7/11/22

(13) Method overriding and Array Introduction

↳ Hyder Abbas.

① Method overloading :- if a class has Multiple Methods having same name but different in parameters, it is known as Method overloading.

Eg :- Dev1

```
Class Calculator1
{
① => int add (int a, int b)
   {
     retun a+b;
   }
② => int add (int a, int b, int c)
   {
     return a+b+c;
   }
③ => float add (int a, float b)
   {
     return a+b; }
④ => float add ( float a, float b)
   {
     return a+b;
   }
⑤ => float add (int a, float b, float c)
   {
     return a+b+c;
   }
⑥ => double add (int a, int b, double c)
   {
     return a+b+c;
   }
⑦ => double add (double a, double b, double c)
   {
     return a+b+c;
   }
⑧ => double add (int a, double b, int c) {
     return a+x+c;
   }
```

Dev ②

```
Public Class LaunchMO {
   Public static Void main (String[] args){
      Calculator1 Calc = new Calculator1();
      int a=10, b=30, c=20;
      float m=10.5f, n=20.5f, o=30.5f;
      double x=15.5, y=25.5, z=35.f;
S.o.p ( Calc.add(a,b)); → result print on Console.
S.o.p (Calc.add(m,n)); → adding two float num.
S.o.p ( Calc.add(a,b,c)); → Adding 3 no int no.
S.o.p (Calc.add(x,y,z));
S.o.p (Calc.add(a,b,x));
   }
}
```

(1) developers has to remember all methods name in a class in order to call if methods have different names.

(2) to avoid this problem In Java class we can write multiple methods with same names but different in parameters.

(3) Method overloading refers to process of writing more than one method with same name and different parameters within same class.

(4) developer effort has reduced.

(5) 1: many called polymorphism.

(6) add - one method performing multiple activities (illusion.
↳ false polymorphism.

(7) In reality one method performing one task.

(8) 1 class → many methods
↓
Same name
↓
Same no. of parameters
↓
not same data type

(9) all add methods are active when called add method.

Calc.add(a b)

↳ All add method which are accepting two inputs

(10) Compile time polymorphism:

Compiler resolving this issue based on
(1) Parameter (no. of Parameters)
(2) Data type of Parameter
(3) Order of data type of parameter.

Calc.add(a,b,x); → (6) becomes active.
int double

↳ Add method which is accepting 3 parameters (2 integers, 1 double)

(i) double add(int a, int b, double c){
return a+b+c; ← This becomes active.
}

(ii) double add(int a, double b, int c)
{
return a+b+c;
}

Eg ①

Calc.add (int a, b, X)

int

double.

① Calls add method; All add method get active.

② Methods with 3 parameters get active

③ Methods accepting 2 integer and 1 double get active.

→ In built method in java : using Concept of Method overloading.

⇒ Inbuilt Methods are using Method overloading

Eg :- System.out.println("hello");
System.out.println(a);

→ Method overloading also called as Early binding / Compile time polymorphism.

⇒ Compiler resolve the Conflict :-

(a) no. of Parameters
(b) Data type of parameters
(c) order of data type of parameters

Eg ② Coding Snippets

① int add(int a, int b)
{
return a+b;
}

Void add(int a, int b) ← Cal.add(10,20)
{
int res = a+b;
S.o.p(res);
}

→ given Compiletime Error

→ return type has no role play, it's only method name parameter.

② Method overloading with numeric Promotion :-

float add(float a, int b)
{
return a+b;
}

"implicit type conversion
int → float

S.o.p(Calc.add(10,20))
↓ int
30

float add(float a, float b, int c)
{
return a+b+c;
}

→ System.out.println( );
↳ is one of the statement of the program
↳ it is not output.

(2)

~~float add(int a, int b) {~~

~~return a+b;~~

~~}~~

~~float add(~~

(3)

```
float add(float a, int b)
{
    return a+b;
}
float add(int c, float a)
{
    return a+c;
}
```

Calc.add(10,20);

Calc.add(10,20) → both accepts two parameters
↳ Method having capacity to accept two integer values.
↳ 2 Methods have Capacity.
↳ Compiler get confused (or) ambiguous (not clear)
↳ Compiler give Error

(4) General Method

```
Void disp() {
    S.o.p("i am a neuron");
}
Void disp(String name) {
    S.o.p(name);
}
Void disp(int age) {
    s.op(age);
}
```

d.disp();
d.disp(28);
d.disp("ashish");

→ Can we overload main method :-

We can overload main method however JVM will call such a main method which accepts String[] args as parameters.

```
Public static Void main(String[] args) {
```

(1) ⇒
```
PSVM(String[] args) {
    S.o.p("it's actual main method");
}
```

(2) ⇒
```
PSVM(int[] arss) {
    S.o.p("it accepts int arss");
}
```

(3) ⇒
```
PSVM(double[] args) {
    S.o.p("double Value");
}
```

→ JVM will search start point.