ABSTRACTION & INTERFACE

Additional References:

https://www.tutorialspoint.com/java/java_abstraction.htm

Why Abstraction?

- To hide the implementation details from the user, only the functionality will be provided to the user. In other words, the user will have the information on what the object does instead of how it does it.

Abstract Class

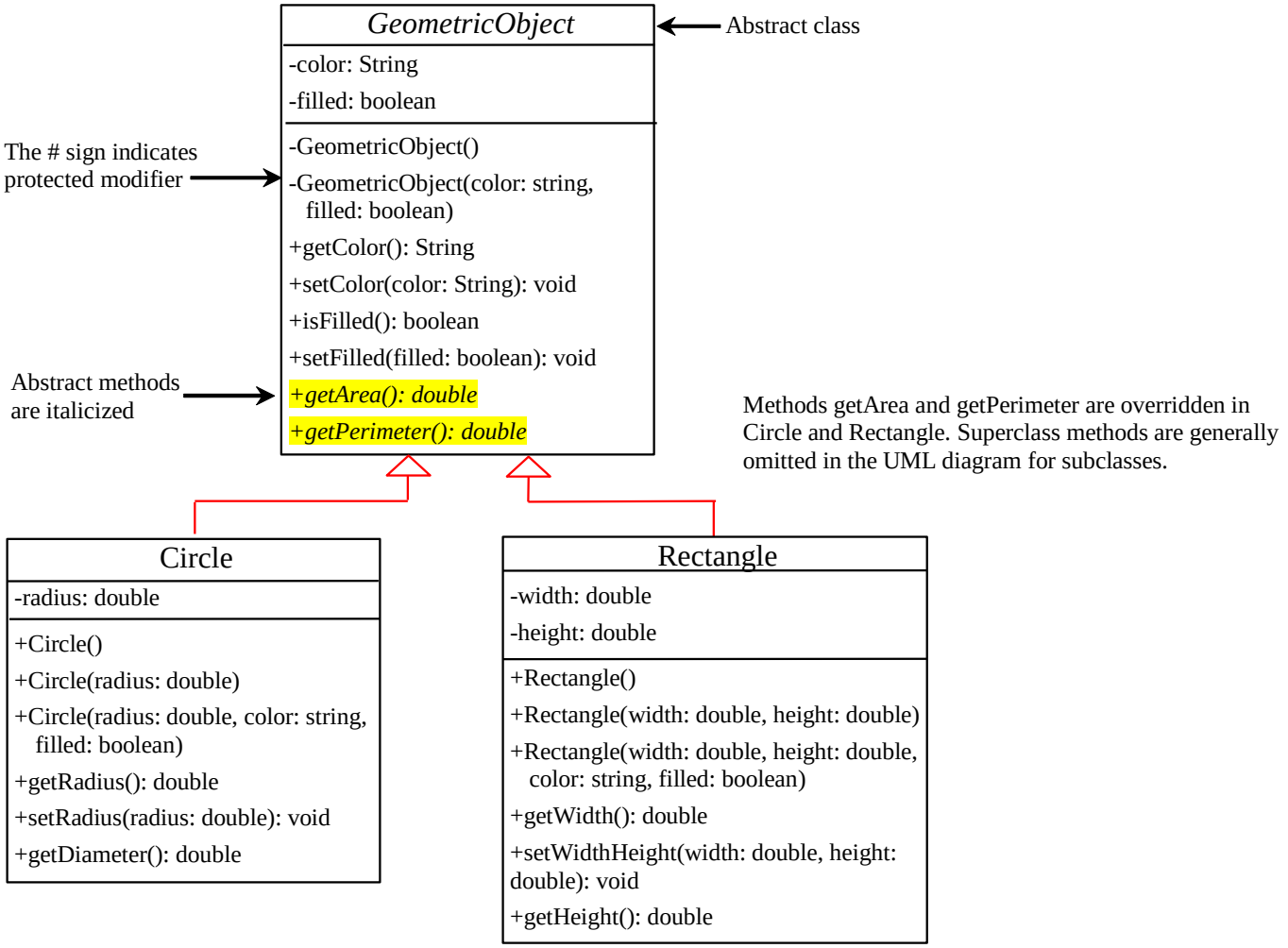
- Abstract classes may or may not contain *abstract methods*, i.e., methods without body (public void get();)
- But, if a class has at least one abstract method, then the class **must** be declared abstract.
- If a class is declared abstract, it cannot be instantiated.
- To use an abstract class, you have to inherit it from another class, provide implementations to the abstract methods in it.
- If you inherit an abstract class, you have to provide implementations to all the abstract methods in it.

Abstract Method

If you want a class to contain a particular method but you want the actual implementation of that method to be determined by child classes, you can declare the method in the parent class as an abstract.

- `abstract` keyword is used to declare the method as abstract.
- You have to place the `abstract` keyword before the method name in the method declaration.
- An abstract method contains a method signature, but no method body.
- Instead of curly braces, an abstract method will have a semi colon (;) at the end.

Abstract Classes and Abstract Methods



GeometricObject Class

```
public abstract class GeometricObject {
  private String color;
  private boolean filled;

  public GeometricObject() {
          this.color = "blue";
  }
  public GeometricObject(String color, boolean filled) {
          this.color = color;
          this.filled = filled;
  }

  public String getColor() {
          return color;
  }
  public void setColor(String color) {
          this.color = color;
  }
  public boolean isFilled() {
          return filled;
  }
  public void setFilled(boolean filled) {
          this.filled = filled;
  }
  public abstract double getArea();
  public abstract double getPerimeter();
}
```

Circle Class

```
public class Circle extends GeometricObject {
  private double radius;

  public Circle() {
          this(1.0);
  }
  public Circle(double radius) {
          this(radius, "white", true);
  }
  public Circle(double radius, String color, boolean filled) {
          super(color, filled);
          this.radius = radius;
  }
  public double getRadius() {
          return radius;
  }
  public void setRadius(double radius) {
          this.radius = radius;
  }
  public double getArea() {
          return radius*radius*Math.PI;
  }
  public double getPerimeter() {
          return 2*radius*Math.PI;
  }
  public double getDiameter(){
          return 2*radius;
  }
}
```

Rectangle Class

```
public class Rectangle extends GeometricObject {
  private double width;
  private double height;
  public Rectangle() {
          this(1.0, 1.0);
  }
  public Rectangle(double width, double height) {
          this(width, height,"green", true);
  }
  public Rectangle(double width, double height, String color, boolean filled) {
          super(color, filled);
          this.width = width; this.height = height;
  }
  public double getWidth() {
          return width;
  }
  public double getHeight() {
          return height;
  }
  public void setWidthHeight(double width, double height) {
          this.width = width; this.height = height;
  }
  public double getArea() {
          return width*height;
  }
  public double getPerimeter() {
          return (2*width)+(2*height);
  }
}
```

Interface

- An interface is a classlike construct that contains only constants and abstract methods.

- In many ways, an interface is similar to an abstract class, but the intent of an interface is to specify behavior for objects.

Interface

- An interface is similar to a class in the following ways
  - An interface can contain any number of methods.
  - An interface is written in a file with a **.java** extension, with the name of the interface matching the name of the file.
  - The byte code of an interface appears in a **.class** file.
  - Interfaces appear in packages, and their corresponding bytecode file must be in a directory structure that matches the package name.

Interface

- However, an interface is different from a class in several ways, including
  - You cannot instantiate an interface.
  - An interface does not contain any constructors.
  - All of the methods in an interface are abstract.
  - An interface cannot contain instance fields. The only fields that can appear in an interface must be declared both static and final.
  - An interface is not extended by a class; it is implemented by a class.
  - An interface can extend multiple interfaces.

Define an Interface

- To distinguish an interface from a class, Java uses the following syntax to define an interface:

```
public interface InterfaceName {
  constant declarations;
  method signatures;
}
```

- Example

```
public interface GeometricObjectInterface {
  //describe how to calculate area
  public double getArea();
}
```

Define an Interface

- The `interface` keyword is used to declare an interface. Here is a simple example to declare an interface

- Interfaces have the following properties
  - An interface is implicitly abstract. You do not need to use the `abstract` keyword while declaring an interface.
  - Each method in an interface is also implicitly abstract, so the abstract keyword is not needed.
  - Methods in an interface are implicitly public.

Implement an Interface

- A class uses the `implements` keyword to implement an interface. The implements keyword appears in the class declaration following the extends portion of the declaration.
- Example

```
public class Circle2 implements GeometricObjectInterface
   {
       public double getArea(){
             return radius*radius*Math.PI;
       }
   }
```

Implement an Interface

- When implementation interfaces, there are several rules −
  - A class can implement more than one interface at a time.
  - A class can extend only one class, but implement many interfaces.
  - An interface can extend another interface, in a similar way as a class can extend another class.

GeometricObjectInterface Class

```
public interface GeometricObjectInterface {

	public double getArea();

	public double getPerimeter();

}
```

Circle2 Class

```
public class Circle2 implements GeometricObjectInterface {
  private double radius;
  private final double PI = Math.PI;
  private String color;
  private boolean filled;
  public String getColor(){
          return color;
  }
  public void setColor(String color){
          this.color = color;
  }
  public boolean isFilled(){
          return filled;
  }
  public void setFilled(boolean filled){
          this.filled = filled;
  }
  public double getArea(){
          return radius*radius*Math.PI;
  }
  public double getPerimeter(){
          return 2*radius*Math.PI;
  }
}
```

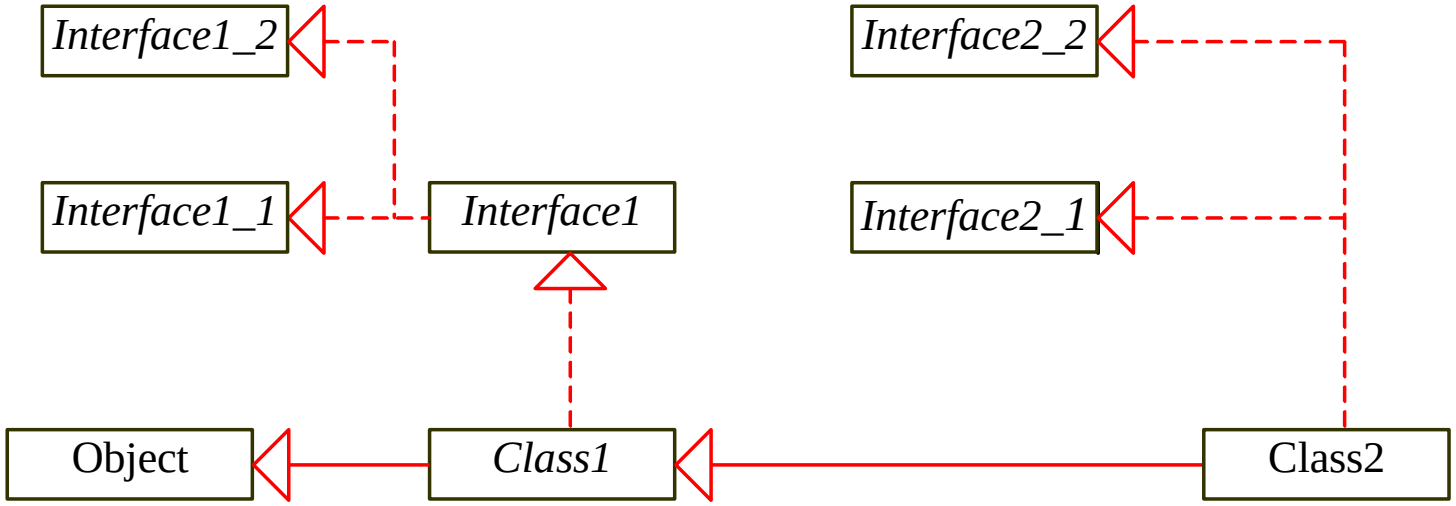
Interfaces vs. Abstract Classes

- In an interface, the data must be constants; an abstract class can have all types of data.
- Each method in an interface has only a signature without implementation; an abstract class can have concrete methods.

	Variables	Constructors	Methods
Abstract class	No restrictions	Constructors are invoked by subclasses through constructor chaining. An abstract class cannot be instantiated using the new operator.	No restrictions.
Interface	All variables must be public static final	No constructors. An interface cannot be instantiated using the new operator.	All methods must be public abstract instance methods

Interfaces vs. Abstract Classes, cont.

- All classes share a single root, the Object class, but there is no single root for interfaces. Like a class, an interface also defines a type. A variable of an interface type can reference any instance of the class that implements the interface. If a class extends an interface, this interface plays the same role as a superclass. You can use an interface as a data type and cast a variable of an interface type to its subclass, and vice versa.



Whether to use an interface or a class?

- Abstract classes and interfaces can both be used to model common features. How do you decide whether to use an interface or a class?
  - In general, a strong is-a relationship that clearly describes a parent-child relationship should be modeled using classes. For example, a staff member is a person. So their relationship should be modeled using class inheritance.
  - A weak is-a relationship, also known as an is-kind-of relationship, indicates that an object possesses a certain property. A weak is-a relationship can be modeled using interfaces. For example, all strings are comparable, so the String class implements the Comparable interface.