Midterm Review

Hartmut Kaiser

https://teaching.hkaiser.org/spring2024/csc3380/

Equivalence vs. Equality

- Equational reasoning must be applied:
  - Equivalence is reflexive, symmetric, and transitive:

$$a \cong a$$

$$a \cong b \Leftrightarrow b \cong a$$

$$(a \cong b) \land (b \cong c) \Leftrightarrow (a \cong c)$$

  - Equality implies substitutability:

for any function f on T, a $== b \Rightarrow f(a) == f(b)$

  - Inequality must be the negation of equality:

$$(a \neq b) \Leftrightarrow \neg(a == b)$$

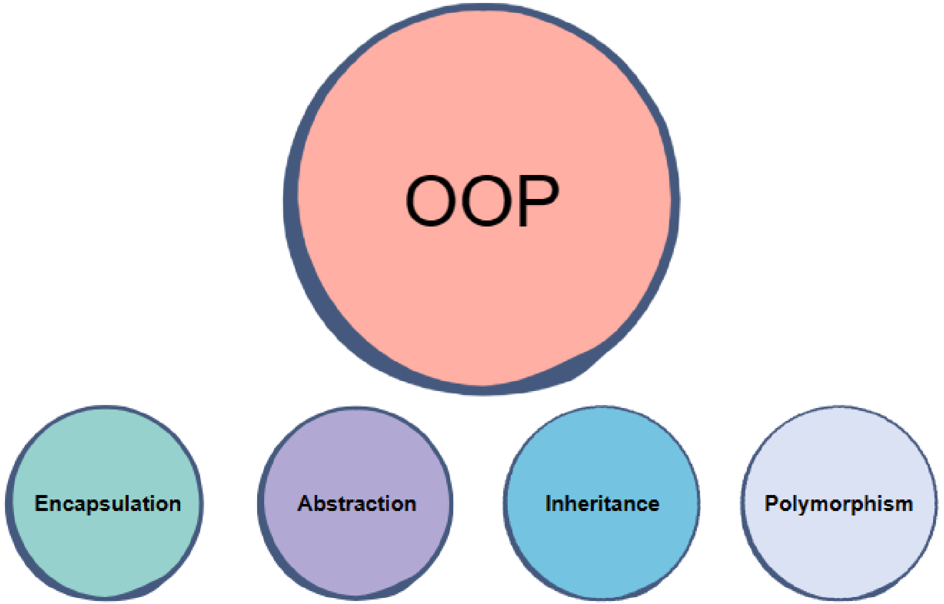
StrictWeak and Total Ordering

- A `StrictWeakOrdering` is a Binary Predicate that compares two objects, returning true if the first precedes the second
  - Applying `TotalOrdering` to equivalence classes
  - Invoke function on an element and totally order what it returns

- `StrictWeakOrdering`
  - Partial ordering:
    - Irreflexivity: `!f(x, x)`
    - Antisymmetry: `f(x, y) ⬄ !f(y, x)`
    - Transitivity: `f(x, y) && f(y, z) ⬄ f(x, z)`
  - Transitivity of equivalence
    - if $x \cong y$ and $y \cong z$, then $x \cong z$

- `TotallyOrdered`
  - Additionally connectedness: `!f(a, b) && !f(b, a) ⬄ a == b`
  - Transitivity of equality
    - if $x == y$ and $y == z$, then $x == z$

Object Oriented Programming



- The four pillars of object-oriented programming are:
- **Encapsulation**: containing information in an object, exposing only selected information
- **Abstraction**: only exposing high-level public methods for accessing an object
- **Inheritance**: child classes inherit data and behaviors from the parent class
- **Polymorphism**: many methods can do the same task

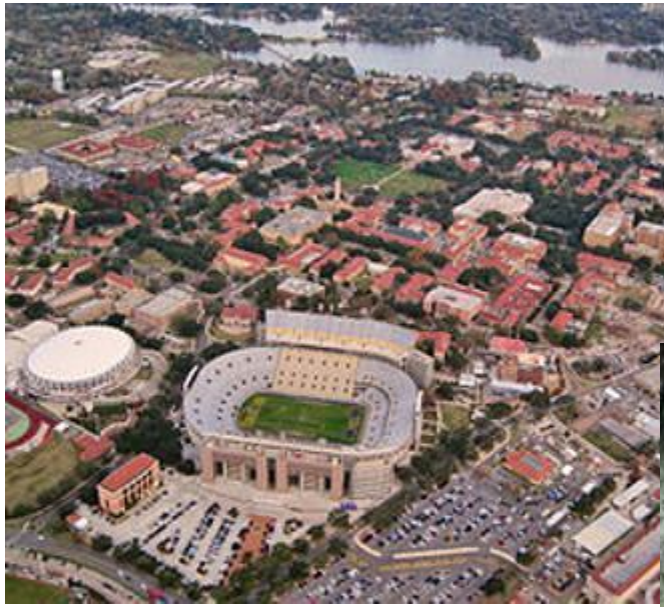
See also: [What is object-oriented programming? OOP explained in depth]

What is a ‘type’?

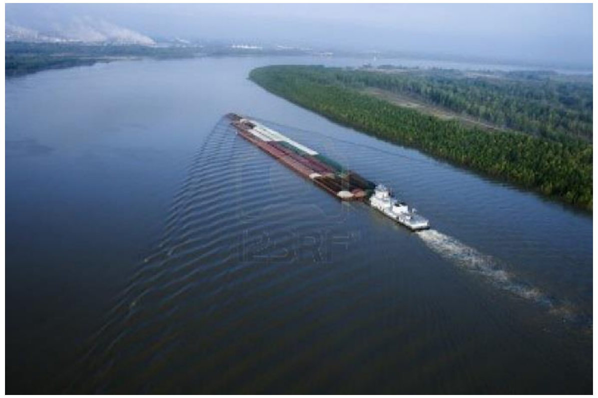
- A ‘type’ (of an object) defines the following things:
  - The amount of memory required to store all the data that is needed to support the operations valid for a type
  - The rules of how to interpret the bits in that memory as values in order to be able to make sense of the bit-salad
  - The set of values that are valid
  - The set of operations that are valid on those values
- Examples of types:
  - `int`, `double`, `float` (built-in types)
  - `token`, `token_stream`, `std::vector`, etc. (user-defined types)

What is an 'object'?

- An object is an instance of a type
  - Occupies memory
  - Has an optional name (is a variable)
  - Has a lifetime
- Objects in C++ don't change their type
  - C++ is a type-safe language
  - C++ checks types and type compatibility at compile time
- Examples of objects:
  - `int i = 0;`
  - `token t('+');`
  - `std::vector<int> v = {1, 2, 3, 4, 5};`



LSU CENTER FOR COMPUTATION & TECHNOLOGY



STE||AR GROUP