I) Refactoring

The next code contains three classes :

```
package shape;
public abstract class Triangle{
  protected int sideA,sideB,sideC;
  public Triangle(int a, int b, int c) {
    this.sideA = a;
    this.sideB = b;
    this.sideC = c;
  }
  public String toString() {
    return "I am a Triangle of the 1st side "+sideA + " , the 2nd
side " +sideB + " and the 3rd side " +sideC;
  }
  public abstract String getArea();
}
```

```
package shape;

public class Equilateral_triangle extends Triangle{

  public Equilateral_triangle(int side) {
    super(side,side,side);
  }

  public String getArea() {
    double val = (double)(this.sideA * this.sideA * 0.433);
    return "-->" + val;
  }

  public String toString() {
    return "I am a Equilateral Triangle!";
  }
}
```

```
package shape;

public class Test{
  public static void main(String [] arg){
    Equilateral_triangle et1 = new Equilateral_triangle(3);
    System.out.println(et1);
    System.out.println("The area is " + et1.getArea());
  }
}
```

Refactor these classes such that the Triangle class

- calculates the area. The getArea() method should not be abstract anymore
- protected attributes become private ones

II) Exception management

An exception is a standard high-level programming paradigm that enable exception behavior management. The mechanism is sufficiently powerful that you can handle standard exception, create your own exception, manage your own exceptions.

1) Runtime exceptions

A runtime exception is an exception that may appear at runtime on certain condition that may not be treated by the caller site. The caller site is the class method that uses the method or the operation.

For instance, the arithmetic division '/' may generate an exception if the denominator is 0. The indice [ ] operator may generate an exception if the index is out of the bounds.

```
package test;
public class TestOne {
  public static void main(String [] arg) {
    System.out.println("-->"+arg[0]);
    System.out.println("-->"+(5/2));
    System.out.println("-->"+(5/0));
  }
}
```

Running this class without parameter will generate the following exception.

```
~/Documents/cours/IST/ $ java test/TestOne
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 0
      at test.TestOne.main(TestOne.java:4)
```

That means at line 4 of the code the [ ] accesses an unreachable index.

Running the class with one parameter will generate the other exception.

```
~/Documents/cours/IST $ java test/TestOne a
-->a
-->2
Exception in thread "main" java.lang.ArithmeticException: / by zero
      at test.TestOne.main(TestOne.java:6)
```

When an exception occurs, the line in the running method is immediately interrupted and returns to the calling method from the call stack.

One may handle exception in surrounding the “suspicious” code with a try/catch block of code. The try section contains several method calls. Some of them may raise an exception. If this occurs the line in the try block is interrupted and the control flow jumps to the catch block. A finally block may be added. The finally block is always executed whereas an Exception occurred or not.

```
package test;
public class TestOne {
  public static void main(String [] arg) {
    try {
      System.out.println("-->"+arg[0]);
      System.out.println("-->"+(5/2));
      System.out.println("-->"+(5/0));
    } catch (Exception e) {
      System.out.println("An Exception occured");
    } finally {
      System.out.println("Finally is always executed");
    }
    System.out.println("The code continues");
  }
}
```

Exercise: write a functionally equivalent code that does not use the length array attribute.

```
package test;
public class Test2 {
  public static void main(String [] arg){
    for (int i=0; i<arg.length; i++){
      System.out.println("->"+arg[i]);
    }
    System.out.println("Finished");
  }
}
```

2) User Exceptions

User exception or compile time exceptions are exceptions declared at the method declaration that forces the caller to handle them. Many standard Java classes declare such exceptions.

Exercise: Compile this class file and find two ways to manage the compile time exception.

```
package test;
public class TestIO {
  public static void main(String [] arg) {
    java.io.FileInputStream dis = new java.io.FileInputStream("/tmp/toto");
  }
}
```

/* Do not hesitate to have a look at the java.io.FileInputStream class description at the javadoc Oracle web site */

A method may generate many exceptions.

```
package shape;
public class Triangle{
  private static final int MIN_SIDE_A = 10;
  private static final int MAX_SIDE_A = 100;

  private int sideA,sideB,sideC;

  public Triangle(int a, int b, int c) throws TooSmallException {
    if ( a < Triangle.MIN_SIDE_A ) {
      throw new TooSmallException(
          "The Triangle is too small change the a size smaller than"
                +Triangle.MIN_SIDE_A);
    }
    if (a > Triangle.MAX_SIDE_A) {
      throw new TooBigException(
         "The Triangle is too Big a "
              +a+" is greater than "+Triangle.MAX_SIDE_A);
    }
    this.sideA = a;
    this.sideB = b;
    this.sideC = c;
  }
}
```

```
package shape;
public class TooBigException extends Exception {
  public TooBigException(String val) {
    super(val);
  }
}
```

```
package shape;
public class TooSmallException extends Exception {
  public TooSmallException(String val) {
    super(val);
  }
}
```

The caller site must handle all exceptions. Since Exception inherits from an Exception parent class, the caller may handle all exception in a generic way or with a specific catch clause for each type of Exception.

```
package shape;
public class Test{
  public static void main(String [] arg) {
    try {
      Equilateral_triangle et1 = new Equilateral_triangle(11);
    } catch (TooBigException e) {
      System.out.println("The triangle is too big");
    } catch (TooSmallException e) {
      System.out.println("The triangle is too Small");
    }
    // OR
    try {
      Equilateral_triangle et1 = new Equilateral_triangle(11);
    } catch (Exception foo) {
      System.out.println("The triangle is not good");
    }
  }
}
```

/* You may have a look at the Exception class to understand the various methods of the foo variable in the catch method */

For the next course, you must write a code that declares and uses a user declared Exception in your inheritance code.