Programming Language Concepts
Type Systems

Janyl Jumadinova

3 April, 2023

Type System consists of:

1. a mechanism to define types and associate them with certain language constructs, and
2. a set of rules.

Type System consists of:

1. a mechanism to define types and associate them with certain language constructs, and
2. a set of rules.

- The purpose of **type checking** is to verify that operations performed on a value are in fact permissible.
- *Type checking* cannot prevent all meaningless operations but it catches enough of them to be useful.

Examples of Data Types

Examples of Data Types

- Primitive data types
- Reference data types
- ADT (abstract data types)

Data Types

What are types good for?

- implicit context,
- checking,
- make sure that certain meaningless operations do not occur.

What Does “Implicit Context” Mean?

When we see a statement such as: `total = num1 + num2;`
are we:

- adding two `int` values, storing in an `int`?
- adding two `int` values, storing in a `double`?
- concatenating a `String` and an `int`, storing in a `String`
- adding an `int` and a `double`, storing in a `double`?

What Does “Implicit Context” Mean?

- If we were writing machine code, WE WOULD HAVE TO SPECIFY THIS, e.g.
    - explicitly convert `int` to `double` before adding to a `double` or storing as a `double`, or
    - have to reserve space for the new `String`, etc.

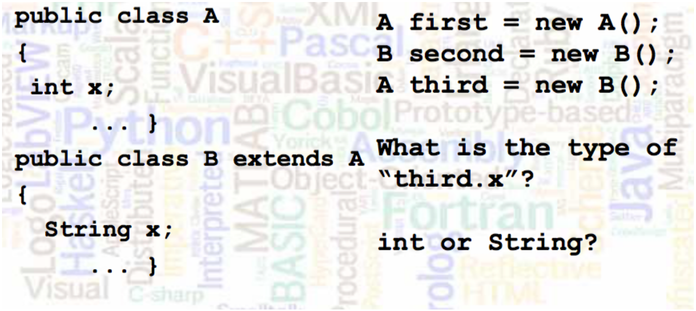
What Does “Implicit Context” Mean?

- If we were writing machine code, WE WOULD HAVE TO SPECIFY THIS, e.g.
  - explicitly convert `int` to `double` before adding to a `double` or storing as a `double`, or
  - have to reserve space for the new `String`, etc.
- Type information gives the compiler or interpreter a context that enables it to figure this out.

What is Polymorphism?

- **Polymorphism** results when the compiler finds that it doesn't need to know certain things.
- In this context we are concerned with situations when the same variable can refer, at different times, to values of different types.
- The most familiar example to Java programmers occurs in subclasses.

Polymorphism



```
public class A
{
 int x;
    ... }
public class B extends A
{
  String x;
    ... }
```

```
A first = new A();
B second = new B();
A third = new B();
```

What is the type of "third.x"?

int or String?

Polymorphism

In Java, a subclass cannot override an instance variable of the parent class; however, it can “shadow it”.
On the other hand, methods CAN be overridden.

Data Types

STRONG TYPING has become a popular buzz-word

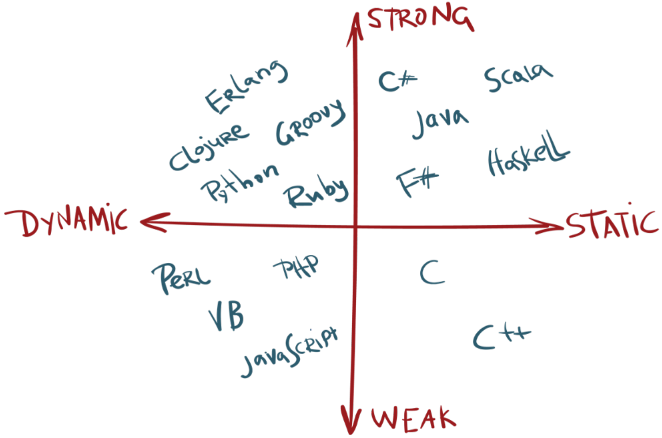
– informally, it means that the language prevents you from applying an operation to data on which it is not appropriate.

Data Types

STRONG TYPING has become a popular buzz-word
– informally, it means that the language prevents you from applying an operation to data on which it is not appropriate.

STATIC TYPING
means that the compiler can do all the checking at compile time.

Type Systems: Examples



Credit: Mayank Bhatnagar

Type Systems: Common Terms

- **Discrete types** - countable
  integer, boolean, char, enumeration, subrange

Type Systems: Common Terms

- **Discrete types** - countable
  integer, boolean, char, enumeration, subrange
- **Scalar types** - one-dimensional

Type Systems: Common Terms

- **Discrete types** - countable
  integer, boolean, char, enumeration, subrange
- **Scalar types** - one-dimensional
- **Composite types**
  records (unions), arrays, sets, pointers, lists, files

Composite types

- Several values of varying types under a common name.
- Made somewhat obsolete by classes and instances in object-oriented programming.

Composite types

- Several values of varying types under a common name.
- Made somewhat obsolete by classes and instances in object-oriented programming.

```
struct rec { /* here we declare the type */
     int i; double x; char s[10];
 };
 struct rec a,b,c; /* declare variables */
 a.i = 10; b.x = 4.14
```

Composite types: Unions

Several values of varying types under a common name and sharing the same memory.

```
union share { /* here we declare the type */
    int i; double x; char s[10];
};
union share a,b,c; /* declare variables */
a.i = 10; /* this changes a.x and a.s also */
```

Composite types: Enumerated types

Symbolic names (actual underlying values not important).

```
enum weekday {mon,tue,wed,thu,fri,sat,sun};
   enum weekday day;
   day = mon;
   if (day < fri) ...
```

Type Systems

A collection of features is **orthogonal** if there are no restrictions on the ways in which the features can be combined

.

- ORTHOGONALITY is a useful goal in the design of a language, particularly its type system.
- It makes a language easy to understand, easy to use, and easy to reason about

Orthogonality Examples

1. *Pascal* is more orthogonal than *Fortran*, (because it allows arrays of anything, for instance), but it does not permit variant records as arbitrary fields of other records (for instance).

Orthogonality Examples

1. *Pascal* is more orthogonal than *Fortran*, (because it allows arrays of anything, for instance), but it does not permit variant records as arbitrary fields of other records (for instance).
2. In *C*, parameters are passed by value, unless they are arrays (which are passed by reference).

Orthogonality Examples

1. *Pascal* is more orthogonal than *Fortran*, (because it allows arrays of anything, for instance), but it does not permit variant records as arbitrary fields of other records (for instance).
2. In *C*, parameters are passed by value, unless they are arrays (which are passed by reference).
3. The most orthogonal programming language is *ALGOL 68*. Every language construct in ALGOL 68 has a type, and there are no restrictions on those types.

Type Checking

A TYPE SYSTEM has rules for:

- **type equivalence** (when are the types of two values the same?)

Type Checking

A TYPE SYSTEM has rules for:

- **type equivalence** (when are the types of two values the same?)
- **type compatibility** (when can a value of type A be used in a context that expects type B?)
- **type inference** (what is the type of an expression, given the types of the operands?)