CASTING OBJECTS

Changing data types

Delroy A. Brinkerhoff

CASTING:
OPERATOR AND EXPRESSIONS

- Casting is done with an operator: `(`*`type`*`)exp` or *`type`*`(exp)`
- The cast operator forms an expression
- Casting does not change the original value
- Example:
  - `double pi = 3.14159;`
  - `int i = (int)pi;`
  - `int i = int(pi);`
  - The value stored in pi is unchanged
  - The value stored in i is 3

CASTING OBJECTS

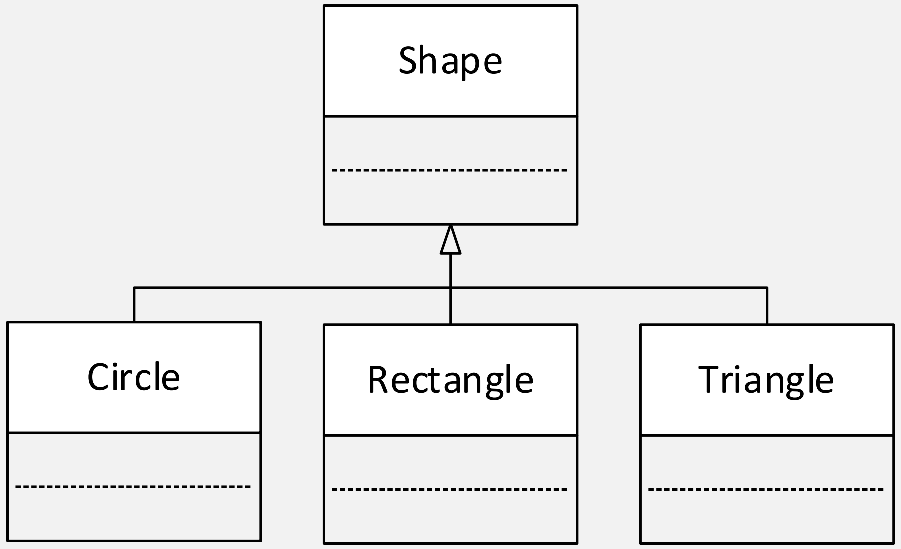
- Casting objects is only possible when the objects are instances of classes related by inheritance

- A Circle "is a" Shape
- A Student "is a" Person
- Does it make sense to cast a Student into a Shape?

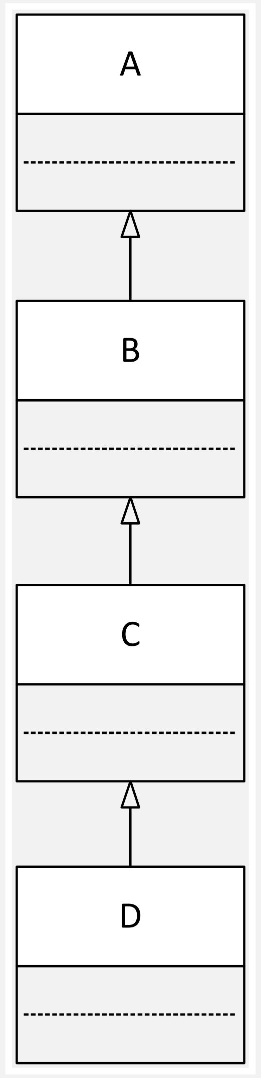
```
friend ostream& operator<<(ostream& out, Student& me)
{
   out << (Person &)me << " " << me.gpa;
   return out;
}
```

UPCASTING

- Upcasting takes place when a subclass object is converted into a superclass object
- Upcasts are safe and take place automatically without casting notation:
  - `Circle* c = new Circle;`
  - `Shape* s = c;`



CASTING OPTIONS



DOWNCASTING

- Downcasting may cause a loss of precision and requires an explicit downcast
- Example:
  - `double pi = 3.14159;`
  - `int i = (int)pi;`

- What are the consequences of downcasting objects?
- Example 1
  - `Circle* c = new Circle;`
  - `Shape* s = c;`
  - `Circle* c2 = (Circle *)s;`
- Example 2
  - `Shape* s = new Shape;`
  - `Circle* c = (Circle *)s;`