

INFO3220 Object Oriented Design

Week 9 tutorial

Tutorial

- Write a program with an inheritance hierarchy
- Base class **Shape**:
 - Abstract function called `area()`
- Two subclasses **Rectangle** and **Square**:
 - Rectangle: width and height
 - Square: side length
- Implement inappropriate get methods

Tutorial

- Should Square inherit from Rectangle?
Should Rectangle inherit from Square?
- Why or why not?

Outline

moo.h

```
class A {  
    void f();  
};
```

```
class B {  
    void g();  
};
```

moo.cc

```
void A::f() {  
    // blah  
}
```

```
void B::g() {  
    // blah  
}
```

Polymorphism

```
vector<Shape*> v;
```

```
Square s(5);
```

```
Rectangle t(4, 6);
```

```
v.add(&s);
```

```
v.add(&t);
```

```
for(vector<Shape*>::iterator i = v.begin(); i  
    != v.end(); ++i) {  
    cout << (*i)->area() << endl;  
}
```

Liskov Substitution Principle

- In Plain English

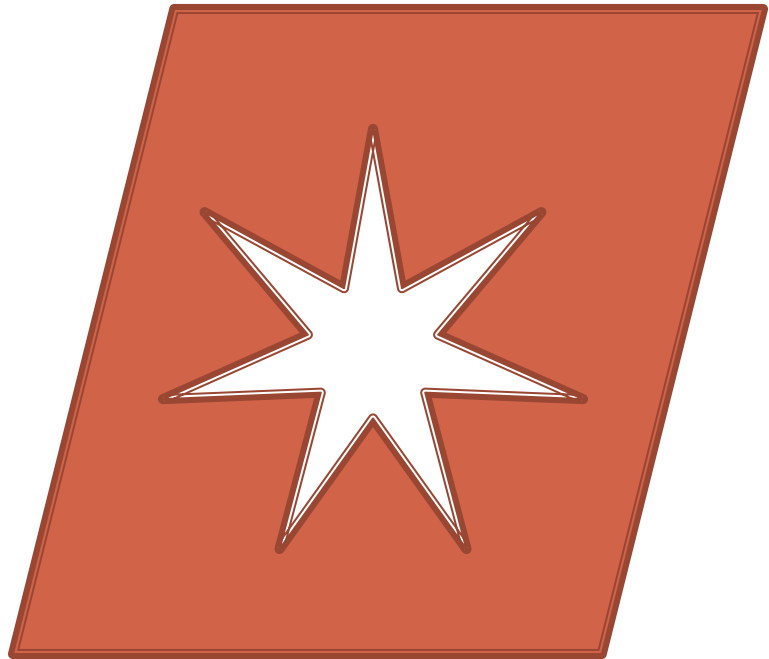
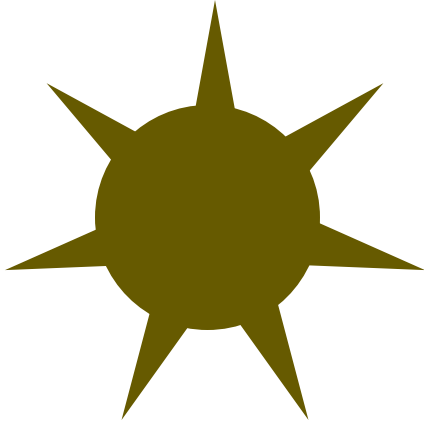
B is a subtype of A.

Then I should be able to use an object of type B wherever I expect an object of type A.

Preconditions

- Should the preconditions of `A::someRoutine()` be **more strict** or **less strict** than the preconditions of `B::someRoutine()`?

Precondition



Postconditions

- Should the postconditions of `A::someRoutine()` be **more strict** or **less strict** than the postconditions of `B::someRoutine()`?

Postcondition

Output

