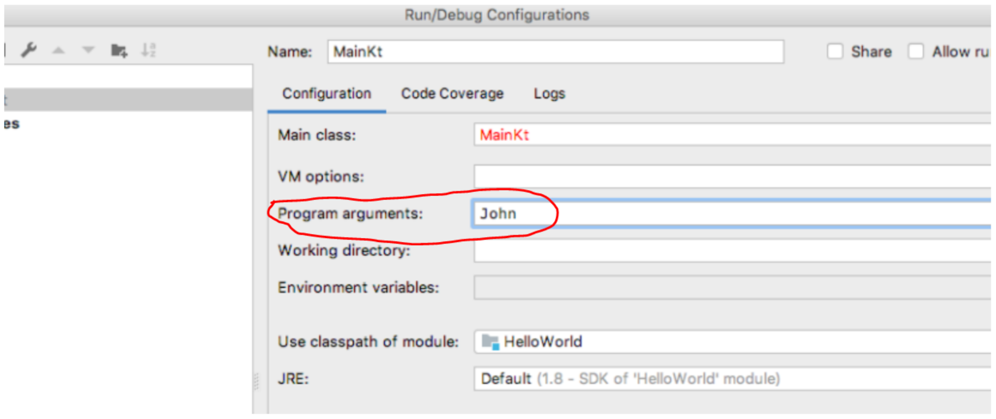
To input Command line arguments

Run -> Edit Configurations

Add the command line parameters in Program arguments as shown below and click apply. In this case, we have added a name "John".



Run/Debug Configurations

Name: MainKt

Share Allow ru

Configuration | Code Coverage | Logs

Main class: MainKt

VM options:

Program arguments: John

Working directory:

Environment variables:

Use classpath of module: HelloWorld

JRE: Default (1.8 - SDK of 'HelloWorld' module)

public static void main(String[] args)

fun main()

fun main(args: Array<String>)

Kotlin Data Types

```
// Mutable Data - var

var a: Int = 3  // Variables

var b = 6 // No need to specify the type. Kotlin infer the type from the value

b= 5.6

println(a + b)

// Declaring various number types

    val doubleNum: Double = 123.45 //64 bit number

    val floatNum: Float = 123.45f // 32 bit

    val longNum = 1237819283712L // 64 bit

// String

  //Declaring String

var name: String = "Kotlin"  // Constant

    var hero: String
    hero = "batman"

    println(hero)

    hero = "superman"

    println(hero)


//Boolean

// Declaring Boolean

    var isAwesome:Boolean = true

    println("Is " + name + " awesome? The answer is : " + isAwesome)

// Declaring Constants – Immutable – Val

val value = 3.14959265358979323


Multiline Strings

val x: String = """Kotlin

            supports

            Multiline

            Strings"""

val x: String = """|Kotlin

            |supports

            |Multiline

            |Strings""".trimMargin()

val name : String = "Kotlin"  // Constant
//  name = name + " " + "Programming"


// String Template and Calling String methods

var x = "Kotlin"
println("Hello " + x )

println("Hello $x" )

println("Main Signatures")
val x = "David Abraham"
val y = "My name is $x"
println(y)
println("My name is $x with the length ${x.length}")
```

// Null Check

```
var username : String = "Anne Mathew"
//username = null
```

```
var nullableusername : String? = "Anne Mathew"
// nullableusername = null
```

// Traditional Approach

```
val l = if (nullableusername != null) nullableusername.length else -1
println(l)
```

// Safe Call operator – Do the functionality if not null, otherwise return null

```
println(nullableusername?.length)
```

// Print default value if null – Elvis operator ?:

```
val len = nullableusername?.length ?: -1
println(len)
val nousername = nullableusername ?: "No one knows me..."
println(nousername)
```

// !! Assertion Operator

```
var nodata : String? = "Hello"

    println(nodata?.length)
 //   nodata = null;
    //Not Null Assertion - !! ( Recommended to use only the input is not null)
//   println(nodata!!.toUpperCase())
```

Array and ArrayList declaration

```
// var plist = ArrayList<Double>()
 var plist1 = Array<Double>(10){0.0} // declare array of 10
values and initialize with 0.0
 plist1[0] = 24.5
 println(plist1.size)
```

// Functions

```
fun main(args: Array<String>) {
    val count = 5
    fun displayString() {
        for (index in 1..count) {
                println("Java")
            }
        }
// Calling the function
    displayString()

}
```

// Var args

```
fun main(args: Array<String>) {
    dStrings("one", "two", "three", "four")
   }
fun dStrings(vararg strings: String){
    for (string in strings) {
        println(string)
    }
}
```

// Default arguments

```
fun main(){
    // Valid calls
   var message = bmsg("Jack",50)
    println(message)
    message = bmsg("Jack")
    println(message)
    // Pass with argument name
    message = bmsg(count = 10) // Valid
  //  message = bmsg(10) // Inalid
}
fun bmsg(name: String = "Customer", count: Int = 0): String {
    return("$name, you are customer number $count")
}
```

// Single Expression Function

```
fun main(){
 println(sum(5,6))
    println(sum1(5,6))
    println(sum2(5,6))
}
// Regular Approach
fun sum(x:Int, y:Int) : Int{
    return x + y
}
// Kotlin Approach 1
fun sum1(x:Int, y:Int) : Int = x + y

// Kotlin Approach 2
fun sum2(x:Int, y:Int) = x + y
```

Declare and Later initialize the values

```
var array = IntArray(5) // Intialize array with [0,0,0,0,0]
array[0] = 10
array.forEach { println(it) }
```

<u>Java Classes</u>	**<u>Kotlin Classes</u>**

public class Person {
 private String name;
  public Person(String name) {
        this.name = name;
 }
public String getName() {
return name;
}
public void setName() {
this.name = name;
}
}
	
	class Person(**val** name: String)

public class Person {
 private String name;
private int age;
  public Person(String name) {
        this.name = name;
 }
public Person(String name, int age) {
        this.name = name;
        this.age = age;
 }
public String getName() {
return name;
}
public void setName() {
this.name = name;
}
}
	// Without Primary Constructor
	class Person {
	  var name:String
	  var age:Int = 0
	  constructor(name:String) {
	    this.name = name
	  }
	  constructor(name:String, age:Int) {
	   this(name)
	 // this.name = name
	    this.age = age
	  }
	}

public class Person {
 private String name;
private int age;
 	
 public String getName() {
return name;
}
public void setName(String name) {
this.name = name;
}
	// Default Constructor
	class Person {
	    lateinit var name: String
	    var age: Int = 0
	    override fun toString(): String {
	        return "$name, age = $age"
	    }
	}

public String getAge() {
return age;
}
public void setAge(int age) {
this.age = age;
}
 	
}	

// Default Constructor

Person.kt

```
class Person {
    lateinit var name: String
    var age: Int = 0
    override fun toString(): String {
        return "$name, age = $age"
    }
}
```

TestPerson.kt

```
fun main(){
    var p1 = Person()
    var p2 = Person()
    p1.age = 50;
    p1.name = "Tom"
    println(p1)
    p2.age = 30;
    p2.name = "Vina"
    println(p2)
}
```

```
Class Person{
 Person( String name, int age ){
  This.name = name;
This.age = age;
}
Person(String name, int age, String Prof){
   This(name, age);
```

This.prof = prof;

}

class MyParentClass {

int myProperty

MyParentClass(int myProperty){

this. myProperty = property;

}

}

class MySubClass extends MyParentClass {

MySubClass(int myProperty) {

super(myProperty)

}

}

class MySubClass(**myProperty: Int**) : MyParentClass(**myProperty**) {

}

Replace of void in Kotlin

`Unit` is an analogue of `void` in Java

```
fun f(): Unit {
    println("Nothing return can use Unit similar like Void")
}
// If there is no return type mentioned work as void
fun f1() {
    println("No return type similar like Void")
}
```

The `Nothing` type is used as a return type of functions that don't terminate normally.

```
fun fail(message: String): Nothing
{ throw
IllegalStateException(message)
}
```

Interface

```kotlin
interface SampleIF {
    // Property Declaration
    val x:Int // Abstract declaration without value
    // If you want initialize value, use accessor
    // val y:Int = 10 // Give compilation error
    val y:Int
        get() = 20
    // Abstract method
    fun add(a:Int,b:Int):Int
    // Default method
    fun hello(){
        println("Default Hello")
    }
    // Static Method
    companion object{
        fun sayBye(){
            println("Static Bye")
        }
    }
}
class MyClass:SampleIF {
    override val x: Int = 35

    override fun add(a: Int, b: Int): Int {
     return a+b
    }

}
fun main(){
    var ob = MyClass()
    println(ob.x)
    println(ob.y)
    ob.hello()
    // Calling Static
    SampleIF.sayBye()
}
```