PROYECTO DE LA ASIGNATURA
DESARROLLO DE PROGRAMAS
Curso 2022/23

Grado en Ingeniería en Informática en:
Ingeniería del Software - Ingeniería de Computadores

DP Ciclismo

Fecha límite: 12 de diciembre a las 08.30 AM (Inaplazable)

En este documento se especifican las acciones que deben llevarse a cabo para implementar el proyecto completo de la asignatura Desarrollo de Programas. En color azul aparecen las acciones relacionadas con la entrega Final del proyecto.

1.- Introducción

El objetivo del proyecto de la asignatura es construir un sistema que permita realizar una simulación de una competición de carreras ciclistas. En el caso de la asignatura, hemos utilizado algunas ideas de las competiciones reales pero hemos hecho una adaptación propia y ficticia a los conceptos del Paradigma de Orientación a Objetos impartidos en la asignatura.

Así, el proyecto consistirá en simular como una **Organización** de competiciones ciclistas gestiona una carrera compuesta por una serie de **Etapas**. En estas carreras participan distintas **Equipos** compuestos por **Ciclistas** y las **Bicicletas** que los **Equipos** les asignan para competir. El objetivo de los **Ciclistas** será acabar cada **Etapa** con el menor tiempo posible y así tratar de ganar el campeonato de **Ciclistas** y que sus **Equipos** ganen el campeonato por **Equipos**.

2.- Descripción de las clases principales existentes en el proyecto “DP Ciclismo”

2.1- *Etapa*

La clase **Etapa** representa al recorrido donde competirán cada jornada los **Ciclistas** con sus **Bicicletas**. La diferencia entre cada *Etapa* viene marcada por características como la diferente **dificultad** del recorrido y la **longitud** del mismo. Estas características van a influir de diversas formas en el rendimiento del **Ciclista** con su **Bicicleta**.

Requisitos de la clase **Etapa**:

- cada objeto de la clase **Etapa** se caracteriza por tener, al menos, los siguientes campos:
  - **nombre** de **Etapa**.
  - **dificultad**, mide la dureza del recorrido. Así, la **velocidad** que es capaz de alcanzar un ***Ciclista*** con un Bicicleta en una **Etapa** depende **inversamente** de este valor. Es decir, a **mayor dificultad**, **menor será la velocidad** que el **Ciclista** con su **Bicicleta** pueden alcanzar en esa **Etapa**. Este valor se deberá usar para el cómputo de la **velocidad** de cada Ciclista. En cuanto a su **dificultad**, las Etapas pueden tener una dificultad SENCILLA (0.9), NORMAL (1.0) o COMPLEJA (1.1). Entre paréntesis se indica el valor asignado a cada tipo. (Requisito realizable tras trabajar “Unidad 06 - Comportamientos más avanzados”)
  - **distancia**, kilómetros a realizar en el recorrido de la **Etapa**. Influye de forma **directa** en los **minutos** que necesita un **Ciclista** con su **Bicicleta** para finalizar la carrera. Es decir, a **mayor distancia** en una **Etapa**, **mayor será el tiempo necesario** que el **Ciclista** con su **Bicicleta** necesita para finalizar esa **Etapa**. En cuanto a su distancia, una **Etapa** puede tener una distancia CORTA (150), INTERMEDIA (200) o LARGA (225). Entre paréntesis se indica el valor asignado a cada tipo. (Requisito realizable tras trabajar “Unidad 06 - Comportamientos más avanzados”)

Funcionalidad de la clase **Etapa**:

- Se debe poder mostrar las características de una **Etapa**.

2.2- *Bicicleta*

La clase **Bicicleta** representa a las distintas bicicletas que los **Ciclistas** pueden usar en las **Etapas**. La diferencia entre cada **Bicicleta** viene marcada por el diferente **peso** y **perdurabilidad** de la misma. Estas características van a influir de diversas formas en el rendimiento del **Ciclista** y la **Bicicleta** que usa en cada **Etapa**.

Requisitos de la clase **Bicicleta**:

- cada objeto de la clase **Bicicleta** se caracteriza por tener, al menos, los siguientes campos:
  - **nombre** de la **Bicicleta**.
  - **peso**, almacena el peso de la bicicleta concreta. Hay que tener en cuenta que la **velocidad** alcanzada por un **Ciclista** con una **Bicicleta** durante una **Etapa** **dependerá directamente** de la **habilidad** del **Ciclista** e **inversamente** de este campo **peso** y de la **dificultad** de la **Etapa**. En cuanto a su **peso**, una **Bicicleta** puede ser LIGERA (7.35), NORMAL (7.50) o PESADA (7.85). Entre paréntesis se indica el valor asignado a cada tipo. (Requisito realizable tras trabajar “Unidad 06 -

Comportamientos más avanzados")

Funcionalidad de la clase **Bicicleta**:

- Debe ser capaz de calcular su **velocidad** cuando es usada por un **Ciclista** en particular en una **Etapa** en concreto, siendo esta **velocidad** el resultado de calcular:
  - **velocidad** = (**habilidad** del **Ciclista * 100) /** (**peso** de la **Bicicleta** * **dificultad** de la etapa);
- Debe ser capaz de proporcionar el **tiempo necesario (medido en minutos)** para terminar la **Etapa** cuando es usada por un **Ciclista** en particular en una **Etapa** concreta, siendo este **tiempo** el resultado de calcular:
  - **tiempo (en minutos)** = (**distancia** de la **Etapa / velocidad** de la **Bicicleta** con **Ciclista** concreto en **Etapa** concreta) *** 60**;

Además de los modelos normales de la clase Bicicleta, existirán **también dos tipos especiales de Bicicleta**: **BicicletaRapida** y **BicicletaPrototipo**. (Requisito realizable tras trabajar "Unidad 04 - Herencia")

Estos dos tipos serán excluyentes entre sí. Es decir, tendremos objetos de tipo **Bicicleta** normal, objetos de tipo **BicicletaRapida** u objetos de tipo **BicicletaPrototipo** pero no de varios tipos a la vez.

2.2.1- BicicletaRapida

Requisitos de la clase

- Tienen las mismas características que las Bicicletas normales.
- Además, se caracterizan por tener un campo llamado velocidadExtra. Este campo indica la velocidad extra que pueden alcanzar este tipo de bicicletas y será inicializado a la hora de crear objetos de la clase BicicletaRapida.

Funcionalidad de la clase **BicicletaRapida**:

- Debe ser capaz de calcular su **velocidad** cuando es usada por un **Ciclista** en particular en una **Etapa** en concreto. Para un objeto de **BicicletaRapida** esta **velocidad** se calcula de **la misma forma** que en la clase **Bicicleta**, sumándole después su valor de **velocidadExtra**.
- Debe ser capaz de proporcionar el **tiempo necesario** para terminar la **Etapa** cuando es usada por un **Ciclista** en particular en una **Etapa** concreta. Este tiempo se calculará de **la misma forma** que en la clase **Bicicleta**.

2.2.2- BicicletaPrototipo

Requisitos de la clase

- Tienen las mismas características que las Bicicletas normales.

Funcionalidad de la clase **BicicletaPrototipo**:

- Debe ser capaz de calcular su **velocidad** cuando es usada por un **Ciclista** en particular en una **Etapa** en concreto. Para un objeto de **BicicletaPrototipo** esta **velocidad** se calcula de **la misma forma** que en la clase **Bicicleta**.
- Debe ser capaz de proporcionar el **tiempo necesario (medido en minutos)** para terminar la **Etapa** cuando es usada por un **Ciclista** en particular en una **Etapa** concreta, siendo este **tiempo** el resultado de calcular:
  - **tiempo (en minutos)** = (**distancia** de la **Etapa / (velocidad** de la **Bicicleta** con **Ciclista** concreto en **Etapa** concreta * **destreza** del **Ciclista** concreto**)**);

2.3- Ciclista

La clase **Ciclista** representa a los ciclistas que competirán con sus **Bicicletas** cada **Etapa**. La diferencia entre cada **Ciclista** viene marcada por la diferente **habilidad** y **energía** del **Ciclista**. Estas características van a influir de diversas formas en el rendimiento del **Ciclista** y su **Bicicleta** en cada **Etapa**.

Requisitos de la clase **Ciclista**:

- cada objeto de la clase **Ciclista** se caracteriza por tener, al menos, los siguientes campos:
  - **nombre** del **Ciclista**,
  - **bicicleta** del **Ciclista**, es la **Bicicleta** asignada por el **Equipo** al que pertenece el **Ciclista** antes de una **Etapa**. Para poder competir, cada **Ciclista** debe llegar a la **Etapa** con una **Bicicleta** asignada por su **Equipo**. Esta **Bicicleta** puede variar entre **Etapa** y **Etapa** e incluso estar a un valor *nulo* si el **Equipo** no tiene una **Bicicleta** concreta para asignar al **Ciclista**. En el caso de que el **Equipo** no tenga disponible una **Bicicleta** concreta para asignar al **Ciclista**, el **Ciclista** no tendrá **Bicicleta** asignada y se mostrará un mensaje indicando que el **Ciclista** no puede ser enviado a esa **Etapa** por no tener una **Bicicleta** disponible y que **abandona** la competición.
  - **habilidad**, mantiene un valor con la habilidad del **Ciclista**. La habilidad de un Ciclista puede ser LENTA (4.0 ~~90.0~~), NORMAL (6.0 ~~100.0~~) o BUENA (8.0 ~~110.0~~). Entre paréntesis se indica el valor asignado a cada tipo. (Requisito realizable tras trabajar "Unidad 06 - Comportamientos más avanzados")
  - **energía**, almacena la energía restante que le queda a un **Ciclista** para finalizar la competición. Se pierde **una unidad de energía por cada minuto de tiempo compitiendo**. Si un **Ciclista** se queda sin **energía**, tendrá que abandonar la competición en ese momento sin terminar ya la Etapa en la que le haya sucedido.
  - **resultados**, cada **Ciclista** debe guardar un registro con los **resultados** obtenidos (tiempo) en cada Etapa que haya disputado que vaya disputando de la competición.
  - **equipo**, es el **Equipo** al que pertenece el **Ciclista**

Funcionalidad de la clase **Ciclista**:

- Debe ser capaz de recibir la **Bicicleta** que le asigne su **Equipo** y poder cambiarla entre carrera y carrera.
- Debe ser capaz de comprobar e informar si ha **abandonado** o no (si tiene suficiente energía) para seguir participando en el campeonato.
- Debe ser capaz de gestionar y proporcionar información sobre el resultado obtenido en cualquier **Etapa**.
- Debe ser capaz de devolver el número total de Etapas en las que ha participado, las que ha terminado, total de tiempo acumulado en las etapas que consiguió terminar, y, su caso, la Etapa en la que abandonó.
- Debe proporcionar la funcionalidad necesaria para que el *Ciclista* pueda **usar** una ***Bicicleta*** en una **Etapa**. Así, esta funcionalidad se encargará de:
  - Si el ***Ciclista***, conduciendo la **Bicicleta** asignada en la **Etapa** indicada, no ha tenido que abandonar por falta de energía:
    - guardar en **resultados** el **tiempo** (en minutos) que ha necesitado para terminar la carrera
    - restar a su **energía** el número de minutos que ha tardado en finalizar la carrera.
  - Si el Ciclista se quedó sin **energía** antes de poder terminar una **Etapa**:
    - guardar como **resultado** de esa Etapa el **número negativo** que indica **los minutos de energía** extra que hubiera necesitado para poder terminar la carrera.
- Debe ser capaz de calcular la **destreza** de un Ciclista. Esta **destreza** dependerá de su **habilidad** y del tipo de **Ciclista** que sea. La explicación sobre los tipos de **Ciclista** y la

fórmula concreta de cada tipo de **Ciclista** para calcular su **destreza** se explica a continuación.

2.3.1- Tipos de Ciclista

Existirán tres tipos de Ciclista: CiclistaNovato, **CiclistaExperimentado** y **CiclistaEstrella**. (Requisito realizable tras trabajar "Unidad 04 - Herencia")

- **Estos tres tipos serán excluyentes entre sí y de los únicos que se pueden crear objetos de tipo Ciclista**. Es decir, no habrá la posibilidad de crear objetos de tipo **Ciclista** que no sean de tipo **CiclistaNovato**, **CiclistaExperimentado** o **CiclistaEstrella**.
- Los tres tipos de *Ciclista* se diferencian en cómo calculan la **destreza** de un *Ciclista* a partir de ciertos factores constantes:.
    - **CiclistaNovato**:
        - **destreza** = ((**habilidad** del *Ciclista* **+ 2**) **/ 120**) * 10**;**
    - **CiclistaExperimentado**:
        - **destreza** = ((**habilidad** del *Ciclista* **+ 4**) **/ 130**) * 10**;**
    - **CiclistaEstrella**:
        - **destreza** = (((**habilidad** del *Ciclista* **+ 6**) **/ 140**) * 10**;**
- Además, la clase **CiclistaEstrella** añade la siguiente funcionalidad:
    - Sólo los Ciclistas de tipo **Estrella** implementarán una característica llamada "SerPopular" que consiste en aumentar o disminuir su nivel de popularidad en redes sociales dependiendo de los resultados que vaya consiguiendo (esta característica puede plantearse únicamente para este tipo de ciclistas o de una forma más genérica para que pueda ser reutilizada por otros tipos de objetos de distintas clases o jerarquías (deportistas, actores, cantantes, etc.). Para ello:
        - Los ciclistas de tipo **CiclistaEstrella** tendrán un valor inicial de popularidad de 6 unidades.
        - Este valor se aumentará (siempre en 4 unidades) si después de usar la bicicleta en una etapa hace un tiempo menor a 160 en dicha etapa (y aún le queda energía para continuar). ~~Por su parte, si el tiempo realizado en la etapa es mayor o igual a 160.~~ En caso contrario, su nivel de popularidad disminuirá en 1 unidad.

2.4.- *Equipo*

Representa los ***Equipos*** que competirán tanto por ganar la **clasificación por Equipos** como que uno de sus **Ciclistas** gane el **campeonato individual** de **Ciclistas**.

Cada **Equipo** tendrá una serie de **Ciclistas** y **Bicicletas** en exclusiva. Su labor principal es mantener un equipo de **Ciclistas** y **Bicicletas**. Cada **Equipo** se caracteriza además por poder tener distintos criterios de ordenación para sus *Bicicletas* y sus *Ciclistas* para decidir qué *Bicicleta* asignar a un *Ciclista* y en qué orden enviar en cada **Etapa** a la **Organización** los **Ciclistas** junto con su **Bicicleta** asignada**.** Es decir, un **Equipo** puede preferir ordenar sus Ciclistas y Bicicletas de manera que, por ejemplo, primero envía al **Ciclista** con más **energía** junto con la **Bicicleta** con menos **peso,** al segundo **Ciclista** con más **energía** junto con la segunda **Bicicleta** con menos **peso,** etc.

Requisitos de la clase **Equipo**:

- cada objeto de tipo **Equipo** se caracteriza por tener, al menos, los siguientes campos:
    - **nombre** del equipo.
    - **ciclistas**, una estructura de datos donde se almacenan los **Ciclistas** del **Equipo**.
        - Esta estructura se mantendrá ordenada de acuerdo a un criterio de comparación de **Ciclistas**, que se podrá aplicar de forma ascendente o

descendente, basado en el uso de una o varias de sus características como **energía, nombre, total de minutos acumulados**, etc.. Dicho orden podrá ser completamente distinto para cada **Equipo** e incluso podría ser modificado durante el campeonato.

- **ciclistasAbandonado**, una estructura de datos donde se almacenan los **Ciclistas** del **Equipo que han abandonado**. Esta estructura se mantendrá ordenada de acuerdo al orden de inserción.
- **bicicletas**, una estructura de datos donde se almacenan las **Bicicletas** del **Equipo**.
    - Esta estructura se mantendrá ordenada de acuerdo a un criterio de comparación de **Bicicletas**, que se podrá aplicar de forma ascendente o descendente, basado en el uso de una o varias de sus características como **peso** o **nombre**. Dicho orden podrá ser completamente distinto para cada **Equipo** e incluso podría ser modificado durante el campeonato.

Funcionalidad de la clase **Equipo**:

- Debe ser capaz de ordenar sus *Ciclistas* y sus *Bicicletas* de acuerdo a los criterios de comparación que tenga el **Equipo** teniendo en cuenta además que dichos criterios podrían ser **modificados durante** el campeonato.
- Debe ser capaz de proporcionar información sobre el total de tiempo acumulado por sus **Ciclistas**.
- Debe ser capaz de enviar en cada **Etapa** a sus **Ciclistas** sin abandonar junto con las **Bicicletas** asignadas. Se considera que un equipo siempre tendrá al menos tantas Bicicletas como Ciclistas tenga el Equipo..

2.5.- *Organización*

Representa a la clase capaz de gestionar el campeonato.

Requisitos de la clase **Organización**:

- se caracteriza por tener, al menos, los siguientes campos:
    - **etapas**, estructura con las Etapas que componen el campeonato, ordenadas según un criterio de comparación de Etapas basado en una o varias de sus características (**dificultad**, **distancia** o **nombre**). Dicho criterio de comparación será establecido en cada simulación en el momento de creación de la instancia de Organización y **no** será cambiado a lo largo del campeonato. Además, en la entrega final, las etapas se almacenarán en una colección que no permita etapas duplicadas y las mantenga ordenadas según el criterio de ordenación definido al crear la organización (requisito realizable tras trabajar “Unidad 06 - Comportamientos más avanzados”).
    - **equipos**, estructura con todos los ***Equipos*** que se han inscrito para participar en la carrera. Esta estructura se mantendrá ordenada de acuerdo a un criterio de comparación de Equipos, que se podrá aplicar de forma ascendente o descendente, basado en el uso de una o varias de sus características como, **nombre, total de minutos acumulados por sus corredores**, etc..
    - **ciclistasCarrera**, estructura con los *Ciclistas* que van a competir en una determinada carrera. Esta estructura se mantendrá ordenada de acuerdo a un criterio de comparación de **Ciclistas**, que se podrá aplicar de forma ascendente o descendente, basado en el uso de una o varias de sus características como **energía, nombre, total de minutos acumulados**, **nombre**, etc.. Dicho orden podrá ser completamente distinto para cada **Equipo** e incluso podría ser modificado durante el campeonato.

Funcionalidad de la clase **Organización**:

- Debe ser capaz de inscribir **Equipos** en la competición por etapas que organiza.
- Debe ser capaz de gestionar el desarrollo de la carrera de manera que, una vez tiene las

Etapas que componen la competición y los **Equipos** que van a competir:

- ordenará y mostrará las Etapas que componen la carrera
- mostrará las Equipos que van a competir
- gestionará la competición, realizando todas las Etapas que haya disponibles (el detalle de la gestión de cada Etapa se muestra más adelante) siempre que no se dé alguna de las siguientes **condiciones de finalización**:
  - No hay **Ciclistas** disponibles para competir porque todos han abandonado.
  - Solo queda un **Ciclista** para competir porque el resto ya ha abandonado.
- Una vez finalice la carrera, realizará la **Ceremonia de Entrega de Premios**:
  - En caso de que **todos los *Ciclistas* hayan abandonado**, se declarará el ganador de **Ciclistas** y de **Equipos** como desierto. Es decir, ningún **Ciclista** ni **Equipo** habrá ganado el campeonato.
  - En caso de que **no todos los *Ciclistas* hayan abandonado,** el **Ciclista** ganador de la carrera será el **Ciclista** con menos minutos acumulados entre los **Ciclistas sin abandonar** y el **Equipo ganador aquel que menos minutos de media acumule entre sus *Ciclistas* sin abandonar**.
  - En **todos los casos** se mostrará como resumen final del campeonato:
    - los **Ciclistas** sin abandonar, si los hay, junto con los resultados obtenidos, ordenados de forma ascendente de acuerdo al total de minutos acumulados. En caso de empate, de forma descendente por el nombre del Ciclista.
    - los **Ciclistas** que han abandonado, si los hay, junto con los resultados obtenidos hasta su abandono, ordenados de forma ascendente de acuerdo al total de minutos acumulados hasta el momento de su abandono. En caso de empate, de forma descendente por el nombre del Ciclista.
    - los **Equipos** con **Ciclistas** sin abandonar, si los hay, junto con los minutos acumulados y el estado de sus ***Ciclistas*** al terminar el campeonato. Estarán ordenados de forma ascendente de acuerdo a la medial total de minutos acumulados por sus **Ciclistas sin abandonar**. En caso de empate, de forma descendente por el nombre del **Equipo**.
    - Se mostrarán los **Equipos**, si los hay, en los que todos los ***Ciclistas*** han abandonado junto con sus **Ciclistas**. Estarán ordenados de forma ascendente por el nombre del **Equipo**.

- Debe ser capaz de gestionar la **celebración de cada Etapa del campeonato**, de manera que:
  - Solicitará en cada turno a cada **Equipo** que le envíe los **Ciclistas** que tiene.
    - la **Organización** irá incluyendo a cada ***Ciclista***, junto con su Bicicleta asignada, en el orden recibido en la estructura de datos **Ciclistas.**
    - Si **hay al menos un *Ciclista* disponible, se puede competir.** Para ello:
      - se ordenará a los *Ciclistas* disponibles de manera **descendente** de acuerdo al total de minutos acumulados en los **resultados** de **Etapas** anteriores de cada **Ciclista**, y, en caso de igualdad, de forma **ascendente** por el nombre del Ciclista.
      - siguiendo este orden, a cada **Ciclista** se le pedirá que conduzca su **Bicicleta** en la **Etapa** correspondiente a ese turno.
        - De cada **Ciclista** que vaya compitiendo se mostrará:
          - sus características
          - las características de su **Bicicleta**

        - la velocidad que es capaz de alcanzar en esa **Etapa** con ese **Bicicleta**
        - en caso de terminar la carrera:
            - el tiempo obtenido
        - en caso de no terminar la carrera:
            - la cantidad de **energía** que le faltaba para finalizar la **Etapa**
            - el tiempo que llevaba en la **Etapa** en el momento del abandono
    - Se mostrarán los resultados obtenidos en la carrera por todos los **Ciclistas**, incluyendo:
        - los **Ciclistas** que han terminado la carrera ordenados por su posición en la misma
        - los ***Ciclistas*** que han abandonado
    - Al finalizar la carrera se devolverán los **Ciclistas** (junto con sus **Bicicletas**) a sus **Equipos**.

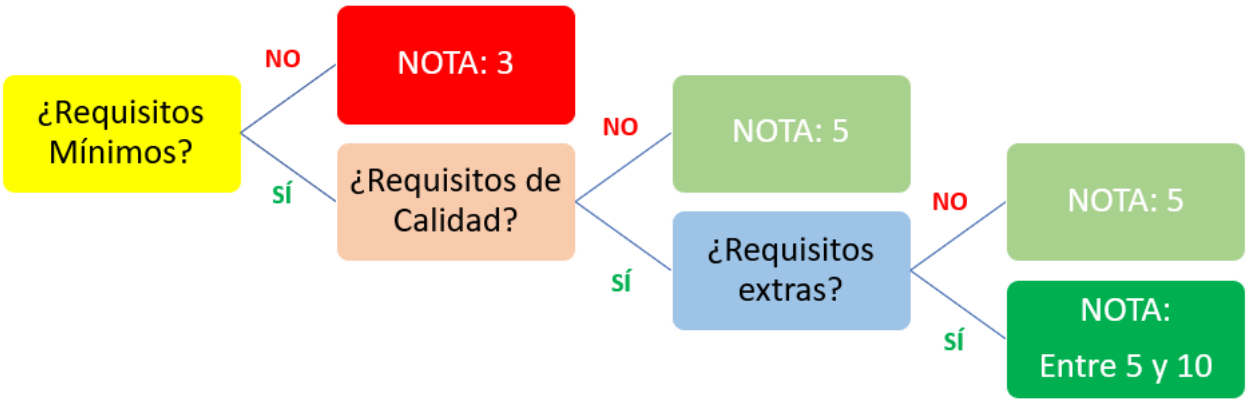
3.- Datos iniciales proporcionados

Se proporcionarán dos ficheros con unos datos iniciales de prueba para poder ejecutar la simulación con los mismos datos de partida. Estos ficheros son "**DatosCampeonatoCompleto.java**" y ""DatosCampeonatoAbandonos y están en la carpeta del proyecto del Campus Virtual (CVUEx).

4.- Requisitos y Evaluación

En este apartado se definen los **requisitos mínimos,** los **requisitos de calidad** y los **requisitos extras de la entrega inicial del proyecto.**

La **forma** en que el **cumplimiento de estos requisitos afecta a la nota final** se muestra en la siguiente figura:



a. Requisitos mínimos

5. **Todas las clases deben incluir los constructores, métodos set y get de cada campo y métodos auxiliares que se consideren necesarios** (toString, equals, etc..) para que su funcionalidad sea correcta. Además, deben definir correctamente su ámbito de uso con los **modificadores de acceso**

correspondientes.

6. El proyecto **debe proporcionar la funcionalidad detallada** en este documento. Para ello, **se deberá poder comprobar el correcto funcionamiento** utilizando una clase llamada "***CiclismoDemo.java***" desarrollada por cada grupo que leerá el conjunto de datos de inicialización que se utilizará en cada caso para realizar la simulación e invocará el/los método/s necesarios del Campeonato para llevar a cabo la simulación. **La funcionalidad proporcionada debe asegurar que**:
    a. El sistema funciona siempre dentro de unos parámetros de tiempo y uso de recursos adecuados.
    b. El sistema controla todos los posibles errores de funcionamiento, avisando al usuario correctamente de su causa.

- La ejecución de la simulación del proyecto generará una salida **por pantalla**. Esta salida **deberá seguir el formato especificado en la plantilla de salida disponible en la carpeta con el enunciado del proyecto**.
    - Se puede tomar como referencia de salida de ejemplo con esta plantilla los ficheros de salida generados con los 2 conjuntos de datos iniciales disponibles en la carpeta con el enunciado del proyecto.

- El proyecto entregado por cada grupo **será capaz de adaptarse a cualquier conjunto de datos iniciales y generar la salida adecuada**.
    - Junto con el código del proyecto al menos deberán entregarse los ficheros con los 2 conjuntos de datos iniciales adaptados al proyecto así como la salida generada por el proyecto por cada uno de esos conjuntos de datos iniciales.

- El proyecto debe ejecutarse desde el principio hasta el final **sin necesidad de solicitar la intervención del usuario**.

- **Aplicación correcta de los conceptos clase, encapsulación y composición**:
    - Se definen correctamente las clases necesarias para el proyecto, especificando adecuadamente su interfaz funcional público.
    - Todas las clases aplican correctamente el principio de encapsulación sin exponer en su interfaz funcional pública detalles de implementación de sus datos.
    - Se utiliza adecuadamente la composición en la definición de las clases del proyecto.

- En el proyecto se habrá hecho un **uso correcto de Collections:** (mínimo **List, Set y Map**)
    - Se utilizarán correctamente, junto con los métodos que proporcionan, **al menos una vez cada Collection**, utilizándola de forma apropiada en cada caso, de acuerdo a sus características.

- **Formato de entrega adecuado:**
    - El proyecto entregado seguirá las especificaciones del formato de entrega detalladas en el apartado 5 de este enunciado.

- **Uso correcto de la funcionalidad ofrecida por las Colecciones y los**

Comparadores:

- Se utilizará en cada momento el método más adecuado de los ofrecidos por las Collections para operar sobre los campos **competidores** y **eliminados** de la clase **Campeonato**:
  - Así, para valorar este apartado correctamente se tendrá en cuenta que en ningún momento se haga uso de variables auxiliares de tipo Collections en la clase **Organización**. Para realizar la funcionalidad requerida para la clase **Organización** es suficiente con usar los métodos adecuados de las Collections.
- Se entregarán, al menos, los siguientes comparadores implementados de forma correcta (o se implementará correctamente en la parte correcta su alternativa usando Lambdas y Streams):
  - Uno que compare las habilidades de 2 ciclistas y que en caso de empate use el nombre del ciclista como criterio de desempate.
  - Uno que compare la energía de 2 ciclistas y que en caso de empate use el nombre del ciclista como criterio de desempate.
  - Uno que compare el total de tiempo acumulado por 2 ciclistas y que en caso de empate use el nombre del ciclista como criterio de desempate.
  - Uno que compare el último tiempo conseguido por 2 ciclistas y que en caso de empate use el nombre del ciclista como criterio de desempate.
  - Uno que compare el peso de 2 bicicletas y que en caso de empate use el nombre de la bicicleta como criterio de desempate.
  - Uno que compare la dificultad de 2 etapas y que en caso de empate use el nombre de la etapa como criterio de desempate.
  - Uno que compare la media de tiempo acumulado por los ciclistas de 2 equipos y que en caso de empate use el nombre del equipo como criterio de desempate.

- En el proyecto se habrá hecho **una aplicación correcta de los conceptos herencia y polimorfismo**:
  - Se definen todas las jerarquías de clases necesarias para el proyecto de manera correcta.
  - Se define correctamente el interfaz funcional público de la clase base de la jerarquía para poder sacar provecho del polimorfismo.
  - Se definen todos los algoritmos polimórficos y todas las estructuras de datos polimórficas necesarias para el sistema, sacando el máximo provecho del polimorfismo y evitando en todos los casos la repetición de código.

- **Pruebas unitarias del proyecto con JUnit**:
  - Se ha elaborado un conjunto completo de pruebas de los **métodos** de las **clases** (y de los **métodos específicos** de sus **posibles subclases**). Como mínimo y de forma obligatoria se incluirán pruebas para **las clases y métodos siguientes:**
    - Ciclista:
      - correr(Etapa etapa)
      - getTotalMinutosEnCarrerasTerminadas() <- con al menos 2 carreras realizadas
    - Bicicleta:
      - calcularVelocidadConCiclistaEnEtapa(Ciclista ciclista, Etapa etapa)
      - calcularMinutosEnEtapa(Ciclista ciclista, Etapa etapa)
    - Equipo:

        - getMediaMinutosEquipo() <- con al menos 2 ciclistas en equipo y con al menos 2 carreras realizadas cada ciclista
        - enviarSiguienteCiclista() <- con al menos 2 ciclistas en equipo
    - Es decir, estas pruebas contendrán **un método de test** con sus correspondientes **aserción/es para cada método público** que **devuelva** algún valor (excepto los métodos **get** de los campos de la clase) o **realice algún cambio en el estado de los objetos** (excepto de los métodos **set** de los campos de la clase) de la clase a probar.
    - Además, debe incluir de forma obligatoria **la creación de objeto/s necesario/s para las pruebas mediante el uso de *Fixtures***.

---

c. Requisitos de calidad

- **Estilo de código adecuado. El código entregado cumple las especificaciones del estilo de código vistas a lo largo de la asignatura**, entre las que por ejemplo se encuentran:
    - Correcta indentación de todo el código
    - Uso de un único estilo en los nombres de clases, variables y métodos así como en los comentarios, nombres de ficheros, declaraciones de funciones, bucles, especificación de condiciones e inicialización de variables.
    - Utilizar de forma adecuada las distintas estructuras de iteración. En este sentido:
        - solo se utilizan bucles for para iteración definida
        - bucles while para iteración indefinida

- **Documentación interna y externa adecuada. El código entregado usa de forma correcta la funcionalidad JavaDoc y sus anotaciones para proporcionar:**
    - un resumen de lo que realiza cada clase
    - la explicación de todos sus métodos (y sus entradas y salidas) correspondientes (@param, @return, etc.)

---

d. Requisitos extras

Como aportaciones adicionales a la entrega se pueden realizar las siguientes:

- **Uso de la funcionalidad de Git proporcionada por BlueJ** (hasta **1 punto**):
    - Se valorará que se haya hecho un **uso adecuado y continuado** a lo largo del desarrollo del proyecto (al menos desde 2 semanas antes de la fecha de la entrega).
    - Se valorará especialmente este apartado en los proyectos realizados en grupo y, sobre todo, en los grupos de 3 miembros.

- **Uso correcto de Iterator** (**0,5 puntos**):
    - Se utilizará correctamente, en la funcionalidad apropiada a sus características, junto con los métodos que proporciona, **al menos dos veces**.

- **Usar cuando sea apropiado y de forma correcta un tipo Enum** (hasta **1 punto**)

- **Usar cuando sea apropiado y de forma correcta Interfaces** en relación a la herencia definida (hasta **0,5 puntos**)

- En el proyecto se habrá hecho un **uso correcto y justificado del mecanismo de excepciones** (lanzar y capturar excepciones) (hasta **1 punto**)

- **Escritura de la salida por fichero con el mismo formato que el utilizado para la salida por defecto por pantalla**. (hasta **1 punto**)

- **Pruebas unitarias del proyecto con JUnit** (hasta **1 punto en las pruebas nuevas añadidas en la entrega Final**):
    - Se ha elaborado un conjunto completo de pruebas de los **métodos** de las **clases** (y de los **métodos específicos** de sus **posibles subclases**). **Como mínimo** se incluirán pruebas para **las clases y métodos siguientes:**
        - Ciclista:
            - getDestreza()
            - Etapa getEtapaAbandono( ) (devuelve la etapa en la que ha abandonado)
            - getResultadoEtapa(Etapa e) (devuelve el resultado en una etapa concreta)
            - getEtapasTerminado( ) (devuelve el número total de etapas que ha finalizado)
        - Equipo:
            - ordenarCiclistas ( )
            - ordenarBicicletas ( )
        - Organización:
            - inscribirEquipo(Equipo e)
            - comprobarTodosCiclistasAbandonado( ) (método que comprueba si todos los ciclistas han abandonado o hay ciclistas para competir)

    - Es decir, estas pruebas contendrán **un método de test** con sus correspondientes **aserción/es para cada método público** que **devuelva** algún valor (excepto los métodos **get** de los campos de la clase) o **realice algún cambio en el estado de los objetos** (excepto de los métodos **set** de los campos de la clase) de la clase a probar.
    - Además, debe incluir de forma obligatoria **la creación de objeto/s necesario/s para las pruebas mediante el uso de *Fixtures***.
    - En la entrega Final se valorarán y puntuarán aquellas nuevas pruebas realizadas además de las solicitadas en la entrega inicial.

5.- Formato de la entrega

El código de la entrega contendrá **exclusivamente el código necesario para satisfacer las especificaciones** de la misma. Para realizar esta entrega se habilitará una **tarea en el aula virtual**.

Requisitos sobre la entrega:

- Además de tener en cuenta todas las consideraciones detalladas en el apartado “Requisitos mínimos” de este enunciado, los estudiantes deben también tener en cuenta:
    - El proyecto entregado debe poder abrirse y probarse en el entorno **BlueJ** 5.0.2.o posterior con Java 11 sin que sea necesario por parte de los profesores realizar ninguna acción adicional para su ejecución y pruebas.
    - El proyecto entregado **solo contendrá una carpeta cuyo contenido será**

exclusivamente:

- Las **clases Java** que implementan la funcionalidad del proyecto
- El **fichero de inicialización** creado a partir del conjunto de datos de inicialización proporcionado.
- Un fichero "**Resumen.txt**" (o "**Readme.txt**") que especifique:
  - nombre y apellido de cada miembro del grupo
  - nombre de la clase que hay que ejecutar para realizar una simulación
  - los extras que en opinión del grupo tiene su proyecto

- El nombre de dicha carpeta será:
  - apellidos1y2DeEstudiante1_apellidos1y2DeEstudiante2_apellidos1y2DeEstudiante3(*)
    (*) se aplicará el orden alfabético sobre el primer apellido para decidir el orden de los estudiantes
- La carpeta del proyecto deberá entregarse comprimida:
  - El formato a usar será **exclusivamente** el formato **.zip.**
    - **La entrega en cualquier otro formato, como .rar, se considerá no entregada.**
  - El nombre de este fichero.zip tendrá el mismo nombre que la carpeta del proyecto

6.- Fecha de la entrega

Cada grupo de estudiantes deberá realizar la **entrega del proyecto antes de las 08:30 am horas del 12 de diciembre**.
El fichero .zip de la entrega debe ser subido POR TODOS los miembros del grupo.

Anexo I. Archivos con plantilla de formato, datos de inicio para pruebas y su salida esperada.

Junto con el enunciado del proyecto, los estudiantes tienen a su disposición una carpeta que contiene los siguientes ficheros:

- CiclismoDemo**Final**.java: contiene (en pseudocódigo) la clase con el método *main* de la simulación. Básicamente este método *main* en primer lugar cargará los datos de inicio y después invocará al método de Organización que *gestiona* el campeonato. El pseudocódigo mostrado en esta clase deberá adaptarse a la implementación del proyecto de cada grupo de trabajo.
- DatosCampeonatoCompleto**Final**.java y DatosCampeonatoAbandonos**Final**: contienen (en pseudocódigo) las clases con los datos de inicio de la simulación.
- SalidaDatosCampeonatoCompleto**Final** y SalidaDatosCampeonatoAbandonos**Final**: son las plantillas que contienen el formato de cómo debe mostrarse la información por pantalla al realizar las simulaciones.