If two objects are `equal()` they must have the same `hashCode()`.

If the class is final and inherits equals from `Any`. There is no issue of subtype equality in this case.

Anonymous class instances won't be equal to non-anonymous ones.

In general, subtypes won't be able to equal supertypes.

Ignoring type parameters with `MyClass[_]` notation.

e.g., `Map[_, _]`

```
41* (
  41* (
    41* (
      41 + a.hashCode
    ) + b.hashCode
  ) + c.hashCode
) + d.hashCode
```

It's only as good as the hash codes you make it out of.

For example, most collections override `hashCode` for you, but Arrays do not.

For `Arrays`, has each element or use `java.util.Arrays.hashCode`.

Mult: odd primes minimize the potential for information loss on overflow.

Add: avoid the first field being zero, assuming zero is more likely than -41. Any non-zero integer is equally good.

- Reflexive
- Symmetric
- Transitive
- Consistent: provided info used by equals was not modified
- Not `equal()` to null

- Defining `equals()` with wrong signature.
- Changing `equals()` without changing `hashCode()`.
- Defining `equals()` in terms of mutable fields.
- Failing to define `equals()` as an equivalence relation.

```
class X extends Y {
  def canEqual(other:Any) : Boolean = {
    other.isInstanceOf[X]}
override def equals(other:Any) : Boolean =
  other match {
    case that: X => {super.equals(that) &&
    (that canEqual this) && fields match }
    case _ => false
if extending AnyRef no super call,
}
```

Start your `hashCode()` with that invocation.

```
41 * (
  super.hashCode()
  ) + a.hashCode()
) + b.hashCode()
```

For immutable objects, override `hashCode` with similarly named val.

For mutable objects, use caching.

Subtypes.

Say A extends B.

```
val a = new A ; val b = new B
```

"a equals b" and "b equals a" use different versions of equals!
So just overriding A's is insufficient.

Scala's takes Any instead of `AnyRef`/`Object`.

It's just a fiction of the compiler; it's the same method.

```
final def == (that:Any) : Boolean =
  if (null eq this) {null eq that}
  else {this equals that}
```