```
// Step 3
/**
 * This is our code! It's goal is to play with subclasses.
 * CS 312 - Lab 2
 * @author Emma Smith, Mari Sisco, Aidan Shaughnessy
 * @version 1.0 15 Sept 2022
 */

public class Tester
{
        public static void main (String [] args)

    {
        System.out.println("hello world");
        LeftArm arm1  =   new LeftArm();
        System.out.println(arm1);

        GrippingArm arm2 = new GrippingArm();
        System.out.println(arm2);

        StarFish justSome  = new StarFish();
        PisasterBrevispinus patrick = new PisasterBrevispinus();
        int theAnswer = 42;
        //step11
        System.out.println("Step 11");
        justSome.arm(theAnswer);
        justSome.arm(8.2);
        justSome.arm("patrick");
        justSome.arm(arm1);
        justSome.arm(arm2);


        System.out.println("Step 12");
        patrick.arm(theAnswer);
        patrick.arm(8.2);
        patrick.arm("patrick");
        patrick.arm(arm1);
        patrick.arm(arm2);
        patrick.leg("leg");

        //Step 13
        justSome.arm(patrick.leg("leg"));

        //Step 17
        byte num1  = 8;
        float num2 = 2.718f;;
        patrick.arm(num1);
        //We think that widening casting is used since the parameter
        // is entered as a byte and turns into an int
        // We were correct because the integer message outputtedn
        //
        // Step 18
        patrick.arm(num2);
        //We think that widening casting is used since the parameter
        //is entered as a float and turns into a double
        //We were correct, the output message was as a double/

        //Step 19
        patrick.arm((int) num2);
        // We think that narrowing casting is used since the parameter
        // is entered as a float but the int prefix will narrow it

        //Step 20
        StarFish iWasCastUp = new StarFish();
        iWasCastUp = patrick; //implicit
        iWasCastUp = (PisasterBrevispinus) patrick; // explicitly

        // Step 21
        PisasterBrevispinus iWasCastDown = new PisasterBrevispinus();
```



```
        //iWasCastDown = justSome; // implicitly
        //ERROR: incompatible types StarFish cannot be converted to PisasterBrev
ispinus
        //iWasCastDown = (StarFish) justSome; //explicitly
        //ERROR: incompatible types StarFish cannot be converted to PisasterBrev
ispinus
        //Cannot be downcast implicitly or explicitly


        //Step 24A
        long num3 = 123456789;;
        patrick.arm(num3);

        // this results in a long being widened to a double
        // this is obvious because it outputs the double message
    }
}

// Step 5
 class LeftArm
{
    // Step 1
    // Step 24B
    // Taking @Override out of this method does not affect whether the code runs
 or not
    // We think that this is because LeftArm isn't a subclass of anything, just
a concrete class
    // so it has nothing to override
    @Override
    public String toString()
    {
        return ("I am a left arm");
    }
}

// Step 6
 class GrippingArm
{
    // Step 15
    @Override
    public String toString()
    {
        return ("I am a gripping arm");
    }
}

//Step 7
class StarFish
{
    public StarFish()
    {
    }

    void arm(int i)
    {
        System.out.println("A StarFish's integer arm " + i);
    }

    void arm(double d)
    {
        System.out.println("A StarFish's double arm " + d);
    }

    void arm(String s)
    {
        System.out.println("A StarFish's String arm " + s);
    }

    void arm(LeftArm l)
```

```
    {
        System.out.println("A StarFish's LeftArm arm " + l);
    }

    void arm(GrippingArm g)
    {
        System.out.println("A StarFish's GrippingArm arm " + g);
    }

}


//Step 8
class PisasterBrevispinus extends StarFish
{
    @Override
    void arm(int i)
    {
        System.out.println("A PisasterBrevispinus's integer arm is different " + i);
    }

    @Override
    void arm(GrippingArm g)
    {
        System.out.println("A PisasterBrevispinus's GrippingArm arm " + g);
    }

    // Step 14
    //We think that both arm methods will work because we need to override the S
tarFish equivalents
    //WE think that the leg method will not work because there is nothing to ove
rride
    //after testing the compiler we were correct e
    String leg(String s)
        {
        return("must be Patrick... only Patrick..." + s  );
    }

    //Step 22 and 23
    //@Override
    //Including the Override gives an error message because the StarFish class d
oes not have a method with
    //a short parameter
    void arm(short s)
    {
        System.out.println("a PisasterBrevispinus's short arm " + s);
    }
}

//Step 24
//For our first experiment we want to learn more about casting so we will repeat
 step  17 to better understand the concept
//For our second experiment we want to learn more about the @Override command so
 we will retry adding/removing it from various methods
//
//
```