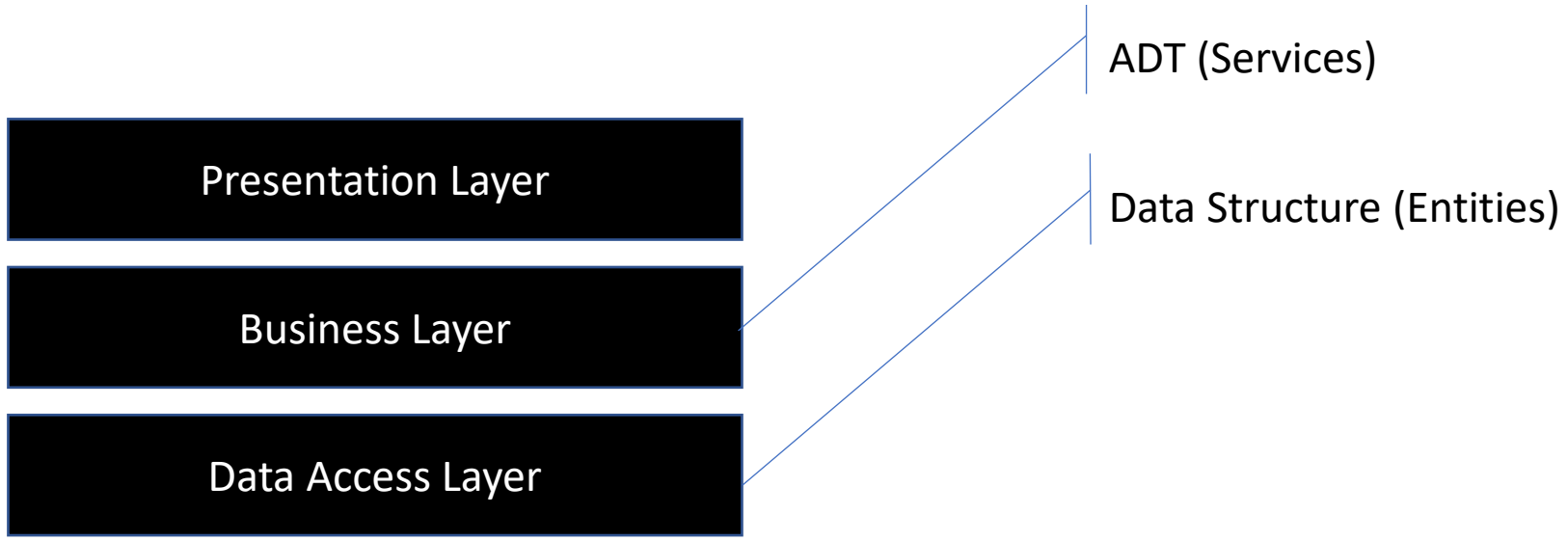
Abstract Data Type
vs
Data Structure

Software

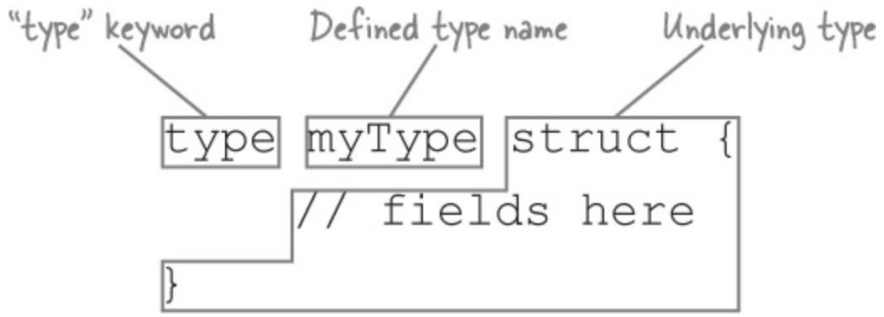
x and f(x) *is a Data Structure*
f *is an Abstract Data Type (ADT)*

Software Architecture

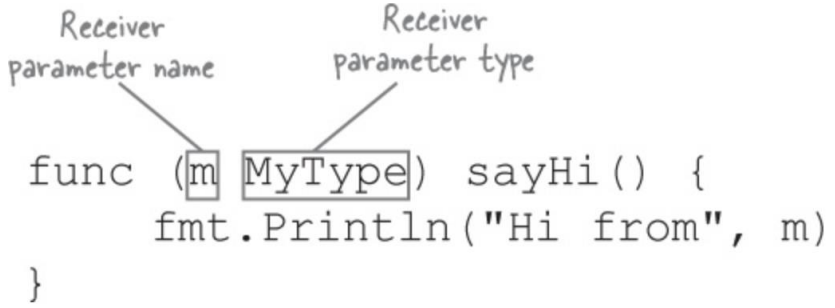
Layered Architecture



Golang

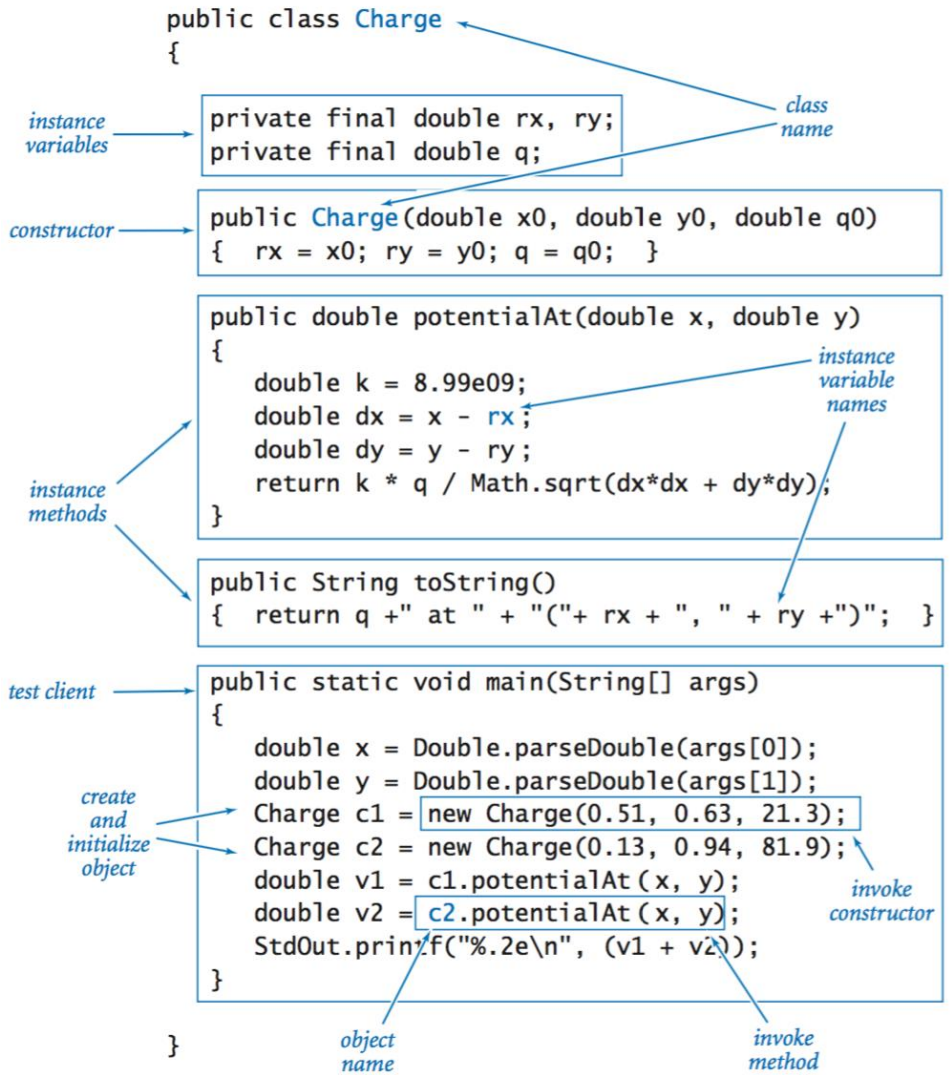


Data Structure



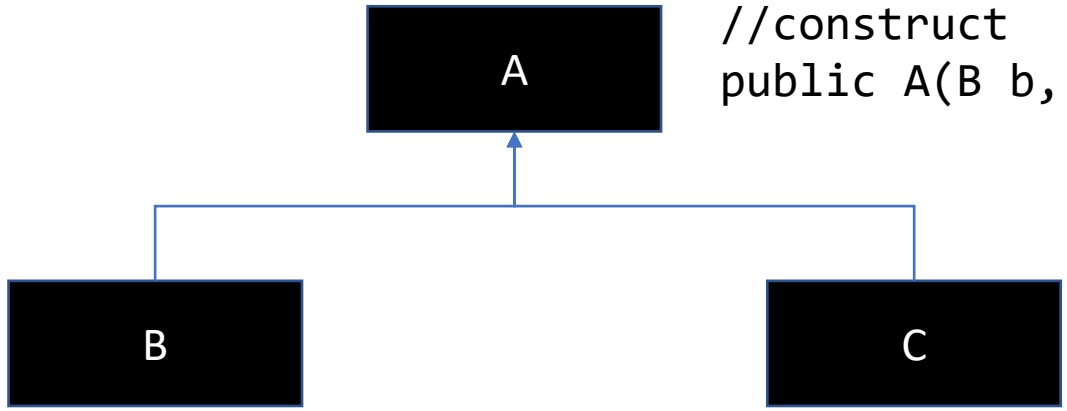
ADT

Java


public class Charge
{
instance variables
private final double rx, ry;
private final double q;
class name
constructor
public Charge(double x0, double y0, double q0)
{ rx = x0; ry = y0; q = q0; }
public double potentialAt(double x, double y)
{
double k = 8.99e09;
double dx = x - rx;
double dy = y - ry;
return k * q / Math.sqrt(dx*dx + dy*dy);
}
instance variable names
instance methods
public String toString()
{ return q +" at " + "("+ rx + ", " + ry +")"; }
test client
public static void main(String[] args)
{
double x = Double.parseDouble(args[0]);
double y = Double.parseDouble(args[1]);
create and initialize object
Charge c1 = new Charge(0.51, 0.63, 21.3);
Charge c2 = new Charge(0.13, 0.94, 81.9);
double v1 = c1.potentialAt(x, y);
double v2 = c2.potentialAt(x, y);
invoke constructor
StdOut.printf("%.2e\n", (v1 + v2));
}
}
object name
invoke method

Modularity

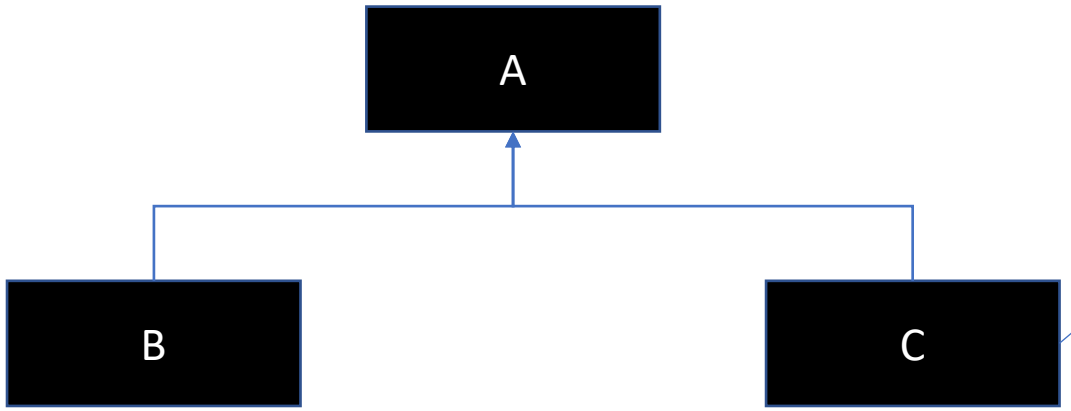
- Keeps the complexity of a large program manageable
- Isolates errors
- Eliminates redundancies
- Encourages reuse (write libraries)
- A modular program is:
  - Easier to write
  - Easier to read
  - Easier to modify

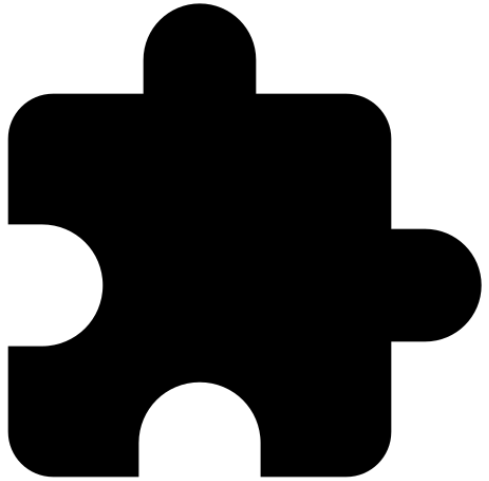



```
//construct
public A(B b, C c);
```

Note: A, B and C are modules. Other hand, a module is an ADT

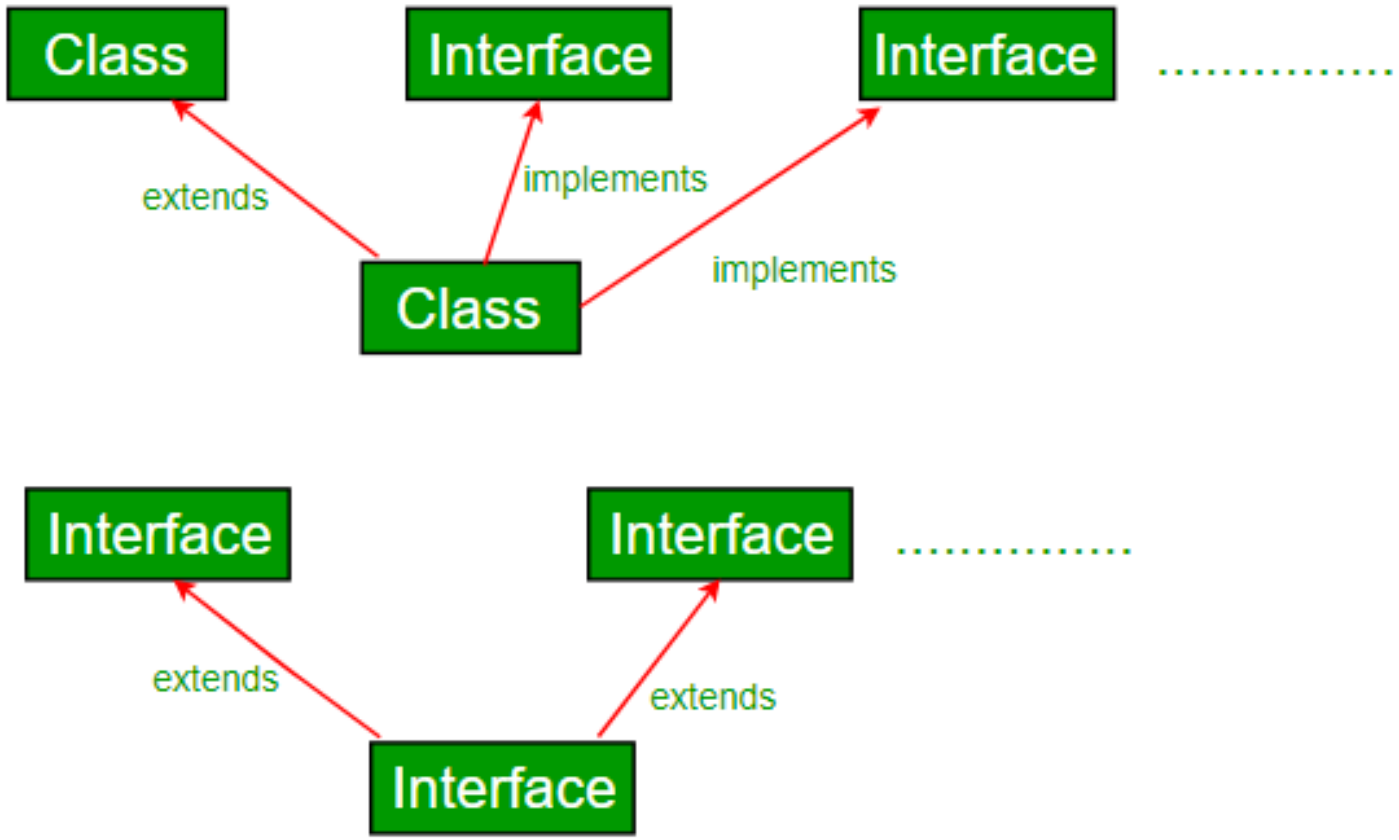
Abstract Data Type



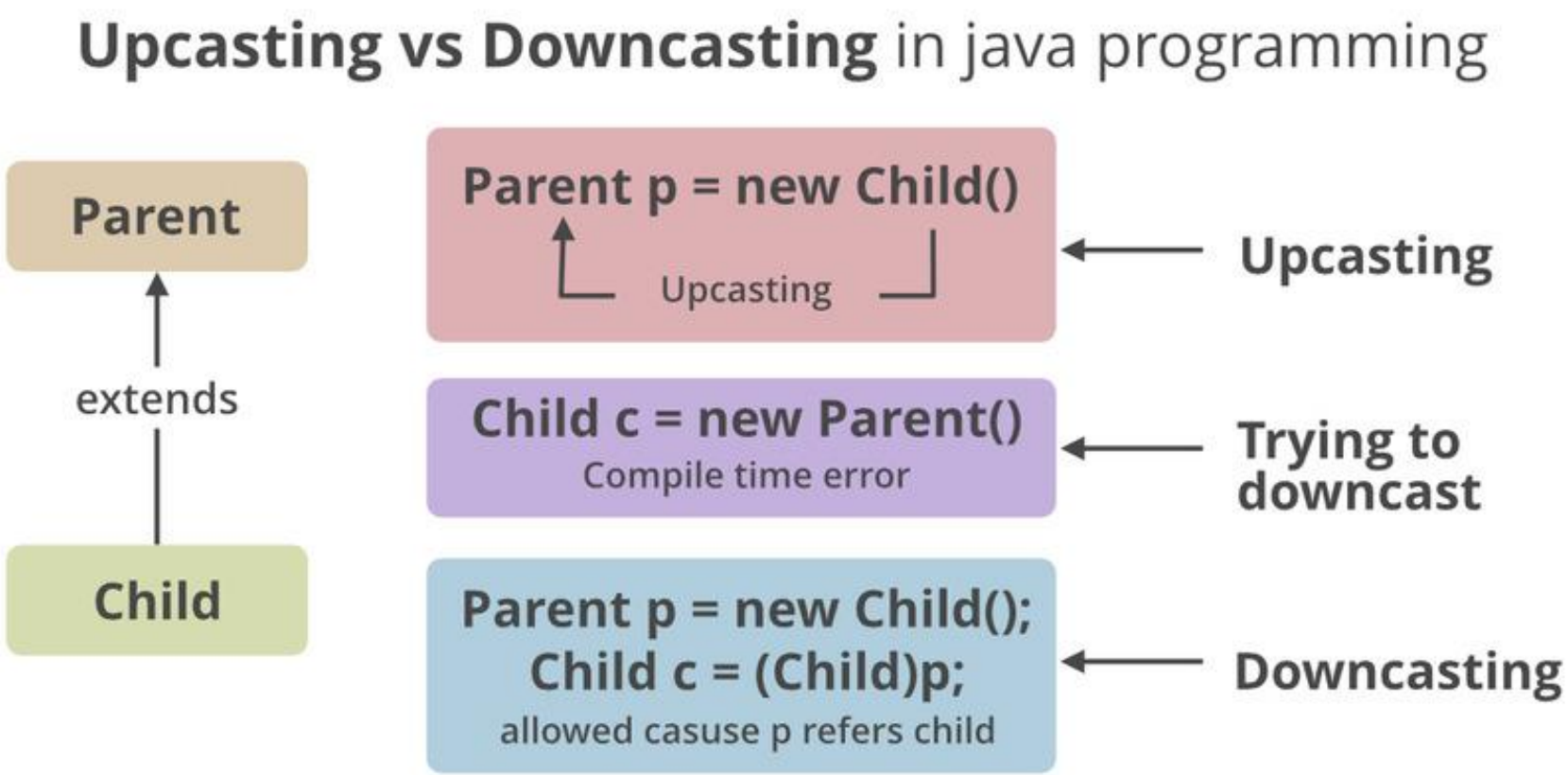


Create subtype with interfaces or Inherency

Subtype


Class
Interface
Interface
.....
extends
implements
implements
Class
Interface
Interface
.....
extends
extends
Interface

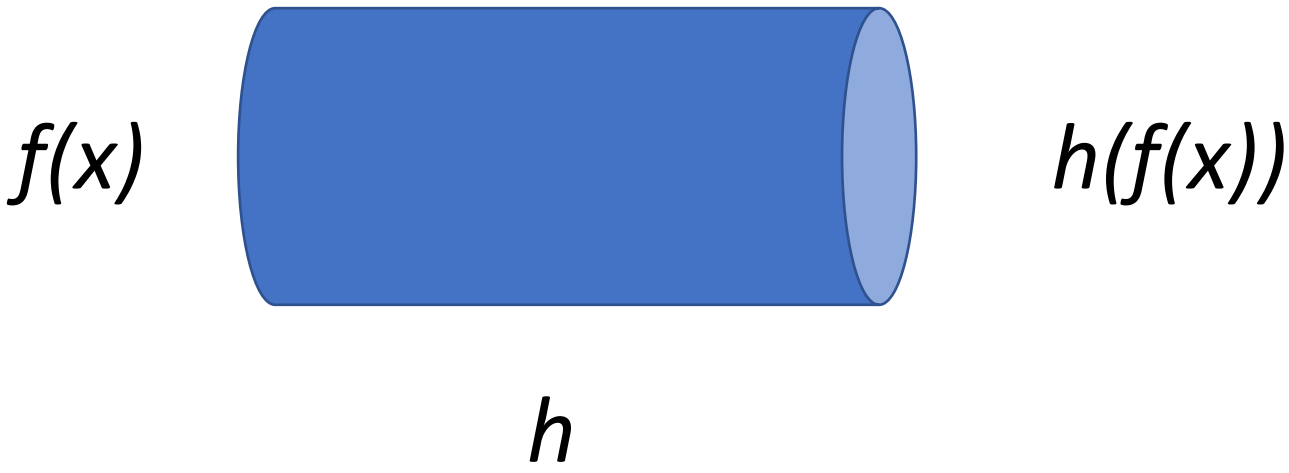
Example


Upcasting vs Downcasting in java programming
Parent
extends
Child
Parent p = new Child()
Upcasting
Upcasting
Child c = new Parent()
Compile time error
Trying to downcast
Parent p = new Child();
Child c = (Child)p;
allowed casuse p refers child
Downcasting

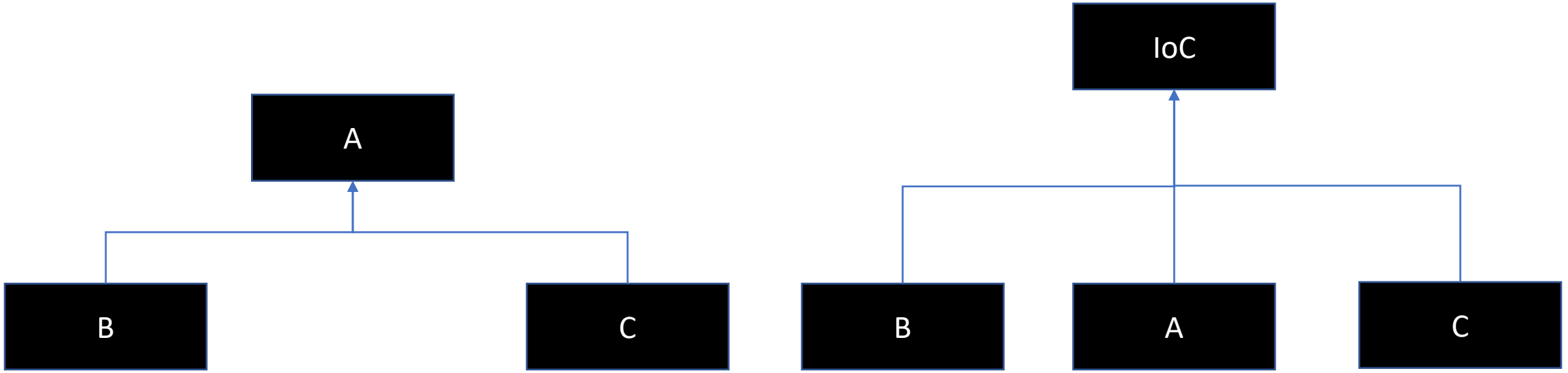
Liskov Substitution Principle

if S is a subtype of T, then objects of type T in a program may be replaced with objects of type S without altering any of the desirable properties of that program

Higher-order function instead of ADTs



Dependency Injection Container



Reference

- https://www.youtube.com/watch?v=0iyB0_qPvWk
- https://www.youtube.com/watch?v=hxGOiiR9ZKg&t=302s
- https://www.eecs.yorku.ca/~jackie/teaching/lectures/2020/F/EECS3311/slides/02a-Modularity-ADTs.pdf
- https://www.cs.auckland.ac.nz/compsci105s1c/lectures/Bruce/12-Abstract_Data_Type.pdf
- https://www.youtube.com/watch?v=_jTc1BTFdIo