

Föreläsning 2



Innehåll

Vad är programmering?	2
Kotlin	2
Variabler	3
Egenskaper	4
Metoder	4
Parametrar och argument	5
Villkorssatser	5
Loopar	5
Arrayer	6

Vad är programmering?

Programmering är processen att skriva instruktioner som en dator kan följa för att utföra specifika uppgifter. Dessa instruktioner, eller kod, skrivs i olika programmeringsspråk som är designade för att vara förståeliga för både mänsklig och datorer. Programmering handlar om att lösa problem genom att bryta ner dem i mindre, hanterbara delar och sedan utveckla logiska steg för att uppnå en lösning. Genom att kombinera dessa steg kan man skapa allt från enkla kalkylatorer till komplexa system som styr flygplan eller hanterar stora datamängder i molnet.

Ett av de centrala koncepten inom programmering är *algoritmer*, som är systematiska steg-för-steg-instruktioner för att lösa ett problem. Programmerare använder också *datastrukturer*, som arrayer, listor och träd, för att organisera och manipulera data på ett effektivt sätt. För att skapa flexibla och återanvändbara lösningar använder sig programmerare ofta av *funktioner* eller *metoder*, vilket är block av kod som kan anropas flera gånger med olika indata.

Ett annat viktigt koncept inom programmering är *kontrollflöde*, som styr ordningen i vilken instruktionerna utförs. Detta inkluderar villkorssatser, som tillåter olika kodvägar beroende på specifika förhållanden, samt loopar som gör det möjligt att upprepa vissa operationer.

Objektorienterad programmering (OOP) är en populär paradigm där program struktureras kring objekt, som är instanser av klasser, och deras interaktioner.

Programmering är också en kreativ process där kod kan optimeras och förbättras för att bli mer effektiv och lättare att underhålla. Medan syntaxen för olika språk kan variera, är grundläggande koncept som variabler, loopar och funktioner gemensamma. Genom att behärska dessa koncept kan programmerare bygga programvara som inte bara löser problem, utan också skapar nya möjligheter för innovation och utveckling inom nästan alla områden av modern teknik.

Kotlin

Kotlin är ett modernt, statiskt typat programmeringsspråk som utvecklats av JetBrains och är designat för att vara fullt kompatibelt med Java. Det introducerades 2011 och har snabbt blivit ett populärt val för Android-utveckling tack vare sin konkisa syntax, säkerhet och höga interoperabilitet med Java. Kotlin förenklar många av de komplexa aspekterna av Java, till exempel genom att eliminera null pointer exceptions, vilket gör kodningen mer robust och mindre benägen för fel. Kotlin erbjuder också avancerade funktioner som data-klasser, smart casting och utökande funktioner, vilket gör det möjligt att skriva mer kompakt och effektiv kod. En av de största fördelarna med Kotlin är dess stöd för korutiner, vilket gör det enklare att hantera asynkrona uppgifter utan att behöva hantera trådar på låg nivå. Sedan 2017 har Kotlin varit ett officiellt språk för Android-utveckling, och Google rekommenderar det starkt för alla nya Android-projekt. Dessutom kan Kotlin användas för utveckling av server-side applikationer, webbapplikationer, och även för iOS genom Kotlin/Native. Med sitt breda användningsområde och stora stöd från både utvecklare och industrin fortsätter Kotlin att växa som ett kraftfullt verktyg för modern mjukvaruutveckling.

Variabler

Variabler i Kotlin används för att lagra data som kan användas och manipuleras under programmens gång. De kan deklareras med antingen val, för värden som inte ska ändras, eller var, för värden som kan ändras senare. Variabler kan vara av olika typer, såsom Int för heltal eller String för text. En viktig del av variabler i Kotlin är att typen ofta kan härledas automatiskt, vilket gör koden mer kompakt och lättläst. Här är några av de vanligaste datatyperna:

- **Int** (Används för att representera heltal)

```
val age: Int = 25
val year: Int = 2024
```

- **Double** (Används för att representera tal med decimaler, som flyttal)

```
val price: Double = 19.99
val weight: Double = 72.5
```

- **Float** (Används för att representera flyttal med enkel precision. Flyttal med dubbel precision (**Double**) är vanligare, men **Float** används ibland när minnesanvändningen är kritisk)

```
val pi: Float = 3.14f
val distance: Float = 1234.56f
```

- **Boolean** (Används för att representera sanningsvärden, antingen true eller false)

```
val isActive: Boolean = true
val hasFinished: Boolean = false
```

- **String** (Används för att representera text)

```
val name: String = "Alice"
val greeting: String = "Hello, world!"
```

- **Char** (Används för att representera ett enskilt tecken)

```
val initial: Char = 'A'
val grade: Char = 'B'
```

- **Long** (Används för att representera stora heltal som går utanför gränsen för **Int**)

```
val population: Long = 7830000000L
val distanceToSun: Long = 149600000000L
```

- **Short** (Används för att representera små heltal när minnesanvändningen är viktig)

```
val smallNumber: Short = 32767
```

- **Byte** (Används för att representera mycket små heltal, ofta i låg nivå eller nätverksrelaterad programmering)

```
val byteValue: Byte = 127
```

Metoden main() nedan innehåller sju variabler av olika typer.

```
fun main() {
    val age: Int = 30
    val height: Double = 1.75
    val isEmployed: Boolean = true
    val name: String = "John Doe"
    val initial: Char = 'J'
    val population: Long = 7800000000L
    val byteValue: Byte = 100

    println("Name: $name")
```

```

        println("Age: $age")
        println("Height: $height meters")
        println("Employed: $isEmployed")
        println("Initial: $initial")
        println("World Population: $population")
        println("Byte Value: $byteValue")
    }
    
```

Egenskaper

Egenskaper, eller properties, är variabler som är knutna till ett objekt eller en klass och används för att lagra dess tillstånd. De kan ha anpassade getter- och setter-metoder för att kontrollera hur värden tilldelas eller hämtas, vilket ger flexibilitet och säkerhet i hur data hanteras. Egenskaper kan vara både privata och offentliga, beroende på hur du vill exponera dem för andra delar av programmet.

```

// name är en förändringsbar egenskap med en anpassad getter och setter. isAdult är
// en val-egenskap med en anpassad getter som returnerar true om age är 18 eller äldre.
// age är en annan förändringsbar egenskap.
    
```

```

class Person {
    var name: String = "Unknown"
        get() = field
        set(value) {
            field = value
        }

    val isAdult: Boolean
        get() = age >= 18

    var age: Int = 0
}

val person = Person()
person.name = "Alice"
println(person.isAdult)
    
```

Metoder

Metoder, eller funktioner, är block av kod som utför specifika uppgifter och kan återanvändas på olika ställen i programmet. De kan innehålla logik för att manipulera data, interagera med användargränssnittet, eller anropa andra metoder. En metod kan ha ett returvärde, vilket är resultatet av metoden, eller vara void om den inte returnerar något. Metoder bidrar till att strukturera och organisera koden på ett sätt som gör den mer modulär och lätt att underhålla.

```

// skriver ut "Hello!" när den anropas och tar inga parametrar.
fun greet() {
    println("Hello!")
}
    
```

Parametrar och argument

Parametrar är variabler som definieras i metoddeklarationen och används för att skicka in data till metoden. När en metod anropas skickas specifika värden, kallade argument, in som fyller dessa parametrar. Parametrar gör metoder flexibla genom att de kan utföra samma uppgift men på olika data beroende på de argument som skickas in.

```
// Denna metod skriver ut "Hello, Alice!" när den anropas. Den tar en parameter
fun greetUser(name: String) {
    println("Hello, $name!")
}
greetUser("Alice")
```

Villkorssatser

Villkorssatser används för att fatta beslut i koden, baserat på vissa villkor. Den vanligaste typen av villkorssats är if-else, som utför olika kodblock beroende på om ett uttryck är sant eller falskt. Kotlin har även when-satser, som är ett kraftfullt verktyg för att hantera flera fall utan att behöva skriva långa if-else-kedjor. Villkorssatser är grundläggande för att styra flödet i ett program.

- if – else

```
// kontrollerar om age är 18 eller högre. Om det är sant, skriver den ut att personen är
// vuxen, annars att personen inte är vuxen.
val age = 18
if (age >= 18) {
    println("You are an adult.")
} else {
    println("You are not an adult.")
}

// kontrollerar värde på variabeln day. Om day är lika med "Monday", skriver den ut "Start
// of the week!", osv.
val day = "Monday"
when (day) {
    "Monday" -> println("Start of the week!")
    "Friday" -> println("Almost weekend!")
    else -> println("Just another day.")
}
```

Loopar

Loopar är konstruktioner som används för att upprepa en uppgift flera gånger, ofta tills ett visst villkor är uppfyllt. Kotlin erbjuder flera typer av loopar, inklusive for-loopar för att iterera över en samling och while-loopar som körs så länge ett villkor är sant. Loopar är viktiga för att effektivt hantera repetitiva uppgifter i kod, som att bearbeta alla element i en lista.

- for

```
// itererar från 1 till 5 och skriver ut varje värde av i i varje iteration
for (i in 1..5) {
    println(i)
```

```

}

• while

// fortsätter att iterera så länge count är större än 0. Den skriver ut count och minskar
// det med 1 för varje iteration
var count = 5
while (count > 0) {
    println(count)
    count--
}
• do – while

var count = 5
do {
    println(count)
    count--
} while (count > 0)

```

Arrayer

Arrayer är datastrukturer som lagrar flera värden av samma typ i en ordnad sekvens. I Kotlin kan du skapa en array och sedan komma åt eller ändra dess element genom deras index. Arrayer är användbara när du behöver hantera en fast mängd data som ska behandlas på liknande sätt, som en lista med användarnamn eller ett antal siffror. Kotlin erbjuder även en rad metoder för att manipulera och bearbeta arrayer på ett effektivt sätt.

```

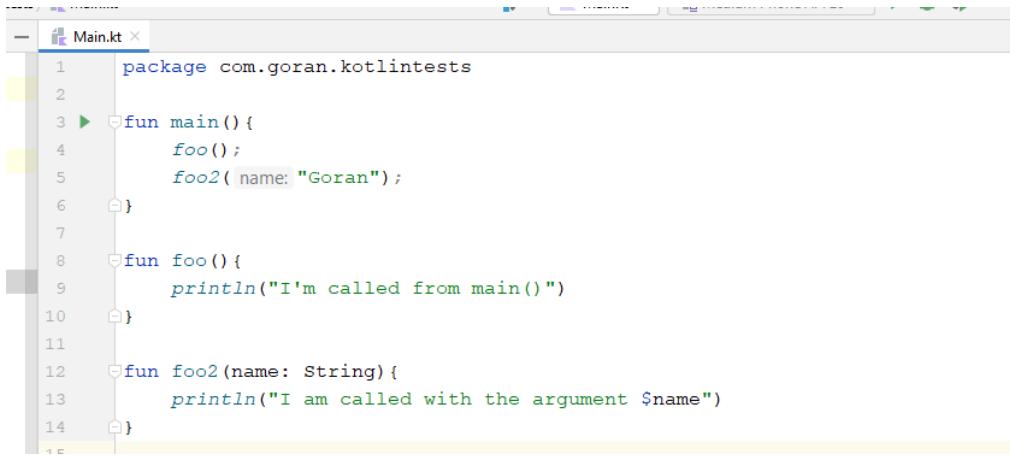
// numbers är en array som innehåller fem heltal. for-loopen itererar genom varje
// element i arrayen och skriver ut dem.
val numbers = arrayOf(1, 2, 3, 4, 5)

for (number in numbers) {
    println(number)
}

```

Testa gärna alla kodexempel och försök att skriva egen kod baserad på innehållet i denna föreläsning. Du kan till exempel skapa en metod som tar ett flyttal som argument, radius, och sedan beräknar och skriver ut omkretsen och arean av en cirkel som definieras av det givna radiet. För att köra Kotlin-kod utan att skapa en app, gör så här:

- Skapa ett nytt projekt i Android Studio: Gå till *File > New > New Project...* och välj alternativet *No Activity*.
- Klicka på *Next*, ange ett namn för ditt projekt och välj var det ska sparas. OBS: Kontrollera att *Kotlin* är valt som programmeringsspråk.
- För att din kod ska kunna köras måste den skrivas inuti metoden *main()*. Om du har andra metoder som du vill köra, måste dessa anropas från *main()*.
- För att köra din kod, klicka på den gröna pilen till vänster om *main()*-metoden och välj *Run 'MainKt'*.



```
1 package com.goran.kotlintests
2
3 fun main() {
4     foo();
5     foo2(name: "Goran");
6 }
7
8 fun foo(){
9     println("I'm called from main()")
10}
11
12 fun foo2(name: String){
13     println("I am called with the argument $name")
14}
```

Med denna uppsättning kan du enkelt testa och köra din Kotlin-kod i Android Studio utan att behöva skapa en fullständig Android-app.