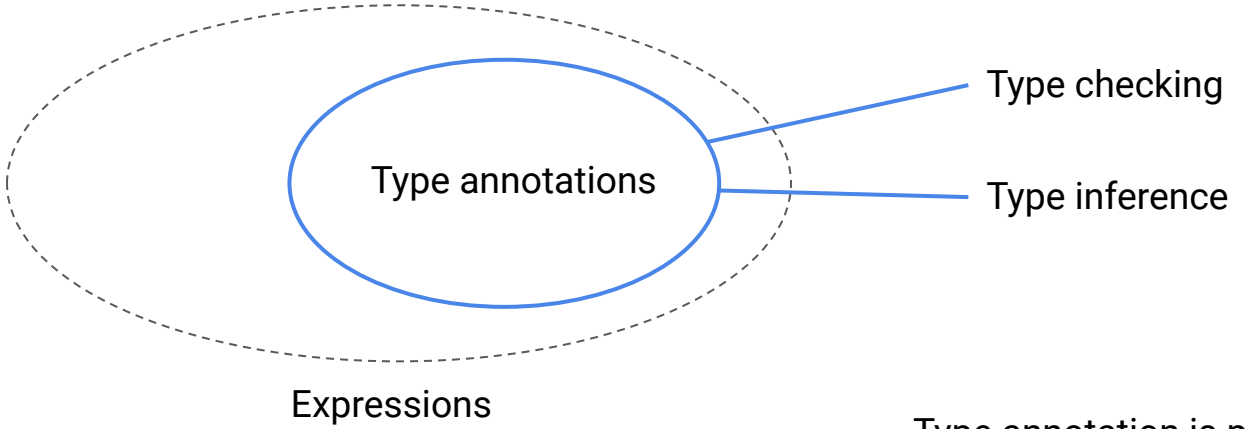
Elements of Type Systems

Ken Q Pu, Faculty of Science, Ontario Tech University

Elements of Type Systems



Type annotation is part of the syntax of the programming language.

Type annotation over expressions

Concrete types

- Complete descriptions of the data.
- Allows construction of new instances in memory.

Abstract types

- Partial description of data
- Only applicable to data already constructed in memory.

Type annotations

Basic types

- Physical data layout in memory

Concrete class

- Composite data layout
- Access by members and methods

Extension and Composition

- Building more complex types using existing types

Abstract classes

- Functional specification of data
- Partially concrete specs

Interfaces

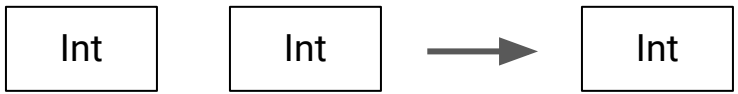
- Purely abstract with no concrete specs
- Spec on both field and methods
- Default method implementation

Generics

- Composite types with type parameters

Function Types

- A function is described by function types.
- A function type describes the types of the parameters and the type of the return value.



Some Types

Basic types

- Boolean: 1B
- Byte: 1B
- Char: 2B
- Int and UInt: 4B
- Int64 and UInt: 8B
- Float32
- Float64 and Double: 8B

They are not data, but description of some data to be to created or manipulated during runtime of the code.

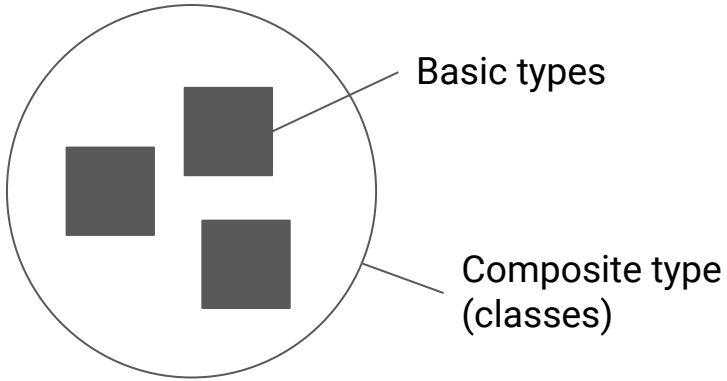
1. Data are instances of types.
2. Evaluation is performed on instances of types, not the types themselves.
3. Evaluation only occurs when the expression passes type checking.

All basic types are provided by programming language - not the program. So, we cannot define any additional basic types.

Concrete Classes

Examples

- String
- BigDecimal



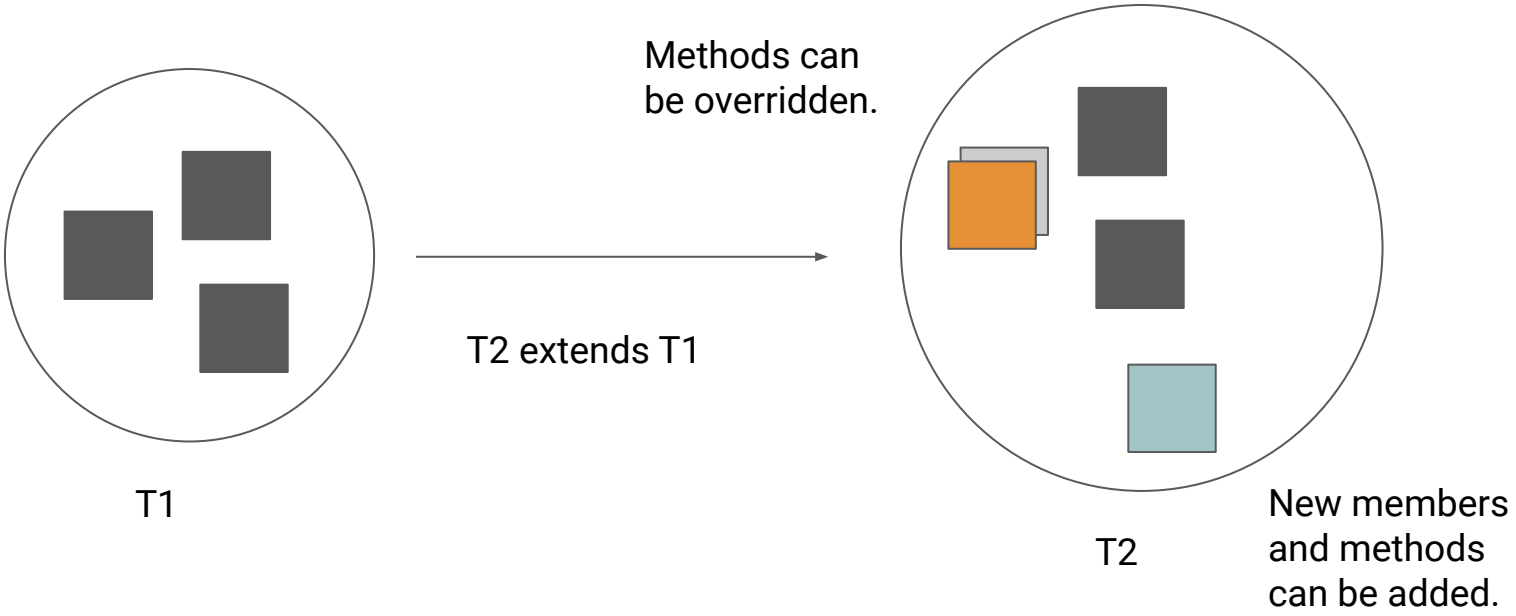
Objects are instances of classes.

How do we access parts of an object?

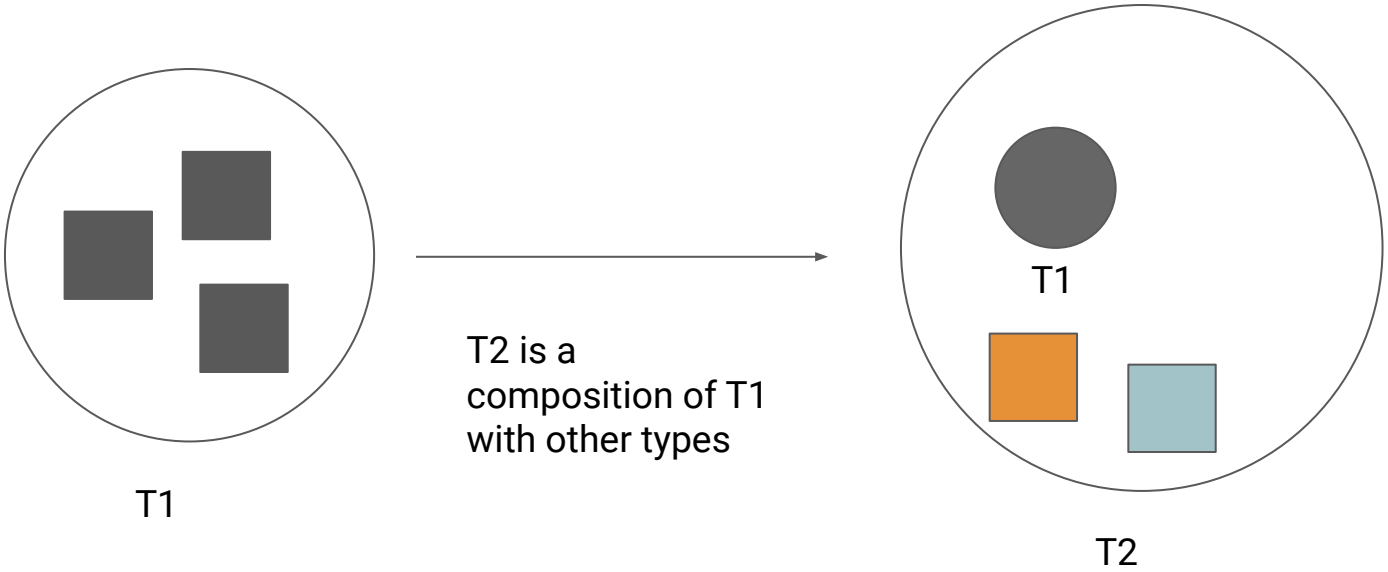
- Members (aka fields, properties, ...)
- Methods

Programs can define many concrete classes.

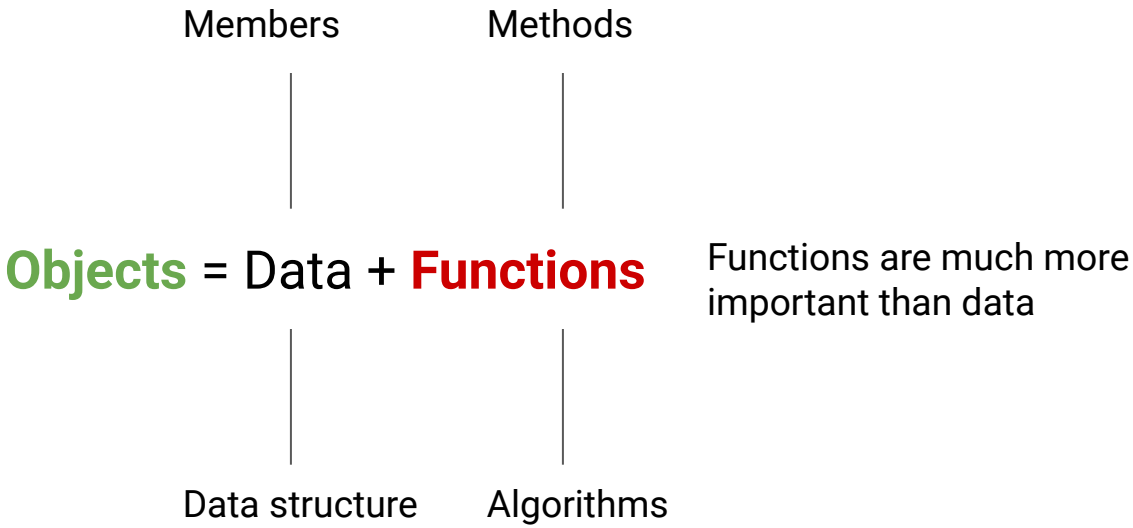
Extension


Methods can
be overridden.
T2 extends T1
T1
T2
New members
and methods
can be added.

Composition



Object-oriented programming



Functional specification of data

We specify objects by the members and methods they **should** have.

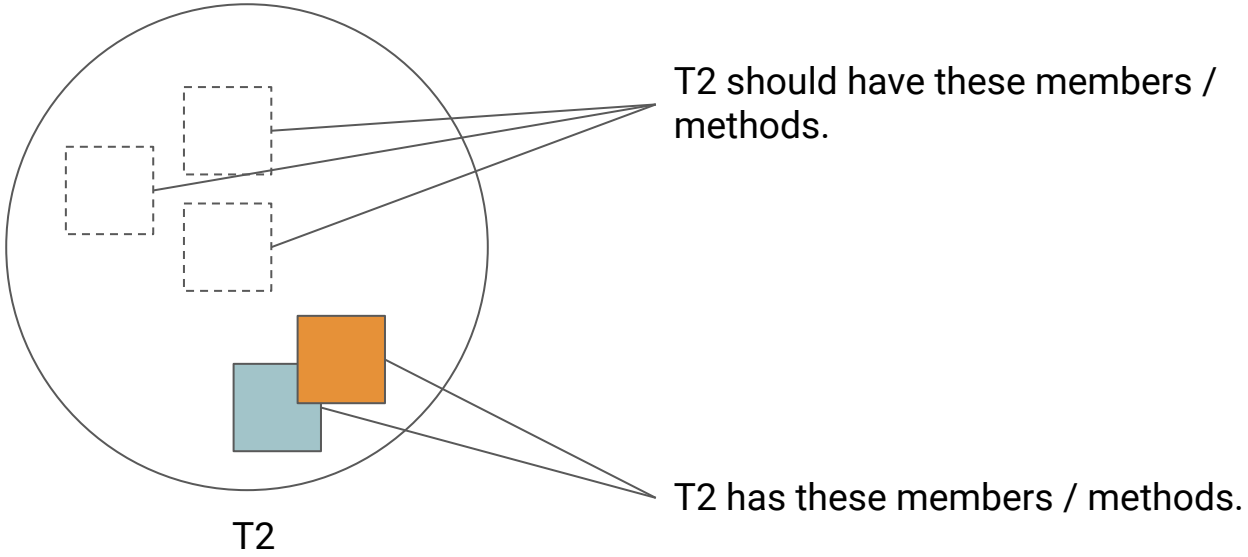
A student in CSCI3055U **should** be able to program in Clojure.

This is an abstract specification on objects.

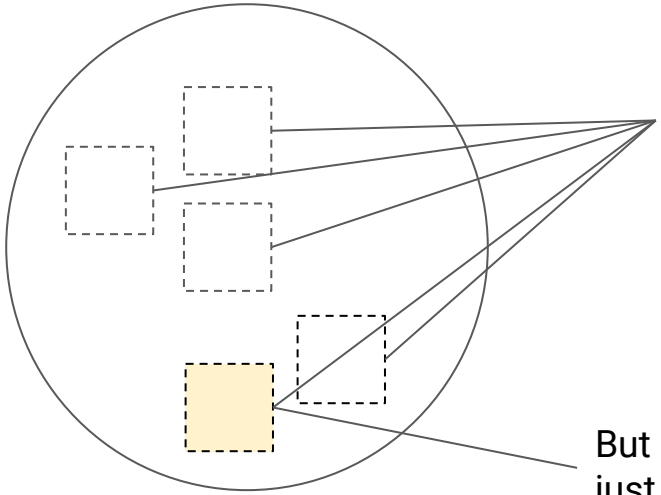
A student in CSCI3055U **writes** Clojure code like this ...

This is a concrete description of objects.

Abstract Classes: concrete and abstract



Interfaces: 100% abstract (sort of)



I3

I3 objects should have these members / methods.

But if an object doesn't have this method, just use this implementation...

Generics

Consider a data structure: **List**.

- **List** of *what*?
- But all lists have a common set of methods and members
  - List.add(___)
  - List.get(i)
  - List.size
- Using generics, we can still describe **List** as a type without knowing *what* is.

Generics with type parameters

Let T be a type.

List of T is a type that has the following methods:

- add(thing: T)
- get(index: Integer) -> T
- size: Integer

- T is a type parameter.
- T is used to describe **List**.
- This makes **List** a generic type, with one type parameter T.

`List<T>` means List of T things.

Ken Q Pu, Faculty of Science, Ontario Tech University

Multiple type parameters

Consider HashMap as a type. It has two type parameters: one for the keys of the hashmap, and another for the values: K, V.

Hashmap of <K, V>

- add(key: K, value: V)
- get(key: K) -> V
- size: Integer