Day 2: Case Classes

Synopsis

Case classes give you a lot of stuff for free. Companion objects contain `apply` and `unapply`, which allow object construction without `new`, and pattern matching against objects, respectively. Better `toString`, better object equality comparison (via `hashCode`), free `copy` method, and can do more with your `apply` method!

Note: case class to case class inheritance is prohibited, due to the usage of `apply` for construction instead of standard Java constructors.

Note: if you want to override methods of a case class at object creation time (step zero: ask why?), you can use the `new` keyword to allow that.

More specific details in this good article

Examples:

```
class C(a: Int, b: String)

case class CC(a: Int, b: String) {
  def this(a: Int) = this(a, "")
  def this(b: String) = this(-1, b)

  def member: Int = 3
}

new CC(3, "hi") // uses `apply`, `new` keyword not necessary
CC(3, "hi") // uses `apply`
new CC(3) // uses first aux. constructor
new CC("hi") // uses second aux. constructor

// CC(3, "hi") { override def member: Int = 4 } // not allowed! need to use `new` keyword
new CC(3, "hi") { override def member: Int = 4 } // allowed!

val cc1 = CC(3, "hi")
println(cc1.a, cc1.b) // provides free getters by setting constructor fields to member
fields

val cc2 = new CC(3)
println(cc2.a, cc2.b) // using an aux constructor doesn't change anything

val c1 = new C(3, "hi")
// println(c1.a, c1.b) // can't do this, no private access
```

```
26
27 println(c1, cc1) // memory location vs. contents, CC has better `toString`
28 println(cc1 == CC(3, "hi")) // true, case classes can do element-wise comparisons
29
30 // can deconstruct case class via `unapply`
31 cc1 match {
32   case CC(3, "hi") => true
33   case _ => true
34 }
35
36 /* can't do this, no unapply
37 c1 match {
38   case C(3, "hi") => true
39   case _ => true
40 } */
```