Prueba 2 de Evaluación Continua
Análisis y Diseño de Software (2016/2017)

Contesta cada apartado en hojas separadas

Apellidos:
Nombre:

Ejercicio 1: (3,5 puntos)

Una aplicación gestiona préstamos bancarios de dos tipos, para comprar viviendas y para consumo general. Un préstamo se caracteriza por el valor prestado y el importe a devolver (expresado mediante n cuotas mensuales). El importe a devolver siempre excederá al valor prestado a consecuencia de los intereses. Los intereses totales se pueden obtener restando al valor prestado el importe total de las cuotas. El coste total del préstamo es la suma de esos intereses y una comisión, que se paga de una sola vez al conceder el préstamo. La comisión varía según el tipo de préstamo y la categoría asociada al cliente. En los préstamos para consumo la comisión es, en principio, el 1% del valor prestado cuando la cuota es de 1500€ o superior, y el 2% si la cuota es inferior. Ahora bien, a los clientes con categoría de nivel superior a EMPLEADO se les elimina el 25% de esas comisiones. Las categorías, ordenadas de menor a mayor nivel, son RECIENTE, VETERANO, EMPLEADO, DIRECTIVO y ACCIONISTA. Los préstamos para vivienda tienen una comisión del 2,5% del valor prestado, salvo que se especifique explícitamente otro porcentaje al constituir el préstamo. Por otro lado, para promocionar los préstamos para vivienda, a los 100 primeros creados se les reducirá automáticamente a la mitad la comisión que les corresponda. Se asumirá que la cuota y el número de cuotas se mantendrá constante durante el préstamo.

Se pide: Codifica las clases necesarias para completar la aplicación descrita, de forma que el siguiente programa produzca la salida de más abajo.

```
public class Ej11 {
  public static void main(String[] args) {
        Prestamo [] pres = {
           new PrestamoConsumo(10000, 1000, 11, Categoria.DIRECTIVO), //10.000 devolver en 11 cuotas de 1.000
           new PrestamoConsumo(20000, 2000, 11, Categoria.EMPLEADO),  //20.000 devolver en 11 cuotas de 2.000
           new PrestamoVivienda(100000, 2000, 120, 0.04),   // tendrá comisión reducida a la mitad
           new PrestamoVivienda(100000, 1000, 120) };       // tendrá comisión reducida a la mitad

        for( Prestamo p : pres)
           System.out.println( p.coste() +" = "+ p.intereses() +" + "+ p.comision()  );

        for( int i = 1; i <= 98; i++ )  new PrestamoVivienda(100_000, 1000, 120);
        // llegamos a 100 préstamos vivienda
        Prestamo p1 = new PrestamoVivienda(100_000, 2000, 120, 0.04);
        // p1 y p2 iguales que arriba pero sin comisión reducida
        Prestamo p2 = new PrestamoVivienda(100_000, 1000, 120);
        System.out.println( p1.comision() );
        System.out.println( p2.comision() );
  }
}
```

Salida esperada:

```
1150.0 = 1000 + 150.0
2200.0 = 2000 + 200.0
142000.0 = 140000 + 2000.0
21250.0 = 20000 + 1250.0
4000.0
2500.0
```

Prueba 2 de Evaluación Continua
Análisis y Diseño de Software (2016/2017)

Contesta cada apartado en hojas separadas

Una posible solución

```
public enum Categoria {RECIENTE, VETERANO, EMPLEADO, DIRECTIVO, ACCIONISTA}

public abstract class Prestamo {
	private final double prestado;
	private final double cuota;
	private final int cuotas;
	private final double pComision;
	public Prestamo(double prestado, double cuota, int cuotas, double pComision) {
		this.prestado = prestado;
		this.cuota = cuota;
		this.cuotas = cuotas;
		this.pComision=pComision;
	}
	public double coste() {
		return intereses()+comision();
	}
	public double intereses() {
		return (cuota*cuotas) - prestado;
	}
	public double comision(){
		return pComision*prestado;
	}
}

public class PrestamoConsumo extends Prestamo {
	public PrestamoConsumo(double prestado, double cuota, int cuotas, Categoria c) {
		super(prestado, cuota, cuotas, calcComision(cuota, c));
	}
	private static double calcComision(double cuota, Categoria c){
		double fCategoria= c.ordinal()>Categoria.EMPLEADO.ordinal() ? 0.75 : 1;
		return fCategoria* (cuota > 1500 ? 0.01: 0.02);
	}
}

public class PrestamoVivienda extends Prestamo {
	private static int creados=0;
	public PrestamoVivienda(double prestado, double cuota, int cuotas, double pComision) {
		super(prestado, cuota, cuotas, (creados<100)?pComision/2:pComision);
		creados++;
	}
	public PrestamoVivienda(double prestado, double cuota, int cuotas) {
		this(prestado, cuota, cuotas, 0.025);
	}
}
```

Prueba 2 de Evaluación Continua
Análisis y Diseño de Software (2016/2017)

Contesta cada apartado en hojas separadas

Ejercicio 2 (3,5 puntos)

Se quiere construir una clase SemiGrupo, que modele el concepto matemático de semigrupo. Un semigrupo es un conjunto con una operación que toma dos elementos del conjunto y produce otro elemento del conjunto. Por simplificar, consideraremos que los semigrupos son conjuntos de enteros y no haremos ninguna suposición adicional sobre la operación(*).

Así pues, la clase SemiGrupo debe tener un constructor que tome como parámetros la operación y una colección de enteros. La operación debe ser conforme a la siguiente interfaz.

```
interface IOperacion {
	Integer operar(Integer a, Integer b);
}
```

Un semigrupo debe tener un método opera, que invoca la operación y lanza una excepción de tipo ElementoNoValido si los parámetros o el resultado no pertenecen al conjunto. Como ves en el código de más abajo, los objetos de tipo SemiGrupo son compatibles con conjuntos de enteros, y la operación SumaModulo implementa la interfaz IOperacionGrupo realizando la suma módulo el número que se le pasa como parámetro en el constructor.

```
public class SemiGrupoEj {
  public static void main(String[] args) {
	SemiGrupo mod4 = new SemiGrupo(new SumaModulo(4), Arrays.asList(0, 1, 2, 3));
	System.out.println("Semigrupo "+mod4);
	Set<Integer> conjuntoSubyacente = mod4;
	System.out.println("Conjunto "+conjuntoSubyacente);
	try {
	   System.out.println(mod4.opera(3, 2));
	   System.out.println(mod4.opera(3, 7)); //Excepción ya que 7 no está en el conjunto
	} catch (ElementoNoValido e) {
	   System.out.println(e);
	}
  }
}
```

Codifica la clase SemiGrupo y el resto de clases y excepciones necesarias, de tal manera que el código anterior produzca la salida de más abajo.

```
Semigrupo [0, 1, 2, 3]
Conjunto [0, 1, 2, 3]
1
Elemento no válido: 7
```

(*) En un semigrupo, la operación además debe ser asociativa, pero no es necesario que tu código lo compruebe.

Prueba 2 de Evaluación Continua
Análisis y Diseño de Software (2016/2017)

Contesta cada apartado en hojas separadas

Una posible solución

```
class ElementoNoValido extends Exception {
	private int num;
	public ElementoNoValido(Integer res) {	this.num = res; }
	public String toString() { return "Elemento no valido: "+this.num;  }
}

class SemiGrupo extends HashSet<Integer> {
	private IOperacion operacion;
	public SemiGrupo(IOperacion op, Collection<Integer> contents) {
		super(contents);
		this.operacion = op;
	}

	public Integer opera(Integer a, Integer b) throws ElementoNoValido {
		if (!this.contains(a)) throw new ElementoNoValido(a);
		if (!this.contains(b)) throw new ElementoNoValido(b);
		Integer res = this.operacion.operar(a, b);
		if (!this.contains(res)) throw new ElementoNoValido(res);
		return res;
	}
}

class SumaModulo implements IOperacion {
	private int modulo;
	public SumaModulo(int mod) { this.modulo = mod; }
	@Override public Integer operar(Integer a, Integer b) {
		return (a+b)%this.modulo;
	}

}
```

Prueba 2 de Evaluación Continua
Análisis y Diseño de Software (2016/2017)

Contesta cada apartado en hojas separadas

Ejercicio 3 (3 puntos)

Se pide: Completa la siguiente clase Elemento así como los espacios dejados en blanco en el método main, con el objetivo de que la ejecución del programa final produzca la salida de más abajo.
Nota: Utiliza el número que se muestra en cada espacio en blanco para indicar el código que falta.

```
import java.util.*;

class Elemento implements Comparable<Elemento> {   // para poder usar TreeSet<Elemento> abajo
   private String nombre,simbolo;
   public Elemento(String simbolo, String nombre) { this.nombre = nombre; this.simbolo = simbolo; }
   @Override public String toString() { return simbolo +":" + nombre; }
```

```
   @Override
        public boolean equals(Object obj) {          // para eliminar los elementos iguales en Set
                if (obj instanceof Elemento){
                        Elemento e = (Elemento) obj;
                        return simbolo.equals(e.simbolo); // son iguales si tienen igual símbolo
                }
                else return false;
        }
```

```
   @Override
   public int hashCode() {          // establece un hashCode compatible con equals por símbolo
                return simbolo.hashCode();
        }
```

```
        @Override
        public int compareTo(Elemento other) {              // implementa la ordenación por símbolo
                return this.simbolo.compareTo(other.simbolo);
        }
}

public class Ej3 {
  public static void main(String[] args) {
        Elemento h = new Elemento( "H", "hidrogeno");
        Elemento s = new Elemento( "S", "azufre");
        Elemento o = new Elemento( "O", "oxigeno");
        Elemento oi = new Elemento( "O", "oxygen");  // notese que oi y o son “iguales” para los Set
        Elemento c = new Elemento( "C", "carbono");

        List<Set<Elemento>> a = Arrays.asList(   // List permite a.get(índice),  Set elimina iguales
                        new HashSet<Elemento>(Arrays.asList( o, h, s, oi)),  // sin orden específico
                        new TreeSet<Elemento>(Arrays.asList( s, h, o, h)),     // ordena por Símbolo
                        new LinkedHashSet<Elemento>(Arrays.asList( o, s, h, s))); // orden de llegada
        Map<Elemento, Set<Elemento>> m = new LinkedHashMap<>();

        m.put(oi, null);         m.put(o, a.get(0));    m.put(h, a.get(1));       m.put(c, a.get(2));

        m.get(h).add( new Elemento("N", "Nitrogeno") );
        m.get(c).add( new Elemento("N", "Nitrogeno") );

        System.out.println(m);
  }
}
```

Salida esperada (con saltos de línea y blancos añadidos por legibilidad):

```
{O:oxygen=[S:azufre, H:hidrogeno, O:oxigeno],
 H:hidrogeno=[H:hidrogeno, N:Nitrogeno, O:oxigeno, S:azufre],
 C:carbono=[O:oxigeno, S:azufre, H:hidrogeno, N:Nitrogeno]}
```