159 XTENDED

A WEEKLY REVIEW

INHERITANCE RECAP

Pull up **Socrative.com**

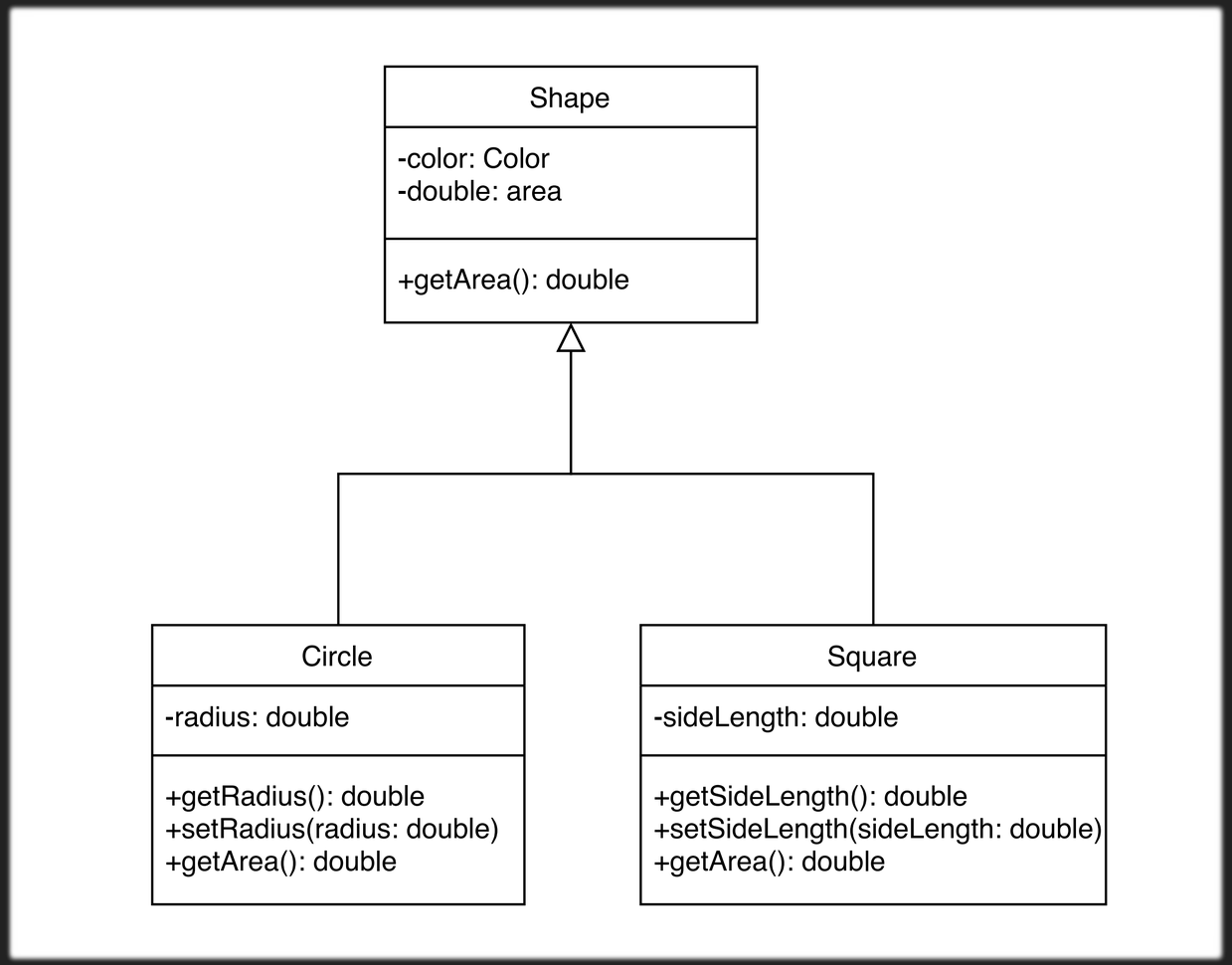
and join room **XTEND159**

WHAT IS INHERITANCE?

- Inheritance allows us to reuse classes.
- When classes have attributes and methods in common, we can pull those things out in to a generic **Superclass**.
- This eliminates redundancy in our code, makes our system simpler, and makes our code easier to use, maintain, and update.

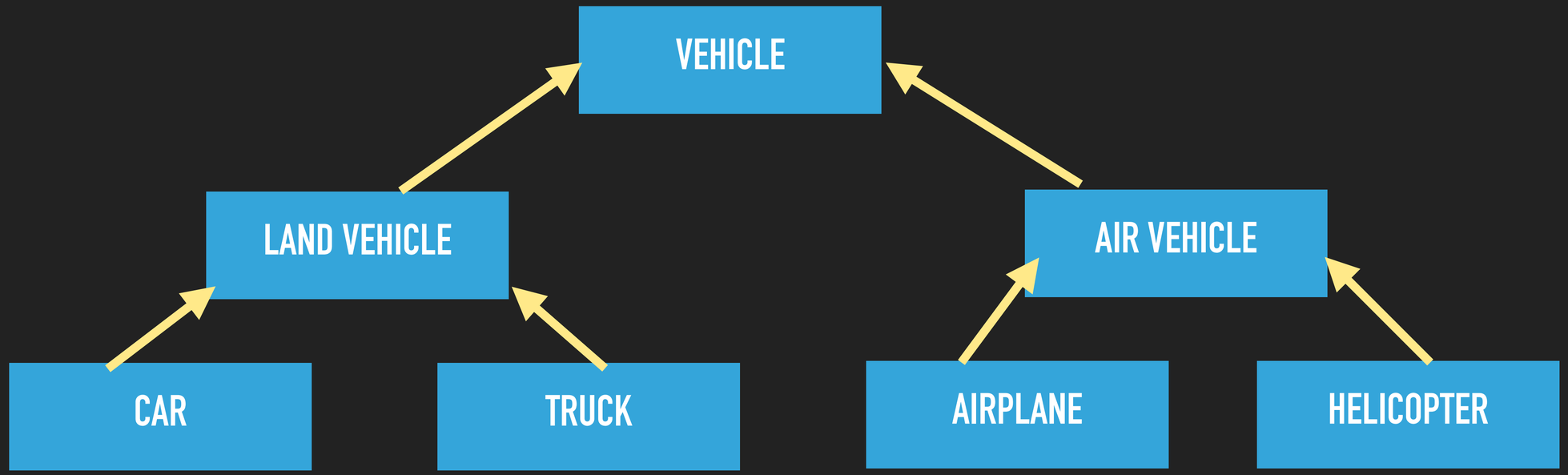
WHAT IS INHERITANCE?

Circle	Square
-radius: double -color: Color -double: area	-sideLength: double -color: Color -double: area
+getRadius(): double +setRadius(radius: double) +getArea(): double	+getSideLength(): double +setSideLength(sideLength: double) +getArea(): double



- On the left, we see that our Circle and Square classes have many methods and attributes in common.
- We can pull those up in to a generic Shape superclass.

RELATIONSHIPS

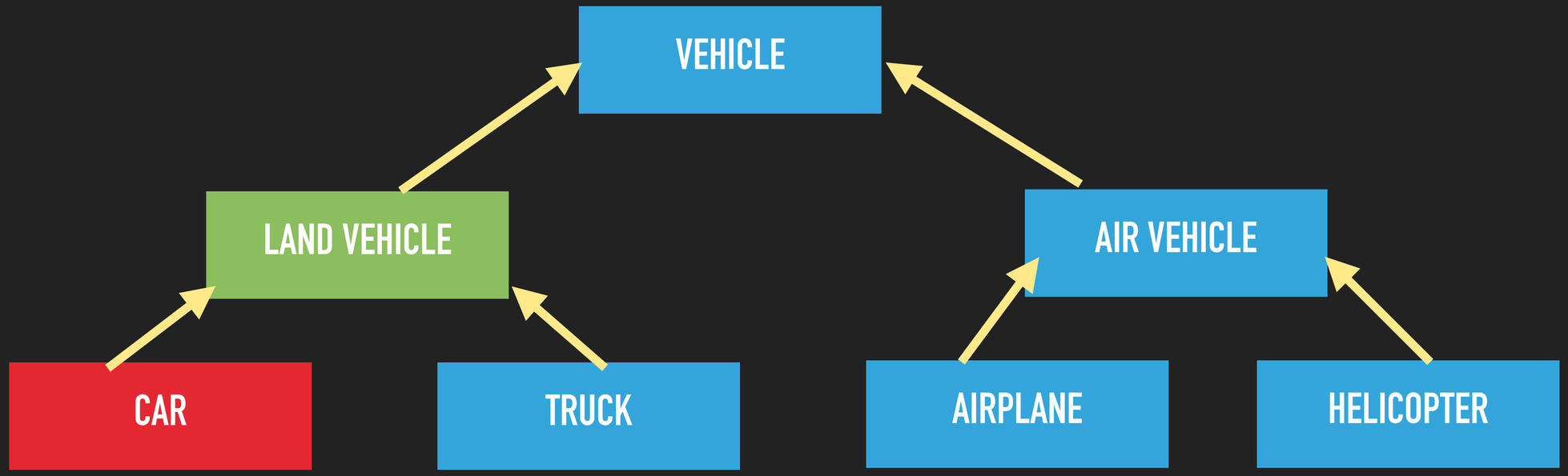


A Truck is a Land Vehicle.

More Specific
Less Generic

Less Specific
More Generic

RELATIONSHIPS



- A superclass is a more generalized class.
- A subclass is a more specialized class.

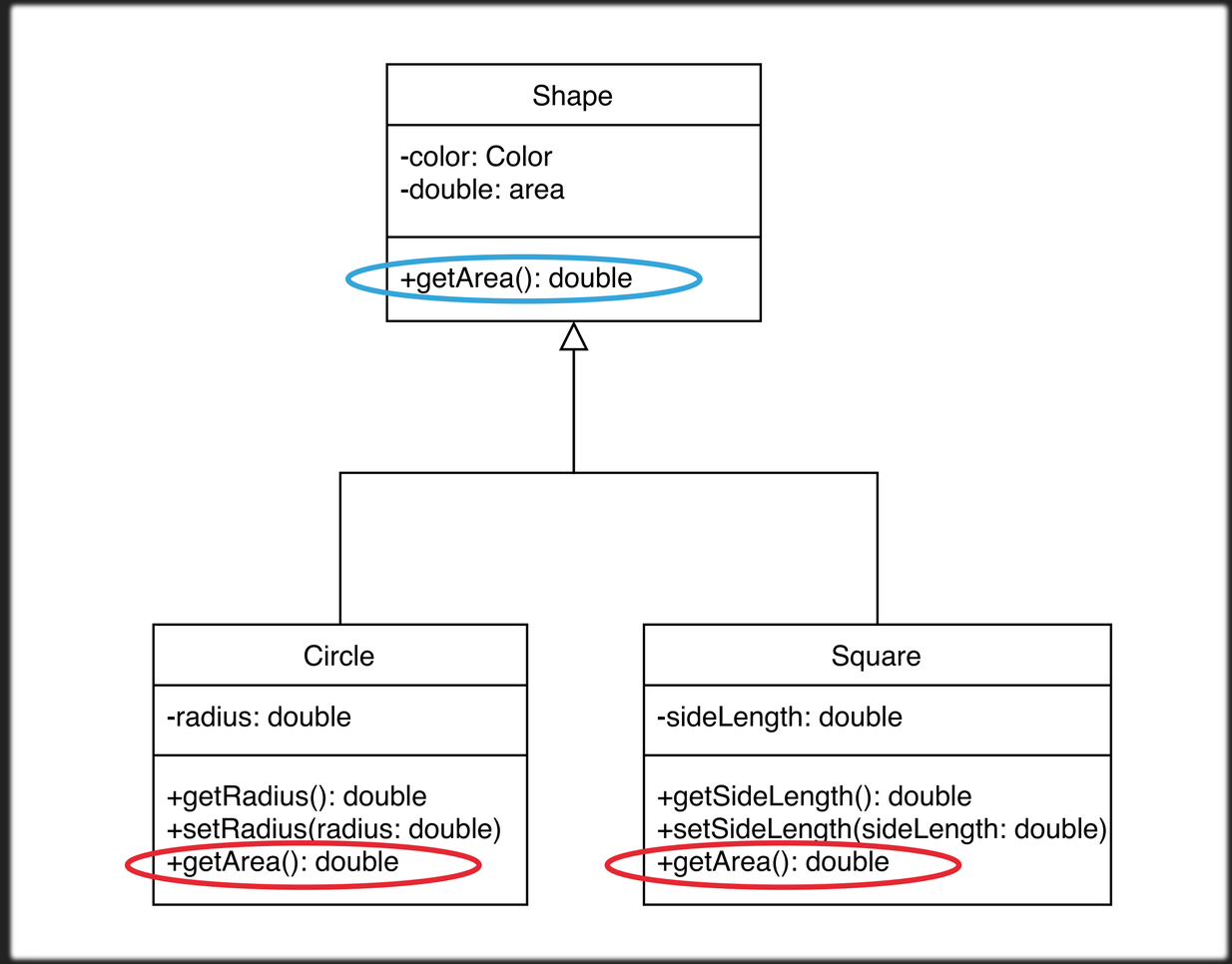
WHAT EXACTLY DOES A SUBCLASS INHERIT FROM A SUPERCLASS?

Pull up **Socrative.com**

and join room **XTEND159**

OVERRIDING METHODS

- A subclass inherits methods from a superclass.
- Sometimes, it is necessary for a subclass to modify the implementation of a superclass method.
- This is referred to as *Method Overriding*.



- Here, `getArea()` in Circle and Square **override** `getArea()` in Shape.

IN YOUR OWN WORDS, DESCRIBE WHAT IT MEANS FOR A SUBCLASS METHOD TO OVERRIDE A SUPERCLASS METHOD.

Pull up **Socrative.com**

and join room **XTEND159**

159 XTENDED

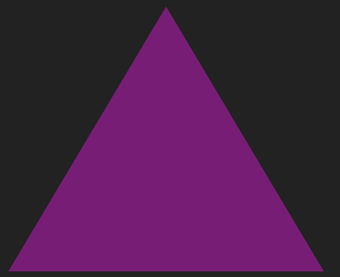
A WEEKLY REVIEW

ABSTRACT CLASSES AND INTERFACES REVIEW

Pull up **Socrative.com**

and join room **XTEND159**

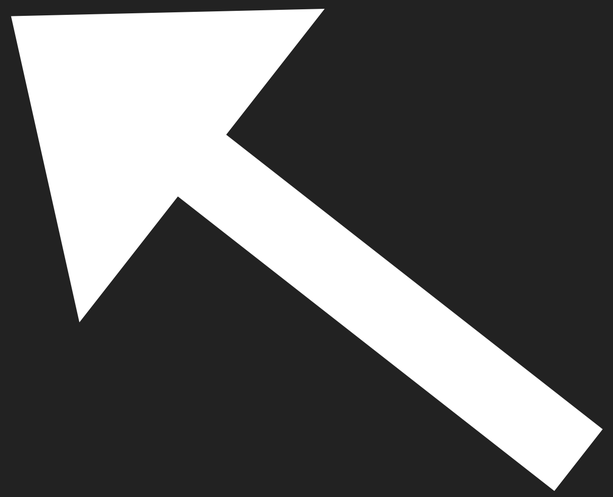





TODAY, WE'RE BUILDING THE

JMU CS SHAPE EMPORIUM





WHAT ARE SOME THINGS THAT ALL SHAPES HAVE IN COMMON?
THINK ABOUT PHYSICAL PROPERTIES OR DIMENSIONS.

Pull up **Socrative.com**

and join room **XTEND159**

YOU MAY HAVE THOUGHT OF:

SIDES

PERIMETER

COLOR

AREA

OR OTHER THINGS.

LET'S LOOK AT A CLASS.

```
public class Shape
{
    // Instance Variables
    private int sides;
    private Color color;

    public Shape(int sides, Color color)
    {
        this.sides = sides;
        this.color = color;
    }

    public Shape(int sides)
    {
        this(sides, Color.black);
    }

    public Shape()
    {
        this(4, Color.black);
    }

. . .
```

```
. . .
    public int getSides()
    {
        return sides;
    }

    public void setColor(Color color)
    {
        this.color = color;
    }

    public void getColor()
    {
        return this.color;
    }

    public double calculateArea()
    {
        return 0;
    }
}
```

LET'S LOOK AT A CLASS.

```
Public class Shape
{
. . .
    public int getSides()
    {
        return sides;
    }

    public void setColor(Color color)
    {
        this.color = color;
    }

    public void getColor()
    {
        return this.color;
    }

    public double calculateArea()
    {
        return 0;
    }
}
```

- The surface area of a shape is the amount of space it will occupy.
- Because we know all shapes will have this, we include a `calculateArea()` method in our superclass.
- We intend for Circle, Square, and other Shape subclasses to **override** this method.
- Key question: Why do we have this here?

ABSTRACT METHODS

- In a superclass, we can define a method without implementing it.
- Essentially, we can write a signature for a method, but not write any code inside of it.
- This method can then be overridden and implemented by a subclass.
- This is called an **Abstract Method**.

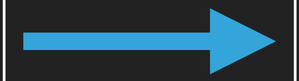
ABSTRACT METHODS

```
public class Shape
{
. . .
    public int getSides()
    {
        return sides;
    }

    public void setColor(Color color)
    {
        this.color = color;
    }

    public void getColor()
    {
        return this.color;
    }

    public double calculateArea()
    {
        return 0;
    }
}
```



Here, we are converting `calculateArea()` to an abstract method.

```
Public class Shape
{
. . .
    public int getSides()
    {
        return sides;
    }

    public void setColor(Color color)
    {
        this.color = color;
    }

    public void getColor()
    {
        return this.color;
    }

    public abstract double calculateArea();
}
```

WRITE AN ABSTRACT METHOD FOR THE SHAPE CLASS CALLED `calculatePerimeter()` THAT WILL RETURN A DOUBLE.

Pull up **Socrative.com**

and join room **XTEND159**

ABSTRACT METHODS

```
public class Shape
{
    // Instance Variables
    private int sides;
    private Color color;

    public Shape(int sides, Color color)
    {
        this.sides = sides;
        this.color = color;
    }

    public Shape(int sides)
    {
        this(sides, Color.black);
    }

    public Shape()
    {
        this(4, Color.black);
    }

. . .
```

```
. . .
    public int getSides()
    {
        return sides;
    }

    public void setColor(Color color)
    {
        this.color = color;
    }

    public void getColor()
    {
        return this.color;
    }

    public abstract double calculateArea();

    public abstract double calculatePerimeter();
}
```

ABSTRACT METHODS

```
public class Shape
{
    // Instance Variables
    private int sides;
    private Color color;

    public Shape(int sides, Color color)
    {
        this.sides = sides;
        this.color = color;
    }

    public Shape(int sides)
    {
        this(sides, Color.black);
    }

    public Shape()
    {
        this(4, Color.black);
    }

. . .
```

```
. . .
    public int getSides()
    {
        return sides;
    }

    public void setColor(Color color)
    {
        this.color = color;
    }

    public void getColor()
    {
        return this.color;
    }

    public abstract double calculateArea();

    public abstract double calculatePerimeter();
}
```

KEY QUESTION:

WILL THIS CLASS COMPILE?

ABSTRACT CLASSES

- When it comes to class inheritance,
  a Superclass is more generic, while a subclass is more specific.
- Sometimes, a superclass is so generic that it does not make sense to instantiate it.
- In such a case, we can make our superclass an **abstract class**.

CONSIDERING OUR CLASS FROM BEFORE...

```
public class Shape
{
    // Instance Variables
    private int sides;
    private Color color;

    public Shape(int sides, Color color)
    {
        this.sides = sides;
        this.color = color;
    }

    public Shape(int sides)
    {
        this(sides, Color.black);
    }

    public Shape()
    {
        this(4, Color.black);
    }

. . .
```

```
. . .
    public int getSides()
    {
        return sides;
    }

    public void setColor(Color color)
    {
        this.color = color;
    }

    public void getColor()
    {
        return this.color;
    }

    public abstract double calculateArea();

    public abstract double calculatePerimeter();
}
```

CONSIDERING OUR CLASS FROM BEFORE...

```
public abstract class Shape
{
    // Instance Variables
    private int sides;
    private Color color;

    public Shape(int sides, Color color)
    {
        this.sides = sides;
        this.color = color;
    }

    public Shape(int sides)
    {
        this(sides, Color.black);
    }

    public Shape()
    {
        this(4, Color.black);
    }

. . .
```

HOW DOES ADDING THIS KEYWORD CHANGE OUR CLASS?

```
. . .
    public int getSides()
    {
        return sides;
    }

    public void setColor(Color color)
    {
        this.color = color;
    }

    public void getColor()
    {
        return this.color;
    }

    public abstract double calculateArea();

    public abstract double calculatePerimeter();
}
```

IN YOUR OWN WORDS, WHAT IS AN ABSTRACT METHOD?

Pull up **Socrative.com**

and join room **XTEND159**

IN YOUR OWN WORDS, WHAT IS AN ABSTRACT CLASS?

Pull up **Socrative.com**

and join room **XTEND159**

TRUE OR FALSE:
AN ABSTRACT CLASS CAN BE INSTANTIATED USING THE NEW OPERATOR.

Pull up **Socrative.com**

and join room **XTEND159**

TRUE OR FALSE:
AN ABSTRACT METHOD CAN BE CONTAINED IN A NON-ABSTRACT CLASS.

Pull up **Socrative.com**

and join room **XTEND159**

TRUE OR FALSE:
AN ABSTRACT CLASS CAN CONTAIN CONSTRUCTORS.

Pull up **Socrative.com**

and join room **XTEND159**

TRUE OR FALSE:
AN ABSTRACT CLASS CAN NOT EXTEND A NON-ABSTRACT CLASS.

Pull up **Socrative.com**

and join room **XTEND159**

TRUE OR FALSE:
IN JAVA, A SUBCLASS CAN OVERRIDE A SUPERCLASS METHOD TO DEFINE IT AS ABSTRACT.

Pull up **Socrative.com**

and join room **XTEND159**

RECAP

WHAT IS AN ABSTRACT CLASS?

- A class that is not instantiated, but is extended by other classes.
  AccessSpecifier abstract class *ClassName*

WHAT IS AN ABSTRACT METHOD?

- A method that has no body and must be overridden in subclasses.
  AccessSpecifier abstract *ReturnType MethodName*(*ParameterList*);

WHAT IS AN INTERFACE?

- An interface specifies a behavior of a class.
- Looks similar to a class, except the keyword interface is used instead of the keyword class.
- public interface InterfaceName
  {
  (Method signatures…)
  }

CONSIDERING OUR CLASS FROM BEFORE...

```
public abstract class Shape
{
    // Instance Variables
    private int sides;
    private Color color;

    public Shape(int sides, Color color)
    {
        this.sides = sides;
        this.color = color;
    }

    public Shape(int sides)
    {
        this(sides, Color.black);
    }

    public Shape()
    {
        this(4, Color.black);
    }

. . .
```

```
. . .
    public int getSides()
    {
        return sides;
    }

    public void setColor(Color color)
    {
        this.color = color;
    }

    public void getColor()
    {
        return this.color;
    }

    public abstract double calculateArea();

    public abstract double calculatePerimeter();
}
```

CREATING AN INTERFACE

1. WE USE THE KEYWORD “INTERFACE” INSTEAD OF CLASS.

2. WE INCLUDE ONLY ABSTRACT METHODS AND ATTRIBUTES.

```
public interface Spacious
{
    public abstract double calculateArea();

    public abstract double calculatePerimeter();
}
```

OUR INTERFACE, “SPACIOUS”, DEFINES THE BEHAVIOR OF SOMETHING THAT TAKES UP SPACE.

IMPLEMENTING AN INTERFACE

```
public class Circle implements Spacious
{
    // Attributes
    int radius;

    public Circle(int radius)
    {
        this.radius = radius;
    }

    @Override
    public double calculateArea()
    {
        return Math.pi * Math.pow(radius, 2);
    }

    @Override
    public double calculatePerimeter()
    {
        return calculateCircumference();
    }

    public double calculateCircumference()
    {
        return 2 * Math.pi * radius;
    }
}
```

```
public interface Spacious
{
    public abstract double calculateArea();

    public abstract double calculatePerimeter();
}
```

OUR INTERFACE, "SPACIOUS", DEFINES THE BEHAVIOR OF SOMETHING THAT TAKES UP SPACE.

OUR CLASS, "CIRCLE", PROVIDES THE IMPLEMENTATION OF THAT BEHAVIOR.

IMPLEMENTING AN INTERFACE

```
public class Circle implements Spacious
{
    // Attributes
    int radius;

    public Circle(int radius)
    {
        this.radius = radius;
    }

    @Override
    public double calculateArea()
    {
        return Math.pi * Math.pow(radius, 2);
    }

    @Override
    public double calculatePerimeter()
    {
        return calculateCircumference();
    }

    public double calculateCircumference()
    {
        return 2 * Math.pi * radius;
    }
}
```

WE USE THE KEYWORD "IMPLEMENTS" IN AN INHERITANCE RELATIONSHIP.

OUR CLASS NOW MUST CONTAIN IMPLEMENTATIONS OF ALL OF THE ABSTRACT METHODS DEFINED IN OUR INTERFACE.

DEFINE AN INTERFACE IN YOUR OWN WORDS.

Pull up **Socrative.com**

and join room **XTEND159**

TRUE OR FALSE:
IF A CLASS IMPLEMENTS COMPARABLE, AN INSTANCE OF THE CLASS CAN CALL "COMPARETO".

Pull up **Socrative.com**

and join room **XTEND159**

TRUE OR FALSE:
AN INTERFACE CAN EXTEND ANOTHER INTERFACE.

Pull up **Socrative.com**

and join room **XTEND159**

TRUE OR FALSE:
AN INTERFACE CAN CONTAIN NON-ABSTRACT METHODS.

Pull up **Socrative.com**

and join room **XTEND159**

IN YOUR OWN WORDS, WHAT IS THE DIFFERENCE BETWEEN AN ABSTRACT CLASS AND AN INTERFACE?

Pull up **Socrative.com**

and join room **XTEND159**