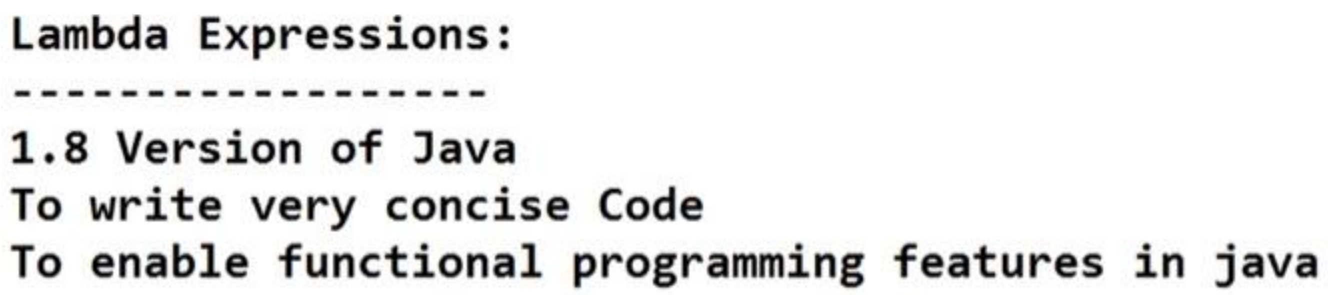
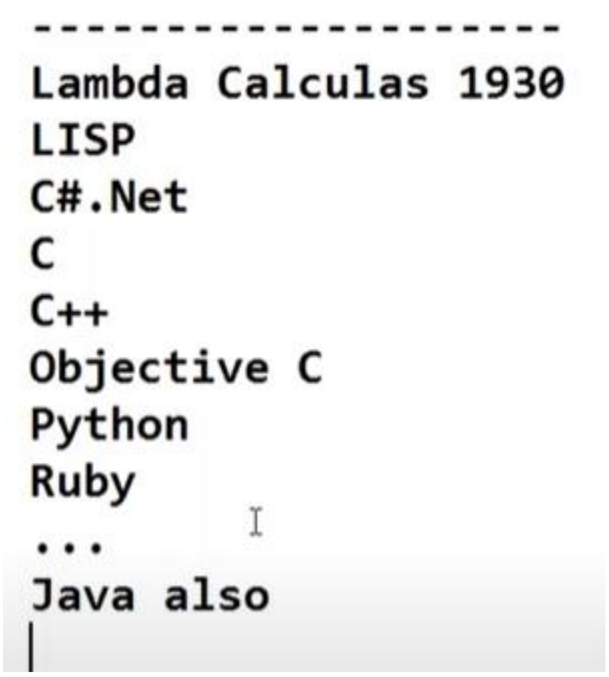
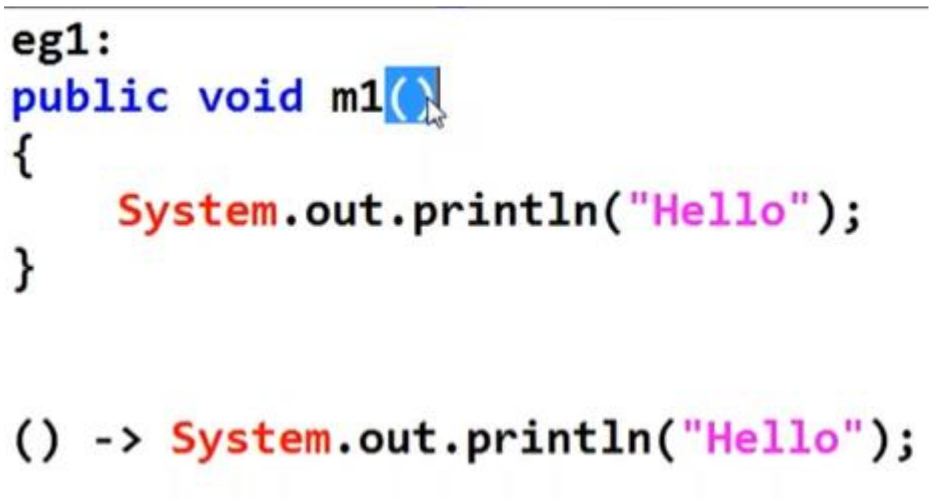
Lambda



```
Lambda Expressions:
-------------------
1.8 Version of Java
To write very concise Code
To enable functional programming features in java
```

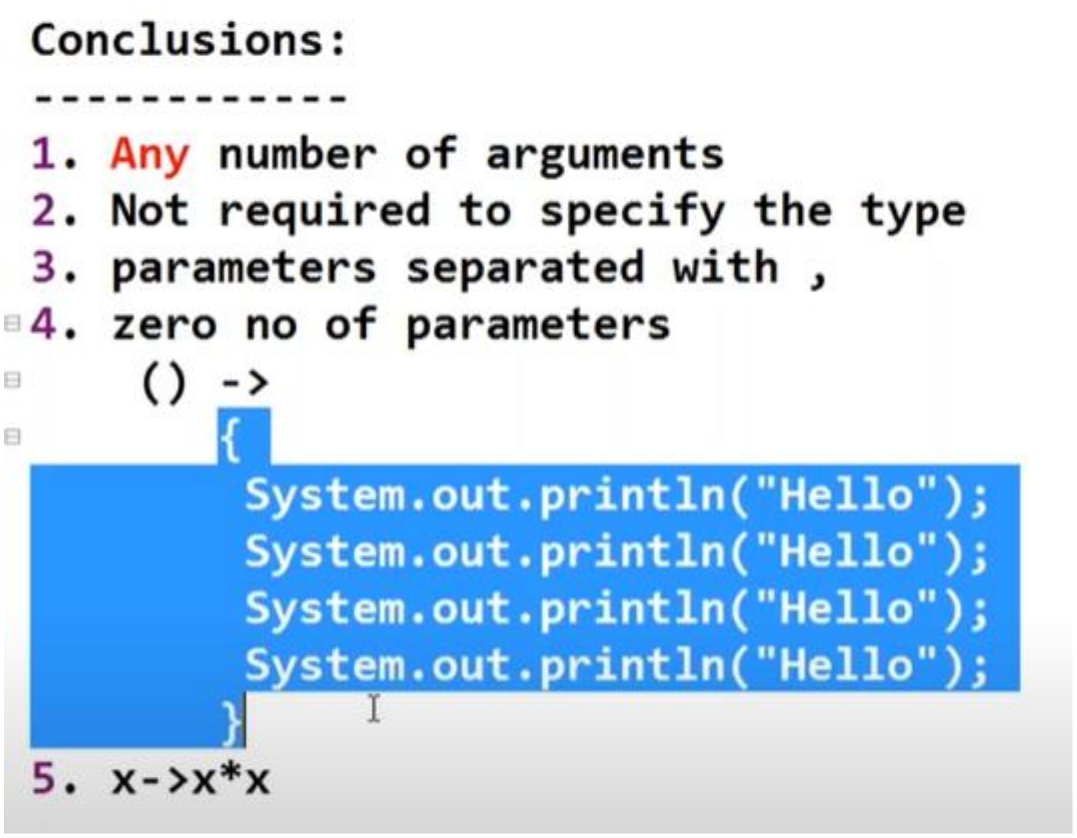


```
---------------------
Lambda Calculas 1930
LISP
C#.Net
C
C++
Objective C
Python
Ruby
...
Java also
```

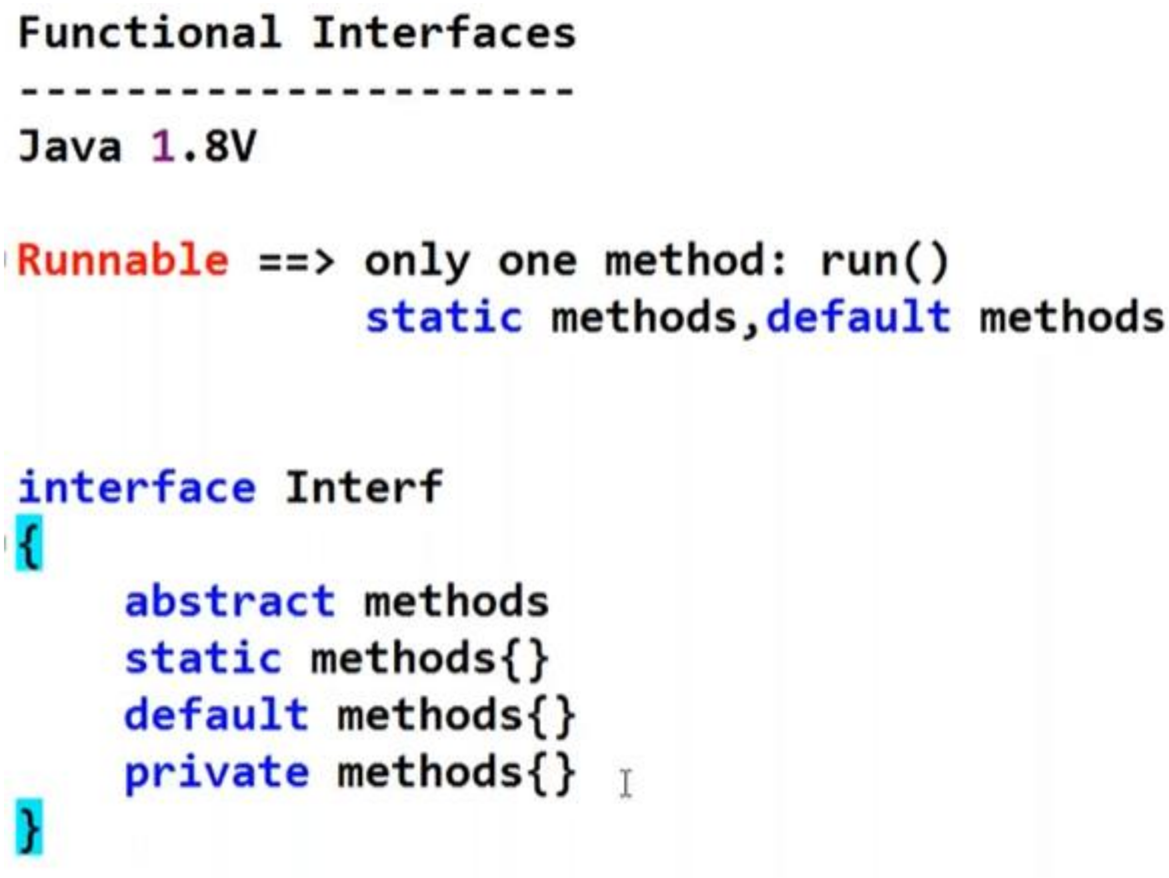


```
eg1:
public void m1()
{
    System.out.println("Hello");
}


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

```
eg2:
public void add(int a,int b)
{
    System.out.println(a+b);
}


(int a,int b)-> System.out.println(a+b);

(a,b)-> System.out.println(a+b);

    type reference

eg3:
public int squareIt(int x)
{
    return x*x;
}

(int x)->{return x*x;}
(int x)->return x*x;
(x)->return x*x;
x->return x*x;
x->x*x;
```



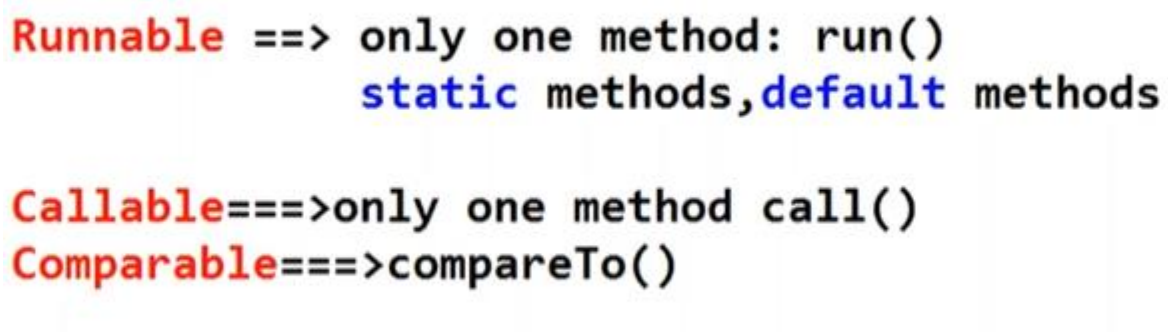
```
Conclusions:
------------
1. Any number of arguments
2. Not required to specify the type
3. parameters separated with ,
4. zero no of parameters
     () ->
        {
         System.out.println("Hello");
         System.out.println("Hello");
         System.out.println("Hello");
         System.out.println("Hello");
        }
5. x->x*x
```



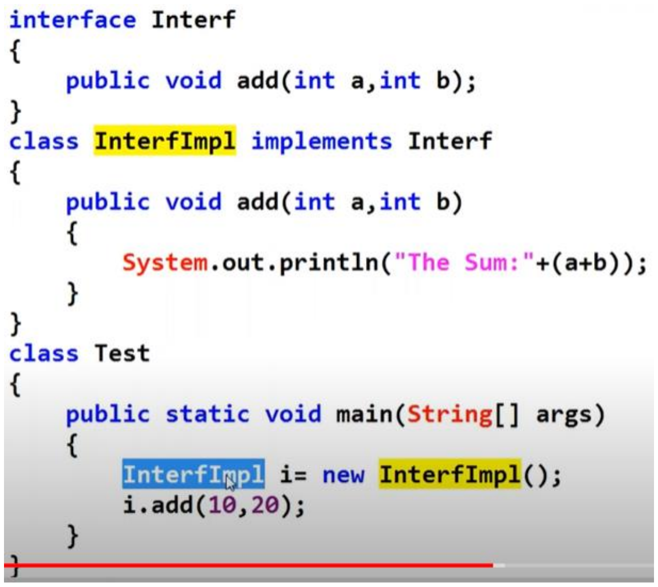
```
Functional Interfaces
---------------------
Java 1.8V

Runnable ==> only one method: run()
             static methods,default methods


interface Interf
{
    abstract methods
    static methods{}
    default methods{}
    private methods{}
}
```

Interface contain only one abstract meth call **functional** interface

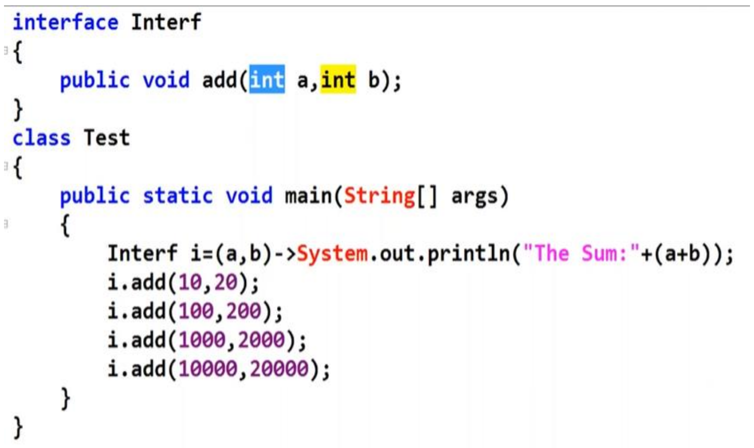
Ex of **functional**



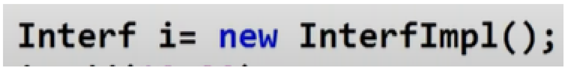
```
Runnable ==> only one method: run()
             static methods,default methods

Callable===>only one method call()
Comparable===>compareTo()
```



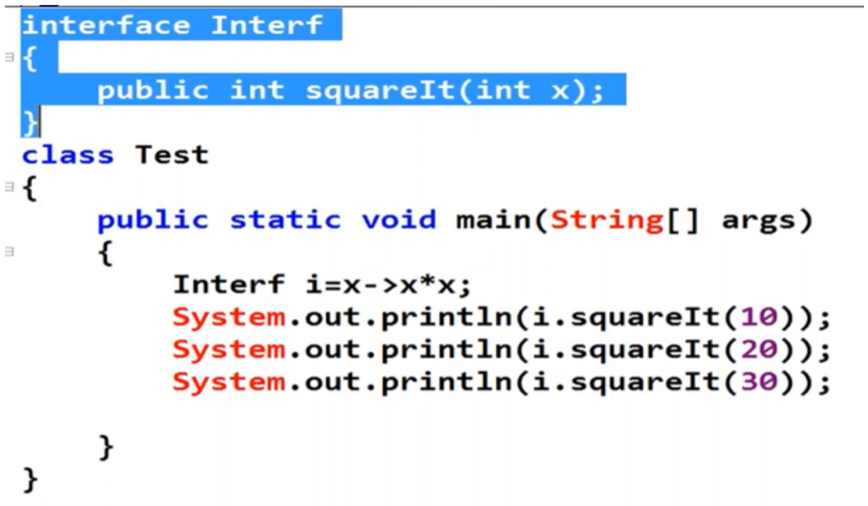
```
interface Interf
{
    public void add(int a,int b);
}
class InterfImpl implements Interf
{
    public void add(int a,int b)
    {
        System.out.println("The Sum:"+(a+b));
    }
}
class Test
{
    public static void main(String[] args)
    {
        InterfImpl i= new InterfImpl();
        i.add(10,20);
    }
}
```



```
interface Interf
{
    public void add(int a,int b);
}
class Test
{
    public static void main(String[] args)
    {
        Interf i=(a,b)->System.out.println("The Sum:"+(a+b));
        i.add(10,20);
        i.add(100,200);
        i.add(1000,2000);
        i.add(10000,20000);
    }
}
```



```
Interf i= new InterfImpl();
```



```
interface Interf
{
    public int squareIt(int x);
}
class Test
{
    public static void main(String[] args)
    {
        Interf i=x->x*x;
        System.out.println(i.squareIt(10));
        System.out.println(i.squareIt(20));
        System.out.println(i.squareIt(30));

    }
}
```

```java
class Test
{
    public static void main(String[] args)
    {
        Runnable r = ()->{ for(int i=0;i<10;i++) System.out.println("Child Threa
        Thread t = new Thread(r);
        t.start();
        for(int i =0; i<10; i++)
        {
            System.out.println("Main Thread");
        }
    }
}
```

Predicate==>Predefined Functional Interface

boolean test(T t)

```java
interface Predicate<T>
{
    public boolean test(T t);
}
```

java.util.function package

```java
interface Predicate<T>
{
    public boolean test(T t);
}
class PredicateImpl implements Predicate
{
    public boolean test(Integer i)
    {
        if(i>10)
        {
            return true;
        }
        else
        {
            return false;
        }
    }
}
```

```java
import java.util.function.*;
class Test
{
    public static void main(String[] args)
    {
        Predicate<Integer> p= i->i>10;
        System.out.println(p.test(100));
        System.out.println(p.test(5));
    }
}
```

```
Predicate Joining:
--------------

test()==>abstract method
```

$$\left.\begin{array}{l}\texttt{and()}\\ \texttt{or()}\\ \texttt{negate()}\end{array}\right\}\text{default method}$$