Inheritance II: Overriding Methods and Constructors

- A derived class is allowed to "rewrite" methods that originate in a base class.
  - Very common; done to alter the way a derived class behaves.
- This is called ***overriding***.
- Overriding a method in a derived class "hides" the base class method code and replaces it with your new code.

```
class A {
    public void method1() { System.out.println("This is method 1 in A."); }
    public void method2() { System.out.println("This is method 2 in A."); }
}
class B extends A {
    public void method1() { System.out.println("This is method 1 in B."); }
    // this line above overrides method 1 for objects of class B.
}

public static void main(String[] args) {
    A objA = new A();
    B objB = new B();
    objA.method1();  // Prints This is method 1 in A.
    objA.method2();  // Prints This is method 2 in A.
    objB.method1();  // Prints This is method 1 in B.
    objB.method2();  // Prints This is method 2 in A.
}
```

The keyword `super` may be used inside of a class to call methods of the base class (often used when you are overriding a method and want to call the base class's version of the method as part of the overriding).

```
class A {
    public void method1() { System.out.println("This is method 1 in A."); }
    public void method2() { System.out.println("This is method 2 in A."); }
}
class B extends A {
    public void method1() {
        super.method1();  // calls the original method1() from class A.
        System.out.println("Calling method 1 in B.");
    }
}

public static void main(String[] args) {
    A objA = new A();
    B objB = new B();
    objA.method1();  // Prints This is method 1 in A.
    objA.method2();  // Prints This is method 2 in A.
    objB.method1();  // Prints This is method 1 in A.   (and the next line)
                     //        This is method 1 in B.
    objB.method2();  // Prints This is method 2 in A.
}
```

Constructors

- When a derived class inherits from a base class, and an object of the derived class is created, at least one constructor from the base class and the derived class must be called, even if one or both of them are default, no-argument constructors that Java calls automatically.

- Constructors are the only methods that are not automatically inherited by derived classes. If you want a derived class to have a constructor with certain behaviors or arguments, you must define it yourself.

- You may call the base class's constructors by using the `super` keyword as if it were a function: `super(arg1, arg2, ...)`

Practice:

We will practice inheritance with some simple classes that you can imagine might be used in a car racing game.

Car.java defines a Car class that knows its current speed, its top speed (as fast as the car can possibly go), and the total distance the car has traveled in the race.

There is a drive() method that attempts to accelerate the car by 5mph unless the top speed is reached, and then drives the car forward for one second of time.

CarDemo.java simulates a car race that lasts 60 seconds (it only has one car right now, though).

Create two new classes that inherit from Car, with the following modified behaviors:

- Create a class called Racecar that:
  - can accelerate at 10 mph every second, rather than 5 mph every second
  - has a top speed of 200 mph.

- Create a class called Clunker that:
  - still accelerates at 5 mph per second.
  - has a top speed of 50 mph.
  - But after calling drive() 3 times, the car dies, immediately stops, can't be fixed, and you have to call a friend to pick you up. (In other words, the current speed of the car immediately drops to zero.)