 20 21 22

23 24 25 26 27 28 29

21 Thursday

Main Method Syntax

public static void main (String[] args)

- public → to call by JVM from anywhere
- static → without existing object also JVM has to call this method
- void → main() method won't return anything to JVM
- main → this is the name which is configured inside JVM
- (String[] args) → command line arguments

22 Friday

At runtime JVM always searches for the main() method with the following prototype:

July 2016

Sun	Mon	Tue	Wed	Thu	Fri
31					1
3	4	5	6	7	8
10	11	12	13	14	15
17	18	19	20	21	22
24	25	26	27	28	29

23 Saturday Time July 2016

(i) static public ✓

(ii) main (String[] args) ✓

main (String []args) ✓
main (String args[]) ✓

(iii) main (String[] abc) ✓

(iv) main (String... args) ✓

24 Sunday Time July 2016

Even though above syntax is very strict, the following changes are acceptable :)

(i) instead of public static we can take static public, that is the order of modifiers is not important.

25 Monday Time July 2016

(iii) We can declair string array in any acceptable form.

(iii) Instead of args, we can take any valid java identifier.

(iv) We can overplace String[] with varargs parameters.

26 Tuesday Time July 2016

We can declair main method with the following modifiers:

- ✓ final
- ✓ synchronized
- ✓ strictfp

static final synchronized strictfp public void main (String... args) {
    S.O.P("valid");
}

27 Wednesday Time July 2016

case 1: Overloading of the main method is possible, but JVM will always call String[] args main method only. The other overloaded method we have to call explicitly, like normal method call.

28 Thursday Time July 2016

```
class Test{
    public static void main(String[] args)
    {
        S.O.P("string");
    }
    public static void main(int[] args)
    {
        S.O.P("int[]");
    }
}
```

Overloaded methods

o/p: String

29 Friday Time July 2016

case2;

P.Java

```
class P {
public static void main(String[] args){
S.O.P("Parent main");
}
}
```

· class C

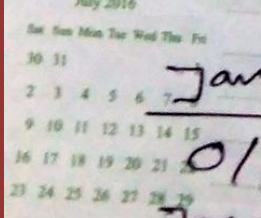
~~public~~ static void ~~main~~(s

30 Saturday Time July 2016

```
class C extends P {
}
```

Java C P.Java

P.class C.class



Java P ↵

O/P: Parent main

Java C ↵

O/P: Parent main

31 Sunday

Time

July 2016

Inheritance concept applicable for main method, hence while executing child class, if child doesn't contain main method, then parent class main method will be executed.

01 Monday

Time

August 2016

Case 3:

```
class P {
    P S V main (String[] args) {
        S.O.P("Parent main");
    }
}
class C extends P {
    P.S V main (String[] args) {
        S.O.P("child main");
    }
}
```

P.Java method hiding but not method overriding

Javac P.Java

P.class C.class

Java P ←

O/P: "Parent main

Java C ←

O/P: child main

04 Thursday Time August 2016

It seems overridding concept applicable for main method, but it is not overridding and it is method hiding.

for main method inheritance and overloading concepts are applicable but overriding concept is not applicable.

05 Friday Time August 2016

Instead of overriding method hiding is applicable.

August 2016

Sat	Sun	Mon	Tue	Wed	Thu	Fri
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

06 Saturday

Time

August 2016

class Test 2

3

1.6 V

javac Test.java ✓

java Test

RE: NoSuchMethodError: main

1.7 V

javac Test.java ✓

java Test

Error: Main method not found in class Test, please declare main method as:

public static void main (String[] ar

07 Sunday

Time

August 2016

August 2016

Sun	Mon	Tue	Wed	Thu	Fri
	1	2	3	4	5
7	8	9	10	11	12
14	15	16	17	18	19
21	22	23	24	25	26
28	29	30	31		

08 Monday Time

August 2016

```
class Test{
    static{
        System.out.println("Static Block");
    }
}
```

Java 1.6 ✓	1.7 ✓
Javac Test.java ✓	Javac Test.java ✓
Java Test ←	Java Test ←

09 Tuesday Time

August 2016

O/p: Static Block	Error: Main method not found in class test.
RE: NoSuchMethodError: main	

From 1.7v main method is mandatory, to start program execution, hence eventhough class contain ~~main method~~, static block, it won't be executed if the class doesn't contain main method.

10 Wednesday Time


```
class Test{
    static{
        System.out.println("static block");
        System.exit(0);
    }
}
```

1.6 V

Javac Test.java

Java Test

O/P: static Block

1.7 V

Javac Test.java

Java Test

Error: Main method not found in class Test.

11 Thursday Time

12 Friday Time August 2016

```
class Test{
  static{
    System.out.print("Static Block");
  }
  [redacted]
  p: s v main(String[] args){
    Syste.out.println("main method");
  }
}
```

13 Saturday Time August 2016

1.6 V	1.7 V
Javac Test.java	Javac Test.java ✓
Java Test ←	Java Test ←
o/p: Static Block	o/p: Static Block
Main Method	main method

14 Sunday Time August 2016

Command line Arguments:

The arguments which are passing from command prompt are called command line arguments. With this command line arguments JVM will create an array and by passing that array as argument, JVM will call main method.

15 Monday Time August 2016

```
Java Test A B C
     args[0]
     args[1]
     args[2]
     args.length => 3
```

16 Tuesday Time August 2016

Purpose of command line arguments:

The main objective of command line argument is, we can customize behaviour of the main method.

Case 1:

```
class Test {
```

17 Wednesday Time August 2016

```
public s v main(String[] args){
for(int i=0; i<=args.length; i++){
    S.O.P(args[i]);
}
}
}
```

java Test A B C ↵ A B C RE: A1OB Ex

java Test A B ↵ A B RE: A1OB Ex

java Test ↵ RE: A1OB Ex

A1OB → Array index out of Bound Ex

August 2016
Sat Sun Mon Tue Wed Thu Fri
1 2 3 4
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

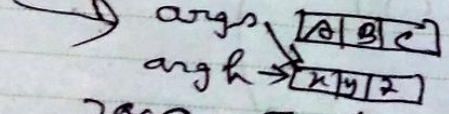
18 Thursday Time August 2016

Case 2:

```
class Test {
public s v main(String[] args){
String[] argh = {"x", "y", "z"};
args = argh;
for(String s : args){
S.o.p(os);
}
}
}
```

19 Friday Time August 2016

java Test A B C ↵ x y z



java Test A B ↵ x y z

java Test ↵ x y z

20 Saturday Time August 2016

Case 3:

```
class Test {
public s v main(String[] args){
s.o.p(args[0] + args[1]);
}
}
```

Java Test 10 30 ↵ 1020

Within main method command line arguments are available in String form.

21 Sunday Time August 2016

Case 4: Class Test {

```
public s v main(String[] args){
s.o.p(args[0]);
}}
```

Java Test "Note Book"

O/p: Note Book

August 2016

Sat	Sun	Mon	Tue	Wed	Thu	Fri
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

22 Monday

Time

Java coding Standards:

```
class A
{
  public int m1(int x, int y)
  {
    return x+y;
  }
}
```

23 Tuesday

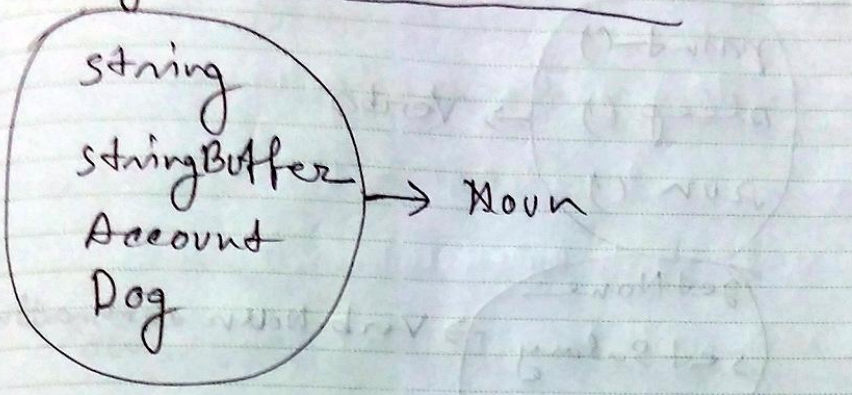
Time

```
package com.moudud.hassan

public class Calculator
{
  public static int add(int num1, int num
  {
    return num1+num2;
  }
}
```

August 2016
Sat Sun Mon Tue Wed Thu Fri
1 2 3 4 5
7 8 9 10 11 12
14 15 16 17 18
21 22 23 24 25 26
28 29 30 31

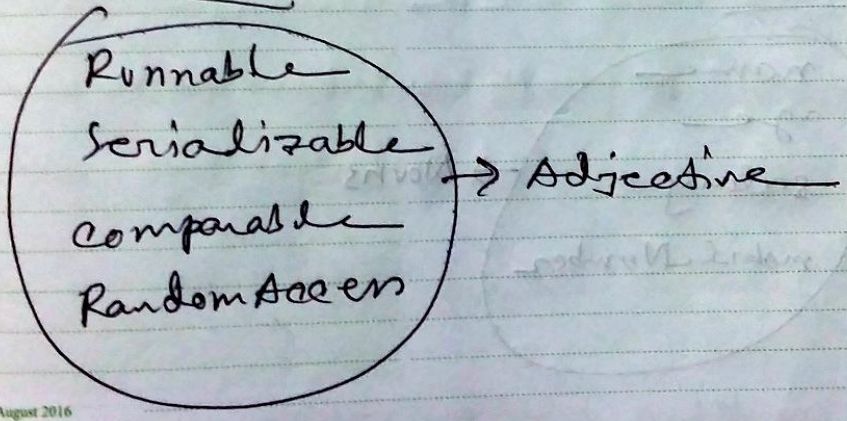
24 Wednesday Time August 2016

Coding standard for classes:



25 Thursday Time August 2016

Interface:

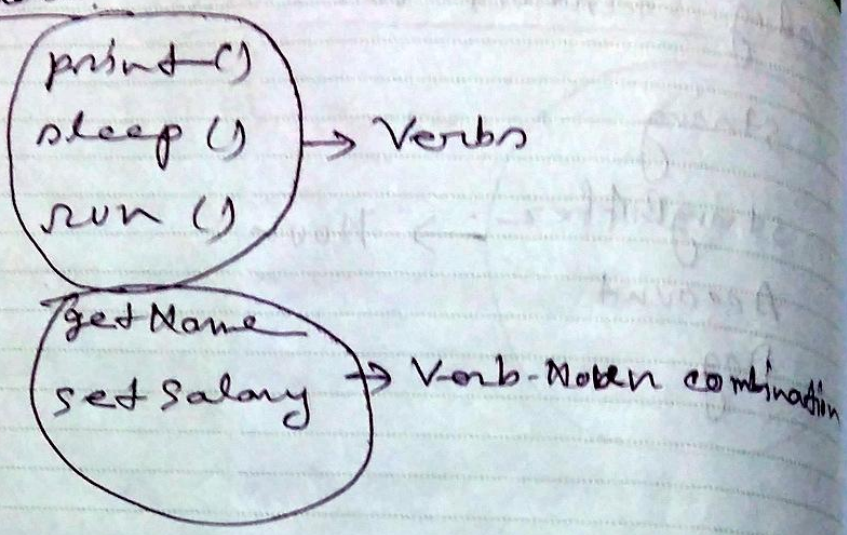


August 2016

Sat	Sun	Mon	Tue	Wed	Thu	Fri
		1	2	3	4	5
6	7	8	9	10	11	12
13	14	15	16	17	18	19
20	21	22	23	24	25	26
27	28	29	30	31		

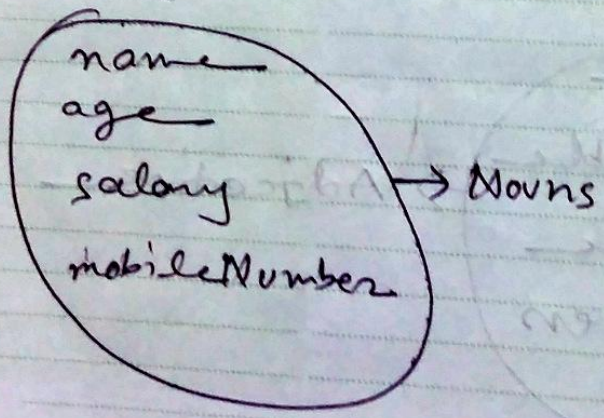
26 Friday Time August 2016

* Methods:



27 Saturday Time August 2016

* Variable:



August 2016
Sat Sun Mon Tue Wed Thu Fri
1 2 3 4 5
6 7 8 9 10 11 12
13 14 15 16 17 18 19
20 21 22 23 24 25 26
27 28 29 30 31

August 2016

28 Sunday

Graphic Arts Institute

August 2016

Constant:

MAX_VALUE
MAX_PRIORITY → nouns
NORM_PRIORITY
PI

This value are fixed, not changeable.

public static final int MAX_VALUE = 5;

29 Monday

August 2016

JavaBean? → Simple Java class

```
public class StudentBean
{
  private String name;

  public void setName(String name)
  {
    this.name = name;
  }

  public String getName()
  {
    return name;
  }
}
```

30 Tuesday Time August 2016

Listeners:

Case 1: To register a listener:

✓ public void addMyActionListener(MyActionListener l)

✗ public void registerMyActionListener(MyActionListener l)

✗ public void addMyActionListener(ActionListener l)

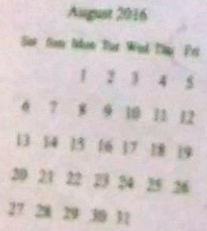
should be prefixed with "add" for method

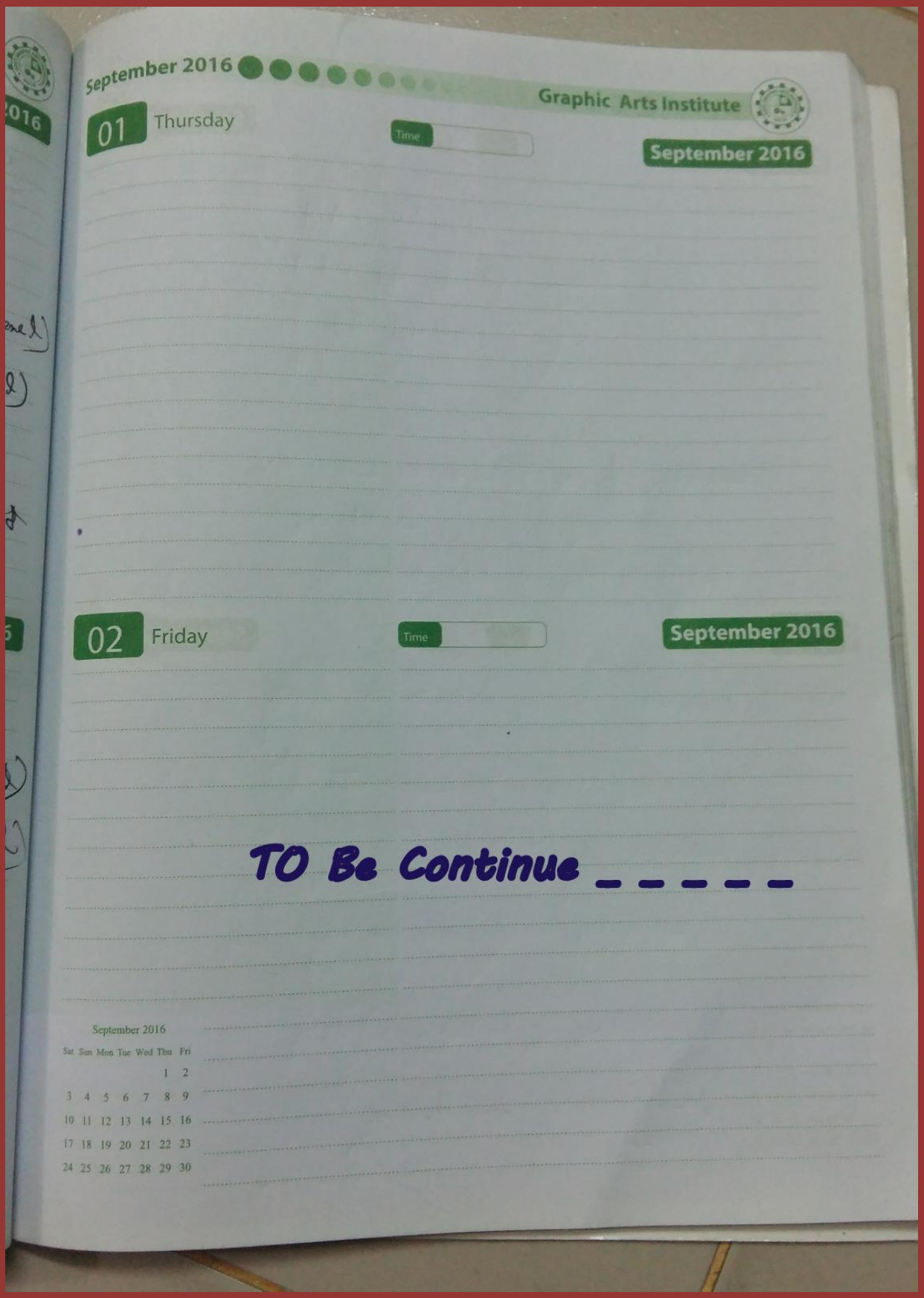
31 Wednesday Time August 2016

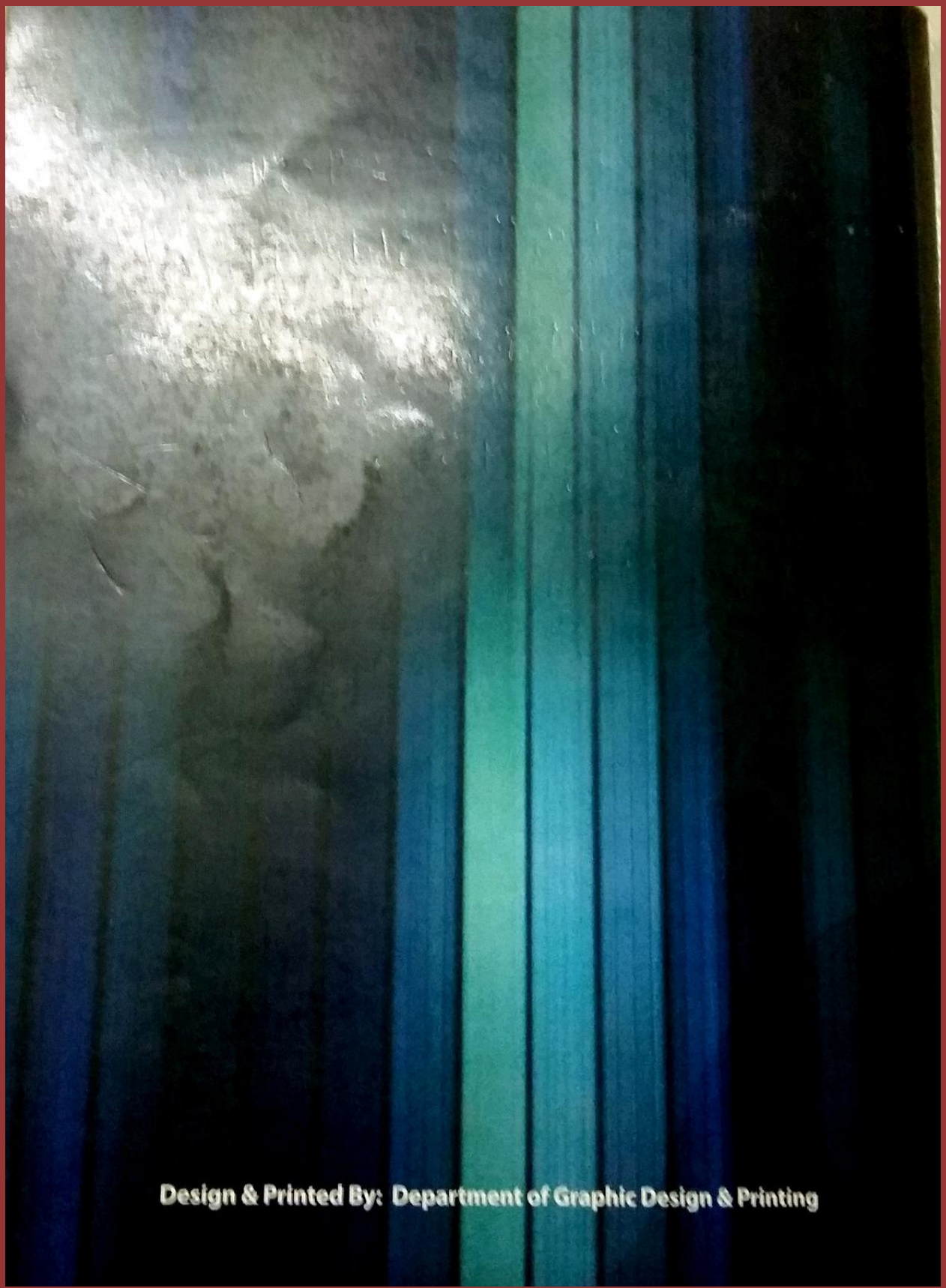
Case 2: To unregister a listener:

✓ public void removeMyActionListener(MyActionListener l)

✗ public void unregisterMyActionListener(MyActionListener l)




September 2016
Graphic Arts Institute
01 Thursday
Time
September 2016
02 Friday
Time
September 2016
TO Be Continue _ _ _ _ _
September 2016
Sat Sun Mon Tue Wed Thu Fri
1 2
3 4 5 6 7 8 9
10 11 12 13 14 15 16
17 18 19 20 21 22 23
24 25 26 27 28 29 30


Design & Printed By: Department of Graphic Design & Printing