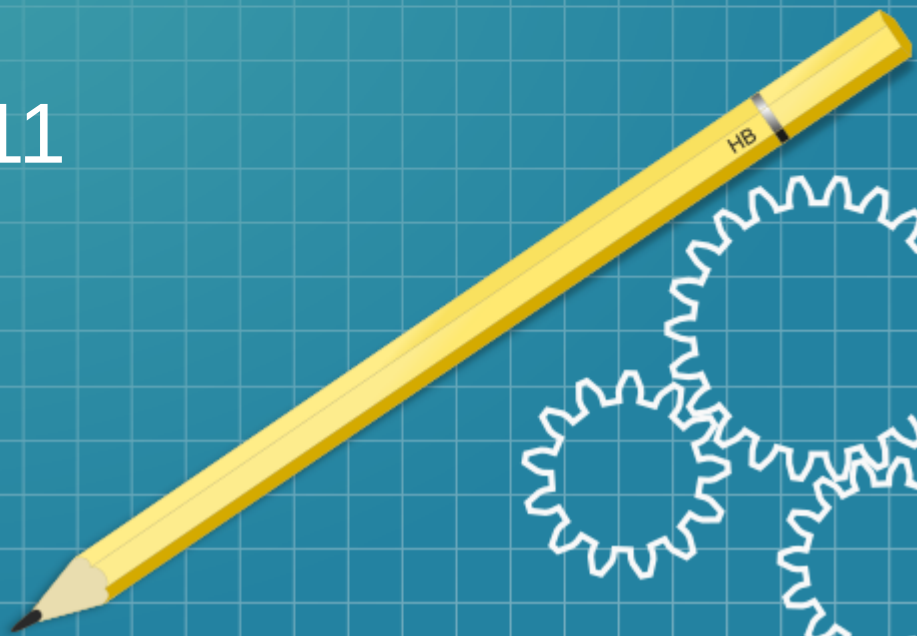
Object-Oriented Programming: Inheritance

Chapter 11



Inheritance: terminology

Inheritance is a technique that allows us to define a new (child / derived) class based upon an existing (parent / base) class.

The child / derived class inherits all of the members of its parent / base class.

Inheritance: terminology

Inheritance is a technique that allows us to define a new (child / derived) class based upon an existing (parent / base) class.

The child / derived class inherits all of the members of its parent / base class.

- It reduces the duplication of existing code, and
- It can save time during program development by taking advantage of proven, high-quality, already defined classes

Inheritance: terminology

The child / derived class may

- *introduce one or more behaviors beyond* those that are inherited (augmenting the parent / base class)
- *specialize one or more of the inherited behaviors* from the parent (provide an alternative definition for the inherited method, i.e. override the original definition)

Inheritance: terminology

The child / derived class may

- *introduce one or more behaviors beyond* those that are inherited (augmenting the parent / base class)
- *specialize one or more of the inherited behaviors* from the parent (provide an alternative definition for the inherited method, i.e. override the original definition)

- these techniques are not necessarily used in isolation

- a single class can serve as parent / base class for many different child / derived classes

- single child / derived class can inherit from multiple parent / base classes (*multiple inheritance*)

is-a vs has-a relationships

The relationship between a parent and child class when using inheritance is often termed as *is-a relationship*, meaning that the object of the child class also can be treated as an object of its parent class.

Example: square is a quadrilateral.
We can define `class Quadrilateral` and it will be the base class of `class Square`.

When one class is implemented using an instance variable of another, it is termed as *has-a relationship*.

Example: `class MixedNumber` can have objects of types `int` and `Rational` as its attributes.

is-a vs has-a relationships

In general, there is not always a clear-cut rule for when to use *inheritance* and when to use *has-a relationship*.

The decision comes down to the number of potentially inherited behaviors that are undesirable versus the number of desirable ones that would need to be explicitly regenerated if using a *has-a relationship*.

is-a vs has-a relationships

In general, there is not always a clear-cut rule for when to use *inheritance* and when to use *has-a relationship*.

The decision comes down to the number of potentially inherited behaviors that are undesirable versus the number of desirable ones that would need to be explicitly regenerated if using a *has-a relationship*.

Base class	Derived classes
`Student`	`GraduateStudent, UndergraduateStudent`
`Shape`	`Circle, Triangle, Rectangle, Sphere, Cube`
`Loan`	`CarLoan, HomeImprovementLoan, MortgageLoan`
`Employee`	`Faculty, Stuff`
`Account`	`CheckingAccount, SavingsAccount`

Inheritance

C++ offers `public`, `protected` and `private` inheritance.

```
class derived-class: access-specifier base-class
```

Inheritance

C++ offers `public`, `protected` and `private` inheritance.

```
class derived-class: access-specifier base-class
```

Public Inheritance: with `public` base class, public members of the base class become public members of the derived class and protected members of the base class become protected members of the derived class. A base class's private members are never accessible directly from a derived class, but can be accessed through calls to the public and protected members of the base class.

Protected Inheritance: with `protected` base class, public and protected members of the base class become protected members of the derived class.

Private Inheritance: with `private` base class, public and protected members of the base class become private members of the derived class.

Inheritance

C++ offers `public`, `protected` and `private` inheritance.

```
class derived-class: access-specifier base-class
```

- When access specifier is not used, it is `private` by default.
- In this chapter we will work only with public inheritance.
- Inheritance relationships form class hierarchies
- A derived class represents a more *specialized group of objects*.

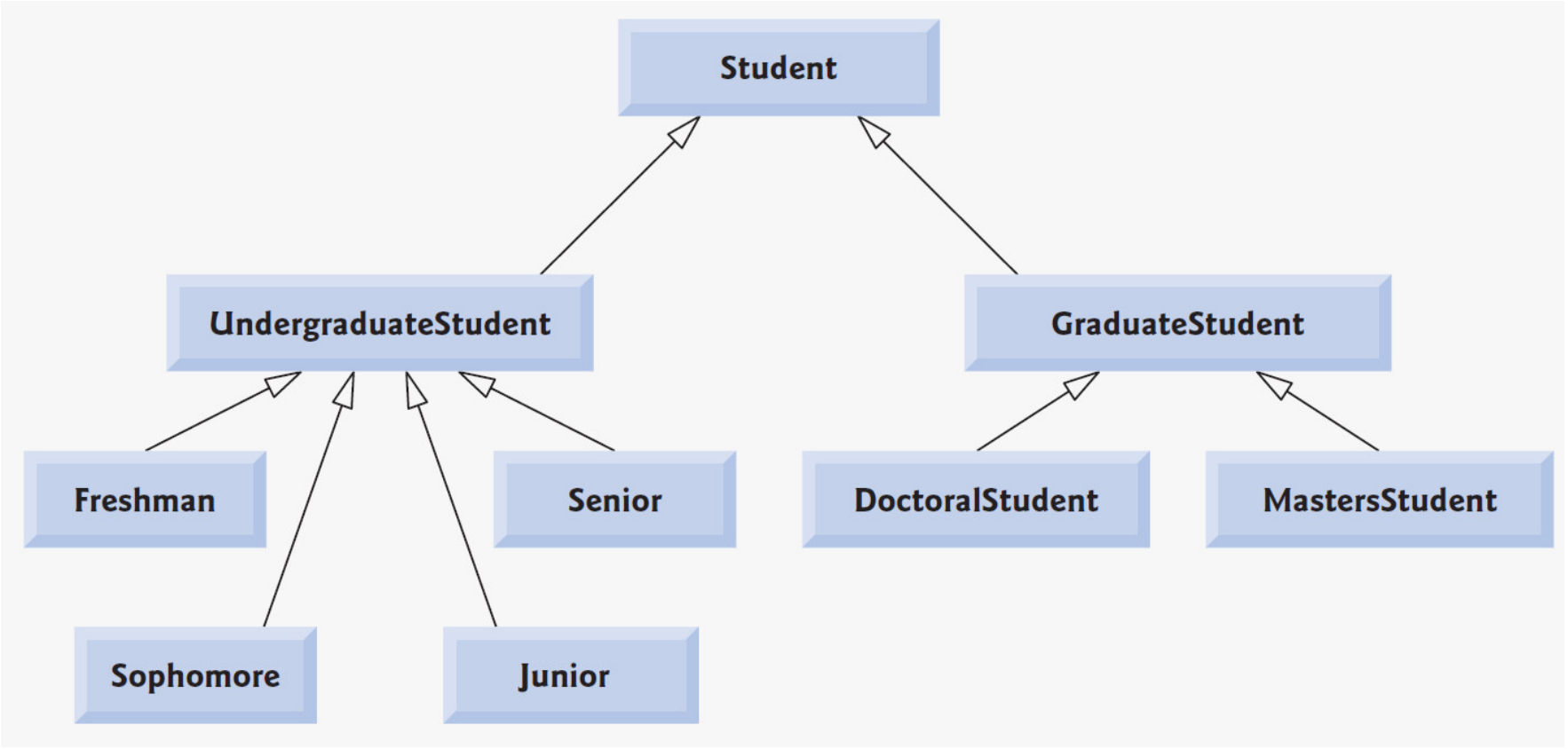
Access types

the different access types according to who can access them:

Access	public	protected	private
same class	yes	yes	yes
child/derived class	yes	yes	no
outside class	yes	no	no

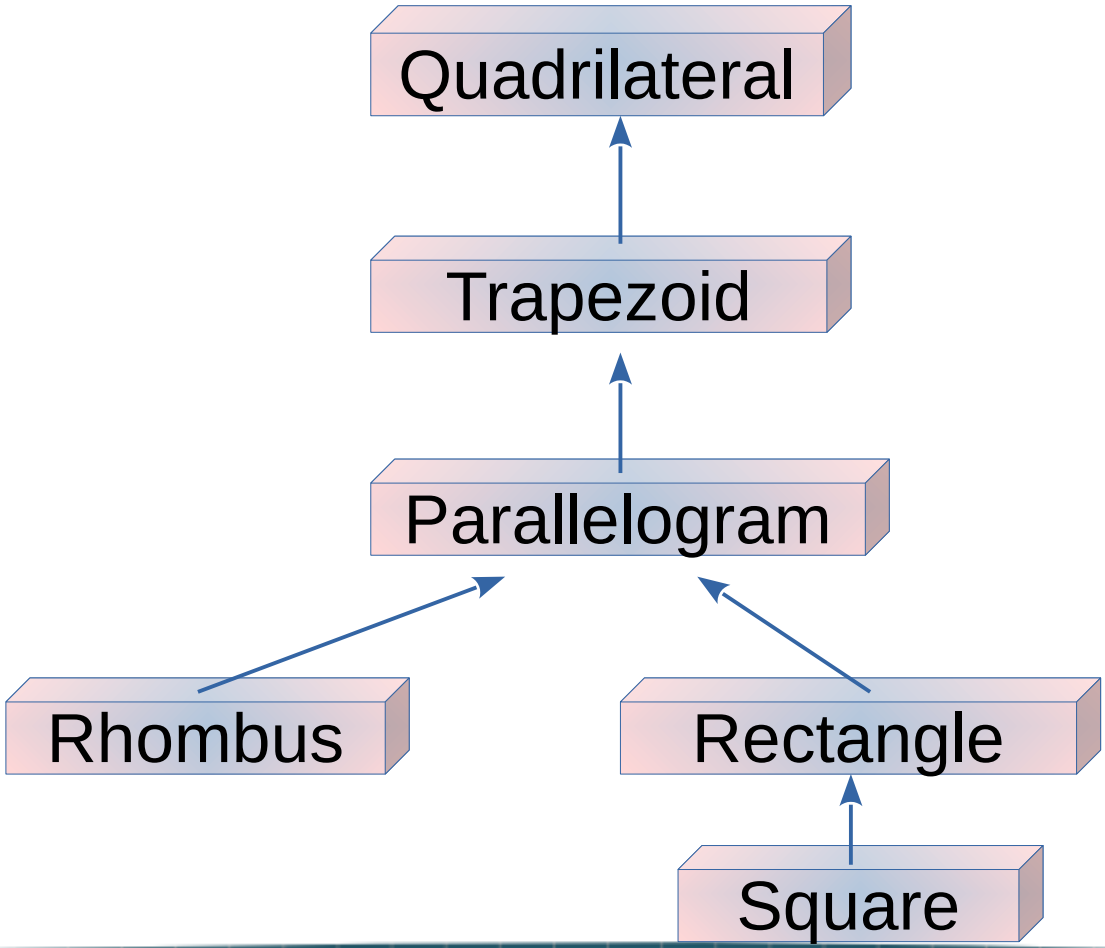
Class Hierarchy

Consider a student inheritance hierarchy at a university (exercise 11.8):



Example

Let's create a quadrilaterals inheritance hierarchy that would include *trapezoids*, *parallelograms*, *rhombuses*, *rectangles* and *squares*. We will use the `class Quadrilateral` as the base/parent class for this hierarchy:



HW assignment

1) Use the classes we defined, write the header and the implementation of the `Rectangle` and `Square` classes

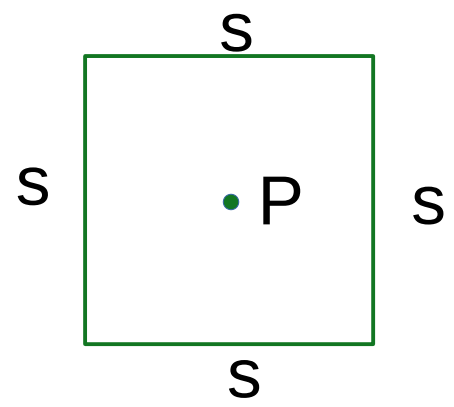
(a) The objects of `class Rectangle` will be created from two points : top left point and bottom right point:

An example of the declaration:

```
Rectangle r1(Point(10,200),Point(80,50));
```

HW assignment

(b) The objects of `class Square` will be created from the center point and the length of the side:



An example of the declaration:

```
Square s1(Point(60,100), 80));
```

(c) For both of the classes, define `Area` member function, overload `Perimeter` member function and overload *output stream operator* `<<` to display the corresponding information about them.

HW assignment

(d) Define the `Point:slope` method. At this moment, the method simply returns 0.

(e) for the `class Rectangle`, add public methods `getLength()` and `getWidth()` that return the length and the width, correspondingly.

(f) for the `class Square`, add public method `getSide()` that returns the length of each side.

HW assignment

Suggested exercises
(*not for grade, but the questions related to these will appear on a quiz or a test*):
1) Chapter 11, *Self-Review Exercises*