=

`.equals(...)`

=

Structural Equality

Value Equality

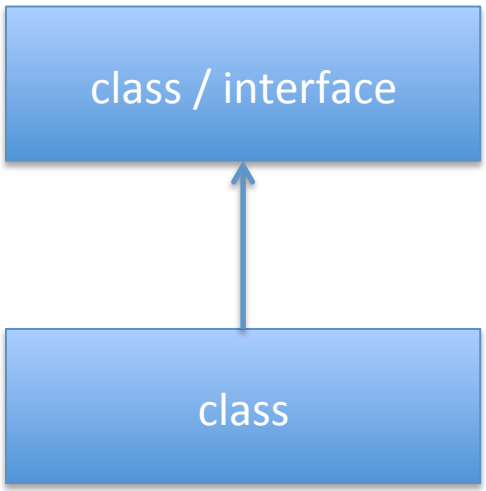
= ?

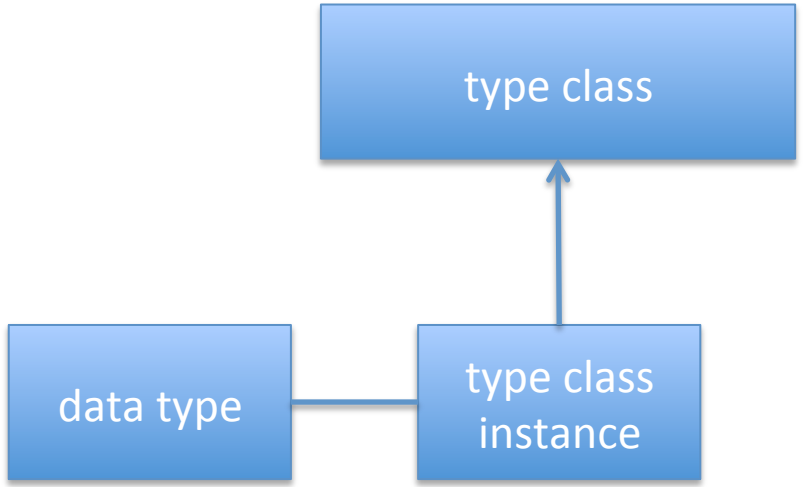
- Same value
- Conditional statements
- Testing
- Substitution
- Identity
- Searching
- Uniqueness

```java
package com.techtangents.eq.examples.foo;

/* In java.lang:
class Object {
  boolean equals(Object o);
}
*/

public class Foo /* extends Object */ {

  public final String s;

  public Foo(final String s) {
    this.s = s;
  }

  @Override
  public boolean equals(Object o) {
    return o instanceof Foo && s.equals(((Foo)o).s);
  }

  public static void main(String[] _) {
    Foo x = new Foo("x");
    Foo y = new Foo("y");
    boolean b = x.equals(y);
  }
}
```

```haskell
module Foo where

{- from Prelude:

class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool

-}

data Foo = Foo String

instance Eq Foo where
  (Foo x) == (Foo y) = x == y

x = Foo "x"
y = Foo "y"
b = x == y
```


Java


class / interface
class


type class
data type
type class instance

```java
package com.techtangents.eq.examples.foo;

/* In java.lang:
class Object {
  boolean equals(Object o);
}
*/

public class Foo /* extends Object */ {

  public final String s;

  public Foo(final String s) {
    this.s = s;
  }

  @Override
  public boolean equals(Object o) {
    return o instanceof Foo && s.equals(((Foo)o).s);
  }

  public static void main(String[] _) {
    Foo x = new Foo("x");
    Foo y = new Foo("y");
    boolean b = x.equals(y);
  }
}
```

```haskell
module Foo where

{- from Prelude:

class Eq a where
  (==) :: a -> a -> Bool
  (/=) :: a -> a -> Bool

-}

data Foo = Foo String

instance Eq Foo where
  (Foo x) == (Foo y) = x == y

x = Foo "x"
y = Foo "y"
b = x == y
```

```
Foo("x") == 72;
-- compile error
```


```
new Foo("x").equals(72);
-- false
```

```
new Foo("x").equals(Color.purple);
-- false
```

```
new Foo("x").equals(
new com.sun.java.swing.plaf.nimbus.
InternalFrameInternalFrameTitlePane
InternalFrameTitlePaneMaximizeButton
Painter(null, 3)
);
-- false
```

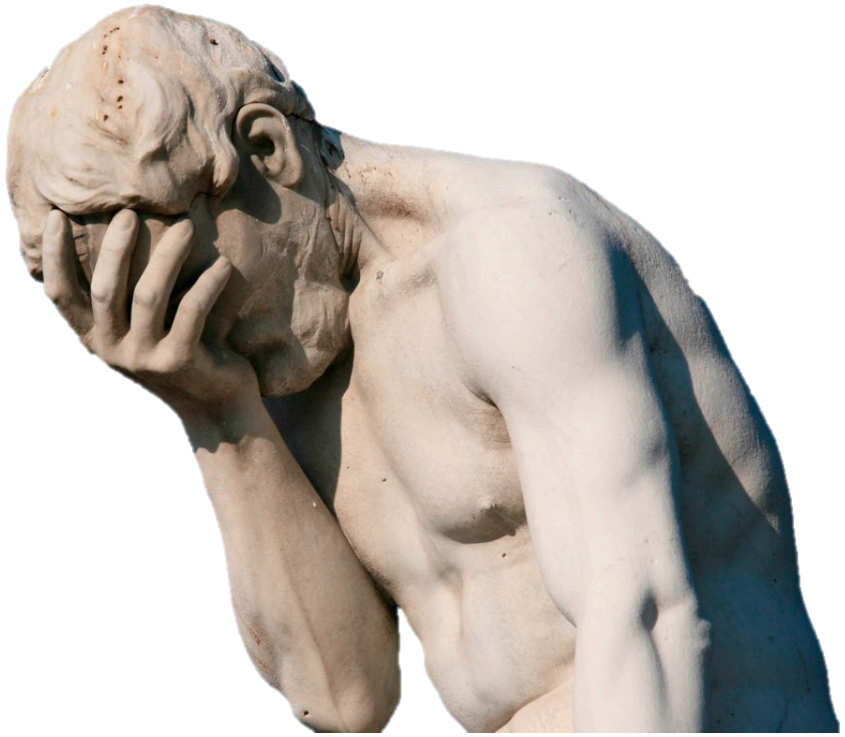
= on values of
different types
is silly

we already know
they're different!

"... in general, the aim [of a type system] is to prevent operations expecting a certain kind of value from being used with values for which that operation does not make sense (logic errors);"

- https://en.wikipedia.org/wiki/Type_system

```
public class Object {
    ....
    public boolean equals(Object obj) {
      ...
    }
    ....
}
```



```java
public class Pair<A, B> {
  public final A a;
  public final B b;

  private Pair(final A a, final B b) {
    this.a = a;
    this.b = b;
  }

  public static <A, B> Pair<A, B> pair(final A a, final B b) {
    return new Pair<A, B>(a, b);
  }
}
```

```java
public class Pair<A, B> {
  public final A a;
  public final B b;

  private Pair(final A a, final B b) {
    this.a = a;
    this.b = b;
  }

  public static <A, B> Pair<A, B> pair(final A a, final B b) {
    return new Pair<A, B>(a, b);
  }

  @Override
  public boolean equals(final Object other) {
    return other instanceof Pair && eq((Pair<A, B>)other);
  }

  private boolean eq(final Pair<A, B> other) {
    return a.equals(other.a) && b.equals(other.b);
  }
}
```

```java
public class Pair<A, B> {
  public final A a;
  public final B b;

  private Pair(final A a, final B b) {
    this.a = a;
    this.b = b;
  }

  public static <A, B> Pair<A, B> pair(final A a, final B b) {
    return new Pair<A, B>(a, b);
  }

  @Override
  public boolean equals(final Object other) {
    return other instanceof Pair && eq((Pair<A, B>)other);
  }

  private boolean eq(final Pair<A, B> other) {
    return a.equals(other.a) && b.equals(other.b);
  }
}
```


Mister Maker
G
Let's Try Something

```
public interface InstanceEq<T> {
  boolean eq(final T other);
}
```

```java
public class Pair<A, B> implements InstanceEq<Pair<A, B>>{
  public final A a;
  public final B b;

  private Pair(final A a, final B b) {
    this.a = a;
    this.b = b;
  }

  public static <A, B> Pair<A, B> pair(final A a, final B b) {
    return new Pair<A, B>(a, b);
  }

  @Override
  public boolean eq(final Pair<A, B> other) {
    return a.equals(other.a) && b.equals(other.b);
  }
}
```

```
public class Cheese implements Eq<Chalk> {
...
}
```

http://licehunter.files.wordpress.com/2010/06/head_scratch_final_2.jpg

```java
public class Pair<A, B> implements InstanceEq<Pair<A, B>>{
  public final A a;
  public final B b;

  private Pair(final A a, final B b) {
    this.a = a;
    this.b = b;
  }

  public static <A, B> Pair<A, B> pair(final A a, final B b) {
    return new Pair<A, B>(a, b);
  }

  @Override
  public boolean eq(final Pair<A, B> other) {
    return a.equals(other.a) && b.equals(other.b);
  }
}
```

= on parameterized types

requires

= on type parameters

```java
public class Pair<A extends InstanceEq<A>, B extends InstanceEq<B>>
  implements InstanceEq<Pair<A, B>>{

  public final A a;
  public final B b;

  private Pair(final A a, final B b) {
    this.a = a;
    this.b = b;
  }

  public static <A extends InstanceEq<A>, B extends InstanceEq<B>>
    Pair<A, B> pair(final A a, final B b) {
    return new Pair<A, B>(a, b);
  }

  @Override
  public boolean eq(final Pair<A, B> other) {
    return a.eq(other.a) && b.eq(other.b);
  }
}
```

~~String implements InstanceEq<String>~~

```
pair("a", "b");
pair(person("mary"), person("bob"));
```

```java
package com.techtangents.eq.examples;

import com.techtangents.eq.InstanceEq;

public class Stringq implements InstanceEq<Stringq> {
  public final String s;

  private Stringq(final String s) {
    this.s = s;
  }

  public static Stringq stringq(final String s) {
    return new Stringq(s);
  }

  @Override
  public boolean eq(final Stringq other) {
    return s.equals(other.s);
  }

  public static void main(final String[] args) {
    stringq("x").eq(stringq("y"));
  }
}
```

```java
package com.techtangents.eq.examples.stringq;

import com.techtangents.eq.InstanceEq;

public class EqAdapter<T> implements InstanceEq<EqAdapter<T>> {
  public final T t;

  private EqAdapter(final T t) {
    this.t = t;
  }

  public static <T> EqAdapter<T> eqAdapter(final T t) {
    return new EqAdapter<T>(t);
  }

  @Override
  public boolean eq(final EqAdapter<T> other) {
    return t.equals(other.t);
  }

  public static void main(final String[] args) {
    eqAdapter("x").eq(eqAdapter("y"));
    eqAdapter(34).eq(eqAdapter(34));
  }
}
```

Do all types have equality?

Integer
String
List*
Maybe*
Tree*
Unit

Void

Stream

IO

Function

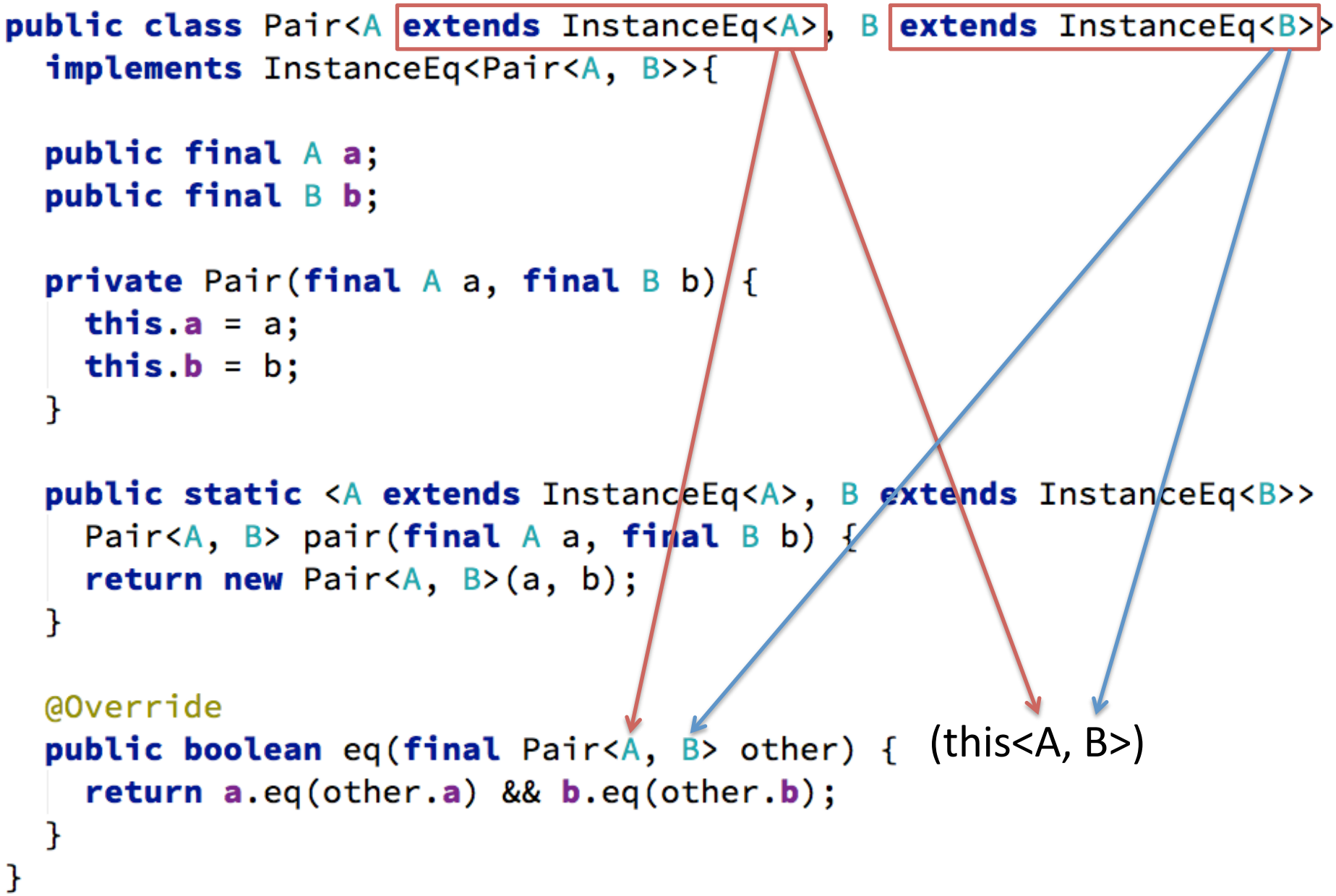
Irrational number

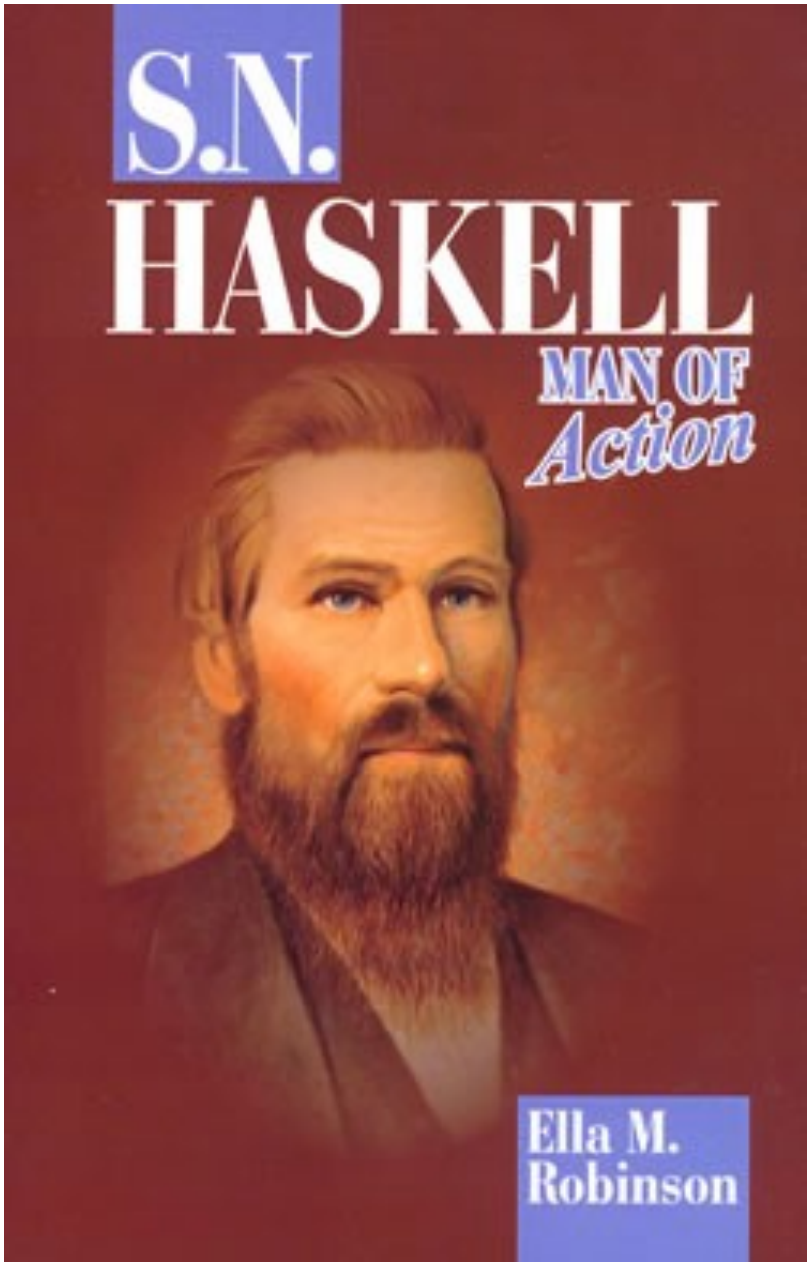
Random number generator

Do all types have equality?

Can all types be in Pairs?

```java
public class Pair<A extends InstanceEq<A>, B extends InstanceEq<B>>
  implements InstanceEq<Pair<A, B>>{

  public final A a;
  public final B b;

  private Pair(final A a, final B b) {
    this.a = a;
    this.b = b;
  }

  public static <A extends InstanceEq<A>, B extends InstanceEq<B>>
    Pair<A, B> pair(final A a, final B b) {
    return new Pair<A, B>(a, b);
  }

  @Override
  public boolean eq(final Pair<A, B> other) {
    return a.eq(other.a) && b.eq(other.b);
  }
}
```


public class Pair<A extends InstanceEq<A>, B extends InstanceEq<B>>
implements InstanceEq<Pair<A, B>>{
public final A a;
public final B b;
private Pair(final A a, final B b) {
this.a = a;
this.b = b;
}
public static <A extends InstanceEq<A>, B extends InstanceEq<B>>
Pair<A, B> pair(final A a, final B b) {
return new Pair<A, B>(a, b);
}
@Override
public boolean eq(final Pair<A, B> other) {
return a.eq(other.a) && b.eq(other.b);
}
}
(this<A, B>)


S.N.
HASKELL
MAN OF
Action
Ella M.
Robinson

```java
public interface Eq<A> {
  boolean eq(A a, A b);
}
```

```haskell
class Eq a where
  (==) :: a -> a -> Bool
```

```java
package com.techtangents.eq.examples.pairs.example5;

import com.techtangents.eq.Eq;

public class Pair<A, B> {
  public final A a;
  public final B b;

  private Pair(final A a, final B b) {
    this.a = a;
    this.b = b;
  }

  public static <A, B> Pair<A, B> pair(final A a, final B b) {
    return new Pair<A, B>(a, b);
  }

  public static <A, B> Eq<Pair<A, B>> pairEq(final Eq<A> eqa, final Eq<B> eqb) {
    return (p1, p2) -> { return eqa.eq(p1.a, p2.a) && eqb.eq(p1.b, p2.b); };
  }
}
```

```java
public class Example {
  public static void main(final String[] _) {
    final Pair<String, Integer> p1 = pair("a", 3);
    final Pair<String, Integer> p2 = pair("a", 3);
    final boolean eq1 = pairEq(stringEq, intEq).eq(p1, p2);
    println("eq1 = " + eq1);

    final Pair<Person, Integer> mary = pair(person("mary"), 33);
    final Pair<Person, Integer> fred = pair(person("fred"), 87);
    final boolean eq2 = pairEq(personEq, intEq).eq(mary, fred);
    println("eq2 = " + eq2);
  }
}
```

```
data Pair a b = Pair a b

instance (Eq a, Eq b) => Eq (Pair a b) where
  (Pair a b) == (Pair a' b') = a == a' && b == b'

q = (Pair "cat" 7) == (Pair "dog" 7)
```

```
(Pair "cat" 7) == (Pair "dog" 7)
```

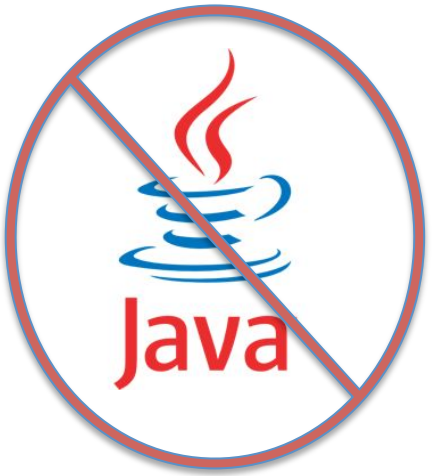

```
pairEq(intEq, stringEq).eq(pair("cat", 7), pair("dog", 7))
```

```
instance (Eq a, Eq b) => Eq (Pair a b) where
  ...
```

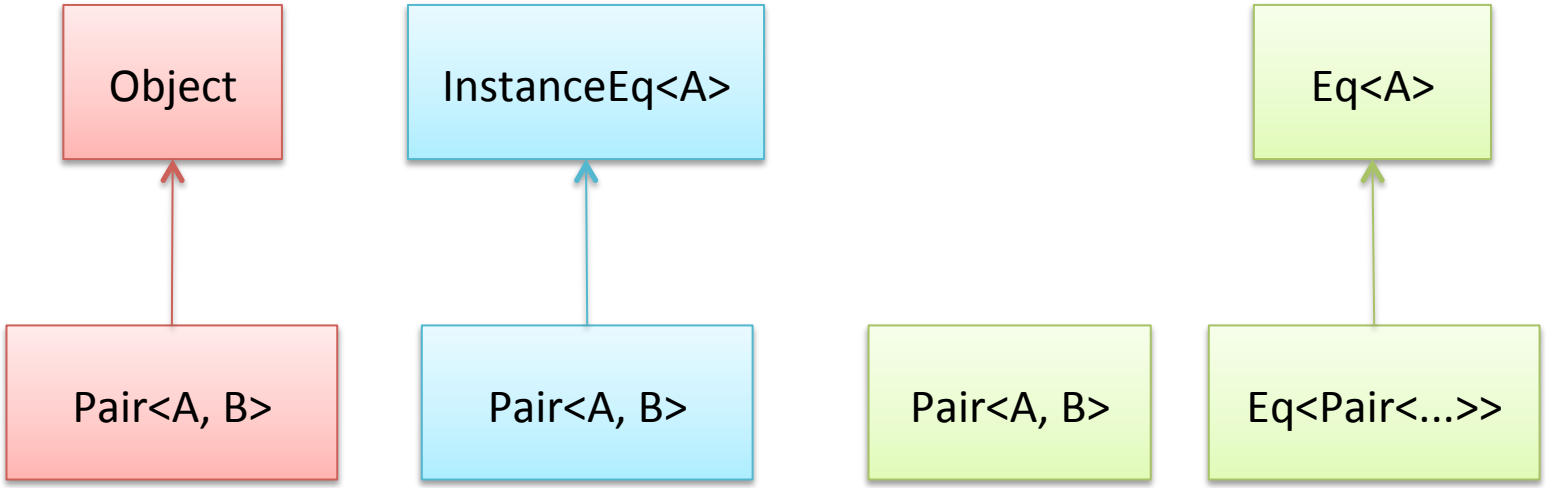
```
static <A, B> Eq<Pair<A, B>>
  pairEq(Eq<A> eqa, Eq<B> eqb) {
...
}
```

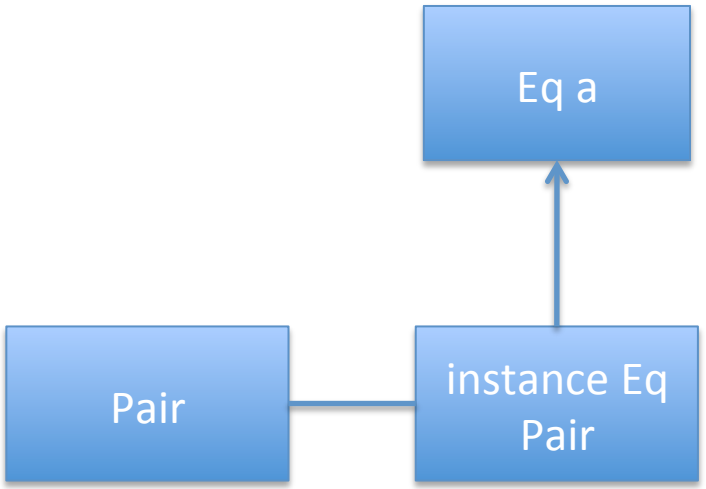
Conditional implementation of interface

```
interface InstanceEq {
  boolean eq(this other)
}

public class Pair<A, B>
  implements InstanceEq
  (when A extends InstanceEq, B extends InstanceEq)
{
    @Override
    eq(Pair<A, B> other) {
    ...
   }
}
```




Java


Object
Pair<A, B>
InstanceEq<A>
Pair<A, B>
Pair<A, B>
Eq<A>
Eq<Pair<...>>


Eq a
Pair
instance Eq
Pair