Announcements

- P6 posted. Due Thurs, 5/5, at 6pm
- Prelim 3: pick up at Carpenter during consulting hrs only, in general 2:30-6pm daily (see Staff link of website for exact times)
  Today is last day to submit regrade request

Previous Lecture:
- Overriding methods
- Using super to access methods in superclass

Today's Lecture:
- Polymorphism
- Object class
- Abstract

Reading:
- Sec 9.2, Sec 8.3 exclude “interface hierarchies”

Suppose class Plane extends Vehicle

```
Vehicle mover; //a Vehicle reference
Plane flyer;   //a Plane reference
mover= new Vehicle(...);
flyer= new Plane(...);
// A plane is a vehicle
mover= new Plane(...);
mover= flyer;
// A vehicle is not a plane
flyer= new Vehicle(...); //invalid
```

Another polymorphic example

```
Vehicle[] mover = new Vehicle[5];
mover[0]= new Vehicle(...);
mover[1]= new Plane(...);
mover[2]= new Plane(...);
mover[3]= mover[1];
```

Accessing methods/variables through polymorphic references

Ask two questions:
1. What determines whether a method/variable can be accessed?
2. For an overridden method, what determines which version gets invoked?

```
class V {
    int num1;
    void vmethod() { num1++; }
}
class W extends V {
    int num2;
    void wmethod() { num2++; }
    //wmethod cannot be called thru reference //of type V
}
```
Client code:

```java
V x= new W();
System.out.println(x.num1);
System.out.println(x.num2);  //invalid
x.vmethod();    //invalid

System.out.println( ((W) x).num2 );
((W) x).wmethod();
```

Client code:

```java
V x;  // x references type V or its subtype
String s= "Which type, V or W? ";
System.out.println(s);
char input= Keyboard.readChar();
if (input=='V')
x= new V();
else
  x= new W();

System.out.println(x.num1);
System.out.println(x.num2);  //invalid
x.vmethod();
x.wmethod();    //invalid
```

### Accessing overridden methods through polymorphic references

The **type of the object** determines which version of the method gets invoked.

- Class `Vehicle` has method `toString` that class `Plane` overrides:

  ```java
  Vehicle v1= new Vehicle(...);
  Vehicle v2= new Plane(...);
  System.out.println(v1); //Vehicle’s version
  System.out.println(v2);  //Plane’s version
  ```

### The Object class

If a class is not explicitly defined to be the child of an existing class, it is assumed to be the child of the **Object class**

⇒ All classes are derived from the **Object class**

```java
class Room
  is the same as
class Room extends Object
```

### The Object class

- If a class is not explicitly defined to be the child of an existing class, it is assumed to be the child of the **Object class**

⇒ All classes are derived from the **Object class**

- `toString`: “default” instance method defined in the **Object class**
- Arrays are **Objects**, literally!
Abstract class

- A placeholder in a class hierarchy that represents a generic concept
- Cannot be instantiated
- Modifier: `abstract`
  ```java
  public abstract class Geometry
  ``
- Can contain abstract methods
  ```java
  public abstract double Area();
  ``
- Subclasses of abstract classes will “fill out” these abstract methods