



# Abstract Classes & Interfaces

# Learning Objectives

- Learn and use **Java Abstraction**
- Learn and use **Java Interface**



# Agenda

Do Now

Mini Lesson:

- Abstract Classes
- Interfaces

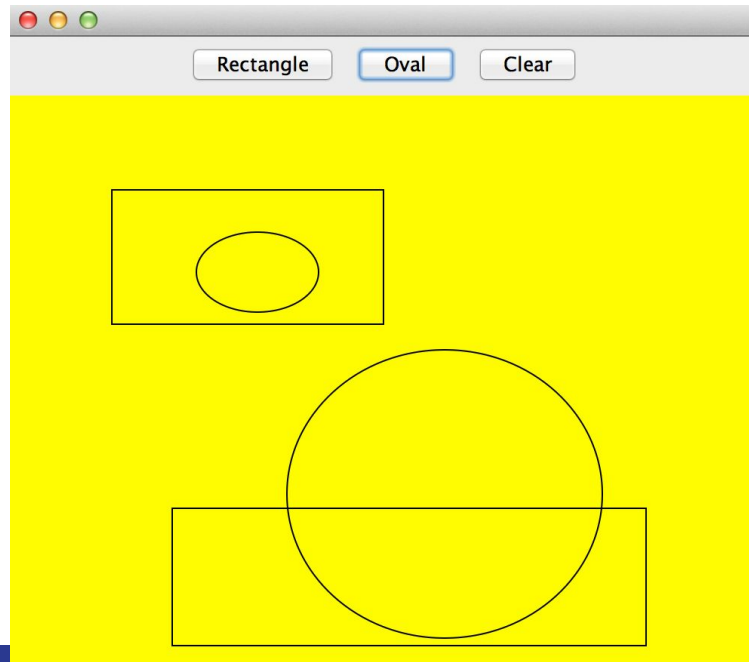
Practice - Questions

Exit-Ticket

---

# Do Now

If you were creating a program to allow users to draw rectangles and ovals by clicking the mouse at a location and then dragging and releasing to define the width, what classes would you need?



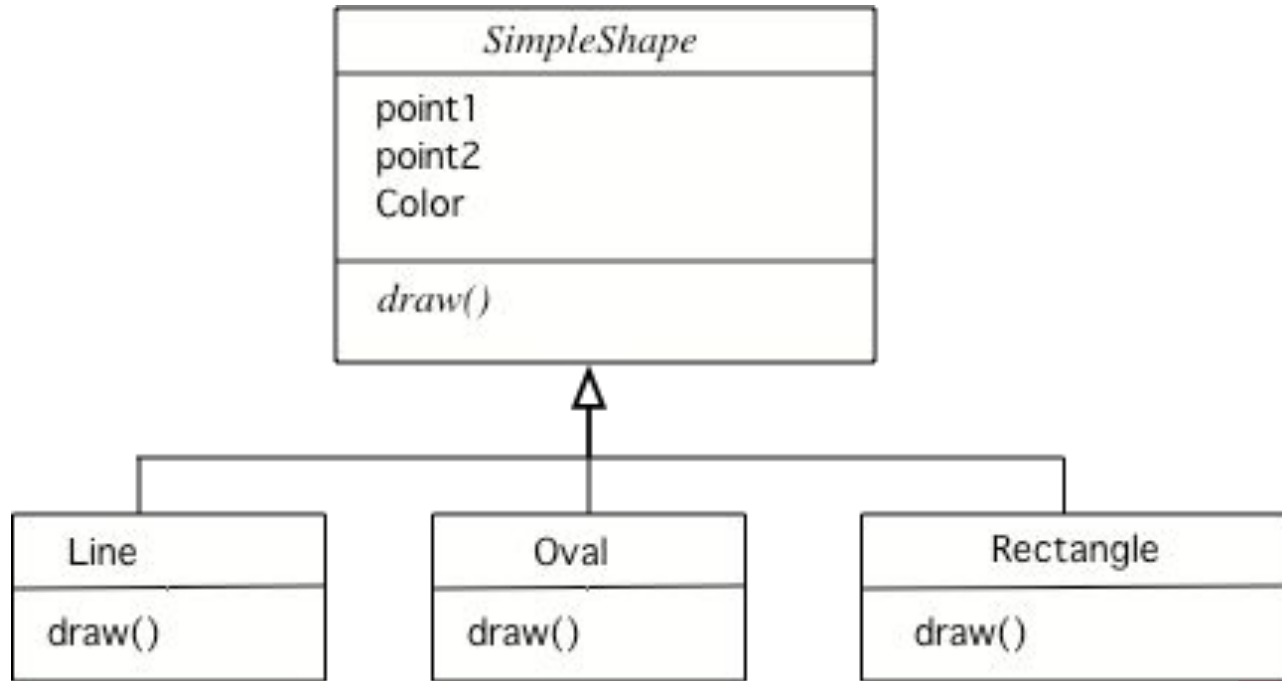
# Possible Solution

- Create two classes: Rectangle and Oval
- These two classes are shapes that can be defined by two points. Can we create a Shape class to define those two points and add other shape attributes like color for example? What would it look like? How would you draw the shapes? Where the drawing methods should be implemented?

Yes, we can have a Shape object, but we don't know what Shape looks like we can make the class abstract which means you cannot create any objects of that type.



# Abstraction



# Abstraction

Data **abstraction** is the process of hiding certain details and showing only essential information to the user.

Abstraction can be achieved with **abstract classes** or **interfaces**.



# Abstraction

The **abstract keyword** is a non-access modifier, used for classes and methods:

- **Abstract class (restricted class)**

Cannot be used to create objects

Must be inherited from another class

- **Abstract method**

Can only be used in an abstract class

Does not have a body. The body is provided by the subclass (inherited from).





# Example

```
abstract class Animal {  
    public abstract void animalSound();  
    public void sleep() {  
        System.out.println("Zzz");  
    }  
}
```

```
Animal myAnimal = new Animal();  
  
// will generate an error
```

ERROR



```
// Abstract class

abstract class Animal {

    // Abstract method

    //(does not have a body)

    public abstract void animalSound();

    // Regular method

    public void sleep() {

        System.out.println("Zzz");

    }

}
```

```
// Subclass (inherit from Animal)

class Dog extends Animal {

    public void animalSound() {

        // The body of animalSound() is provided here

        System.out.println("The dog says: woof woof");

    }

}
```

```
class Driver {

    public static void main(String[] args) {


        Dog myDog = new Dog(); // Create a Dog object

        myDog.animalSound();

        myDog.sleep();

    }

}
```

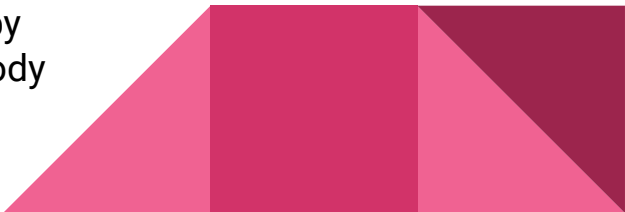


# Interfaces

An **interface** is a completely "abstract class" that is used to group related methods with empty bodies:

```
// interface
interface Animal {
    public void animalSound(); // interface method (does not have a body)
    public void run(); // interface method (does not have a body)
}
```

To access the interface methods, the interface must be "implemented" by another class with the **implements** keyword (instead of **extends**). The body of the interface method is provided by the "implement" class.



```
// Interface
interface Animal {
    public void animalSound(); // interface method (does not have a body)
    public void sleep(); // interface method (does not have a body)
}
```

```
// Dog "implements" the Animal interface
class Dog implements Animal {
    public void animalSound() {
        // The body of animalSound() is provided
        here
        System.out.println("The dog says: woof
        woof");
    }
    public void sleep() {
        // The body of sleep() is provided here
        System.out.println("Zzz");
    }
}
```

```
class Main {
    public static void main(String[] args)
    {
        // Create a Dog object
        Dog myDog = new Dog();
        myDog.animalSound();
        myDog.sleep();
    }
}
```