Nombre y Apellido: ..... N° Legajo: .....

Recuperatorio del Segundo Parcial de Programación Orientada a Objetos (72.33)

02/07/2018

Ejercicio 1	*Ejercicio 2*	*Ejercicio 3*	*Nota*	*Firma Docente*
/3	/3	/4		

- **Condición mínima de aprobación: Sumar 5 puntos**
- **Las soluciones que no se ajusten al paradigma OO, no serán aceptadas.**
- **Las soluciones que no se ajusten estrictamente al enunciado, no serán aceptadas.**
- **Puede entregarse en lápiz.**
- **No es necesario escribir las sentencias `import`.**
- **Además de las clases solicitadas se pueden agregar las que consideren necesarias.**
- **Escribir en cada hoja Nombre, Apellido, Legajo, Número de Hoja y Total Hojas entregadas.**

Ejercicio 1

Se desea contar con la clase `ReversedListImpl` para contar con una lista cuyo iterador retorne los elementos en el orden inverso al que se encuentran almacenados. Asimismo, se debe ofrecer un método alternativo para iterar por los elementos en el orden habitual.

Implementar todo lo necesario para que, con el siguiente programa,

```
import java.util.Iterator;

public class ReversedListTester {

   public static void main(String[] args) {
       ReversedList<Integer> list = new ReversedListImpl<>();
       list.add(1);
       list.add(2);
       list.add(3);
       list.add(4);
       Iterator<Integer> defaultIterator = list.defaultIterator();
       while(defaultIterator.hasNext()) {
           System.out.print(defaultIterator.next() + " ");
       }
       System.out.println();
       Iterator<Integer> iterator = list.iterator();
       while(iterator.hasNext()) {
           System.out.print(iterator.next() + " ");
       }
   }

}
```

se obtenga la siguiente salida:

```
1 2 3 4
4 3 2 1
```

Ejercicio 2

Se desea contar con la clase `LocalDateInterval` para definir un intervalo de fechas (desde-hasta) con cierto paso definido en cantidad de días.

Implementar todo lo necesario para que, con el siguiente programa,

```
import java.time.LocalDate;
import java.util.Iterator;

public class LocalDateIntervalTester {
```

```
   public static void main(String[] args) {
       // Ejemplo de uso de la clase LocalDate
       LocalDate today = LocalDate.now();
       System.out.println(today);
       LocalDate todayPlus10 = today.plusDays(10);
       System.out.println(todayPlus10);
       System.out.println(todayPlus10.isAfter(today));
       System.out.println("----------");
       // Ejemplo de uso de la clase LocalDateInterval
       LocalDateInterval localDateInterval = new LocalDateInterval(today, todayPlus10, 3);
       for(LocalDate localDate : localDateInterval) {
           System.out.println(localDate);
       }
       System.out.println("----------");
       Iterator<LocalDate> localDateIterator = new LocalDateInterval(today, today.plusDays(2),
1).iterator();
       while(localDateIterator.hasNext()) {
           System.out.println(localDateIterator.next());
       }
   }

}
```

se obtenga la siguiente salida:

```
2018-06-28
2018-07-08
true
----------
2018-07-01
2018-07-04
2018-07-07
----------
2018-06-29
2018-06-30
```

Ejercicio 3

Se quiere modelar una carrera y sus participantes para luego tener información sobre los resultados. Para esto se define la clase `Race`. Esta clase lleva un registro de todos los participantes de la carrera, permitiendo para cada uno establecer tiempos de partida y de llegada, y además permite imprimir los resultados (lista de todos los participantes con el tiempo total de carrera de cada uno).

```
import java.util.*;

public class Race {

   protected Map<String, Participant> participants = new HashMap<>();

   /** Agrega a un participante a la carrera, especificando el nombre y la edad. */
   public void addParticipant(String name, int age) {
       if (participants.containsKey(name)) {
           throw new IllegalArgumentException("Duplicated participant!");
       }
       participants.put(name, new Participant(name, age));
   }

   /** Registra la hora en la que un participante pasa por la largada. */
   public void registerStartTime(String participant, int time) {
       getParticipant(participant).start(time);
   }

   /** Registra la hora en la que un participante cruza la meta.  */
   public void registerEndTime(String participant, int time) {
       getParticipant(participant).end(time);
   }

   /** Obtiene el tiempo total de carrera del participante. */
   public double getTotalTime(String participant) {
```

```
        return getParticipant(participant).getTotalTime();
    }

    /** Obtiene un participante existente a partir de su nombre.  */
    protected Participant getParticipant(String name) {
        Participant participant = participants.get(name);
        if (participant == null) {
            throw new IllegalArgumentException("Invalid participant name");
        }
        return participant;
    }

    /** Imprime la lista de todos los participantes inscriptos, indicando el nombre
     * y el tiempo neto de cada uno. */
    public void printParticipants() {
        printParticipants(participants.values());
    }

    /** Imprime por consola el nombre y el tiempo neto de un grupo de participantes.  */
    protected void printParticipants(Iterable<Participant> part) {
        for (Participant p: part) {
            System.out.println(p.getName() + " " + (p.hasTime() ? p.getTotalTime() : "--"));
        }
    }

}
```

Esta clase usa como clase auxiliar a la clase `Participant`, que permite definir un participante en base a su nombre y edad. Además permite definir el momento en que comenzó la carrera y el momento en que la finalizó (si es que logró llegar a la meta). Esta clase ofrece un método capaz de calcular la duración total de la carrera para dicho participante.

```
public class Participant {

    private String name;
    private int age;
    private int startTime;
    private int endTime;

    public Participant(String name, int age) {
        this.name = name;
        this.age = age;
    }

    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }

    public void start(int time) {
        startTime = time;
    }

    public void end(int time) {
        endTime = time;
    }

    public boolean hasTime() {
        return startTime != 0 && endTime != 0;
    }

    public int getTotalTime() {
        return endTime - startTime;
    }

}
```

Los horarios de partida y de llegada se almacenan en variables de tipo long que representan unidades de tiempo arbitrarias, la resta entre la llegada y la partida es el tiempo total de carrera.

Se pide implementar la clase `CategoryRace`, que representa una carrera subdividida en categorías por edad no necesariamente excluyentes (puede haber una categoría de 20 a 30 años y a su vez una de 25 a 35 años). Esta clase es capaz de listar a los participantes de cierta categoría ordenados según el tiempo total de carrera de cada uno y también genera un ranking completo (sin discriminar por categoría).

Implementar todo lo necesario para que, con el siguiente programa,

```
public class RankingRaceTest {

   public static void main(String[] args) {
       CategoryRace race = new CategoryRace();
       race.addCategory("Categoria1", 20, 40);
       race.addCategory("Categoria2", 30, 50);
       race.addParticipant("Persona A", 25);  // Etapa inscripción
       race.addParticipant("Persona B", 32);
       race.addParticipant("Persona C", 33);
       race.addParticipant("Persona D", 45);
       race.addParticipant("Persona E", 65);
       race.addParticipant("Persona F", 41);
       race.registerStartTime("Persona A", 1000); // Se larga la carrera
       race.registerStartTime("Persona B", 1110);
       race.registerStartTime("Persona C", 1050);
       race.registerStartTime("Persona D", 1200);
       race.registerStartTime("Persona E", 1000);
       race.registerStartTime("Persona F", 1300);
       race.registerEndTime("Persona A", 2000);   // Comienzan a llegar a la meta
       race.registerEndTime("Persona B", 2600);
       race.registerEndTime("Persona C", 2240);
       race.registerEndTime("Persona D", 3100);
       race.registerEndTime("Persona E", 2100);
       System.out.println("Participantes:");  // Se obtienen resultados
       race.printParticipants();
       System.out.println("Ranking completo:");
       race.printGeneralRanking();
       System.out.println("Categoria 1");
       race.printCategoryRanking("Categoria1");
       System.out.println("Categoria 2");
       race.printCategoryRanking("Categoria2");
   }

}
```

se obtenga la siguiente salida:

```
Participantes:
Persona A 1000
Persona B 1490
Persona C 1190
Persona D 1900
Persona E 1100
Persona F --
Ranking completo:
Persona A 1000
Persona E 1100
Persona C 1190
Persona B 1490
Persona D 1900
Categoria 1
Persona A 1000
Persona C 1190
Persona B 1490
Categoria 2
Persona C 1190
Persona B 1490
Persona D 1900
```