Objects

Objects are the basis of object-oriented programming. In Kotlin, every piece of data is an object.

An object

- stores data (has state), and
- provides methods to access or manipulate its state.

The object ensures that its state is consistent.

In most classes, the state is hidden: we can only access the state of an object through its methods.

For instance, `List` is implemented using an array, but we cannot access this array directly. And we don't even know how `Map` and `Set` are implemented.

Hidden state

We need an accumulator that keeps a running total of some purchases:

```
class Accumulator {
  var sum = 0
  fun add(n: Int) { sum += n }
}
```

no class arguments

must initialize!

```
>>> val acc1 = Accumulator()
>>> val acc2 = Accumulator()
>>> acc1.add(13); acc2.add(13)
>>> acc1.add(17); acc2.add(4)
>>> acc2.add(44)
>>> acc1.sum
30
>>> acc2.sum
61
```

still needed!

accum1.kts

Data classes and general classes

So far, we have only defined simple classes where all fields are publicly visible and defined as class arguments:

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

In general classes, fields can be defined both as class arguments and inside the class:

```
class ColoredPoint(val x: Int, val y: Int) {
  var color = Color.WHITE
}
```

```
>>> val p = ColoredPoint(2, 3)
>>> p.color
Color(r=255, g=255, b=255)
>>> p.color = Color.RED
>>> p.color
Color(r=255, g=0, b=0)
```


Privacy

Clients of `Accumulator` should consider it as a black box with two operations: Add a number to the running sum; and read out the running sum.

Solution: make methods and fields `private`, and they can only be used from within methods of the class:

```
class Accumulator {
  private var current = 0
  fun add(n: Int) { current += n }
  fun sum(): Int = current
}
```

accum2.kts

Class arguments

Classes can have class arguments that are not fields:

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

class Rect(x: Int, y: Int,
           val width: Int, val height: Int) {
  var corner = Point(x, y)
  init { require(width > 0 && height > 0) }
}
```

Not fields!

Created during construction of Rect

Blackjack

(There are 52 cards. Each card has a face and a suit. The suits are clubs, spades, hearts, and diamonds. The faces are 2, 3, . . . , 10, Jack, Queen, King, Ace.)

```
data class Card(val face: String, val suit: String) {
  init {  require(suit in Suits && face in Faces)  }

  fun value(): Int = when(face) {
    "Ace" -> 11
    "Jack" -> 10
    "Queen" -> 10
    "King" -> 10
    else -> face.toInt()
  }
}
```

blackjack1.kt

The Deck

The Deck class stores an entire deck of cards. It stores the list of cards as hidden state:

```
class Deck {
  private val cards = mutableListOf<Card>()
  init {
    generateDeck()
    shuffleDeck()
  }
  private fun generateDeck() { ... }
  private fun shuffleDeck() { ... }
  fun draw(): Card {
    assert(!cards.isEmpty())
    return cards.removeAt(cards.lastIndex)
  }
}
```

empty list

code to fill deck with cards

private methods

blackjack2.kt

The game

```
fun blackjack(): Int {
  val deck = Deck()

  // initial cards
  var player = mutableListOf(deck.draw())
  println("You are dealt " + player.first())
  var dealer = mutableListOf(deck.draw())
  println("Dealer is dealt a hidden card")

  player.add(deck.draw())
  println("You are dealt " + player.last())
  dealer.add(deck.draw())
  println("Dealer is dealt " + dealer.last())
  println("Your total is ${handValue(player)}")
  // ...
}
```

blackjack-game.kt