Identifiers, Reserved keywords

⇒ Data Type in JAVA

Date 13/10/2022

PART - 1

⇒ OOPS:- It stands for object orientation principles.

→ object --> Real time instance or an entity.
e.g. car, Student, Employee

⇒ So, Every object in realtime will have 2 Parts.
(i) Has Part, (ii) Does Part.

e.g.- (CAR)
- brandName
- noofwheels
- model
- speed, color

} Has Part

- move
- accelerate
- brake

} Does-Pars

JAVA Code:-

e.g#1

```
class Car {
    // HAS-PART of an object is represented as a "variable"
    String brandname;
    int noofwheels;
    String model;

    // DOES-PART of an object is represented through "methods"
    public void move()
    {
        // logic of moving a vehicle
    }
    Public void acceleration()
    {
        // logic of acceleration a vehicle
    }
}
```

e.g#2

```
class Student
{
    // HAS-PART (variables)
    String name;
    int id;
    float height;
    // Does-Part (methods)
    Public void Play(){
        // logic of playing
    }
    Public void Study(){
        // logic of studying
    }
}
```

```
Public class Main {
    P.S.V.m(String[] args){
        Student Std = new Student();
        Std.Identity();
}}
```

(*) main() should under a public class. and in a programm there can have so many class. you can make it as normal. But you have to main class as public

(*) **Identifiers:-**

→ It is a name in Java Program.

→ It can be a classname, methodname, variable name and label name.

#① ex:- class Text { → class name, → method name, → class, → variable

```
Public Static void main (String[] args){
    int x = 10;
}}
```

Ans:- Totally 5 identifiers present above.

#②:-
```
class Text{
    Public static void main (String[] args){
        System.out.println ("Sachin");
}}
```
→ final class, → variable, → method

Ans:- Totally 07 identifiers present above

(*) ⇒ **Rules** (syntax for compiler jvm) for writing an **Identifiers**:-

(i) **Rule 1**:- The only allowed characters in Java identifiers are → a to z, A to z, 0 to 9, _ (underscore), $

(ii) **Rule 2**:- If we use any other characters it would result in compile time error.

ex- int total#=10; (inv) int A* = 10; (invalid) / int total = 10; (valid)

(iii) **Rule 3**:- Identifiers are not allowed to start with digits.

e.g.- int sworup01=100 (valid) / int 01sworup = 10; (Invalid)

(iv) **Rule 4**:- Java Identifiers are case sensitive, meaning number and Number is different.

(v) **Rule 5**:- There is no length limit on java identifiers, but still it is a good practice to keep the length of the identifiers not more than 15 characters. (recommanded).

int priority of The Thread With Min Value = 1;

(vi) Rule 6:- we can't use reserve words as a identifiers.

e.g:- int if = 10; // compilation Error

(vii) Rule 7:- Predefined class names can be used as identifiers like String, Runnable, Integer, Byte, Short, Long.

eg#1 → String Runnable = "sachin";

S.o.Pln (Runnable); // sachin

eg#2 → int String = 10; int int = 10; (invalid)

S.o.Pln (String); // 10

Note:- Even though predefined class names can be used as a identifiers, it is not a good practice to keep.

I.Q:) int If = 10; // if and If is different ? (yes, different)

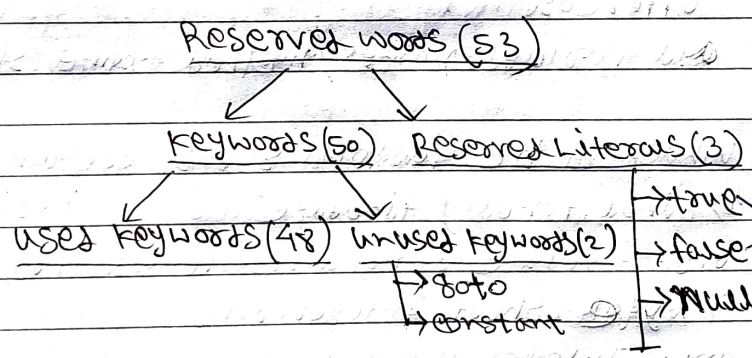
System.out.println(If); // 10 (if) ⇒ reserved word, (If) = Predefined class.

int Integer = 10; // (10)

int int = 10; // (E.E)

ReservedWords:- / Keywords / Built in words:-

It is a built in words/keywords which has already a predefined meaning to it. There total have 53 reserved word present in Java.



⇒ Reserved words for Datatypes (8);

1) byte, 2) short, 3) int,
4) long, 5) float, 6) double,
7) char, 8) boolean.

⇒ Reserved for flow control: (11)

1) if, 2) else, 3) switch, 4) case
5) default, 6) for, 7) do, 8) while
9) break, 10) continue 11) return

OOP's in Java

⇒ keywords for modifiers (11)

1) Public, 2) Private, 3) Protected
4) static, 5) final, 6) abstract
7) synchronized 8) native
9) strictfp (1.2 version) 10) transient
11) volatile.

⇒ class related keywords: (6)

1) class, 2) package, 3) import,
4) extends, 5) implements, 6) interface

P.T.O

OOP's

⇒ object Related keywords: (4)
1) new, 2) instanceof, 3) super, 3) this

Exception Handling

⇒ keywords for Exception Handling (5)
1) try, 2) catch, 3) finally, 4) throw, 5) throws, 6) assert (1.4 version)

void → keyword associated with method

unused keywords: goto, constant.

Literal :-

Literal → any constant value which can be assigned to a variable is called Literal.

⇒ int data = 10; // Literal → 10
// data → variableName / identifier.
// int → datatype / reserveword.

NOTE:- for boolean datatypes the only values allowed for a variable is "true, false", other than this, if we try → to keep any values it would result "compileTimeError".

⇒ all reserved words names would start with "LowerCase".

⇒ In Java all classnames / interfacenames start with "UpperCase"

⇒ all variablenames / method name start with "Lower case"

Q) Which of the following list contain only reservewords / keywords / builtin words?

1) final, finally, finalize
ans:- finalize is not a reserveword, it is a method in object class

2) byte, Short, Integer, long
ans:- Integer is not a reserve word, it is a predefined class

3) break, continue, return, exit
ans:- exit is not a reserve word, it is a method in System class

4) throw, throws, thrown
ans:- thrown is not a reserve word, it is a userdefined variable.

DATA TYPES#

⇒ Every variable has a type, every expression has a type and all types are strictly typed/define in java, beeoz java is strictlytype/statically typed language.

compiler role → compiler will check the value stored can be handled by datatype or not? This checking which is done by compiler is called "Type checking/ Strictlytype checking".

e.g:- int a=10; // variable.
int b=20;
int result = a*b; // expression.

⇒ There are two types of datatypes present →

① Primitive Datatype :-

→ Meaning → data which is commonly used and supported by any language to store directly. Primitive datatype can't be breakable to others type.

a) Numeric values ⇒ to store number.
(i) whole number, (ii) real number.

b) character values ⇒ to store character type of data.

c) boolean values ⇒ to store logical values.

→ Number Data :-
To store whole number we have 4 datatypes.
(i) byte, (ii) short, (iii) int, (iv) long. | real number → float, double.

→ Datatype information like :-
a) size of datatype (how much memory is allocated on the ram for that datatype by JVM).
b) min value what it can keep.
c) max value what it can keep.

commands (for size)

→ Note :-

[⇒ Byte :- System.out.println("size of byte is:" + Byte.SIZE); (class)
System.out.println("min value of byte :" + Byte.MIN_VALUE);
System.out.println("max value of byte is:" + Byte.MAX_VALUE);

output :- size → 8 bits (1 Byte)
min value → -128
max value → 127

e.g: byte marks = 35; (valid) → found: int type, required: byte
byte marks = 135; (C.E: possible-x)
byte marks = -3; (valid) → loss of precission

byte a = true; //C.E (incompatible type) x
byte b = "nittin"; //C.E (") x

OOP's

=> Object related keywords: (4)
1) new, 2) instanceof, 3) super,
4) this

Exception Handling

=> keywords for exception Handling (6);
1) try, 2) catch, 3) finally
4) throws, 5) throw, 6) assert (1.4 ver.)

void -> keyword associated with method

unused keywords: goto, constant

Q) When to use byte datatype?

=> It is commonly used when we handle the five operation or, data which is comming from stream, network.

Stream --> java.io package (we learn here)

=> " " -> means String data.

=> ' ' -> char data.

=> Incompatible type => This is is a error. means except the value of data type.

Possible	Not Possible
Small > Big (✓)	Big > Small (✗)
(Source) (to) (Destination)	(Source) (to) (Destination)
e.g:- byte > int ✓	int > byte
(to)	(to)

[=> Short :-

S.o.pln("size of short is:" + Short.SIZE);
Sout("MIN VALUE :" + Short.MIN_VALUE);
Sout("MAX VALUE :" + Short.MAX_VALUE);

output Size:- 16 bits (2 Byte)

minvalue:- -32768 | maxvalue: +32767

eg:- Short data = 137; (valid)

Short data = true; // CE: incompatible types.

(*) -> NOTE: (i) Short is not at all used in java now and this is best suited only if u used old processors size 8086

(ii) The most commonly used datatype for storing whole number is "int" only and by default if we specify any literal of number type, compiler will try to keep it as "int" only, but we can keep either is short or byte also

[=> Long :-

Size: 64 bits (8 bytes)

min value:- -9223372036854775808

max value:- 9223372036854775807

-> If the data goes beyond the range of int, then to keep the data in site long datatype we need to

expliciitly suffix the data with 'L' or 'l' otherwise it would result in "compileTimeError".

e.g: long firstData = 9234567; // X (C.E) (No problem)
long firstData = 9234567L; // (valid)
long secondData = 10L; // No problem because 10 is integer
long number = 5l;

Note:- When int is not enough to hold the big values then we use long data type. When we work with large files, data would come to java program in terms of GB's.

long size = file.length();

[⇒ Integer:-
size:- 32 bits (4 bytes)
min value:- -2147483648
max value:- +2147483647

[⇒ Float:- (unprecisetable)
size:- 32 bits (4 bytes)
min value:- 1.4E45
max value:- 3.4028235E38

⇒ Double:- (unprecisetable)
size:- 64 bits (8 bytes)
min value:- 4.9E-324
max value:- 1.79769313486231570E308

② NON-primitive DataTypes:-

⇒ It can be break to Primitive data types. char=
ex:- String, Arrays etc. ⇒ String = "Swarup" ⇒ 'S' 'W' 'A' 'R' 'U' 'P'
(we break to characters)

CHART