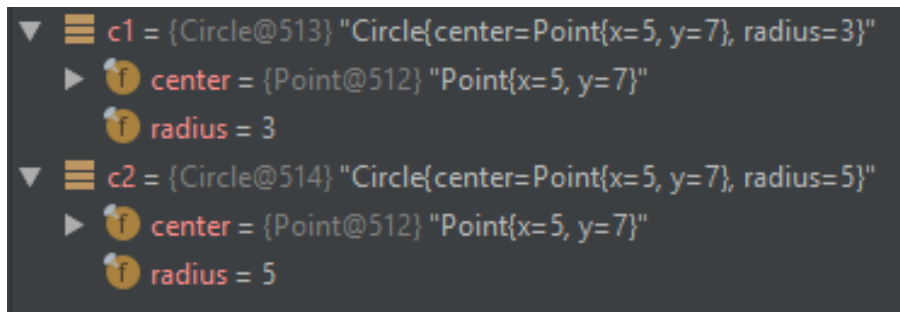








1 Theory

- (a) Thanks to genericity, we can get rid of casting, which means that the classes declared which use specific types don't have to handle an Object type and we don't need to convert every Object to the type we need every time.
- (b) A great example of this is the case where we want an ArrayList of a specific type, but without genericity, every time we want to access one of the elements of the array, we have to convert it from an Object type to the type we need.
- (c) Classes and methods can be generic.
- (d) A generic class is defined with a syntax such as this one "class MyClass<T,...>"
- (e) The code works, but usually the type which is in the "diamond" is a single upper case letter.
- (f) Since we already have Box<Cigar> as the type of the new variable, the context makes it possible to skip the specification of the type of the new Box.
- (g) Yes, we can say that the type of genericity must be an extension of another class or must implement another class. For example class NumberSet<T extends Number> is a class which only handles number types (integers, floats etc.)
- (h) Yes, we can define our class in such a way that it only handles types of objects which extend from another type. For example class NumberSet<T extends Number> only handles types like int or float.
- (i) No because although Integer still inherits Number, Box<Integer> doesn't inherit Box<Number>.
- (j) An enumerative type is a type which can represent a finite set of predefined elements.
- (k) If we have a class of cars which can only have one of three colors, we can create something like enum Color RED, AMBER, GREEN ; to make our life easier in choosing one of the three colors and also making sure we choose one of the possible colors and no other.
- (l) We can compare the elements with equals(other) and compareTo(other), we can get the position in the set with ordinal(), we can access the name with name() and toString(), and we can access the values of the set with values().
- (m) Every element should be in all upper cases.
- (n) We can add fields and methods (even constructors).

3 Debugging

- (a) Because the center of both circles are the same Point object, hence when one circle modify the position of the point with its method `moveTo` it also changes the center for the other circle.



```
▼  c1 = {Circle@513} "Circle{center=Point{x=5, y=7}, radius=3}"  
▶  center = {Point@512} "Point{x=5, y=7}"  
   radius = 3  
▼  c2 = {Circle@514} "Circle{center=Point{x=5, y=7}, radius=5}"  
▶  center = {Point@512} "Point{x=5, y=7}"  
   radius = 5
```

(b)