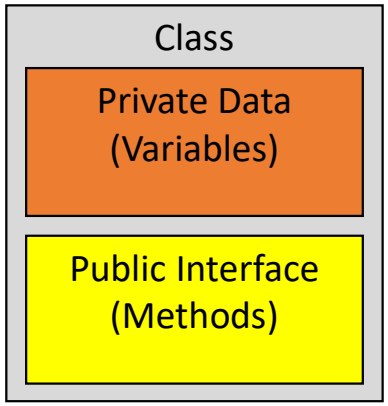
# Program Structure

- Typical Java program consists of
  - User written classes
  - Java Application Programming Interface (API) classes
- Java application
  - Has one class with a *main* method
- Java program basic elements:
  - Packages
  - Classes
  - Data fields
  - Methods

# Diagram of a Class

- Private Data
  - Each object has its own private data that other objects cannot directly access
  - Methods of the public interface provide access to private data, while hiding implementation details:
  - This is called Encapsulation
- Public Interface
  - Each object has a set of methods available for other objects to use

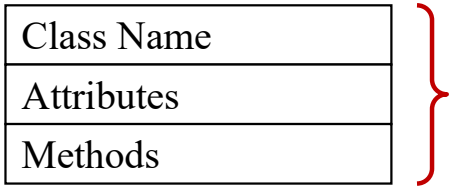


# Designing a Class

- When designing a class, decisions about the following must be made.
  - what data must be accounted for
  - what actions need to be performed
  - what data can be modified
  - what data needs to be accessible
  - any rules as to how data should be modified
- Class design typically is done with the aid of a Unified Modeling Language (UML) diagram.

# UML Class Diagram

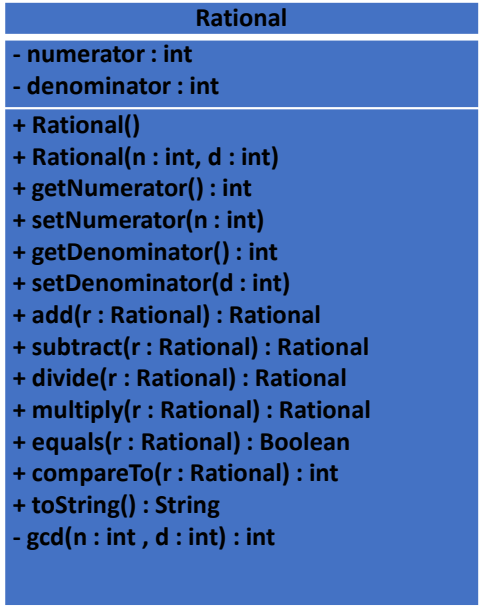
- A UML class diagram is a graphical tool that can aid in the design of a class.
- The diagram has three main sections.




**UML diagrams are easily converted to Java class files. There will be more about UML diagrams a little later.**

  - The class name should concisely reflect what the class represents.

# Rational Numbers UML

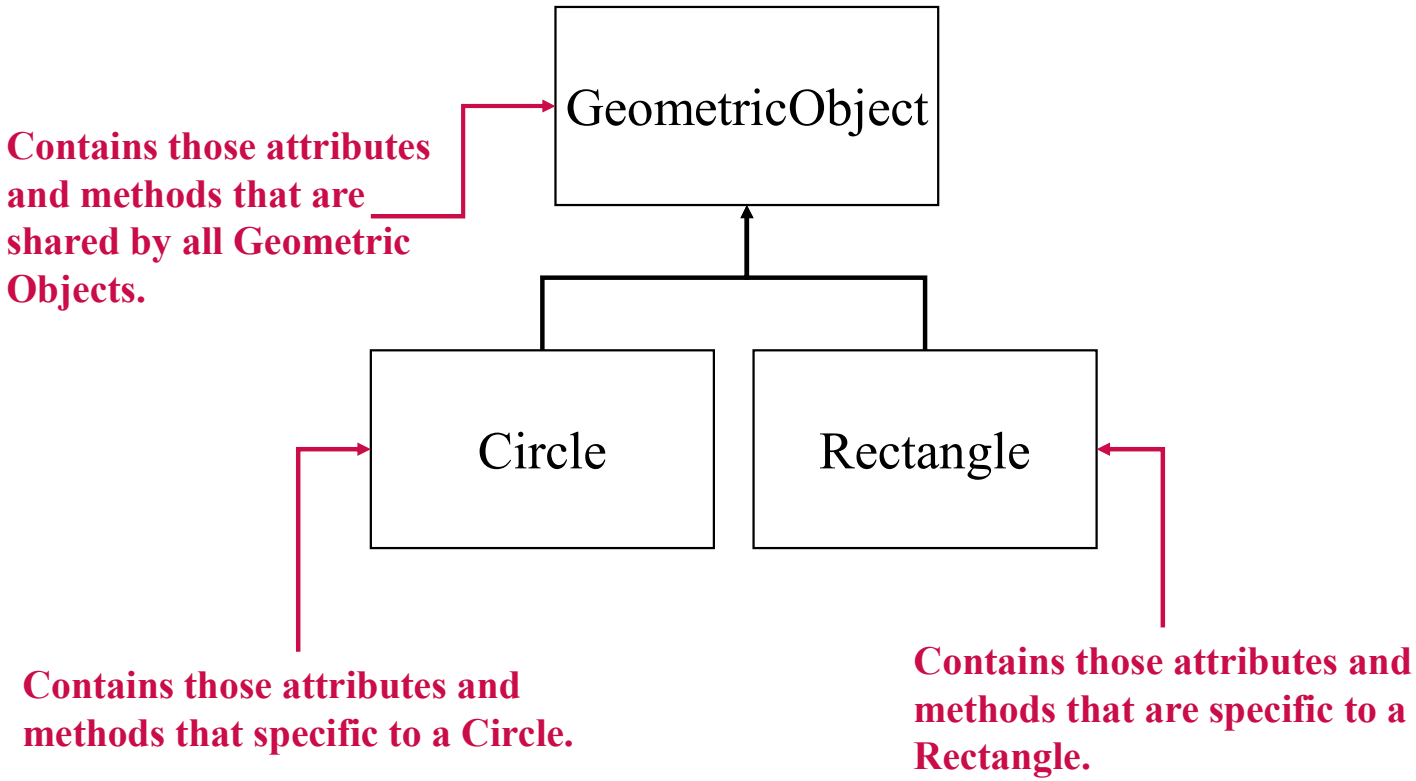



**default value for a Rational object is 0/1**
**Rational objects must be in reduced form**
**Rational objects should not be modified by any of the methods**
**toString() format is "numerator/denominator"**

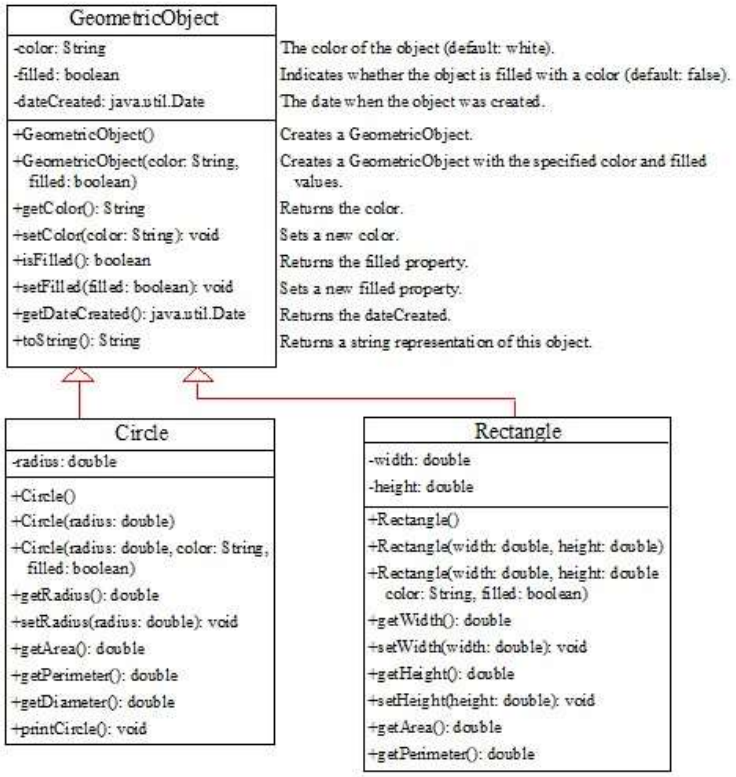
# Inheritance

- Technique for creating a new class that is based on one that already exists.
  - Desire to add new features
  - Desire to define a more specific data type
  - We don’t want to change the original class
- Example:  GeometricObject and circle, Rectangle
  - We already have the GeometricObject class
  - Circle and Rectangle will be everything a GeometricObject is, but more.

# Inheritance



# Superclasses and Subclasses




GeometricObject

Circle

Rectangle

TestCircleRectangle

# Inheritance

- Terminology
  - Base class (or superclass): the original class from which we create the new one
  - Derived class (or subclass): the new class we create
  - We say that the subclass inherits data members and operations of its superclass.
- Accessibility
  - Subclass has access to attributes of its superclass, but the superclass cannot access attributes of its subclass(s)

# Useful Java Classes

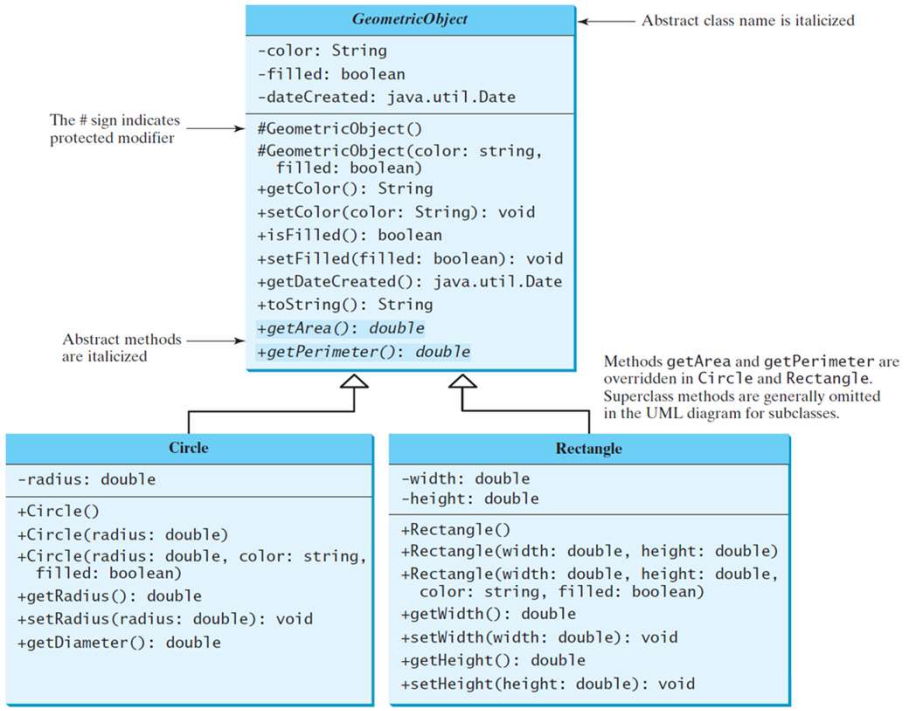
- The `Object` class
  - Java supports a single class inheritance hierarchy
    - With class `Object` as the root
  - More useful methods
    - **`public boolean`** `equals(Object obj)`
    - **`protected void`** `finalize()`
    - **`public int`** `hashCode()`
    - **`public String`** `toString()`

# Useful Java Classes

- The `Arrays` class
  - `import java.util.Arrays;`
  - Contains static methods for manipulating arrays
- Commonly used examples
  - Sort (does it in ascending order)
  - Binary search (quickly finds a value in the array)
  - toString
- Example: Let's say `a` is an array of 1000 `int`s

```
Arrays.sort(a);
```

# Abstract Classes and Abstract Methods




GeometricObject

Circle

Rectangle

TestGeometricObject

# Abstract Method in Abstract Class

An abstract method cannot be contained in a nonabstract class. If a subclass of an abstract superclass does not implement all the abstract methods, the subclass must be defined abstract. In other words, in a nonabstract subclass extended from an abstract class, all the abstract methods must be implemented, even if they are not used in the subclass.

# Abstract Class as Type

You cannot create an instance from an abstract class using the new operator, but an abstract class can be used as a data type. Therefore, the following statement, which creates an array whose elements are of GeometricObject type, is correct.

```
GeometricObject[] geo = new GeometricObject[10];
```

# Interfaces

What is an interface?

Why is an interface useful?

How do you define an interface?

How do you use an interface?

# What Is an Interface? Why Is an Interface Useful?

An interface is a class like construct that contains only constants and abstract methods. In many ways, an interface is similar to an abstract class, but the intent of an interface is to specify common behavior for objects. For example, you can specify that the objects are comparable, edible, cloneable using appropriate interfaces.

# Define an Interface

To distinguish an interface from a class, Java uses the following syntax to define an interface:

```
public interface InterfaceName {
  constant declarations;
  abstract method signatures;
}
```

# Example

```
public interface MyList<E>  {

  /** Remove the element at the specified position in this list
   *  Shift any subsequent elements to the left.
   *  Return the element that was removed from the list. */
  public E remove(int index);

  /** Remove the element at the beginning of this this list
   *  Return the element that was removed */
  public E removeFirst();

  /** Remove the element at the end of this this list
   *  Return the element that was removed */
  public E removeLast();

  /** Add a new element at the specified position*/
  public void add(int index, E e);

  /** Add a new element at the end of this list */
  public  void addLast(E e);

  /** Add a new element at the beginning of this list */
  public  void addFirst(E e);

  /** Return true if this list contains no elements */
  public  boolean isEmpty();


  /** Return size of this list */
  public int size();

}
```

# Example: The Comparable Interface

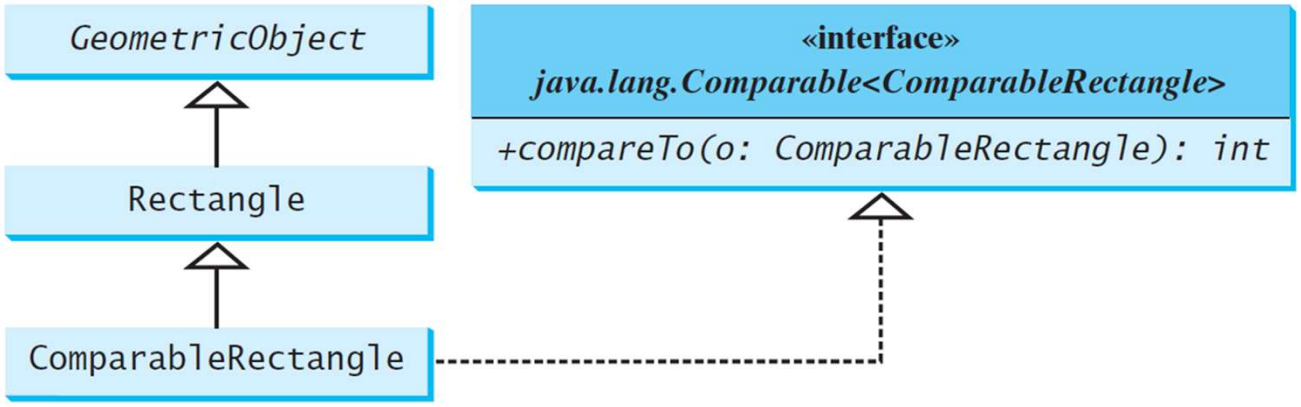
```
// This interface is defined in
// java.lang package
package java.lang;

public interface Comparable<E> {
  public int compareTo(E o);
}
```

# The toString, equals, and hashCode Methods

Each wrapper class overrides the toString, equals, and hashCode methods defined in the Object class. Since all the numeric wrapper classes and the Character class implement the Comparable interface, the compareTo method is implemented in these classes.

# Defining Classes to Implement Comparable




ComparableRectangle

SortRectangles

# Interfaces vs Abstract Classes (1 of 2)

In an interface, the data must be constants; an abstract class can have all types of data.

Each method in an interface has only a signature without implementation; an abstract class can have concrete methods.

| | **Variables** | **Constructors** | **Methods** |
|---|---|---|---|
| Abstract class | No restrictions. | Constructors are invoked by subclasses through constructor chaining. An abstract class cannot be instantiated using the new operator. | No restrictions. |
| Interface | All variables must be **public static final** | No constructors. An interface cannot be instantiated using the new operator. | All methods must be public abstract instance methods |