Guía N° 4: Java Avanzado

Contents

Ejercicio 01: Igualdad de objetos

1. Crear una clase Ciudad, que representará una ciudad en una ruta, con los siguientes atributos:
   - String nombre
   - Integer codigoPostal
2. Crear 2 constructores
   2.1. El constructor por defecto, sin argumento y con cuerpo vacío.
   2.2. El constructor que recibe el nombre y el código postal como argumento y lo setea a sus atributos
3. Generar los métodos getters y setters
4. Sobrescribir el método toString en la clase Ciudad
5. Sobrescribir el método equals, de forma tal que dos ciudades son iguales **si y solo si tienen el mismo código postal, sin importar si sus nombres coinciden o son diferentes.**
6. Generar una clase de prueba para testear el funcionamiento de la clase Ciudad

7. Crear la clase Ruta, que tendrá solo una variable de instancia de tipo ArrayList, que almacenará punteros a objetos de tipo Ciudad, creados en el punto 1.
8. Crear en ruta el constructor por defecto en cuyo cuerpo se inicializa el ArrayList, haciéndolo apuntar a una nueva instancia de ArrayList
9. Crear en ruta los métodos getters y setters
10. Crear el m[método “agregarCiudad(Ciudad c)” que agrega una ciudad a la ruta si es que la misma no estaba agregada. **Para verificar que una ciudad está en la ruta puede utilizar el método de arrayList, contains (https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#contains-java.lang.Object-)**
11. Generar una clase de prueba para testear el funcionamiento de la clase Ruta

Ejercicio 02: Comparar objetos

12. Crear una clase Equipo, que represente un equipo de futbol que tiene los siguientes atributos:

- String nombre
- Integer puntos
- Integer partidosJugados
- Integer goles a favor
- Integer goles en contra
-

13. Crear los métodos gettters y setters.
14. Crear en la clase Equipo el método registrarPartido(Integer golesFavor,Integer golesEncontra)
    14.1. Este método incrementa los goles a favor, y los goles en contra.
    14.2. si la cantidad de goles a favor es mayor que la de goles en contra suma 3 puntos.
    14.3. Si la cantidad es igual, suma un punto y si es menor no suma ningún punto.
    14.4. Todo equipo que gane un partido por una diferencia de más 4 o más goles a favor sumará un punto bonus extra.
15. Sobrescribir el método equals() que determine que dos equipos son iguales si tienen el mismo nombre.
16. Sobrescribir el método toString() para que cuando se envie el equipo por consola muestre el nombre, los puntos y los partidos jugados.
17. Hacer que Equipo implemente la interface Comparable. Un equipo aparece antes que otro, si tiene más puntos que otro. Si tienen el mismo puntaje, aparece antes el que tiene mayor diferencia de gol. Si persiste la igualdad aparecerá en primer lugar el que tiene mas goles a favor. Finalmente se determina el ordenamiento alfabetico.
18. Crear un método main donde se realice lo siguiente
    18.1. Se crean 5 equipos y se registran diversos resultados en cada uno de ellos.
    18.2. Se agregan los 5 equipos a un ArrayList
    18.3. Se imprime el ArrayList
    18.4. Se invoca sobre el ArrayList el método "Collections.sort"
    18.5. Se imprime el ArrayList ahora ordenado
    18.6. Se invoca sobre el ArrayList el método "Collections.sort" pasando como argumento una clase anónima que implemente Comparator y compare los equipos por promedio de puntos.
    18.7. Se imprime el ArrayList ahora ordenado por promedio de puntos.
    18.8. Se invoca sobre el ArrayList el método "Collections.sort" pasando como argumento una expresion lambda que compara los equipos por diferencia de gol.
    18.9. Se imprime el ArrayList ahora ordenado por diferencia de gol.

Ejercicio 02 Parte 2: Interface comparator

19. Crear la clase Torneo. La clase torneo tiene como atributos un titulo y un ArrayList de equipos.
    19.1. Crear el método “addEquipo(String nombre)” que recibe como parámetro un equipo y si no pertenece a la lista lo agrega. **Como verifico que el equipo no pertence a la lista? La clase ArrayList posee el método contains (https://docs.oracle.com/javase/8/docs/api/java/util/ArrayList.html#contains-java.lang.Object-)**

20. Crear el método registrarPartido(String nombreLocal, Integer golLocal,String nombreVisita,Integer golVisita)
    20.1. Este método realiza lo siguiente: busca los equipos en la lista, y a cada equipo le invocan el método "registrarPartido(int,int)" de la clase Equipo.
21. Crear el método "void tablaPosiciones()" que imprime la lista de equipos ordenado por el puntaje que poseen.
22. Crear el método "void tablaPromedios()" que imprime la lista de equipos ordenado por el promedio de puntos obtenidos en la cantidad de partidos jugados. **Para esto puede crear una implementación de la interface Comparator<Equipo,Equipo> que compare los equipos según el promedio y pasarla como parámetro al método Collections.sort(lista,comparador)**
    https://docs.oracle.com/javase/8/docs/api/java/util/Collections.html#sort-java.util.List-java.util.Comparator-

Ejercicio 03: Clases abstractas

Una universidad necesita imperiosamente mejorar su sistema de gestión académica y se le solicita que cree en una primer etapa un sistema que permita registrar las incripciones de los alumons a los cursos de posgrado. Los cursos de posgrado son de dos tipos, **Obligatorios** y **Opcionales**. Todos los cursos tienen un nombre, un cupo máximo de alumnos y otorgan un número de créditos cuando son aprobados.

La diferencia radica en que los cursos obligatorios tienen un arreglo de correlativas. Esto indica que un usuario se puede inscribir a un curso obligatorio si y solo si tiene aprobadas todas las correlativas. Si un curso obligatorio no tiene este requisito, tendrá la lista de correlativas necesarias vacias.

En tanto los cursos opcionales, solamente pueden ser cursados cuando el alumno ha aprobado cursos que le otorgan los créditos requeridos.

Implementar las siguientes clases:

Curso: será una clase abstracta.

- **Atributos**
  .1. `private String nombre;`
  .2. `private int cupo;`
  .3. `private int creditos;`
  .4. `private ArrayList<Alumno> alumnos;`
- **Métodos**
  .1. `public Curso()` → constructor por defecto inicializa lista de alumnos.
  .2. `public Curso(String nombre,int creditos,int cupo)` → constructor sobrecargado para setear el resto de los atributos
  .3. `public void inscribir(Alumno a)` → inscribe alumno a curso si hay cupo
  .4. `public final void aprobar(Alumno a)` → recibe un alumno como parámetro y lo quita de la lista de alumnos (método remove)
  .5. `public Integer plazasDisponibles()` → cantidad de cupos disponibles

.6. `public boolean equals(Object obj)` → dos cursos son iguales si tienen el mismo nombre

CursoOpcional: clase concreta que extiende de Curso.

- **Atributos**
  .1. `private ArrayList<Curso> correlativas;`

- **Métodos**
  .1. `public CursoOpcional()` → constructor por defecto inicializa lista de alumnos.
  .2. `public CursoOpcional (String nombre,int creditos,int cupo ,int requeridos)` → constructor sobrecargado para setear el resto de los atributos entre ellos la cantidad de créditos requeridos
  .3. `public void inscribir(Alumno a)` → inscribe alumno a curso si hay cupo y si el alumno tiene los créditos necesarios
-

CursoObligatorio: clase concreta que extiende de Curso.

- **Atributos**
  .1. `private ArrayList<Curso> correlativas;`

- **Métodos**
  .1. `public CursoObligatorio()` → constructor por defecto inicializa lista de alumnos.
  .2. `public CursoObligatorio(String nombre,int creditos,int cupo)` → constructor sobrecargado para setear el resto de los atributos
  .3. `public void agregarCorrelativa(Curso c)` → agrega un curso como correlativo
  .4. `public Boolean tieneCorrelativas(Alumno a)` → verifica si un alumno tiene las materias correlativas aprobadas
  .5. `public void inscribir(Alumno a)` → inscribe alumno a curso si hay cupo y si el alumno tiene las correlativas aprobadas

`Alumno`: clase concreta que representa un alumno

- **Atributos**
  .1. `private Integer libreta;` → nro de libreta universitaria
  .2. `private String nombre;` → nombre del alumno
  .3. `private ArrayList<Curso> inscriptos;` → está cursando
  .4. `private ArrayList<Curso> aprobados;` → ya aprobó

- **Métodos**
  .1. `public Alumno()`
  .2. `public Alumno(Integer id,String nombre)`
  .3. `public boolean equals(Object obj)` → dos alumnons son iguales si tienen el mismo numero de LU

.4. `public void inscribir(Curso c)` agrega un curso a la lista de inscriptos
.5. `public void aprobar(Curso c)`→ quita un curso de la lista de inscriptos y la agrega a la lista de aprobados
.6. `public Integer creditos()` → cantidad de créditos acumulados en los cursos aprobados

Verificar que puede crear un alumno.

a) Verificar que puede crear un curso opcional, que no tenga créditos requeridos, pero que otorgue 5 creditos.
b) Verificar que el alumno se puede inscribir a dicho curso.
c) Aprobar al alumno que tomó dicho curso.
d) Crear un nuevo curso opcional que requiera 4 creditos y otorgue 2 creditos.
e) Verificar que el alumno se puede inscribir a dicho curso.
f) Aprobar al alumno que tomó dicho curso.
g) Crear un nuevo curso opcional que requiera 8 creditos y otorgue 2 creditos.
h) Verificar que el alumno NO se puede inscribir a dicho curso.
i) Crear un curso obligatorio y agregarle como correlativa tiene el curso opcional creado en el punto a).
j) Verificar que el alumno se puede inscribir a dicho curso.
k) Aprobar al alumno que tomó dicho curso.
l) Agregar al curso creado en el punto i) como curso correlativo el curso creado en el punto g).
m) Verificar que el alumno NO se puede inscribir a dicho curso.

Ejercicio 04: Excepciones

Como continuación del ejercicio 6 crearemos 3 tipos de excepciones

- `InscripcionCursoException` **que extiende de** `Exception`
  a) Esta excepción tiene 2 constructores, el constructor por defecto que seteará en el padre el mensaje `"Hubo un problema al inscribir al curso"`) y el constructor que recibe como argumento un string con el mensaje
- `CupoNoDisponibleException` **que extiende de** `InscripcionCursoException`
  b) Posee el constructor `CupoNoDisponibleException(int cupo)` cuyo objetivo es setear el mensaje → `"El cupo máximo de "+cupo+" alunos ya a sido alcanzado"`
- `CreditosInsuficientesException` **que extiende de** `InscripcionCursoException`
- Posee el constructor `CreditosInsuficientesException(int requeridos,int obtenidos)` cuyo objetivo es setear el mensaje → `"Se requieren "+requeridos+" creditos y ha obtenido " +obtenidos+" creditos"`

Una vez creadas las excepciones modificar los siguientes métodos

- En la clase Curso, modificar el método inscribir(Alumno a) para que declare que si no hay cupo lance el método la excpeción **throws** `InscripcionCursoException. En la implementación`

del método, cuando no hay cupo, lanzar una nueva instancia de `CupoNoDisponibleException`

- En la clase CursoOpcional, si el alumno no tiene créditos, lanzar una instancia de `CreditosInsuficientesException`
- En la clase CursoObligatorio si el alumno no tiene las correlativas lanzar una instancia de `InscripcionCursoException`