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

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

03/12/2018

Ejercicio 1	*Ejercicio 2*	*Ejercicio 3*	*Nota*	*Firma Docente*
/3	/2	/5		

- **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

La clase `FilteredKeyMapIterator` modela un **iterador de mapas**. Permite iterar sobre una instancia de `Map` (que recibe en su constructor), accediendo a cada una de las claves del mismo **que cumplan un criterio** (que recibe en su constructor) con el método `next()` y al valor asociado a esa clave en el mapa mediante el método `getValue()`.

Aclaración: no importa el orden en que se recorren las claves del mapa.

Para modelar el criterio que deben cumplir las claves, se utiliza la interfaz funcional `Predicate<T>` presente en la biblioteca de Java. El cuerpo relevante de la misma es el siguiente:

```
package java.util.function;

import java.util.Objects;

@FunctionalInterface
public interface Predicate<T> {

   /**
    * Evaluates this predicate on the given argument.
    *
    * @param t the input argument
    * @return {@code true} if the input argument matches the predicate,
    * otherwise {@code false}
    */
   boolean test(T t);

   ...

}
```

Implementar todo lo necesario para que, con el siguiente programa

```
import java.util.HashMap;
import java.util.Map;

public class FilteredKeyMapIteratorTester {

   public static void main(String[] args) {
       Map<Integer, String> map = new HashMap<>();
       map.put(1, "One");
       map.put(2, "Two");
       map.put(3, "Three");
       FilteredKeyMapIterator<Integer, String> mapIterator
                                       = new FilteredKeyMapIterator<>(map, k -> k % 2 == 1);
```

```
        while (mapIterator.hasNext()) {
            System.out.println("Key: " + mapIterator.next());
            System.out.println("Value: " + mapIterator.getValue());
        }
        mapIterator = new FilteredKeyMapIterator<>(map, k -> k != null);
        try {
            mapIterator.getValue();
        } catch (Exception ex) {
            System.out.println(ex.getClass());
        }
        System.out.println(mapIterator.next());
        System.out.println(mapIterator.getValue());
        System.out.println(mapIterator.getValue());
        while (mapIterator.hasNext()) {
            mapIterator.next();
        }
        mapIterator.next();
    }

}
```

se obtenga la siguiente salida:

```
Key: 1
Value: One
Key: 3
Value: Three
class java.util.NoSuchElementException
1
One
One
Exception in thread "main" java.util.NoSuchElementException
...
```

Ejercicio 2

Dada la siguiente jerarquía de clases, con los métodos de instancia indicados para cada una, se cuenta con tres instancias homónimas a la clase a la cual pertenecen.

	m1	m2	m3
A			
B			
C			

```
class A {

 int m1() {
  return this.m3();
 }

 int m2() {
  return 0;
 }

 int m3() {
  return m2();
 }

}
```

```
class B extends A {

 int m1() {
  return super.m1();
 }

 int m2() {
  return m2();
 }

 int m3() {
  return 3;
 }

}
```

```
class C extends B {

 int m2() {
  return 2;
 }

 int m3() {
  return super.m3();
 }

}
```

Completar el cuadro de doble entrada (clase y mensaje) indicando qué se obtiene al enviar cada uno de los mensajes a instancias de cada una de las clases.

Ejercicio 3

Se desea implementar un conjunto de clases que permitan administrar el préstamo de libros de la **biblioteca** de una universidad a sus alumnos y docentes.

Ya se cuenta con la clase `BookInfo` implementada, la cual relaciona el nombre del libro con el stock actual.

```
public class BookInfo {

   private String name;
   private int stock;

   public BookInfo(String name, int stock) {
       this.name = name;
       this.stock = stock;
   }

   public void borrowBook() {
       if (stock == 0) {
           throw new IllegalStateException();
       }
       stock--;
   }

   public void returnBook() {
       stock++;
   }

}
```

Las reglas de la biblioteca son las siguientes:

- **Se puede prestar únicamente un libro por persona.**
- Si es **alumno**, el préstamo del libro es por 2 días.
- Si es **profesor**, el tiempo del préstamo del libro dependerá de su cargo:
  - Responsable: 12 días
  - Jefe de Trabajos Prácticos: 11 días
  - Ayudante: 10 días.

Implementar todo lo necesario y completar los ..... para que, con el siguiente programa de prueba:

```
import java.time.LocalDate;

public class LibraryTester {

   public static void main(String[] args) {
       // Ejemplo de uso para determinar si una fecha está después que otra fecha
       System.out.println(LocalDate.of(2018,12,3).isAfter(LocalDate.of(2018,12,1)));

       Library library = new Library().addBook("Book 1", 1)
               .addBook("Book 2", 1)
               .addBook("Book 3", 3);

       // El estudiante Student 1 pide el libro Book 1

       ..... s1 = .....;
       library.borrowBook(s1, "Book 1", LocalDate.of(2018, 12, 1));
       // El estudiante Student 2 pide el libro Book 3

       ..... s2 = .....;
       library.borrowBook(s2, "Book 3", LocalDate.of(2018, 12, 1));
       // El estudiante Student 3 pide el libro Book 3

       ..... s3 = .....;
       library.borrowBook(s3, "Book 3", LocalDate.of(2018, 12, 2));

       // Lista los préstamos vencidos para la fecha recibida
       library.printDueLoansBooks(LocalDate.of(2018, 12, 4));
       library.returnBook(s1); // El estudiante Student 1 devuelve el libro que pidió
       library.returnBook(s2); // El estudiante Student 2 devuelve el libro que pidió
       library.returnBook(s3); // El estudiante Student 3 devuelve el libro que pidió
```

```
        // El profesor Professor 1 es Responsable y pide el libro Book 1

        ..................... p1 = .....................................;
        library.borrowBook(p1, "Book 1", LocalDate.of(2018, 12, 4));
        // El profesor Professor 2 es Jefe de Trabajos Prácticos y pide el libro Book 3

        ..................... p2 = .....................................;
        library.borrowBook(p2, "Book 3", LocalDate.of(2018, 12, 4));
        // El profesor Professor 3 es Ayudante y pide el libro Book 3

        ..................... p3 = .....................................;
        library.borrowBook(p3, "Book 3", LocalDate.of(2018, 12, 4));

        library.printDueLoansBooks(LocalDate.of(2018, 12, 10));
        library.printDueLoansBooks(LocalDate.of(2018, 12, 15));
        library.printDueLoansBooks(LocalDate.of(2018, 12, 16));
        library.printDueLoansBooks(LocalDate.of(2018, 12, 17));

        try {
            library.borrowBook(s1, "Other Book", LocalDate.of(2018,12,3));
        } catch (Exception ex) {
            System.out.println(ex.getMessage());
        }
    }

}
```

se obtenga la siguiente salida:

```
true
Due loans at 2018-12-04
Student 2 (Book 3 - 2018-12-01)
Student 1 (Book 1 - 2018-12-01)

Due loans at 2018-12-10

Due loans at 2018-12-15
Professor 3 (Book 3 - 2018-12-04)

Due loans at 2018-12-16
Professor 3 (Book 3 - 2018-12-04)
Professor 2 (Book 3 - 2018-12-04)

Due loans at 2018-12-17
Professor 3 (Book 3 - 2018-12-04)
Professor 2 (Book 3 - 2018-12-04)
Professor 1 (Book 1 - 2018-12-04)
Book not found.
```