### 2.1. Inheritance and references to the base class

#### Exercise

\*\*\*\* First of all, create the class diagram \*\*\*\*

Download the following code listings attached ( [Position.java](http://www.it.uc3m.es/java/gitt/units/oo-herencia/guides/6/cf_HerenciaReferenciasClaseBase/Position.java), [Figure.java](http://www.it.uc3m.es/java/gitt/units/oo-herencia/guides/6/cf_HerenciaReferenciasClaseBase/Figure.java), [Castle.java](http://www.it.uc3m.es/java/gitt/units/oo-herencia/guides/6/cf_HerenciaReferenciasClaseBase/Castle.java), [Queen.java](http://www.it.uc3m.es/java/gitt/units/oo-herencia/guides/6/cf_HerenciaReferenciasClaseBase/Queen.java) ) and analyze both the class hierarchy and the Position class that is going to be used.

We are going to deal with a set of figure elements. As Figure class is abstract, we can't create instances from it, therefore, the collection of elements will be made up of objects from the Castle and Queen classes, indistinctly.

Implement the necessary code to hold, in the same collection and indistinctly, 2 objects of Queen class and 4 objects of Castle class.

Once the previous code has been implemented, traverse the collection invoking public void whoAmI() method to test the correct operation of the base class references.

#### Further questions

Once this exercise has been finished, answer the following questions:

* Which methods can be really invoked on the collection elements?
* If Castle class has implemented void castle() method, could it be possible to invoke that method from a reference to the base class? Why?
* What should we have to do in order to be able to use the previous void castle() method from an object of Castle class that is pointed by a reference to Figure class?
* What should we do to know exactly to which class belongs every object pointed by a reference to the base class?

### Exercise Section2.2. Exercise to practice with Figures

In this exercise you are going to review the main concepts of Object Oriented Programming (OOP) in Java with an application to create a class hierarchy to define geometric figures.

IMPORTANT NOTE: Remember that it is essential that you test your code whenever you implement any method, even though you are not asked to do so explicitly.

\*\*\*\* First of all, create the class diagram \*\*\*\*

#### The Figure class.

All classes that are defined in this exercise must provide a basic set of methods, such as calculating the area of a geometric figure. For this exercise, every class that is created must extend the Figure class.

#### The geometric figure.

An abstract class is a class that can not be instantiated. This is shown by the use of the abstract keyword in its declaration. An abstract class has "at least" one abstract method. That is, a method which is only declared but not implemented. This abstract method should be implemented by any subclass of the class. There is no point in instantiating objects of an abstract class and only subclasses of it can instantiate objects.

Remember that an abstract class can have constructors, though objects from it can not be instantiated.

1. Write the code of the GeometricFigure abstract class, which must store the common information for all geometric figures (for example, a descriptive label) and has to provide all methods that can be shared among figures and are independent from the concrete shape of them.

This class must extend the Figure class. Every class that can be represented by a geometric figure will inherit from GeometricFigure class.

Every figure will have a descriptive label, so you should define an attribute of type String to hold the text of the label.

1. Write a constructor that receives as input parameter the descriptive label of the figure.
2. Write the get and set methods that allow to modify the text label's attribute. As these methods should not be modified by anybody, declare them as final to avoid any subclass to change the code of them (method overriding).
3. Implement the printDescription() method. Notice that is not an abstract method although it invokes other abstract methods. Also, to avoid subclasses to change the code of it (method overriding), declare it as final.

The method must print a text description of the figure on the console, including the label's text, the type of figure and its area, with the following format:

Tag: C-5

Figure Type: Square

Area: 25

Remember that this class must extend the Figure class, so it must provide the code for the complete set of methods that have been defined in such class. These methods should be implemented with the available information in the class, that is: getTag and printDescription methods. It is not necessary to include those methods which are impossible to be programmed in the class (because they are abstract methods), but subclasses are responsible to provide their code implementations.

#### The rectangle.

The next class to implement represents a rectangle and, obviously, its name will be Rectangle. This class inherits from GeometricFigure and extends the Figure class. A rectangle is defined by two dimensions, the base and the height and both are assumed to be integer values.

1. Write the class declaration and its correspondings attributes.
2. Write the constructor and the basic accesor get and set methods.
3. Write the following methods of the class:
4. public String getFigureType();
5. public double area();

public void drawTxt();

The drawTxt() method must print the figure on the console. For example, a rectangle of base 6 and height 3 can be printed as follows:

\*\*\*\*\*\*

\*\*\*\*\*\*

\*\*\*\*\*\*

1. Implement unit tests for this particular class.
2. Answer the following questions:
   * Show the difference between a class and an object.
   * Which steps are involved in the instantiation's process of an object?
   * How is an object instantiated in Java?

#### The square.

A square is a type of rectangle where both base and height are equals. In Java, we can easily create a Squareclass by deriving it from Rectangle class.

1. Write the Square class so that it inherits from the Rectangle class of the previous section. Square class just only needs a constructor which receives a single input parameter: the side value. The constructor will invoke the constructor of the Rectangle superclass with the same value for the parameters of base and height
2. Test the new class by adding its unit tests.
3. Notice that, thanks to inheritance, it is possible to invoke methods of the Rectangle superclass on an object of a derived type Square without the need of programming them again.
4. Answer the following questions:
   * What is inheritance?
   * How do you express in Java that one class inherits from another?
   * Which methods of the superclass are visible from the subclasses?
   * What is the meaning of method overriding?
   * Remember that, opposite to the rest of the methods of a class, subclasses don't automatically inherit constructors from the superclass, but they can be invoked by the use of super() keyword.

#### Reference to interfaces.

In this section, you must improve FiguresTesting class to provide a user interface in text mode that allows the user to choose the figure that he/she wants to print on the console, then asks for the correct parameters for each figure, creates an instance of the correct class and, finally, prints the figure description and the figure itself on the console.

Use the following menu:

1.- Create rectangle

2.- Create square

3.- Display figure

0.- Exit

If the user selects one of the 2 first options, the program must ask him/her for the correct data. If he/she selects option 3, the last figure will be printed on the console (including its description). The menu must be shown until the user select the option Exit.

To simplify the generation of the correct object from the data entered by the user, it would be useful to add a readFigureData() method to each class. This method would receive, as an input parameter, an object of type BufferedReader from which, it will read the data introduced by the user. Then, it would return a correct instantiated object of the class with the entered data. Declare it as a static method.

Answer the following questions:

* Which type are both instantiated objects (in options 1 and 2)?
* Which type is the variable that references them?
* Which methods from superclass are visible from the subclass?
* Can you use the same variable as a reference for different types of figures? Why?