PM1/PT Ruby: Objektidentität und Objektgleichheit

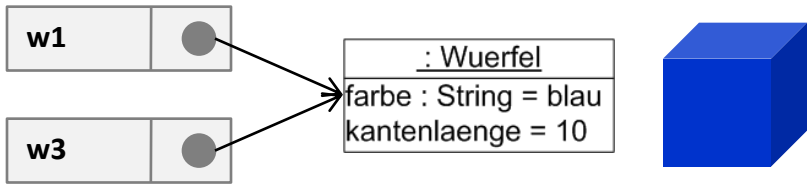
Inhalt

- Objektgleichheit versus Objektidentität
- Eigenschaften einer Gleichheitsrelation
- Die zwei Arten von Gleichheit in Ruby
- Der Zusammenhang von ***eql?*** und ***hash***
- Verträge zwischen ***Array*** und ***==***
- Verträge zwischen ***Hash, Set*** und ***eql?*** / ***hash***

Objekte haben Identität

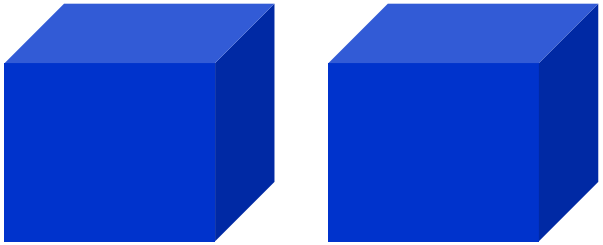
- **Identität** bezeichnet kennzeichnende und von anderen unterscheidende Eigenschaften eines Individuum.
- Wenn Sie einen Freund abends in der Kneipe treffen, dann wissen Sie, dass Ihnen genau dieser Freund gegenüber steht.
- Bei Namen verhält es sich schon anders. Wenn Sie in der Zeitung über Bernard Madoff (ein „berühmter“ Wirtschafts-betrüger) lesen, dann fragen Sie sich ob dieser Herr derselbe wie ihr Finanzberater ist.
- In Programmiersprachen wird Identität durch Adresse des Objektes im Speicher abgebildet.
- Die Frage nach Identität, ist die Frage, ob zwei Variablen auf dasselbe Objekt im Speicher zeigen (dasselbe Objekt referenzieren).

```ruby
class Wuerfel
  attr_writer :farbe
  attr_reader :farbe
  def initialize(kantenlaenge,farbe)
    @kantenlaenge = kantenlaenge
    @farbe = farbe
  end

end

w1 = Wuerfel.new(10,'blau')
w3 = w1
```



Gleiche Objekte müssen nicht identisch sein

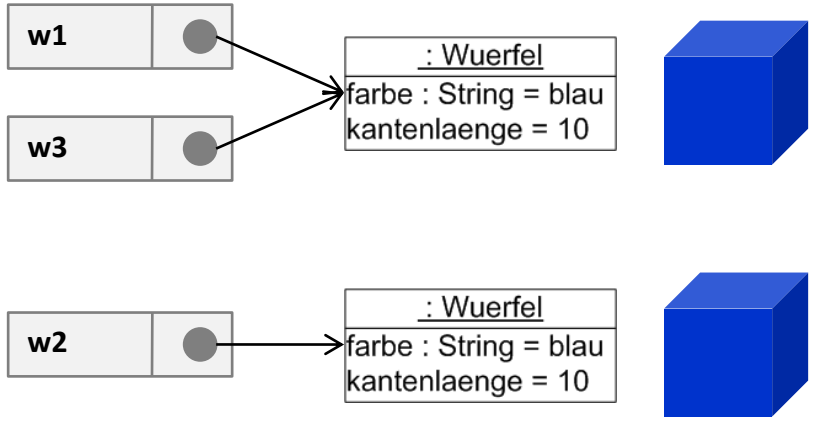
- **Gleichheit** bezeichnet die Übereinstimmung von Objekten im Hinblick auf eine Menge von Eigenschaften.
- Die Beispiele rechts zeigen gleiche aber nicht identische Objekte. Also nicht das „selbe“ Objekt.
- Eineiige Zwillinge sind gleich, aber nicht identisch: Sie unterscheiden sich in ihrer DNA.
- Zwei blaue Würfel mit gleicher Kantenlänge und gleicher Farbe sind gleich, aber nicht identisch. Sie unterscheiden sich in ihrer Adresse im Speicher (ausgedrückt durch ihre Objekt-Id).
- Auch wenn Objekte
  - → „gleich aussehen“ = gleiche Werte haben
  - → sind sie nicht identisch



Gleiche versus identische Objekte

- ***w1*** und **w3** sind **identische** Würfel. Beide Variablen zeigen auf dasselbe Objekt.
- ***w1*** und ***w2*** sind **gleiche** Würfel. Jede Variable zeigt auf ein eigenes Objekt. Die Objekte haben unterschiedliche Identität.

```
class Wuerfel
  attr_writer :farbe
  attr_reader :farbe
  def initialize(kantenlaenge,farbe)
    @kantenlaenge = kantenlaenge
    @farbe = farbe
  end

end

w1 = Wuerfel.new(10,'blau')
w2 = Wuerfel.new(10,'blau')
w3 = w1
```



Gleiche versus identische Objekte

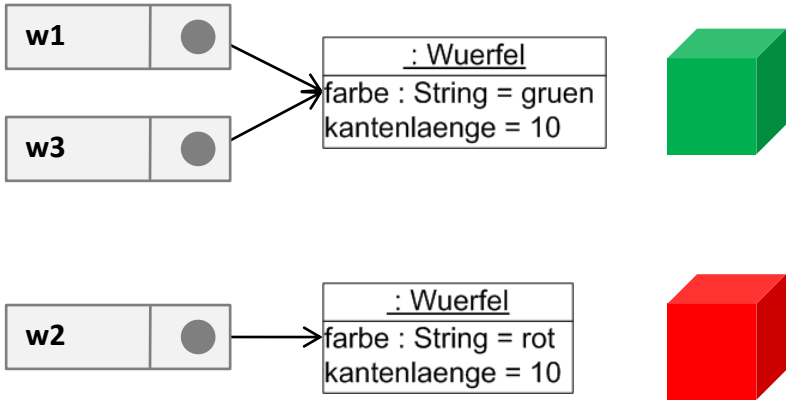
- ***w1*** und ***w3*** sind **identisch**: Änderungen an dem ***w1-Wuerfel*** Objekt werden auch in ***w3*** sichtbar und umgekehrt.
- ***w1*** und ***w3*** sind zu ***w2*** **gleich** aber nicht identisch: Änderungen an dem ***w1*** oder ***w3-Wuerfel*** Objekt werden in ***w2*** **nicht sichtbar** und umgekehrt.
- In Ruby wird Identität durch ***equal?*** geprüft.
- Dabei wird überprüft, ob zwei Variablen auf dasselbe Objekt zeigen.

```
class Wuerfel
  attr_writer :farbe
  attr_reader :farbe
  def initialize(kantenlaenge,farbe)
    @kantenlaenge = kantenlaenge
    @farbe = farbe
  end

end

w1 = Wuerfel.new(10,'blau')
w2 = Wuerfel.new(10,'blau')
w3 = w1

puts w1.equal?(w2)        ➔ false
puts w1.equal?(w3)        ➔  true
```

Gleiche versus identische Objekte

```
class Wuerfel
  attr_writer :farbe
  attr_reader :farbe
  def initialize(kantenlaenge,farbe)
    @kantenlaenge = kantenlaenge
    @farbe = farbe
  end

end

w1 = Wuerfel.new(10,'blau')
w2 = Wuerfel.new(10,'blau')
w3 = w1
w3.farbe = 'gruen'
w2.farbe = 'rot'
puts w1.farbe()          ➔ "gruen"
puts w3.farbe()          ➔ "gruen"
puts w2.farbe()          ➔ "rot"
```

- Ändern wir die Farbe eines Objektes, dann gilt die Änderung für alle Variablen, die das selbe Objekt referenzieren (***w1*** und ***w3***).
- Ändern wir die Farbe von zwei gleichen Objekten in unterschiedliche Farben, dann sind die Objekte anschließend **nicht mehr gleich**.



Gleiche versus identische Objekte

Identität

- Es handelt sich um das selbe Objekt. Zwei Variablen ***v1*** und ***v2*** zeigen auf **dasselbe** Objekt im Speicher.

Gleichheit

- Es handelt sich um Objekte

  1. die gleiche Eigenschaften haben
  2. die gleiche Beziehungen haben
  3. die gleichen Typ haben oder gleiches Verhalten zeigen.

- Zwei Variablen ***v1*** und ***v2*** sind auch dann gleich, wenn sie auf je ein Objekt im Speicher zeigen, deren Inhalte gleich sind.

Gleiche versus identische Objekte

Objektidentität in Ruby

- Wir prüfen Objektidentität mit ***equal?***
- Der Vergleich zweier Variablen mit ***equal?*** (***v1.equal?(v2)***) liefert dann und nur dann ***true***, wenn beide Variablen auf dasselbe Objekt zeigen.
- ***Object*** implementiert ***equal?*** für alle einheitlich.
- Identität ist elementar für objektorientierte Sprachen, daher darf die Implementierung von ***equal?*** **nie** überschrieben werden.

Objektgleichheit in Ruby

- Wir prüfen Objektgleichheit mit ***==*** (oder ***eql?***).
- Objekte erben die Implementierung von mit ***==*** (und ***eql?***) von ***Object.***
- ***Object*** definiert ***==*** (und ***eql?***) als **Identität.**
- Wir wollen aber **Gleichheit** und **nicht Identität** bei ***==*** (und ***eql?***) überprüfen.
- Daher müssen eigene Klassen die Methoden ***==*** (und ***eql?***) überschreiben.

Eigenschaften der Gleichheitsrelation

Eigenschaften

- **reflexiv**: *x == x*
- *x. equal? x => true*
- **symmetrisch**: *x== y => y == x*
- *x. equal? y => y equal? x*

- **transitiv**: *x== y && y == z => x == z*
- **transitiv**: *x.equal? y &&*
- *y.equal? z => x equal? z*
- Ist eine dieser Eigenschaften verletzt, dann können gleiche Objekte in Objektsammlungen nicht wiedergefunden werden.

Vorläufige Definition von *==*

- Zwei Objekte sind *==* -gleich genau dann, wenn
  - wenn eines der beiden Objekten *nil* ist, dann muss auch das andere gleich *nil* sein.
  - sie identisch sind
  - oder sie in **allen** relevanten Instanzvariablen gleich sind. Dabei müssen natürlich auch die Instanzvariablen der Superklassen berücksichtigt werden. Dies gilt, wenn Sie == entsprechend überschreiben.

- Zwei Objekte sind *eql?*-gleich genau dann, wenn
  - wenn sie *==*-gleich sind **und**
  - sie gleichen Typ haben.

Warum wollen wir Gleichheit überprüfen können? - den Vertrag mit Array eingehen? -

- um Dubletten aus Arrays zu löschen.
- um gleiche Objekte in Arrays wieder zu finden
- ***Array*** sucht Objekte mit ***==***
- Die Methoden ***include?***, ***member? , index*** und ***delete*** von ***Array*** verwenden ***==***, um Objekte in einem Array aufzuspüren oder zu löschen.
- **Vertrag**: Wenn Objekte eigener Klassen in Arrays verwendet werden, dann müssen die Klassen ***==*** überschreiben.

```
class C0
  def initialize(iarray)
    @iarray = iarray
  end
end

c01 =  C0.new([1,2,3])
c02 =  C0.new([1,2,3])
c03 = C0.new([0,2,3])
c04 = C0.new([0,2,3])
p 'c01 == c02: ' + (c01 == c02).to_s
p 'c01 eql? c02: ' + c02.eql?(c01).to_s
p 'c01 equal? c02: ' +
    c01.equal?(c02).to_s
ary = [c03, c01, c03]

p 'ary.index(C0.new([1,2,3])): ' +
    ary.index(C0.new([1,2,3])).to_s
p 'ary.index(c04): ' +
    ary.index(c04).to_s
```

false

werden nicht gefunden

Szenario

- Wir speichern den Inhalt eines Arrays von Kunden in einer Datei, weil wir die Kundenliste, die ein Sachbearbeiter überarbeitet hat, zwischen zwei Programmläufen retten wollen.
- Wir laden am nächsten Tag diese Kundenliste und wollen Personen in dieser Liste wiederfinden.
- Da wir beim Laden der Liste in einem neuen Ruby-Prozess sind, haben die Kundenobjekte eine andere Identität. Ihr Inhalt ist hingegen unverändert.
- Wir können also Objekte nicht über ihre Identität wiederfinden.

```
kermit = Kunde.new("Kermit", "der Frosch")
tier =  Kunde.new("das", "Tier")
homer =  Kunde.new("Homer","Simpson")

kunden = [kermit, tier, homer]

p(kunden)

# kundenliste in Datei schreiben
File.open("kundenliste","w"){ |datei|
  Marshal.dump(kunden,datei)
}
```

Identitäten der Kunden beim Schreiben

```
[#<Kunde:0x2bd06d0 @name="der Frosch",
@vorname="Kermit">, #<Kunde:0x2bd05b8
@name="Tier", @vorname="das">,
#<Kunde:0x2bd057c @name="Simpson",
@vorname="Homer">]
```

Szenario

- Kunde ist wie folgt definiert.
- Kunde erbt die Methode ***==*** von ***Object***.

```
class Kunde
  attr_reader :vorname,:name
  def initialize(vorname = "",name = "")
    @vorname = vorname
    @name = name
  end

  def to_s
    "#{@vorname}, #{@name} "
  end
end
```

Szenario

- Wir speichern den Inhalt eines Arrays in einer Datei, weil wir die Kundenliste, die ein Sachbearbeiter überarbeitet hat, zwischen zwei Programmläufen retten wollen.

- Wir laden am nächsten Tag diese Kundenliste und wollen Personen in dieser Liste wiederfinden.

- Wenn wir die Liste der Kunden laden, bekommen die Kundenobjekte eine andere Identität. Ihr Inhalt ist hingegen unverändert.

- Wir finden den Kunden ***tier*** in der Liste nicht wieder.

```
kunden = []
File.open("kundenliste","r") do
 |datei|
    # Marshal load liest den
    # Inhalt  der Datei:  ein Array
    kunden = Marshal.load(datei)
end
# Kundenobjekte haben neue Identität
p kunden
tier =  Kunde.new("das", "Tier")
p tier
# tier ist nicht in der Liste
puts kunden.include?(tier)
```

```
[#<Kunde:0x2bd057c @name="der Frosch",
    @vorname="Kermit">, #<Kunde:0x2bd0504
    @name="Tier", @vorname="das">,
    #<Kunde:0x2bd04b4 @name="Simpson",
    @vorname="Homer">]
#<Kunde:0x2bd039c @name="Tier",
    @vorname="das">
false
```

Identität von Objekten ändert sich beim (De)Serialisieren

Identitäten vor dem Schreiben

- `[#<Kunde:0x2bd06d0 @name="der Frosch", @vorname="Kermit">, #<Kunde:0x2bd05b8 @name="Tier", @vorname="das">, #<Kunde:0x2bd057c @name="Simpson", @vorname="Homer">]`

Identitäten nach dem Lesen

- `[#<Kunde:0x2bd057c @name="der Frosch", @vorname="Kermit">, #<Kunde:0x2bd0504 @name="Tier", @vorname="das">, #<Kunde:0x2bd04b4 @name="Simpson", @vorname="Homer">]`

Nach dem Laden haben Objekte eine neue Identität

- Die Identitäten aller Objekte haben sich nach dem Laden aus der Datei geändert, allerdings ist deren Inhalt gleich geblieben.
- Nach dem Laden können wir die Objekte also nicht mehr über ihre Identität wiederfinden.
- Uns bleibt keine Wahl: Wir müssen Objekte über den inhaltlichen Vergleich wieder-finden.
- Wir suchen in dem Array ***kunden*** mit ***include?*** nach dem Kunden ***"das Tier".***
- **Das Ergebnis**: Der Kunde wurde in der geladenen Kundenliste nicht gefunden. **WARUM?**

```
kunden = []
File.open("kundenliste","r") { |datei|
    # Marshal load liest den
    # Inhalt  der Datei:  ein Array
     kunden = Marshal.load(datei)
}
# Kundenobjekte haben neue Identität
p kunden
# tier ist nicht in der Liste
tier =  Kunde.new("das", "Tier")
puts kunden.include?(tier) ➔ false
```

Objekte über Gleichheit wiederfinden

Antwort auf das WARUM?

- Wir finden die Objekte nicht wieder, da unsere Kunden für die Prüfung der Gleichheit (***==***) die Implementierung von ***Object*** erben.
- In ***Object*** ist ***==*** als Identitätsprüfung definiert (***equal?()***)
- Da sich die Identitäten geändert haben, können wir die Objekte nicht wiederfinden.
- **Was müssen wir tun?** Die Methode ***==*** von ***Object*** der Klasse ***Kunde*** überschreiben, so dass sie für ***Kunde*** Gleichheit und nicht Identität prüft.

```
class Kunde
  attr_reader :vorname,:name

  def initialize(vorname="",name="")
    @vorname = vorname
    @name = name
  end

  def ==(other)
    return false if other.nil?
    return true if self.equal?(other)
    return[@vorname,@name] ==
            [other.vorname,other.name]
  end
end
```

Erläuterung zur Definition von Gleichheit (==)

- Zwei Objekte (ungleich *nil*) sind *==* genau dann, wenn
  - wenn beide ungleich *nil* sind und
  - sie entweder identisch sind
  - oder sie in **allen** relevanten Instanz-variablen gleich sind.

- Um die Übereinstimmung der Instanz-variablen zu prüfen, nutzen wir hier die *==* Implementierung von *Array*, die gleich lange Arrays solange überprüft, bis sich die Elemente das erste Mal unterscheiden.

- Dieses Element bestimmt dann, ob Arrays gleich, kleiner oder größer sind.

```
class Kunde
  attr_reader :vorname,:name

  def initialize(vorname = "",name = "")
    @vorname = vorname
    @name = name
  end

  def ==(other)
    return false if other.nil?
    return true if self.equal?(other)
    return [@vorname,@name] ==
           [other.vorname,other.name]
  end

end
```

Suchen von Kunden nach dem Laden mit überschriebenem *==* in *Kunde*

- Mit der Definition von ***==*** in der Klasse ***Kunde*** können wir jetzt Kunde ***tier*** in der geladenen Kundenliste wiederfinden.

```
kunden = []
# Datei kunden zum Lesen (r für read)
    öffen
File.open("kundenliste","r") do
 |datei|
    kunden = Marshal.load(datei)
end
p kunden
tier =  Kunde.new("das", "Tier")
p tier
puts kunden.include?(tier)
```

```
[#<Kunde:0x2bd0770 @name="der Frosch",
    @vorname="Kermit">, #<Kunde:0x2bd06f8
    @name="Tier", @vorname="das">,
    #<Kunde:0x2bd06a8 @name="Simpson",
    @vorname="Homer">]
#<Kunde:0x2bd0590 @name="Tier",
    @vorname="das">
true
```

Beispiel *VerySimpleSet*

- Sie sollen eine Klasse ***VerySimpleSet*** implementieren, eine Menge, in die sie nur Elemente einfügen können.
- ***VerySimpleSet*** ist eine Menge, d.h. es dürfen nicht zwei gleiche Objekte eingefügt werden.
- Wenn wir Elemente mit ***add*** hinzufügen, dann prüfen wir, ob das Element bereits enthalten ist. Wenn ja, dann fügen wir es nicht ein.
- Wir erzeugen zwei gleiche aber nicht identische ***Grundfarben*** Objekte und fügen diese in ein ***VerySimpleSet*** Objekt (***vss***) ein.
- Dann prüfen wir den Inhalt von ***vss*** .

```
class Grundfarbe
  def initialize(name)
     @name = name
 end
end
rot1 = Grundfarbe.new('rot')
rot2 = Grundfarbe.new('rot')
vss = VerySimpleSet.new()
vss.add(rot1)
vss.add(rot2)
p vss
```

- **FEHLER! *vss*** enthält zwei gleiche Grundfarben Objekte.
- **Was haben wir falsch gemacht?**

Beispiel *VerySimpleSet*

```ruby
class VerySimpleSet
  def initialize()
    @my_set = Array.new()
  end
  # add(Object)
  def add(an_object)
    if ! @my_set.include?(an_object)
      @my_set.add(an_object)
    end
  end
end
rot1 = Grundfarbe.new('rot')
rot2 = Grundfarbe.new('rot')
vss = VerySimpleSet.new()
vss.add(rot1)
vss.add(rot2)
p vss
```

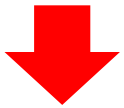
```ruby
class Grundfarbe
  def initialize(name)
     @name = name
 end
end
```

```
#<VerySimpleSet:0x2b51c04
@my_set=[#<Grundfarbe:0x2b51c40
@name="rot">, #<Grundfarbe:0x2b51c18
@name="rot">]>
```

Beispiel *VerySimpleSet*

- ***Arrays*** prüfen auf ***==*** Gleichheit von Objekten, wenn sie Objekte suchen (z.B. ***include?***) oder löschen.
- Wir haben vergessen, die **==** Gleichheit für Grundfarben zu definieren,
- d.h, wir haben die **==** Methode von ***Object*** nicht überschrieben.
- Daher wird beim Einfügen der Grundfarben in ***VerySimpleSet*** auf Identität geprüft.
- ***rot1*** und ***rot2*** sind nicht identisch. Das sehen wir an den unterschiedlichen Objekt Kennungen.

```
rot1 = Grundfarbe.new('rot')
rot2 = Grundfarbe.new('rot')
vss = VerySimpleSet.new()
vss.add(rot1)
vss.add(rot2)
p vss
```



```
#<VerySimpleSet:0x2b51c04
@my_set=[#<Grundfarbe:0x2b51c40
@name="rot">, #<Grundfarbe:0x2b51c18
@name="rot">]>
```

Beispiel *VerySimpleSet*

- Holen wir die Implementierung von **==** für Grundfarben nach, dann verhält sich ***VerySimpleSet*** wie erwartet und enthält nur eine rote Grundfarbe.

```
class Grundfarbe
  attr_reader :name
  protected :name
  def initialize(name)
    @name = name
  end
  def ==(other)
    return false if other.nil?
    return true if
         self.equal?(other)
    return (self.name == other.name)
  end
end
```

```
rot1 = Grundfarbe.new('rot')
rot2 = Grundfarbe.new('rot')
vss = VerySimpleSet.new()
vss.add(rot1)
vss.add(rot2)
p vss
```

```
#<VerySimpleSet:0x2bd0784
    @my_set=[#<Grundfarbe:0x2bd0e78
    @name="rot">]>
```

== Gleichheit : Wir fassen soweit zusammen

Arrays Leistungen

- ***Array*** verwendet ***==*** Gleichheit, um Objekte in dem ***Array*** aufzufinden.
- Nahezu alle Klassen des Ruby Standard-API's implementieren daher die Methode ***==***.

Vertrag für eigene Klassen

- Wird ***==*** in der eigenen Klasse nicht überschrieben, dann wird der ***Identitätsvergleich*** in ***Object*** verwendet.
- Eigene Klassen müssen daher immer die Methode ***==*** überschreiben, sonst kann nicht nach inhaltsgleichen Objekten gesucht werden. Objekte sind nicht mehr auffindbar.

Set verwendet *eql?* und *hash*

- Sie werden einwenden, warum eine eigene Klasse für Mengen, wenn es die Klasse ***Set*** bereits gibt.
- Überredet, der Quelltext rechts macht genau dieses und fügt zwei gleiche rote Grundfarben (***rot1, rot2***) in den Set ***grundfarben*** ein.
- Obwohl wir die ***==*** Gleichheit definiert haben, enthält ***grundfarben*** jetzt wieder zwei rote Grundfarben.
- **FEHLER: Was haben wir falsch gemacht?**

```
rot1 = Grundfarbe.new('rot')
rot2 = Grundfarbe.new('rot')
grundfarben = Set.new
grundfarben.add(rot1)
grundfarben.add(rot2)
p grundfarben
```

```
#<Set: {#<Grundfarbe:0x2bc5b2c
    @name="rot">,
    #<Grundfarbe:0x2bc620c
    @name="rot">}>
```

Set verwendet *eql?* und *hash*

- ***Set*** nutzt ***eql?*** Gleichheit und ***hash***, um Elemente im Set zu suchen.
- Wir haben vergessen, die ***eql?*** Gleichheit für Grundfarben zu definieren,
- d.h, wir haben die ***eql?*** Methode von ***Object*** nicht überschrieben.
- Daher werden beim Einfügen der Grundfarben in ***Set*** die Elemente auf Identität geprüft.
- ***rot1*** und ***rot2*** sind nicht identisch. Das sehen wir an den unterschiedlichen Objekt Kennungen.
- Die Implementierung der ***eql?*** Methode werden wir jetzt nachholen.
- Zuvor schreiben wir eine Definition von ***eql?*** auf und vergleichen diese mit deren Schwester ***==***.

eql? und *==* Gleichheit

Objekte *o1* und *o2* sind *eql?* gleich

- wenn ***o1*** und ***o2*** identisch sind ***(o1.equal?(o2)).***
- oder:
  - wenn ***o2*** nicht ***nil*** ist
  - und ***o1*** und ***o2*** vom selben Typ sind.
  - und **alle relevanten** Eigenschaften (Werte der Instanzvariablen) von ***o1*** und ***o2*** wechselseitig ***eql?*** gleich sind.

Objekte *o1* und *o2* sind *==* gleich

- wenn ***o1*** und ***o2*** identisch sind ***(o1.equal?(o2)).***
- oder:
  - wenn ***o2*** nicht ***nil*** ist
  - und **alle relevanten** Eigenschaften (Werte der Instanzvariablen) von ***o1*** und ***o2*** wechselseitig ***==*** gleich sind.

Set verwendet *eql?* und *hash*

- Holen wir die Definition von ***eql?*** für Grundfarben nach, dann verhält sich das Beispiel **immer noch fehlerhaft**.
- Die Menge ***grundfarben*** enthält noch immer zwei rote Grundfarben-Objekte. (siehe nächste Folie)
- **Frage: Was haben wir jetzt wieder falsch gemacht?**

```
class Grundfarbe
  attr_reader :name
  protected :name
  def initialize(name)
    @name = name
  end

  def eql?(other)
    return false if other.nil?
    return true if
         self.equal?(other)
    return false if
         self.class != other.class
    return
     (self.name.eql?( other.name))
  end

end
```

Set verwendet *eql?* und *hash*

```
rot1 = Grundfarbe.new('rot')
rot2 = Grundfarbe.new('rot')
grundfarben = Set.new()
grundfarben.add(rot1)
grundfarben.add(rot2)
p grundfarben
```

```
#<Set: {#<Grundfarbe:0x2bc620c @name="rot">, #<Grundfarbe:0x2bc56b8
@name="rot">}>
```

Das Grundfarbenbeispiel mit *Hash*

- Bevor wir der Antwort auf den Grund gehen, schauen wir uns an, was passiert, wenn wir zwei **gleiche** aber nicht identische rote Grundfarben als Schlüssel in einem ***Hash*** verwenden.
- Wir fügen unter ***rot1*** einen kurzen Text ein und wollen diesen Text über ***rot2*** nachschauen.
- Keine Chance, der Zugriff mit dem Schlüssel ***rot2*** liefert ***nil***.
- **Fehler: Aber warum?**

```
rot1 = Grundfarbe.new('rot')
rot2 = Grundfarbe.new('rot')
gfh = Hash.new()
gfh[rot1] = "rote Farbe"
puts gfh[rot2]
```

```
nil
```

Das Grundfarbenbeispiel mit *Hash*

- ***Hash*** verwendet für die Suche nach Schlüsseln im Hash ***eql?*** und ***hash***.
- ***Hash*** und ***Set*** stimmen in diesem Verhalten überein, da ***Set*** intern einen ***Hash*** für die Verwaltung der Objekte in der Menge verwendet.
- Da wir ***eql?*** richtig geschrieben haben, kann der Fehler nur in einer fehlerhaften ***hash*** Methode von ***Grundfarbe*** liegen.
- Und so verhält es sich.
- Da wir ***hash*** nicht implementiert haben, wird die Definition von ***Object*** verwendet.
- In ***Object*** haben nicht identische Objekte unterschiedliche ***hash*** Werte.
- Im Beispiel haben wir zwei ***eql?*** gleiche rote Grundfarben Objekte mit unterschiedlichen ***hash***-Werte.

Die Verwendung von *hash* und *eql?* in *Hash*

- ***Hash*** zerlegt mit Hilfe des ***hash***-Wertes von Objektes die Schlüssel in Teilmengen. In diesen Teilmengen stehen alle Schlüssel mit demselben ***hash***-Wert.

- Wenn Hash einen Schlüssel sucht, dann wird zuerst die Teilmenge der Schlüssel berechnet und dann in dieser Teilmenge mit ***eql?*** nach dem angefragten Schlüssel gesucht.

- Da ***rot1*** und ***rot2*** auf nicht identische Objekte zeigen, haben sie einen unterschiedlichen ***hash*** Wert. Zu ***rot2.hash*** gibt es keinen Eintrag.

- Der zweite Schritt, in der Teilmenge mit ***eql?*** zu suchen, wird nie durchgeführt und daher der Schlüssel (***rot2***) nicht gefunden.

- **Daher muss immer folgende Regel gelten:**

  o1.eql?(o2) => o1.hash == o2.hash

- Diese Bedingung ist in der vorliegenden Implementierung für Grundfarben-Objekte verletzt, wenn ***o1*** und ***o2*** ***nicht*** identisch sind.

- Im Beispiel:

  rot1.eql?(rot2) **UND** ***rot1.hash != rot2.hash***

Das Grundfarbenbeispiel mit *Hash*

- Im Grundfarbenbeispiel galt vorher:

  rot1.eql?(rot2) && rot1.hash != rot2.hash

- Nach dem Überschreiben der ***hash-***Methode können wir Grundfarben als Schlüssel in Hashes korrekt verwenden.

```
class Grundfarbe
  def eql?(other)
     ...
  end
  def hash
    @name.hash
  end
end

rot1 = Grundfarbe.new('rot')
rot2 = Grundfarbe.new('rot')
gfh = Hash.new()
gfh[rot1] = "rote Farbe"
puts gfh[rot2]
```

```
rote Farbe
```

Das Grundfarbenbeispiel mit *Set*

- Mit dem Überschreiben der ***hash-*** Methode von Objekt erhalten wir auch das korrekte Verhalten in ***Set***.
- In ***grundfarben*** ist anschließend nur ein rotes Farben Objekt.

```
rot1 = Grundfarbe.new('rot')
rot2 = Grundfarbe.new('rot')
grundfarben = Set.new()
grundfarben.add(rot1)
grundfarben.add(rot2)
p grundfarben
```

```
#<Set: {#<Grundfarbe:0x2b518e4
@name="rot">}>
```

Wozu brauchen wir *hash* Codes?

- **Für den schnellen Zugriff auf Objekte in Hash und Set.**
- Denken Sie an das Telefonbuch Deutschlands und starten die Suche nach einem Eintrag.
- Im schlimmsten Fall müssen alle Einträge (> 60 Mio) durchsucht werden.
- Das ist zu teuer (zeitaufwendig).
- Daher gruppiert man Sektionen eines Telefonbuches zu Listen und definiert für diese Sektion einen eindeutigen Schlüssel (***hash***).
- Der Zugriff erfolgt zweistufig:
  - zuerst wird die Sektion über den Hashwert des Schlüssels ermittelt
  - dann in der Sektion der angefragte Schlüssel.
- ***hash*** berechnet einen Zahlenwert (den ***Hashcode***) für ein Objekt, mit dem dieses Objekt schneller auffindbar wird.
- Verwendet wird der Hashcode von Ruby in der Klasse ***Hash*** und ***Set*** um Objekte aufzufinden.
  - Beim schlüsselbasierten Zugriff auf einen Hash muss zuerst der passende Schlüssel gefunden werden.
  - Mit Hilfe des Hashcodes wird diese Suche effizienter, da die Menge der zu vergleichenden Objekte auf die Menge der Objekte mit gleichem Hashcode reduziert wird.

Wozu brauchen wir *hash* Codes?

- Wir wollen die Schlüssel des Telefonbuchs nach den Anfangsbuchstaben der Nachnamen der Einträge zerlegen.
- Dann haben Sie die Sektionen ***A-X***.
- Alle Personen, deren Namen mit ***"A"*** beginnen, liegen in der Sektion mit der Nummer ***"A".hash*** usw..
- Dazu schreiben wir für die ***Person*** eine ***hash*** Methode, die den Hashwert nur aus dem ersten Zeichen des Nachnamens berechnet.

Das Telefonbuch-Beispiel

- Wir modellieren das Telefonbuch als ***Hash***.
- Das Telefonbuch ordnet Personen-Objek-ten Telefonnummern (Zeichenketten) zu.
- ***Person*** hat eine Instanzvariable ***@adresse***, die ein Adressobjekt referenziert.
- Da Personen Schlüssel im Hash sind, müssen wir für Personen die Methoden ***eql?*** und ***hash*** implementieren.
- Da wir die Schlüsselmenge anhand der Anfangsbuchstaben der Namen zerlegen, wird ***hash*** nur in Abhängigkeit von der Instanzvariable ***@name*** definiert.
- **Prüfung**: Gilt dann

  p1.eql?(p2) => p1.hash == p2.hash

```
class Person
  ...

  def eql?(other)
    return false if other.nil?
    return true if self.equal?(other)
    return false if self.class !=
          other.class
    return [name,vorname,adresse].eql?
     ([other.name,
      other.vorname,other.adresse])
  end

  def hash
    return @name[0,1].hash
                    if @name.size > 0
    return 0
  end
end
```

eql? muss für alle Klassen der referenzierten Objekte definiert sein

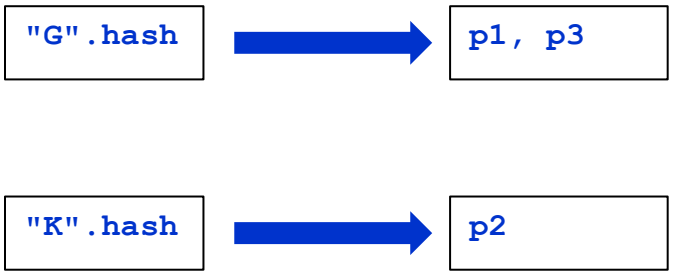
- Da ***Person eql?*** auf allen Instanzvariablen definiert, auch auf ***@adresse,*** müssen wir ***eql?*** auch für Adressen implementieren.
- In unserem Beispiel benötigen wir für Adresse keine Methode ***hash***, da wir Adresse nicht als Schlüssel im Hash oder als Elemente in Sets verwalten wollen.
- **Aber merke:** Sobald für eine Klasse ***eql?*** definiert ist, muss auch ***hash*** implementiert sein.

```
class Adresse
  attr_reader :strasse, :nr, :plz, :ort
  protected :strasse, :nr, :plz, :ort
  def
    initialize(strasse="",nr="",plz="",
    ort="")
     @strasse = strasse
     @nr = nr
     @plz = plz
     @ort = ort
  end

  def eql?(other)
    return false  if other.nil?
    return true if self.equal?(other)
    return false if self.class !=
                                other.class
    return
    [@strasse,@nr,@plz,@ort].eql?(
          [other.strasse,other.nr,
           other.plz,other.ort])
  end
end
```

Personen als Schlüssel für Hashes

- Tragen wir nun untern den Schlüsseln ***p1, p2, p3*** in den ***Hash telefonbuch*** Telefonnummern ein, dann werden die Schlüssel ***p1*** und ***p3*** in der internen Tabelle unter ***"G".hash*** und ***p2*** unter ***"K".hash*** abgelegt.

```
telefonbuch = Hash.new()
a = Adresse.new()
p1 = Person.new("Gross", "Felix", a)
p2 = Person.new("Klein", "Max", a)
p3 = Person.new("Gross", "Max", a)
p4 = Person.new("Gross", "Felix", a)

telefonbuch =  { p1 => "040-23342",
  p2 => "040-678594",
  p3 => "040-8542301"}
```



Personen als Schlüssel für Hashes

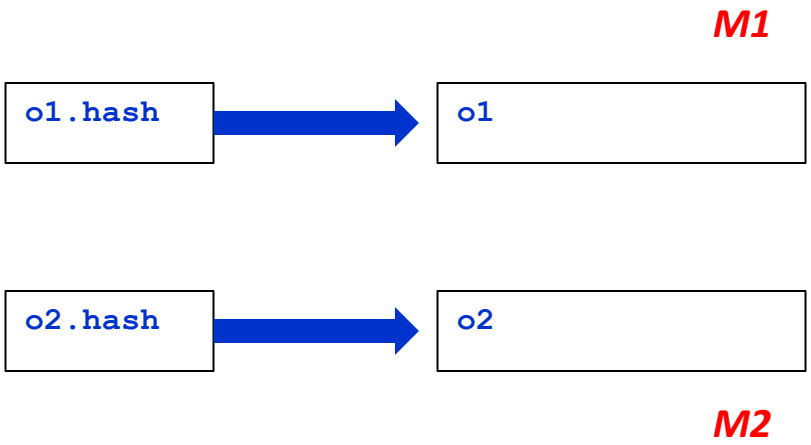
- Beim Zugriff ***telefonbuch[p1]*** wird mit ***p1.hash*** der Schlüssel für die interne Tabelle berechnet. Dieser ist ***"G".hash .***
- Der Hashwert wird in der internen Tabelle gesucht .
- In unserem Fall ist unter ***"G".hash*** die Liste [***p1,p3***] abgelegt.
- In [***p1,p3***] wird mit ***eql?*** nach ***p1*** gesucht.

```
telefonbuch = Hash.new()
a = Adresse.new()
p1 = Person.new("Gross", "Felix", a)
p2 = Person.new("Klein", "Max", a)
p3 = Person.new("Gross", "Max", a)
p4 = Person.new("Gross", "Felix", a)

telefonbuch =  { p1 => "040-23342",
  p2 => "040-678594",
  p3 => "040-8542301"}
```

eql? Gleichheit und hash gehören zusammen

- Objekte, die ***eql?*** gleich sind, müssen den gleichen **Hashcode** (***Object.hash***) haben.

➔ Es muss immer gelten:
o1.eql?(o2) => o1.hash == o2.hash



- Was passiert, wenn
o1.eql?(o2) => o1.hash == o2.hash *nicht gilt*, also
o1.eql?(o2) *und* ***o1.hash != o2.hash***

- **Dramatisches:**
  - → Für ***eql?*** gleiche Schlüsselobjekte ***o1*** und ***o2*** werden separate Listen ***M1*** und ***M2*** in der internen Hashtabelle erzeugt.
  - → mit dem ***hash*** von ***o1*** kann ***o2*** nicht gefunden werden und umgekehrt.

eql? Gleichheit und *hash* gehören zusammen

- Was passiert, wenn ***o1.hash == o2.hash*** und ***!(o1.eql?(o2))*** *?*
- **Nichts Dramatisches:**

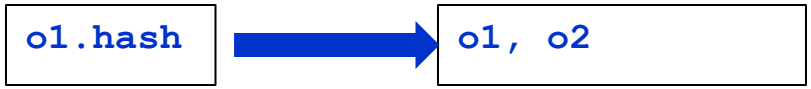
  → ***o1*** und ***o2*** stehen in der gleichen internen Hashtabelle.

  → Das Auswerten der Treffermenge ***M1*** für die Schlüssel ist evtl. etwas teurer, da alle Objekte der Treffermenge mit ***eql?*** verglichen werden müssen.

  → Auf jeden Fall werden alle zu ***o1*** oder ***o2*** ***eql?*** gleichen Schlüssel gefunden.

  → es gibt immer nur einen Eintrag für ***eql?*** gleiche Objekte in den internen Hashtabellen.

M1



Eine alternative *hash*-Methode für Person

- Unsere Zerlegung der Schlüssel eines Telefonbuchs nach Anfangsbuchstaben führt dazu, dass die Teilmengen der Schlüssel unterschiedlich groß werden.
- Bei gleichmäßig verteiltem Zugriff, führt das zu abweichenden Antwortzeiten. Daher gibt es Algorithmen, die für eine gleichmäßige Aufteilung von Schlüsseln optimiert sind.
- Der Quellcode rechts ist eine Implementierung eines Algorithmus zur Berechnung eines Hashcodes.

```
class Person
  attr_reader :name, :vorname, :adresse
  def initialize(name, vorname, adresse)
    ...
  end
  def eql? (other)
   ...
  end
  def hash
    prime = 31
    result = 1
    result = prime * result + ((@adresse
    == nil) ? 0 : @adresse.hash())
    result = prime * result + ((@name ==
    nil) ? 0 : @name.hash())
    result = prime * result + (@vorname
    == nil) ? 0 : @vorname.hash()
    return result
  end
end
```

hash Methode für Adresse

Da in dieser Implementierung von Person auch der Hashcode von Adresse verwendet wird, **müssen** wir jetzt auch in der Klasse Adresse eine Methode *hash* implementieren.

```
class Adresse
...
  def eql?(other)
    return false  if other.nil?
    return true if self.equal?(other)
    return false if self.class != other.class
    return
    [@strasse,@nr,@plz,@ort].eql?([other.strasse,other.nr,other.plz,other.ort])
  end
  def hash
    prime = 31
    result = 1
    result = prime * result + (@strasse == nil?)? 0: @strasse.hash()
    result = prime * result + (@nr == nil) ? 0 : @nr.hash()
    result = prime * result + (@plz == nil) ? 0 : @plz.hash()
    result = prime * result + (@ort == nil) ? 0 : @ort.hash()
    return result
  end
end
```

Übungen

- **Ü-12-b-1:** Was würde passieren, wenn wir mit der alternativen Implementierung von ***hash*** in Person die Implementierung von ***hash*** in der Klasse ***Adresse*** auslassen? Geben Sie ein Beispiel, das diesen Effekt demonstriert!

- **Ü-12-b-2**: Schreiben Sie bitte für die Klasse ***Stack*** die Methoden ***eql?*** und ***hash***!

- **Ü-12-b-3:** Sie wollen Objekte eigener Klassen in **Arrays** verwenden. Welche Methode aus ***Object*** muss Ihre Klasse überschreiben?

- **Ü-12-b-4:** Sie wollen Objekte eigener Klassen in **Sets** verwenden. Welche Methoden aus ***Object*** muss Ihre Klasse jetzt überschreiben?

eql? –Gleichheit und *hash* : Vertrag zwischen *Hash Set* und Klassen

- ***Set*** verwendet ***eql?*** und ***hash***, um Objekte im ***Set*** aufzufinden.
- ***Hash*** verwendet ***eql?*** und ***hash***, um Schlüssel aufzufinden.
- Nahezu alle Klassen des Ruby Standard-Bibliothek implementieren ***eql?*** und ***hash***

- Wird ***eql?*** in der eigenen Klasse nicht überschrieben, dann wird ***eql?*** aus ***Object*** verwendet (***Identitätsvergleich***).
- In eigenen Klassen muss daher immer die Methode ***eql?*** überschrieben werden.
- Wird ***hash*** in eigenen Klassen nicht überschrieben, dann haben nicht identische Objekte unterschiedliche ***hash*** Werte (-> ***Object. hash***)
- Wird ***eql?*** überschrieben muss auch ***hash*** überschrieben werden, damit die Bedingung

  o1.eql?(o2) => o1.hash == o2.hash immer erfüllt ist.

Probleme mit der vorläufigen == Definition

- Abschließend wollen wir die vorläufige **==** Definition am Beispiel der Klasse ***Person*** auf Fehlerquellen untersuchen.
  - Liefert **==** in allen Fällen einen booleschen Wert?
  - Erfüllt diese Definition von **==** in allen Fällen die Symmetrieeigenschaft?
- Dazu nehmen wir eine Vereinfachung vor und verwenden für Adressen nur Zeichenketten und keine komplexen Objekte.

```
class Person
  attr_reader :name, :vorname, :adresse
  def initialize(name, vorname, adresse)
    @name = name
    @vorname = vorname
    @adresse = adresse
  end

  def ==(other)
    return false if other.nil?
    return true if self.equal?(other)
    return [name,vorname,adresse] ==
    [other.name,
    other.vorname,other.adresse]
  end
end
```

Probleme mit der vorläufigen == Definition

- Wir wissen nicht, ob ***other*** von der gleichen Klasse stammt wie ***self,*** setzen aber voraus, dass ***other*** die gleichen Methoden versteht wie ***self*** . Das führt zu Laufzeitfehlern, wenn wir z.B. ein ***Person*** gegen ein Array auf ***==*** prüfen.

- Mit ***eql?*** tritt dieses Problem nicht auf, da ***eql?*** vor dem Aufruf von Methoden auf PM1/PT***other*** immer auf Klassengleichheit prüft.

```
ary = [1,2,3]
p1 = Person.new("Rudi","Ratlos",
    "Nirgendwo")
p1.eql?(ary)
p1==ary
```

```
false
D:/haw/vorlesungen/rubypr1/sose10/works
    pace/V9-
    ObjektidentitaetUndGleichheit/probl
    eme == vorlaeufige definition/Perso
    n.rb:12:in `==': undefined method
    `name' for [1, 2, 3]:Array
    (NoMethodError)
from
    D:/haw/vorlesungen/rubypr1/sose10/w
    orkspace/V9-
    ObjektidentitaetUndGleichheit/probl
    eme == vorlaeufige definition/test_
    == vorlaeufig.rb:6
```

Probleme mit der vorläufigen == Definition

- Wenn ***other*** von einer Subklasse unserer eigenen Klasse stammt, dann überprüfen wir nur eine Teilmenge der Eigenschaften auf ***==.***
- Das Ergebnis: eine **nicht symmetrische Gleichheitsrelation**.
- Um dies zu demonstrieren lassen wir die Klasse Student von Person ableiten.
- In der Implementierung von ***==*** delegiert Student auf das ***==*** der Superklasse ***Person***.

```
class Student < Person
  def
   initialize(name,vorname,adresse,m
   atnr)
    super(name,vorname,adresse)
    @matnr = matnr
  end
  def ==(other)
    return false unless other
    return true if
   self.equal?(other)
    return false unless super
    return (self.matnr ==
   other.matnr)
  end
end
```

Probleme mit der vorläufigen == Definition

- Wir erzeugen eine Person ***p1*** und einen Studenten ***s1*** mit gleichem Namen, Vornamen und gleicher Adresse wie ***p1*** und vergleichen ***p1*** mit ***s1.*** Das Ergebnis: ***true***.

- Anschließend vergleichen wir ***s1*** mit ***p1*** . Das Ergebnis: Ein Laufzeitfehler, das ***p1*** die Methode ***matnr*** nicht versteht.

```
p1 = Person.new("Rudi","Ratlos",
    "Nirgendwo")
s1 = Student.new("Rudi","Ratlos",
    "Nirgendwo",45455455)
p p1==s1
p s1==p1
```

```
true
D:/haw/vorlesungen/rubypr1/sose10/works
    pace/V9-
    ObjektidentitaetUndGleichheit/probl
    eme == vorlaeufige definition/Stude
    nt.rb:13:in `==': undefined method
    `matnr' for #<Person:0x2bcf30c>
    (NoMethodError)
from
    D:/haw/vorlesungen/rubypr1/sose10/w
    orkspace/V9-
    ObjektidentitaetUndGleichheit/probl
    eme == vorlaeufige definition/test
    == vorlaeufig.rb:12
```

Probleme mit der vorläufigen == Definition

- Wir erzeugen eine Person ***p1*** und einen Studenten ***s1*** mit gleichem Namen, Vornamen und gleicher Adresse wie ***p1*** und vergleichen ***p1*** mit ***s1.*** Das Ergebnis: ***true***.
- Wir fügen diese in ein ***Array*** ein und fragen, ob ***s1*** enthalten ist. (-> ***true***).
- Dann erfragen wir den ***index*** von ***s1*** und erhalten ***0***.
- Das Objekt an Position ***0*** ist allerdings eine ***Person*** und kein ***Student***.
- **Fehler wegen Verletzung der Symmetrieeigenschaft in der Implementierung von ==.**

```
p1 = Person.new("Rudi","Ratlos",
   "Nirgendwo")
s1 = Student.new("Rudi","Ratlos",
   "Nirgendwo",45455455)
puts "p1 == s1 #{p1 == s1}"
s_array = [p1,s1]
puts s_array.include?(s1)
i_s1 = s_array.index(s1)
s3 = s_array[i_s1]
puts s3
```

```
p1 == s1 true
true
#<Person:0x2bcf000>
```

Erster Lösungsvorschlag für Probleme mit == Definition

- Die Ursache für die genannten Probleme liegen hier in dem nicht vorhandenen Klassenvergleich, der in ***eql?*** gefordert, aber in ***==*** nicht notwendig ist.
- Wir müssen daher beim Vergleich zweier Objekte mit ***==*** sicherstellen, dass sie in allen relevanten Instanzvariablen übereinstimmen.
- Dazu müssen beide Objekte die gleichen Instanzvariablen haben und die Werte der Instanzvariablen müssen gleich sein.
- Der Quellcode rechts ist eine Implementierung für ***==***, die alle Eigenschaften einer Gleichheitsrelation erfüllt.

```
class Person
  def ==(other)
    return true if self.equal?(other)
    # instance_variables liefert alle
    # Instanzvariablen
    self_ivs =
    self.instance_variables().sort()
    other_ivs =
    other.instance_variables().sort()
    # gleiche Instanzvariablen?
    return false if ! (self_ivs ==
    other_ivs)
    # Wert aller Instanzvariablen beider
    # Objekte gleich?
    self_ivs.each do |self_iv|
      return false  if
    !(self.instance_variable_get(self_iv)
    ==
    other.instance_variable_get(self_iv))
    end
    return true
  end
end
```

Zweiter Lösungsvorschlag für Probleme mit == Definition

- Da es Situationen gibt, in denen nicht alle Instanzvariablen zur Prüfung der Gleichheit verwendet werden dürfen, ist es in den meisten Fällen einfacher in der == Methode die Klassenprüfung auf zu nehmen.
- Die zuvor beschriebenen Fehler können damit beseitigt werden.

```
class Person
  attr_reader :name, :vorname, :adresse

  def ==(other)
    return false if other.nil?
    return true if self.equal?(other)
    return false if self.class !=
    other.class
    return [name,vorname,adresse] ==
    [other.name,
      other.vorname,other.adresse]
  end
end
```

Nicht alle Instanzvariablen gehen in die Definition von Gleichheit ein

- ***Kreis*** hat vier Instanzvariablen, 3 davon beschreiben die Eigenschaften eines Kreises, die 4‘te ***@counter*** ist eine technische Größe, um unterschiedlichen Objekten ein Nummer zuzuordnen.

- Wenn wir die allgemeine Definition für ***==*** in Kreis verwenden, geht die Variable ***@counter*** in den Vergleich ein.

```
class Kreis
  attr_reader :x,:y,:radius
  protected :x,:y,:radius
  @@counter =0
  def initialize(x,y,radius)
    @x=x
    @y=y
    @radius = radius
    @counter = @@counter
    @@counter +=1
  end

  def ==(other)
    return true if self.equal?(other)
    self_ivs =
    self.instance_variables().sort()
    other_ivs =
    other.instance_variables().sort()
    return false if ! (self_ivs == other_ivs)
    self_ivs.each do |self_iv|
      return false  if
    !(self.instance_variable_get(self_iv) ==
      other.instance_variable_get(self_iv))
    end
    return true
  end
end
```

Nicht alle Instanzvariablen gehen in die Definition von Gleichheit ein

- Wenn wir die allgemeine Definition für ***==*** in Kreis verwenden, geht die Variable ***@counter*** in den Vergleich ein.
- Dann liefert der Vergleich 2‘er Kreise ***k1*** und ***k2*** mit gleichen ***x,y*** Koordinaten und gleichem Radius ***false***.
- Von außen betrachtet würden wir sagen ***k1*** und ***k2*** sind gleich.

```
require
    "probleme_==_vorlaeufige_definition/Kr
    eis"

k1 = Kreis.new(1,1,2)
k2 = Kreis.new(1,1,2)

puts k1 == k2
```

```
false
```

Nicht alle Instanzvariablen gehen in den Definition von Gleichheit ein

- In diesem Fall müssen wir in der Implementierung von ***==*** darauf achten, dass die internen technischen Zustands-größen nicht in den Vergleich eingehen.

- Ersetzen wir die ***==*** Definition durch jene rechts, dann liefert ***k1 == k2 true***.

```
class Kreis
  attr_reader :x,:y,:radius
  protected :x,:y,:radius
  @@counter =0
  def initialize(x,y,radius)
    @x=x
    @y=y
    @radius = radius
    @counter = @@counter
    @@counter +=1
  end

  def ==(other)
    return false if other.nil?()
    return true if self.equal?(other)
    return false if self.class !=
   other.class
    return [@x,@y,@radius]==
   [other.x,other.y,other.radius]
  end

end
```

Zusammenfassung

- **Objektidentität** ist in Programmiersprachen dann gegeben, wenn zwei Variablen auf dasselbe Objekt zeigen.
- **Objektgleichheit** ist in Programmiersprachen dann gegeben, wenn zwei Objekte gleichen Inhalt haben.
- Ruby prüft Identität mit ***equal?***
- Ruby prüft Gleichheit in Arrays mit ***==***, in Hash und Set werden ***eql?*** und ***hash*** verwendet.
- Folgende Implikation muss immer gelten:
  o1.eql?(o2) => o1.hash == o2.hash
- Die Implementierung einer Gleichheitsrelation muss die Eigenschaften **reflexiv**, **symmetrisch** und **transitiv** haben. Ist eine der Eigenschaften verletzt, dann werden Objekte nicht korrekt wiedergefunden, oder der Vergleich liefert einen Laufzeitfehler.
- Die ***==*** Methode muss immer auf Symmetrieeigenschaften überprüft werden, insbesondere im Zusammenhang mit Vererbungshierarchien.
- Es gehen nicht immer alle Instanzvariablen in die Definition von ***==*** ein.