DEPARTAMENTO LENGUAJES Y CIENCIAS DE LA COMPUTACIÓN	**PROGRAMACIÓN ORIENTADA A OBJETOS** (Prueba del 12 de junio de 2017)	
APELLIDOS, Nombre	TITULACIÓN Y GRUPO	
	MÁQUINA	

NOTAS PARA LA REALIZACIÓN DEL EJERCICIO:

- El ejercicio se almacenará en el directorio **C:\POO\JUN17**. En caso de que no exista deberá crearse, y si ya existiese, deberá borrarse todo su contenido antes de comenzar.
- Al inicio del contenido de cada fichero deberá indicarse **el nombre del alumno, titulación, grupo y código del equipo** que está utilizando**.**
- La evaluación tendrá en cuenta la claridad de los algoritmos, del código y la correcta elección de las estructuras de datos, así como los criterios de diseño que favorezcan la reutilización.
- Los diferentes apartados tienen una determinada puntuación. Si un apartado no se sabe hacer, el alumno *no debe pararse en él indefinidamente*. Puede abordar otros.
- **Está permitido:**
  - Consultar el API y la Guía rápida de la API.
- **No está permitido**:
  - Utilizar otra documentación electrónica o impresa.
  - Intercambiar documentación con otros compañeros.
  - Utilizar soportes de almacenamiento.
- Una vez terminado el ejercicio subir un fichero comprimido sólo con los ficheros **.java** que hayáis realizado a la tarea creada en el campus virtual para ello.

Proyecto prRegata

En este ejercicio se va a desarrollar un conjunto de clases que nos permitirán controlar los barcos que participan en una regata. Para ello, se creará un proyecto `prRegata` con las clases siguientes en el paquete `regata`, donde la clase `Posicion` se proporciona en el campus virtual y el resto de clases deben crearse (también se proporcionan en el campus virtual varias clases de prueba (`Main2`, `Main3` y `MainGUI`) y se indica al final del enunciado después de qué apartado debe ejecutarse cada una de ellas):

1) Para la realización del ejercicio, se proporciona la clase `Posicion` que determina una posición conocida su latitud y longitud (en grados decimales). Esta clase dispone de un constructor donde se proporcionan los valores para la latitud y longitud (`double`). La latitud será un valor entre -90 (latitud sur) y 90 (latitud norte) y la longitud entre 0 y 360 grados a partir del meridiano de Greenwich en sentido este. Tanto la latitud como la longitud se normalizan internamente si están fuera de rango. La clase dispone de los siguientes métodos:
   a. `double getLatitud()` y `double getLongitud()` que devuelve la latitud y la longitud.
   b. `double distancia(Posicion p)` que calcula la distancia (en Km) desde el receptor a la posición `p`.
   c. `Posicion posicionTrasRecorrer(int minutos, int rumbo, int velocidad)` que calcula la posición final si partimos de la posición del receptor y viajamos los minutos dados en los argumentos con el rumbo (valor entre 0 y 359 siendo 0 el rumbo norte, 90 el rumbo este, etc.) y velocidad (dada en km/h) también dados en los argumentos.
   d. `String toString()` devuelve una cadena que representa a la posición. Por ejemplo, la posición de latitud 35 y longitud 156 se representa por `"l = 35 L = 156".`

2) (**0.25 ptos.**) Crear la excepción `RegataException` *no comprobada* para tratar las situaciones excepcionales.

3) (***1.25 ptos.***) La clase `Barco` debe mantener información sobre un barco. En concreto, tendrá una variable de instancia de tipo `String` para el `nombre`, otra de tipo `Posicion` para la `posicion` y dos de tipo `int` para el `rumbo` y la `velocidad.` El rumbo es un ángulo (0 para rumbo norte, 90 para rumbo este, etc. Así hasta 359.) y la velocidad se mide en km/hora. Todas las variables son *protected*.
   a. Definir un constructor que cree un barco conocidas las cuatro variables descritas anteriormente. Comprobar que el rumbo se encuentra entre 0 y 359. Si no es así, lanzar una excepción de tipo `RegataException`.
   b. Definir métodos de acceso a cada variable (`String getNombre()`, `Posicion getPosicion()`, `int getVelocidad()` y `int getRumbo()`).
   c. Dos barcos son iguales si lo son sus nombres, ignorando mayúsculas y minúsculas.
   d. Los objetos de la clase Barco se ordenan de forma natural por nombre, ignorando mayúsculas y minúsculas.
   e. El método `void avanza(int mnt)` cambia la posición del barco a la posición donde estaría una vez que transcurran `mnt` minutos (según su posición, rumbo y velocidad).
   f. El método `String toString()` debe mostrar el nombre, la latitud y longitud de la posición, el rumbo y la velocidad de un barco con el formato indicado a continuación. Por ejemplo, para un barco de nombre *gamonal*, situado en la posición *(-30, 290)*, con rumbo *0* y velocidad *24* este método mostraría la información de la siguiente manera:

```
gamonal: l= -30 L= 290 R= 0 V= 24
```

4) (***0,5 ptos***.) Crear una aplicación (clase distinguida `PruebaBarco`) que cree cuatro barcos y los introduzca en un array. Luego los ordene con el método `Arrays.sort(array)` y por último imprima el menor y el mayor.

5) ***(0,5 ptos.)*** Crear la clase `Velero` que se comporta como la clase `Barco`, pero cuando su rumbo es menor o igual a 45 o mayor o igual a 315, avanza a una velocidad 3km/hora inferior a su velocidad. Igualmente, si su rumbo está comprendido entre 145 y 225 avanza a una velocidad 3km/hora superior a su velocidad. En otro caso se comporta como un barco normal (está simulando que hay viento del norte). Redefine para ello el método `avanza()`. Define también un constructor para la clase `Velero` que reciba como argumentos su nombre, posición, rumbo y velocidad.

6) (***0,5 ptos.***) Definir un orden alternativo (clase `SatBarco`) que permita ordenar los barcos por la distancia que les separa de la posición de latitud 0 y longitud 0. En caso de que dos barcos estén a la misma distancia se ordenarán por el orden natural.

7) (***4 ptos.***) La clase `Regata` mantiene información de todos los barcos participantes en una regata en una variable de instancia llamada `participantes`. Esta variable será un conjunto ordenado por el orden natural.
   a. El constructor crea las estructuras adecuadas.
   b. El método `void agrega(Barco b)` agrega el barco `b` a los participantes si no estaba ya incluido.
   c. El método `void avanza(int mnt)` hace que todos los barcos de `participantes` se sitúen en la posición que quedarían transcurridos `mnt` minutos.
   d. El método `Set<Barco> getParticipantes()` que devuelve la colección con los barcos participantes en la regata.
   e. El método `Set<Barco> ordenadosPorDistancia()` devuelve una colección de participantes ordenados por el orden alternativo definido anteriormente.

f. El método `boolean velocidadSuperiorA(int velocidad)` devuelve cierto si hay al menos un barco en `participantes` cuya velocidad sea superior o igual a la dada.

g. El método `List<Barco> dentroDelCirculo(Posicion p, int km)` devuelve una lista con los barcos de `participantes` que se encuentra a una distancia menor que `km` de la posición `p`.

h. Se llama cota de un barco al entero que resulta de dividir la velocidad del mismo por 10. El método `Map<Integer, Set<Barco>> barcosPorVelocidad()` devuelve una correspondencia que para cada cota asocie los barcos de `participantes` con esa cota. Por ejemplo, si un barco va a 34km/h se asociará a la cota 3 (34/10 = 3).

8) (***2 ptos.***) Añadir a la clase `Regata` los siguientes métodos que facilitan la entrada/salida.

a. El método `Barco creaBarcoString(String)` que crea un barco y lo devuelve con los datos que aparecen en la cadena que se pasa como argumento. Esa cadena tendrá un formato como el del ejemplo:

```
Gamonal, -30, 290, 0,24
```

Como se ve, el delimitador es "`[ ,]+`".

Si se produce alguna excepción no comprobada, transformarla en una `RegataException`.

b. El método `void leeFichero(String)` lee los barcos de una regata de un fichero cuyo nombre se pasa como argumento y donde cada línea tiene el formato anterior. Se apoyará en el método `void lee(Scanner)` que ha de implementarse también.

c. El método `void escribeFichero(String)` escribe los participantes en un fichero cuyo nombre se pasa como argumento con e mismo formato que devuelve el método `toString()` de `Barco`. Se apoyará en el método `void escribe(PrintWriter)` que ha de implementarse también.

9) ***(1 pto.)*** Con objeto de construir una interfaz gráfica de usuario para usar con una regata, en el campus virtual se proporciona la interfaz `VistaRegata` y la clase `PanelRegata` que la implementa. Construir un controlador que se comporte de la siguiente manera:

a. Cuando se pulse el botón `Lee`, creará una regata (el modelo se crea aquí) y le añadirá los barcos que se encuentren en el fichero de entrada. En concreto:

- Creará la regata, leerá los barcos desde el fichero de entrada y se los añade, limpiará el área de texto y mostrará los barcos, uno por línea. Indicará en un mensaje que el proceso ha acabado satisfactoriamente.

b. Cuando se pulse `Avanza`, avanzará la regata 10 minutos.

- Avanzará la regata 10 minutos, limpiará el área de texto y mostrará los barcos uno por línea. Informará de que el proceso ha acabado satisfactoriamente.

c. Cuando se pulse `Guarda`, guardará la regata en el fichero de salida. En concreto:

- Guardará la información de la regata en el fichero de salida e informará de que el proceso ha acabado satisfactoriamente.

Si algún proceso anterior falla, se debe controlar la excepción y mostrar un mensaje informando de lo que ha ocurrido en el área de mensajes.

En el campus virtual tenéis varios programas de prueba:

- Main2 (ejecutar al completar el apartado 7). Produce como resultado:

```
[alisa: l= 30,00 L= 240,00 R= 80 V= 20, gamonal: l= 0,00 L= 100,00 R= 0 V= 24, kamira: l= 80,00
L= 182,00 R= 230 V= 33, veraVela: l= -30,00 L= 290,00 R= 20 V= 14]
[alisa: l= 30,36 L= 241,77 R= 80 V= 20, gamonal: l= 2,49 L= 100,00 R= 0 V= 24, kamira: l= 77,80
L= 181,55 R= 230 V= 33, veraVela: l= -28,93 L= 290,34 R= 20 V= 14]
true
[alisa: l= 30,36 L= 241,77 R= 80 V= 20, gamonal: l= 2,49 L= 100,00 R= 0 V= 24, kamira: l= 77,80
L= 181,55 R= 230 V= 33]
[gamonal: l= 2,49 L= 100,00 R= 0 V= 24, alisa: l= 30,36 L= 241,77 R= 80 V= 20, kamira: l= 77,80
L= 181,55 R= 230 V= 33, veraVela: l= -28,93 L= 290,34 R= 20 V= 14]
{1=[veraVela: l= -28,93 L= 290,34 R= 20 V= 14], 2=[alisa: l= 30,36 L= 241,77 R= 80 V= 20,
gamonal: l= 2,49 L= 100,00 R= 0 V= 24], 3=[kamira: l= 77,80 L= 181,55 R= 230 V= 33]}
```

- Main3 (ejecutar al completar el apartado 8). Produce como resultado:

```
alisa: l= 30,00 L= 240,00 R= 80 V= 20
gamonal: l= -30,00 L= 290,00 R= 0 V= 14
kamira: l= 80,00 L= 182,00 R= 230 V= 33
vera: l= 0,00 L= 260,00 R= 20 V= 25
alisa: l= 30,36 L= 241,77 R= 80 V= 20
gamonal: l= -28,55 L= 290,00 R= 0 V= 14
kamira: l= 77,80 L= 181,55 R= 230 V= 33
vera: l= 2,43 L= 260,89 R= 20 V= 25
```

y se debe haber creado el fichero `salida.txt` con el contenido de las 4 últimas líneas.

- MainGUI (ejecutar al completar el ejercicio 8). Después de pulsar Lee y Avanza, produce como resultado