



Universidad Rey Juan Carlos
Departamento de Lenguajes y Sistemas Informáticos I
10 de Septiembre de 2009

Duración: 2 horas.

Lea atentamente todo el enunciado de cada pregunta antes contestar.

Especifique nombre y apellidos en todas las hojas que entregue.

Cualquier suposición o decisión sobre el enunciado del examen debe ser detallada y justificada convenientemente.

Apellidos y Nombre:

Número de expediente:

PROGRAMACIÓN ORIENTADA A OBJETOS – SEPTIEMBRE 2009

1. ¿Qué es un *stream* en Java y qué tipos de *streams* hay? Justifique razonadamente su respuesta [0,5 puntos]

Solución:

Un *stream* en Java representa un flujo de información. Hay dos tipos de flujos: de entrada cuando el flujo de información procede de una determinada fuente (p.e. teclado, fichero, memoria, red, ...) o de salida si la información se dirige hacia un destino (pantalla, fichero, ...). Los *streams* en Java se encuentran definidos en el paquete *java.io*.

2. Justifique razonadamente las ventajas y desventajas de utilizar AWT frente a Swing [0,5 puntos]

Solución:

AWT es más antigua y contiene más funcionalidades. Las aplicaciones desarrolladas con AWT pueden visualizarse correctamente en los navegadores web; sin embargo las que están desarrolladas con componentes Swing no están soportadas directamente por los navegadores Web, requiere la instalación en el cliente de JRE. Por otro lado, las aplicaciones que utilizan Swing tienen mejor estética (p.e. permite botones redondos, los botones y etiquetas pueden mostrar tanto imágenes como texto, etc). Además, Swing tiene una API de accesibilidad y consta con generación de código automática en distintos IDEs.

3. ¿Para qué sirven los diagramas de interacción y qué elementos contienen? Justifique razonadamente su respuesta [0,5 puntos]

Solución:

Los diagramas de interacción sirven para modelar los aspectos dinámicos que ocurren dentro de un sistema, es decir, las colaboraciones que se producen entre los objetos. Los elementos de estos diagramas son los objetos que colaboran, las relaciones que existen entre ellos y que soportan el tanto el envío como la recepción de mensajes y los propios mensajes que se envían entre objetos.

4. ¿Qué es un paquete? ¿Para qué sirve? ¿Cómo se crean e importan paquetes en Java? Justifique razonadamente su respuesta [0,5 puntos]

Solución:

Los paquetes sirven para agrupar clases que se encuentran relacionadas entre sí. La propia API de Java se organiza en paquetes. Un paquete es equivalente a un directorio de archivos donde se incluye el código de las clases que pertenecen a ese paquete. Para definir que una clase pertenece a un paquete se incluye una línea al inicio de su código indicando el nombre del paquete al que pertenece (`package nombrePaquete`). El nombre de un paquete puede constar de varios nombres unidos por puntos (p.e. `java.awt.event`), que se corresponde con la jerarquía de directorios donde se guarda la/s clase/s. Si se desea importar un determinado paquete para utilizar alguna de sus clases o todo el paquete es necesario incluir una línea al principio del fichero que empieza por la palabra reservada `import` seguida del nombre del paquete a importar (p.e. `import java.io.*`) o de la clase a importar (incluyendo su ruta completa, p.e. `import paquete.usuario.aplicacion.class`).

5. Se desea realizar un programa que permita gestionar pares de objetos de las clases `Libro` y `Contacto`. El objetivo de este programa es permitir la creación de pares de objetos ya sean de la clase `Libro` o de la clase `Contacto` y poder realizar comparaciones de elementos de tal manera que dado un par de elementos se obtenga cuál es el elemento mínimo.

Dadas las definiciones parciales de las clases `Libro` y `Contacto`, se quiere realizar un programa que permita comparar dos objetos `Libro` o dos objetos `Contacto`. Para cada contacto, se almacenará su nombre y número de teléfono; mientras que para cada libro se almacenará información sobre el autor, el título y el número de páginas. Parte del código de las clases `Contacto` y `Libro` se muestra a continuación:

```
public class Contacto implements Comparable{
    private String nombre;
    private long telefono;

    public Contacto(String n, long t){ nombre = n; telefono = t; }
    public String getNombre(){ return nombre;}
    public long getTelefono(){ return telefono;}
    public int compareTo(Object o) {...}
}
```

```
public class Libro implements Comparable {
    private String titulo;
    private String autor;
    private int anyo;

    public Libro( String t, String a, int y){
        titulo = t;
        autor = a;
        anyo = y;
    }
}
```

```
public int getAnyo(){ return anyo;}
public String getTitulo(){ return titulo;}
public String getAutor(){ return autor;}
public int compareTo(Object o) {...}
}
```

Se pide:

a) **[1,5 puntos]** Implementar la clase `Par` que permite comparar dos objetos de una determinada clase. La clase `Par` contendrá:

- Dos atributos de un determinado tipo que servirán para almacenar los elementos que conforman el par de objetos.
- Un constructor que se encargue de crear un objeto de la clase `Par` recibiendo como parámetros los dos elementos del par.
- Un método `minimo` que compare los dos elementos del par y devuelva el mínimo. En el caso de objetos de la clase `Libro`, el mínimo será aquel cuyo año de publicación sea inferior. En el caso de que el año coincida, el mínimo será el que tenga el autor menor alfabéticamente hablando. Si ambos atributos coinciden, ambos libros se consideran iguales. Para objetos de la clase `Contacto`, el mínimo será aquel que tenga un nombre menor (en orden alfabético). En el caso de que coincidan, se comparará el teléfono.

Observación: Es obligatorio el uso de genéricos.

b) **[0,5 puntos]** Implementar los métodos `compareTo` de la interfaz `Comparable` necesarios para realizar la comparación de objetos de la clase `Libro` y de la clase `Contacto`.

Solución:

```
public class Par<T extends Comparable> {
    private T e1;
    private T e2;
    public Par (T a, T b){ e1 = a; e2 = b;}
    public T minimo(){
        if (e1.compareTo(e2) > 0) return e1;
        return e2;
    }
}

public class Libro implements Comparable {
    private String titulo;
    private String autor;
    private int anyo;
    public Libro( String t, String a, int y){
        titulo = t;
        autor = a;
        anyo = y;
    }
}
```

```
public int getAnyo(){ return anyo;}
public String getTitulo(){ return titulo;}
public String getAutor(){ return autor;}

public int compareTo(Object o) {
    Libro l = (Libro) o;
    if (getAnyo() == l.getAnyo())
        return -(autor.compareTo(l.getAutor()));
    else if (getAnyo() < l.getAnyo()) return 1;
    return -1;
}
}

public class Contacto implements Comparable{
    private String nombre;
    private long telefono;

    public Contacto(String n, long t){ nombre = n; telefono = t; }
    public String getNombre(){ return nombre;}
    public long getTelefono(){ return telefono;}

    public int compareTo(Object o) {
        Contacto c = (Contacto) o;
        int aux = nombre.compareTo(c.getNombre());
        if (aux == 0){
            if (telefono < c.getTelefono()) return 1;
            if (telefono == c.getTelefono()) return 0;
            return -1;
        }
        return aux;
    }
}
```

6. Se desea realizar un programa que permita realizar operaciones con dos clases de números compuestos: los números racionales y los números complejos. Por un lado, un número racional consta de numerador y denominador, ambos de tipo entero. Por otro lado los números complejos están definidos por una parte real y otra imaginaria, también enteras.

Las operaciones a realizar tanto con números racionales como con números complejos serán:

- Creación de números con y sin argumentos
- Suma de dos números
- Multiplicación de dos números
- Devolución del número como cadena de caracteres, pudiendo pasar opcionalmente como argumento el carácter de separación a utilizar.

- Métodos get y set asociados a los atributos

En el futuro, será interesante que se puedan crear otro tipo de números compuestos, con estas mismas operaciones.

En el caso particular de los números racionales es importante tener en cuenta los siguientes aspectos:

- Un número racional se representará como: `numerador / denominador` (p.e. 3 / 5)
- Se pueden crear números racionales de varias formas: i) sin parámetros de tal manera que el número racional resultante sea 0 / 1; ii) suministrando el numerador (en este caso, el denominador será 1) o iii) facilitando tanto el numerador como el denominador del número racional.
- La suma de dos números racionales se realiza de la siguiente forma:

$$a / b + c / d = (a*d + b*c) / (b*d)$$
- La multiplicación de dos números racionales se realiza multiplicando sus numeradores y denominadores respectivamente.
- Debe implementarse un método que permita simplificar un número racional. Para ello, se hará uso de un método previamente definido en la propia clase que devuelve el máximo común divisor de dos números. **La implementación de este último método, llamado `mcd`, NO es necesario realizarla.**
- Por último, se debe implementar un método que devuelva el número racional como una cadena de caracteres utilizando el carácter '/' como carácter de separación entre numerador y denominador.

En el caso particular de los números complejos, los aspectos a considerar son los siguientes:

- Un número complejo se representará como: `parte_real + parte_imaginaria i` (p.e. 4 + 3i)
- Se pueden crear números complejos de tres formas: i) sin parámetros de tal manera que el número complejo resultante sea 0 + 0i; ii) suministrando la parte real (la parte imaginaria será 0) o iii) facilitando tanto la parte real como la parte imaginaria.
- La suma de dos números complejos se realiza de la siguiente forma:

$$(a + bi) + (c + di) = (a + c) + (b + d)i$$
- La multiplicación de dos números complejos se realiza de la siguiente forma:

$$(a + bi) * (c + di) = (a*c - b*d) + (a*d + b*c) i$$
- Debe implementarse un método que permita obtener el conjugado de un número complejo. El conjugado se obtiene cambiando de signo a la parte imaginaria. Por ejemplo, el conjugado del número complejo (2 + 9i) es (2 -9i).
- Por último, se debe tener en cuenta, que en el caso de números complejos, el carácter de separación es el símbolo '+' en el método que devuelve el número complejo como una cadena de caracteres.

Para hacerse una idea más concisa de las operaciones que debe permitir el código a implementar, a continuación se muestra el código del programa principal:

```
public static void main(String[] args) {
    NumeroRacional nr1 = new NumeroRacional(5,10);
    NumeroRacional nr2 = new NumeroRacional(2);
    NumeroRacional nr3 = new NumeroRacional(3,4);
    NumeroCompuesto res;

    System.out.println("NR1: " + nr1.toString('/'));
    res = nr2.suma(nr3);
    System.out.println("NR2 + NR3: " + res.toString('/'));
    res = nr2.multiplicacion(nr3);
    System.out.println("NR2 * NR3: " + res.toString('/'));
    System.out.println("\n-----");

    NumeroComplejo nc1 = new NumeroComplejo();
    NumeroComplejo nc2 = new NumeroComplejo(2);
    NumeroComplejo nc3 = new NumeroComplejo(3,4);

    System.out.println("NC1: " + nc1.toString('+') + 'i');
    res = nc3.getConjugado();
    System.out.println("Conjugado NC3:" +res.toString('+') + 'i');
    res = nc2.suma(nc3);
    System.out.println("NC2 + NC3: " + res.toString('+') + 'i');
    res = nc2.multiplicacion(nc3);
    System.out.println("NC2 * NC3: " + res.toString('+') + 'i');
}
```

La salida de este programa principal se puede ver en la siguiente figura:

```
NR1: 1 / 2
NR2: 2 / 1
NR3: 3 / 4
NR2 + NR3: 11 / 4
NR2 * NR3: 3 / 2

-----
NC1: 0 + 0i
NC2: 2 + 0i
NC3: 3 + 4i
Conjugado NC3: 3 + -4i
NC2 + NC3: 5 + 4i
NC2 * NC3: 6 + 8i
```

Figura 1. Salida del programa principal – Ejercicio 6

Se pide :

- a) **[2,5 puntos]** Realizar el diseño del diagrama de clases para este programa, indicando cada una de las clases, atributos para cada una de ellas, métodos, relaciones, multiplicidades (si las hubiera), etc.

Solución1:

NumeroCompuesto es una clase que contiene un par de números y los métodos que realizan las operaciones comunes tanto de números racionales como de números compuestos. Las clases NumeroRacional y NumeroComplejo heredan de NumeroCompuesto e implementan los métodos definidos en la interfaz Operaciones. Dado que en el enunciado no se especifica si podría tener sentido crear objetos de la clase NumeroCompuesto existen dos posibles soluciones de diseño.

La primera es entender que sí se podrían crear objetos de la clase NumeroCompuesto entendiendo que podrían ser útiles estos objetos para reflejar objetos compuestos por un par de números. En este caso, la clase NumeroCompuesto tendría dos constructores que permitirían crear objetos de esta clase y que las clases heredadas podrían utilizar para crear objetos con `super(..)`. En este caso, tiene sentido definir una interfaz llamada Operaciones que tenga los métodos para realizar la suma y la multiplicación de dos números compuestos. De esta manera, las clases NumeroRacional y NumeroComplejo implementarían los métodos de esta interfaz aparte de heredar los métodos definidos en la clase padre NumeroCompuesto.

El diagrama de clases para esta solución se muestra a continuación en la figura 2.

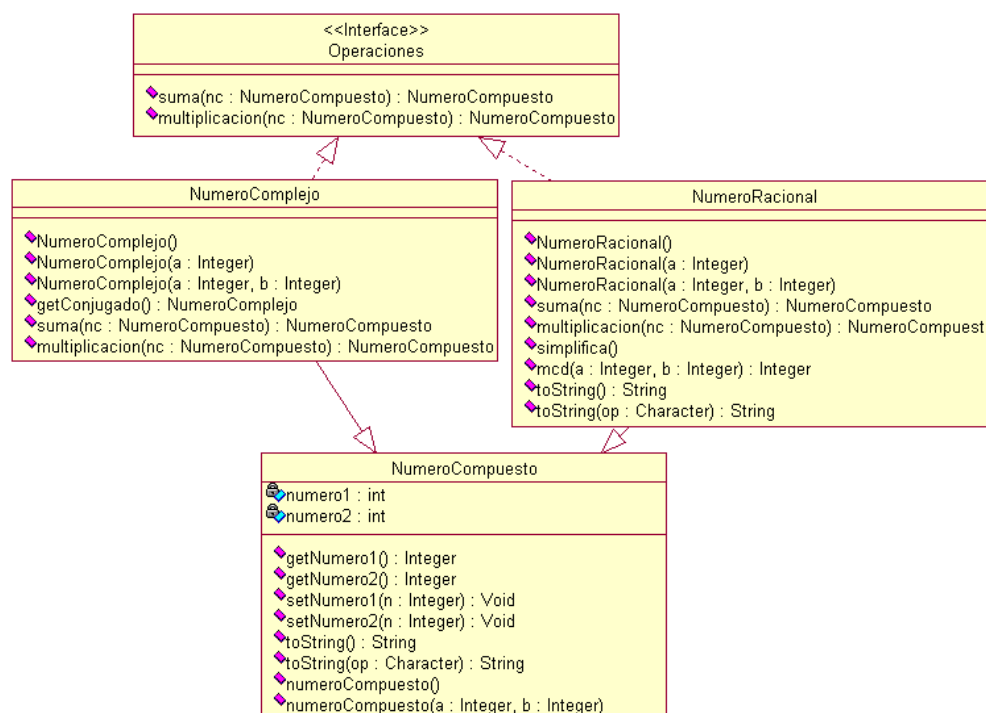


Figura 2. Diagrama de clases – Ejercicio 6 – Solución 1

Solución2:

La segunda interpretación sería pensar que la entidad NumeroCompuesto debería ser una clase abstracta y que no tendría sentido crear objetos de esta clase. Partiendo de esta

suposición, la clase `NumeroCompuesto` tendría los métodos comunes a las clases `NumeroRacional` y `NumeroComplejo` y además la definición de dos métodos abstractos para las operaciones de suma y multiplicación (definidos en la solución previa dentro de una interfaz). El diagrama de clases de esta solución sería más simple ya que habría solamente tres clases (ver figura 3).

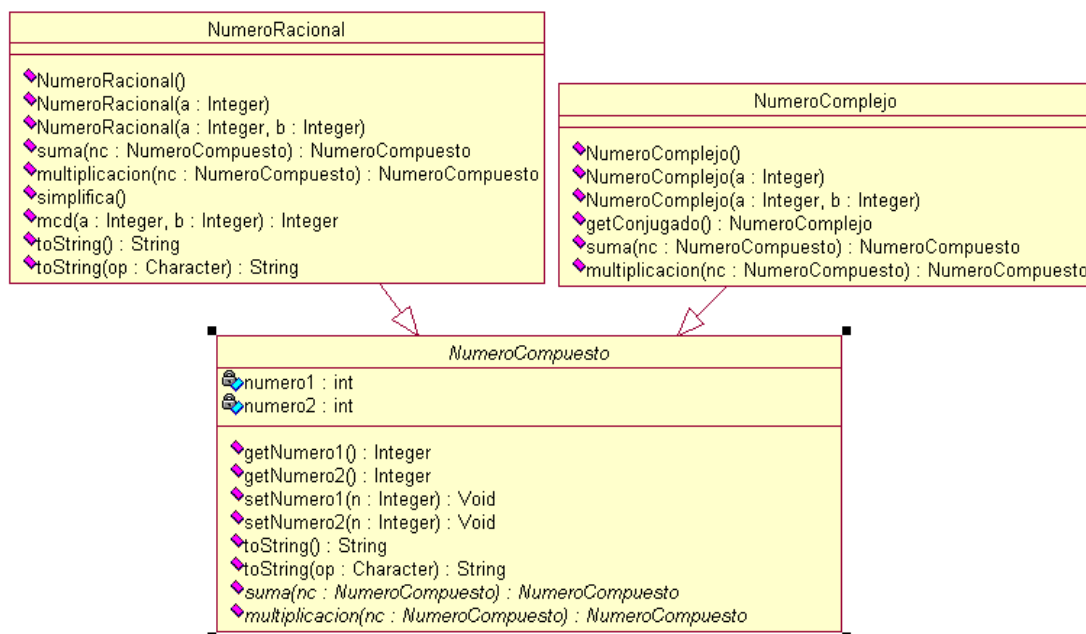


Figura 3. Diagrama de clases – Ejercicio 6 – Solución 2

- b) **[3,5 puntos]** Implementar el código de las clases que aparecen dentro del diagrama de clases. Todo el código desarrollado en este apartado permitirá el correcto funcionamiento del programa principal de este ejercicio.

A continuación se muestran las dos posibles soluciones de implementación correspondientes a los dos posibles diseños del apartado anterior.

Solución1:

```

public interface Operaciones {
    public NumeroCompuesto suma(NumeroCompuesto nc);
    public NumeroCompuesto multiplicacion(NumeroCompuesto nc);
}
  
```

```

public class NumeroCompuesto {
    private int num1;
    private int num2;

    public NumeroCompuesto(){ num1 = 0; num2 = 0;}
    public NumeroCompuesto(int a, int b){ num1 = a; num2 = b; }
  
```

```
public int getNum1(){ return this.num1;}
public int getNum2(){ return this.num2;}
public void setNum1(int n){ this.num1 = n;}
public void setNum2(int n){ this.num2 = n;}
public String toString(){ return (num1 + " " + num2);}
public String toString(char o)
{ return (num1 + " " + o + " " + num2); }
}
```

```
public class NumeroRacional extends NumeroCompuesto implements
Operaciones{
    public NumeroRacional () { super(0,1); }
    public NumeroRacional (int a) { super(a,1); }
    public NumeroRacional (int a, int b){ super (a,b);}

    public NumeroCompuesto suma(NumeroCompuesto nr){
        int n = this.getNum1() * nr.getNum2() +
                this.getNum2() * nr.getNum1();
        int d = this.getNum2() * nr.getNum2();
        return (new NumeroRacional(n,d));
    }
    public NumeroCompuesto multiplicacion (NumeroCompuesto nr){
        return new NumeroRacional(getNum1() * nr.getNum1(),
                getNum2() * nr.getNum2());
    }
    public void simplifica(){
        int aux = mcd(this.getNum1(), this.getNum2());
        this.setNum1(this.getNum1() / aux);
        this.setNum2(this.getNum2() / aux);
    }
    public int mcd (int a, int b){ ... }
    public String toString(char c){
        this.simplifica();
        return super.toString(c);
    }
}
```

```
public class NumeroComplejo extends NumeroCompuesto implements
Operaciones{
    public NumeroComplejo () { super(); }
    public NumeroComplejo (int a) { super(a,0); }
    public NumeroComplejo (int a, int b){ super (a,b);}

    public NumeroComplejo getConjugado(){
        return new NumeroComplejo(getNum1(), - (getNum2()));
    }
}
```

```

    }
    public NumeroCompuesto suma(NumeroCompuesto nc){
        return new NumeroComplejo(this.getNum1() + nc.getNum1(),
                                   this.getNum2() + nc.getNum2());
    }
    public NumeroCompuesto multiplicacion(NumeroCompuesto nc){
        int r = getNum1() * nc.getNum1() - getNum2() * nc.getNum2();
        int i = getNum1() * nc.getNum2() + getNum2() * nc.getNum1();
        return new NumeroComplejo (r, i);
    }
}

```

Solución2:

```

public abstract class NumeroCompuesto {
    private int num1;
    private int num2;

    public int getNum1(){ return this.num1;}
    public int getNum2(){ return this.num2;}
    public void setNum1(int n){ this.num1 = n;}
    public void setNum2(int n){ this.num2 = n;}
    public String toString(){ return (num1 + " " + num2);}
    public String toString(char o){ return (num1+ " " +o+ " " +num2); }

    public abstract NumeroCompuesto suma(NumeroCompuesto nc);
    public abstract NumeroCompuesto multiplicacion(NumeroCompuesto nc);
}

```

```

public class NumeroComplejo extends NumeroCompuesto{
    public NumeroComplejo () { super.setNum1(0); super.setNum2(0);}
    public NumeroComplejo (int a) { super.setNum1(a); super.setNum2(0);}
    public NumeroComplejo (int a, int b){
        super.setNum1(a);
        super.setNum2(b);
    }
    public NumeroComplejo getConjugado(){
        return new NumeroComplejo(this.getNum1(), - (this.getNum2()));
    }
    public NumeroCompuesto suma(NumeroCompuesto nc){
        return new NumeroComplejo(this.getNum1() + nc.getNum1(),
                                   this.getNum2() + nc.getNum2());
    }
}

```

```
public NumeroCompuesto multiplicacion(NumeroCompuesto nc){
    int r = getNum1() * nc.getNum1() - getNum2() * nc.getNum2();
    int i = getNum1() * nc.getNum2() + getNum2() * nc.getNum1();
    return new NumeroComplejo (r, i);
}
}



---



public class NumeroRacional extends NumeroCompuesto{
    public NumeroRacional () { super.setNum1(0); super.setNum2(1);}
    public NumeroRacional (int a) { super.setNum1(a); super.setNum2(1);}
    public NumeroRacional (int a, int b){
        super.setNum1(a);
        super.setNum2(b);
    }
    public NumeroCompuesto suma(NumeroCompuesto nr){
        int num = getNum1() * nr.getNum2() + getNum2() * nr.getNum1();
        int den = getNum2() * nr.getNum2();
        return (new NumeroRacional(num,den));
    }
    public NumeroCompuesto multiplicacion (NumeroCompuesto nr){
        return new NumeroRacional(this.getNum1() * nr.getNum1(),
            this.getNum2() * nr.getNum2());
    }
    public void simplifica(){
        int aux = mcd(this.getNum1(), this.getNum2());
        this.setNum1(this.getNum1() / aux);
        this.setNum2(this.getNum2() / aux);
    }
    public int mcd (int a, int b){ ...}
    public String toString(char c){
        this.simplifica();
        return super.toString(c);
    }
}
}
```