Code with restraints

Tesla Ice Zhang

About me

Undergraduate junior at Penn State, 2 years of experience in SDE intern

- Sourcebrella (IDE plugin & frontend), PingCAP (TiKV, gRPC, protobuf), JetBrains Research (Arend & intellij-arend & arend-lib)
- IDE & editor development, compiler & typechecker (mostly DTLC)
- (Homotopy) type theory and its constructive interpretations
- Interested in making friends

About this talk

—

I’m gonna talk about type systems and design patterns.

Most code examples will be using simple Java syntax.

Type Systems

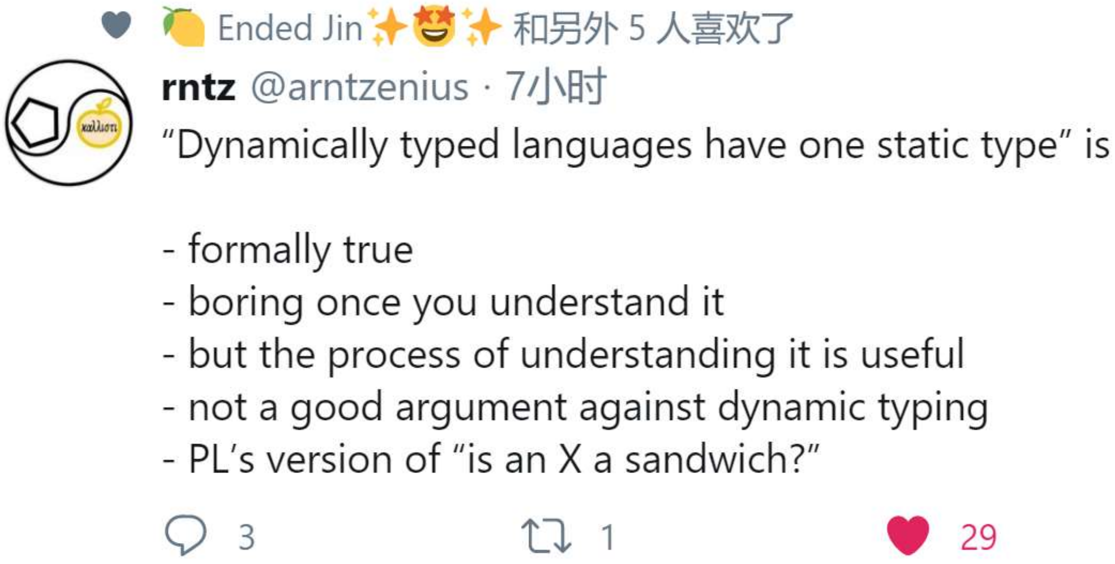
Definition?

Don’t look up Wikipedia.

My definition: a mechanism in the translator/evaluator of a programming language that “checks” your program against a set of constraints constituted of types.

Q: do JavaScript have a type system?

We don’t talk about uni-typed languages’ type systems, because they’re not interesting.



🍋 Ended Jin✨🤩✨ 和另外 5 人喜欢了

rntz @arntzenius · 7小时

“Dynamically typed languages have one static type” is

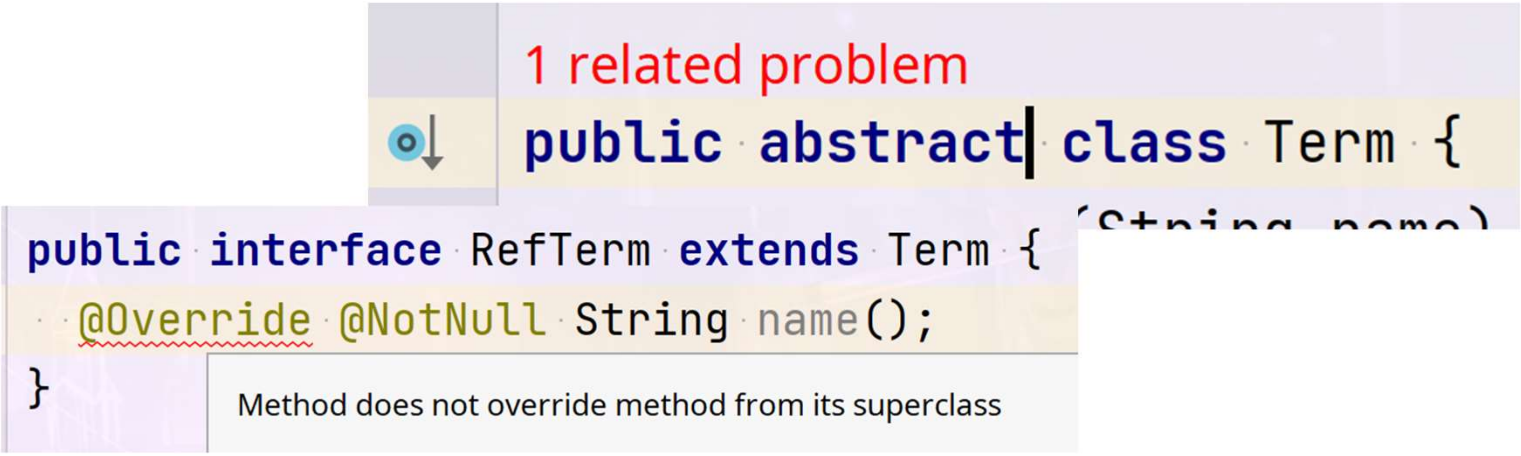
- formally true
- boring once you understand it
- but the process of understanding it is useful
- not a good argument against dynamic typing
- PL’s version of “is an X a sandwich?”

Q: why do we want type systems?

To save us from searching StackOverflow due to simple errors, when you’re using a new language (if Groovy is a static language then it’ll tell you that `Closure<void>` doesn’t have a field “`name`”).

```
def something = {}
something.name = 1
```

To keep us aware of the unrefactored parts of a huge codebase when you’re rewriting some important component of it.



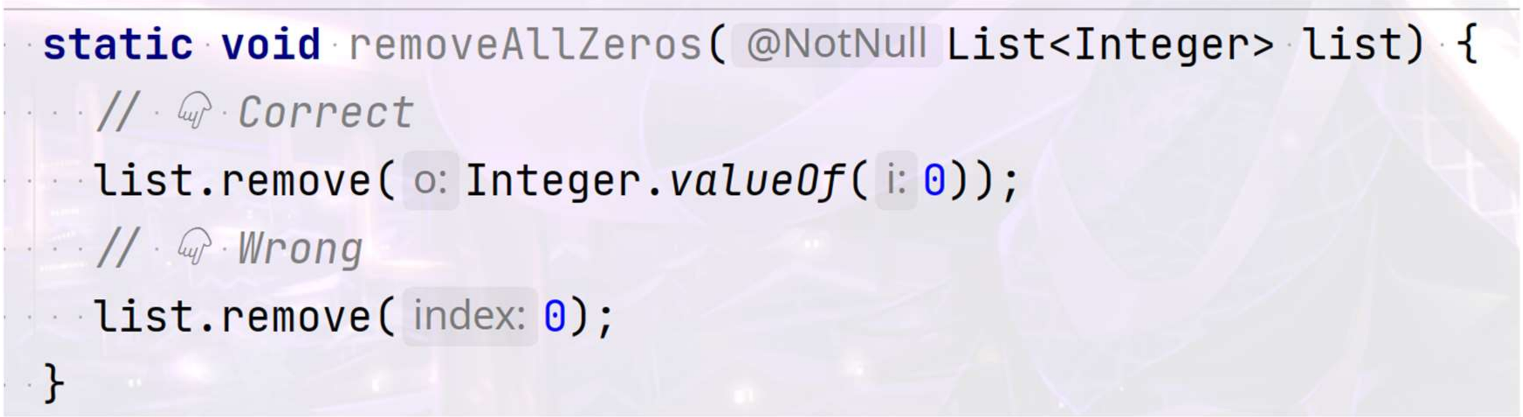
1 related problem

```
public abstract class Term {
                   (String name)
```

```
public interface RefTerm extends Term {
  @Override @NotNull String name();
}
```

Method does not override method from its superclass

Being able to overload a method, though sometimes it can be confusing.



```java
static void removeAllZeros( @NotNull List<Integer> list) {
    // 👇 Correct
    list.remove( o: Integer.valueOf( i: 0));
    // 👇 Wrong
    list.remove( index: 0);
}
```

OOP

In particular, the “subtyping polymorphism”.

Quiz: Is OOP opposite to FP?

OOP

Consider this piece of Java code, it represents an input to a polynomial expression printer.

```java
public interface Term {
  record RefTerm(String name) implements Term {}
  record AddTerm(Term a, Term b) implements Term {}
}
```

OOP

```java
public interface Term {
  record RefTerm(String name) implements Term {}
  record AddTerm(Term a, Term b) implements Term {}
}
```

```java
static void printTerm(StringBuilder builder, Term term) {
  if (term instanceof RefTerm ref) builder.append(ref.name());
  else if (term instanceof AddTerm add) {
    builder.append("(");
    printTerm(builder, add.a());
    builder.append(" + ");
    printTerm(builder, add.b());
    builder.append(")");
  }
}
```

```java
var builder = new StringBuilder();
printTerm(builder, new AddTerm(
  new RefTerm("a"),
  new AddTerm(
    new RefTerm("c"),
    new RefTerm("b"))));
System.out.println(builder);
```

The output is (a + (c + b)). It works!

Now, let’s start using subtraction.

```
public interface Term {
  record RefTerm(String name) implements Term {}
  record AddTerm(Term a, Term b) implements Term {}
  record SubTerm(Term a, Term b) implements Term {}
}

static void printTerm(StringBuilder builder, Term term) {
  if (term instanceof RefTerm ref) builder.append(ref.name());
  else if (term instanceof AddTerm add) {
    builder.append("(");
    printTerm(builder, add.a());
    builder.append(" + ");
    printTerm(builder, add.b());
    builder.append(")");
  }
}
```

```
var builder = new StringBuilder();
printTerm(builder, new AddTerm(
  new RefTerm("a"),
  new SubTerm(
    new RefTerm("c"),
    new RefTerm("b"))));
System.out.println(builder);
```

The output is (a +).

Wait what? I want my (a + (c – b))!

You forgot to handle `SubTerm` in `printTerm`!

I want an error to occur if I forgot to handle a certain subtype of `Term` (and allow me to manually ignore a certain subtype, of course).

OOP

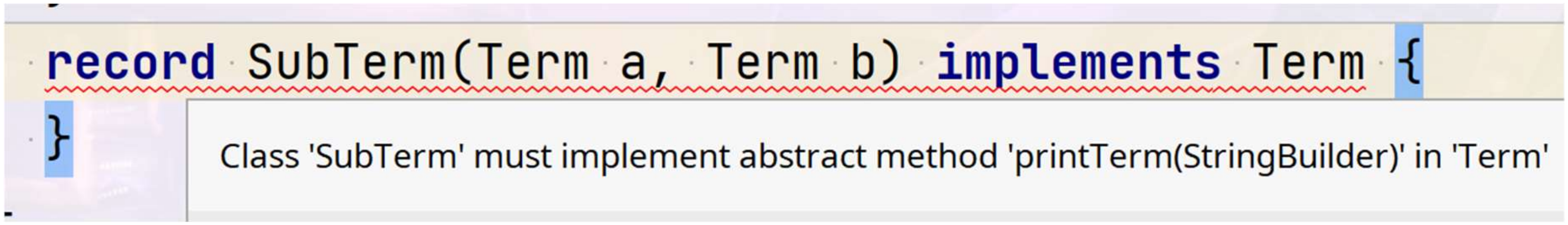
There is subtyping polymorphism, so we can rewrite `printTerm` as a method of `Term`.

```java
public interface Term {
  void printTerm(StringBuilder builder);

  record RefTerm(String name) implements Term {
    @Override
    public void printTerm(StringBuilder builder) {
      builder.append(name);
    }
  }

  record AddTerm(Term a, Term b) implements Term {
    @Override
    public void printTerm(StringBuilder builder) {
      builder.append("(");
      a().printTerm(builder);
      builder.append(" + ");
      b().printTerm(builder);
      builder.append(")");
    }
  }
}
```

And this error will be shown, if you write an empty class `SubTerm`:



```
record SubTerm(Term a, Term b) implements Term {
}
```

Class 'SubTerm' must implement abstract method 'printTerm(StringBuilder)' in 'Term'

Nice!

That’s not the end of the story. Before this change, we can publish `Term` classes as a library, and downstream users can write functions like `printTerm` to *use* `Term`.

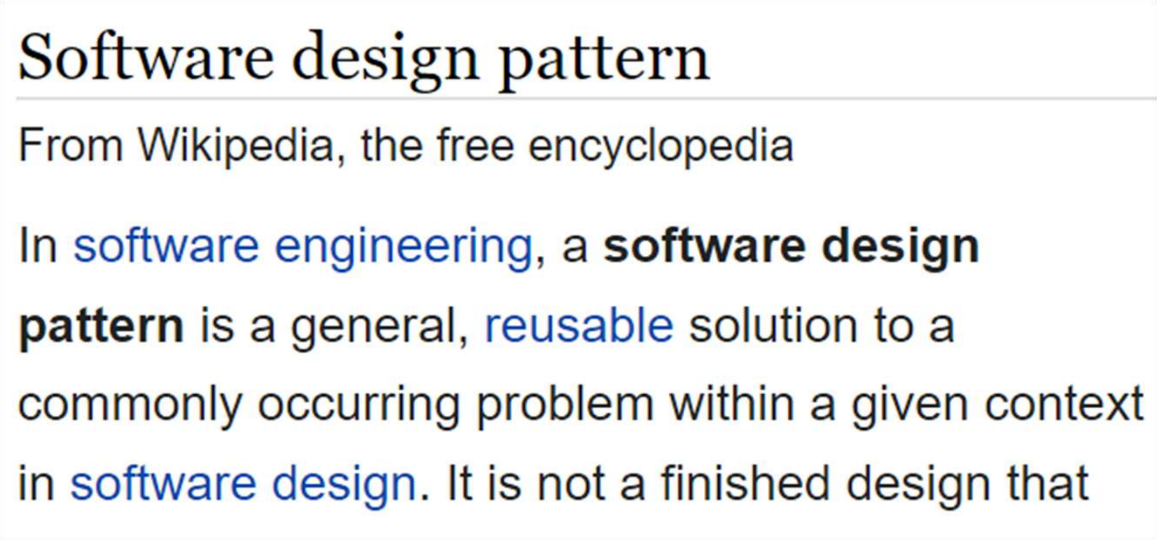
With this change, adding new functions requires modifying the definition of `Term` itself! You know I don’t modify upstream libraries.

There is gonna be a way to refactor this code, to fulfill both requirements. Both deep dark requirements.

I guess I should start doing some live programming at this time and introduce the visitor design pattern.

Design Patterns

Definition?



Software design pattern

From Wikipedia, the free encyclopedia

In software engineering, a **software design pattern** is a general, reusable solution to a commonly occurring problem within a given context in software design. It is not a finished design that

My definition: write code in a circuitous way to make it tolerable of certain refactoring situation.

Each design pattern has a corresponding programming language feature (and mostly related to the type system).

Singletons – objects (like in Kotlin)
Abstract factories – module signatures
Lazy initialization – lazy evaluation
Proxy – duck-typing interfaces
Observer – properties
Visitor – tagged unions (or, sum types)

I want to talk about sum types in particular.

They're supported in Rust.

https://doc.rust-lang.org/stable/rust-by-example/custom_types/enum.html
https://doc.rust-lang.org/stable/rust-by-example/flow_control/match.html

Advanced version of visitors – object algebra.

Corresponding language feature is called “row polymorphism”.

Thank you
for your attention

—

jetbrains.com