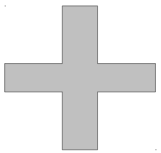
GK SOFTWARE



JDT Embraces Java 8

Stephan Herrmann

Committer:
- JDT
- Object Teams

Eclipse Track @ JavaLand 2014, Brühl, Germany

Berlin
Köln

GK SOFTWARE

Schöneck, Vogtland
St. Ingbert, Saarland

THE RETAIL APPLICATION Company

- POS
- Store Integration
- Mobile Devices
- ...

...

Domain Specific Languages

Java (Server & Client)

Hardware Integration

Eclipse Track @ JavaLand 2014, Brühl, Germany

GK SOFTWARE

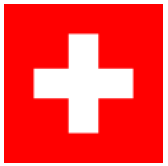
Released March 18th!

- Andy Clement
- Steve Francisco
- Michael Rennie
- Olivier Thomann
- Curtis Windatt

- Walter Harley
- David Williams

- Jesper S. Møller

- Stephan Herrmann



- Dani Megert
- Markus Keller

- Jay Arthanareeswaran
- Anirban Chakarborty
- Manoj Palat
- Shankha Banerjee
- Manju Mathew
- Noopur Gupta
- Deepak Azad
- **Srikanth Sankaran**

http://download.eclipse.org/eclipse/downloads

http://wiki.eclipse.org/JDT/Eclipse_Java_8_Support_For_Kepler

GK SOFTWARE

Eclipse and Java™ 8

- Java 8 features supported:
  - JSR 308 - Type Annotations.
  - JEP 120 - Repeating Annotations.
  - JEP 118 - Method Parameter Reflection.
  - JSR 269 - Pluggable Annotation Processor API
    & javax.lang.model API enhancements for Java 8.
  - JSR 335 – Lambda Expressions
    - Lambda expressions & method/constructor references
    - Support for “code carrying” interface methods
    - Enhanced target typing
      new overload resolution & type inference

GK SOFTWARE

The Engine behind Lambda Expressions

GK SOFTWARE

Prologue: Parser

- Eclipse uses the Jikes Parser Generator
  - requires an LALR(1) grammar
  - generates an automaton
- Java 8 grammar
  - is not LALR(1)
  - rewrite estimated to 10-15 person years
- Rewrite could be avoided
  - a number of advanced tricks
  - unorthodox communication between scanner & parser

GK SOFTWARE

Goal: Brevity

- What we do in Java 7 (instance of an anonymous class):

```
new BinaryOperator<Integer>() {
    @Override
    public Integer apply(Integer i1, Integer i2) {
        return i1+i2;
    }
}
```

- What want in Java 8 (lambda):

```
(i1, i2) -> i1+i2
```

- What can be omitted:
  - the functional interface
  - its generic type arguments
  - the function signature
  - …
- How come this works?

GK SOFTWARE

Goal: Reuse

- Lambdas are typically arguments to generic library functions

```
public static <T, K, U, M extends Map<K, U>>
Collector<T, ?, M> toMap(Function<? super T, ? extends K> keyMapper,
                         Function<? super T, ? extends U> valueMapper,
                         BinaryOperator<U> mergeFunction)
```

- Would you prefer to call:

```
Collectors.<Person,String,Integer,Map<String,Integer>>toMap(..)
```

or:

```
Collectors.toMap(..)
```

?

- Again: omitting “unnecessary” boiler plate
- Similar for instantiation of generic class (Java 7)

```
Collector<String,Integer> coll = new MyCollector<>();
```

- How ***does*** it work?

GK SOFTWARE

Solution: Type Inference

- Present since Java 5 – but completely rewritten
- Let the compiler collect *all available* type information
- Let it deduce the types
  - instantiation of type parameters of
    - generic methods
    - diamond instantiations
    - functional interface
  - lambda parameters
- If inference finds a solution, it is type safe – by definition
- If it doesn't find a solution ... ? ...

GK SOFTWARE

Type Inference Details

- Solving this little example

```
Map<String, Integer> test3(Stream<Person> persons) {
    return persons.collect(Collectors.toMap(
                                p -> p.getLastName(),
                                p -> p.getAge(),
                                (i1, i2) -> i1+i2));
}
```

- Produces these constraints:

GK SOFTWARE

- Solving th

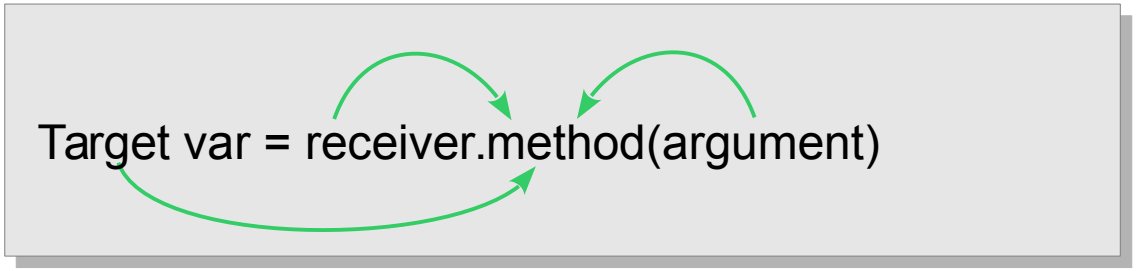
```
Map<Strin
    retur


}
```

- Produces

```
TypeBound  K#3 :> java.lang.String
Dependency K#3 = K#3
TypeBound  K#3 = java.lang.String
TypeBound  K#3 <: java.lang.Object
Dependency R#0 = java.util.Map<K#3,U#4>
Dependency R#0 = java.util.Map<java.lang.String,U#4>
Dependency R#0 = java.util.Map<K#3,java.lang.Integer>
TypeBound  R#0 = java.util.Map<java.lang.String,java.lang.Integer>
TypeBound  R#0 <: java.util.Map<java.lang.String,java.lang.Integer>
TypeBound  R#0 <: java.lang.Object
TypeBound  T#2 :> Person
Dependency T#2 = T#2
TypeBound  T#2 = Person
TypeBound  T#2 <: java.lang.Object
Dependency Map<K#3,U#4>#6 = java.util.Map<K#3,U#4>
Dependency Map<K#3,U#4>#6 = java.util.Map<java.lang.String,U#4>
Dependency Map<K#3,U#4>#6 = java.util.Map<K#3,java.lang.Integer>
TypeBound  Map<K#3,U#4>#6 = java.util.Map<java.lang.String,java.lang.Integer>
Dependency Map<K#3,U#4>#6 = R#0
TypeBound  Map<K#3,U#4>#6 <: java.lang.Object
TypeBound  Map<K#3,U#4>#6 <: java.util.Map<java.lang.String,java.lang.Integer>
TypeBound  A#1 = java.lang.Object
TypeBound  A#1 <: java.lang.Object
TypeBound  U#4 :> java.lang.Integer
TypeBound  U#4 = java.lang.Integer
Dependency U#4 = U#4
TypeBound  U#4 <: java.lang.Object
TypeBound  ?#5 = java.lang.Object
Dependency ?#5 = A#1
TypeBound  ?#5 <: java.lang.Object
```

Type Inference Details

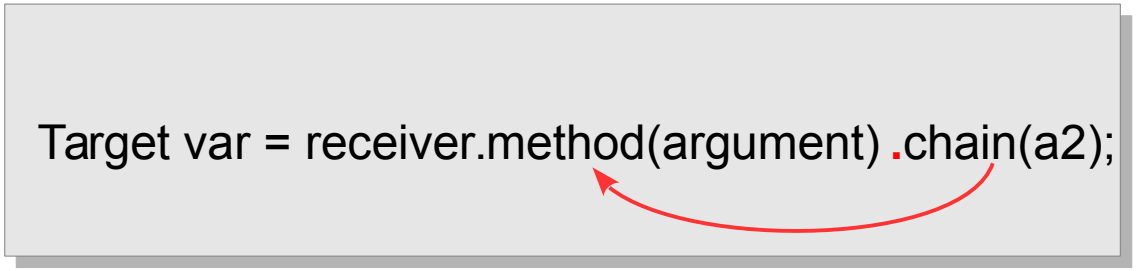
- Information flows:
  - Used for inference:
    - receiver type
    - argument types
    - target type



GK SOFTWARE

Type Inference Details

- Information flows:
  - Used for inference:
    - receiver type
    - argument types
    - target type
  - Never
    - what's right of the next dot



GK SOFTWARE

Type Inference Details

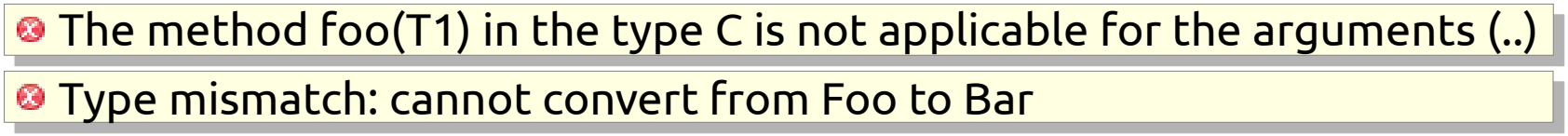
- Information flows:
  - Used for inference:
    - receiver type
    - argument types
    - target type
  - Never
    - what's right of the next dot
  - Nested inference
    - expressions nested as call arguments
    - combine inner and outer to one inference

```
Map<String, Integer> test3(Stream<Person> ps) {
    return ps
        .collect(
            Collectors.toMap(
               p -> p.getLastName(),
               p -> p.getAge(),
               (i1, i2) -> i1+i2));
}
```

GK SOFTWARE

If it fails … ?

- To make sure you know what you're doing:
  - insert explicit types
- There's no guarantee that inference finds every solution
  - *“reduction is not completeness-preserving” (JLS 18.2.)*
- Inference is constraint solving
  - unlike common sense reasoning
  - if it fails, it's **hard** to say, why



    - Better error reporting: future work
- Simplification for users
  - bought by tremendous complexity in the compiler

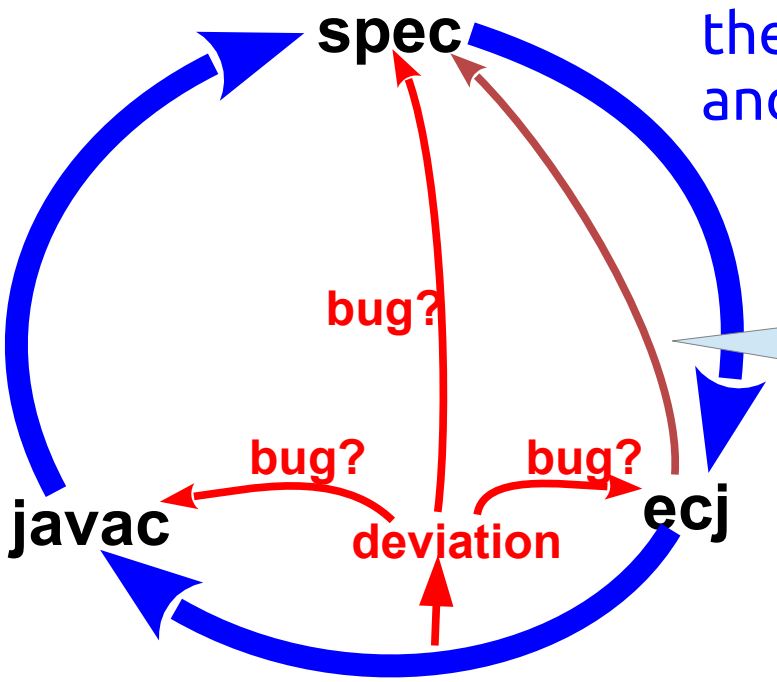
GK SOFTWARE

The Process behind a new Java Feature

GK SOFTWARE

Java Quality Assurance

abstractly capture
the concepts



implement the spec,
the full spec,
and nothing but the spec

spec is not implementable
(incomplete)
(Apr – Sep 2013)

compare behavior

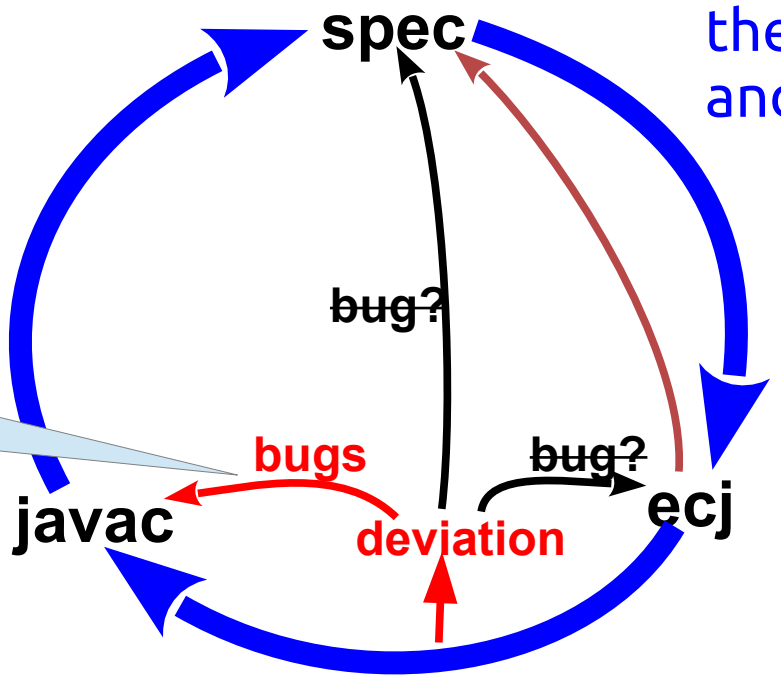
GK SOFTWARE

Java Quality Assurance

implement the spec,
the full spec,
and nothing but the spec

abstractly capture
the concepts

far too risky
to cram late into 8
(Nov 2013)



compare behavior

GK SOFTWARE

Raw Types in Type Inference

- Where are unchecked conversions allowed?
  - List <: List<String> ?
    - spec: NO
    - javac: maybe, sometimes
    - ecj ?
  - raw types
    - supported in Java 5 to allow migration
    - some programs using raw types will eventually be rejected
- Serving two masters – JLS & javac?
  - should we exactly copy javac behavior?
  - is precisely implementing the spec better?
  - competition helps improve quality

GK SOFTWARE

Status Reached

- JLS 8 is more precise than previous versions
  - will be a good judge in case of doubt
- Ecj implementation is very close to the spec
  - bugs can be easily identified and fixed
  - 1 unimplemented branch, if you see:

    Problem detected during type inference: ...

    you win! :)

GK SOFTWARE

- DEMO JDT Support for Lambda
  - Quick assist
    - to / from lambda
    - expression ↔ block
  - Search / Type hierarchy / Refactoring
    - are lambda aware
  - Hover
    - to see the functional type

GK SOFTWARE

Breathing Life into Types

GK SOFTWARE

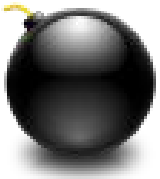
Annotations in more Places

- Java 5: annotate declarations
  - ElementType: packages, classes, fields, methods, locals …
- Java 8: annotate types
  - ElementType.TYPE_USE
  - ElementType.TYPE_PARAMETER

GK SOFTWARE

Why Care About Types?

- Dynamically typed languages
  - anything goes
  - but may fail at runtime
  - e.g.: “method not understood”



```
dog1 = new Dog();
dog1.bark();

dog2 = new Object();
dog2.bark();
```

- Type = Constraints on values

  To *statically* detect anomalies
  - missing capability
  - incompatible assignment
  - undeclared capability

```
Dog dog1 = new Dog();
dog1.bark();

Object dog2 = new Object();
dog2.bark();

Dog dog3 = new Object();
dog3.bark();

Object dog4 = new Dog();
dog4.bark();
```

GK SOFTWARE

Why Care About Types?

- Dynami
  - anyth
  - but m
  - e.g.: "

Annotate source code:
- make assumptions explicit
- convince the compiler that code is safe

```
);

ct();
```

- Type = Constraints on values

  To *statically* detect anomalies
  - missing capability
  - incompatible assignment
  - undeclared capability

```
Dog dog1 = new Dog();
dog1.bark();

Object dog2 = new Object();
dog2.bark();

Dog dog3 = new Object();
dog3.bark();

Object dog4 = new Dog();
dog4.bark();
```

GK SOFTWARE

Why Care About Types?

- Constraint checking avoids errors
  - No Such Method / Field
    - Basic statically typed OO
  - ClassCastException
    - Generics
  - ??Exception
    - SWTException("Invalid thread access")
    - …
    - **NullPointerException**

```
void letemBark(Vector dogs) {
    Dog aDog = (Dog) dogs.get(0);
    aDog.bark();
}
```

GK SOFTWARE

Pluggable Type System

- Make it easier to add new constraints
  - Only one new syntax for all kinds of constraints
- Make it easier to add new type checkers
  - Checker Framework (Michael Ernst – U of Washington)
- Examples
  - @NonNull
  - @Interned equals(== , equals)
  - @Immutable value cannot change (Java 5 ?)
  - @ReadOnly value cannot change *via this reference*
  - @UI code requires to run on the UI thread

GK SOFTWARE

Can't Java 5 Do All This?

```
@Target(ElementType.PARAMETER)
@interface NonNull5 {}

void java5(@NonNull5 String arg);
```

arg is qualified to be non-null

```
@Target(ElementType.METHOD)
@interface NonNull5 {}

@NonNull5 String java5();
```

java5 is qualified to be non-null

```
@Target(ElementType.TYPE_USE)
@interface NonNull8 {}

void java8(@NonNull8 String arg);
```

String is qualified to be non-null

```
@Target(ElementType.TYPE_USE)
@interface NonNull8 {}

@NonNull8 String java8();
```

String is qualified to be non-null

- We've been lying about the method result
  - but we can't lie about everything, e.g.:

```
void printFirstDog(@NonNull List<Dog> dogs) {
    dogs.get(0).bark();
}
```

NPE?

GK SOFTWARE

Annotated Generics

```
void good() {
   @NonNull List<@NonNull String> l1 = new ArrayList<>();
   l1.add("Hello");
   for (String elem : l1)
      System.out.println(elem.toUpperCase());

   @NonNull List<@Nullable String> l2 = new ArrayList<>();
   l2.add(null);
   for (String unknown : l2)
      if (unknown != null)
         System.out.println(unknown.toUpperCase());
}
```

l1 cannot contain **null** elements

l2 can contain **null** elements

GK SOFTWARE

Annotated Generics

```
void bad(List<String> unknown, List<@Nullable String> withNulls) {
   @NonNull List<@NonNull String> l1 = new ArrayList<>();
   l1.add(null);
   String
   if (fir
   l1 = unknown;
   l1 = withNulls;

   String canNull = withNulls.get(0);
   System.out.println(canNull.toUpperCase());
}
```

Null type mismatch: required '@NonNull String' but the provided value is null

GK SOFTWARE

Annotated Generics

```
void bad(List<String> unknown, List<@Nullable String> withNulls) {
   @NonNull List<@NonNull String> l1 = new ArrayList<>();
   l1.add(null);
   String first = l1.get(0);
   if (first == null) return;

   l1 =
   l1 =

   String canNull = withNulls.get(0);
   System.out.println(canNull.toUpperCase());
}
```

Null comparison always yields false:
The variable first cannot be null at this location

GK SOFTWARE

Annotated Generics

```
void bad(List<String> unknown, List<@Nullable String> withNulls) {
   @NonNull List<@NonNull String> l1 = new ArrayList<>();
   l1.add(null);
   String first = l1.get(0);
   if (first == null) return;

   l1 = unknown;
   l1 = 

   Strin
   System.out.println(canNull.toUpperCase());
}
```

Null type safety (type annotations): The expression of type 'List<String>' needs unchecked conversion to conform to '@NonNull List<@NonNull String>'

GK SOFTWARE

Annotated Generics

```
void bad(List<String> unknown, List<@Nullable String> withNulls) {
   @NonNull List<@NonNull String> l1 = new ArrayList<>();
   l1.add(null);
   String first = l1.get(0);
   if (first == null) return;

   l1 = unknown;
   l1 = withNulls;

   Strin
   Syste
}
```

Null type mismatch (type annotations):
required '@NonNull List<@NonNull String>'
but this expression has type 'List<@Nullable String>'

GK SOFTWARE

Annotated Generics

```
void bad(List<String> unknown, List<@Nullable String> withNulls) {
   @NonNull List<@NonNull String> l1 = new ArrayList<>();
   l1.add(null);
   String first = l1.get(0);
   if (first == null) return;

   l1 = unknown;
   l1 = withNulls;

   String canNull = withNulls.get(0);
   System.out.println(canNull.toUpperCase());
}
```

Potential null pointer access:
The variable canNull may be null at this location

GK SOFTWARE

Status

- Null Annotations
  - org.eclipse.jdt.annotation_2.0.0
    - @NonNull, @Nullable specify Target(TYPE_USE)
    - @NonNullByDefault: more fine tuning
- Null Analysis (per compiler option)
  - Nullness is an integral part of the type system
  - Fine tuned defaults only in Luna
  - TODO: Strict checking against type variables

GK SOFTWARE

Status

- Null Annotations
- Null Analysis (per compiler option)
- Planned: @Uninterned
  - Detect accidental comparison using == or !=
- Proposed: @UiEffect, @Ui …
  - by [Colin S. Gordon, Werner Dietl, Michael D. Ernst, and Dan Grossman]
  - SWT: bye, bye, "Invalid thread access"

GK SOFTWARE

Status

- Null Annotations
- Null Analysis (per compiler option)
- Planned: @Uninterned
- Proposed: @UiEffect, @Ui …
- JDT/UI
  - OK: completion, refactoring, etc.
  - TODO: show in hover
  - TODO: update / add more quickfixes

GK SOFTWARE

Status

- Null Annotations
- Null Analysis (per compiler option)
- Planned: @Uninterned
- Proposed: @UiEffect, @Ui …
- JDT/UI

Do You care about Types?

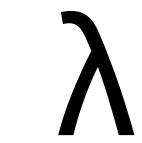
GK SOFTWARE

Breathing Life into Objects (beyond Java)

GK SOFTWARE

Areas Covered

- Algorithms
  - lambda expressions
  - library updates for lambdas
- Types
  - type annotations
- Objects? – Connecting objects to a module / a system?
  - references
    - bad for analyzeability / modularity
  - inheritance
    - doesn't scale
    - rigid



GK SOFTWARE

Introducing Object Teams

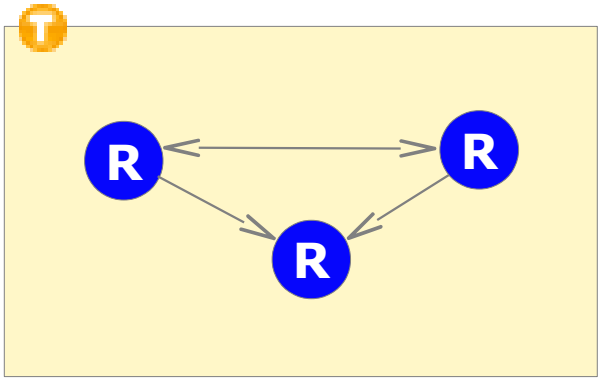
- 2002: Research on Language Design beyond standard OO
  - Object Teams – programming with roles and teams
  - OT/J – extends Java

- 2010: Eclipse Object Teams Project established
  - Outgrown the research niche
  - Mature language (OT/J) & IDE (OTDT)
  - Object Teams Development Tooling written in OT/J

Object Containment

eclipse.org/objectteams

- OO: Message passing among objects



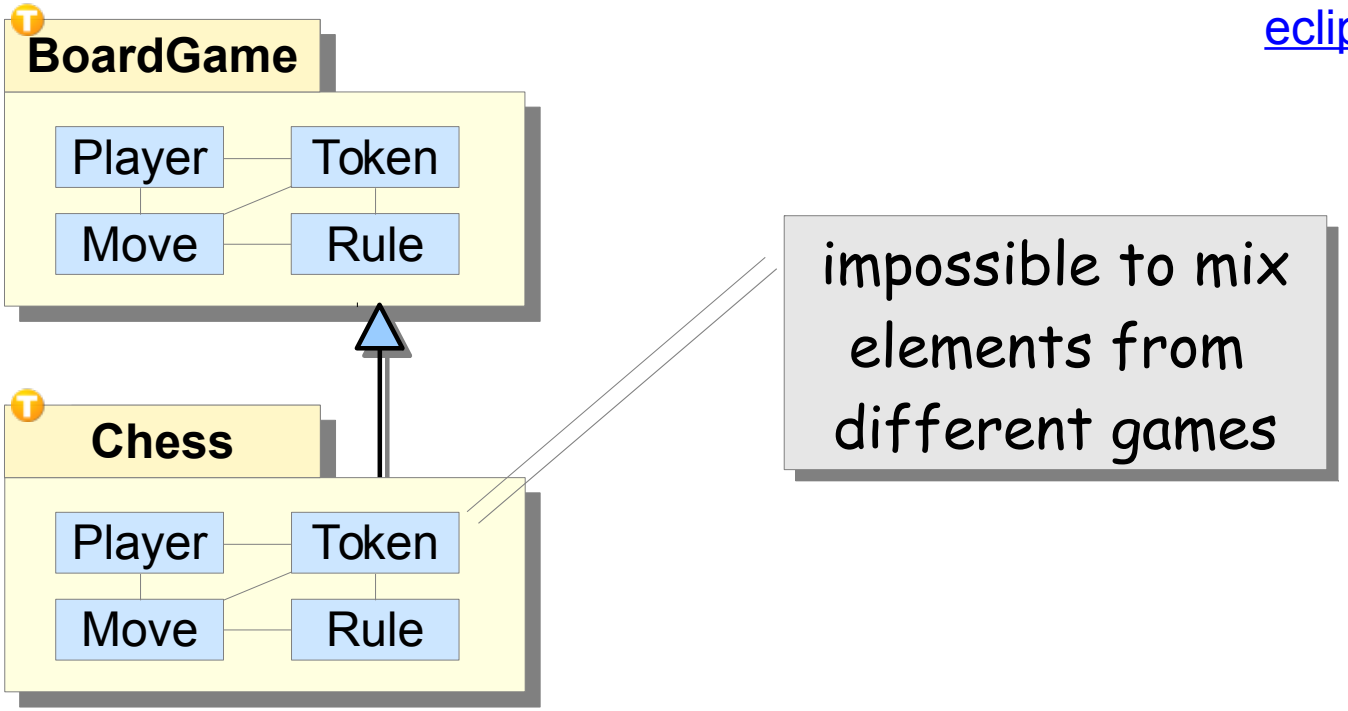
- Team: A group of interacting Roles
  - A team instance **owns** its role instances
  - Team provides a boundary for **isolation**

```
team class Company {

    class Employee {

    }

}
```

GK SOFTWARE

Team Inheritance

eclipse.org/objectteams

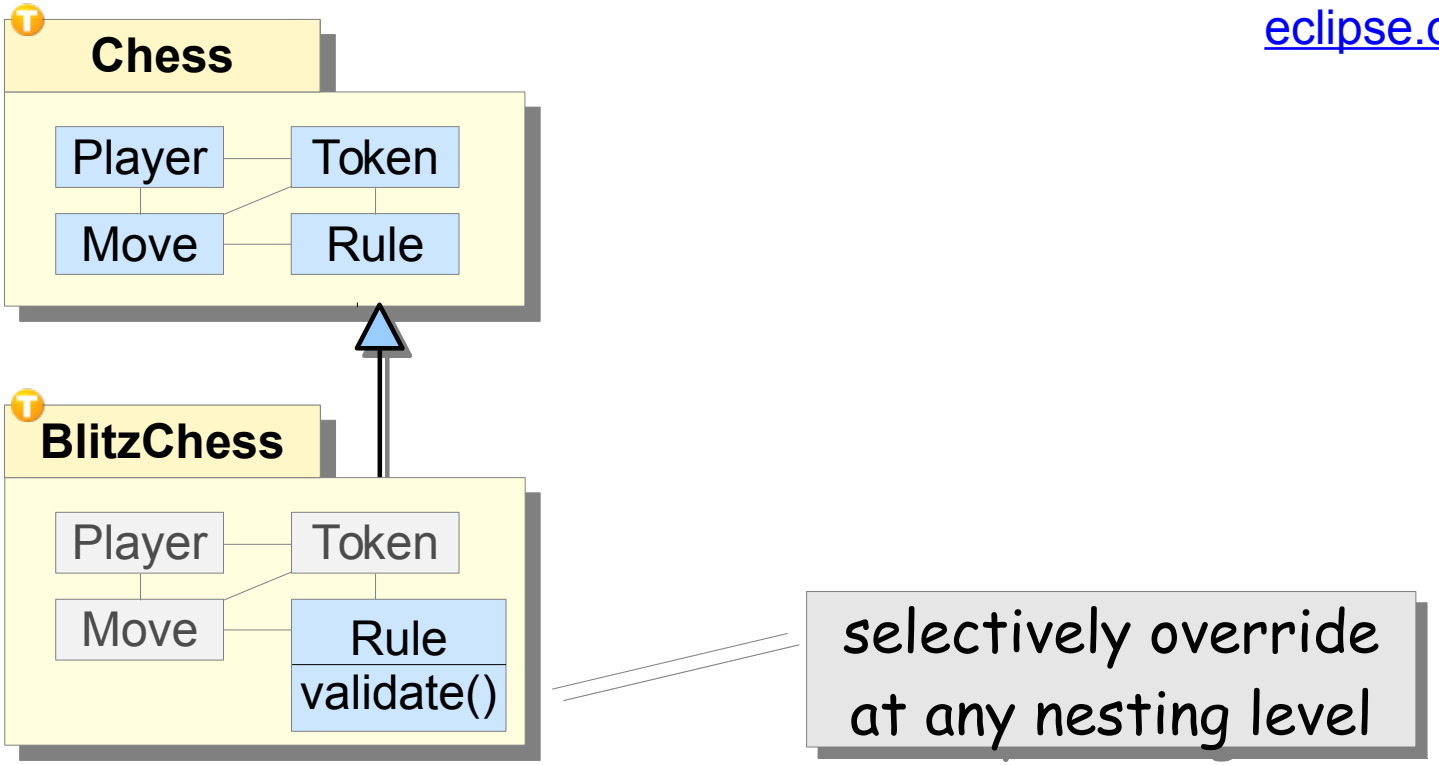


- Members of a team are virtual classes
  - consistent refinement of all members

GK SOFTWARE

Team Inheritance

eclipse.org/objectteams



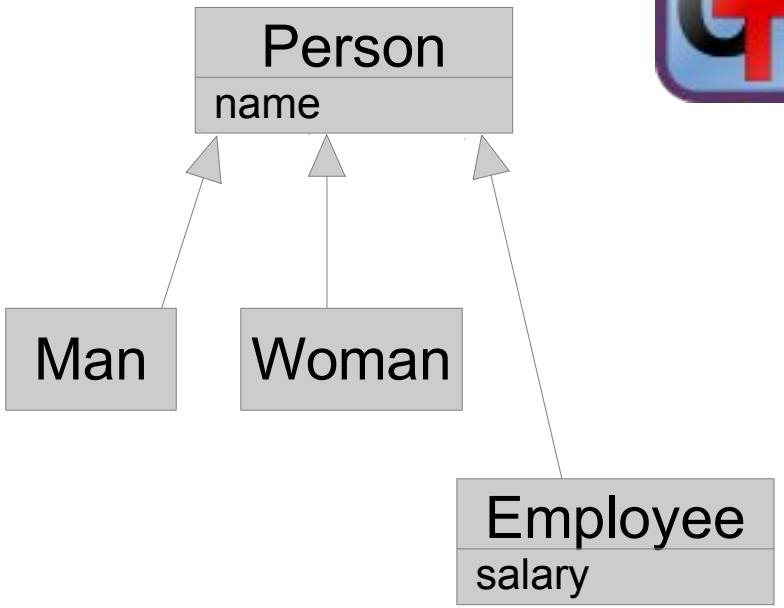
- consistent refinement of all members
- deep overriding
- flexible & modular

Limitations of Inheritance

- **Inheritance** is great, but ...

  A text book example:

  - A man/woman **is a** person, OK
  - An employee **is a** person, OK?
    - Male/female Employees?
    - Born as an employee?
    - Dying when loosing the job?
    - Several jobs, yet only one salary?



Role playing

eclipse.org/objectteams

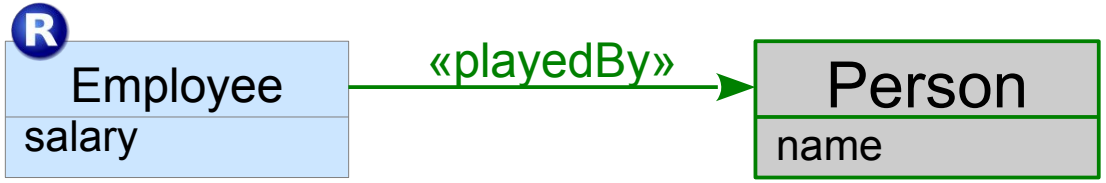
- `playedBy` relationship

joe: Man
name=”joe”

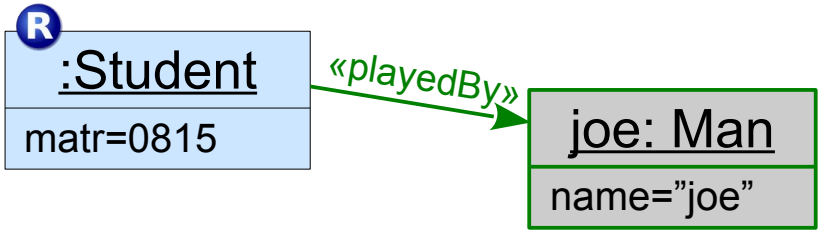
Role playing

eclipse.org/objectteams

- `playedBy` relationship



- Advantages:
  - **Dynamism**:
    roles can come and go
    (same base object)

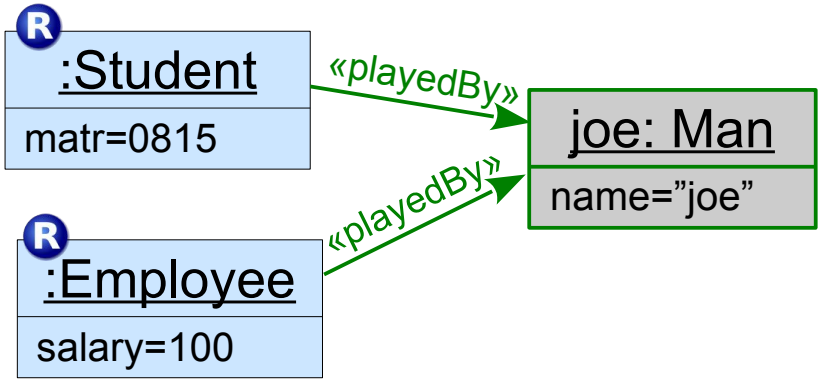


Role playing

eclipse.org/objectteams

- `playedBy` relationship

- Advantages:
  - **Dynamism**:
    roles can come and go
    (same base object)
  - **Multiplicities:**
    one base can play several roles
    (different/same role types)

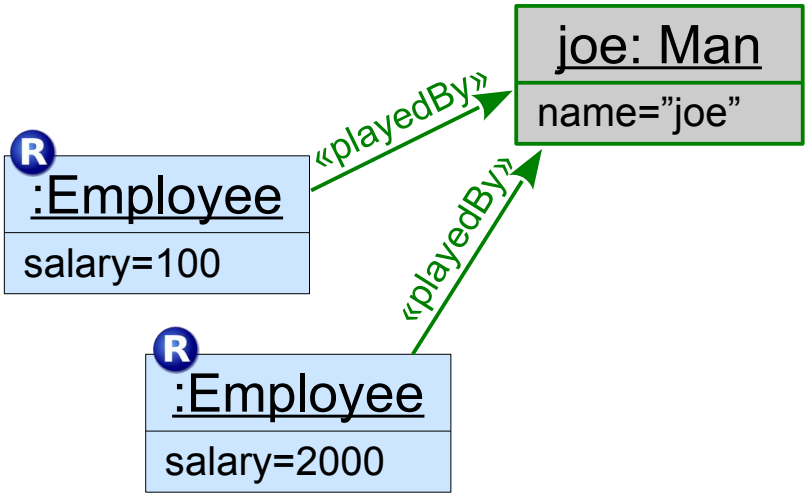


Role playing

eclipse.org/objectteams

- `playedBy` relationship

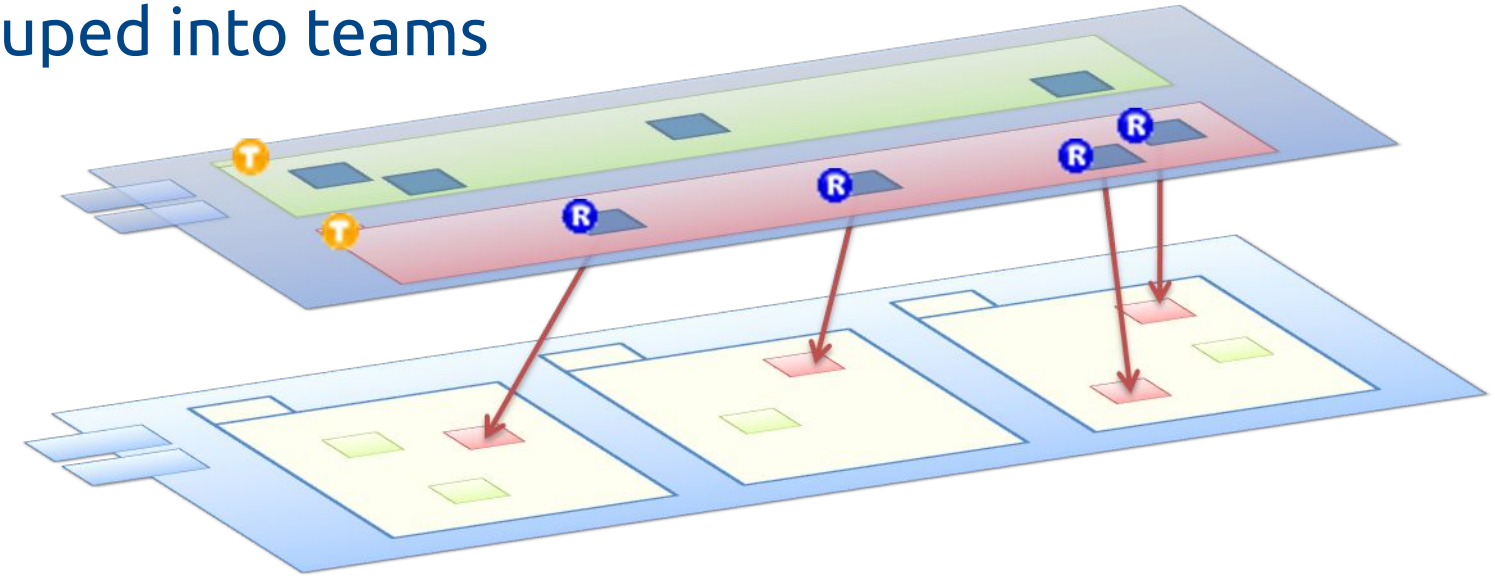
- Advantages:
  - **Dynamism**:
    roles can come and go
    (same base object)
  - **Multiplicities:**
    one base can play several roles
    (different/same role types)



Role playing

eclipse.org/objectteams

- Use role playing for
  - dynamic specialization
  - modes / behavior that should be turn on/off at runtime
  - unanticipated adaptation
- Roles are grouped into teams



GK SOFTWARE

Areas Covered

- Methods & algorithms
  - lambda expressions
  - library updates for lambdas
- Types
  - type annotations
- Connecting objects to a module / a system
  - role containment
  - team inheritance
  - role playing