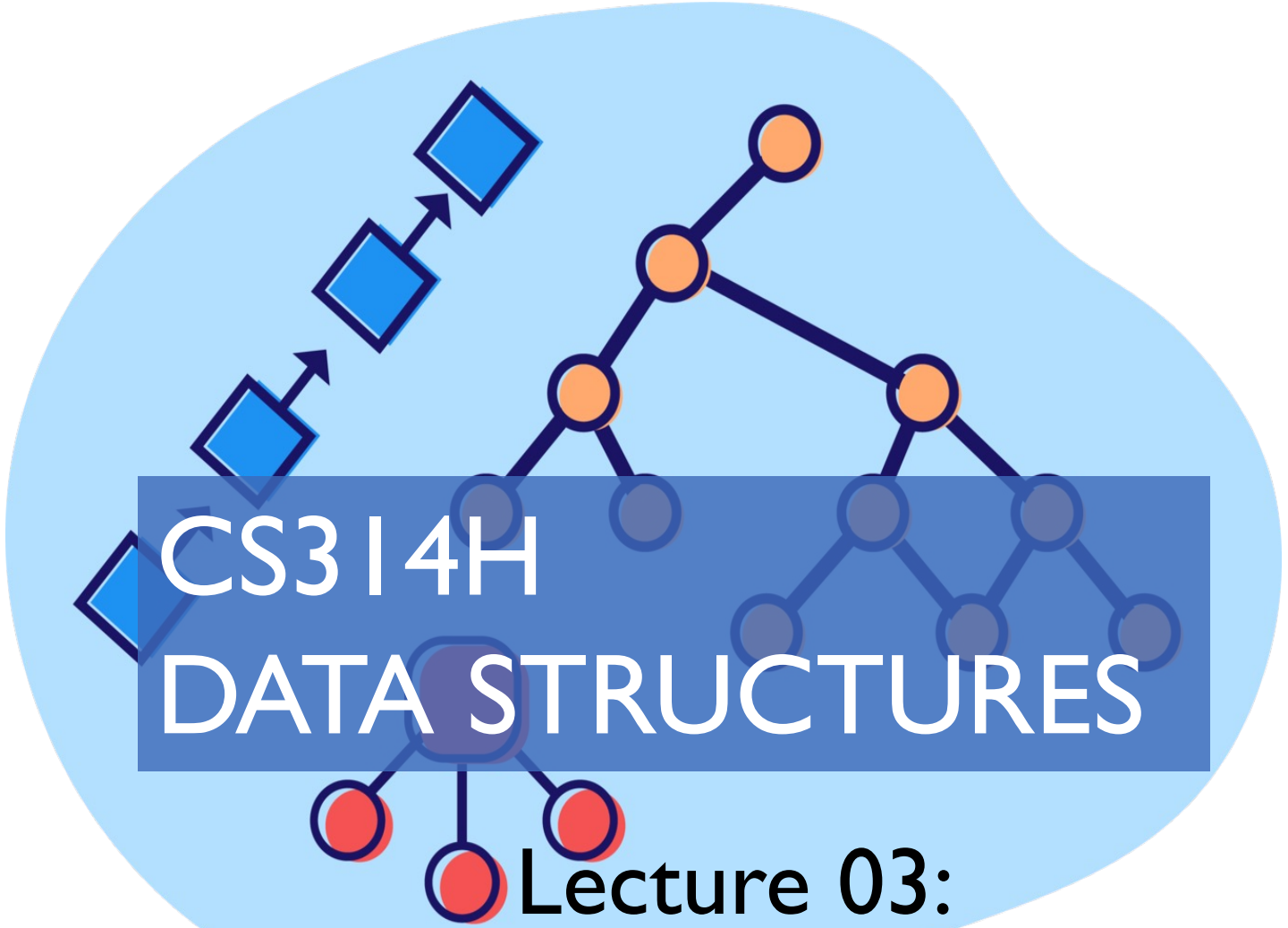


CS314H
DATA STRUCTURES

Lecture 03:
OOP Inheritance
and Subtype Polymorphism

Mikyung Han

Please, interrupt and ask questions AT ANY TIME !

Reminders

- Assignment 0 Part 2 (Canvas) due tonight
- Prog1 spec reading (Perusall) due tonight
- Read Ch3/Ch4 if you haven't done so
- UW Coding Style Guide
  - Commenting
  - Collections and objects

Outline

1. Administrative
2. OO Model Recap

Pick 2 from below and tell your neighbor about it

- 1 What is Object-oriented Model? What does it focus on?
- 2 What is information hiding principle?
- 3 How does Java provides encapsulation?
- 4 What are the 3 defining characteristics of OO Language?

The Object-Oriented Model

- Encapsulates all program data inside of objects and can be accessed through operations defined on these objects

- What does it focus on? Data >> Operation
- Why is this good? Separation of concern
- Comparison with NOT OO paradigm
  - Imperative/procedural

Defining characteristics of OO Languages

- Encapsulation
- Inheritance
- Subtype Polymorphism

Outline

1. Administrative
2. Recap
3. Types

Before talking about inheritance we first need to talk about TYPES

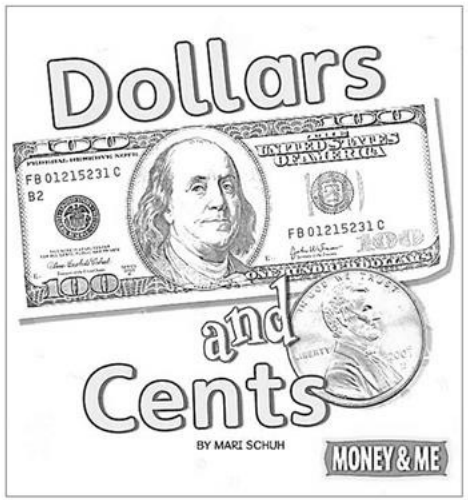
- What is a type in a programming language?

- Example
  - int: min/max
  - int a = 3;
    int b = 4;
    a = b % a;
  - float: min/max
  - float x = 3;
    float y = 4;
    x = y % x;

Why are types useful?

What are some real-world examples of TYPES?

US DOLLARS and COINS



1 cent
1 ¢
PENNY

5 cents
5 ¢
NICKEL

10 cents
10 ¢
DIME

25 cents
25 ¢
QUARTER

50 cents
50 ¢
HALF DOLLAR

1 DOLLAR
$ 1
DOLLAR

Real world examples of TYPES

- What are the possible values?
- What can we do with it?
- What can we NOT do with it?

Why are types useful in programming language?

- If we know a type of a variable… what can we do?

Type Checking in Programming Languages

- When does it happen?

- Static type checking
  - double x = 3;
    String y = "hi";
    x = y + x; // legal?

- Dynamic type checking
  - void foo(Object o) {
      Shape s = (Shape) o; //legal?
    }

Java does type checking statically when possible, dynamically when necessary

Outline

1. Administrative
2. Recap
3. Type
4. Inheritance

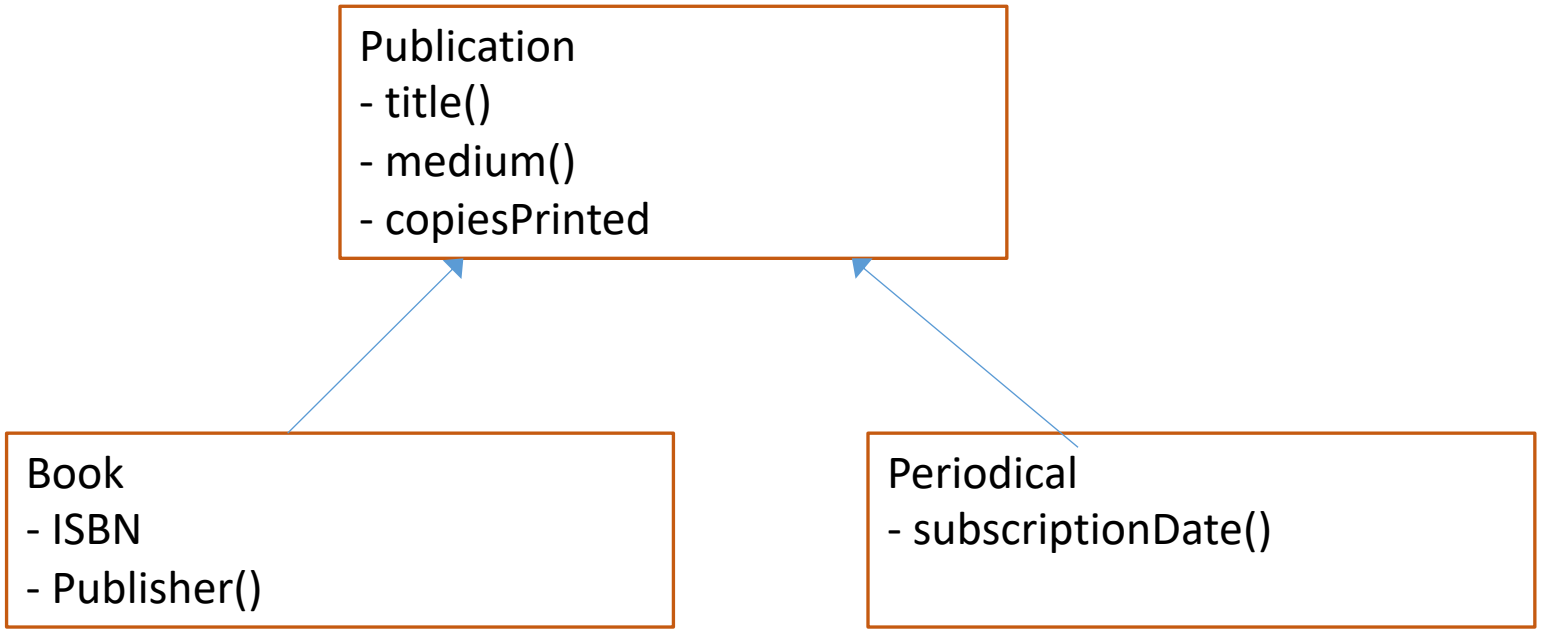
Inheritance: defines a new object in terms of existing object

- Let Y (new object) inherits from X (old object)
  - What is the benefit?

- Y "IS A" X

In Java, child class inherits public/protected methods and fields from parent class

Inheritance is a way to REUSE code



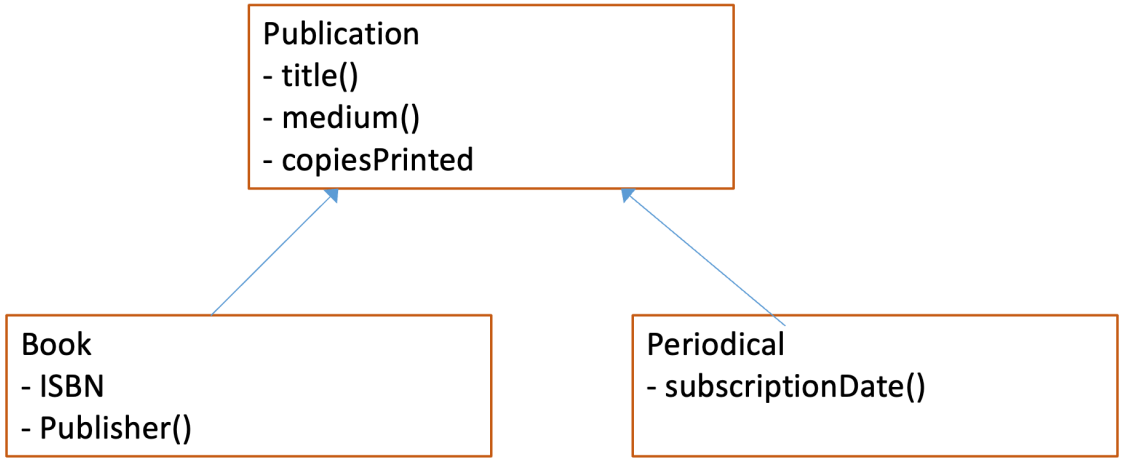
Parent class == super class
Child class == sub class

Class in Java introduces types
Inheritance introduces sub-types

Discuss with your friend: Is this legal?


```
Publication p;
Book b;

p = new Publication();
b = new Book();
int i = b.ISBN;      // Is this legal?
p = b;       // Is this legal?
copies = p.CopiesPrinted;   // legal?
i = p.ISBN;      // legal?
b = new Publication();  // legal?
```



Outline

1. Administrative
2. OO Model Recap
3. Type
4. Inheritance
5. Subtype polymorphism

Subtype polymorphism allows subtypes to be used whenever supertype is expected

- What is the benefit?

```
Shape s = new Triangle();
foo(s);

void foo(Shape s) {
    s.getArea();
}
```

Allows writing code now to be reusable in the future

Inheritance and Subtype Polymorphism

- Both allows code reuse and facilitate changes
- Inheritance allows to reap the benefits now from the past
  - by reusing the past code
  - Also allows added/new behaviors
- Polymorphism allows to reap the benefits in the future from the present
  - I am writing Shape client code now
  - I think I might introduce new shape Triangle or shape Ghost, etc.
  - I can just plug in Triangle or Ghost instance to wherever Shape is expected!
  - No need to change any Shape client code