Chapter 13 Abstract Classes and Interfaces

CS165
Colorado State University

Original slides by Daniel Liang
Modified slides by Wim Bohm, Sudipto Ghosh

Motivation

- An *abstract class* is used when you want to share code among closely related classes
  - Methods that must be defined in subclasses
  - Methods that can be reused by subclasses
- An *interface* defines common behavior for classes (including unrelated classes).
- While a class can inherit from only one class, an interface can inherit from multiple interfaces and a class can implement multiple interfaces

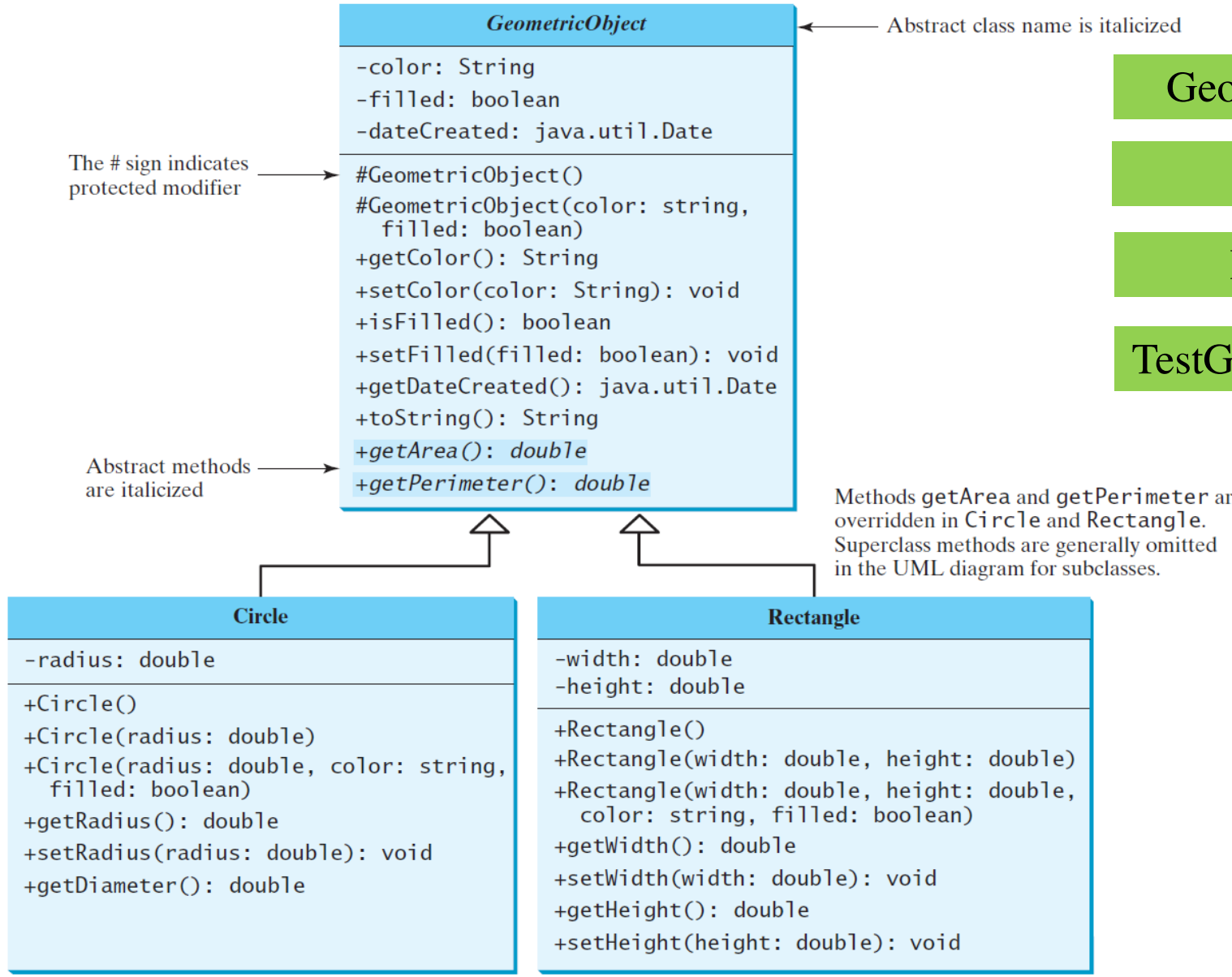
What is an abstract class?

- A class that is declared abstract
- May (or may not) contain abstract methods
  - Method declarations without implementations
- We use abstract methods when the super class has no way to implement the method, e.g. area() for geometricObject (completely different for circle and rectangle).

Abstract method

- Must be contained in an abstract class.
- A concrete subclass of a superclass must implement all the abstract methods,
- otherwise, the subclass must be defined abstract.

Abstract Classes and Abstract Methods



GeometricObject

Circle

Rectangle

TestGeometricObject

Run

Abstract class constructors

- An abstract class cannot be instantiated using the new operator.
- However, constructors can be defined to initialize the instance variables of the abstract class
- Constructors of a subclass invoke the constructors of the super class.
- The constructors of GeometricObject are invoked in the Circle class and the Rectangle class.

Superclass of abstract class may be concrete

- A subclass can be abstract even if its superclass is concrete.
- New abstract methods can be declared in the subclass.
- For example, the Object class is concrete, but its subclasses, such as GeometricObject, may be abstract.

Concrete method overridden to be abstract

- Rarely done.
- Useful when the implementation of the method in the superclass becomes invalid in the subclass.
- A subclass can override this method from its superclass to define it abstract.
- The subclass must be defined abstract.

Using abstract class as type

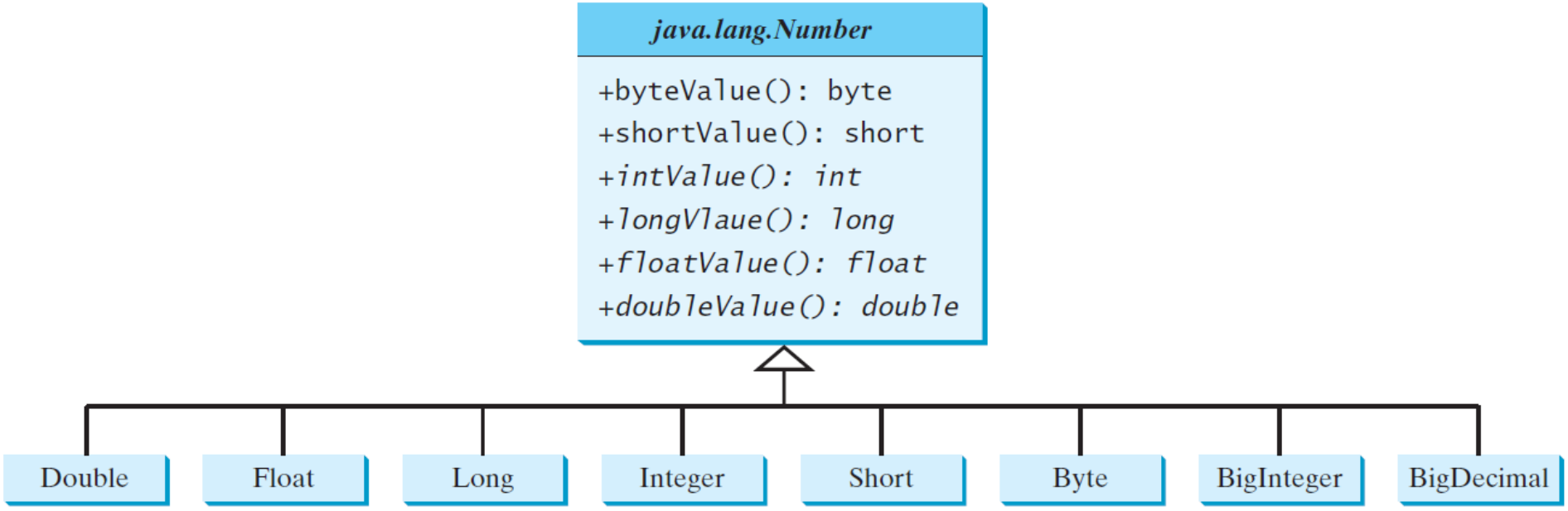
- An abstract class can be used as a data type.
- The below statement correctly creates an array whose elements are of GeometricObject type.

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

- Note that the array elements are references to GeometricObject instances. No GeometricObjects are instantiated.

Cannot write: geo[0]= new GeometricObject();

Case Study: the Abstract Number Class



LargestNumbers Run

What is an interface?

- Defines what operations an object can perform.
- You can specify common behavior for objects using appropriate interfaces
  - comparable, edible, cloneable.
- Contains only constants and abstract methods.
- Similar to abstract classes in many ways
- Operations are defined by the classes that implement the interface

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 Edible {
  /** Describe how to eat */
  public abstract String howToEat();
}
```

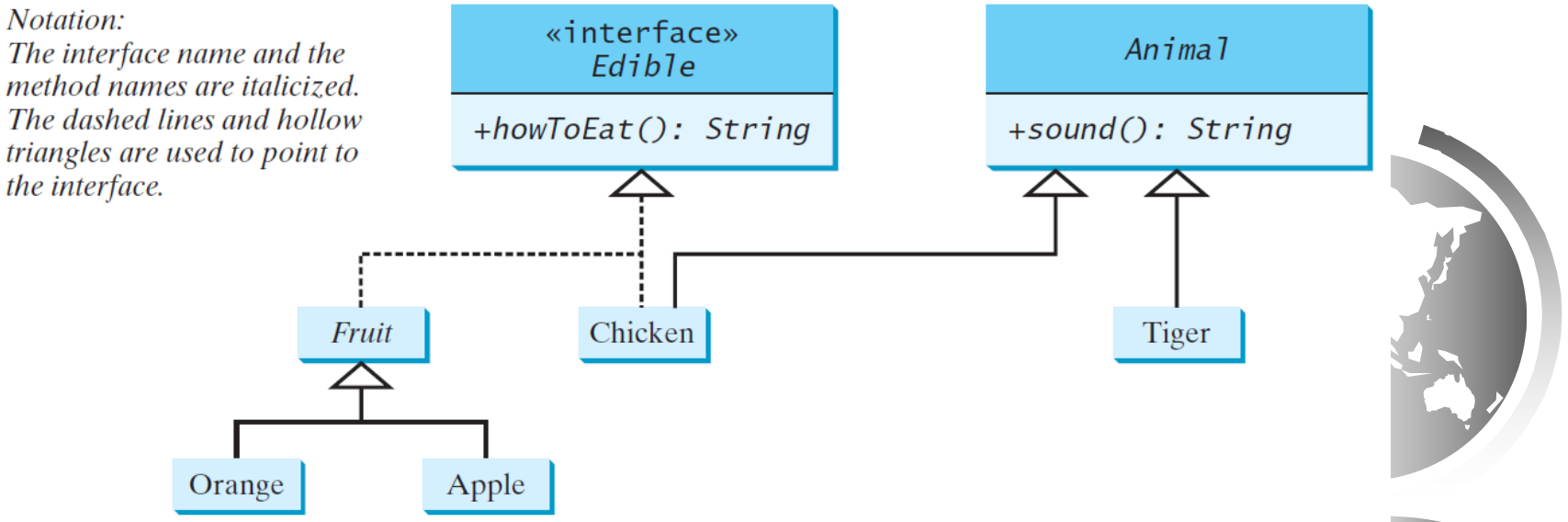
Interface is a Special Class

- Treated like a special class in Java.
- Each interface is compiled into a separate bytecode file, just like a regular class.
- Like an abstract class, you cannot create an instance from an interface using the new operator.
- In most cases you can use an interface more or less the same way you use an abstract class.
- For example, you can use an interface as a data type for a variable, as the result of casting, and so on.

Example

- The Edible interface specifies whether an object is edible.
- Classes Chicken and Fruit implement the Edible interface
- They use the *implements* keyword.
- For example, (See TestEdible).

Edible TestEdible Run



Omitting Modifiers in Interfaces

- All data fields are *public final static*.
- All methods are *public abstract*.
- For this reason, these modifiers can be omitted.

```
public interface T1 {
  public static final int K = 1;

  public abstract void p();
}
```

Equivalent

```
public interface T1 {
  int K = 1;

  void p();
}
```

- A constant defined in an interface can be accessed using syntax InterfaceName.CONSTANT_NAME
  - (e.g., T1.K).

Example: The Comparable Interface

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

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

compareTo() returns a negative integer, zero, or a positive integer when this object is less than, equal to, or greater than the object o.

Wrapper classes

- Each wrapper class (Integer, Double, etc.) 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.

Example

```
System.out.println(new Integer(3).compareTo(new Integer(5)));
System.out.println("ABC".compareTo("ABE"));

java.util.GregorianCalendar cal1 =
    new java.util.GregorianCalendar (2013, 1, 1);
java.util. GregorianCalendar cal2 =
    new java.util. GregorianCalendar(2012, 1, 1);
System.out.println(cal1.compareTo(cal2));
```

Generic `sort` Method

Let **n** be an **Integer** object, **s** be a **String** object, and **d** be a **Date** object. All the following expressions are **true**.

```
n instanceof Integer
n instanceof Object
n instanceof Comparable
```

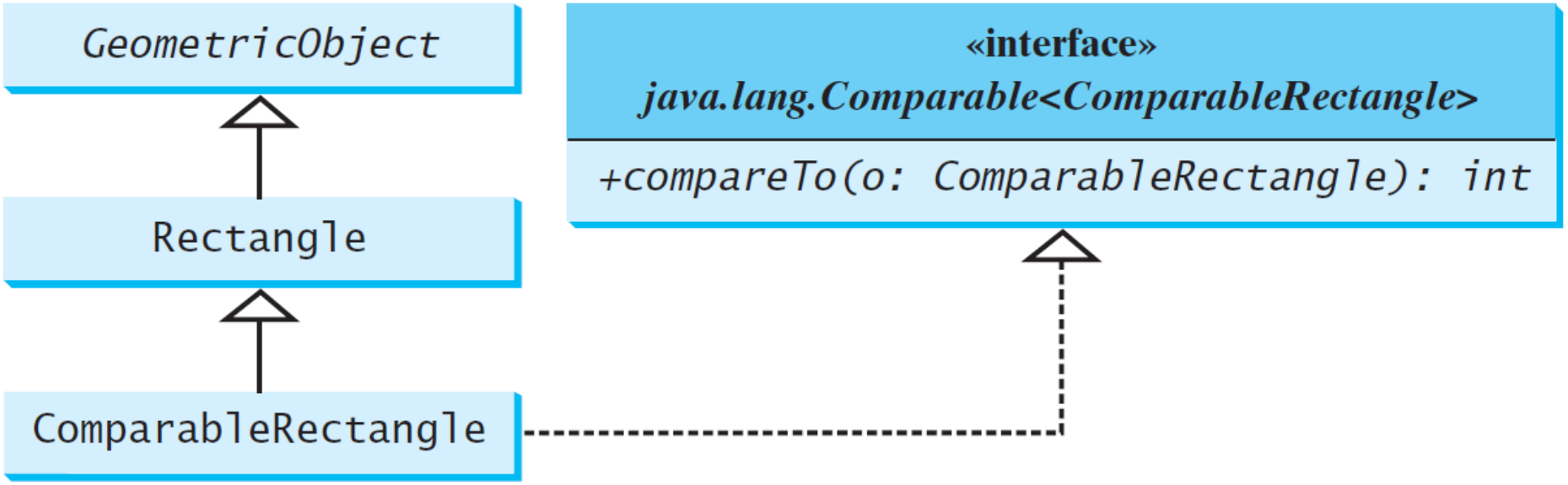
```
s instanceof String
s instanceof Object
s instanceof Comparable
```

```
d instanceof java.util.Date
d instanceof Object
d instanceof Comparable
```

The java.util.Arrays.sort(array) method requires that the elements in an array are instances of Comparable<E>.

SortComparableObjects Run

Defining Classes to Implement Comparable



ComparableRectangle

SortRectangles

Run

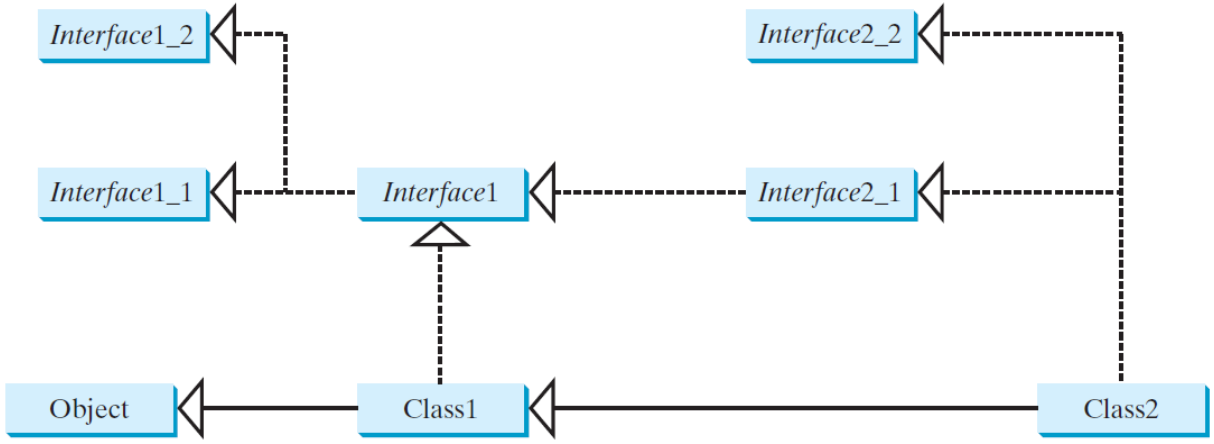
Interfaces vs. Abstract Classes

- 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

Interfaces vs. Abstract Classes, cont.

- All classes share a single root, the Object class, but there is no single root for interfaces.
- Like a class, an interface also defines a type.
  - A variable of an interface type can reference any instance of the class that implements the interface.
  - If a class implements an interface, this interface plays the same role as a superclass. You can use an interface as a data type and cast a variable of an interface type to its subclass, and vice versa.



Suppose that c is an instance of Class2. c is also an instance of Object, Class1, Interface1, Interface1_1, Interface1_2, Interface2_1, and Interface2_2.

Caution: conflicting interfaces

- In rare occasions, a class may implement two interfaces with conflicting information, e.g.,
  - two same constants with different values
  - two methods with same signature but different return type.
- The compiler detects this type of errors.

Whether to use an interface or a class?

- Abstract classes and interfaces can both be used to model common features.
- In general, a strong is-a relationship that clearly describes a parent-child relationship should be modeled using classes.
  - For example, a staff member is a person.
- Interfaces can be used to for weak is-a relationship, also known as an is-kind-of relationship, indicates that an object possesses a certain property.
  - For example, all strings are comparable, so the String class implements the Comparable interface.

Whether to use an interface or a class?

- Interfaces can be used to circumvent the restriction imposed by single inheritance in classes.
- If you need to use multiple inheritance, you have to design one as a superclass, and others as interfaces.

Designing a Class

- Provide a public no-arg constructor.

  Subclasses may need it for Constructor haining

- Consider overriding the equals, toString and hashCode methods.

Programming style and conventions

- Follow standard Java programming style and naming conventions.
- Always place the data declaration before the constructor, and place constructors before methods.
- Always provide a constructor and initialize variables to avoid programming errors.
- Choose informative names for classes, data fields, and methods.

Using Visibility Modifiers

- Each class can present two contracts
  - one for the users of the class
  - one for the extenders of the class.
- Make the fields private and accessor methods public if they are intended for the users of the class.
- Make the fields or method protected if they are intended for extenders of the class.
- The contract for the extenders encompasses the contract for the users.
- The extended class may increase the visibility of an instance method from protected to public, or change its implementation, but you should never change the implementation in a way that violates that contract.

Using Visibility Modifiers, cont.

- A class should use the private modifier to hide its data from direct access by clients.
- You can use get methods and set methods to provide users with access to the private data, but only to private data you want the user to see or to modify.
- A class should also hide methods not intended for client use.