```
import dataClasses.Point

fun main() {
    val point1 = Point(3,5)
    val point2 = Point(3,5)

    println("Point 1: $point1")
    println("Ponit 2 : $point2")
    println()//uj sor
    println("Are points equals ? :")
    println(point1==point2)//equals
    println()//uj sor
    println("hash code of Point 1 : ${point1.hashCode()}")//hashcode
    println("hash code of Point 2 : ${point2.hashCode()}")

}
```

```
package dataClasses

data class Point(val x:Int, val y:Int)
```

Kimenet Output :

```
Point 1: Point(x=3, y=5)
Ponit 2 : Point(x=3, y=5)

Are points equals ? :
true

hash code of Point 1 : 98
hash code of Point 2 : 98

Process finished with exit code 0
```

Nagyon jó kérdés, Ati! Lépésenként tisztázzuk, **mi az a `hashCode()`, miért működik itt a `==`**, és **mi köze van az `equals()`-hoz**, valamint **miért működik `String`-re is a `==` Kotlinban**.

✅ Mi az a `hashCode()`?

A `hashCode()` egy **egész szám (Int)**, amit egy objektum alapján számol ki a rendszer. Ez a szám segít az objektumok **gyors összehasonlításában**, főleg:

- **Hash-alapú kollekciókban**, pl. `HashSet`, `HashMap`
- Ha két objektum `equals()` szerint egyenlő, akkor a `hashCode()`-juknak is azonosnak **kell lennie**

Példa:

```
val a = Point(3, 5)
val b = Point(3, 5)

println(a == b)             // true, mert equals()
println(a.hashCode() == b.hashCode()) // true
```

Ha `data class`, akkor **automatikusan** generálja a megfelelő `equals()` és `hashCode()` függvényeket, az **összes property** alapján.

✅ Miért használhatjuk `==`-t az `equals()` helyett?

Kotlinban a `==` operátor **nem referenciát** hasonlít össze, hanem a `equals()`-t hívja meg:

```
a == b  // Kotlin -> a.equals(b)
```

Referencia-összehasonlításhoz:

Használhatod a `===` operátort, ami azt vizsgálja, hogy **ugyanarra a memóriacímre** mutatnak-e:

```
val a = Point(3, 5)
val b = Point(3, 5)
val c = a

println(a == b)   // true – érték szerinti összehasonlítás (equals)
println(a === b)  // false – nem ugyanaz az objektum (nem ugyanaz a referencia)
println(a === c)  // true – ugyanaz az objektum
```

✅ Miért működik `==` `String`-ekre is?

Ugyanazért: Kotlinban a `==` mindig az `equals()`-t hívja meg. A `String` típuson az `equals()` tartalom szerint hasonlít:

```
val s1 = "hello"
val s2 = "hello"

println(s1 == s2)  // true
println(s1 === s2) // lehet true vagy false, attól függ, internálva van-e
```

🔁 Összefoglalás

Művelet	Jelentés Kotlinban	Megjegyzés
`==`	`equals()` hívás	Tartalom szerinti összehasonlítás
`===`	Referencia-azonosság	Ugyanaz az objektum
`hashCode()`	Objektum „azonosító” szám	Hash-alapú kollekciók használják
`data class`	Automatikusan generálja `equals()` + `hashCode()`	Ezért kényelmes

Nagyszerű kérdés, Ati! Nézzük meg részletesen, **hogyan számolja ki a `hashCode()`-t a Kotlin (és Java)**, és **mi történik egy `data class`-nál**.

🔢 Mi az a `hashCode()` ténylegesen?

A `hashCode()` egy **egész szám (`Int`)**, amit az objektum tulajdonságai alapján számolnak ki. Ez a szám lehetőleg **egyedi az objektum állapota** szerint, de nem garantáltan különböző (ütközés előfordulhat).

🔍 `data class` hashCode kiszámítása

Amikor írsz egy `data class`-t, pl.:

```
data class Point(val x: Int, val y: Int)
```

akkor a fordító automatikusan generál egy ehhez hasonló `hashCode()` metódust:

```
override fun hashCode(): Int {
    var result = x
    result = 31 * result + y
    return result
}
```

Ez egy **tipikus Java stílusú hash kód számítási séma**, amit a Kotlin is használ.

🔢 Mi történik itt pontosan?

Tegyük fel:

- `x = 3`
- `y = 5`

A `hashCode()` kiszámítása:

```
var result = x           // result = 3
result = 31 * result + y // result = 31 * 3 + 5 = 93 + 5 = 98
```

> A `31` egy **prímszám**, amit gyakran használnak hash kódoknál, mert:
>
> - hatékony számítás (szorzás és összeadás gyors),
> - jól szórja szét az értékeket,
> - kevesebb ütközést okoz a gyakorlatban.

🧠 Általános szabály:

Ha van egy ilyen osztályod:

```
data class User(val name: String, val age: Int)
```

A hashCode automatikusan kiszámítódik, pl.:

```
override fun hashCode(): Int {
    var result = name.hashCode()
    result = 31 * result + age
    return result
}
```

A `String` is rendelkezik saját `hashCode()`-val (minden beépített típus igen), így az objektum **összes mezője hozzájárul a végső hash értékhez**.

🤖 Ha nem `data class`?

Ha sima `open class`, és te szeretnéd használni `==` és `hashCode` összehasonlítást, akkor **magadnak kell felülírni**:

```
class Point(val x: Int, val y: Int) {
    override fun equals(other: Any?): Boolean {
        if (this === other) return true
        if (other !is Point) return false
        return x == other.x && y == other.y
    }

    override fun hashCode(): Int {
        var result = x
        result = 31 * result + y
        return result
    }
}
```

Ezért a `data class` **óriási könnyítés**: nem kell ezt mindig kézzel megírni.
