Concepts of Programming Languages:

Static vs. Dynamic Typing

Toni Schumacher

Institute for Software Engineering and Programming Languages

23. November 2015

Table of Contents

- Motivation
- Typing
  - Static Typing
  - Dynamic Typing
- Comparison
  - Static Typing
    - Advantages of Static Typing
    - Disadvantages of Static Typing
  - Dynamic Typing
    - Advantages of Dynamic Typing
    - Disadvantages of Dynamic Typing
  - Programming Concepts
    - Type Inference
    - Subtyping
    - Generics
- Outlook and Conclusion
  - Hybrid Languages
  - Conclusion

Motivation

Quote 1:

> *Once syntactic verbosity [...] is removed from statically typed languages, there is absolutely no advantage in using a dynamically typed language.*

jooq.org/2014/12/11/the-inconvenient-truth-about-dynamic-vs-static-typing/

Quote 2:

> *With unit tests [...] the types will also get checked, so you may as well go for dynamic typing and benefit from its advantages.*

teamten.com/lawrence/writings/java-for-everything.html

Motivation

- Widely discussed topic
- No exact/clear definitions

↪ Which issues do we want to tackle?

- Distinguish statically and dynamically typed languages
- Knowing benefits and disadvatages of both
- When to use which technique

Outline

Motivation

Typing
- Static Typing
- Dynamic Typing

Comparison
- Static Typing
  - Advantages of Static Typing
  - Disadvantages of Static Typing
- Dynamic Typing
  - Advantages of Dynamic Typing
  - Disadvantages of Dynamic Typing
- Programming Concepts
  - Type Inference
  - Subtyping
  - Generics

Outlook and Conclusion
- Hybrid Languages
- Conclusion

Typing

Definition: Type System

- Collection of type rules for a programming language
- Classifies expressions according to the kinds of values it compute
- Assigns type information to values

Definition: Type Checker

- Checks types of values for correctness
- Tracks type violation

Differentations:

- Strong / Weak
- Optional / Explicit
- Static / Dynamic

Static Typing

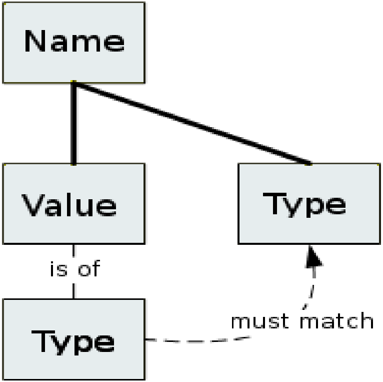
Definition: Static Typing

The type checker tries to assign objects to their particular type *during the compile process*.

Static Typing

Definition: Static Typing

The type checker tries to assign objects to their particular type *during the compile process*.



Static Typing

- Failure: compile attempt of the program code is canceled
- Considered as the origin of dynamic typing
- E.g. Ada, C, C++, Java, Fortran, Haskell, ML, Pascal, Perl and Scala

Dynamic Typing

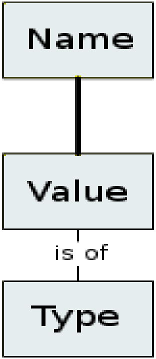
Definition: Dynamic Typing

Variables are associated with their contained values *during run-time* by tagging them with identifiers such as num, bool or fun.

Dynamic Typing

Definition: Dynamic Typing

Variables are associated with their contained values *during run-time* by tagging them with identifiers such as num, bool or fun.



Dynamic Typing

↪ Is inherently a restricted form of static typing with only a single type during compile-time

- Failure: partial or complete failure running the program
- E.g. Groovy, JavaScript, Objective-C, Perl, PHP, Prolog, Python, Ruby and Smalltalk

Static - Dynamic

- No clear boundaries between both
- Programming languages can't be equated with typing techniques

↪ Can use both static and dynamic type checking

- E.g. in static languages the main focus is the static type-checker and the dynamic typing (if existing) is not superficial

↪ Leads to controverse discussions about the topic

Outline

- Motivation
- Typing
  - Static Typing
  - Dynamic Typing
- Comparison
  - Static Typing
    - Advantages of Static Typing
    - Disadvantages of Static Typing
  - Dynamic Typing
    - Advantages of Dynamic Typing
    - Disadvantages of Dynamic Typing
  - Programming Concepts
    - Type Inference
    - Subtyping
    - Generics
- Outlook and Conclusion
  - Hybrid Languages
  - Conclusion

Advantages of Static Typing

- Earlier detection of programming mistakes
- More opportunities for compiler optimizations

↪ Increased runtime efficiency and reduced memory usage

- Better developing experience
- Better documentation in form of type annotations

Disadvantages of Static Typing

- Too rigid
- Can't handle changing requirements
- Code is less reusable
- Define some exceptions as dynamic errors (e.g. array-out-of-bound)
- Should be more complete

↪ Complex and overly complicated concepts added

- Can't handle a changing variable type

Disadvantages of Static Typing

- Too rigid
- Can’t handle changing requirements
- Code is less reusable
- Define some exceptions as dynamic errors (e.g. array-out-of-bound)
- Should be more complete

↪ Complex and overly complicated concepts added

- Can’t handle a changing variable type

Example: Changing variable type

```
employeeName = 9;
employeeName = "Steve";
```

Disadvantages of Static Typing

- Too rigid
- Can’t handle changing requirements
- Code is less reusable
- Define some exceptions as dynamic errors (e.g. array-out-of-bound)
- Should be more complete

↪ Complex and overly complicated concepts added

- Can’t handle a changing variable type

Example: Changing variable type

```
employeeName = 9;
employeeName = "Steve";
```

↪ Type error

Advantages of Dynamic Typing

- Better for prototyping systems with changing or unknown requirements
- Allows programs to generate types and functionality based on run-time data

↪ Much more flexible

Advantages of Dynamic Typing

- Better for prototyping systems with changing or unknown requirements
- Allows programs to generate types and functionality based on run-time data

↪ Much more flexible

Example: Eval function in dynamic languages

```
function example(str){
  var x = 10;
  var y = 20;
  var a = eval("x * y");
  var b = eval("2 + 2");
  var c = eval("x + 17");
  var d = eval(str);
}
```

Advantages of Dynamic Typing

- Better interaction with systems or modules with unpredictable changing output
- Important for data intensive programming

↪ Indispensable for dealing with truly dynamic program behavior

Disadvantages of Dynamic Typing

- Significantly more runtime errors

↪ More costs in development process

- More effort of writing exceptions
- Late detection of errors

↪ Complex troubleshooting and error fixing

- Type checker must check all classes during run-time

↪ Worse execution time

Programming Concepts

- Advantages and disadvantage of both typing techniques applied on important programming concepts:
  1. Type Inference
  2. Subtyping
  3. Genercis

Type Inference

Definition: Type Inference

- Process of finding a type for a program within a given type system

- Type inference $\neq$ dynamic typing
- Allows you to omit type information when declaring a variable

Example: Type inference in SML

```
fun fak(n) = if (n = 0) then 1 else n * fak(n-1);
```

- Relies on the availability of static type information

↪ Redundant for dynamic languages

- Only in statically typed languages like SML, Haskel, F# etc.

Subtyping

Definition: Subtyping

- Reflexive and transitive relation over types
- Satisfies subsumption:
  - If a term has type A, which is a subtype of a type B, then the term also has type B

- Ability to override existing super types with a related datatype
- Static type-checker has the type information needed to automatically lift inferred variables to required types

Subtyping

Example: Subtyped addition on nullable integers in C#

```
int? a = null;
int? b = 1;
int? c = a + b;
```

- Dynamic type-checker associates values with classes

↪ Exclude value types immediately

- Very inefficient with dynamic type checker
- Construct of dynamic typing needs to be rebuild to implement subtyping

Generics

Definition: Generics

- Reference type that has one or more type parameters

↪ Parameterized type

- Specifying a type argument to declare and instantiate a constructed type

- Help to avoid writing the same code multiple times

Dynamic type checking:

- Type informations are at first available at runtime

↪ Any collection or method is automatically generic

↪ Create highly reusable libraries

Example: Generics in dynamically typed languages

```
new Set<object.getClass()>(object);
```

Generics

Static type checking:

- Write a new function for any element type and any kind of collection

Example: Generics in statically typed languages

```
class Set{
  public Set(boolean b) { ... }
  public Set(int i) { ... }
  .. other constructors.
}
new Set<Object>(object);
```

Generics

Static type checking:

- Write a new function for any element type and any kind of collection

Example: Generics in statically typed languages

```
class Set{
  public Set(boolean b) { ... }
  public Set(int i) { ... }
  .. other constructors.
}
new Set<Object>(object);
```

↪ Endless number of types

Generics

- Better: dynamically scoped variables like arbitrary type T

Example: Usage of Generics in C#

```
interface IEnumerator<T> {
  T Current{get;}
}
```

↪ Generics are not impossible in static typing

↪ Much more easier to implement with dynamic type checking

Outline

- Motivation
- Typing
  - Static Typing
  - Dynamic Typing
- Comparison
  - Static Typing
    - Advantages of Static Typing
    - Disadvantages of Static Typing
  - Dynamic Typing
    - Advantages of Dynamic Typing
    - Disadvantages of Dynamic Typing
  - Programming Concepts
    - Type Inference
    - Subtyping
    - Generics
- Outlook and Conclusion
  - Hybrid Languages
  - Conclusion

Hybrid Languages

- Dynamic type checking and static type checking appear to be incompatible
- Can coexist harmoniously
- Different techniques of solving this misery:
  - Static type-checker verifies what it can and dynamic checks verify the rest
  - Distinguish between statically typed and dynamically typed variables

Hybrid Languages

Example: Static and dynamic variables in C#

```
class ExampleClass{
  public ExampleClass() { }
  public void exampleMethod1(int i) {}
}
static void Main(string[] args){
  ExampleClass ec = new ExampleClass();
  //would cause compiler error
  ec.exampleMethod1(10, 4);

  dynamic dynamic_ec = new ExampleClass();
  // no compiler error, but  cause run-time exception.
  dynamic_ec.exampleMethod1(10, 4);
}
```

Conclusion

Questions?

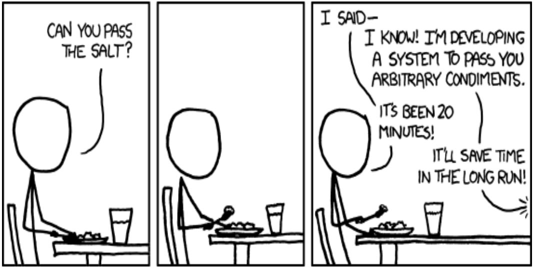
Conclusion

Static or Dynamic type checking?

Conclusion

My opinion:

- Dynamic typing for small programs and scripts (fast development, no major safety requirements)
- Static typing mechanisms for applications relevant to security
- Fully expressive language supports the interplay between static and dynamic techniques
- Static typing where possible and dynamic typing when needed


CAN YOU PASS THE SALT?
I SAID—
I KNOW! I'M DEVELOPING A SYSTEM TO PASS YOU ARBITRARY CONDIMENTS.
IT'S BEEN 20 MINUTES!
IT'LL SAVE TIME IN THE LONG RUN!

xkcd.com

Conclusion

Thank you for your attention.