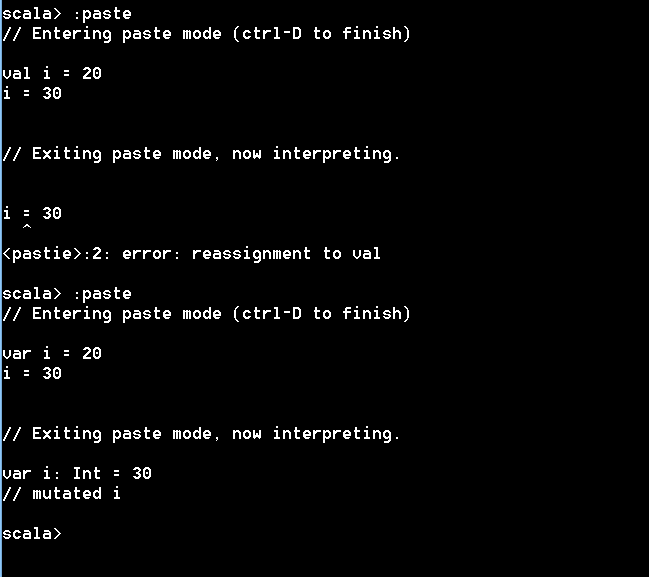


Declaration:

Val is immutable

Var is mutable



Val i = 1 + 2

Output 3:

Here everything is functional, + is also considered to be a function

**String functions:**

scala> val s = “Hello World”

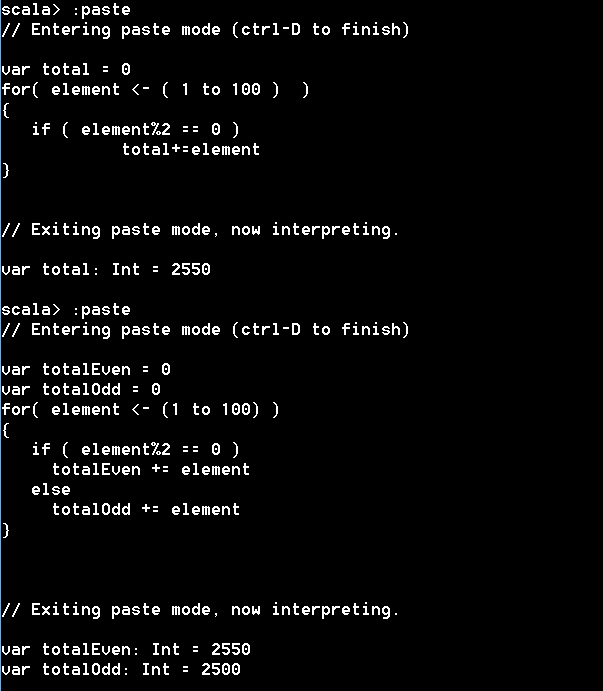
----------------------------------

( ) is optiona for using functions

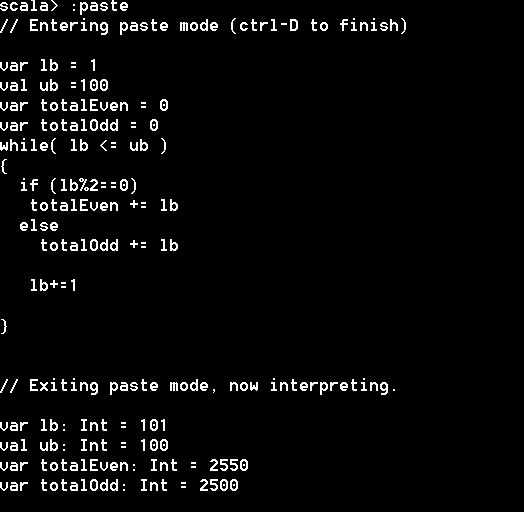
s.toUpperCase or s.toUpperCase() same.

\*print sum of even numbers upto 100

\*print sum of even and odd numbers upto 100

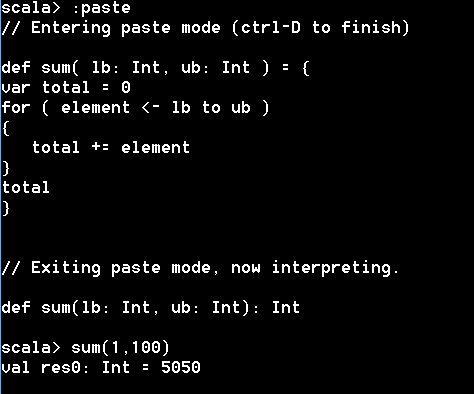


While loop:

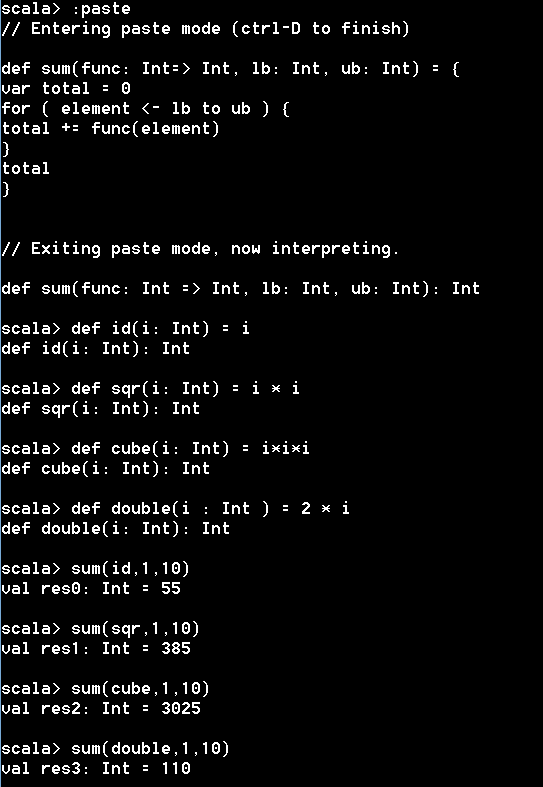




Define functions:



Calling a function in another function



**Anonymous functions**

Without giving name we can define

In the above example, we are calling sum( func,v1, v2)

Here in func, we can send defined function or anonymous function

Examples shown below:

======================

**scala> sum( i => i , 1,10 )**

**val res4: Int = 55**

**scala> sum( i => i \* i , 1,10 )**

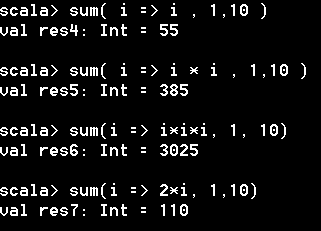
**val res5: Int = 385**

**scala> sum(i => i\*i\*i, 1, 10)**

**val res6: Int = 3025**

**scala> sum(i => 2\*i, 1,10)**

**val res7: Int = 110**



**We can write or define any logic may be complex, want to add square of number if number is even and add double of number if number is odd.**

**==================**

**scala> sum ( i => {**

**| if ( i%2 == 0 )**

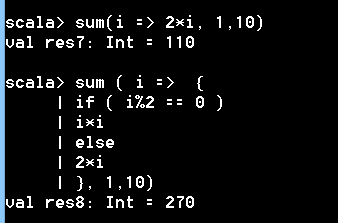
**| i\*i**

**| else**

**| 2\*i**

**| }, 1,10)**

**val res8: Int = 270**





**\*DEFINING CLASS**

**=======================================================**

**scala> class Order(orderId: Int, orderDate: String,orderCustomerId: Int, orderStatus: String){**

**| println("I am inside Order Constructor")**

**| }**

**class Order**

**scala> :javap -p Order**

**Compiled from "<console>"**

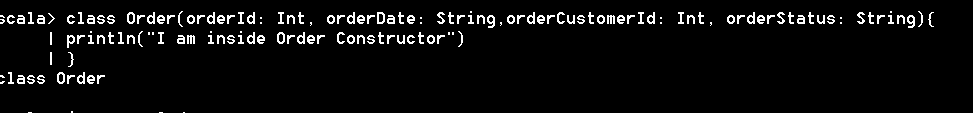
**public class $line3.$read$$iw$Order {**

**public final $line3.$read$$iw $outer;**

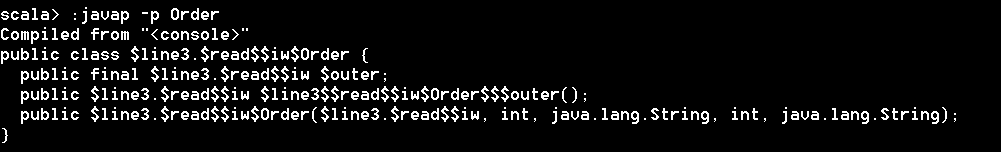
**public $line3.$read$$iw $line3$$read$$iw$Order$$$outer();**

**public $line3.$read$$iw$Order($line3.$read$$iw, int, java.lang.String, int, java.lang.String);**

**}**



**Compile**

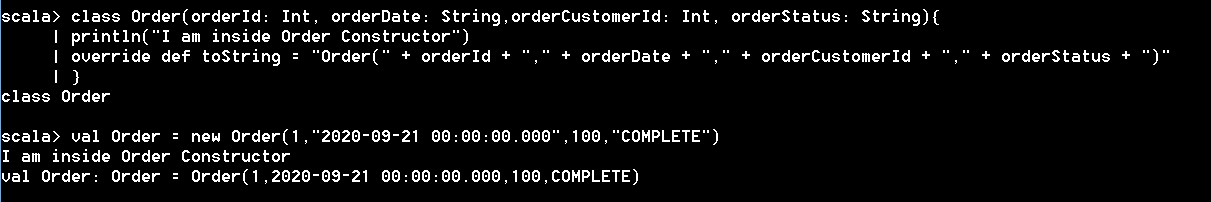


**Incase if gives error, we need to set path .**

**In Linux : export PATH=…..**

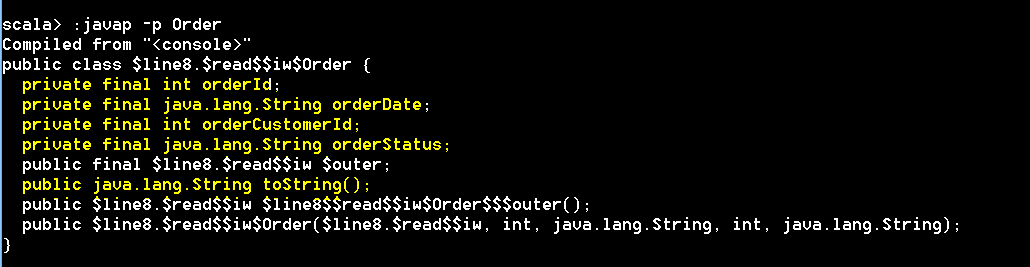


**Override toString method:**



**We can call toString directly with println**





**Private variables created, method toString defined**

**\*\*\* Private variables created, and toString method defined.**

**getter methods using val**

**====================================================**

**scala> class Order(val orderId:Int, val orderDate: String, val orderCustomerId: Int, val orderStatus: String){**

**| println(" I am inside Order Constructor")**

**| override def toString = "Order(" + orderId + "," + orderDate + "," + orderCustomerId + "," + orderStatus + ")"**

**| }**

**class Order**

**scala> :javap -p Order**

**Compiled from "<console>"**

**public class $line12.$read$$iw$Order {**

**private final int orderId;**

**private final java.lang.String orderDate;**

**private final int orderCustomerId;**

**private final java.lang.String orderStatus;**

**public final $line12.$read$$iw $outer;**

**public int orderId();**

**public java.lang.String orderDate();**

**public int orderCustomerId();**

**public java.lang.String orderStatus();**

**public java.lang.String toString();**

**public $line12.$read$$iw $line12$$read$$iw$Order$$$outer();**

**public $line12.$read$$iw$Order($line12.$read$$iw, int, java.lang.String, int, java.lang.String);**

**}**

**\* create Order object**

**scala> val Order = new Order(1,"2020-09-21 00:00:00.000",100,"COMPLETE")**

**I am inside Order Constructor**

**val Order: Order = Order(1,2020-09-21 00:00:00.000,100,COMPLETE)**

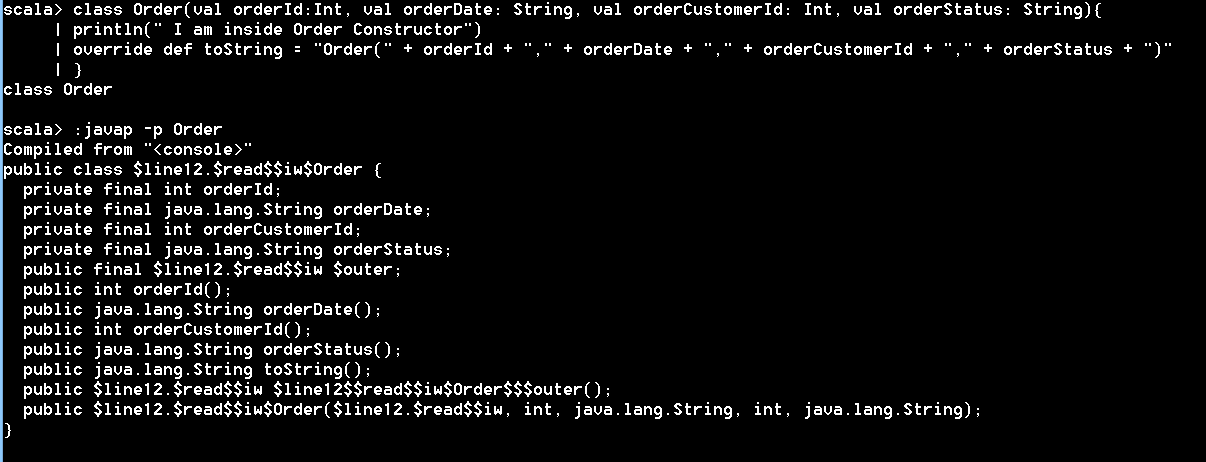
**\*access values using object.variable**

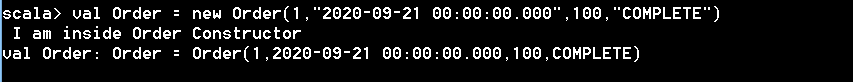
**scala> Order.orderId**

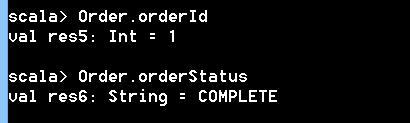
**val res5: Int = 1**

**scala> Order.orderStatus**

**val res6: String = COMPLETE**







**Actually :**

**Order.orderId 🡪 indicates we are calling Order.orderId() method which has been created by making parameters val and compiled .**

* **So getter methods are created by val**

**Use var in parameters to make the variables mutable**

**scala> :paste**

**// Entering paste mode (ctrl-D to finish)**

**class Order(var orderId:Int, var orderDate: String, var orderCustomerId: Int, var orderStatus: String){**

**println(" I am inside Order Constructor")**

**override def toString = "Order(" + orderId + "," + orderDate + "," + orderCustomerId + "," + orderStatus + ")"**

**}**

**// Exiting paste mode, now interpreting.**

**class Order**

**scala> :javap -p Order**

**Compiled from "<pastie>"**

**public class $line18.$read$$iw$Order {**

**private int orderId;**

**private java.lang.String orderDate;**

**private int orderCustomerId;**

**private java.lang.String orderStatus;**

**public final $line18.$read$$iw $outer;**

**public int orderId();**

**public void orderId\_$eq(int);**

**public java.lang.String orderDate();**

**public void orderDate\_$eq(java.lang.String);**

**public int orderCustomerId();**

**public void orderCustomerId\_$eq(int);**

**public java.lang.String orderStatus();**

**public void orderStatus\_$eq(java.lang.String);**

**public java.lang.String toString();**

**public $line18.$read$$iw $line18$$read$$iw$Order$$$outer();**

**public $line18.$read$$iw$Order($line18.$read$$iw, int, java.lang.String, int, java.lang.String);**

**}**

**scala> val Order = new Order(1,"2020-09-21 00:00:00.000",100,"COMPLETE")**

**I am inside Order Constructor**

**val Order: Order = Order(1,2020-09-21 00:00:00.000,100,COMPLETE)**

**scala> Order.orderId**

**val res7: Int = 1**

**scala> Order.orderId = 3**

**// mutated Order.orderId**

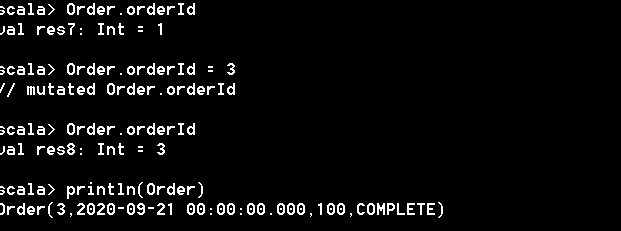
**scala> Order.orderId**

**val res8: Int = 3**

**scala> println(Order)**

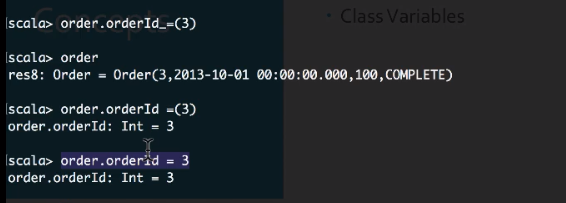
**Order(3,2020-09-21 00:00:00.000,100,COMPLETE)**





**\*\*\*\* Variables are not final as in the case of val declaration, so with var, we can change the variable values**

**We can use \_ for setting methods**





**Object 🡪 Singleton**

**scala> object HelloWorld {**

**| def main(args: Array[String]):Unit ={**

**| println("Hello World")**

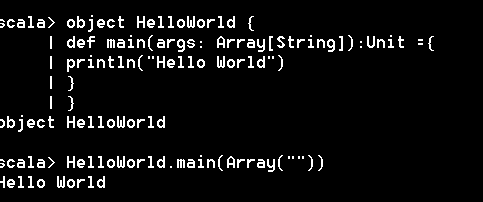
**| }**

**| }**

**object HelloWorld**

**scala> HelloWorld.main(Array(""))**

**Hello World**



**Companion object – Classname = objectname**

**scala> :paste**

**// Entering paste mode (ctrl-D to finish)**

**class Order(var orderId:Int, var orderDate: String, var orderCustomerId: Int, var orderStatus: String){**

**println(" I am inside Order Constructor")**

**override def toString = "Order(" + orderId + "," + orderDate + "," + orderCustomerId + "," + orderStatus + ")"**

**}**

**object Order {**

**def apply(orderId: Int, orderDate: String, orderCustomerId: Int, orderStatus: String) : Order = {**

**new Order(orderId, orderDate, orderCustomerId, orderStatus)**

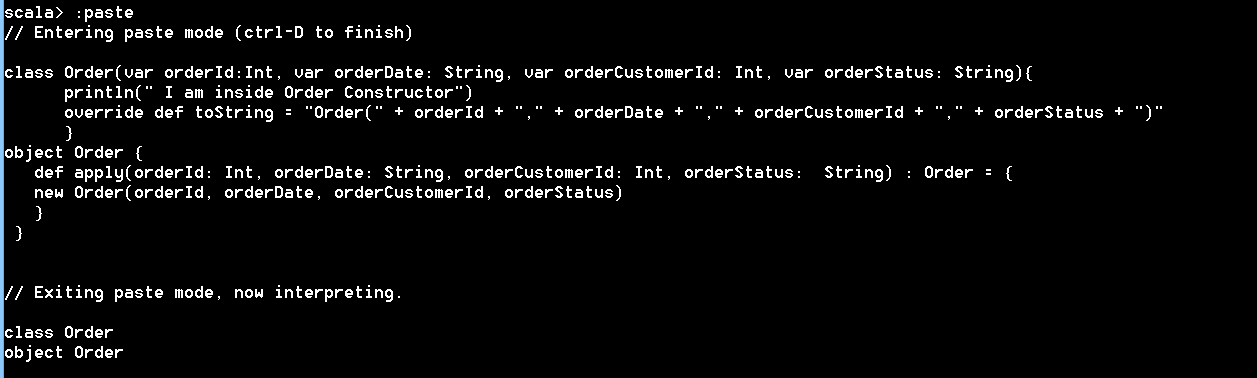
**}**

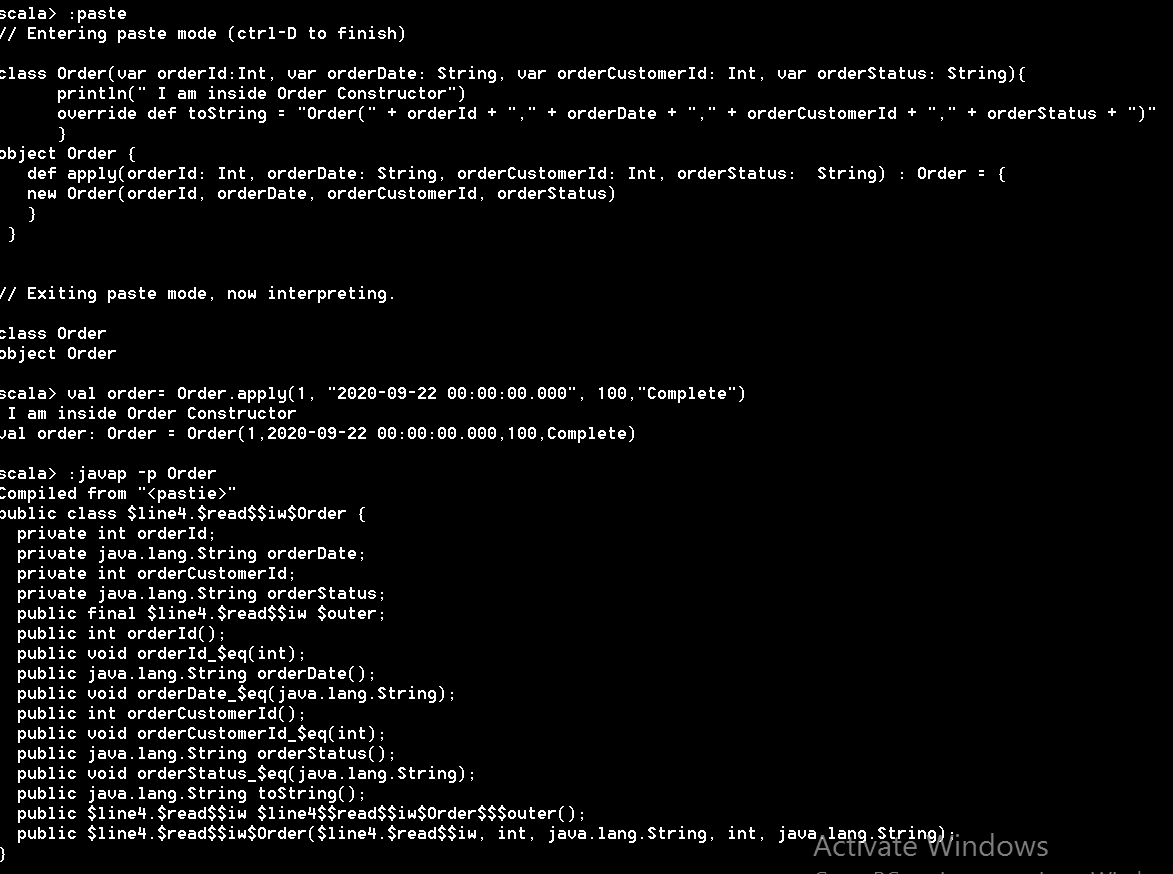
**}**

**// Exiting paste mode, now interpreting.**

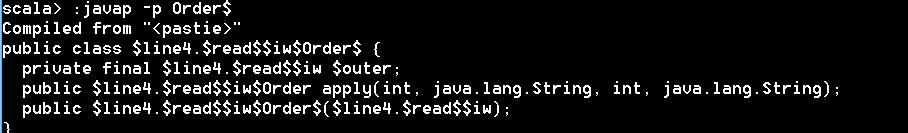
**class Order**

**object Order**

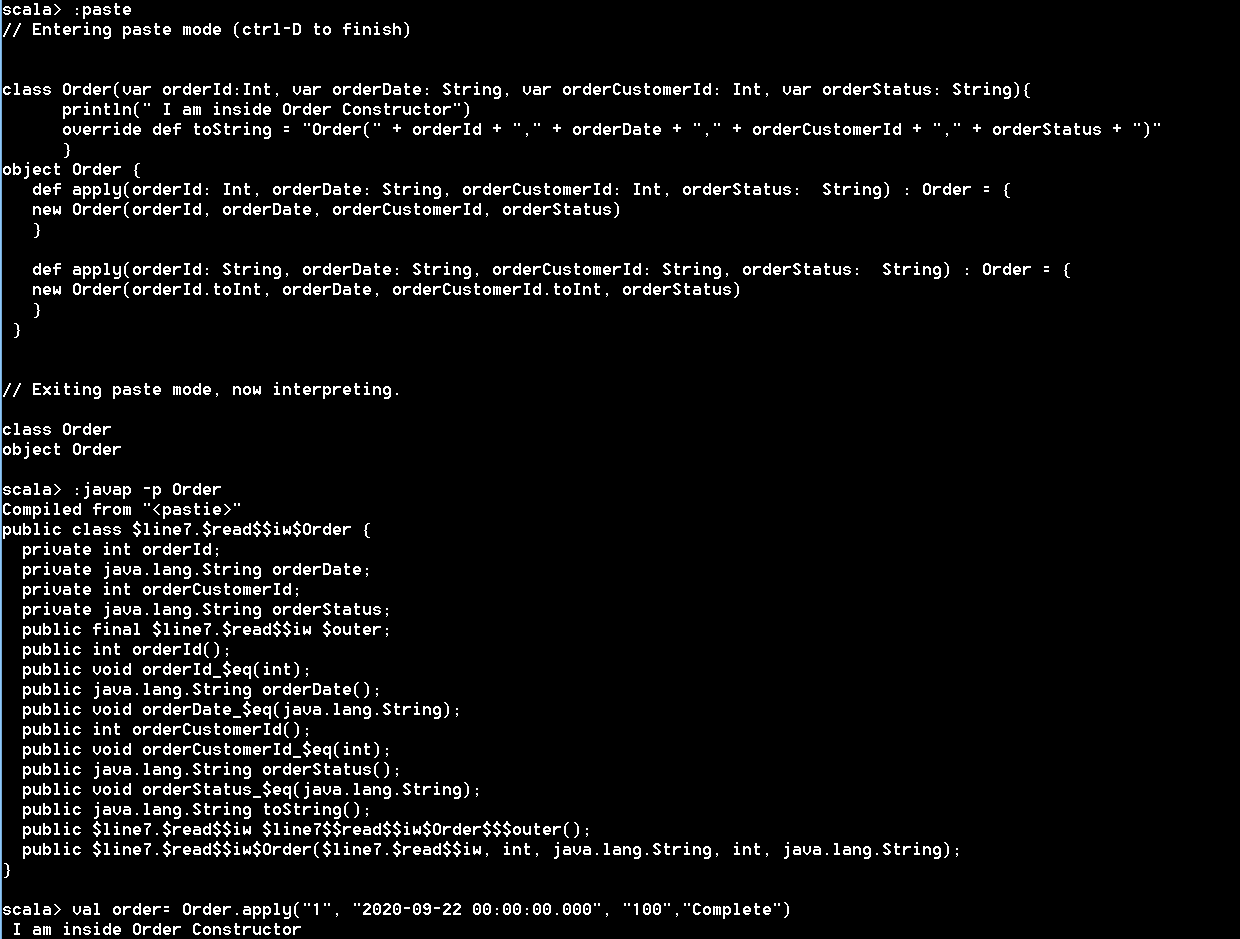


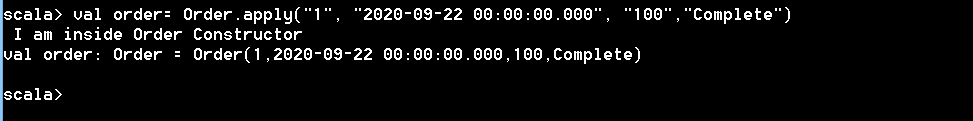


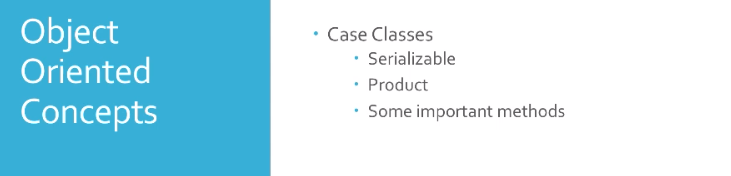
**For singleton class, apply is like static method.**



**We can write any number of apply methods like overloading**

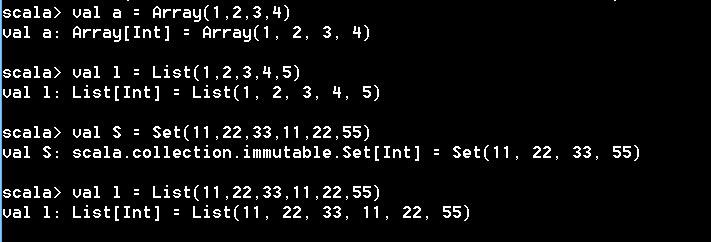






**Need to complete**

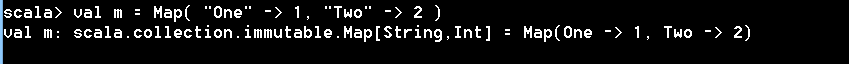




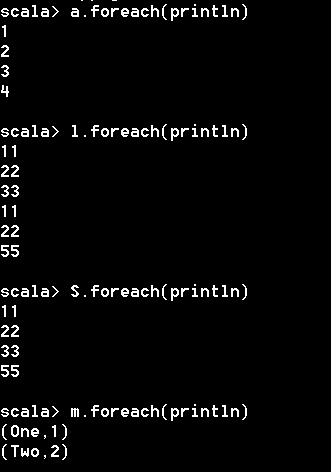
**List allows duplicates**

**Set does not allow duplicates**

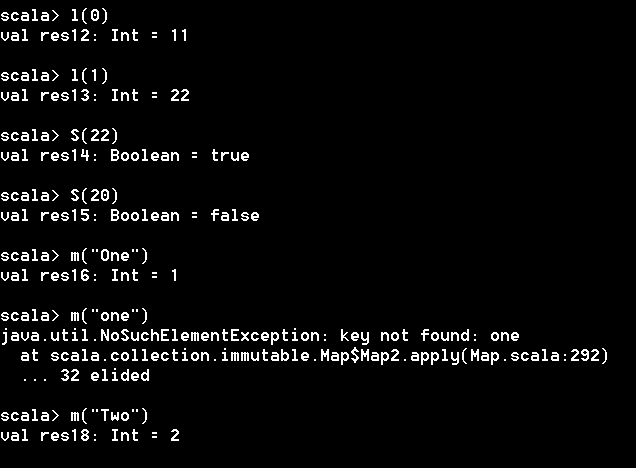
**Map:**



**Print elements:**



**Accessing Elements:**



**List operations**

**min, max, size , toSet**

