Guillem Escriba Molto - u188331

David Pérez Carrasco - u188332

Programación Orientada a Objetos

Informe Práctica 4

En esta cuarta práctica el objetivo principal es implementar el diseño realizado en el seminario 4, basado esencialmente en mover y dibujar regiones, pero a diferencia de las anteriores ahora empezaremos el programa de nuevo. Se deben implementar varias clases como *Entity*, *TriangularRegion, RectangularRegion, CircularRegion,...* A parte de estas también teníamos que modificar algunas ya creadas previamente como *Region, EllipsoidalRegion, PolygonalRegion, Point,...* Además también nos daban otras clases con las que trabajar, principalmente centradas en el apartado gráfico.

Una de las primeras clases que hicimos fue *Entity* para agilizar el proceso ya que eran muchas las clases que heredaban sus métodos abstractos y así la programación sería más rápida al poder sobreescribir los métodos ya declarados y reusar la API. Paralelamente a esto, hicimos las clases de *Point* y *Vector*.

En *Point* simplemente debimos añadir dos funciones, *move*, que recibía como argumentos dos reales y estos se sumaban a las coordinadas x e y de cada punto, y *difference*, que recibía un punto como argumento y retornaba un vector cuyos componentes eran la resta del componente del punto de entrada menos el del punto de la clase que está llamando al método. Para ello tuvimos que implementar la clase *vector*, que era prácticamente idéntica a la de *Point*, con sus dos coordenadas como doubles, sus getters y sus setters pero con el método *CrossProduct* que básicamente retornaba el resultado del producto escalar entre dicho vector y el vector de entrada.

Una vez implementados *Point* y *Vector* y las clases abstractas *Entity* y *Region*, la cual únicamente contenía el atributo, el get y el set de fillColor y el método abstracto getArea, nos dispusimos a implementar las novedades de *PolygonalRegion*. Básicamente tuvimos que añadir los atributos fillColor y LineColor al constructor para inicial izarlos y llamar al setColor de graphics con el fillColor y el lineColor en el método draw para dibujar las regiones. También tuvimos que implementar el método translate, que para cada uno de los puntos de dicha región llamaba a su método move con el respectivo desplazamiento. Finalmente implementamos *isPointInside*, que calculaba los vectores entre cada par de puntos consecutivos y entre el primer punto de éstos y el punto a comprobar y calculaba el producto escalar de ambos, de modo que si algún resultado tenía un signo diferente al resto retornábamos false, y en caso de que todos tuvieran el mismo signo retornábamos true. Además implementamos *isSelected(Point)*, que simplemente retornaba true si isPointInside del mismo punto era cierto y false en caso contrario.

En cuanto a *TriangularRegion* y *RectangularRegion*, simplemente debíamos crear estas clases de forma que heredasen todas las funciones de *PolygonalRegion* pero que contuvieran 3 y 2 puntos respectivamente como atributos, y que en su constructor llamasen a super con la linked list de dichos puntos como argumento, mientras que en grtArea simplemente teníamos que llamar al súper de getArea.

En *EllipsoidalRegion* y *CircularRegion* no encontramos muchas dificultades, ya que si bien la mayoría de sus métodos eran nuevos, también eran bastante intuitivos. El único que supuso una mínima dificultad fue el método de *isPointInside* ya que a pesar de que su implementación es muy sencilla gracias a la fórmula proporcionada, a causa de un error nuestro, la escribimos mal, lo que luego nos supuso bastante trabajo al encontrar lo que no funcionaba en el ejercicio opcional, pero aún así la clase ha sido bastante sencilla. En esta práctica, a diferencia de las anteriores hemos usado muchos más métodos abstractos y se han

tenido que hacer más *overrides* que en las anteriores prácticas. Un buen ejemplo de esto es entre *Entity* y las distintas clases de regiones, dónde la segundas sobreescriben prácticamente todo de la primera. En contraste a esto tenemos la relación de *EllipsoidalRegion* y *CircularRegion* en la que la segunda hereda todo de la primera sin modificar nada más que un atributo.

Una de las cosas más problemáticas fue la implementación del ejercicio opcional, si bien en sí no fue difícil su implementación en sí, al ser una librería que nunca habíamos utilizado costó un poco al principio pero una vez la conseguimos fue bastante sencillo. Uno de los problemas que tuvimos con esto que si bien fue trivial también fue molesto, es que al principio no nos detectaba la librería y desconocíamos el motivo, pero al final, el propio Visual Studio, solucionó el problema. Luego también nos encontramos con que no se seleccionaban las entidades y al principio pensamos que era culpa del *MouseEvent* pero más adelante nos dimos cuenta de que era a causa de que las funciones de *isPointSelected* no estaban bien definidas.

En esta práctica no hemos tenido ningún gran problema de implementación, más allá de errores de código o errores lógicos que nos han hecho estancarnos y no conseguir el translado o la selección de entidades, llegando a creer que debíamos implementar métodos adicionales. Más que problemas, quizás hemos tenido alguna duda sobre la el planteamiento del enunciado que contenía cierta ambigüedad, como si las clases getArea en TriangularRegion, RectangularRegion y CircularRegion debían ser implementadas de forma distinta a las de sus padres o si simplemente debíamos usar el método súper, o el caso de RectangularRegion, en el que tan solo habían dos puntos como argumentos en el constructor y no entendíamos si realmente teníamos que crear un polígono con dos puntos o a partir de esos dos puntos debíamos tratar de crear un rectángulo de alguna forma.

En definitiva, esta práctica, a pesar de tener sus dificultades y complicaciones, nos ha sido más fácil de implementar que la anterior, puesto que invertimos mucho tiempo en el *Ocean*, y en esta, el opcional, era más sencillo, o al menos a nuestro parecer. Creemos que el resultado es bastante adecuado, ya que la precisión al seleccionar los polígonos con el ratón es muy alta, y ofrece muchas posibilidades a la hora de trasladar de un lado a otro regiones. El único aspecto negativo a resaltar, tal vez sería la falta de tiempo, ya que si bien era más corta que la anterior, no lo era tanto como para tener solo una semana y aunque hayamos logrado implementar todo, de tener más tiempo hubiésemos intentado perfeccionar más aún el código pues a nuestro parecer, a pesar de cumplir con creces los objetivos, hay secciones que se podrían optimizar.