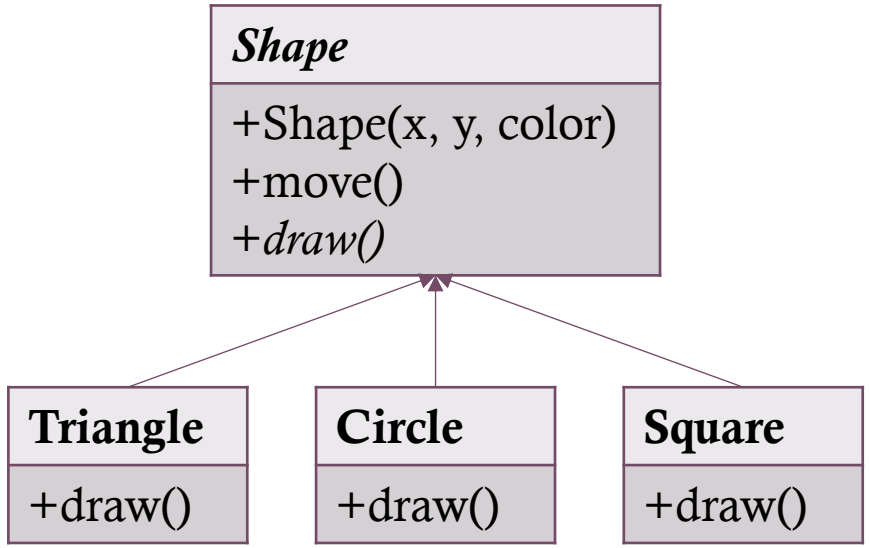
WEEK TWO

Acknowledgements: Slides created based off material provided by Dr. Travis Doom and Dr. Michael Raymer

ABSTRACT CLASSES & METHODS

ABSTRACT CLASSES

- A class which we cannot create an object for (instance of)
- Sometimes we only need child classes
- Creates a contract the child class must follow (with/without default implementation)
- Can have static and non-static members (non-static must be called by subclass objects)
- Denoted via italics in UML
- Can be made up of abstract and non-abstract methods



ABSTRACT METHODS

- Can only exist in an abstract class
- Used when a parent class has no reasonable default
- Forces subclasses to implement/override the method
- Defines contract (return type, parameters), not implementation
- Has no body

```
public abstract class Shape{
        public Shape (int x, int y, Color color){
                // implementation
        }
        public void move (){
                //implementation
        }
        public abstract void draw ();
}
```

CAN I INHERIT FROM MORE THAN ONE CLASS?

No, we can only extend one class.

What if I want to implement multiple different functionalities in one class?

INTERFACES

- Support multiple inheritance
- Similar to abstract classes, but there is **NO** implementation (100% abstract)
  - Creates the contract but forces the subclass to implement
  - Define how the class should behave
- No constructor needed
- Use *interface* keyword to declare

```
public interface Shape{
        public abstract void move();
        public abstract void draw();
}
```

INTERFACES CONTINUED

- Use *implements* keyword to utilize
- If implementing multiple interfaces, use commas to separate them
- Still can only extend one class

```
public class GUICard extends Card implements Shape, Comparable{
	// implementation must override move, draw, and compareTo
	// (from Comparable)
}
```

USEFUL INTERFACES

Comparable, Runnable, Serializable

Comparator, Deque, List, Map, Queue, Set

COMPARABLE INTERFACE

- Has one method
  - `int compareTo(T o)`
- Allows us to define a way for our class to be sorted
- `compareTo()` is called under the hood from `Collections.sort()`

COMPARATOR INTERFACE

- Only has one method we have to override
  - `int compare(T o1, T o2)`
- Allows us to define multiple different ways to sort an Object
- To use, we must pass a new instance of the class into `Collections.sort()` as a parameter
  - `Collections.sort(o, new SortClass());`