CSE 215L: Programming Language II Lab
Section: 7

Fall 2020

Abstract Class: An abstract class is a class that is declared abstract- it may or may not include abstract methods. Abstract classes cannot be instantiated, but they can be subclassed.

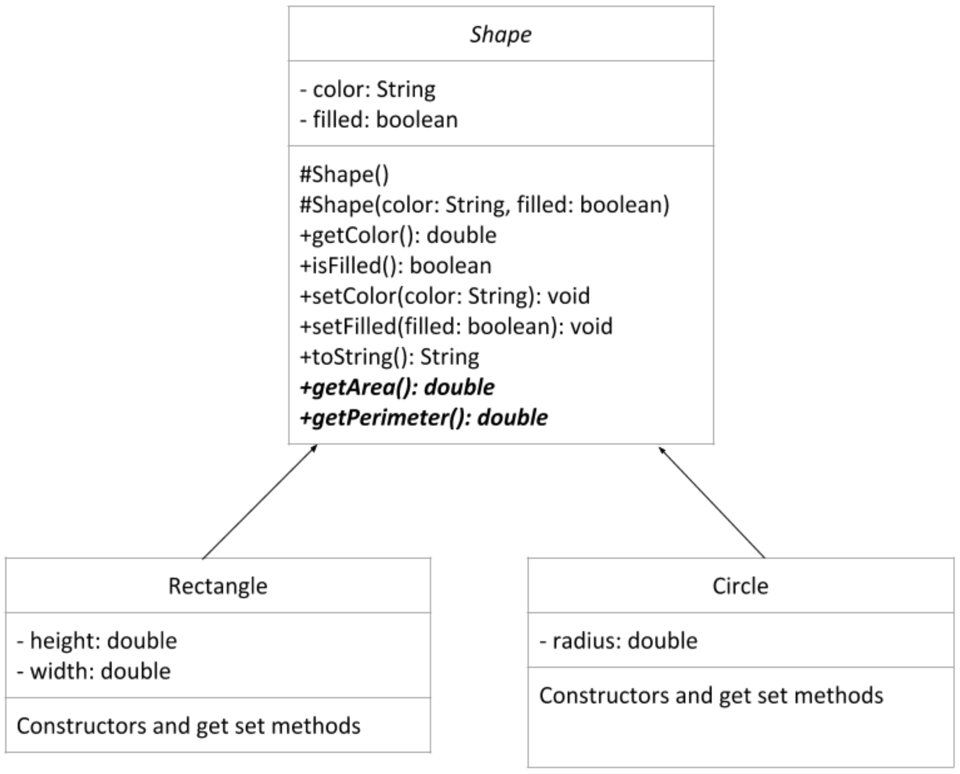
- An abstract method is a method that is declared without any implementation. For example:

  abstract double getArea(); //no method body

- If a class includes abstract methods, then the class must be declared as abstract, as in:

  abstract public class Shape {
  //properties and behaviors
  abstract double getArea();
  }

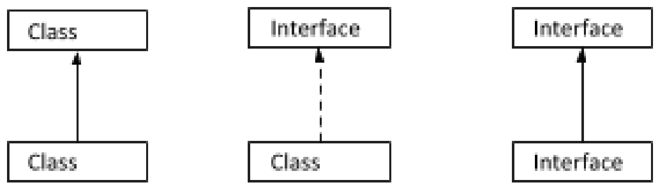
- When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.



Shape
- color: String
- filled: boolean
#Shape()
#Shape(color: String, filled: boolean)
+getColor(): double
+isFilled(): boolean
+setColor(color: String): void
+setFilled(filled: boolean): void
+toString(): String
+getArea(): double
+getPerimeter(): double
Rectangle
- height: double
- width: double
Constructors and get set methods
Circle
- radius: double
Constructors and get set methods

Interface: A java interface is a bit like a class, except a java interface can only contain method signatures and fields. A java interface cannot contain an implementation of the methods, only the signatures: name, parameters and expressions.

- A class that implements an Interface must implement all the methods declared in the Interface. The methods must have the exact same signatures as declared in the interface. The class does not need to implement/declare the properties of the interface.
- Relationship between classes and interfaces:



Consider using abstract classes if any of these statements apply to your situation:	Consider using interfaces if any of these statements apply to your situation:
1. You want to share code among several closely related classes. 2. You expect that classes that extend your abstract class have many common methods or fields or require access modifiers other than public (such as protected and private). 3. You want to declare non-static or non-final fields. This enables you to define methods that can access and modify the state of the object to which they belong.	1. You expect that unrelated classes would implement your interface. For example, the interfaces Comparable and Cloneable are implemented by many unrelated classes. 2. You want to specify the behavior of a particular data type, but not concerned about who implements its behavior. 3. You want to take advantage of multiple inheritances.