Inheritance and Polymorphism

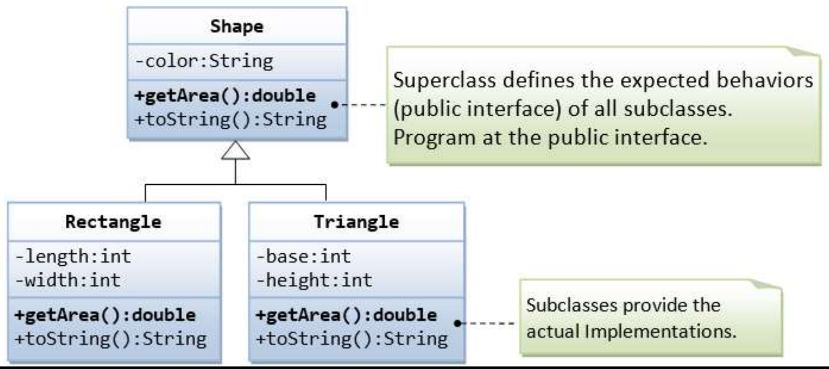
CSCI 2300

Polymorphism and Substitutability

- *Polymorphism* - the condition of occurring in several different forms
- A subclass has all attributes and behaviors of its parent class/superclass
- We can **substitute** a subclass instance when a superclass instance is expected (LSP)
- Overriding methods often breaks LSP

Polymorphism Example

- Powerful tool
- Separates interface from implementation
- Programmer works at the interface level to design complex systems



Polymorphism via abstract classes

- Parent class can be 'abstract' – unimplemented
- Child classes implement the methods declared in the parent class

```
public abstract class Vehicle
{
   public abstract void drive(int miles);
}
```

Abstract classes cannot be instantiated

```
public class ToyCar extends Vehicle
{
   public void drive(int miles)
   {
      //make noises, flash lights
   }
}
```

```
public class Scooter extends Vehicle
{
   public void drive(int miles)
   {
      if (miles > 0 && miles < 50)
        …
   }
}
```

Abstract class and abstract method overview

- A method can be declared **abstract** if is left unimplemented
  - The class needs to be declared **abstract** as well
- Abstract classes cannot be instantiated
- References of abstract type can be used to store objects of subclasses
- A class can be declared abstract even if it has no abstract methods
- A class can have abstract and non-abstract methods

Will this code work?

```
public abstract class Shape
{
   public abstract double getArea();
   String toString(){…}
}
```

```
public class Rectangle extends Shape
{
   public double getArea()
   {
      //calculates and returns area
   }
   String toString()
   {
      //creates and returns a string
   }
}
```

```
1.   Shape s = new Shape();
2.   Shape r = new Rectangle();
3.   System.out.println(s);
4.   System.out.println(r);
```

A. Yes
B. No, because you cannot assign a Rectangle object to a Shape reference (line 2)
C. No, you cannot instantiate Shape class on line 1, because Shape is abstract
D. B and C

```
public abstract class Shape
{
   public abstract double getArea();
   String toString(){…}
}
```

```
public class Rectangle extends Shape
{
   public double getArea()
   {
      //calculates and returns area
   }
   String toString()
   {
      //creates and returns a string
   }
}
```

Will this code work?

```
1. void displayShape(Shape s)
2. {
3.    System.out.println(s)
4. }
 ...
5. Rectangle r = new Rectangle()
6. displayShape(r);
```

A. Yes

B. No, because you cannot pass a Rectangle object to a Shape argument (line 6)

C. No, you cannot use Shape as an argument, because Shape is abstract (line 1)

D. B and C

Lab 6 prep

- Review Cylinder example (on class web page) from Jan 30
- Identify problems in the design:
  - Where does this design break LSP?

Lab 6

- Fix the problems in the Cylinder example using abstract class.