* 1. ***TP N°4: Interfaces, Excepciones y Enumerativos***

|  |
| --- |
| La siguiente guía cubre los contenidos vistos en la clase teórica **3. Java Enums, Interfaces** |

* + 1. ***Ejercicio 1***

Indicar si los siguientes fragmentos de código compilan o no compilan, y en los casos que sí compila, indicar la salida obtenida. Justificar.

***1.1***

|  |
| --- |
| public interface A {  boolean even(int value);  } |

|  |
| --- |
| public interface B {  boolean even(int value);  } |

|  |
| --- |
| public class C implements A, B {  @Override  public boolean even(int value) {  return value % 2 == 0;  }  } |

***1.2***

|  |
| --- |
| public interface Greeting {  String initialGreeting();  } |

|  |
| --- |
| public abstract class GreetingImpl implements Greeting {  public String finalGreeting();  } |

***1.3***

|  |
| --- |
| public interface Greeting {  String initialGreeting();  } |

|  |
| --- |
| public class GreetingImpl implements Greeting {  @Override  public String initialGreeting() {  return "Hola";  }  } |

|  |
| --- |
| public class GreetingTester {  public static void main(String[] args) {  Greeting greeting = new GreetingImpl();  if (greeting instanceof Greeting) {  Greeting var = (Greeting) greeting;  System.out.println(var.initialGreeting());  }  if (greeting instanceof GreetingImpl) {  GreetingImpl var = (GreetingImpl) greeting;  System.out.println(var.initialGreeting());  }  }  } |

***1.4***

|  |
| --- |
| public interface A {  int sum(int num1, int num2);  } |

|  |
| --- |
| public interface B extends A {  double sum(double num1, double num2);  } |

|  |
| --- |
| public class C implements B {  @Override  public double sum(double num1, double num2) {  return num1 + num2;  }    } |

* + 1. ***Ejercicio 2***

Se cuenta con una interfaz Function que ofrece un método evaluate con el fin de evaluar una función matemática de una variable real:

|  |
| --- |
| public interface Function {  double evaluate(double x);  } |

De esta forma, se pueden tener implementaciones diversas que correspondan a distintas funciones matemáticas, como se muestra a continuación:

|  |
| --- |
| public class LinearFunction implements Function {  private final double a, b;  public LinearFunction(double a, double b) {  this.a = a;  this.b = b;  }  @Override  public double evaluate(double x) {  return a \* x + b;  }  } |

|  |
| --- |
| public class QuadraticFunction implements Function {  private double a, b, c;  public QuadraticFunction(double a, double b, double c) {  this.a = a;  this.b = b;  this.c = c;  }  @Override  public double evaluate(double x) {  return a \* Math.pow(x,2) + b \* x + c;  }  } |

|  |
| --- |
| public class SineFunction implements Function {  @Override  public double evaluate(double x) {  return Math.sin(x);  }  } |

Realizar otra implementación que incluya al módulo Function que permita representar la **composición de dos funciones**. La clase a implementar debe llamarse **CompositeFunction** y debe recibir en el constructor dos funciones a componer, como se muestra en el siguiente programa de prueba. La salida esperada se indica en los comentarios:

|  |
| --- |
| public class FunctionTester {  public static void main(String[] args) {  Function f1 = new LinearFunction(2, 0); // y = 2x  Function f2 = new QuadraticFunction(1, 0, 0); // y = x^2  Function f3 = new CompositeFunction(f1, f2); // y = (2x)^2  System.out.println(f3.evaluate(1)); // 4.0  System.out.println(f3.evaluate(2)); // 16.0  Function f4 = new SineFunction(); //y = sin(x)  Function f5 = new CompositeFunction(f1, f4); // y = sin(2x)  Function f6 = new CompositeFunction(f5, f1); // y = 2 sin(2x)  System.out.println(f6.evaluate(0)); // 0.0  System.out.println(f6.evaluate(Math.PI / 4.0)); // 2.0  }  } |

Analizar desde el punto de vista del paradigma de la programación orientada a objetos las distintas implementaciones de la interfaz Function. ¿Se podría plantear alguna jerarquía de clases distinta para poder reutilizar código?

* + 1. ***Ejercicio 3***

Se cuenta con la interfaz Movable que cuenta con cuatro métodos pensados para mover objetos geométricos en las cuatro direcciones del plano 2D.

|  |
| --- |
| public interface Movable {  void moveNorth(double delta);  void moveSouth(double delta);  void moveWest(double delta);  void moveEast(double delta);  } |

Implementar todo lo necesario para que las figuras geométricas del ***TP N°3*** puedan moverse en el plano, donde mover una figura en una dirección implica mover en esa dirección a todos los puntos que la definen. Actualizar el diagrama de clases. Realice un programa de prueba que mueva a una elipse.

1. ***Ejercicio 4***

Se desea agregar la posibilidad de mover los objetos geométricos en las ocho direcciones del plano. Esto es, permitir además mover un objeto hacia el noreste, noroeste, sureste y suroeste. Por ejemplo el método moveNorthEast recibirá dos delta (un delta para el eje x y otro delta para el eje y). **Implementar todo lo necesario, actualizar el diagrama de clases y actualizar el programa de prueba.**

***Ejercicio 5***

HTML es un lenguaje utilizado en el desarrollo de páginas web. Permite describir la estructura de un documento, así como darle formato. Para hacer esto, utiliza “etiquetas” que permiten indicar el formato a aplicar. A continuación se describen algunas de estas etiquetas:

* **b**: Permite definir un texto en **negrita**. Ejemplo: <b>hola</b>
* **i**: Permite definir un texto en **cursiva**. Ejemplo: <i>hola</i>
* **a**: Permite definir un **enlace**, agregando un atributo que indica la página a la cual se desea ir al hacer clic en el enlace. Ejemplo: <a href="www.itba.edu.ar">Ir a la página del ITBA</a>

**Estas etiquetas pueden anidarse para combinar distintos formatos**. Por ejemplo, el siguiente texto HTML muestra la cadena de texto “hola” en negrita y en cursiva: <b><i>hola</i></b>. El siguiente código hace lo mismo: <i><b>hola</b></i>.

Se cuenta con una interfaz HTMLText que representa un texto HTML y provee un método para obtener el código fuente.

|  |
| --- |
| public interface HTMLText {  String source();    } |

¿Por qué no se puede incluir en la interfaz un método default con la implementación del método toString()?

Implementar todo lo necesario para que con el siguiente programa de prueba

|  |
| --- |
| public class HTMLTester {  public static void main(String[] args) {  PlainText text = new PlainText("Hola");  HTMLText boldText = new BoldText(text);  HTMLText italicText = new ItalicText(text);  System.out.println(boldText);  System.out.println(italicText);  HTMLText boldItalicText = new BoldText(italicText);  System.out.println(boldItalicText);  text.setText("ITBA");  System.out.println(boldText);  System.out.println(italicText);  System.out.println(boldItalicText);  HTMLText linkText = new LinkText(text, "itba.edu.ar");  HTMLText linkBoldText1 = new LinkText(boldItalicText, "itba.edu.ar");  HTMLText linkBoldText2 = new BoldText(linkText);  System.out.println(linkText);  System.out.println(linkBoldText1);  System.out.println(linkBoldText2);  text.setText("Ejemplo");  System.out.println(linkBoldText1);  System.out.println(linkBoldText2);  }  } |

se obtenga la salida siguiente salida:

|  |
| --- |
| <b>Hola</b>  <i>Hola</i>  <b><i>Hola</i></b>  <b>ITBA</b>  <i>ITBA</i>  <b><i>ITBA</i></b>  <a href:itba.edu.ar>ITBA</a>  <a href:itba.edu.ar><b><i>ITBA</i></b></a>  <b><a href:itba.edu.ar>ITBA</a></b>  <a href:itba.edu.ar><b><i>Ejemplo</i></b></a>  <b><a href:itba.edu.ar>Ejemplo</a></b> |

Realizar el diagrama de clases correspondiente.

* + 1. ***Ejercicio 6***

Implementar la familia de clases que modelan a las expresiones de verdad not, or y and de forma que con el siguiente programa de prueba se obtenga la salida indicada en los comentarios.

|  |
| --- |
| public class ExpressionTester {  public static void main(String[] args) {  SimpleExpression exp1 = new SimpleExpression(true);  SimpleExpression exp2 = new SimpleExpression(false);  Expression exp3 = exp1.not();  Expression exp4 = exp1.or(exp2);  Expression exp5 = exp3.and(exp4);  System.out.println(exp1.evaluate()); // true  System.out.println(exp3.evaluate()); // false  System.out.println(exp4.evaluate()); // true  System.out.println(exp5.evaluate()); // false  exp1.setValue(false);  System.out.println(exp3.evaluate()); // true  System.out.println(exp4.evaluate()); // false  System.out.println(exp5.evaluate()); // false  exp2.setValue(true);  System.out.println(exp5.evaluate()); // true  }  } |

En cuanto a Expression ¿corresponde que sea una clase abstracta o una interfaz? ¿Por qué?

***Ejercicio 7***

Implementar la clase Interval, que permite representar un conjunto de valores en un intervalo dado. Se debe ofrecer como mínimo el siguiente comportamiento:

* + Interval(double start, double end, double increment) Crea un intervalo con todos los valores entre start y end, separados entre sí por una distancia de increment.
  + Interval(double start, double end) Crea un intervalo con todos los valores entre start y end, en donde el incremento es 1.
  + long size() Devuelve la cantidad de elementos que posee el intervalo.
  + double at(long index) Devuelve el elemento que ocupa el lugar 1, 2, etc.
  + long indexOf(double valor) Devuelve el lugar que ocupa el valor en el intervalo (1, 2, etc) o 0 si no pertenece al mismo.
  + boolean includes(double valor) Devuelve true si el valor pertenece al intervalo y false en caso contrario.
  + String toString()
  + boolean equals(Object other)
  + int hashCode()

En los casos en donde los parámetros podrían ser inválidos (por ejemplo si en el método at el índice es inválido, o si en el constructor se especifica un incremento 0) validarlos, y lanzar una excepción de tipo IllegalArgumentException. ¿Es necesario agregar la cláusula throws IllegalArgumentException a estos métodos? ¿Por qué?

Escribir un programa de prueba para testear los métodos de la clase Interval.

* + 1. ***Ejercicio 8***

Añadir a la implementación anterior el método count, que cuente la cantidad de elementos dentro del intervalo que cumplen con una determinada condición especificada por el usuario. La idea consiste en modelar la condición a través de una interfaz con un único método, que se utilizará para evaluar cada elemento del intervalo. Diseñar esta interfaz IntervalCondition e implementar el método count.

Modificar el programa de prueba de forma que la invocación al método count reciba una clase anónima que implemente la interfaz IntervalCondition.

***Ejercicio 9***

Agregar a la implementación del ***Ejercicio 7*** del ***T.P. Nº3*** una excepción no verificada para que cuando se intente hacer una extracción y la cuenta bancaria no tenga fondos, se obtenga un error en lugar de simplemente no hacer la operación.

Para el mismo programa de prueba del ***Ejercicio 7*** del ***T.P. Nº3***:

|  |
| --- |
| public class BankAccountTester {  public static void main(String[] args) {  CheckingAccount myCheckingAccount = new CheckingAccount(1001, 50);  myCheckingAccount.deposit(100.0);  System.out.println(myCheckingAccount);  myCheckingAccount.extract(150.0);  System.out.println(myCheckingAccount);  SavingsAccount mySavingsAccount = new SavingsAccount(1002);  mySavingsAccount.deposit(100.0);  System.out.println(mySavingsAccount);  mySavingsAccount.extract(150.0); // No se realiza por falta de fondos  System.out.println(mySavingsAccount);  }  } |

se debe obtener la siguiente salida:

|  |
| --- |
| Cuenta 1001 con saldo 100,00  Cuenta 1001 con saldo -50,00  Cuenta 1002 con saldo 100,00  Exception in thread "main" java.lang.RuntimeException: No cuenta con los fondos necesarios.  at ar.edu.itba.poo.tp4.bank.BankAccount.extract(BankAccount.java:18)  at ar.edu.itba.poo.tp4.bank.BankAccountTester.main(BankAccountTester.java:15) |

***Ejercicio 10***

Agregar a la implementación del ***Ejercicio 8*** del ***T.P. Nº3*** las excepciones verificadas para los siguientes casos:

* Cuando al agregar un nuevo número amigo, si se superó el límite de amigos del grupo se lance un error en lugar de no agregarlo (TooManyFriendsException)
* Cuando al agregar un nuevo número amigo éste ya existe (AlreadyExistsFriendException)
* Cuando al eliminar un número amigo éste no existe (FriendNotFoundException)

Para el siguiente programa de prueba:

|  |
| --- |
| public class FriendCellPhoneBillTester {  public static void main(String[] args) {  FriendCellPhoneBill my\_bill = new FriendCellPhoneBill("4444-4444", 3);  my\_bill.setCost(98);  try {  my\_bill.addFriend("5555-5555");  my\_bill.addFriend("6666-6666");  } catch (TooManyFriendsException | AlreadyExistsFriendException ex) {  // No ocurre  }  my\_bill.registerCall("5555-5555", 10);  my\_bill.registerCall("6666-5555", 10);  System.out.println(my\_bill.processBill());  try {  my\_bill.addFriend("6666-6666");  } catch (AlreadyExistsFriendException ex){  System.out.println(ex.getMessage());  } catch (TooManyFriendsException ex) {  // No ocurre  }  try {  my\_bill.addFriend("4444-4444");  my\_bill.addFriend("7777-7777");  } catch (TooManyFriendsException ex) {  System.out.println(ex.getMessage());  } catch (AlreadyExistsFriendException e) {  // No ocurre  }  try {  my\_bill.removeFriend("5555-5555");  my\_bill.removeFriend("5555-5555");  } catch (FriendNotFoundException ex) {  System.out.println(ex.getMessage());  }  }  } |

se debe obtener la siguiente salida:

|  |
| --- |
| 9.9  Error for number 6666-6666: Friend already exists  Error for number 7777-7777: Too many friends  Error for number 5555-5555: Friend not found |

¿Existe una similitud entre los mensajes de error? ¿Cómo puede aprovechar eso para el diseño de las clases de excepciones?

* + 1. ***Ejercicio 11***

Implementar la clase Polynomial y realizar el diagrama de clases correspondiente.

Para esta implementación **se deben hacer todas las validaciones necesarias** realizando el manejo de errores con excepciones propias. Implementar todo lo necesario para que el siguiente programa de prueba:

|  |
| --- |
| public class PolynomialTester {  public static void main(String[] args) throws InvalidGradeException, InvalidIndexException {  Polynomial fourthGradePol = new Polynomial(4);  fourthGradePol.set(2, 3.1); // 3.1 \* x^2  fourthGradePol.set(3, 2); // 2 \* x^3 + 3.1 \* x^2  System.out.println(fourthGradePol.eval(2)); // 28.4  System.out.println(new Polynomial(3).eval(5)); // 0  try {  new Polynomial(-4);  } catch (InvalidGradeException e) {  System.out.println(e.getMessage()); // 0  }  fourthGradePol.set(7, 1.5); // 1.5 \* x^7  }  } |

produzca la siguiente salida:

|  |
| --- |
| 28.4  0.0  Grado Inválido  Exception in thread "main" ar.edu.itba.poo.tp5.polynomial.InvalidIndexException: Índice Inválido  at ar.edu.itba.poo.tp5.polynomial.Polynomial.set(Polynomial.java:16)  at ar.edu.itba.poo.tp5.polynomial.PolynomialTester.main(PolynomialTester.java:16) |

Notar la sentencia throws en el *signature* del método main. ¿Qué funcionalidad cumple? ¿Qué cambios debería hacer en el programa de prueba para eliminar esa sentencia e igual poder ejecutar el programa?

* + 1. ***Ejercicio 12***

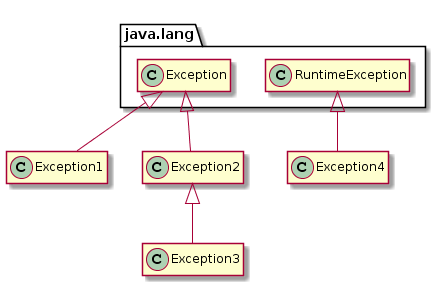
Se cuenta con la siguiente interfaz para representar una lista lineal simplemente encadenada:

|  |
| --- |
| public interface LinearList {  /\*\*  \* Agrega un elemento al final de la lista.  \*/  void add(Object obj);  /\*\*  \* Obtiene el i-ésimo elemento de la lista.  \*/  Object get(int i);  /\*\*  \* Modifica el i-ésimo elemento de la lista colocando un nuevo valor.  \*/  void set(int i, Object obj);  /\*\*  \* Elimina el i-ésimo elemento de la lista.  \*/  void remove(int i);  /\*\*  \* Busca el índice de la primer ocurrencia de un objeto en la lista.  \*/  int indexOf(Object obj);  /\*\*  \* Retorna el tamaño de la lista.  \*/  int size();  } |

Escribir un programa de testeo que use esta interfaz. Luego realizar una implementación de la misma y utilizar dicha implementación en el programa anterior para verificar su correcto funcionamiento.

***Ejercicio 13***

Dada la siguiente jerarquía de excepciones:

[](https://www.plantuml.com/plantuml/img/SoWkIImgAStDuSfAB4lKoqdCIzTphKXEBIZ9pCzJiAdHrLM0Sms9K6F4XM2YIHaZQGygpIl9p4t5POe95rT6PaggGcvgRb9gKJ6NGsfU2j2D0000)

indicar para cada uno de los siguientes programas si compilan o no. En el caso de que compilen, indicar la salida obtenida. En caso contrario explicar el motivo.

|  |
| --- |
| public class Ej1 {  public static void main(String[] args) {  Ej1 ej1 = new Ej1();  try {  ej1.method();  System.out.println("Método ejecutado");  } catch (Exception2 e) {  System.out.println("Excepción 2 capturada");  } finally {  System.out.println("Finalizando");  }  }  public void method() throws Exception3 {  throw new Exception3();  }  } |

|  |
| --- |
| public class Ej2 {  public static void main(String[] args) {  Ej2 ej2 = new Ej2();  try {  try {  ej2.m3();  } catch (Exception3 e) {  System.out.println("Excepción 3 capturada");  } finally {  System.out.println("Finalizando 3");  }  } catch (Exception2 e) {  System.out.println("Excepción 2 capturada");  } finally {  System.out.println("Finalizando 2");  }  try {  ej2.m1();  } catch (Exception4 e) {  System.out.println("Excepción 4 capturada");  }  }  public void m1() {  throw new Exception4();  }  public void m2() throws Exception4 {  throw new Exception4();  }  public void m3() throws Exception2 {  throw new Exception3();  }  } |

|  |
| --- |
| public class Ej3 {  public static void main(String[] args) {  Ej3 ej3 = new Ej3();  try {  ej3.method();  } catch (Exception2 e) {  System.out.println("Excepción 2 capturada");  } catch (Exception3 e) {  System.out.println("Excepción 3 capturada");  }  }  public void method() throws Exception3 {  throw new Exception3();  }  } |

|  |
| --- |
| public class Ej4 {    public static void main(String[] args) {  throw new Exception2();  }    } |

|  |
| --- |
| public class Ej5 {    public static void main(String[] args) {  throw new Exception4();  }    } |

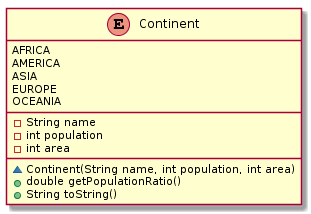
|  |
| --- |
| public class Ej6 {  public static void main(String[] args) {  try {  System.out.println("Dentro del bloque try");  } catch (Exception4 e) {  System.out.println("Dentro del bloque catch");  }  }  } |

|  |
| --- |
| public class Ej7 {    public static void main(String[] args) {  try {  System.out.println("Dentro del bloque try");  } catch (Exception2 e) {  System.out.println("Dentro del bloque catch");  }  }    } |

|  |
| --- |
| public class Ej8 {  public static void main(String[] args) {  Ej8 ej8 = new Ej8();  try {  try {  ej8.method();  } catch (Exception3 e) {  ej8.method();  System.out.println("Excepción 3 capturada");  } finally {  System.out.println("Finalizando 3");  }  } catch (Exception2 e) {  System.out.println("Excepción 2 capturada");  } finally {  System.out.println("Finalizando 2");  }  }  public void method() throws Exception2 {  throw new Exception3();  }  } |

***Ejercicio 14***

Se cuenta con el siguiente enumerativo Continent que modela a los continentes:

[](http://www.plantuml.com/plantuml/img/NP312eCm38RlVOeSbRbtK4e77ZP5sLaw35BGLBhqDBPddxDEtN98Fdxo_uIaTzQE_T0B93-0jCI6a1WU0gRAJbKkiu1djN6T1r3NgYZLZ8LKsMMHcwQPvn5gTeOw83tWefWfVBIZxpKRIpjHEzG_xjTsJhJBIVuYajKTpxO3jDRVUeGEkLotga-FlXjB7Di0KGpYALAaTlh46m00)

|  |
| --- |
| public enum Continent {  AFRICA("África", 1100, 30),  AMERICA ("América", 990, 42),  ASIA("Asia", 4400, 43),  EUROPE("Europa", 730, 10),  OCEANIA("Oceanía", 39, 9);  private String name;  private int population;  private int area;  Continent(String name, int population, int area) {  this.name = name;  this.population = population;  this.area = area;  }  public double getPopulationRatio() {  return (double) population / area;  }  @Override  public String toString() {  return name;  }  } |

Investigar en la documentación de Java los métodos de clase de los enumerativos y completar el siguiente programa de prueba

|  |
| --- |
| public class ContinentTester {  public static void main(String[] args) {  System.out.println("Densidades de población:");  for(Continent continent : Continent.............) {  System.out.println(String.format("%s = %.2f", continent, continent.getPopulationRatio()));  }  System.out.printf("%.2f", Continent..........("AMERICA").getPopulationRatio());  }  } |

para que imprima la siguiente salida

|  |
| --- |
| Densidades de población:  África = 36.67  América = 23.57  Asia = 102.33  Europa = 73.00  Oceanía = 4.33  23,57 |

* + 1. ***Ejercicio 15***

Implementar el enumerativo BasicOperation que modele las cuatro operaciones básicas: suma, resta, multiplicación y división.

Implementar además el enumerativo ExtendedOperation que modele las operaciones de potencia y módulo.

¿Cómo se puede modelar el comportamiento en común que tienen ambos enumerativos?

El siguiente programa de prueba:

|  |
| --- |
| public class OperationTester {  public static void main(String[] args) {  double x = 4;  double y = 2;  for(BasicOperation operation : BasicOperation.values()) {  System.out.printf("%.2f %s %.2f = %.2f\n", x, operation, y, operation.apply(x,y));  }  for(ExtendedOperation operation : ExtendedOperation.values()) {  System.out.printf("%.2f %s %.2f = %.2f\n", x, operation, y, operation.apply(x,y));  }  }  } |

imprime la siguiente salida:

|  |
| --- |
| 4,00 + 2,00 = 6,00  4,00 - 2,00 = 2,00  4,00 \* 2,00 = 8,00  4,00 / 2,00 = 2,00  4,00 ^ 2,00 = 16,00  4,00 % 2,00 = 0,00 |