Objects

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

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

Objects are atomic units. Clients (users of the object) have no access to the details of an object that it does not expose.

A class defines a new type of object. Think about a class as a blueprint for objects. You can create objects from the blueprint with `new`.

Mutable and immutable objects

If the state of an object cannot change after the object has been constructed, it is immutable. In Scala, `String`, `List`, and tuples are immutable.

If the state of an object can change, it is mutable. Arrays are mutable objects.

A simple mutable class for two-dimensional points:

```
class Point(var x: Int, var y: Int)
```

A simple immutable class for blackjack cards:

```
class Card(val face: String,  val suit: String)
```

(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.)

References

What happens here?

```
scala> val A = Array(1, 2, 3, 4)
A: Array[Int] = Array(1, 2, 3, 4)
scala> val B = A
B: Array[Int] = Array(1, 2, 3, 4)
scala> A(2) = 99
scala> B
res1: Array[Int] = Array(1, 2, 99, 4)
```

All Scala objects live on the heap, a memory area of the JVM.

The contents of a variable is a reference to the object. A reference uniquely identifies one object on the heap. (Similar to a pointer in C.)

null

When a variable has value `null`, it means that it does (not yet) contain a reference to an object.

Any operation on such a variable will fail, since there is no object to operate on!

```
scala> var m = null
m: Null = null

scala> m.toString
java.lang.NullPointerException
```

For efficiency reasons, variables of the types `Int`, `Byte`, `Short`, `Long`, `Double`, `Float`, `Char`, `Boolean`, and `Unit` cannot be `null`.

`new Array[String](10)` creates an array full of `null`s.

Local variables

So where do variables live?

If it is a field of an object, it lives inside the object on the heap. In particular, the elements of an array live inside the array object.

The local variables of a method live inside the method's activation record (also called stack frame).

Four local variables:

```
def test(m: Int) {
  val k = m + 27
  val s = "Hello World"
  val A = Array( s.length(), k, m )
}
```

Garbage collection

Many objects are used only briefly, and not needed afterwards. So after some time, the heap of the JVM will become full.

At that point, the JVM performs garbage collection: It checks all objects on the heap, and determines if there is any reference from a variable on some stack frame leading directly or indirectly to this object. If not, the object is destroyed.

You cannot easily predict when garbage collection happens. Modern systems may perform it incrementally.

Scala programs do not have to worry about memory leaks.