DrivingFrenzy

This is a vehicles speed competition. Several vehicles run individually in a track to do the best time. From the Main menu in the ControlCenter you should launch one of the types of races (currently, only simpleRandomRace is implemented).

The main parts in this project are:

- ControlCenter, located in the main package. It is the one that has a main method and allows us to run races.
- Vehicle, located in the vehicles package. It is an interface showing the methods that the ControlCenter can use to simulate how the vehicles drive.
    - In the same package there is a Scooter class, implementing this interface, as an example.
- Track, located in the race package. It is just a placeholder for Section objects - each Track will have one or more sections (think of sections as the different turns and straight roads that a real track could have)
    - In the same package there is a Section interface defining which operations a section should have. There is also a StandardIndoorSection which represents just a simple road.
    - **The max speed in each section is not defined by the vehicle, but by the section. I.e, the max speed in a section is the same for a Ferrari and Go-Go-Action Bronco.**

Note that several of the interfaces define utility methods to return a description; this is used from the ControlCenter to print information in command line.

The goals of this example are to:

- show the use of interfaces - note that to actually do the competition we only need to access to "Vehicle" objects, we do not need to declare them as Scooter or any other vehicle type.
- show how we can create different objects of the same Class with different characteristics.
- show how we can use objects between functions.

The Control Center

The control center simulates careers. After the track and Vehicles are defined, it will:

1. Choose the next vehicle.
2. Ask the vehicle to adapt its speed by giving the vehicle information about the next section. **The vehicles must only use the theoretical max speed of the section to adapt**.
3. Check what is the new speed of the vehicle. That speed will be used for that specific section.
4. Go back to point 2 until there are not any sections left.
5. Go back to point 1 until there are not any vehicles left.
6. Show the result (total time) of each vehicle.

Also, the Control Center will print information about the drivers and the track, and how each vehicle is doing.

The assignment follows; **the code must be properly commented**.

DrivingFrenzy

Setup and Evaluation

The assignment is self-driven and self-paced.

- Each student will be assigned to a random group, each group will have 3 students.
- Only one laptop is allowed per group. Students will take turns to be completing the assignment. This is an adaptation of Pair programming.
- The steps of the assignment are sequential.
- At certain points the students will be questioned to evaluate the job done so far.

These checkpoints are quoted as this line.

- It is not intended that the students reach the end of the assignment; the goal is to reach as far as possible.

DrivingFrenzy

Initial steps

- Run the application several times as it is and try to follow the code, to understand how it works.
- Create a UML Class diagram of the current project. Be specific about the visibility of methods and variables.

The diagrams will be reviewed and evaluated. All of the members of the group must explain it the portion they are asked for.

- Create a method called defaultRace in the ControlCenter. This is a simpler version of the simpleRandomRace method, but in this case everything is predefined (not randomized):
    - We have a track with 5 sections of 1000, 2000, 3000, 2000 and 1000 meters of length; you can choose the max speed.
    - We have just 3 vehicles, they will be Scooters. Give the vehicles proper descriptions and drivers.
- Run the race.

The code will be reviewed. Questions about the code will be asked randomly to the students.

- Sort the results so we print the different vehicles/drivers in order - by position. To do this, the method start can be modified just in the last part, the one with the comment MODIFY THIS to show the results sorted by total time.
- Implement a new type of vehicle. This vehicle is a Kart with 2 gears. Before coding it, add it tou your UML diagram.
    - Each gear will have a minimum and maximum speed.
    - When calling to adaptSpeed, the Kart can do only one of the following:
        - Increase or decrease speed within the current gear.
        - Shift a gear up (new speed will be the minimum speed of the second gear).
        - Shift a gear down (new speed can be any within the range of the first gear).
- Create a new race (i.e, a new method in the Control Center) and test the new Kart.

The code will be reviewed. Questions about the code will be asked randomly to the students.

DrivingFrenzy

Next steps

- Create a race in which both Scooters and Karts compete together. Remember, **the start method cannot be modified at this point**.
    - Choose 3 real one-gear Scooters and 3 real 2-gear Karts to get actual statistics on speed - don't spend more than 10 minutes searching, otherwise, just make up the numbers.
- Create a new type of Section called StandardOutsideSection. This one has its theoretical max speed, but also the actual one. The actual one is just a percentage of the theoretical one.
    - For instance, think of a rainy day. Actual max speed could be 0.7 * maxSpeed. Or, think of a very sunny day; the actual max speed could be 1.1 * maxSpeed.
    - This is not randomized. When creating the Object, the percentage must be provided and also a proper description.

The code will be reviewed. Questions about the code will be asked randomly to the students.

- **From this point on, you can modify the start method as needed.**
- Modify the start method: if a vehicle goes over the actual max speed of a section, it goes out of the track and gets disqualified.
- When listing the positions, you must also show the disqualified vehicles and in which section they were disqualified.
- Run a new Scooters + Karts race, but with both indoor and outdoor sections.

The code will be reviewed. Questions about the code will be asked randomly to the students.

DrivingFrenzy

Final steps

- Implement some randomization to the speed of the vehicles to simulate that the drivers sometimes make mistakes. When adapting the speed, all of the vehicles add a random factor to the speed by multiplying it by a random number between 0.8 and 1.0.
    - Don´t implement this directly in all of the vehicles. Find a way to implement it just once (inheritance). Also, make the code less redundant - likely, Scooter and Kart have attributes and methods in common.
- Automate the creation of a long track - at least, 50 kilometers with 5 sections. It must be random, and combine different types of sections.

The code will be reviewed. Questions about the code will be asked randomly to the students.

- Modify the sections so they provide extra information for the drivers. A section will offer a method getVariationLevel which will be -1 if somehow it is impaired and vehicles should go slower, 0 if standard, or 1 if vehicles can go faster. Do not provide the exact modifier that is being applied...
    - Do it similar to the last vehicles change.
- Implement a new type of vehicles with 6 gears, normal cars. Also, these cars are allowed to use the information of getVariationLevel.
    - Implement the adaptSpeed method similar to the one in the Karts. However, the vehicle has a new attribute driverStyle which can be standard, aggressive or conservative.
        - An aggressive driver will try to play with the odds to go faster than the theoretical max speed. For instance, if the variation level is positive, it will be aggressive, if it is 0 it will still be aggressive to compensate for the its own randomization speed (see point 1).
        - A conservative driver will always try to prevent getting out of the track.
        - A standard driver will have an average behavior.
    - Test it in a race.

The code will be reviewed. Questions about the code will be asked randomly to the students.

- Modify the full code so the vehicles compete simultaneously, and add commentator mentions about when a car overtakes another one, etc. Make it real!

DrivingFrenzy

Esta es una competencia de velocidad de vehículos. Varios vehículos corren individualmente en una pista para hacer el mejor tiempo. Desde el menú principal del Centro de control debes iniciar uno de los tipos de carreras (actualmente, solo está implementado simpleRandomRace).

Las partes principales de este proyecto son:

- ControlCenter, ubicado en el `main` paquete. Es el que tiene un `main` método y nos permite hacer carreras.
- `Vehicle`, ubicado en el `vehicles` paquete. Es una interfaz que muestra los métodos que ControlCenter puede utilizar para simular la conducción de los vehículos.
    - En el mismo paquete hay una `Scooter` clase que implementa esta interfaz, a modo de ejemplo.
- `Track`, ubicado en el `race` paquete. Es solo un marcador de posición para `Section` objetos: cada pista tendrá una o más secciones (piense en las secciones como las diferentes curvas y caminos rectos que podría tener una pista real).
    - En el mismo paquete hay una `Section` interfaz que define qué operaciones debe tener una sección. También hay un `StandardIndoorSection` que representa simplemente un camino simple.
    - **La velocidad máxima en cada tramo no la define el vehículo, sino el tramo. Es decir, la velocidad máxima en una sección es la misma para un Ferrari y un Go-Go-Action Bronco.**

Tenga en cuenta que varias de las interfaces definen métodos de utilidad para devolver una descripción; esto se utiliza para `ControlCenter` imprimir información en la línea de comando.

Los objetivos de este ejemplo son:

- mostrar el uso de interfaces: tenga en cuenta que para participar en la competencia solo necesitamos acceder a los objetos "Vehículo", no necesitamos declararlos como Scooter ni ningún otro tipo de vehículo.
- Muestre cómo podemos crear diferentes objetos de la misma Clase con diferentes características.
- Muestre cómo podemos usar objetos entre funciones.

El centro de control

El centro de control simula carreras. Una vez definidos la pista y los vehículos,:

1. Elige el siguiente vehículo.
2. Solicite al vehículo que adapte su velocidad dándole información sobre el siguiente tramo. **Los vehículos sólo deberán utilizar la velocidad máxima teórica del tramo a adaptarse** .
3. Compruebe cuál es la nueva velocidad del vehículo. Esa velocidad se utilizará para esa sección específica.
4. Vuelve al punto 2 hasta que no queden tramos.
5. Vuelve al punto 1 hasta que no queden vehículos.
6. Muestra el resultado (tiempo total) de cada vehículo.

Además, el Centro de Control imprimirá información sobre los pilotos y la pista, y cómo va cada vehículo.

La tarea sigue; **el código debe estar debidamente comentado**.

DrivingFrenzy

Configuración y evaluación

La tarea es autoimpulsada y a su propio ritmo.

- Cada estudiante será asignado a un grupo aleatorio, cada grupo tendrá 3 estudiantes.
- Sólo se permite una computadora portátil por grupo. Los estudiantes se turnarán para completar la tarea. Esta es una adaptación de la programación en pares .
- Los pasos de la tarea son secuenciales.
- En determinados momentos se interrogará a los estudiantes para evaluar el trabajo realizado hasta el momento.

Estos puntos de control se citan en esta línea.

- No se pretende que los estudiantes lleguen al final del trabajo; el objetivo es llegar lo más lejos posible.

DrivingFrenzy

Pasos iniciales

- Ejecute la aplicación varias veces tal como está e intente seguir el código para comprender cómo funciona.
- Cree un diagrama de clases UML del proyecto actual. Sea específico sobre la visibilidad de métodos y variables.

Los diagramas serán revisados y evaluados. Todos los miembros del grupo deben explicar la ración que se les pide.

Cree un método llamado `defaultRace` en `ControlCenter`. Esta es una versión más simple del `simpleRandomRace` método, pero en este caso todo está predefinido (no aleatorizado):

  - Disponemos de una pista con 5 tramos de 1000, 2000, 3000, 2000 y 1000 metros de longitud; Puedes elegir la velocidad máxima.
  - Disponemos únicamente de 3 vehículos, serán Scooters. Proporcione a los vehículos descripciones y conductores adecuados.

- Corre la carrera.

El código será revisado. Las preguntas sobre el código se harán de forma aleatoria a los estudiantes.

Ordene los resultados para que imprimamos los diferentes vehículos/conductores en orden, por posición. Para ello `start` se puede modificar el método justo en la última parte, la que tiene el comentario `MODIFY THIS to show the results sorted by total time.`

- Implementar un nuevo tipo de vehículo. Este vehículo es un Kart de 2 marchas. Antes de codificarlo, agréguelo a su diagrama UML.
  - Cada marcha tendrá una velocidad mínima y máxima.
  - Al llamar a `adaptSpeed`, el Kart sólo puede hacer una de las siguientes cosas:
    - Aumentar o disminuir la velocidad dentro de la marcha actual.
    - Cambie una marcha hacia arriba (la nueva velocidad será la velocidad mínima de la segunda marcha).
    - Cambie una marcha hacia abajo (la nueva velocidad puede ser cualquiera dentro del rango de la primera marcha).
- Crea una nueva carrera (es decir, un nuevo método en el Centro de control) y prueba el nuevo Kart.

El código será revisado. Las preguntas sobre el código se harán de forma aleatoria a los estudiantes.

Próximos pasos

- Crea una carrera en la que compitan juntos Scooters y Karts. Recuerde, **el `start` método no se puede modificar en este momento**.
    - Elija 3 scooters reales de una marcha y 3 karts reales de 2 marchas para obtener estadísticas reales sobre la velocidad; no pierda más de 10 minutos buscando; de lo contrario, simplemente invente los números.
- Crea un nuevo tipo de `Section` llamado `StandardOutsideSection`. Este tiene su velocidad máxima teórica, pero también la real. El real es sólo un porcentaje del teórico.
    - Por ejemplo, piense en un día lluvioso. La velocidad máxima real podría ser `0.7 * maxSpeed`. O piense en un día muy soleado; La velocidad máxima real podría ser `1.1 * maxSpeed.`
    - Esto no es aleatorio. Al crear el Objeto se debe proporcionar el porcentaje y también una descripción adecuada.

El código será revisado. Las preguntas sobre el código se harán de forma aleatoria a los estudiantes.

- **A partir de este momento, puede modificar el método de inicio según sea necesario.**
    - Modifica el `start` método: si un vehículo supera la velocidad máxima real de un tramo, se sale de la pista y queda descalificado.
- Al enumerar las posiciones, también deberá mostrar los vehículos descalificados y en qué sección fueron descalificados.
- Corre una nueva carrera de Scooters + Karts, pero con secciones tanto interiores como exteriores.

El código será revisado. Las preguntas sobre el código se harán de forma aleatoria a los estudiantes.

DrivingFrenzy

Pasos finales

- Implemente cierta aleatorización en la velocidad de los vehículos para simular que los conductores a veces cometen errores. Al adaptar la velocidad, todos los vehículos añaden un factor aleatorio a la velocidad multiplicándola por un número aleatorio entre 0,8 y 1,0.
    - No implementes esto directamente en todos los vehículos. Encuentre una manera de implementarlo solo una vez (herencia). Además, haga que el código sea menos redundante; es probable que Scooter y Kart tengan atributos y métodos en común.
- Automatiza la creación de una pista larga, de al menos 50 kilómetros con 5 secciones. Debe ser aleatorio y combinar diferentes tipos de secciones.

El código será revisado. Las preguntas sobre el código se harán de forma aleatoria a los estudiantes.

- Modifique las secciones para que proporcionen información adicional para los conductores. Una sección ofrecerá un método `getVariationLevel` que será -1 si de alguna manera está afectado y los vehículos deben ir más lento, 0 si es estándar o 1 si los vehículos pueden ir más rápido. No proporcione el modificador exacto que se está aplicando...
    - Hazlo similar al último cambio de vehículos.
- Implementar un nuevo tipo de vehículos de 6 marchas, los coches normales. Además, estos coches pueden utilizar la información de `getVariationLevel`.
- Implementa el `adaptSpeed` método similar al de los Karts. Sin embargo, el vehículo tiene un nuevo atributo `driverStyle` que puede ser estándar, agresivo o conservador.
        - Un conductor agresivo intentará jugar con las probabilidades de ir más rápido que la velocidad máxima teórica. Por ejemplo, si el nivel de variación es positivo, será agresivo, si es 0 seguirá siendo agresivo para compensar su propia velocidad de aleatorización (ver punto 1).
        - Un conductor conservador siempre intentará evitar salirse de la pista.
        - Un conductor estándar tendrá un comportamiento medio.
    - Pruébalo en una carrera.

El código será revisado. Las preguntas sobre el código se harán de forma aleatoria a los estudiantes.

- Modifica el código completo para que los vehículos compitan simultáneamente y añade menciones de comentaristas sobre cuándo un coche adelanta a otro, etc. ¡Hazlo real!