Kotlin Generic

Eko Kurniawan Khannedy

License

- Dokumen ini boleh Anda gunakan atau ubah untuk keperluan non komersial
- Tapi Anda wajib mencantumkan sumber dan pemilik dokumen ini
- Untuk keperluan komersial, silahkan hubungi pemilik dokumen ini

Eko Kurniawan Khannedy

- Technical architect at one of the biggest ecommerce company in Indonesia
- 10+ years experiences
- youtube.com/c/ProgrammerZamanNow

Pengenalan Generic

Sebelum Belajar Materi Ini

- Kotlin Dasar
- Kotlin Object Oriented Programming
- https://www.udemy.com/course/pemrograman-kotlin-pemula-sampai-mahir/?referralCode=98BE2E779EB8A0BEC230

Pengenalan Generic

- Generic adalah kemampuan menambahkan parameter type saat membuat class atau function
- Berbeda dengan parameter type yang biasa kita gunakan di class di function, generic memungkinkan kita bisa mengubah-ubah bentuk type sesuai dengan yang kita mau.

Manfaat Generic

- Pengecekan ketika proses kompilasi
- Tidak perlu manual menggunakan pengecekan tipe data dan konversi tipe data
- Memudahkan programmer membuat kode program yang generic sehingga bisa digunakan oleh berbagai tipe data

Kode Program : Bukan Generic

```
4   class Data(val data: Any)
5
6   fun main() {
7       val dataString = Data("Eko")
8       val valueString: String = dataString.data as String
9
10      val dataInt = Data(10)
11      val valueInt: Int = dataInt.data as Int
12  }
13
```

Kode Program : Generic

```
4   class Data<T>(val data: T)
5
6   fun main() {
7       val dataString = Data<String>("Eko")
8       val valueString: String = dataString.data
9
10      val dataInt = Data<Int>(10)
11      val valueInt: Int = dataInt.data
12  }
13
```

Generic Class

Generic Type

- Generic type adalah class atau interface yang memiliki parameter type
- Tidak ada ketentuan dalam pembuatan generic parameter type, namun biasanya kebanyakan orang menggunakan 1 karakter sebagai generic parameter type
- Nama generic parameter type yang biasa digunakan adalah :
    - E - Element (biasa digunakan di collection atau struktur data)
    - K - Key
    - N - Number
    - T - Type
    - V - Value
    - S,U,V etc. - 2nd, 3rd, 4th types

Kode Program : Generic Type (1)

```
2
3   class MyData<T>(val firstData: T) {
4
5       fun printData() {
6           return println("Data is $firstData")
7       }
8
9       fun getData(): T {
10          return firstData;
11      }
12
13  }
```

Kode Program : Generic Type (2)

```
4
5  fun main() {
6      val data1: MyData<String> = MyData<String>("Eko")
7      val data2: MyData<String> = MyData("Eko")
8      val data3 = MyData("Eko")
9
10     data1.printData()
11     data2.printData()
12     data3.printData()
13 }
14
```

Multiple Parameter Type

- Parameter type di Generic type boleh lebih dari satu
- Namun harus menggunakan nama type berbeda
- Ini sangat berguna ketika kita ingin membuat generic parameter type yang banyak

Kode Program : Multiple Parameter Type (1)

```
3   class MyData<T, U>(val firstData: T, val secondData: U) {
4
5       fun printData() {
6           return println("Data is $firstData $secondData")
7       }
8
9       fun getSecond(): U {
10          return secondData
11      }
12
13      fun getData(): T {
```

Kode Program : Multiple Parameter Type (2)

```
4
5   fun main() {
6       val data1: MyData<String, Int> = MyData<String, Int>("Eko", 30)
7       val data2: MyData<String, Int> = MyData("Eko", 30)
8       val data3 = MyData("Eko", 30)
9
10      data1.printData()
11      data2.printData()
12      data3.printData()
13  }
14
```

Generic Function

Generic Function

- Generic parameter type tidak hanya bisa digunakan pada class atau interface
- Kita juga bisa menggunakan generic parameter type di function
- Generic parameter type yang kita deklarasikan di function, hanya bisa diakses di function tersebut, tidak bisa digunakan di luar function
- Ini cocok jika kita ingin membuat generic function, tanpa harus mengubah deklarasi class

Kode Program : Generic Function (1)

```
2
3   class Function(val name: String) {
4
5       fun <T> sayHello(param: T) {
6           println("Hello $param, My Name is $name")
7       }
8
9   }
```

Kode Program : Generic Function (2)

```
4
5   fun main() {
6       val function = Function("Eko")
7
8       function.sayHello<String>("Budi")
9       function.sayHello("Budi")
10
11      function.sayHello<Int>(10)
12      function.sayHello(10)
13  }
```

Invariant

Invariant

- Secara default, saat kita membuat generic parameter type, sifat parameter tersebut adalah invariant
- Invariant artinya tidak boleh di subtitusi dengan subtype (child) atau supertype (parent)
- Artinya saat kita membuat object Contoh<String>, maka tidak sama dengan Contoh<Any>, begitupun sebaliknya, saat membuat object Contoh<Any>, maka tidak sama dengan Contoh<String>

Kode Program : Invariant

```
2
3   class Invariant<T>(val data: T)
4
5   fun main() {
6       val data1: Invariant<String> = Invariant("Eko")
7       val data2: Invariant<Any> = data1 // error
8   }
9
```

Covariant

Covariant

- Covariant artinya kita bisa melakukan subtitusi subtype (child) dengan supertype (parent)
- Tidak semua jenis class generic yang mendukung covariant, hanya class generic yang menggunakan generic parameter type sebagai return type function
- Artinya saat kita membuat object Contoh<String>, maka bisa disubtitusi menjadi Contoh<Any>
- Untuk memberitahu bahwa generic parameter type tersebut adalah covariant, kita perlu menggunakan kata kunci out

Kode Program : Covariant

```
3   class Covariant<out T>(val data: T) {
4       fun data(): T {
5           return data
6       }
7   }
8
9   fun main() {
10      val data1: Covariant<String> = Covariant("Eko")
11      val data2: Covariant<Any> = data1
12
13      println(data2.data())
```

Contravariant

Contravariant

- Contravariant artinya kita bisa melakukan subtitusi supertype (parent) dengan subtype (child)
- Tidak semua jenis class generic yang mendukung contravariant, hanya class generic yang menggunakan generic parameter type sebagai parameter function
- Artinya saat kita membuat object Contoh<Any>, maka bisa disubtitusi menjadi Contoh<String>
- Untuk memberitahu bahwa generic parameter type tersebut adalah covariant, kita perlu menggunakan kata kunci in

Kode Program : Contravariant

```
3   class Contravariant<in T> {
4       fun sayHello(name: T) {
5           return println("Hello $name")
6       }
7   }
8
9   fun main() {
10      val data1: Contravariant<Any> = Contravariant()
11      val data2: Contravariant<String> = data1
12
13      data2.sayHello("Eko")
```

Generic Constraints

Generic Constraint

- Kadang kita ingin membatasi data yang boleh digunakan di generic parameter type
- Kita bisa menambahkan constraint di generic parameter type dengan menyebutkan tipe yang diperbolehkan
- Secara otomatis, type data yang bisa digunakan adalah type yang sudah kita sebutkan, atau class-class turunannya
- Secara default, constraint type untuk generic parameter type adalah Any, sehingga semua tipe data bisa digunakan

Kode Program : Generic Constraint

```
3   open class Employee
4
5   class Manager : Employee()
6
7   class VicePresident : Employee()
8
9   class Company<T : Employee>(val employee: T)
10
11  fun main() {
12      val data1 = Company(Manager())
13      val data2 = Company(VicePresident())
```

Where Keyword

- Kadang kita ingin membatasi tipe data dengan beberapa jenis tipe data di generic parameter type
- Secara default, hanya satu tipe data yang bisa digunakan untuk membatasi generic parameter type
- Jika kita ingin menggunakan lebih dari satu tipe data, kita bisa menggunakan kata kunci where

Kode Program : Where Keyword (1)

```
3  interface CanSayHello {
4      fun sayHello(name: String)
5  }
6
7  open class Employee
8
9  class Manager : Employee()
10
11 class VicePresident : Employee(), CanSayHello {
12     override fun sayHello(name: String) {
13         println("Hello $name")
```

Kode Program : Where Keyword (2)

```
16
17  class Company<T>(val employee: T) where T : Employee, T : CanSayHello
18
19  fun main() {
20      val data1 = Company(Manager()) // error
21      val data2 = Company(VicePresident())
22      val data3 = Company("String") // error
23  }
```

Type Projection

Type Projection

- Kadang agak sulit untuk membuat class generic type yang harus covariant atau contravariant, misal karena memang di class generic tersebut terdapat input dan output generic parameter type
- Namun jika kita membuat function untuk memanipulasi data invariant sangat lah sulit, karena generic parameter type nya harus selalu sama
- Kita bisa melakukan type projection, yaitu menambahkan informasi covariant atau contravariant di parameter function, ini memaksa isi function untuk melakukan pengecekan
- Jika covariant, kita tidak boleh mengubah data generic di object
- Jika contravariant, kita tidak boleh ngambil data generic object

Kode Program : Type Projection

```
3   class Container<T>(var data: T)
4
5   fun copy(from: Container<out Any>, to: Container<Any>) {
6       to.data = from.data
7   }
8
9   fun main() {
10      val data1 = Container("Data 1")
11      val data2: Container<Any> = Container("Data 2")
12      copy(data1, data2)
13
```

Star Projection

Star Projection

- Kadang ada kasus kita tidak peduli dengan generic parameter type pada object
- Misal kita hanya ingin mengambil panjang data Array<T>, dan kita tidak peduli dengan isi data T nya
- Jika kita mengalami kasus seperti ini, kita bisa menggunakan Star Projection
- Star projection bisa dibuat dengan mengganti generic parameter type dengan karakter * (star, bintang)

Kode Program : Star Projection

```
fun displayLength(array: Array<*>) {
    println("Length Array is ${array.size}")
}

fun main() {
    val arrayInt = arrayOf(1, 2, 3, 4, 5, 6)
    val arrayString = arrayOf("Eko", "Kurniawan", "Khannedy")
    displayLength(arrayInt)
    displayLength(arrayString)
}
```

Type Erasure

Type Erasure

- Type erasure adalah proses pengecekan generic pada saat compile time, dan menghiraukan pengecekan pada saat runtime
- Type erasure menjadikan informasi generic yang kita buat akan hilang ketika kode program kita telah di compile menjadi binary file
- Compiler akan mengubah generic parameter type menjadi tipe Any (atau Object di Java)

Kode Program : Type Erasure

```
package belajar.generic.app

class TypeErasure<T>(param: T) {
    private val data: T = param
    fun getData(): T = data
}

fun main() {
    val data = TypeErasure("Eko")
    val name = data.getData()
```

```
    d2 = {"Lbelajar/generic/app/TypeEras
)
public final class TypeErasure {
    private final Object data;

    public final Object getData() { retu

    public TypeErasure(Object param) {
}
// TypeErasureKt.java
package belajar.generic.app;
```

Problem Type Erasure

- Karena informasi generic hilang ketika sudah menjadi binary file
- Oleh karena itu, konversi tipe data generic akan berbahaya jika dilakukan secara tidak bijak

Kode Program : Problem Type Erasure

```
7
8   fun main() {
9       val data = TypeErasure("Eko")
10      val name = data.getData()
11
12      val eko = data as TypeErasure<Int>
13      val number = data.getData() // error runtime
14
15      println(name)
16  }
```

Comparable Interface

Comparable

- Sebelumnya kita sudah tahu bahwa operator perbandingan == dan != akan menggunakan metode equals sebagai implementasinya.
- Bagaimana dengan operator perbandingan lainnya? Seperti > >= < <= ?
- Operator perbandingan tersebut bisa kita lakukan, jika object kita mewariskan interface generic Comparable

Kode Program : Comparable (1)

```
2
3   class Fruit(val name: String, val quantity: Int) :
4       Comparable<Fruit> {
5
6       override fun compareTo(other: Fruit): Int {
7           return quantity.compareTo(other.quantity)
8       }
9   }
10
11
```

Kode Program : Comparable (2)

```
2
3   import belajar.generic.data.Fruit
4
5   fun main() {
6       val fruit1 = Fruit("Mangga", 10)
7       val fruit2 = Fruit("Mangga", 100)
8
9       println(fruit1 > fruit2)
10      println(fruit1 < fruit2)
11  }
```

ReadOnlyProperty Interface

ReadOnlyProperty Interface

- Sebelumnya kita sudah belajar tentang delegate di Kotlin
- Di Kotlin, ada sebuah interface generic yang bisa digunakan sebagai delegate property yang sifatnya readonly, alias val (immutable), namanya ReadOnlyProperty
- ReadOnlyProperty bisa digunakan sebagai delegate, sehingga sebelum data kita kembalikan, kita bisa melakukan sesuatu, atau bahkan mengubah value si property

Kode Program : ReadOnlyProperty (1)

```
5
6   class NameWithLog(param: String) {
7       val name: String by LogReadOnlyProperties(param)
8   }
9
10  class LogReadOnlyProperties(val data: String) : ReadOnlyProperty<Any, String> {
11      override fun getValue(thisRef: Any, property: KProperty<*>): String {
12          println("Access property ${property.name} with value $data")
13          return data.toUpperCase()
14      }
15  }
16
```

Kode Program : ReadOnlyProperty (2)

```
14      }
15  }
16
17  fun main() {
18      val name = NameWithLog("Eko")
19      println(name.name)
20  }
```

ReadWriteProperty Interface

ReadWriteProperty Interface

- Selain ReadOnlyProperty, kita juga menggunakan interface generic ReadWriteProperty sebagai delegate
- ReadWriteProperty bisa digunakan untuk variable var (mutable)

Kode Program : ReadWriteProperty (1)

```
6   class StringLogReadWriteProperty(var data: String) : ReadWriteProperty<Any, String> {
7
8       override fun getValue(thisRef: Any, property: KProperty<*>): String {
9           println("You get data ${property.name} is $data")
10          return data
11      }
12
13      override fun setValue(thisRef: Any, property: KProperty<*>, value: String) {
14          println("You set data ${property.name} from $data to $value")
15          data = value
16      }
```

Kode Program : ReadWriteProperty (2)

```
4
5   class Person(param: String) {
6       var name: String by StringLogReadWriteProperty(param)
7   }
8
9   fun main() {
10      val person = Person("Eko")
11      person.name = "Budi"
12      println(person.name)
13  }
```

ObservableProperty Class

ObservableProperty Class

- Generic interface delegate yang sebelumnya kita gunakan (ReadOnlyProperty dan ReadWriteProperty) kita perlu mengatur value datanya secara manual
- Kadang kita hanya butuh melakukan sesuatu sebelum dan setelah data nya diubah
- Untuk kasus seperti ini, kita bisa menggunakan generic class ObservableProperty

Kode Program : ObservableProperty (1)

```
class LogObservableProperty<T>(data: T) : ObservableProperty<T>(data) {

    override fun beforeChange(property: KProperty<*>, oldValue: T, newValue: T): Boolean {
        println("Before change value from $oldValue to $newValue")
        return true
    }

    override fun afterChange(property: KProperty<*>, oldValue: T, newValue: T) {
        println("After change value from $oldValue to $newValue")
    }
```

Kode Program : ObservableProperty (2)

```
3   class Car(brand: String) {
4       var brand: String by LogObservableProperty<String>(brand)
5   }
6
7   fun main() {
8       val car = Car("Toyota")
9       car.brand = "Wuling"
10      println(car.brand)
11  }
```

Object Delegates

Function	Keterangan
Delegates.notNull()	ReadWriteProperty yang nilai awal bisa null, namun error jika masih null
Delegates.vetoable(value, beforeChange)	ObservableProperty dengan beforeChange
Delegates.observable(value, afterChange)	ObservableProperty dengan afterChange

Generic Extension Function

Generic Extension Function

- Generic juga bisa digunakan pada extension function
- Dengan begitu kita bisa memilih jenis generic parameter type apa yang bisa menggunakan extension function tersebut

Kode Program : Generic Extension Function (1)

```
3   class Data<T>(val data: T)
4
5   fun Data<String>.print() {
6       val string = this.data
7       println("String value is $string")
8   }
9
10  fun main() {
11      val data1: Data<Int> = Data(1)
12      val data2: Data<String> = Data("Eko")
13
```

Kode Program : Generic Extension Function (2)

```
9
10  fun main() {
11      val data1: Data<Int> = Data(1)
12      val data2: Data<String> = Data("Eko")
13
14      data1.print() // error
15      data2.print()
16  }
```

Materi Selanjutnya

Materi Selanjutnya

- Kotlin Collection
- Kotlin Coroutine

Eko Kurniawan Khannedy

- Telegram : @khannedy
- Facebook : fb.com/khannedy
- Twitter : twitter.com/khannedy
- Instagram : instagram.com/programmerzamannow
- Youtube : youtube.com/c/ProgrammerZamanNow
- Email : echo.khannedy@gmail.com