Práctica 10

Objetivos: Practicar recorridos y búsquedas en `ArrayList` y el tratamiento de excepciones

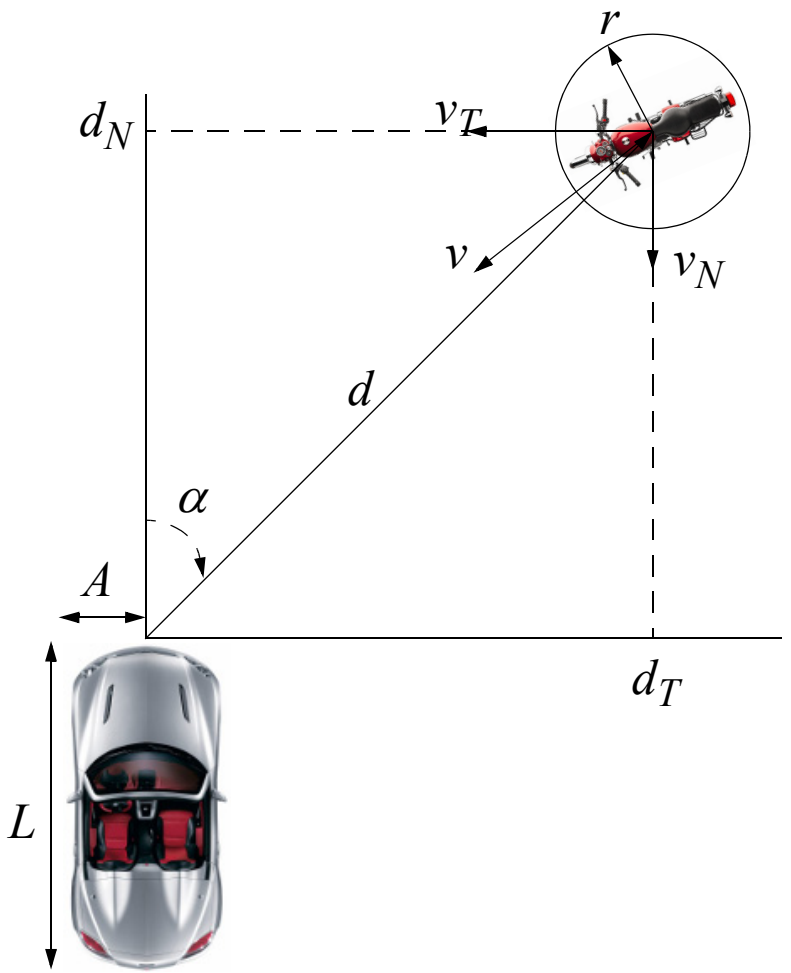
Descripción: En pocos años está previsto que muchos automóviles tengan capacidades de conducción automática

Se desea implementar parte del software de un sistema para evitar colisiones

El automóvil dispone de un radar que ve los obstáculos que tiene delante y mediante un software de análisis de datos determina la velocidad y posición de cada obstáculo, así como su radio

Clase Obstaculo

Obstaculo
-int id -double vT -double vN -double d -double alfa -double r
+Obstaculo(int id, double r) +set(double vT, double vN, double d, double alfa) +double tAlcance(double vC) +double tRebase(double vC, double L) +double margenAlcance(double vC) +double margenRebase(double vC, double L) +int getId() +double getRadio()



Clase `Obstaculo` (cont.)

La clase está ya implementada y dispone de los siguientes atributos:

- `id`: entero que identifica al obstáculo
- `vT`: velocidad tangencial, en m/s
- `vN`: velocidad normal, en m/s
- `d`: distancia al obstáculo, en m
- `alfa`: ángulo al obstáculo, en radianes
- `r`: radio del obstáculo, en m

Los métodos de esta clase son:

- *Constructor*: Recibe como parámetros los valores iniciales de los atributos `id` y `r` (m)
- `set`(): Recibe como parámetros los valores actualizados de los atributos `vT` (m/s), `vN` (m/s), `d` (m) y `alfa` (grados)

Clase `Obstaculo` (cont.)

- `tAlcance`(): Calcula el tiempo hasta alcanzar el obstáculo en la dirección del coche, dada la velocidad del coche `vC` en m/s
- `tRebase`(): Calcula el tiempo hasta rebasar el obstáculo en la dirección del coche, dada la velocidad del coche `vC` en m/s y la longitud del coche `b` en m
- `margenAlcance`(): Calcula el margen de distancia tangencial (m) entre el coche y el obstáculo cuando transcurra el tiempo de alcance
- `margenRebase`(): Calcula el margen de distancia tangencial (m) entre el coche y el obstáculo cuando transcurra el tiempo de rebase
- `getId`(): Observador del atributo `id`
- `getRadio`(): Observador del atributo `r` (m)

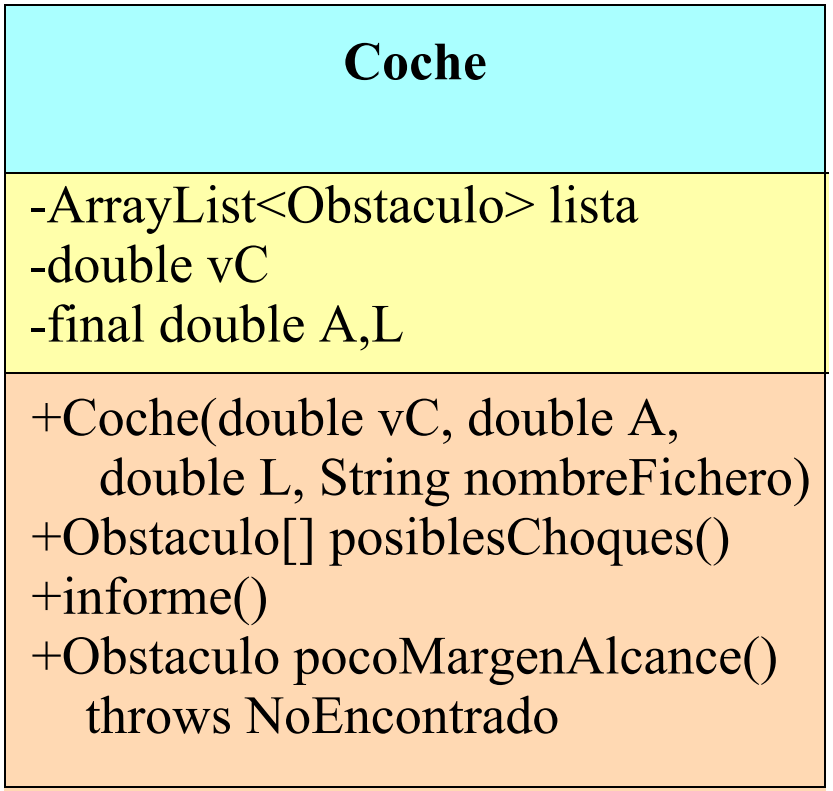
Clase Coche

Lo que se pide es implementar en Java la clase `Coche` pensada para *simular y probar* el funcionamiento del sistema de detección de colisiones

Su diagrama de clases se muestra en la figura

La clase dispone de los siguientes atributos:

- `lista`: una lista de obstáculos
- `vC`: velocidad del coche en m/s
- `A`, `L`: dimensiones del coche (semianchura y longitud) en m



Clase Coche (cont.)

Los métodos de la clase hacen lo siguiente:

- *constructor*: recibe como parámetros la velocidad del coche vC en m/s, las dimensiones A y L del coche en m y el nombre de un fichero de texto del que se leen datos para rellenar la lista de obstáculos
  - este método se da ya hecho
- posiblesChoques(): Retorna un array conteniendo todos los obstáculos para los que se detecta un posible choque. Para este método se usará el siguiente pseudocódigo:

```
método posiblesChoques() retorna Obstaculo[]
   // lista para meter los obstáculos que pueden chocar
   ArrayList<Obstaculo> posiblesChoques = nueva lista vacía
   // Recorrido para obtener los obstáculos que pueden chocar
   para cada obs de lista hacer
      booleano choca=false
      // consideramos que puede haber choque si los tiempos
      // de alcance y rebase están entre 0 y 30s
```

Clase Coche (cont.)

```
si tiempo de alcance de obs entre [0,30] y
   tiempo de rebase de obs entre [0,30]
entonces
   //El choque se produce si se da alguna de estas
   // tres circunstancias:
   si valor absoluto de margen de alcance de obs <= r+A
   entonces
      choca=true
   si no, si valor absoluto del margen de rebase
            de obs <= r+A
   entonces
      choca=true
   si no, si (margen de alcance de obs) *
               (margen de rebase de obs) <0
   entonces
       // el margen de alcance y el de rebase
       //tienen distinto signo
       choca=true
   fin si
fin si
```

Clase **Coche** (cont.)

```
        // si choca lo metemos en la lista
        si choca entonces
            añade obs a la lista posiblesChoques
        fin si
    fin para
    // Crear el array
    Obstaculo[] posibles=
        nuevo array de tamaño igual al de posiblesChoques
    // Metemos los datos de posiblesChoques en el array
    para cada i desde 0 hasta tamaño de posibles-1 hacer
        posibles[i]=casilla i de posiblesChoques
    fin para
    retorna posibles
fin método
```

Clase Coche (cont.)

- `informe`(): Pone en pantalla un informe de todos los obstáculos. Para ello pone en pantalla una línea de encabezamiento y luego recorre la lista de obstáculos mostrando los datos de cada uno, uno por línea. Por ejemplo, obtendríamos algo similar a esto:

```
Id      tiempoAlcance  tiempoRebase  margenAlcance  margenRebase
13456         2.74s         2.95s         15.1m         16.1m
13457         4.33s         4.79s         65.7m         64.7m
13458         1.57s         2.04s          0.2m          0.2m
```

- `pocoMargenAlcance`(): Busca en la `lista` (utilizando uno de los *patrones de búsqueda* en tablas visto en clase) el primer `Obstaculo` cuyo margen de alcance en valor absoluto es menor o igual que `r+A` y lo retorna. Si no lo encuentra lanza `NoEncontrado`

La excepción `NoEncontrado` debe crearse en una clase aparte

Programa principal

Finalmente, se pide hacer un programa principal de prueba de los métodos de la clase `Coche`, en una clase aparte, que haga lo siguiente:

a) Crea un objeto de la clase `Coche` con velocidad 18.5 m/s, dimensiones `A`=1.1 m, `L`=4.5 m y nombre de fichero `"obstaculos.txt"`

b) Muestra en pantalla el identificador del obstáculo obtenido con el método `pocoMargenAlcance`()
   - Si se lanza `NoEncontrado` se muestra un mensaje de error, se salta el paso c), pero se sigue por el paso d)

c) Muestra en pantalla los identificadores de todos los obstáculos obtenidos con `posiblesChoques`()

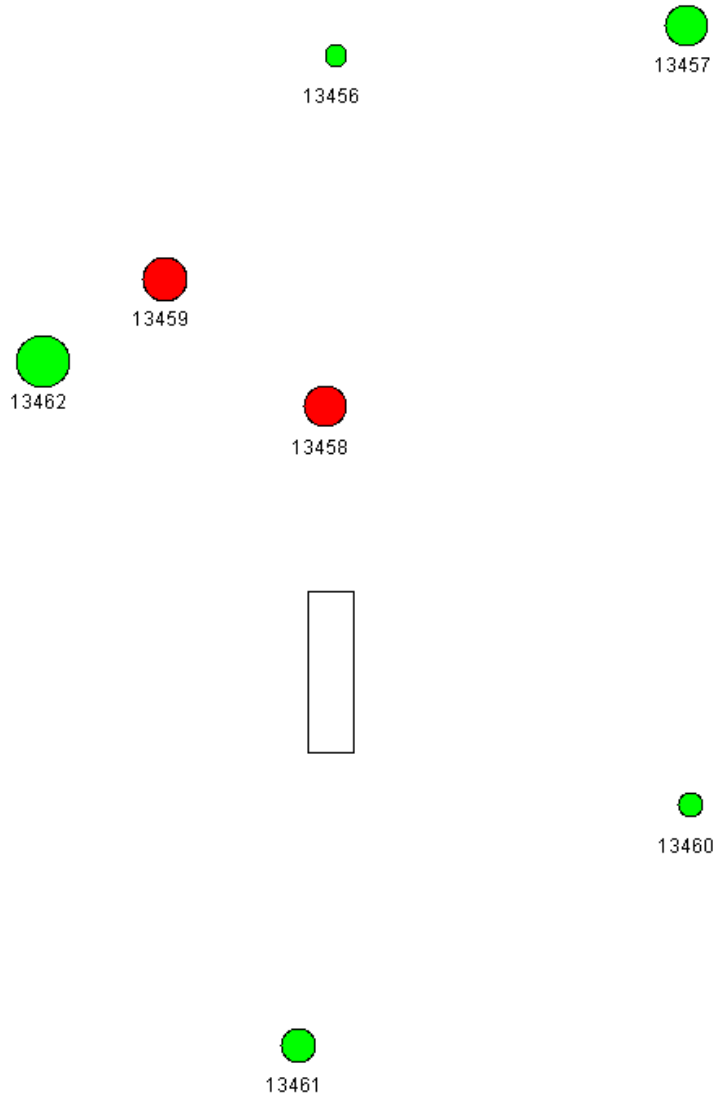
d) Invoca el método `informe`() para obtener un informe de todos los obstáculos existentes

Parte avanzada

Escribir en la clase `Coche` un nuevo método que haga un dibujo del coche (mediante un rectángulo) y de los obstáculos (mediante círculos) y sus IDs

- poner en verde los obstáculos que no tienen posibilidad de chocar y en rojo los posibles choques obtenidos mediante el método `posiblesChoques`()
- *Nota*: se pueden pintar todos los obstáculos de verde y luego encima los que pueden chocar, en rojo

Añadir al `main` una llamada al nuevo método de dibujo

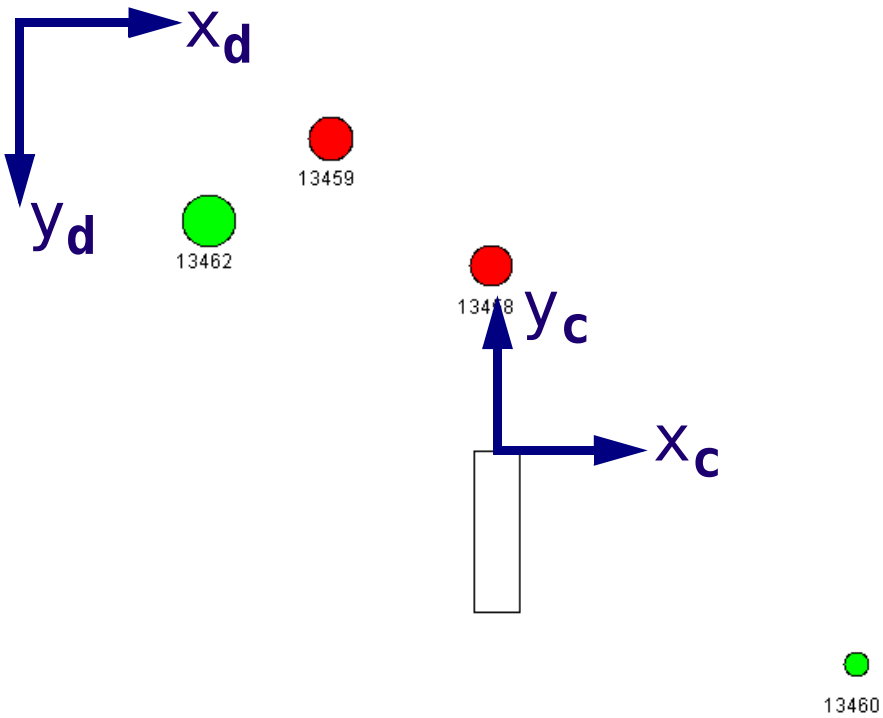


Parte avanzada: sistema de coordenadas

Se sugiere usar un sistema de coordenadas centrado en el frontal del coche: x_c, y_c

El dibujo usa un sistema de coordenadas centrado en la esquina superior izquierda: x_d, y_d

- Será necesario trasladar el origen e invertir el eje y



Se pueden asimilar píxeles a metros y usar un tamaño de ventana de 600*600

- frontal del coche centrado en 300*300 píxeles
- *nota*: en el dibujo se ha exagerado el tamaño del coche

Entregar

El proyecto Bluej en un archivo comprimido

Informe:

- Parte básica:
  - el código de la clase `NoEncontrado`
  - El código de la clase `Coche`
  - El código del programa principal
  - Una captura de pantalla de la ejecución del programa principal
- Parte avanzada
  - El código del nuevo método
  - Una captura de pantalla del dibujo obtenido