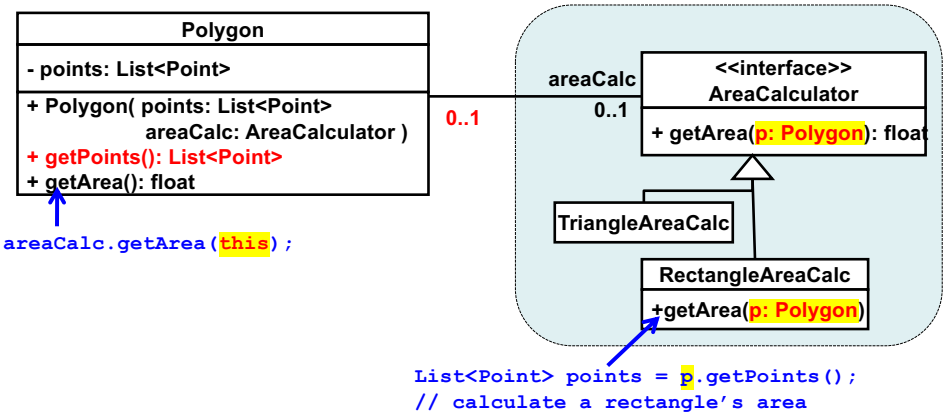
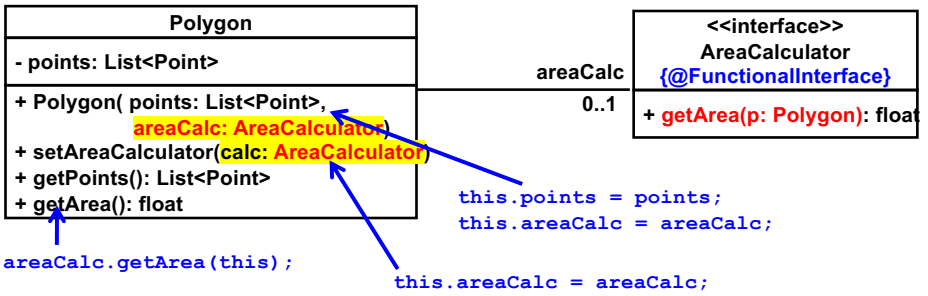
Exercise: Area Calculation w/ *Strategy*



Client of Polygon:

```
List<Point> points = List.of( ... );

Polygon p = new Polygon( points, new RectangleAreaCalc () );
p.getArea();
```


Area Calculation w/ LEs

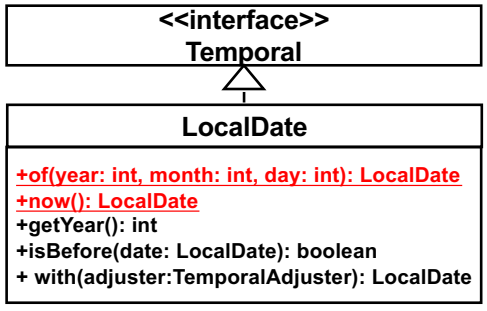


Client of Polygon:

```
Polygon polygon = new Polygon(points, (Polygon p)->{p.getPoints();
                                                    ...} );

polygon.setAreaCalculator( (Polygon p)->{p.getPoints();
                                         ...; } );
```

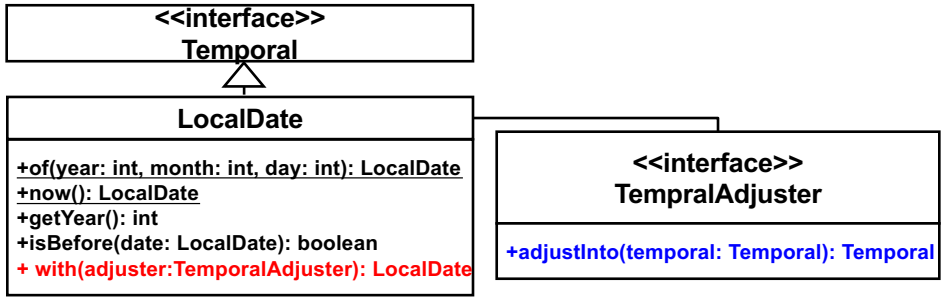

Exercise: Using Date and Time API w/ LEs

- **`LocalDate, LocalTime, LocalDateTime`**
  - Used to represent date and time without a time zone (time difference)




- ```
  LocalDate today =
      LocalDate.now();
  System.out.println(today);
  ```

- ```
  LocalDate newYearDay19 =
      LocalDate.of(2019, 1, 1);
  System.out.println(today);
  ```

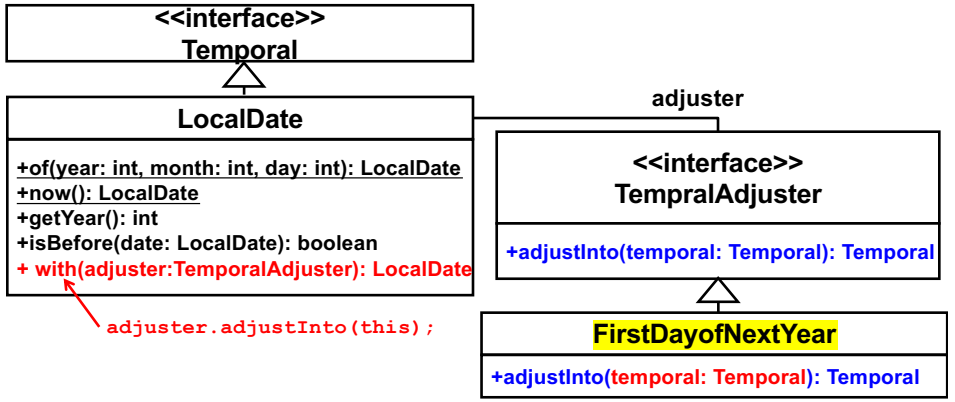



- **`with()`**
  - Returns a local date that is adjusted from "this" (current) date.
- **`TemporalAdjuster`**
  - Interface for *Strategy* classes that implement particular date adjustment algorithms.

Date Adjustment

- **`TemporalAdjusters`**
  - Offers a series of static factory methods to create and return `TemporalAdjuster` instances that implement common date adjustment algorithms.
    - e.g., Getting the first or last day of the month
    - Getting the first day of next month
    - Getting the first or last day of the year
    - Getting the first day of next year
    - Getting the first or last day-of-week within a month, such as "first Wednesday in June"
    - Getting the next or previous day-of-week, such as "next Thursday"

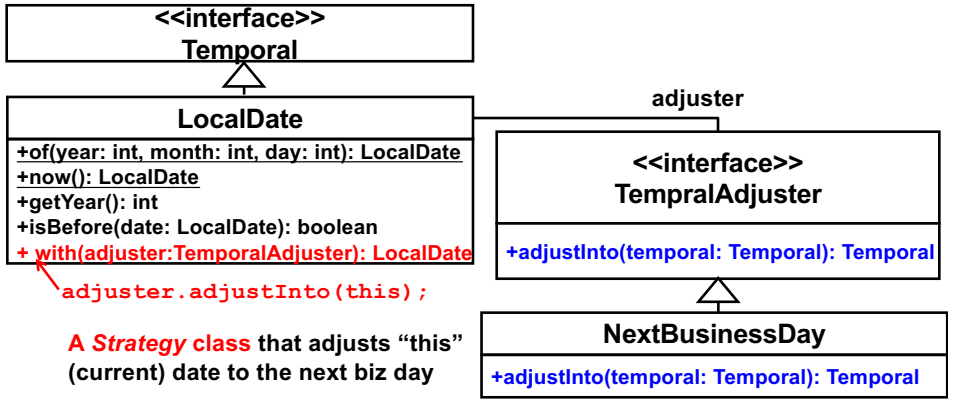
```
LocalDate today = LocalDate.now();
LocalDate nextNewYearDay =
    today.with( TemporalAdjustors.firstDayofNextYear() );
```



A *Strategy* class that adjusts "this" (current) date to the next new year day.

```
LocalDate today = LocalDate.now();
LocalDate nextNewYearDay =
    today.with( TemporalAdjustors.firstDayofNextYear());
```

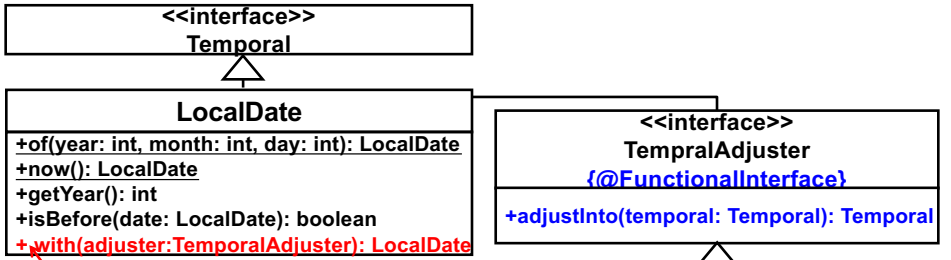
- If `TemporalAdjusters` doesn't provide a date adjustment algorithm that you want…
  - You can implement your own by defining a *Strategy* class.
    - Finding the next business day
    - Finding the next Christmas
    - Finding the next July 4th



A *Strategy* class that adjusts "this" (current) date to the next biz day

```
LocalDate today = (LocalDate)temporal;
if( today.getDayOfWeek() == DayOfWeek.FRIDAY){
  return today.plusDays(3); }
else if( today.getDayOfWeek() == DayOfWeek.SATURDAY){
  return today.plusDays(2); }
else{
  return today.plusDays(1); }
```

```
LocalDate today = LocalDate.now();
LocalDate nextBizDay = today.with( new NextBusinessDay() );
```




```
adjuster.adjustInto(this);
```

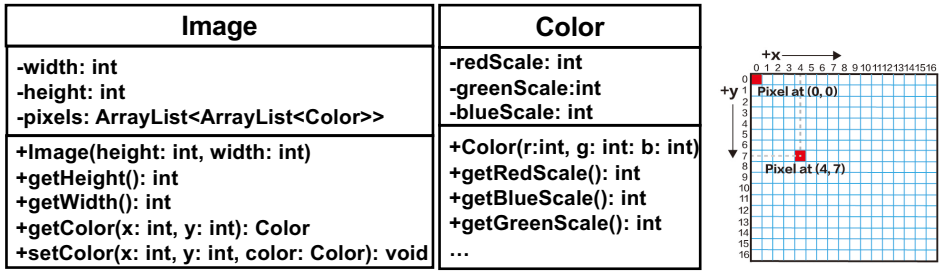
The *Strategy* class `NextBusinessDay` to be replaced by a LE.

```
LocalDate today = (LocalDate)temporal;

if( today.getDayOfWeek() == DayOfWeek.FRIDAY){
  return today.plusDays(3); }

else if( today.getDayOfWeek() == DayOfWeek.SATURDAY){
  return today.plusDays(2); }

else{
  return today.plusDays(1); }
```

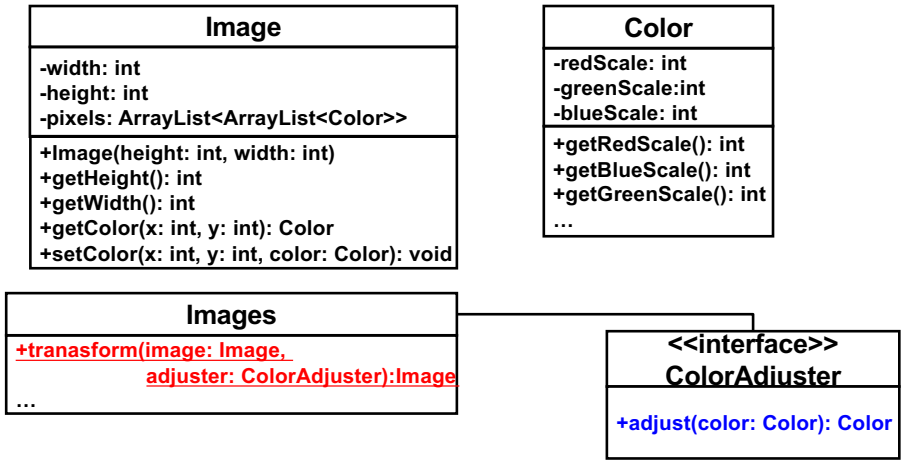
-
```
LocalDate today = LocalDate.now();
LocalDate nextBizDay = today.with( ... );
```


Exercise: Color Adjustment/Filtering in Hypothetical GUI API

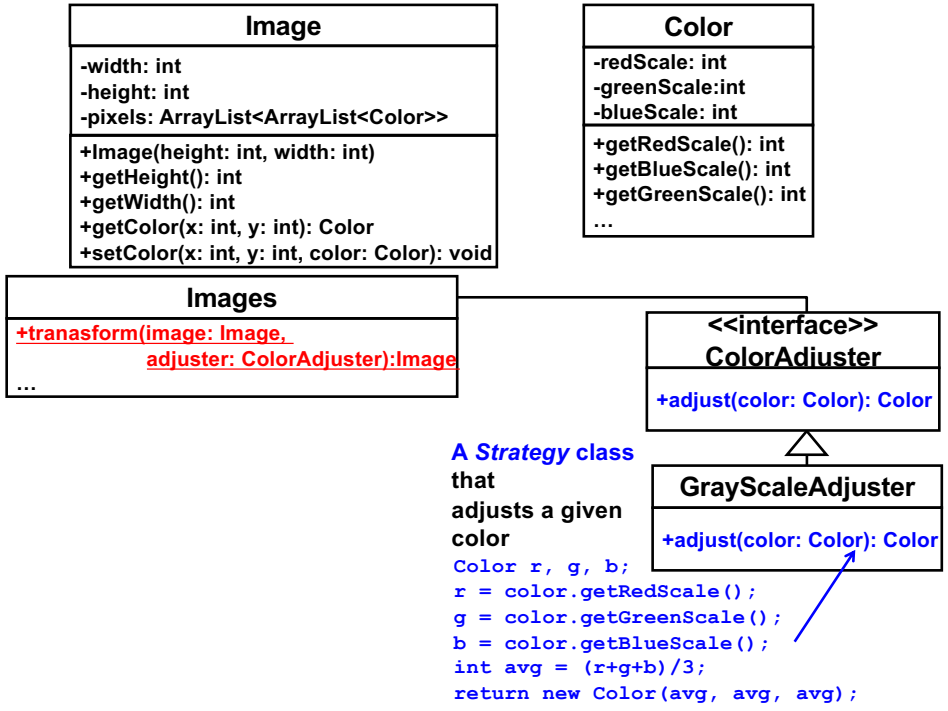
(Simplified Version of Java FX)



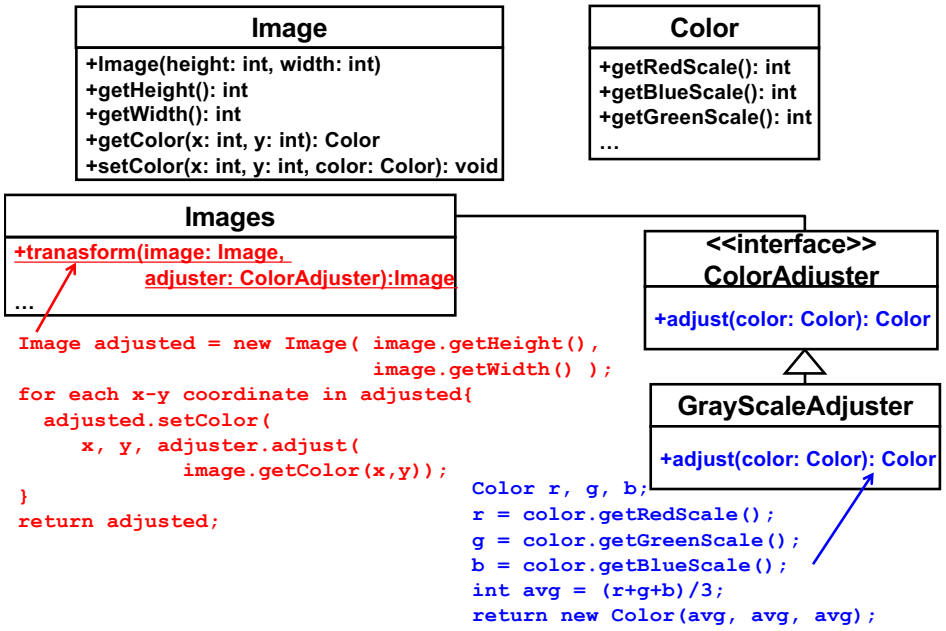
-
```
Image image = new Image(17, 17);       // Image size: 17x17

image.setColor(4,7, new Color(255,255,255));// Set red color

image.getColor(4,7).getRedScale();         // 0
image.getColor(4,7).getBlueScale();        // 0
image.getColor(4,7).getGreenScale();       // 0
```

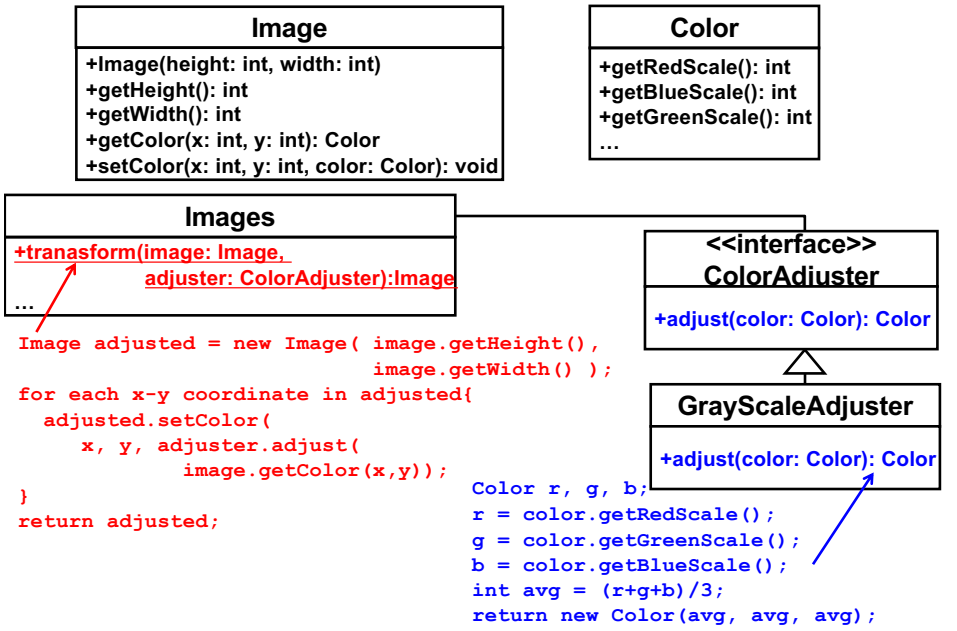



- **`Images`**
  - Utility class to process images
- **`transform()`**
  - Returns a color-adjusted copy of a given image.
- **`ColorAdjuster`**
  - Interface for *Strategy* classes that implement particular color adjustment algorithms.



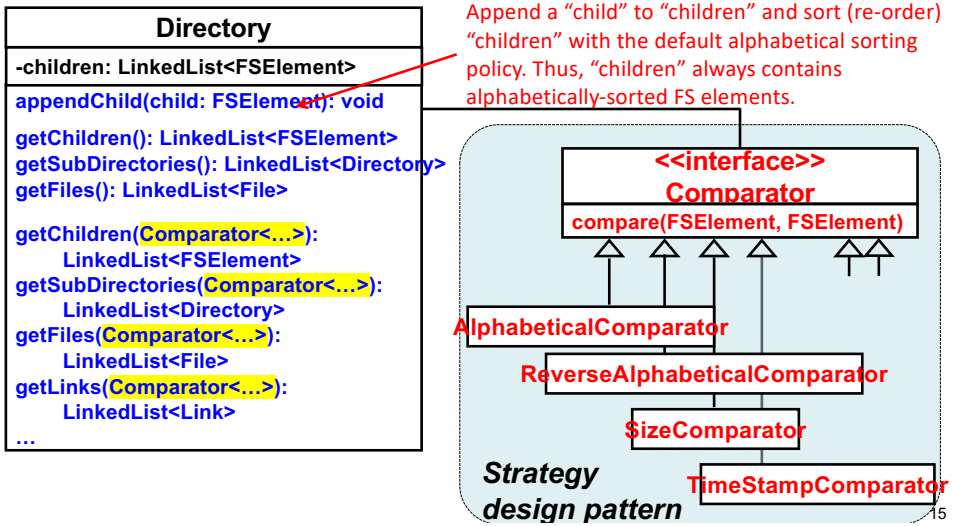

```
Color r, g, b;
r = color.getRedScale();
g = color.getGreenScale();
b = color.getBlueScale();
int avg = (r+g+b)/3;
return new Color(avg, avg, avg);
```





HW 14

- Revise your HW 12 solution with LEs.

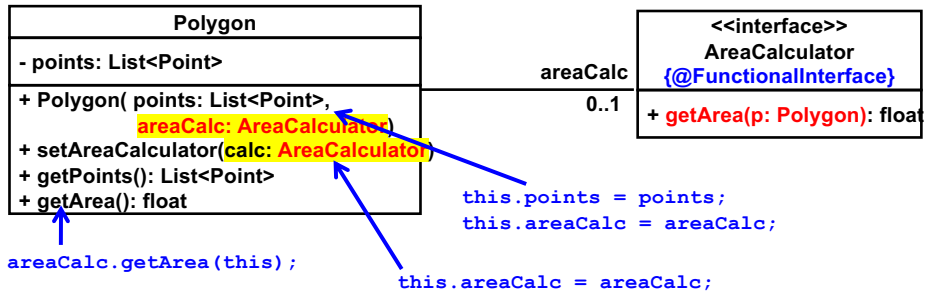


- Instead of defining classes that implement `Comparator<FSElement>`, define the body of each `compare()` method as a LE and pass it to `getChildren()`, `getSubDirectories()`, `getFiles()` and `getFiles()` of `Directory`.
  - Use at least 3 LEs (for at least 3 soring pocilities)
  - No need to change the bodies of `getChildren()`, `getSubDirectories()`, `getFiles()` and `getFiles()`
    - Just change their client code.

Free Variables

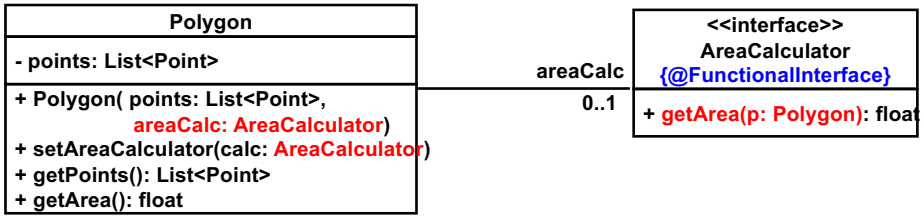
- Variables that….
  - a LE can access, but
  - are not defined in that LE.
    - Not parameters of the LE.
    - Not local variables in the LE.
  - Parameters of an *enclosing* method
  - Local variables of an *enclosing* method
  - Data fields of an *enclosing* class
- A lambda expression can access those free variables.

Recap: Area Calculation w/ LEs

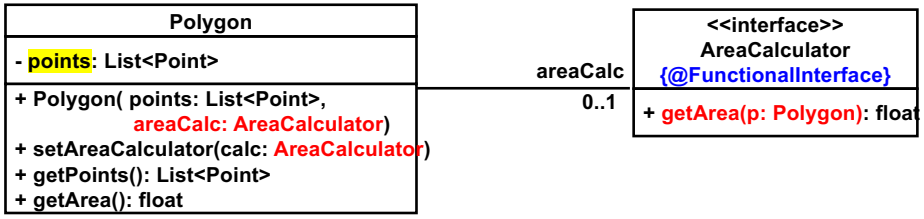


Client of Polygon:

```
Polygon polygon = new Polygon(points, (Polygon p)->{p.getPoints();
                                                    ...} );

polygon.setAreaCalculator( (Polygon p)->{p.getPoints();
                                         ...; } );
```


```
Polygon polygon = new Polygon(...);
polygon.setAreaCalculator( (Polygon p)->{ p.getPoints()...;
                                          ...; } );
```



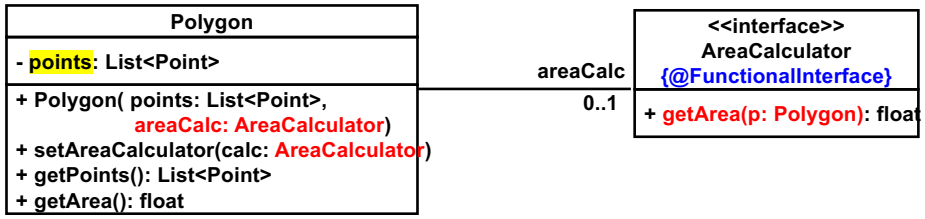

```
Polygon polygon = new Polygon(...);
polygon.setAreaCalculator( ()->{ if(points.size()==3){
                                 ...; } );
```

A Note on Free Variables

- The value of a free variable must be fixed (or immutable).
  - Once a value is assigned to the variable, no re-assignments (value changes) are allowed.
- Traditionally, immutable variables are defined as `final`; free variables are often defined as `final`.
- In fact, a LE can access the variables that are not `final`, but they still have to be *effectively* `final`.
  - Even if they are not `final`, they need to be used as `final` if they are to be used in LEs.

Note that …

- The data field `points` is effectively final.
  - as far as the instance of `List` in `points` is never replaced.
    - `points` contains a reference/pointer to an instance of `List`.
  - You can dynamically change list elements.




```
Polygon polygon = new Polygon(...);
polygon.setAreaCalculator( ()->{ if(points.size()==3){...} });

polygon.addPoint( new Point(...) );
polygon.setAreaCalculator(()->{ if(points.size()==4){...} });
```


LEs in JUnit

- `Assertions.assertThrows(..., ...)`
  - Asserts that a given LE throws a particular exception.
  - `assertThrows(RuntimeException.class, ()->{Foo f = new Foo(); f.methodUnderTest(...);})`
    - `assertThrows()` returns `true`, if the code block (i.e. `methodUnderTest()`) throws a `RuntimeException` (an instance of `RuntimeException` or its subclass).

- `Assertions.assertThrowsExactly(..., ...)`
  - `assertThrows(RuntimeException.class, ()->{Foo f = new Foo(); f.methodUnderTest(...);})`
    - `assertThrows()` returns `true`, if the code block (i.e. `methodUnderTest()`) throws a `RuntimeException` exactly. Subclasses of `RuntimeException` are excluded.

- `Assertions.assertDoesNotThrows(..., ...)`
  - Asserts that a given LE does NOT throw a particular exception.

An Example Negative Test

- Class under test

```
public class Calculator{
 public float multiply(float x,
                       float y){
   return x * y;
 }
 public float divide(float x,
                     float y){
   if(y==0){ throw
    new IllegalArgumentException(
     "division by zero");}
   return x/y;
 }
}
```

- Test class

```
import static org.junit.jupiter.api.
Assertions.*;

import org.junit.jupiter.api.Test;

public class CalculatorTest{
 @Test
 public void divide5By0(){
   Calculator cut = new Calculator();
   try{
     cut.divide(5, 0);
     fail("Division by zero");
   }
   catch(IllegalArgumentException ex){
     assertEquals("division by zero",
                  ex.getMessage() );
   }
 }
}
```

- Class under test

```
public class Calculator{
 public float multiply(float x,
                       float y){
   return x * y;
 }
 public float divide(float x,
                     float y){
   if(y==0){ throw
    new IllegalArgumentException(
      "division by zero");}
   return x/y;
 }
}
```

- Test class

```
import static org.junit.jupiter.api.
Assertions.*;

import org.junit.jupiter.api.Test;

public class CalculatorTest{
 @Test
 public void divide5By0(){
   assertThrows(
     IllegalArgumentException.class,
     ()->{Calculator cut =
            new Calculator();
          cut.divide(5, 0);});
 }
}
```