Interfaces & Abstract Classes

- Interfaces in Java
- Abstract Classes

What's inherited from Object?

- There are three methods
  - String toString()
  - boolean equals(Object other)
  - int hashCode()

- The implementation of these three is called *overriding*

Interface vs. Implementation

- *Interface*: the abstract way in which you interact with an object.

- *Implementation*: the concrete details of how an object work

- Think of two objects that have the same behavior (interface), but their implementation is different
  - Airplanes: propeller and jet
  - Clocks: digital and analog

ArrayLists

- ArrayList is the array implementation of an interface called List

```
ArrayList<Integer> mylist;
ArrayList<E> mylist;
```

- We called the List Class an *interface*

- There are different ways of implementing the same interface. That's the idea -- different objects with similar behaviors have the same interface.

Abstract Classes

- An *abstract* class in Java is a class that has no instances and it is used as the base for other classes.
- It has fields
- It has methods
  - Some are implemented
  - Some are not implemented (*abstract methods*)
- Can have constructor(s) to initialized its fields
- Cannot be instantiated directly

Abstract Classes & Inheritance

```
public abstract class Engine{
        private double speed;
        public abstract  double thrust();
        public double getSpeed(){return speed};
}
public class Jet extends Engine{
        public double thrust(){
                …. implementation …
        }
}
```

Inheritance

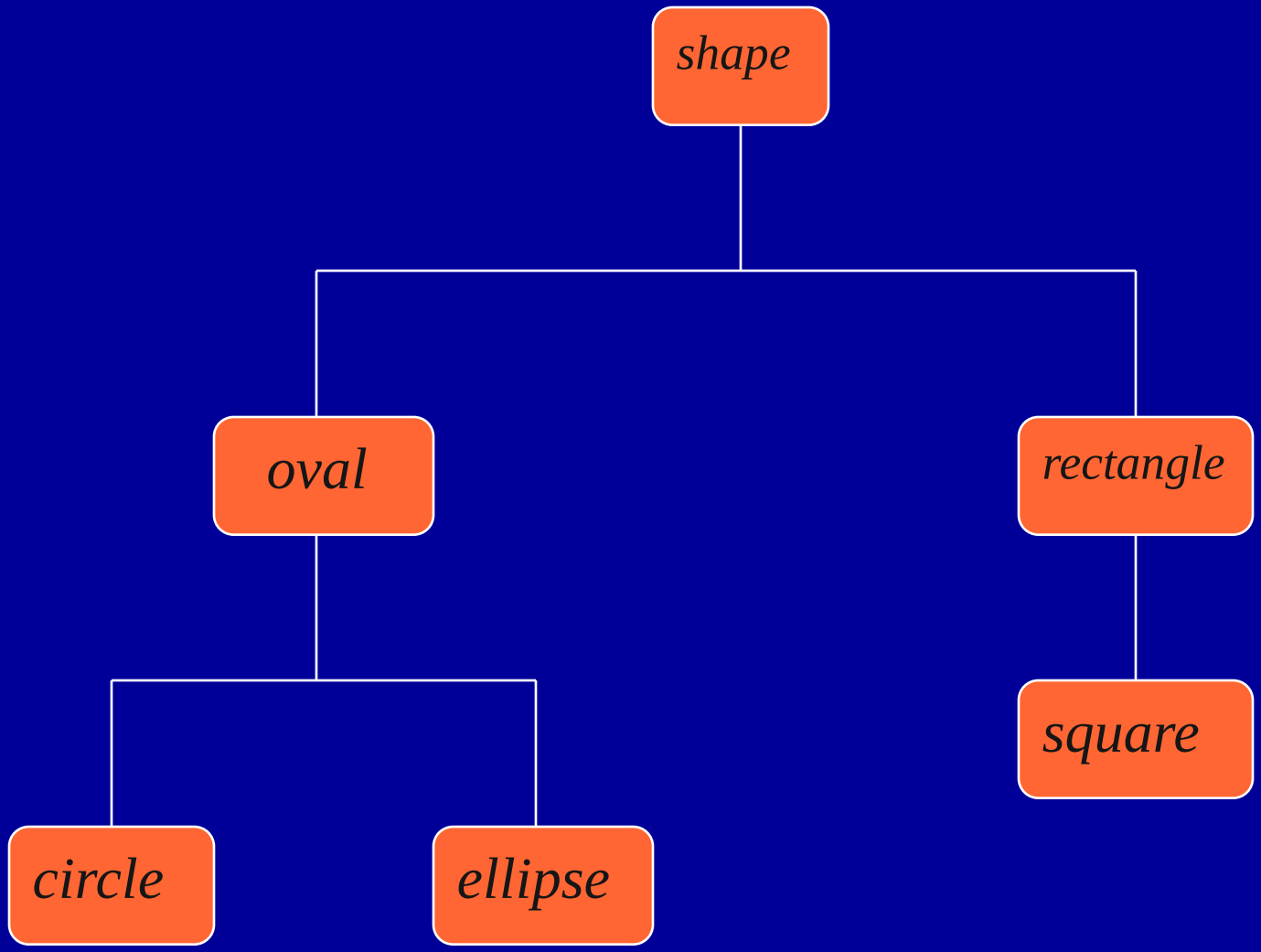
- In Java, we can only inherit from one class (*single inheritance*)

  public class Jet extends Engine

  public class Propeller extends Engine

- Can one inherit from more than one class (*multiple inheritance*) in Java?
  - It gets really messy
  - Very confusing
  - C++

Superclasses & Subclasses



Interfaces

- *Interfaces* in Java specify the common behavior of a set of classes.
  - have no fields and no constructors
  - have no default methods
  - have no implementations

public class Jet implements Engine

public class Propeller implements Engine

So what?

public class Jet implements Engine, Plane

More Interfaces

- The classes Jet and Propeller implement the methods of the classes Plane and Engine
  - Also, they implement their own methods

- In Java, there are three *interfaces*
  - *List*
  - *Set*
  - *Maps*
- In Java, there are two implementations of *List*
  - *ArrayList*
  - *LinkedList*

Inheritance, Interfaces, or Abstract?

- We wish to design a chess game. Let's create a Piece Class.

- What does Piece have to remember?
  - location?
  - color?
  - getMoves()?
  - What else?
- Let's create subclasses
  - Knight
  - Queen
  - Bishop

Piece Class

- Piece is a *superclass*
- Each subclass needs to override getMoves()

- Let's analyze the Piece class:
  - Java allows us to instantiate a new Piece
    - Does it make sense?
  - We must define a default getMoves()
    - Does it make sense?

Piece interface

- Changing Piece to be an interface instead means
  - no fields (color, position)
  - no constructor
  - getMoves()?

- Let's analyze the Piece interface:
  - Each subclass must implement getMoves()
  - Each subclass has its own fields (color, position)

Abstract Class Piece

- it has fields (color, position)
- It has getMoves()
  - An abstract method
- Does not have constructors

Comparable Interface

- We can compare any two objects for equality using the equals method.

  object1.*equals*(object2)

- What if we want to determine "…this object is less than …" or "…this object is greater than …"?
  - with Strings we can use *compareTo()* method
  - But what about other values?

- Can we create our own *compareTo()* method, for every class we implement?

Comparable Interface

java.lang.Comparable;

This interface has a method, *compareTo()*, which returns:

- a negative integer if the current object is less than the specified object.
- zero if the two objects are equal.
- a positive integer if the current object is greater than the specified object.

CompareTo() method

- Its signature is:

  public int compareTo(Object another)

Look up how the *compareTo()* method works in the String class.

Let's revisit our classes Shape, Circle, Rectangle and Stretchable.

Summary

What can a concrete class do that an abstract class cannot?

- We CAN instantiate a concrete class.
- We CANNOT instantiate an abstract class.

What does an abstract class have that a concrete class cannot?

- An abstract class can HAVE BOTH concrete and abstract methods.
- A concrete class can ONLY HAVE concrete methods.

Summary

What can an abstract class do that an interface cannot?

- An abstract class CAN have fields, constructors, and concrete methods.
- An interface CANNOT have any of these.

What can an interface do that an abstract class cannot?

- A class can only *extend* ONE superclass (which may or may not be abstract).
- A class can *implement* MULTIPLE interfaces.

Homework

- Exam this Thursday
  - bring your CMU ID
- Homework #5 (classes & interfaces)

  due on Tuesday 10/13
- Quiz #5 (abstract classes & interfaces) on Wednesday 10/14