Inheritance and Polymorphism

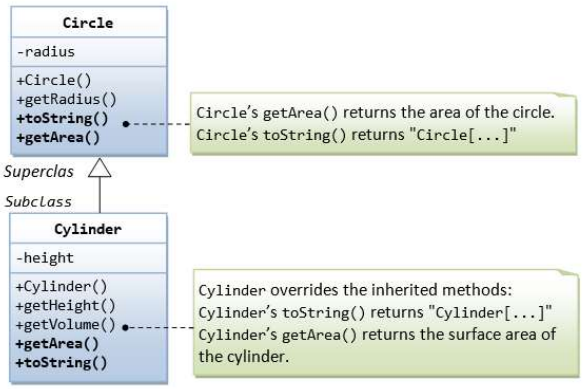
CSCI 2300

Polymorphism and Substitutability

A subclass has all attributes and behaviors of its parent class/superclass

We can **substitute** a subclass instance when a superclass instance is expected

Example: Cylinder is-a Circle (really, it's "more-than-a Circle")



Substitutability in practice

Can create an instance of Cylinder and assign it to a Circle reference

```
Circle c1 = new Cylinder(1, 2);
```

Can invoke all methods defined in the Circle class on c1

```
c1.getRadius();
```

Cannot invoke methods defined in the Cylinder class

```
c1.getHeight(); //COMPILER ERROR
C1.getVolume(); //COMPILER ERROR
```

c1 is a reference to the Circle class but holds an object of subclass Cylinder

`c1` retains its internal identity:

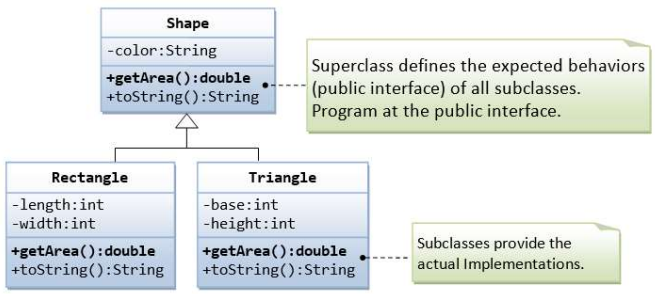
- `c1.getArea(); c1.toString()` invoke the overridden versions defined in `Cylinder`

Polymorphism Example

Powerful tool

Separates interface from implementation

Programmer works at the interface level to design complex systems



Upcasting and Downcasting

Downcasting - revert a substituted instance back to a subclass reference

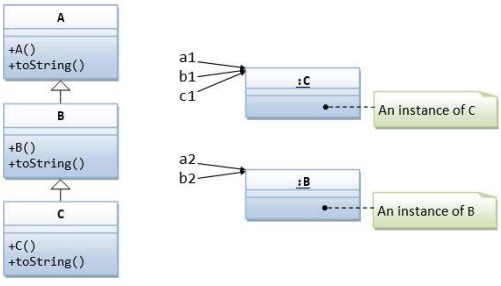
```
Circle c1 = new Cylinder(1, 2);  //auto downcast
Cylinder cy1 = (Cylinder) c1; //upcast needs the casting operator
```

Downcasting uses type-casting operator: (new-type)

Downcasting is not always safe: `ClassCastException` if the instance of to be downcasted does not belong to the correct subclass

A subclass object can be substituted for a superclass object, but not vice versa

Example




```
public class CastingExample{
  public static void main(String []args){
    A a1 = new C(); //upcast
    System.out.println(a1);
    B b1 = (B)a1; //downcast ok
    C c1 = (C)b1; //downcast ok
    A a2 = new B(); //upcast
    System.out.println(a2);
    B b2 = (B)a2; //downcast ok
    C c2 = (C)a2; // ClassCastException
  }
}
```

The 'instanceof' operator

Java binary operator that returns a boolean

- True if a given object is an instance of a particular class
- False otherwise

Syntax:

```
anObject instanceof aClass
```

Example

```
Circle c1 = Cylinder(1, 2);
System.out.println(c1 instanceof Cylinder); // true
if (c1 instanceof Circle){…}
```

Abstract class and abstract method refreshes

A method can be declared **abstract** if is left unimplemented

- The class needs to be declared **abstract** as well

```
public abstract Board(){
   public abstract boolean hasWinner();
}
```

Abstract classes cannot be instantiated

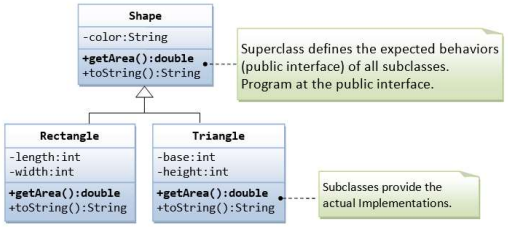
References of abstract type can be used to store objects of subclasses

Exercise 1 - polymorphism

Implement Shape, Rectangle, and Triangle classes, as on the UML diagram

Implement a Test class that does the following in the main method:

```
Shape s = new Shape();
Shape r = new Rectangle();
Shape t = new Triangle();
System.out.println(s);
System.out.println(r);
System.out.println(t)
```



Exercise 2 – abstract classes

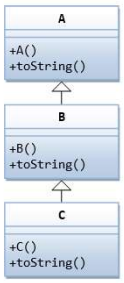
Change your Shape class to be 'abstract' by turning getArea() into an abstract method

Compile your test code

- What happens when you do that?
- Why does that happen?
- Modify your test code to make it work

Exercise 3 – downcasting

Implement classes in the UML diagram



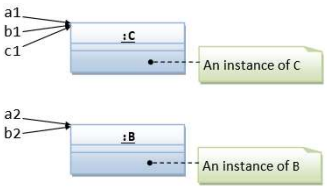
Write a test program that has:

A a1 = new C();

B b1 = new B();

C c1 = (C)b1;

C c2 = (C)a1;



Run the program and observe the ClassCastException exception

Fix the issue by using 'instanceof' operator