ESTRUCTURAS DE DATOS

TRABAJO PRÁCTICO N ° 1

Conceptos básicos de diseño Orientado a Objetos

Departamento de Ciencias e Ingeniería de la Computación - U.N.S.
Primer cuatrimestre de 2019

Bibliografía:

[GT] Michael Goodrich & Roberto Tamassia. *Data Structures and Algorithms in Java. Fourth Edition*. John Wiley and Sons. 2006.
[W12] Mark A. Weiss. *Data Structures and Algorithm Analysis in Java*. Third Edition. Addison-Wesley Pearson Education, Inc. 2012.

Ejercicio 1:

Estudie la diferencia de uso de los modificadores protected y private en el contexto de herencia de clases considerando el siguiente ejemplo:

```
public class Empleado {
        protected String nombre;
         private int sueldo;

         public Empleado( String nombre, int sueldo )
               {          this.nombre = nombre;
                              this.sueldo = sueldo;
                    }
          public Empleado(int sueldo ) { this.sueldo = sueldo; }
          public String getNombre() { return nombre; }
          public int getSueldo() { return sueldo; }
        }
```

```
public class Medico extends Empleado {
          protected String especialidad;

         public Medico( String nombre, int sueldo, String especialidad )
          {        super( sueldo );
                    this.nombre = "Dr. " + nombre;
                   this.especialidad = especialidad;
                   this.sueldo = this.sueldo * 8;  }

         public String getEspecialidad()  { return especialidad; }
}
```

Determine qué errores posee el código de arriba en términos de acceso a atributos de la clase Empleado desde operaciones de la clase Médico comparando la diferencia entre los modificadores *private* y *protected* . Proponga una forma de solucionar los errores hallados (si los hubiera). Discuta en qué situaciones es conveniente calificar al atributo o *nombre* como *private* en lugar de *protected*.

Ejercicio 2:

Relacionado con el concepto manejo de excepciones:

a) ¿Cómo maneja el lenguaje Java el concepto de *excepción*?
b) En el siguiente fragmento de código, explique por qué se imprimirá el mensaje “Intento de acceso fuera del arreglo”:

```
class C
{ private int [] a;

  public C()
          { a = new int[20]; }

  int acceder( int i )
          { return a[i*8]; }
}
```

```
class Principal
{
 public static void main(String [] args)
 { try { C x = new C();
       int i = 3;
       int j = x.acceder( i );
       System.out.print( “j vale: “ + j );
     } catch( IndexOutOfBoundsException e ) {
                  System.out.println(“Intento de acceso fuera del arreglo”);
                                    }
  }
}
```

c) Implemente un programa en Java que lea dos números por teclado e imprima el cociente entre los mismos. En el caso en que el divisor sea cero, el programa debe capturar una ArithmeticException e imprimir el mensaje de error correspondiente.
d) Implemente una clase llamada FractionException derivada de la clase Exception. En su constructor, esta clase debe recibir una cadena explicando la naturaleza del error producido.
e) Implemente una clase llamada Fraction que representa una fracción como un par de números enteros. La clase Fraction debe tener un constructor con la siguiente signatura:

```
Fraction(int numerador, int denominador)
```

En el caso en que el denominador sea cero, el código del constructor debe lanzar una excepción de tipo FractionException.
f) Implementar una clase ProgramaPrincipal que lea dos números por teclado y cree una fracción utilizando la clase Fraction implementada en (e), y utilizando el primer número leído como numerador y el segundo como denominador. El programa debe ser capaz de capturar una excepción FractionException en el caso en que se lanzara la misma al crear la fracción.

Ejercicio 3:
Implemente un programa Java que cree una clase llamada Pair que pueda almacenar dos objetos declarados como tipos genéricos. Demuestre el uso de la clase Pair creando e imprimiendo distintos objetos conteniendo cinco tipos diferentes de pares, como por ejemplo <Integer, String> y <Float, Long>, entre otros. Cree un objeto conteniendo (23,"Juan") y otro conteniendo (23.2f, 21345).

Ejercicio 4:
Dada una colección de elementos que brinda las siguientes operaciones:

insertar (elemento e) Agrega el elemento *e* a la colección.
eliminar (elemento e) Elimina el elemento *e* de la colección.
cantidadElementos(): entero Retorna la cantidad de elementos almacenados en la colección.
pertenece (elemento e): booleano Retorna verdadero si el elemento *e* está almacenado en la colección y falso en caso contrario.

a) Defina una interfaz *ColecciónEnteros* que represente una colección de números enteros.
b) Implemente la interfaz definida en el inciso anterior con un arreglo de enteros.
c) Implemente la interfaz definida en el inciso anterior con un vector de enteros.
d) Defina una interfaz genérica Colección (utilizando genericidad paramétrica).
e) Implemente la interfaz definida en el inciso anterior con un arreglo.
f) Modifique la interfaz definida en el inciso d) y la implementación de la misma asumiendo que al eliminar un elemento que no se encuentra en la colección, se lanza la excepción "ElementoInválido"

Ejercicio 5:
Proponga una implementación de Colección que no utilice genericidad paramétrica y sí genericidad a través de polimorfismo. Defina la interfaz correspondiente y de una implementación concreta. ¿Qué riesgos plantea esta implementación a la hora de garantizar la homogeneidad de los datos en el conjunto?

Ejercicio 6:
Indique qué hace la siguiente porción de código:

```
int[] arr = {1, 2, 3};
for (int i : arr)
        System.out.println(i);
```

Explique brevemente el concepto de iterador en Java.

Ejercicio 7:
Considere el siguiente programa:

```
public class Foo {
        public static void m( float f1, Float f2 ) {
                System.out.println( "f1 = " + f1 );
                System.out.println( "f2 = " + f2 );
                f1 += 1.0f;
                f2 += 1.0f;
                System.out.println( "f1 = " + f1 );
                System.out.println( "f2 = " + f2 );
        }

        public static void main( String [] args ) {
                float x = 1.0f;
                Float y = 1.0f;
                m( x, y );
                System.out.println( "x = " + x );
                System.out.println( "y = " + y );
        }

}
```

a) Realice una traza y determine qué imprime. Justifique apropiadamente.
b) Vistas las dificultades obtenidas en el inciso anterior, proponga una clase que almacene como parte de su estado un número real que pueda ser modificado.

Nota: Prestar especial atención a la sintaxis, en particular a la diferencia entre "float" y "Float"