Lamda Expression

Lamda Expression

Lamda expression → functional Interface
↓
single abstract method();

@FunctionalInterface
interface IDemo { void disp(); }

"→" operator "lamda" or "Arrow"

lamda expression: unnamed (Anonymous)
2 parts: method

() → { sop("Hi"); }
method defn.

() → sop("Hi"); // single stmt.

3. left side data type is optional.
4. Right side 1 stmt {} optional & return stmt is also optional.
5. left side if 1 parameter then, () are optional.

WAP to compute length of string

```
interface CLS
{ int getLength(String str); }

public class Lamda
{ main()
{ CLS c = (str) -> {
return str.length();
}
```

OR

```
c = str -> str.length();

Syout(c.getLength("Hello"));
```

24 Lamda Expression

26 November 2022 14:06

```
//
@FunctionalInterface
interface Add
{
    void add(int a, int b);
}
@FunctionalInterface
interface Sub
{
    int sub(int num1);
}
// to write lambda exp we use lambda operator (->)
// lambda operator divided into 2 parts to writw lambda exp
// left side of lambda operator we write parameters required
//right side of lambda operator we write body or implementation
// left side for parameters datatype is optional
// write side if implementation or body has one statement then {} is optional
// left side if parameter is single then () and type of data both optional
// write side in body if its single line implementaion then return statement is
also optional
// {} is mandtaory if there are more then one statement and also if there is return
statement explictly used by developer
public class LaunchLambda {
    public static void main(String[] args)
    {
        Add add= (a, b)->{
            int res=a+b;
            System.out.println(res);
        };

        add.add(10, 20);

//      Sub sub=  num1->{
//          int res=num1-5;
//          return res;
//      };
//      Sub sub= num1 -> {
//          return num1-5;
//      };
        Sub sub= num1 -> num1-5;


    }
}
```

```
// WAP to compute length of String
@FunctionalInterface
interface CLS
{
    int getLength(String str);
}
//#1
//class LOS implements CLS
//{
//  public int getLength(String str)
//  {
//      int length=str.length();
//      return length;
//  }
//}
public class LaunchLambda2 {
    public static void main(String[] args)
    {
//      LOS l=new LOS();
//      System.out.println(l.getLength("iNeuron.ai"));

//      CLS cls=new CLS() {
//
//          public int getLength(String str)
//          {
//              return str.length();
//          }
//      };
//      System.out.println(cls.getLength("iNeuron.ai"));

        CLS cls= str -> str.length();

        System.out.println(cls.getLength("iNeuron.ai"));
    }
}
```

25 nov Method Hiding

26 November 2022 12:43

Method hiding can be defined as, "if a subclass defines a static method with the same signature as a static method in the super class, in such a case, the method in the subclass hides the one in the superclass." The mechanism is known as method hiding. *It* happens because ***static*** methods are resolved at compile time.

```
//Static method will participate in inheritance but can't be overrided,it will be treated as specialized method.

class Parent {
    public static void disp() {
        System.out.println("parent");
    }
}

class Child extends Parent {

    // public void disp() {//can't be overrided
    //     System.out.println("child");
    // }

    public static void disp(){//treated as Specialized method.
        System.out.println("child");
    }
}

public class MethodHiding {
    public static void main(String[] args) {
        Parent p = new Child();

        p.disp();//Parent

        ((Child)p).disp();//Child :: treated as specialized method
    }
}
-----------------------------------------------------------------------
interface IParent {
    default void foo() {
        System.out.println("Iparent");
    }
}
interface Child1 extends IParent {
    default void foo() {
        System.out.println("Child1");
    }
}
interface Diff {
    static void foo() {
        System.out.println("Differnt");
    }
}
public class Test implements IParent, Child1, Diff {
    public static void main(String[] args) {
        Child1 t = new Child1() {
        };
        t.foo();// Child1,overrided
        Test t1 = new Test();
        t1.foo();// Child1,as Different Static method is not get inherited
    }
}
```

from class perspective static methods gets inherited but can't be overrided.
But from interface " " " doesn't get inherited only.

downcasting

Interface perspective.