Course 2

Getting Started with Programming in Java 11

First java application

Project Name
└→ src
  └→ Main.java

Entry point for java program is Main class which runs a modifier main.

eg

```
public class Main {
    public static void main (String[] args) {
        System.out.println ("Hello World");
    }
}
```

to run in command line

1) If build is done, send app as Main.class
to run > java Main

2) If we have Main.java
so javac Main.java
java Main

Statement Structure

- Programs are made up of statements
- Java statements end with semicolon

eg

```
System.out.println("Hello");
```

Comments

1) Line comment //...
   Text ignored until end of current line

2) Block comment /*...*/
   Text ignored within the block

3) Javadoc comment /**...*/
- Compiler treats similar to block comment
- Can be used to generate documentation
- formatting is different

Packages

- Packages Provide organization
- follow standard naming convention
- Affect source code structure

Package naming convention

1) all lowercase
2) use reverse domain name notation to assure global uniqueness
   eg package com.pluralsight ;
3) add further qualifiers to assure uniqueness within a company or group
   eg package com.pluralsight.search ;
   or package com.pluralsight.sales.accountManage ;

Organization

if want package as com.pluralsight.search

So Project Name
- out
- src
  - com
    - pluralsight
      - search
        - Main.java

in code

```
package com.pluralsight.search ;

public class Main {
    public static void main (String[] args) {
        - - -
    }
}
```

Variables

- Named data storage
- strongly typed

eg

```
int dataValue;
dataValue = 100;
```

or

```
int myAge = 20;
```

Naming convention →

1) use camel case
2) Use only letters and numbers
3) first character can not be number

if we want value of variable won't modify use final keyword

eg

```
final int maxV = 200;
```

value of maxV can't be changed

Primitive Data types

- Integer data types :-

byte	8 bits	-128 to 127
short	16	-32768 to 32767
int	32	-2^{32} to $2^{32}-1$
long	64	-

For long variable, have to add: Literal L at end eg:

```
long milesLightYear = 5879000L;
```

- Floating Point types :-

float	32 bits	Literal 0.0f
double	64	0.0 or 0.0d

eg

```
float km = 42.195f;
```

- Character data types -
  - stores a single Unicode character
  - Literal values placed between single quotes
  - For Unicode code points, use \u followed by 4-digit hex value

eg

```
char regularU = 'U';
char accentedU = '\u00DA'; // Ú
```

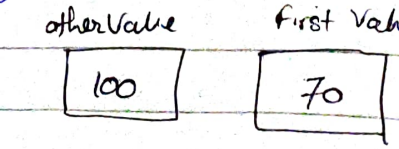
- Boolean data types -
  - true or false

eg

```
boolean iLoveJava = true;
```

→ primitive types are stored by value

```
int firstValue = 100;
int otherValue = firstValue;
firstValue = 70;
```



Arithmetic Operators

- Basic operators
  1) Add (+)
  2) Subtract (-)
  3) Multiply (*)
  4) Divide (/)
  5) Modulus (%)

eg

13.0 / 5.0 = 2.6 , 13 / 5 = 2

13.0 % 5.0 = 3.0 , 13 % 5 = 3

- Prefix and Postfix Operator
  1) ++ → increment
  2) -- → Decrement

→ prefix applies operation before returning value

→ postfix applies operation after returning value

- Compound Assignment operators
  - apply right side value to left side
  - store result in variable on left side

  +=, -=, *=, /=, %=

Precedence

Postfix > Prefix > * / % > + -
(* / % : multiplicative)

Operators of equal precedence evaluated left to right

Type Conversion

explicit conversion using cast operator

eg

```
long value = 50;
int iValue = (int) value;
```

CONDITIONAL LOGIC

Relational Operators

*) > , >= , <= , <

== (equal to) , != (not equal)

Conditional assignment

Condition ? true-value : false-value

if - else

eg

```
if (condition) {
    true-statment; }
else {
    false-statement; }
```

can do nesting

```
if (condition)
- - -
else if (condition-2)
- - -
:
else if (condition-N)
- - -
```

Logical operators

- produce single true or false result from two true or false values
- may combine 2 relational tests or 2 booleans

eg

And	&	also operation 2 & 3 = 2
Or	|	
XOR	^	F^T , t^F = T
Negation	!	

Conditional Logical operators

- Similar to standard logical operators
- Right side executes only when needed
  eg && executes only when left is true

=

And	&&
Or	||

Switch

```
switch (value-to-test) {
    case matching-value-1 :
        statments
        break ;
    .
    .
    .
    default :
        statements
}
```

: only support → int, char, long, byte, short

LOOPS & ARRAYS

1) while loop

```
int a = 4;                    // factorial
int ans = 1;
while (a>1) {
    ans *= a;
    a--;
}
System.out.println(ans);
```

2) Do-while loop

```
do {
 - - -
} while (a<25);
```

3) for loop

```
for (initialize ; condition ; update)
    statement;
```

eg `for (int i = 1; i<100; i*=2)`

4) for-each loop

used for arrays => `for (loop variable : array)`
`statement;`

eg

```
for (float x : val)        -> array
    ans += x;
```

Arrays

1) float[] val = { 10.0f , 20.0f };

2) float[] val = new float[3];
   val[0] = 10.0f;
   val[1] = 20.0f;

3) for length → val.length
4) int[][] flats = new int[2][3]; // 2-D array.

METHODS / FUNCTIONS

- Mechanism for organizing code
- Enables to create reusable code blocks
- can receive & return data

Naming :-
same rules & conventions as variables, camelCase

typed - parameter list :-
- allow data values to be passed in
- can be empty

eg :- name (typed - parameter - list) {
}

eg:-

```
static void doSomething() {
    System.out.println("Inside method");
}
```

Parameters are passed by value

returning a value :-

eg

```
static double calInterest(double amt, double rate, int years)
{
  double interest = amt * rate * years;
  return interest;
}
```

return year wise interest, return array

```
static double[] interest(double amt, double rate, int years)
 {
  double [] ans = new double [years];
  for (int i=0; i<years, i++) {
    int year = i+1;
    ans [i] = amt * rate * year;
  }
    return ans;
}
```

static helps to run a function of class without making object of that class

Date 26/5
Page 16

STRINGS

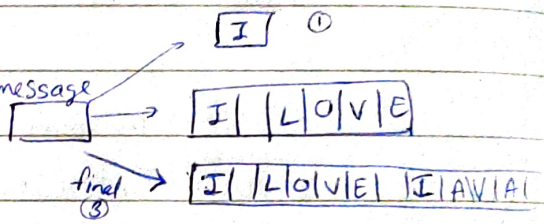
eg String name = "Jack";

- strings can be concatenated with +
- strings are immutable

- string variables do not directly hold string value, they hold reference to instance of string

eg

String message = "I"; ①
message += "Love"; ②
message += "Java"; ③



String equality :-

use .equals method

eg if (S1.equals(S2))

- if use (==) so it does not do character by character comparison, rather check by reference if S1 & S2 reference same string so S1 == S2 is true, else false

So better use equals method

String class methods

1) Length
   length eg name.length()

2) create new string from existing
   concat, replace, toLowerCase, toUpperCase, trim, split

3) Extract substring
   · charAt, substring eg s.substring(3, 8) (8 not included)

4) Test substring
   contains, endsWith, startsWith, indexOf, lastIndexOf

5) Comparison
   equals, equalsIgnoreCase, isEmpty, compareTo, compareToIgnoreCase

6) formatting
   format

7) String for non-string
   valueOf

Converting non-string Types to String

Virtually all data types can be converted to String

eg

```
int val = 100;
String sVal = String.valueOf(val); // "100"
```

or either implicitly

```
int i = 2, j = 3;
int result = i * j;
String output = i + "*" + j + "=" + result;
```

String Builder

- Provide mutable string Buffer
  - Efficiently constructs string values
  - Add new content to end with append
  - add new content within with insert

extract content to string (convert to normal string)

- Use toString

eg

```
String loc = "Florida";
int num = 175;
StringBuilder sb = new StringBuilder();
sb.append("I flew to ");
sb.append(loc);
sb.append("on flight #");
sb.append(num);
String message = sb.toString();
    // I flew to florida on Flight #175;

String time = "9:00";
int pos = sb.indexOf("on");
sb.insert(pos, "at ");
sb.insert(pos+4, time);
message = sb.toString();
// I flew to Florida at 9:00 on flight #175
```

String formatting

- makes strings more readable specially when it have other variables

eg int david = 17, joe = 15, jack = 8, K = 6;

String s = String.format(
"My nephews are %d, %d, %d and %d years old", david, joe, jack, K);

- also can set precision of ans

eg double avg = 3.6666667d;
string s4 = String.format(
"The average is %.1f years", avg);

format conversions

%.d → int's
%.f → decimal, floating point
%.s → String
%.e, %.E → floating point in scientific notation
eg 123.0 → 1.230000e+02

INPUT FROM USER

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

in code

```
Scanner s = new Scanner(System.in);
```
↳ Read from keyboard

```
System.out.println("Enter number");
int a = s.nextInt();
```

to input float

```
sc.nextFloat();
```

Input String

```
String str = s.next();      // read only 1 letter
String str = s.nextLine();  // read full string
```

Ways to print String

```
System.out.print()    → no new line at the end
System.out.println()  → Prints new line at end
```

```
System.out.printf()
or System.out.format()
```
} for printing with format specifier

eg. `System.out.printf("Value of a & b is %d %f", a, b);`

More About METHODS

Calling a method

Let the function we made is not static, in this case we will need to make its object and then call it.
So we do this as

```
Calc obj = new Calc();   → Object creation
obj.mySum(a, b);         → method call on object
```

- static keyword is used to associate a method of a given class with the class rather than the object. Static method in a class is shared by all the objects.

Method Overloading

Two or more class with same name but different parameters,

eg
```
void foo();
void foo(int a);
void foo(int a, int b);
```

Method overloading can not be performed by changing return type of methods.

Variable Arguments (Varargs)

```
public static void foo (int ...arr)
  {
     // all arguments bind as int[] arr
  }
```

foo can be called with zero or any number of arguments : foo(0) , foo() , foo (1, 2, 3, 9, 8)

⇒ if want atleast one argument &
```
public static void bar (int a, int ...arr)
  {
     // code
  }
```
& can call as bar(1) , bar (1, 7, 8, 9)

- to use arguments
```
public static void bar (int ...arr)
  {
    for (int x : arr)
       System.out.print (x);
  }
```