• Previous Lecture:
  • Review: parameter passing, trace
  • Introduction to inheritance
  • extend a class
  • Visibility modifier protected

• Today’s Lecture:
  • Accessing parent class: super
  • Overriding methods
  • Polymorphism

• Reading (JV):
  • Sec 7.1 – 7.3

---

### protected visibility

- Visibility modifiers control which members get inherited

- **private**
  - Not inherited, can be accessed by local class only

- **public**
  - Inherited, can be accessed by all classes

- **protected**
  - Inherited, can be accessed by subclasses

- **Access**: access as though declared locally
  - All variables from a superclass exist in the subclass, but some (private) cannot be accessed directly

---

```java
public class Vehicle {
    protected int plateNum;  // plate #
    private int numWheels;   // # of wheels

    public String toString() {
        return "License plate " + plateNum;
    }
} // class Vehicle

public class Plane extends Vehicle {
    protected double wingSpan;
    private boolean hasPropeller;

    public void writeProperties() {
        System.out.println(plateNum);  // ?
        System.out.println(numWheels); // ?
        if (hasPropeller)              // ?
            System.out.println("Prop plane");
    }
} // class Plane
```

---

### Reserved word super

- **Invoke constructor of superclass**

  ```java
  super(parameter-list);
  ```

  - **parameter-list** must match that in superclass’ constructor

- **Access methods and variables from superclass**
new definition of class Vehicle

```java
public class Vehicle {
    protected int plateNum; // plate #
    private int numWheels; // # of wheels

    public Vehicle(int plate, int wheels) {
        plateNum = plate;
        numWheels = wheels;
    }

    public String toString() {
        return "License plate " + plateNum;
    }

    private void foo() {
        System.out.println("Boo");
    }

} // class Vehicle
```

new definition of class Plane

```java
public class Plane extends Vehicle {
    protected double wingSpan;
    private boolean hasPropeller;

    public Plane(int plate, int wheels, double span, boolean prop) {
        // must call superclass' constructor
        super(plate, wheels);
        wingSpan = span;
        hasPropeller = prop;
    }

    public void writeProperties() {
        System.out.println(plateNum);
        // System.out.println(numWheels);
        if (hasPropeller)
            System.out.println("Prop plane");
    }

    public void poo() { System.out.println(); }

} // class Plane
```

Overriding methods

- Subclass can **override** definition of inherited method in favor of its own
- New method in subclass must have same signature as superclass (but has different method body)
- Which method gets used??
  *The object that is used to invoke a method determines which version is used*
- Method declared to be **final** cannot be overridden
- Do not confuse **overriding** with **overloading**!
Important ideas in inheritance

- Use different hierarchies for different problems
- Single inheritance
- Inherited features are continually passed down the line
- Keep common features as high in the hierarchy as reasonably possible
- Use the superclass’ features as much as possible
- "Inherited" ⇒ "can be accessed as though declared locally"
  
  (private variables in superclass exists in subclasses, they just cannot be accessed directly)

```
public class Client {
    public static void main(String[] args){
        Vehicle v1 = new Vehicle(7443,4);
        Plane pl = new Plane(333,6,35,true);
        System.out.println(v1);  //?
        v1.foo();                //?
        v1.poo();                //?
        System.out.println(pl);  //?
        pl.foo();                //?
        pl.poo();                //?
    }
} //class Client
```

Polymorphism

- "Have many forms"

- A polymorphic reference can refer to different objects (related through inheritance) at different times

```
Vehicle[] mover = new Vehicle[5];

mover[0] = new Vehicle(...);
mover[1] = new Plane(...);
mover[2] = new Plane(...);
mover[3] = mover[1];
```

Another polymorphic example

```
Vehicle[] mover = new Vehicle[5];

mover[0] = new Vehicle(...);
mover[1] = new Plane(...);
mover[2] = new Plane(...);
mover[3] = mover[1];
```