Object-oriented programming CS10003

Lecturer: Do Nguyen Kha

Contents

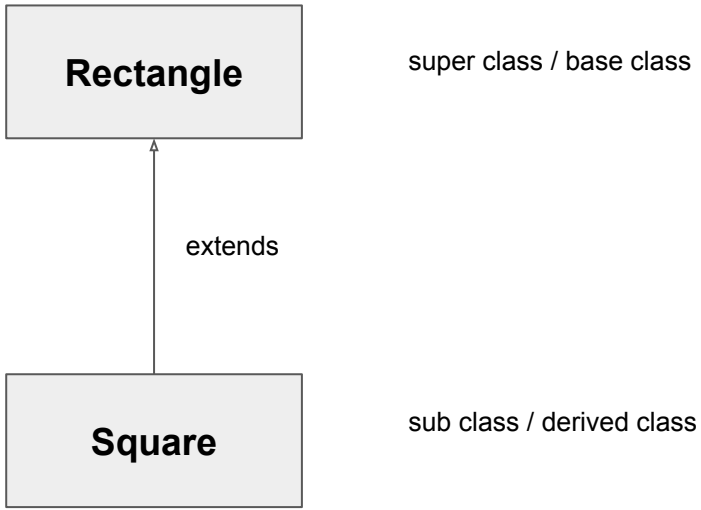
Introduction

- Inheritance relationship (GENERALIZATION & SPECIALIZATION) is natural
  - Example: Rectangle is a special case of parallelogram, and is a general case of Square
- May state:
  - Square inherits Rectangle
  - Square is a sub class, Rectangle is a base class
- Benefits:
  - Subclass reuses the code of base class
  - Subclass has it owns the properties & methods

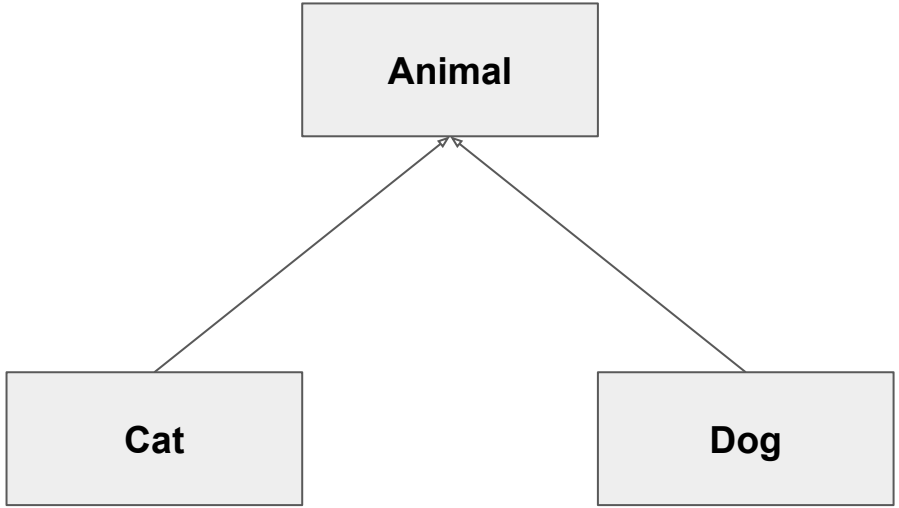
Inheritance

In object-oriented programming, inheritance is the mechanism of basing a class upon another class, retaining similar implementation. Also defined as deriving new classes (subclasses) from existing ones such as super class or base class and then forming them into a hierarchy of classes.

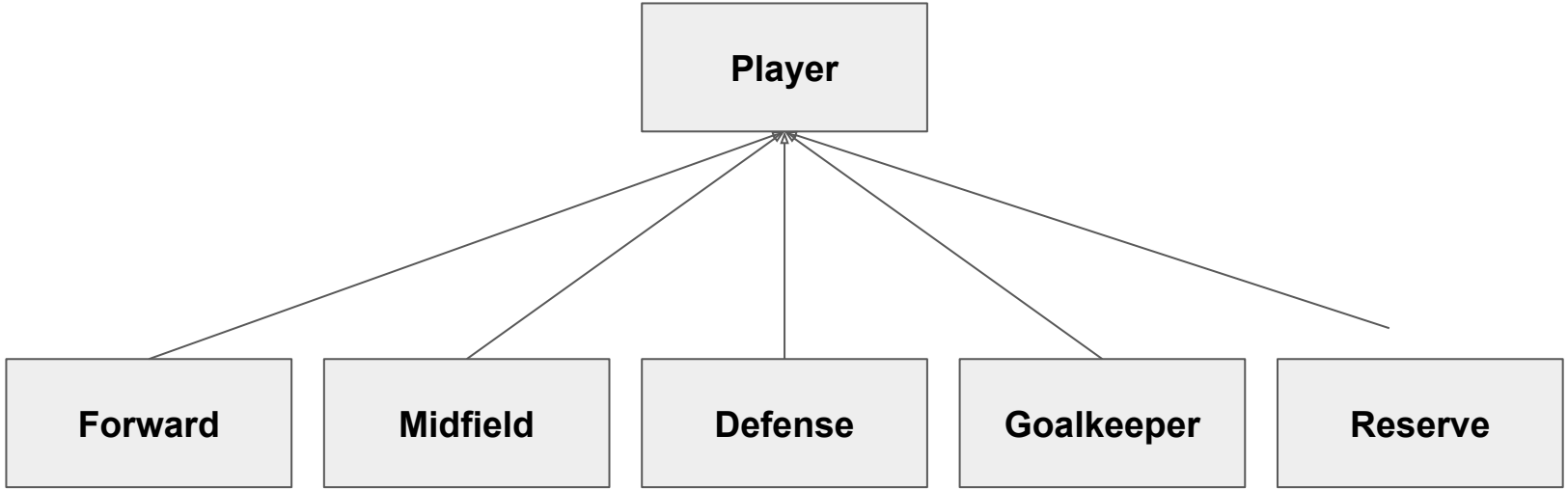
Inheritance



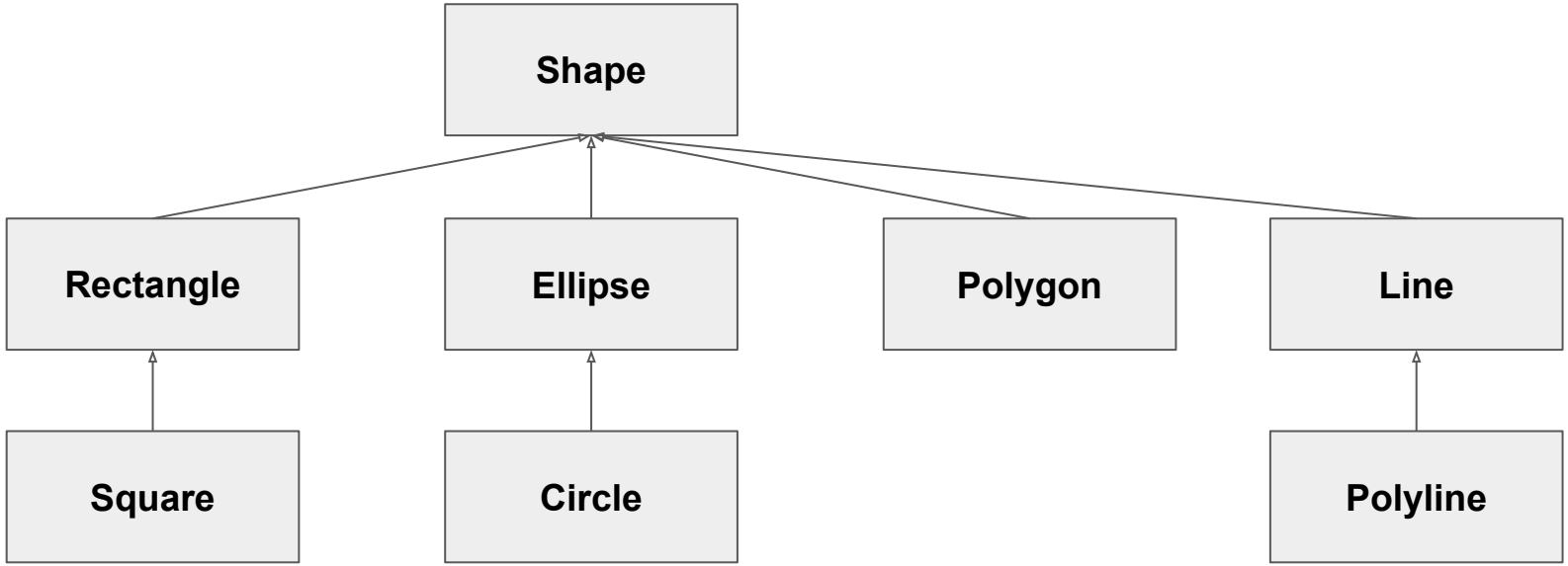
Inheritance



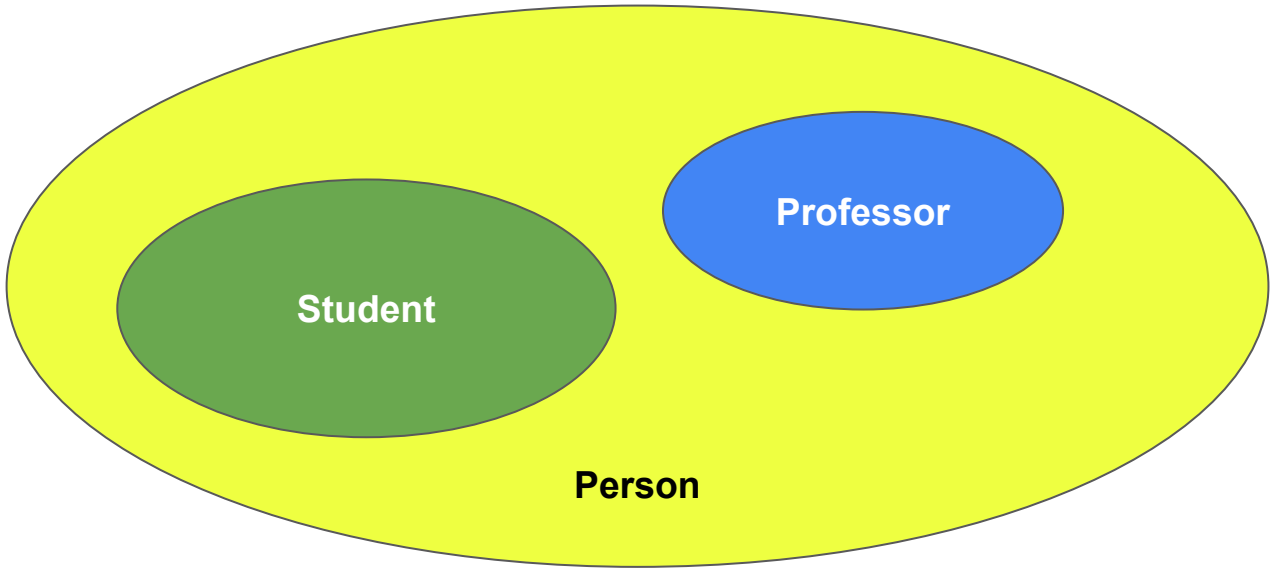
Inheritance



Inheritance



Example: MIT People



https://ocw.mit.edu/courses/6-088-introduction-to-c-memory-management-and-c-object-oriented-programming-january-iap-2010/67b1aec3f2867734ec0fb33034c8b5c8_MIT6_088IAP10_lec05.pdf

Access Modifiers

In C++, there are three access specifiers:

- `public` - members are accessible from outside the class
- `private` - members cannot be accessed (or viewed) from outside the class
- `protected` - members cannot be accessed from outside the class, however, they can be accessed in inherited classes.

Constructor Inheritance (C++)

Constructors are not inherited. They are called implicitly or explicitly by the child constructor.

https://dotnettutorials.net/lesson/how-cpp-constructors-called-in-inheritance/

Polymorphism

Ability of type `A` to appear as and be used like another type `B`

Example: `getArea` of `Rectangle` and `Square`

```
void main() {
  Square s(5);
  Rectangle* p = &s;
  // or Rectangle* p = new Square(5);
  cout << p->getArea();
}
```

Overridden Method

How to call sub class method in polymorphism?

```
void main() {
  Rectangle* p = new Square();
  cout << p->input(); // call Rectangle::input()
}
```

Overridden Method (C++)

Using virtual keyword

```
class Rectangle {
  public:
  virtual void input();
};
```

Virtual Destructor

Always define destructor as `virtual` in (base) classes

Pure Virtual Method and Abstract Class

Pure virtual method has no body and must be implemented in derived class, then that class becomes an abstract class and cannot be instantiated.

```
class Shape {
  public:
  virtual void input() = 0; // Input must be implemented in derived classes
};

void main() {
  Shape s;
  Shape *s = new Shape();
}
```

Inheritance Access Modifier (C++)

Sub class Base class	Inheritance `private`	Inheritance `protected`	Inheritance `public`
Declare `protected`	**`private`** in sub class	**`protected`** in sub class	**`protected`** in sub class
Declare `public`	**`private`** in sub class	**`protected`** in sub class	**`public`** in sub class
Declare `private`	Only use in base class		

Inheritance Abuse

Don’t overuse the inheritance to reuse some properties.

- Example: See `Circle` has a center of `Point2D` type, so allow `Circle` to inherit `Point2D`
- Example: See `Rectangle` has two properties `width` & `height`, and `Square` has only one property, so allow `Rectangle` to inherit `Square` and add one more property.

Type Casting

```
void main() {
  // Upcasting
  Base *pBase;
  Derived d;
  pBase = &d;
  d->print(); // print method must exist in Base class

  // Downcasting
  Derived *pDerived;
  Base *b = new Derived();
  pDerived = (Derived*) b;
}
```

Type Casting (C++)

```
void main() {
  MyClass* m = (MyClass*) ptr;  // C style cast
  MyClass* m = static_cast<MyClass*>(ptr);
  MyClass* m = dynamic_cast<MyClass*>(ptr);
}
```

https://stackoverflow.com/questions/28002/regular-cast-vs-static-cast-vs-dynamic-cast

Type Casting (C++)

- Always use “`dynamic_cast<...>`” to downcast the pointer

Team Project: Milestone 1

- Initialize C++ project and github repository
- Design class diagram and relationship (UML)
- Read **sample.svg** to shape objects of defined classes
- Render sample objects on screen
- Deadline: **16/11/2023**

Team Seminar

- Design pattern topic will be assigned