From Java To Kotlin

Print to Console

Java

```
System.out.print("Amit Shekhar");
System.out.println("Amit Shekhar");
```

Kotlin

```
print("Amit Shekhar")
println("Amit Shekhar")
```

Constants and Variables > Java java String name = "Amit Shekhar"; final String name = "Amit Shekhar"; > Kotlin kotlin var name = "Amit Shekhar" val name = "Amit Shekhar"

Assigning the null value

Java

```
String otherName;
otherName = null;
```

Kotlin

```
var otherName : String?
otherName = null
```

Verify if value is null > Java `java if (text != null) { int length = text.length(); }` > Kotlin `kotlin text?.let { val length = text.length } // or simply val length = text?.length`

Verify if value is NotNull OR NotEmpty

Java

```java
String sampleString = "Shekhar";
if (!sampleString.isEmpty()) {
    myTextView.setText(sampleString);
}
if(sampleString!=null && !sampleString.isEmpty()){
    myTextView.setText(sampleString);
}
```

Kotlin

```kotlin
var sampleString ="Shekhar"
if(sampleString.isNotEmpty()){  //the feature of kotlin extension function
    myTextView.text=sampleString
}
if(!sampleString.isNullOrEmpty()){
   myTextView.text=sampleString
}
```

Concatenation of strings > Java java String firstName = "Amit"; String lastName = "Shekhar"; String message = "My name is: " + firstName + " " + lastName; > Kotlin kotlin var firstName = "Amit" var lastName = "Shekhar" var message = "My name is: $firstName $lastName"

New line in string

Java

```
String text = "First Line\n" +
              "Second Line\n" +
              "Third Line";
```

Kotlin

```
val text = """
        |First Line
        |Second Line
        |Third Line
        """.trimMargin()
```

Substring

Java

```
String str = "Java to Kotlin Guide";
String substr = "";

//print java
substr = str.substring(0, 4);
System.out.println("substring = " + substr);

//print kotlin
substr = str.substring(8, 14);
System.out.println("substring = " + substr);
```

Kotlin

```
var str = "Java to Kotlin Guide"
var substr = ""

//print java
substr = str.substring(0..3) //
println("substring $substr")

//print kotlin
substr = str.substring(8..13)
println("substring $substr")
```

Ternary Operations

Java

```
String text = x > 5 ? "x > 5" : "x <= 5";

String message = null;
log(message != null ? message : "");
```

Kotlin

```
val text = if (x > 5) "x > 5" else "x <= 5"

val message: String? = null
log(message ?: "")
```

Bit- wise Oper- ators > Java

```
java
final
int
andResult
= a
& b;
final
int
orResult
= a
| b;
final
int
xorResult
= a
^ b;
final
int
rightShift
= a
>>
2;
final
int
leftShift
= a
<<
2;
final
int
unsignedRightShift
= a
>>>
2;
```

>
Kotlin

```
kotlin
val
andResult
= a
and
b
val
orResult
= a
or b
val
xorResult
= a
xor
b
val
rightShift
= a
shr
2
val
leftShift
= a
shl
2
val
unsignedRightShift
= a
ushr
2
```

Check the type and casting

Java

```java
if (object instanceof Car) {
    Car car = (Car) object;
}
```

Kotlin

```kotlin
if (object is Car) {
var car = object as Car
}

// if object is null
var car = object as? Car // var car = object as Car?
```

Check the type and casting (implicit)
> Java
`java if (object instanceof Car) { Car car = (Car) object; }`
> Kotlin
“‘kotlin if (object is Car) { var car = object // smart casting }

// if object is null if (object is Car?) { var car = object // smart casting, car will be null } ‘‘‘

Multiple conditions

Java

```
if (score >= 0 && score <= 300) { }
```

Kotlin

```
if (score in 0..300) { }
```

Multiple Conditions (Switch case) > Java

```
java
int
score
= //
some
score;
String
grade;
switch
(score)
{
case
10:
case
9:
grade
=
"Excellent";
break;
case
8:
case
7:
case
6:
grade
=
"Good";
break;
case
5:
case
4:
grade
=
"OK";
break;
case
3:
case
2:
case
1:
grade
=
"Fail";
break;
default:
grade
=
"Fail";
}
```

>
Kotlin

```
kotlin
var
score
= //
some
score
var
grade
=
when
(score)
{ 9,
10
->
"Excellent"
in
6..8
->
"Good"
4, 5
->
"OK"
else
->
"Fail"
}
```

For-loops

Java

```
for (int i = 1; i <= 10 ; i++) { }

for (int i = 1; i < 10 ; i++) { }

for (int i = 10; i >= 0 ; i--) { }

for (int i = 1; i <= 10 ; i+=2) { }

for (int i = 10; i >= 0 ; i-=2) { }

for (String item : collection) { }

for (Map.Entry<String, String> entry: map.entrySet()) { }
```

Kotlin

```
for (i in 1..10) { }

for (i in 1 until 10) { }

for (i in 10 downTo 0) { }

for (i in 1..10 step 2) { }

for (i in 10 downTo 0 step 2) { }

for (item in collection) { }
for ((key, value) in map) { }
```

—

Collections > Java “‘java final List listOfNumber = Arrays.asList(1, 2, 3, 4); final Map<Integer, String> keyValue = new HashMap<Integer, String>(); map.put(1, “Amit”); map.put(2, “Ali”); map.put(3, “Mindorks”); // Java 9 final List listOfNumber = List.of(1, 2, 3, 4); final Map<Integer, String> keyValue = Map.of(1, “Amit”, 2, “Ali”, 3, “Mindorks”); “‘

>
Kotlin
kotlin
val
listOfNumber
=
listOf(1,
2,
3,
4)
val
keyValue
=
mapOf(1
to
"Amit",
2 to
"Ali",
3 to
"Mindorks")

for each

Java

```java
// Java 7 and below
for (Car car : cars) {
  System.out.println(car.speed);
}

// Java 8+
cars.forEach(car -> System.out.println(car.speed));

// Java 7 and below
for (Car car : cars) {
  if (car.speed > 100) {
    System.out.println(car.speed);
  }
}

// Java 8+
cars.stream().filter(car -> car.speed > 100).forEach(car -> System.out.println(car.speed));
cars.parallelStream().filter(car -> car.speed > 100).forEach(car -> System.out.println(car.speed));
```

Kotlin

```kotlin
cars.forEach {
    println(it.speed)
}

cars.filter { it.speed > 100 }
      .forEach { println(it.speed)}

// kotlin 1.1+
cars.stream().filter { it.speed > 100 }.forEach { println(it.speed)}
cars.parallelStream().filter { it.speed > 100 }.forEach { println(it.speed)}
```

##
Split-
ting
ar-
rays
>
java
java
String[]
splits
=
"param=car".split("=");
String
param
=
splits[0];
String
value
=
splits[1];
>
kotlin
kotlin
val
(param,
value)
=
"param=car".split("=")

Defining methods

Java

```java
void doSomething() {
   // logic here
}
```

Kotlin

```kotlin
fun doSomething() {
   // logic here
}
```

Default values for method parameters

Java

```java
double calculateCost(int quantity, double pricePerItem) {
    return pricePerItem * quantity;
}

double calculateCost(int quantity) {
    // default price is 20.5
    return 20.5 * quantity;
}
```

Kotlin

```kotlin
fun calculateCost(quantity: Int, pricePerItem: Double = 20.5) = quantity * pricePerItem
```

```
calculateCost(10, 25.0) // 250
calculateCost(10) // 205
```

Variable number of arguments > Java `java void doSomething(int... numbers) { // logic here }` > Kotlin `kotlin fun doSomething(vararg numbers: Int) { // logic here }`

Defining methods with return

Java

```
int getScore() {
   // logic here
   return score;
}
```

Kotlin

```
fun getScore(): Int {
   // logic here
   return score
}

// as a single-expression function

fun getScore(): Int = score

// even simpler (type will be determined automatically)

fun getScore() = score // return-type is Int
```

Returning result of an operation > Java `java int getScore(int value) { // logic here return 2 * value; }` > Kotlin “‘kotlin fun getScore(value: Int): Int { // logic here return 2 * value } // as a single-expression function fun getScore(value: Int): Int = 2 * value

// even simpler (type will be determined automatically) fun getScore(value: Int) = 2 * value // return-type is int ```

Constructors

Java

```
public class Utils {

    private Utils() {
      // This utility class is not publicly instantiable
    }

    public static int getScore(int value) {
        return 2 * value;
    }

}
```

Kotlin

```
class Utils private constructor() {

    companion object {

        fun getScore(value: Int): Int {
            return 2 * value
        }

    }
}

// another way

object Utils {

    fun getScore(value: Int): Int {
        return 2 * value
    }
```

```
}
```

Getters and Setters > Java “‘java public class Developer { private String name; private int age; public Developer(String name, int age) { this.name = name; this.age = age; } public String getName() { return name; } public void setName(String name) { this.name = name; }

public
int
getAge()
{ re-
turn
age; }
public
void
se-
tAge(int
age) {
this.age
=
age; }
@Override
pub-
lic
boolean
equals(Object
o) {
if
(this
==
o) re-
turn
true;
if (o
==
null ||
get-
Class()
!=
o.getClass())
re-
turn
false;
Developer
devel-
oper
=
(De-
vel-
oper)
o;

```
if
(age
!=
devel-
oper.age)
re-
turn
false;
re-
turn
name
!=
null ?
name.equals(developer.name)
:
devel-
oper.name
==
null;
}
@Override
pub-
lic int
hash-
Code()
{ int
result
=
name
!=
null ?
name.hashCode()
: 0;
result
= 31
*
result
+ age;
re-
turn
re-
sult;
}
```

@Override pub- lic String toString() { re- turn “De- vel- oper{” + “name=‘" + name +”’ +”, age=" + age + ‘}’; } } “‘ > Kotlin “‘kotlin data class De- vel- oper(var name: String, var age: Int) “‘

Cloning or copying

Java

```
public class Developer implements Cloneable {

    private String name;
    private int age;

    public Developer(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    protected Object clone() throws CloneNotSupportedException {
        return (Developer)super.clone();
    }
}

// cloning or copying
Developer dev = new Developer("Mindorks", 30);
```

```java
try {
    Developer dev2 = (Developer) dev.clone();
} catch (CloneNotSupportedException e) {
    // handle exception
}
```

Kotlin

```kotlin
data class Developer(var name: String, var age: Int)

// cloning or copying
val dev = Developer("Mindorks", 30)
val dev2 = dev.copy()
// in case you only want to copy selected properties
val dev2 = dev.copy(age = 25)
```

Class methods > Java “‘java public class Utils { private Utils() { // This utility class is not publicly instantiable } public static int triple(int value) { return 3 * value; } } int result = Utils.triple(3); “‘

Generics

Java

```java
// Example #1
interface SomeInterface<T> {
    void doSomething(T data);
}

class SomeClass implements SomeInterface<String> {
    @Override
    public void doSomething(String data) {
        // some logic
    }
}

// Example #2
interface SomeInterface<T extends Collection<?>> {
    void doSomething(T data);
}

class SomeClass implements SomeInterface<List<String>> {

    @Override
    public void doSomething(List<String> data) {
        // some logic
    }
}
```

```kotlin
interface SomeInterface<T> {
    fun doSomething(data: T)
}

class SomeClass: SomeInterface<String> {
    override fun doSomething(data: String) {
        // some logic
    }
}

interface SomeInterface<T: Collection<*>> {
    fun doSomething(data: T)
}

class SomeClass: SomeInterface<List<String>> {
    override fun doSomething(data: List<String>) {
        // some logic
    }
}
```

Kotlin

```kotlin
fun Int.triple(): Int {
  return this * 3
}

var result = 3.triple()
```

Defining uninitialized objects > Java java Person person; > Kotlin kotlin internal lateinit var person: Person

---

enum

Java

```java
public enum Direction {
        NORTH(1),
        SOUTH(2),
        WEST(3),
        EAST(4);

        int direction;

        Direction(int direction) {
            this.direction = direction;
        }

        public int getDirection() {
            return direction;
        }
    }
```

Kotlin

```kotlin
enum class Direction(val direction: Int) {
    NORTH(1),
    SOUTH(2),
    WEST(3),
    EAST(4);
}
```

Sorting List

Java

```
List<Profile> profiles = loadProfiles(context);
Collections.sort(profiles, new Comparator<Profile>() {
    @Override
    public int compare(Profile profile1, Profile profile2) {
        if (profile1.getAge() > profile2.getAge()) return 1;
        if (profile1.getAge() < profile2.getAge()) return -1;
        return 0;
    }
});
```

Kotlin

```
val profile = loadProfiles(context)
profile.sortedWith(Comparator({ profile1, profile2 ->
    if (profile1.age > profile2.age) return@Comparator 1
    if (profile1.age < profile2.age) return@Comparator -1
    return@Comparator 0
}))
```

Anonymous Class

Java

```
 AsyncTask<Void, Void, Profile> task = new AsyncTask<Void, Void, Profile>() {
    @Override
    protected Profile doInBackground(Void... voids) {
        // fetch profile from API or DB
        return null;
    }

    @Override
    protected void onPreExecute() {
        super.onPreExecute();
        // do something
    }
};
```

Kotlin

```
val task = object : AsyncTask<Void, Void, Profile>() {
    override fun doInBackground(vararg voids: Void): Profile? {
        // fetch profile from API or DB
        return null
    }

    override fun onPreExecute() {
        super.onPreExecute()
        // do something
    }
}
```

Initialization block
> Java
“‘java public class User { { //Initialization block System.out.println(“Init block”); } } “‘
> Kotlin
`kotlin class User { init { // Initialization block println("Init block") } }`

Important things to know in Kotlin

- What is the equivalent of Java static methods in Kotlin?
- What is the difference between “const” and “val”?
- Learn Kotlin - lateinit vs lazy
- Learn Kotlin - apply vs with
- Learn Kotlin - Data Class
- Learn Kotlin - Destructuring Declarations
- Learn Kotlin - Extension Functions
- Learn Kotlin - Sealed Classes
- Understanding Higher-Order Functions and Lambdas in Kotlin
- Understanding inline, noinline, and crossinline in Kotlin
- Mastering Kotlin Coroutines In Android - Step By Step Guide
- Using Scoped Functions in Kotlin - let, run, with, also, apply
- What are Reified Types in Kotlin?

Found this project useful :heart:

- Support by clicking the :star: button on the upper right of this page. :v:

Check out Mindorks awesome open source projects here

License

Contributing to From Java To Kotlin

Just make a pull request. You are in!