# **Exo 1**

2- When we type sysout and press Ctrl + space in a main method, the instruction System.out.println is generated automatically.

3- When we type toStr and press Ctrl + space inside a class, some methods containing the expression ‘toStr’ opn and first of them is the toString method of the class Object

4- When we type main and press Ctrl + space inside a class, some methods containing the letters ‘m-a-i-n’ open and first of them, is the main method.

5- When we press Ctrl + space inside a class, several options appear including the constructor of the class. When we type set and press Ctrl + space inside a class, it is possible to select a type of set as a HashSet we want to create.

6- After selecting the class name and typing alt+shift+R, a window to rename the class opens. In the same way, after selecting the int field foo and typing alt+shift+R, a window to rename this field opens.

7- After declaring a String variable and clic on String with Ctrl key pressed, the file String.class opens which lists and explains each method of a String object.

# **Exo 2**

1- It works because each Java class has a default constructor which initialize the attributes of an object. Then, when we created the objet p, an instance of the Point class, the default constructor initialized w to 0 and y to 0. This is why, the print instruction displays “0 0”.

2- It can’t work because the attributes x and y are private. They are only visible by their class (Point). In order to fix it, we could delcare x and y as protected attributes. Then, they are accessible to all the classes of the same package in which they were declared as the TestPoint class for example.

3- We neet to set all fields visibility to private for security reasons. When a field is private , the caller cannot usually get inappropriate direct access to the field.

4- Acessor method takes no parameters and its return type matches the type of the variable they are accessing. This method returns the value of a private variable. It allows to give access to the value stored in that variable without having a direct access to it. This is why, we have to have an accessor for the Point class.

5- If we create a constructor with arguments (two or another number), we have to specify, in the main method, when we create an object of the class Point, the values of its attributes x and y. We have to add this in the code in order to fix the error.

6- If we want to call the parameters x and y instead of px and py, we have to precise which x or y it is.. For the field of the object created, we have : this.x or this.y and for the value we want to this field we have just x or y.

7- In order to keep track the number of Points that have been created so far, we can create a static int field named counter. Why static ? Because a static variable is linked to a class, not to an object. Then, we can increment this variable counter by 1 in the constructor. Then, each time, we call the constructor (i.e. we create a Point) the counter of the number of Points increases by 1.

8- The two constructors don’t take the same type and the same number of argument(s). When we call a constructor and specify two int arguments, we call the first constructor. When we call a constructor and specify a Point argument, we call the second one. We must be careful not to forget to increment the Points counter in this constructor too.

9- We can override the method toString in the Point class so that a call to System.out.println(point) will print the point coordinates as follow : (x, y).

@Override

**public** String toString() {

**return** "(" + x + ", " + y + ")";

}

# **Exo 3**

1- This code prints true fo the first print and false for the second one even the three points p1, p2 and p3 have the same coordinates. In fact, p1 equals to p2 because they have the same hashCode and p1 not equals to p3 because they have different hashCode. The equality between two objects depend on their hashCode not on the value of their attributes.

2- **public** **boolean** isSameAs(Point p) {

**return** ((**this**.x == p.x) && (**this**.y == p.y));

}

3- In the list, there is one element, the point p1 or p2 (we saw they are the same because their hasCode is the same) and the method indexOf returns the index of the first occurrence of the element specified i.e. 0. The point p3 is not in the list, this is why the p3.indexOf() instruction returns -1 (Cf. Javadoc)

# **Exo 4**

1- **public** **class** PolyLine {

**private** **int** length;

**private** Point polyLine[];

**public** PolyLine(**int** plength) {

**this**.length = plength;

**this**.polyLine = **new** Point[length];

}

}

2- **public** **void** add(Point p, **int** index) {

**this**.polyLine[index] = p;

}

If we add more points that the maximum capacity of the array, we have an error : “out of bounds for length x”. In order to fix that, we can decide to store the PolyLine in an ArrayList for example because it has no limited capacity.

3- After considering the remark, we chose to use an ArrayList for the PolyLine. Then, we can create two method, one to return the capacity of the PolyLine and another to return the number of Points in the PolyLine. We have the code below :

**public** **class** PolyLine {

**private** **int** length;

**private** ArrayList<Point> polyLine;

**public** PolyLine(**int** plength) {

**this**.length = plength;

**this**.polyLine = **new** ArrayList<Point>(length);

}

**public** **void** add(Point p) {

**this**.polyLine.add(p);

}

**public** **int** pointCapacity() {

**return** **this**.length;

}

**public** **int** nbPoints() {

**return** **this**.polyLine.size();

}

}

4- **public** **boolean** contains(Point p) {

**for** (Point point : polyLine) {

**if** (p == point) {

**return** **true**;

}

}

**return** **false**;

}

5- If null is given instead of an actual Point object i.e. if we do add(null), it works i.e. the Object is added to the PolyLine. In order to forbid this insertion, we have to modify the method Add() as follows :

**public** **void** add(Point p) {

Objects.*requireNonNull*(**null**);

**this**.polyLine.add(p);

}

6- ?

# **Exo 5**

1- First option : Without creating a new Point

**public** **void** translate(**int** dx, **int** dy) {

**this**.x = x+dx;

**this**.y = y+dy;

}

Second option : With the creation of a new Point

**public** Point translate(**int** dx, **int** dy) {

**return** **new** Point(x+dx, y+dy);

}

2- **public** **class** Circle {

**private** Point center;

**private** **int** radius;

**public** Circle(Point pcenter, **int** pradius) {

**this**.center = pcenter;

**this**.radius = pradius;

}

}

3- @Override

**public** String toString() {

**return** "The coordinates of the circle's center are : " + **this**.center + " and the radius measures "

+ **this**.radius + " cm.";

}

We remember the toString method has also been rewrited in Point in order to have the coordinates (x,y) written when calling a Point object.

4- **public** Circle translate(**int** dx, **int** dy) {

**int** x = **this**.center.getX();

**int** y = **this**.center.getY();

Point newCenter = **new** Point(x+dx, y+dy);

**return** **new** Circle(newCenter, radius);

}

In this method, we chose to translate the Circle by creating a new Circle.

5- The problem in this code is that we can’t see the Circle did a translation. In fact, the above method makes the translation by creating a new instance of Circle. Then, the circle c2 is exactly the same as before the translation. It keeps its center’s coordinates. In order to avoid this, we can create another method called translate2 that translates the circle’s center without creating another instance of Circle. In order to do this, we have to create first twe setters for changing Point’s coordinates. We have in the Point class :

**public** **void** setX(**int** newX) {

**this**.x = newX;

}

**public** **void** setY(**int** newY) {

**this**.y = newY;

}

And in the Circle class, we have this code :

**public** **void** translate2(**int** dx, **int** dy) {

**int** x = **this**.center.getX();

**int** y = **this**.center.getY();

**this**.center.setX(x + dx);

**this**.center.setY(y + dy);

}

6- We have the same problem as in the previous question. In fact, we call the method translate in the center Point object. Then, it creates a new instance of Point for the circle’s center. This is why, we can’t see the translation of the original center. To fix this problem, we have to call the method translate2 that performs a defensive copy of the center value. We have the code :

Circle c = **new** Circle(**new** Point(1, 2), 1);

c.translate2(1, 1);

c.getCenter();

System.***out***.println(c);

7- **public** **double** area() {

**return** Math.***PI*** \* Math.*pow*(radius, 2);

}

@Override

**public** String toString() {

**return** "The coordinates of the circle's center are : " + **this**.center + ", the radius measures "

+ **this**.radius + " cm and the circle's area is " + area() + " cm^2.";

}

8- **public** **boolean** contains(Point p) {

**return** (Math.*sqrt*(Math.*pow*((p.getX() - **this**.center.getX()), 2)

+ Math.*pow*((p.getY() - **this**.center.getY()), 2)) < **this**.radius);

}

9- **public** **boolean** contains(Point p, Circle... circles) {

**for** (Circle circle : circles) {

**if** (!(circle.contains(p))) {

**return** **false**;

}

}

**return** **true**;

}

?

# **Exo 6**

1- We should use inheritance because a ring is a subtype of a circle. Indeed, it has two radius and one center. It’s relevant to make Ring inherit from Circle.

2- **public** **class** Ring **extends** Circle{

**private** **int** innerRadius;

**public** Ring(Point pcenter, **int** pouterRadius, **int** pinnerRadius) **throws** IllegalStateException {

**super**(pcenter, pouterRadius);

**this**.innerRadius = pinnerRadius;

**if**(**this**.getRadius() < **this**.innerRadius) {

IllegalStateException e = **new** IllegalStateException();

**throw**(e);

}

}

}

With the method getRadius as follows :

**public** **int** getRadius() {

**return** **this**.radius;

}

3- @Override

**public** **boolean** equals(Object o) {

**return**(o == **this**);

}

?

4- If we do a System.out.println(ring) without additional code, we will have the output of the method toString written in the class Circle. To have another output, we have to use polymorphism and redefined the method toString in the class Ring. We have :

@Override

**public** String toString() {

**return** "The coordinates of the ring's center are : " + **this**.getCenter()

+ ", the outer radius measures " + **this**.getRadius() + " cm and the inner radius measures "

+ **this**.innerRadius + " cm.";

}

5- For this question an the last one, we consider that the ring contains a Point if it is contained in the largest circle. We have :

@Override

**public** **boolean** contains(Point p) {

**return** (Math.*sqrt*(Math.*pow*((p.getX() - **this**.getCenter().getX()), 2)

+ Math.*pow*((p.getY() - **this**.getCenter().getY()), 2)) < **this**.getRadius());

}

6- **public** **boolean** contains(Point p, Ring... rings) {

**for** (Ring ring : rings) {

**if**(!(ring.contains(p))) {

**return** **false**;

}

}

**return** **false**;

}