Computer Science Virtual Learning

HS Computer Science A

May 22nd, 2020

Lesson: Abstract Classes

Objective/Learning Target:

Understanding what Abstract classes are and how to apply them in Java

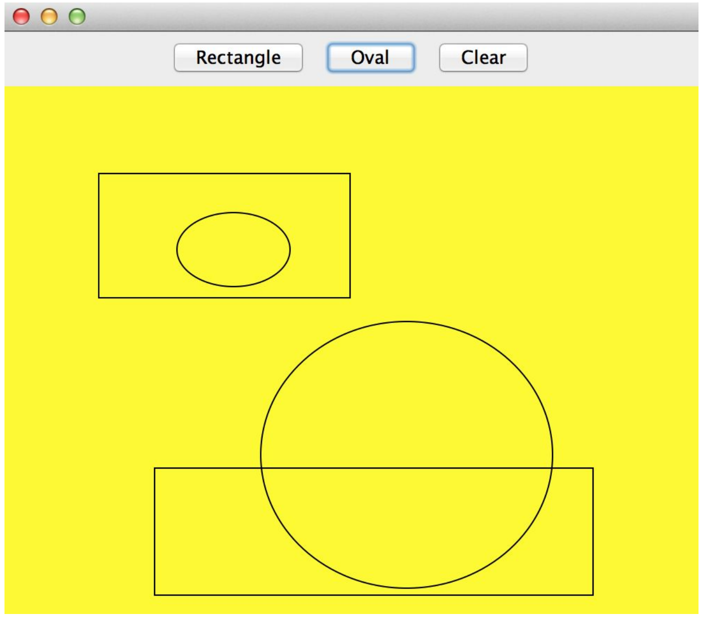
Abstract Classes

If you were creating software that allowed the user 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?

One way to identify the classes you need is to underline the nouns in the description. This would give you `Rectangle` and `Oval` as two possible classes. Both of these are kinds of simple shapes that can be defined by two points. So you could create a `SimpleShape` class that keeps track of two points and perhaps the color to draw the shape in.

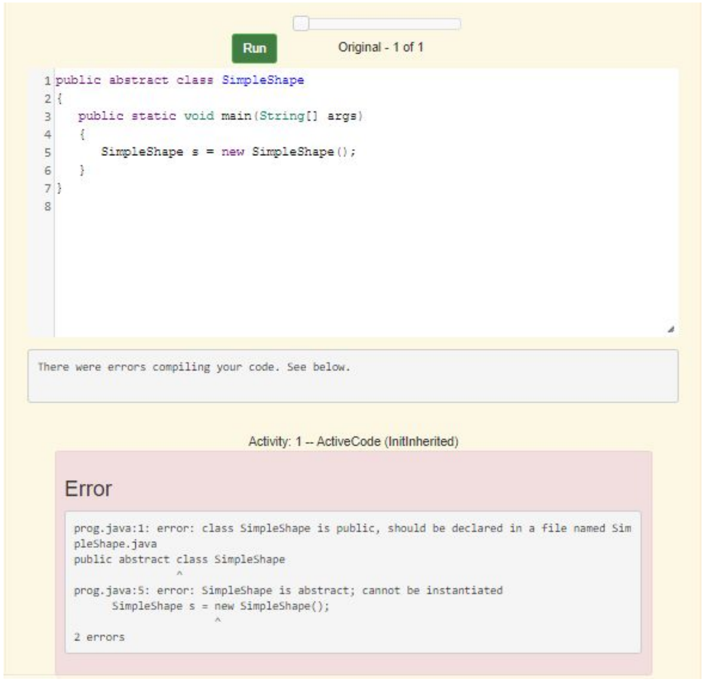
Could you actually create a `SimpleShape` object? What would it look like? How would you draw it? Since we don't know what a SimpleShape looks like we can make the class **abstract** which means that you can not create any objects of that type.

```
public abstract class SimpleShape
```



Abstract Classes Can't Be Instantiated

You can't create a new object of an abstract class. If you try you will get an error. Run the example below to see the error.



```
public abstract class SimpleShape
{
   public static void main(String[] args)
   {
      SimpleShape s = new SimpleShape();
   }
}
```

```
There were errors compiling your code. See below.
```

Activity: 1 -- ActiveCode (InitInherited)

Error

```
prog.java:1: error: class SimpleShape is public, should be declared in a file named Sim
pleShape.java
public abstract class SimpleShape
                ^
prog.java:5: error: SimpleShape is abstract; cannot be instantiated
      SimpleShape s = new SimpleShape();
                      ^
2 errors
```

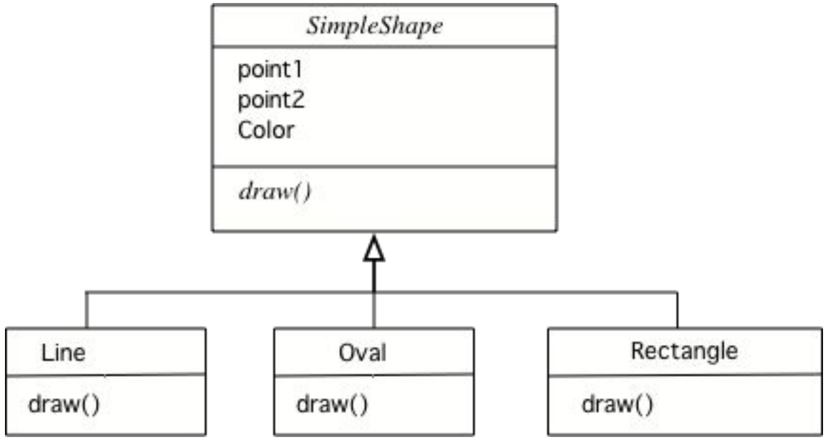
Abstract Classes Exist to be Subclassed

What is an abstract class good for if you can't create any objects from it? You can use it as a parent class for subclasses.

Abstract classes often have at least one abstract method (a method that has the keyword `abstract` in the declaration and no method body), but they don't have to. Abstract classes can have constructors, fields, and methods with bodies (non-abstract methods). If you create a class with at least one abstract method, the class has to be declared to be an abstract class.

An abstract `SimpleShape` class could have constructors and fields to track the points and methods that calculate the width and height of the shape. The only method that has to be abstract is the `draw` method, since we don't know how to draw a `SimpleShape`.

Abstract classes are typically used when you want to put some data and/or behavior in a parent class, but at least one method needs to be abstract and overridden by the child class(es). The `SimpleShape` class can have an abstract `draw` method and then the children classes can specify what `draw` does.



Inheritance and Interfaces

An **interface** in Java is a special type of abstract class that can only contain public abstract methods (every method is assumed to be `public` and `abstract` even if these keywords are not specified) and public class constants. `List` is an interface in Java. Interfaces are declared using the **interface** keyword. One interface can inherit from another interface.

```
public interface Checker
{
      boolean check (Object obj);
}
```

The code above declares an interface called `Checker` that contains a public abstract method called `check` that returns true or false. Classes that implement this interface must provide the body for the `check` method.

Another example of an interface in Java is the **Iterator** interface. It is used to loop through collection classes (classes that hold groups of objects like `ArrayList`).

What is the Purpose of an Interface?

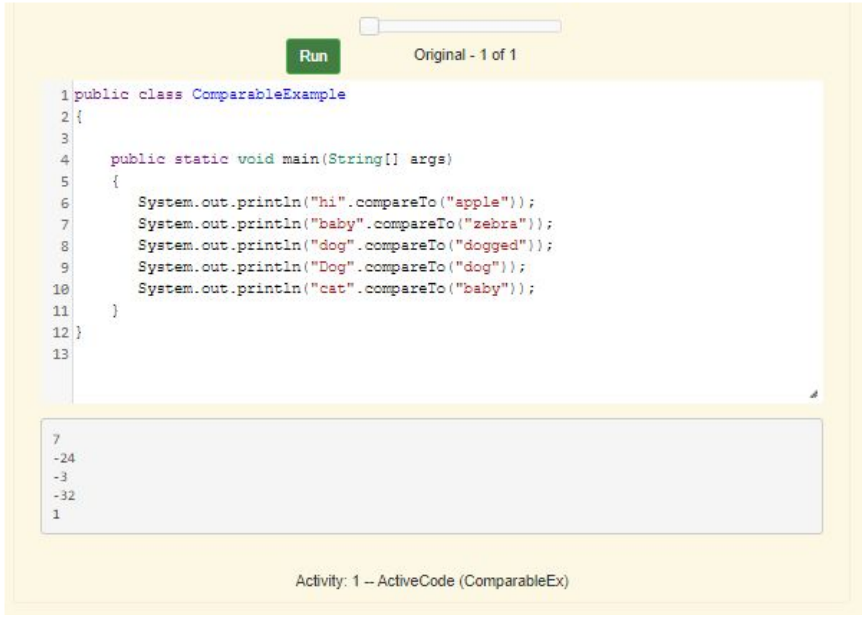
The purpose of an interface is to separate *what* you want a type to be able to do (defined by the method signatures) from *how* it does that. This makes it easy to substitute one class for another if they both implement the same interface and you have declared the variable using the interface type. The `List` interface defines what a class needs to be able to do in order to be considered a `List`. You have to be able to add an item to it, get the item at an index, remove the item from an index, get the number of elements in the list, and so on. There are several classes that implement the `List` interface. You only have to know about `ArrayList` for the exam, which is a class that implements the `List` interface using an array.

The Comparable Interface

In Java, you can sort objects of any class that implements the `Comparable` interface. The `Comparable` interface just specifies the `int compareTo(T o)` method which will return a negative number if the current object is less than the passed one, 0 if they are equal, and a positive number if the current object is greater than the passed one. How do you compare two objects of any class? It really depends on both the class and the context.

One common misconception is that `compareTo` returns -1, 0, or 1 but that is wrong. It returns a negative number (if less than), 0, or a positive number (if greater). Be careful in conditionals to use `< 0` to test for the object it is called on being less than the passed object, `== 0` to test for equals, and `> 0` to test for the object it is called on being greater than the passed object.

The `String` class implements the `Comparable` interface. Let's see what is actually returned when you compare strings.



```java
public class ComparableExample
{

    public static void main(String[] args)
    {
        System.out.println("hi".compareTo("apple"));
        System.out.println("baby".compareTo("zebra"));
        System.out.println("dog".compareTo("dogged"));
        System.out.println("Dog".compareTo("dog"));
        System.out.println("cat".compareTo("baby"));
    }
}
```

```
7
-24
-3
-32
1
```

Activity: 1 -- ActiveCode (ComparableEx)

For More Resources and to Check Answers

Go to: https://runestone.academy/runestone/books/published/apcsareview/OOBasics/ooAbstract.html

https://runestone.academy/runestone/books/published/apcsareview/OOBasics/ooInheritanceAndInterfaces.html

https://runestone.academy/runestone/books/published/apcsareview/OOBasics/ooComparable.html