TP N°5: Introducción a POO en Java

La siguiente guía cubre los contenidos vistos en las clases teórica **5. Introducción a Java**

Ejercicio 1

Evalúe el siguiente programa. ¿Qué salida se obtiene? ¿Por qué?

Son distintos, y es porque comprueba si son la misma instancia

```
public class Ej1 {
   public static void main(String[] args) {
       String s1, s2;
       s1 = new String("hola");
       s2 = new String("hola");
       if (s1 == s2) {
           System.out.println("Son iguales");
       } else {
           System.out.println("Son distintos");
       }
   }
}
```

Modifique el código anterior para obtener el mensaje "Son iguales" de dos formas distintas:

a) Cambiando únicamente la generación de `s1` y `s2`

b) Cambiando únicamente el método de comparación.

Ejercicio 2

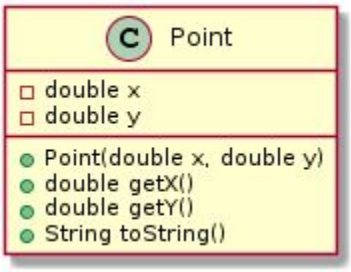
Ejecutar el siguiente código. Explicar qué hace el operador (no es un mensaje) `instanceof` y cuál es la diferencia con el método de instancia `getClass()`. ¿Cuáles son los mensajes de Ruby que cumplen el mismo rol?

instanceof chequea herencias, mientras que el metodo getClass sirve solamente para la clase perteneciente a la instancia.
en ruby, instance_of es para la instancia sola y is_a para herencia

```
public class Ej2 {
   public static void main(String args[]) {
       String s = "hola";
       System.out.println(s instanceof String);
       System.out.println(s instanceof Object);
       System.out.println(s.getClass().equals(String.class));
       System.out.println(s.getClass().equals(Object.class));
   }
}
```

Ejercicio 3

Se cuenta con la clase `Point` que modela un punto en dos dimensiones. A continuación se muestra el diagrama de clases y la implementación correspondiente.




```java
public class Point {
   private double x;
   private double y;

   public Point(double x, double y) {
       this.x = x;
       this.y = y;
   }

   public double getX() {
       return x;
   }

   public double getY() {
       return y;
   }

   @Override
   public String toString() {
       return String.format("{%g,%g}", x, y);
   }
}
```

Indicar qué se obtiene al ejecutar cada uno de los siguientes fragmentos de código (primero resolverlo en papel y luego verificarlo en máquina).

3.1

```java
Point p1 = new Point(2, 1);
Point p2 = new Point(2, 1);
System.out.println(p1 == p2);
```

False ✓

3.2

```java
Point p1 = new Point(2, 1);
Point p2 = new Point(2, 1);
System.out.println(p1.equals(p2));
```

False ✓

3.3

```java
Point p1 = new Point(2, 1);
Point p2 = new Point(2, 1);
System.out.println(p1.getX().equals(p2.getX()) && p1.getY().equals(p2.getY()));
```

~~F~~ Error

3.4

```
Point p = new Point(2, 1);
System.out.println(p instanceof Object);
```

3.5

```
Point p = new Point();
System.out.println(p);
```

3.6

```
Point[] points = new Point[10];
for (Point point : points) {
   System.out.println(point.getX() + ", " + point.getY());
}
```

3.7

```
Point p = new Point(2, 1);
System.out.println(p);
```

Sobreescribir los métodos `equals` y `hashCode` de la clase `Point` para que el ejemplo ***3.2*** imprima **`true`**.

Ejercicio 4

Implementar en Java la clase `ComplexNumber` resuelta en el ***Ejercicio 3*** del ***TP Nº1*** y realizar el diagrama de clases correspondiente.

Dado que en Java no se puede sobreescribir el método **`+`**, proponer una solución para que se pueda obtener una instancia de la clase Complex luego de realizar dichas operaciones. Permitir operar con números que no sean necesariamente complejos (instancias de la clase `Number`) y construir un complejo sin indicar su parte imaginaria (es decir, con parte imaginaria nula).

Reescribir el programa de prueba.

Ejercicio 5

Implementar en Java las clases `BankAccount`, `CheckingAccount` y `SavingsAccount` resueltas en el ***Ejercicio 9*** del ***TP Nº1*** y realizar el diagrama de clases correspondiente.

Para el siguiente programa de prueba

```
package ar.edu.itba.poo.tp5;

import ar.edu.itba.poo.tp5.bank.CheckingAccount;
import ar.edu.itba.poo.tp5.bank.SavingsAccount;

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);
    }

}
```

se obtiene 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.IllegalArgumentException: No cuenta con los
fondos necesarios
       at ar.edu.itba.poo.tp5.bank.BankAccount.extract(BankAccount.java:19)
       at ar.edu.itba.poo.tp5.BankAccountTester.main(BankAccountTester.java:18)
```

Luego de implementar, responda:

a) ¿Fue necesario agregarle un constructor a la clase `SavingsAccount` o se "hereda" el de `BankAccount`?

b) ¿Se puede almacenar una instancia de la clase `SavingsAccount` en una variable de tipo `BankAccount`? Indicar si el siguiente código funciona correctamente o produce un error en compilación o ejecución.

```
BankAccount myBankAccount = new SavingsAccount(1002);
```

c) ¿Se puede almacenar una instancia de la clase `BankAccount` en una variable de tipo `SavingsAccount`? Indicar si el siguiente código funciona correctamente o produce un error en compilación o ejecución.

```
SavingsAccount myBankAccount = new BankAccount(1002);
```

d) Indicar la salida del siguiente código. Explicar.

```
BankAccount myBankAccount = new SavingsAccount(1002);
System.out.println(myBankAccount.getClass());
```

e) Si se agrega el método `getOverdraft()` en la clase `CheckingAccount` para que el usuario pueda conocer cuál es el límite de giro por descubierto de una cuenta corriente:

```
package ar.edu.itba.poo.tp5.bank;

public class CheckingAccount extends BankAccount {

   private double overdraft;

   ...
```

```
    public double getOverdraft() {
        return overdraft;
    }

}
```

¿Por qué falla en compilación el siguiente fragmento de código? ¿Por qué no se puede enviar el mensaje `getOverdraft()` a la variable `myBankAccount`, si ésta contiene una instancia de la clase `CheckingAccount`?

```
BankAccount myBankAccount = new CheckingAccount(1002, 50);
System.out.println(myBankAccount.getOverdraft());
```

f) Modificar el código anterior para que compile y funcione correctamente, sin cambiar el tipo de la variable `myBankAccount` en su declaración.

Ejercicio 6

Se cuenta con las siguientes clases:

```
package ar.edu.itba.poo.tp5.ej6;

public class ClassA {

    public void method(Number n) {
        System.out.println("ClassA: " + n + " " + n.getClass());
    }

}
```

```
package ar.edu.itba.poo.tp5.ej6;

public class ClassB extends ClassA {

    public void method(Integer d) {
        System.out.println("ClassB: " + d + " " + d.getClass());
    }

}
```

Indicar para cada uno de los siguientes fragmentos de código si funcionan correctamente o si generan un error en compilación o en ejecución. En el caso de que funcionen, indicar la salida obtenida.

6.1

```
ClassA a = new ClassB();
a.method(3);
```

ClassA 3 java.lang.Integer

6.2

```
ClassB b = new ClassA();
b.method(3);
```

COMPILACION: Tipos incompatibles

6.3

```
ClassB b = new ClassB();
b.method(3);
```

ClassB 3 class java.lang.Integer

6.4

```
ClassB b = new ClassB();
b.method((Number)3);
```

ClassA 3 java.lang.Integer

6.5

```
ClassA a = new ClassA();
ClassB b = (ClassB)a;
b.method(3);
```

EJECUCION: ClassA no puede ser casteado a ClassB

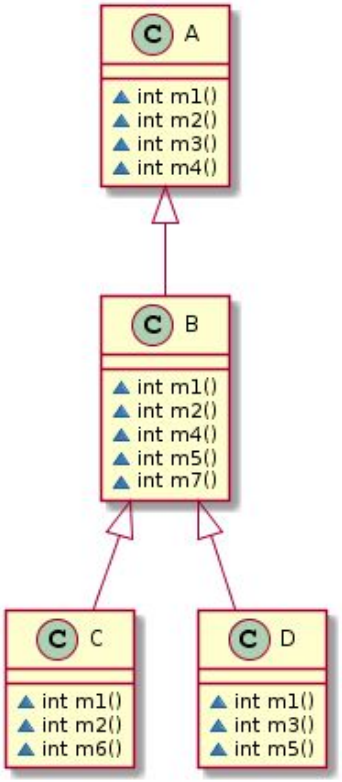
6.6

```
ClassB b = new ClassB();
ClassA a = (ClassA)b;
a.method(3);
```

ClassA 3 class java.lang.Integer

Ejercicio 7

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



```java
class A {

    int m1() {
        return m3();
    }

    int m2() {
        return 10;
    }

    int m3() {
        return 5;
    }

    int m4() {
        return m4();
    }

}
```

```java
class B extends A {

    int m1() {
        return 8;
    }

    int m2() {
        return super.m1();
    }

    int m4() {
        return 20;
    }

    int m5() {
        return m3();
    }

    int m7() {
        return m4();
    }

}
```

```java
class C extends B {

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

    int m2() {
        return m6();
    }

    int m6() {
        return 3;
    }

}

class D extends B {

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

    int m3() {
        return m4();
    }

    int m5() {
        return 2;
    }

}
```

Completar el siguiente cuadro indicando qué se obtiene al enviar los mensajes de la primera fila (por separado, no en cascada) a la instancia de la primera columna. Primero hacerlo en papel para verificar que se entendió la invocación a métodos y el uso de las palabra reservada `super`. Luego verificarlo en el entorno Java.

	m1	m2	m3	m4	m5	m6	m7
A	5 ✓	10 ✓	5 ✓	loop ✓	NO METH ✓	NO METH ✓	NO METH ✓
B	8 ✓	5 ✓	5 ✓	20 ✓	5 ✓	NO METH ✓	20 ✓
C	8 ✓	3 ✓	5 ✓	20 ✓	5 ✓	3 ✓	20 ✓
D	5 ✓	26 ✓	20 ✓	20 ✓	2 ✓	NO METH ✓	20 ✓

Ejercicio 8

Dadas las siguientes clases:

```
package ar.edu.itba.poo.tp5.print;

public class A {
   static final String MESSAGE = "Clase %s: Imprime %s %s que es de tipo %s";

   public void print(Object obj) {
       System.out.println(String.format(MESSAGE,"A","Object",obj,obj.getClass()));
   }

   public void print(Number num) {
       System.out.println(String.format(MESSAGE,"A","Number",num,num.getClass()));
   }

   public void print(Integer num) {
       System.out.println(String.format(MESSAGE,"A","Integer",num,num.getClass()));
   }
}
```

```
public class B extends A {
   public void print(Number num) {
       System.out.println(String.format(MESSAGE,"B","Number",num,num.getClass()));
   }
}
```

indicar cuál es la salida obtenida por el siguiente programa de prueba. Justificar el resultado obtenido para cada una de las invocaciones al método `print`.

```
public class PrintTester {

   public static void main(String[] args) {
       A a = new A();
       a.print(3); Clase A: Imprime Integer 3 que es de tipo java.lang.Integer
       a.print(3.14);Clase A: Imprime Number 3.14 que es de tipo java.lang.Double
       a.print((Number)3);Clase A: Imprime Number 3 que es de tipo java.lang.Integer
       a.print((Object)3);Clase A: Imprime Object 3 que es de tipo java.lang.Integer

       A b1 = new B();
       b1.print(3.14); Clase A: Imprime Number 3.14 que es de tipo java.lang.Double
       b1.print("Hola");Clase A: Imprime Object Hola que es de tipo java.lang.String
       b1.print((Number)3);Clase B: Imprime Number 3 que es de tipo java.lang.Integer
       b1.print((Object)3); Clase A: Imprime Object 3 que es de tipo java.lang.Integer

       B b2 = new B();
       b2.print(3.14); Clase B: Imprime Number 3.14 que es de tipo java.lang.Double
       b2.print("Hola");Clase A: Imprime Object Hola que es de tipo java.lang.String
       b2.print((Number)3);Clase B: Imprime Number 3 que es de tipo java.lang.Integer
       b2.print((Object)3); Clase A: Imprime Object 3 que es de tipo java.lang.Integer
   }

}
```

Ejercicio 9

Implementar en Java la clase `Polynomial` resuelta en el ***Ejercicio 7*** del ***TP Nº3*** y realizar el diagrama de clases correspondiente.

Al igual que en el ejercicio original, 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);
       fourthGradePol.set(3, 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);
   }

}
```

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?

Ejercicio 10

Implementar en Java la clase `CellPhoneBill` y las demás clases asociadas, resueltas en el ***Ejercicio 8*** del ***TP Nº3*** y realizar el diagrama de clases.

Implementar en Java el programa de prueba utilizado en el ejercicio original.