1. Log in
2. Today We Begin Unit 5
   - Inheritance (overriding and more)
   - Polymorphism
   - Abstract Classes
   - Interfaces

Oct 16-7:52 AM

Polymorhic - When a method has been overridden it is considered to be polymorphic.

Nov 11-3:13 PM

Polymorhic - When a method has been overridden it is considered to be polymorphic.

Polymorphism - Selecting the correct method to use when a method is polymorphic (when there are mulitple choices due to overriding).

Nov 11-3:13 PM

A very important difference...

Method Overloading VS Method Overriding

Nov 13-12:26 PM

A very important difference...

Method Overloading **VS** **Method Overriding**

```
public int addIt(int a){
    return a++;
}

public double addIt(double a) {
    return a+a;
}

public int addIt(int a, int b) {
    return a+b;
}
```

Overloading - more than one method with the same name. (Must choose which fits)

Nov 13-12:26 PM

A very important difference...

Method Overriding

```
public class myClass1 {
    public int a;
    public myClass1( ) {
        a=5;
    }
    public int addIt(int x) {
        return x+x;
    }
}
```

```
public class myClass2 extends myClass1
    super( );
    public int b;
    myClass2( ) {
        b=3;
    }
    @Override                    //use Capital O
    public int addIt(int y) {
        return y*y;
    }
```

Overriding - Over-writes a method. "Replaces a method" when appropriate

Nov 13-12:26 PM

A very important difference...

Method Overloading VS Method Overriding

```
public int addIt(int a){
    return a++;
}
public double addIt(double a) {
    return a+a;
}
public int addIt(int a, int b) {
    return a+b;
}
```

Overloading -

1. The computer makes the decision on which to use when the program is compiled.
2. Compiler compares the method header to see which one works.
3. Compiler checks to see which one can run and find the one to use.
4. This is called Early Binding or Static Binding (done before running).

Nov 13-12:26 PM

A very important difference...

Method Overriding

```
public class myClass1 {
    public int a;
    public myClass1( ) {
        a=5;
    }
    public int addIt(int x) {
        return x+x;
    }
}
```

```
public class myClass2 extends myClass1
    super( );
    public int b;
    myClass2( ) {
        b=3;
    }
    @Override
    public int addIt(int y) {      //use Capital O
        return y*y;
    }
```

Overriding -

1. The computer makes the decision on which to use while it is running.
2. While running, all the methods might work but the choice on which to use is made.
3. The decision is made while the program is running (run-time decision).
4. This is called Late Binding or Dynamic Binding (done "on-the-spot" when needed).

Nov 13-12:26 PM

A new scenario: The Fishing Trip

```
public class LaunchBoat {
    public void step1( ){
        System.out.println(" back in");
        step2( );
    }
    public void step2( ){
        System.out.println(" launch boat");
        step3( );
    }
    public void step3( ){
        System.out.println(" go park");
    }
}
```

Nov 13-1:07 PM

A new scenario: The Fishing Trip

Launch Boat

No Instance Variables
No Constructor Method

step1( ) //method
step2( ) //method
step3( ) //method

```
public class LaunchBoat {
    public void step1( ){
        System.out.print(" back in");
        step2( );
    }
    public void step2( ){
        System.out.print(" launch boat");
        step3( );
    }
    public void step3( ){
        System.out.print(" go park");
    }
}
```

Nov 13-1:07 PM

A new scenario: The Fishing Trip

What is the output for the following...

```
LaunchBoat launch1 = new LaunchBoat( );
launch1.step1( );
launch1.step2( );
launch1.step3( );
```

```
public class LaunchBoat {
    public void step1( ){
        System.out.print(" back in");
        step2( );
    }
    public void step2( ){
        System.out.print(" launch boat");
        step3( );
    }
    public void step3( ){
        System.out.print(" go park");
    }
}
```

Nov 13-1:07 PM

A new scenario: The Fishing Trip

What is the output for the following...

```
LaunchBoat launch1 = new LaunchBoat( );
launch1.step1( );      //back in launch boat go park
launch1.step2( );      //launch boat go park
launch1.step3( );      //go park
```

Nov 13-1:07 PM

A new class: GoFishing

GoFishing

No Instance Variables
No Constructor Method

step1( ) //method
step2( ) //method
step3( ) //method

```
public class GoFishing {
   public void step1( ){
      System.out.print(" drive boat");
      step2( );
   }
   public void step2( ){
      System.out.print(" find spot");
      step3( );
   }
   public void step3( ){
      System.out.print(" bait hook");
   }
}
```

Nov 13-1:07 PM

A new class: GoFishing

What is the output for the following...

```
GoFishing fishing1 = new GoFishing( );
fishing1.step1( );
fishing1.step2( );
fishing1.step3( );
```

```
public class GoFishing {
   public void step1( ){
      System.out.print(" drive boat");
      step2( );
   }
   public void step2( ){
      System.out.print(" find spot");
      step3( );
   }
   public void step3( ){
      System.out.print(" bait hook");
   }
}
```

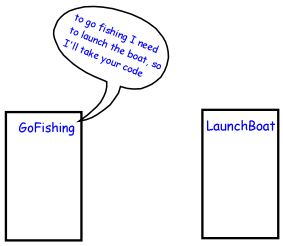
Nov 13-1:07 PM

A new scenario: The Fishing Trip

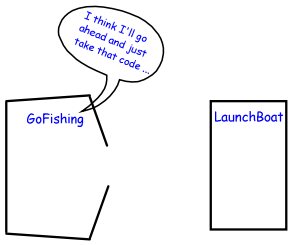
What is the output for the following...

```
GoFishing fishing1 = new GoFishing( );
fishing1.step1( );          //drive boat find spot bait hook
fishing1.step2( );          //find spot bait hook
fishing1.step3( );          //bait hook
```

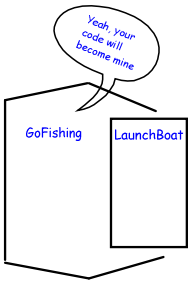
Nov 13-1:07 PM



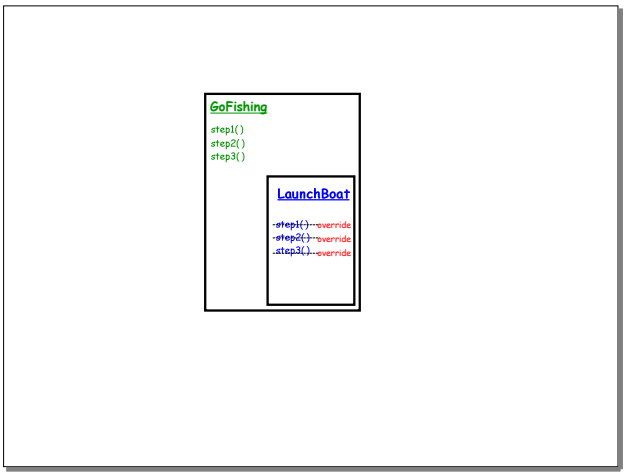
Nov 13-2:07 PM



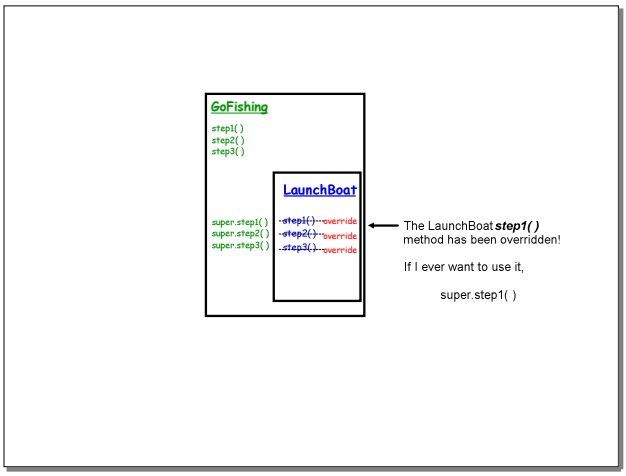
Nov 13-2:09 PM



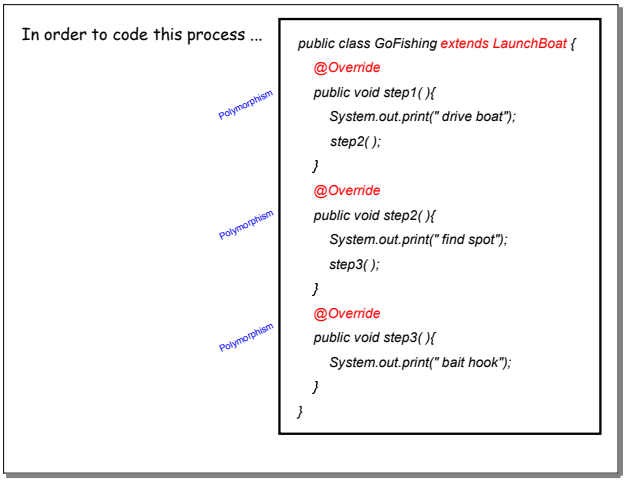
Nov 13-2:10 PM



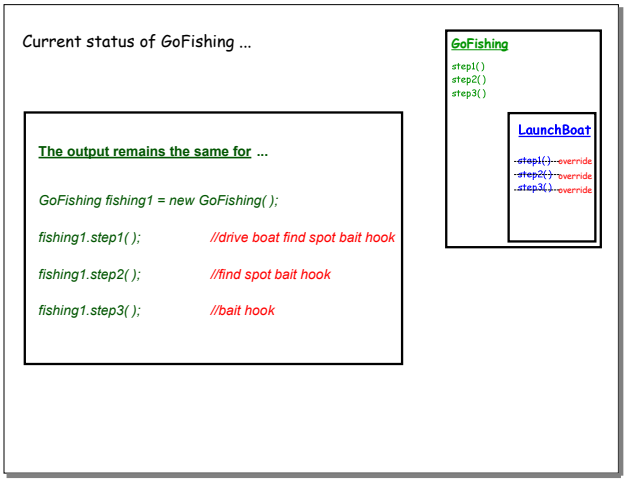
Nov 13-6:24 PM



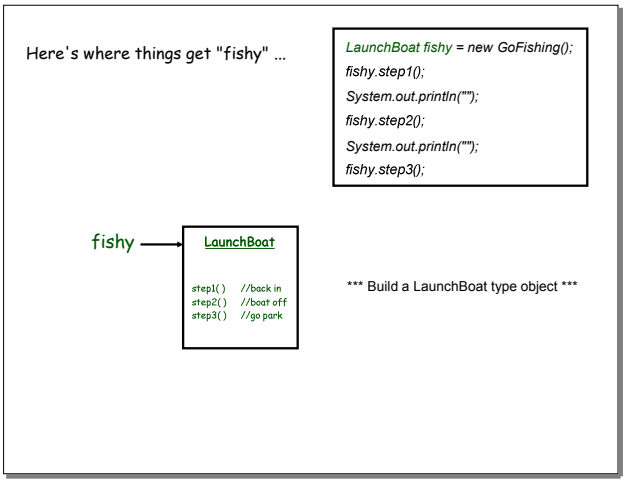
Nov 13-6:24 PM



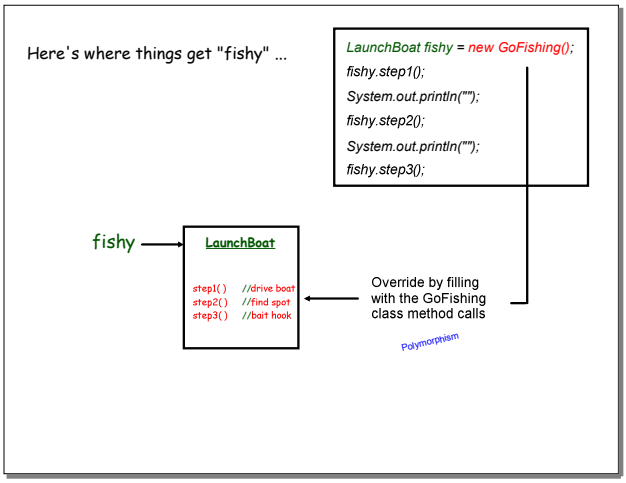
Nov 13-2:13 PM



Nov 13-2:10 PM



Nov 13-2:25 PM



Nov 13-2:25 PM

Comprehending that little fishy...

The output remains the same for ...

```
LaunchBoat fishy = new GoFishing( );
fishy.step1( );        //drive boat find spot bait hook
fishy.step2( );        //find spot bait hook
fishy.step3( );        //bait hook
```

Nov 13-2:10 PM

Things are about to get harder, make this change ...
What if GoFishing wants to use one of the LaunchBoat methods?

super ← call to the superclass!!!

```
public class GoFishing extends LaunchBoat {
   @Override
   public void step1( ){
       super.step1( );
       System.out.print(" drive boat");
       step2( );
   }
   @Override
   public void step2( ){
       System.out.print(" find spot");
       step3( );
   }
   @Override
   public void step3( ){
       System.out.print(" bait hook");
   }
}
```

Nov 13-2:38 PM

```
public class GoFishing extends LaunchBoat {
   @Override
   public void step1( ){
       super.step1( );
       System.out.print(" drive boat");
   }
   @Override
   public void step2( ){
       System.out.print(" find spot");
       step3( );
   }
   @Override
   public void step3( ){
       System.out.print(" bait hook");
   }
}
```

```
public class LaunchBoat {
   public void step1( ){
       System.out.print(" back in");
       step2( );
   }
   public void step2( ){
       System.out.print(" launch boat");
       step3( );
   }
   public void step3( ){
       System.out.print(" go park");
   }
}
```

```
LaunchBoat fishy = new GoFishing();
fishy.step1( );
```

fishy → LaunchBoat
step1( ) //subclass
step2( ) //subclass
step3( ) //subclass

Nov 13-2:38 PM

```
public class GoFishing extends LaunchBoat {
   @Override
   public void step1( ){
       super.step1( );
       System.out.print(" drive boat");
   }
   @Override
   public void step2( ){
       System.out.print(" find spot");
       step3( );
   }
   @Override
   public void step3( ){
       System.out.print(" bait hook");
   }
}
```

```
public class LaunchBoat {
   public void step1( ){
       System.out.print(" back in");
       step2( );
   }
   public void step2( ){
       System.out.print(" launch boat");
       step3( );
   }
   public void step3( ){
       System.out.print(" go park");
   }
}
```

```
LaunchBoat fishy = new GoFishing();
fishy.step1( );
    super.step1( );
    System.out.print(" drive boat");
```

fishy → LaunchBoat
step1( ) //subclass
step2( ) //subclass
step3( ) //subclass

Nov 13-2:38 PM

```
public class GoFishing extends LaunchBoat {
   @Override
   public void step1( ){
       super.step1( );
       System.out.print(" drive boat");
   }
   @Override
   public void step2( ){
       System.out.print(" find spot");
       step3( );
   }
   @Override
   public void step3( ){
       System.out.print(" bait hook");
   }
}
```

```
public class LaunchBoat {
   public void step1( ){
       System.out.print(" back in");
       step2( );
   }
   public void step2( ){
       System.out.print(" launch boat");
       step3( );
   }
   public void step3( ){
       System.out.print(" go park");
   }
}
```

```
LaunchBoat fishy = new GoFishing();
fishy.step1( );
    super.step1( );          System.out.print(" back in");
    System.out.print(" drive boat");   step2( );
```

fishy → LaunchBoat
step1( ) //subclass
step2( ) //subclass
step3( ) //subclass

Nov 13-2:38 PM

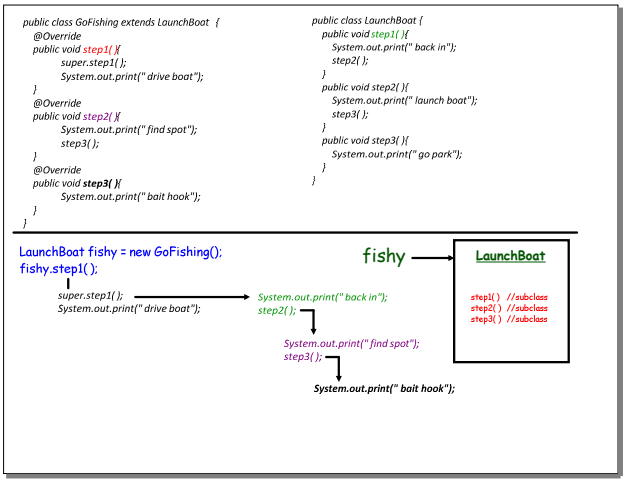
```
public class GoFishing extends LaunchBoat {
   @Override
   public void step1( ){
       super.step1( );
       System.out.print(" drive boat");
   }
   @Override
   public void step2( ){
       System.out.print(" find spot");
       step3( );
   }
   @Override
   public void step3( ){
       System.out.print(" bait hook");
   }
}
```

```
public class LaunchBoat {
   public void step1( ){
       System.out.print(" back in");
       step2( );
   }
   public void step2( ){
       System.out.print(" launch boat");
       step3( );
   }
   public void step3( ){
       System.out.print(" go park");
   }
}
```

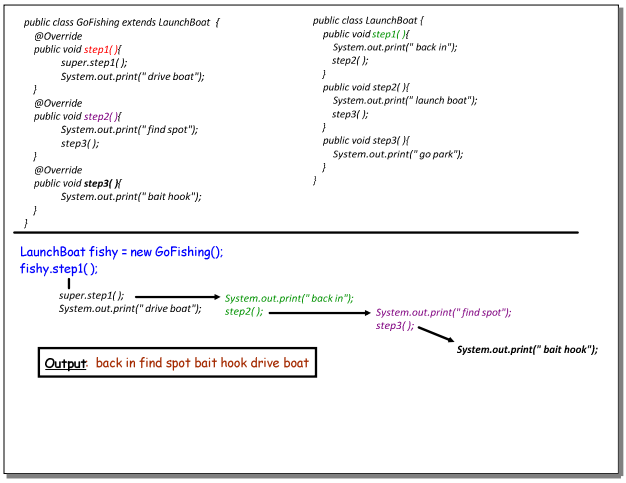
```
LaunchBoat fishy = new GoFishing();
fishy.step1( );
    super.step1( );          System.out.print(" back in");
    System.out.print(" drive boat");   step2( );
                                       System.out.print(" find spot");
                                       step3( );
```

fishy → LaunchBoat
step1( ) //subclass
step2( ) //subclass
step3( ) //subclass

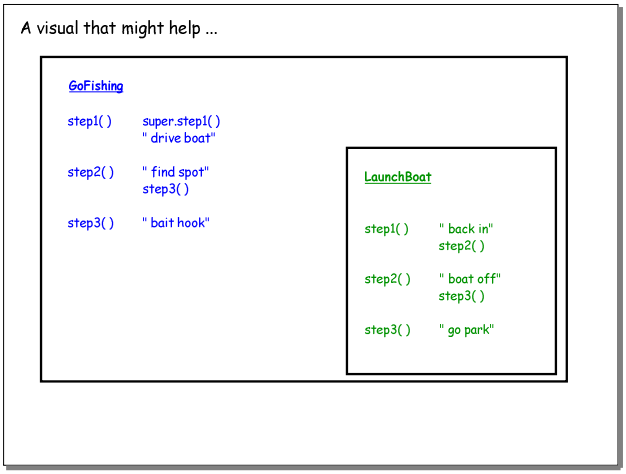
Nov 13-2:38 PM



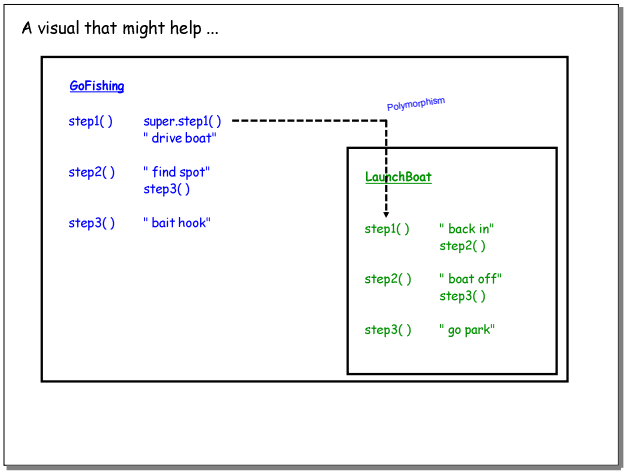
Nov 13-2:38 PM



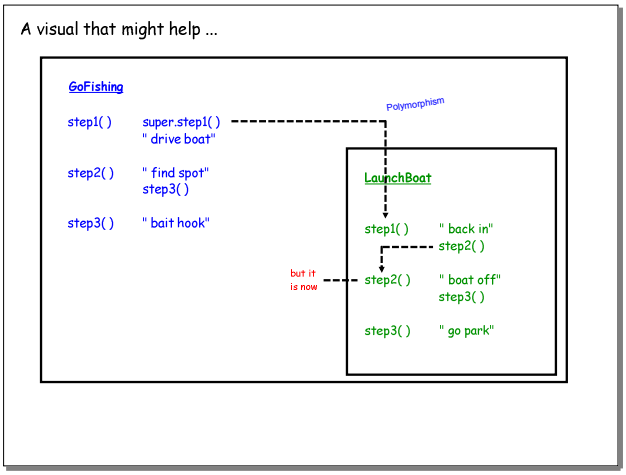
Nov 13-2:38 PM



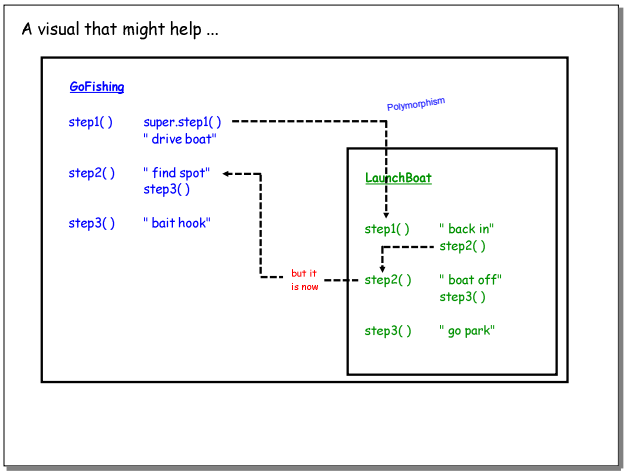
Nov 13-6:35 PM



Nov 13-6:35 PM

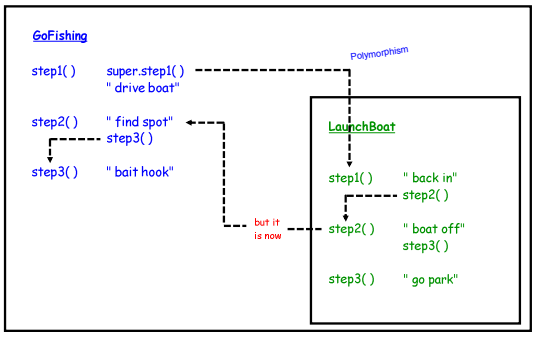


Nov 13-6:35 PM



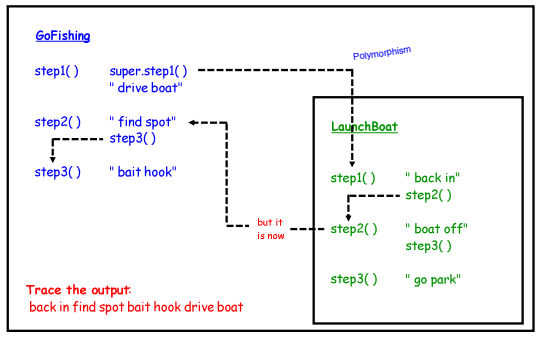
Nov 13-6:35 PM

A visual that might help ...



Nov 13-6:35 PM

A visual that might help ...



Nov 13-6:35 PM

Things to do ...

1. Wrap Up Unit 5 WS 01-03
2. Work on Unit 5 WS04 More on Inheritance

Oct 16-9:12 AM