- Previous Lecture:
  - Overriding a method; accessing components in superclass
  - `class Bathroom extends Room`
- Today's Lecture:
  - `static` variables and methods
  - Polymorphism
  - Review
- Reading:
  - Sec 4.2

Client code:

```
Room r1 = new Room(5);
Bathroom r2 = new Bathroom(20,true);

// Method invocation
 r1.report();
 r2.report();

 r1.clean();  r1.report();
 r2.clean();  r2.report();
 r2.majorCleanUp();  r2.report();
 r1.majorCleanUp();  // ?
```


Recap—extending a class

- Subclass is a more specific version of the superclass
- Subclass inherits public and protected members from superclass
- You can declare new variables and methods in subclass
- You can override inherited methods with new definition
- Do not re-declare variables in subclass
  Called *shadowing*—not good practice in general

`static` methods & variables

- Do not re-declare `static` components!
- Same rules for inheritance (accessibility) with respect to visibility modifiers
- Method: implicitly **`final`**
- Variable: same memory space as superclass

Polymorphism

- "Have many forms"
- A *polymorphic* reference refers to different objects (related through inheritance) at different times

Polymorphism, example

```
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
```

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?

Accessing methods/variables through polymorphic references

The ***type of the reference*** determines the methods and fields that can be accessed

```
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:

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


Client code:

```
V x;  // x references type V or its subtype
String s = "Which type, V or W? ";
System.out.print(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, need cast
x.vmethod();
x.wmethod();  //invalid, need cast
```


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:

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


More client code:

```
// Polymorphism
 Room r3 = new Bathroom(20,false);
 System.out.println(r3);  //?
 r3.clean();              //?
 r3.report();             //?
 r3.majorCleanUp();       //?

 r3.report();

// Static methods and variables
 Room.countRooms();
 Bathroom.countRooms();
```