Programación I

Práctica N°1 – Objetos y Arreglos

Importante: En esta práctica se pide implementar métodos y clases. Estas clases (entre otras cosas) ya están creadas dentro de un proyecto de Eclipse que se encuentra disponible en la página de la materia. Se pide descargar dicho proyecto y trabajar dentro del mismo, ya que en dicho proyecto se dispone de herramientas útiles para algunos de los ejercicios de esta guía.

Ejercicio 1

Teniendo en cuenta la siguiente clase en Java

```
public class Fraccion
{
    int numerador;
    int denominador;
}
```

a) Escribir el constructor `Fraccion(int numerador, int denominador)`
b) Escribir el método de instancia `void imprimir()` que imprime en pantalla la fracción en algún formato cómodo. Crear en un main una `Fraccion` e imprimirla con este método.
c) Escribir el método de instancia `void invertirSigno()` que invierte el signo del número: si era negativo pasa a ser positivo y viceversa.
d) Escribir el método de instancia `void invertir()` que invierte el numerador y el denominador de la fracción. Ej. `invertir`$(1/2) = 2/1$.
e) Escribir el método de instancia `double aDouble()` que convierte el número racional en un `double` (número de punto flotante) y devuelve el resultado.
f) Escribir el método de instancia `void reducir()` que reduce el número racional a sus términos más chicos. Para esto buscar el MCD del numerador y el denominador y luego dividir numerador y denominador por su MCD. Usar la función ya programada de la práctica anterior.
g) Escribir el método de clase `static Fraccion producto(Fraccion q1, Fraccion q2)` que calcula el producto entre dos fracciones en nuevo objeto `Fraccion`. Éste debe estar reducido a su mínima expresión, de modo que el numerador y el denominador no tengan un divisor común mayor a 1. ¿Por qué en este caso hace falta la palabra clave `static`?
h) Escribir el método de clase `static Fraccion suma(Fraccion q1, Fraccion q2)` que calcula la suma de dos fracciones en un nuevo objeto `Fraccion`. Éste debe debe estar reducido a su mínima expresión.

Ejercicio 2

Teniendo en cuenta la siguiente clase en Java

```
public class Punto
{
    double x;
    double y;
}
```

a) Escribir el constructor `Punto()` que inicializa las dos coordenadas a 0.
b) Escribir el constructor `Punto(`**double** `x`, **double** `y)`
c) Escribir el método **void** `imprimir()` que muestra por consola los valores del punto.
d) Escribir el método de instancia **void** `desplazar(`**double** `desp_x`, **double** `desp_y)` que desplace el punto tanto como lo indiquen los parámetros recibidos.
e) Escribir el método de clase **static double** `distancia(Punto p1, Punto p2)` que devuelve la distancia entre los dos puntos.

Ejercicio 3

La siguiente clase en Java describe a un círculo en un plano (representado mediante su radio y las coordenadas de su centro):

```
public class Circulo
{
    double radio;
    Punto centro;
}
```

Además de la clase `Circulo`, pueden encontrar dentro del proyecto que se encuentra en la página de la materia la clase `Dibujador`. Los objetos de esta última permiten dibujar círculos en la pantalla mediante el método `dibujar(Circulo c)`. Para ello es necesario construir un `Dibujador` e invocar sobre éste el método mencionado. Por ejemplo:

```
Dibujador dib = new Dibujador();
Circulo circ1 = new Circulo(100, 200, 80); // Ver constructor más abajo
Circulo circ2 = new Circulo(500, 400, 120); // Ver constructor más abajo
dib.dibujar(circ1);
dib.dibujar(circ2);
```

a) Escribir el constructor `Circulo(`**double** `centro_x`, **double** `centro_y`, **double** `radio)`.
b) Escribir el método **void** `imprimir()` que muestra por consola los valores del círculo.
c) Escribir los métodos de instancia **double** `perimetro()` y **double** `superficie()` que devuelven el perímetro y la superficie del círculo, respectivamente.
d) Escribir el método de instancia **void** `escalar(`**double** `factor)` que modifique el radio del círculo en un factor de escala pasado como parámetro.
e) Escribir el método de instancia **void** `desplazar(`**double** `desp_x`, **double** `desp_y)` que desplace el centro del círculo tanto como lo indiquen los parámetros recibidos.
f) Escribir el método de clase **static double** `distancia(Circulo c1, Circulo c2)` que calcula y devuelve la distancia entre los puntos más cercanos de los círculos. Por ejemplo, un círculo con centro $(0, 0)$ y radio 1 tiene distancia 1 de otro de centro $(3, 0)$ y radio 1. **Observación**: notar que la distancia no puede ser un valor negativo, es decir, si los círculos se solapan, entonces la distancia entre ellos es cero.
g) Escribir el método de clase **static boolean** `seTocan(Circulo c1, Circulo c2)` que devuelva verdadero si las áreas de los círculos pasados como parámetro se solapan y falso si no.
h) Escribir el método de instancia **boolean** `loContiene(Circulo otro)` que devuelva verdadero si toda la superficie del círculo pasado como parámetro está contenida en la superficie del argumento implícito y falso en caso contrario.[1]

[1] Pista: pensar en la relación entre el radio y la distancia al centro del otro círculo más su radio.

Ejercicio 4

Los fractales son dibujos que pueden crearse de forma recursiva. Mediante una función recursiva que dibuja figuras básicas y se llama recursivamente para dibujar figuras más pequeñas es posible lograr figuras muy complejas, muy difíciles de lograr sin recursividad.

a) Escribir un método **static void** `dibujarCirculos(Dibujador dib, int x, int y, int r)` que dibuje un círculo de radio `r` en el punto `(x,y)` del dibujador pasado como parámetro y luego se llame recursivamente para dibujar dos círculos de la mitad del radio a izquierda y a derecha del punto `(x,y)`, a una distancia de `r`. Cuando `r` sea más chico que `10`, el método debe retornar sin dibujar nada, este sería su caso base.
b) Modificar el método anterior para que cada llamado dibuje además dos círculos en los extremos superior e inferior del círculo original.

Ejercicio 5

Teniendo en cuenta la siguiente clase de Java

```
public class Persona
{
    String nombre;
    int edad;
}
```

a) Escribir el constructor `Persona(String nombre, int edad)`.
b) Escribir el método de instancia **boolean** `masJovenQue(Persona otro)` que indica si la instancia es una persona más joven que la persona pasada como parámetro.
c) Escribir el método de instancia **boolean** `tocayo(Persona otro)` que indica si la instancia y la persona pasada como parámetro tienen el mismo nombre.
d) Escribir el método de instancia **boolean** `mismaPersona(Persona otro)` que compare el nombre y la edad de ambas personas para determinar si son la misma persona.
e) Pensar qué ocurre si se agrega la variable de instancia **int** `DNI` con el método del punto anterior para tener en cuenta este nuevo dato. ¿Sería correcto que el método sólo compare los DNI de las personas sin importar el nombre y la edad? ¿En qué contexto podría ser válido tal método?

Ejercicio 6

Escribir en la clase Arreglos, los siguientes métodos de clase:

a) **static boolean** `esSinRepetidos(int[] arr)` que toma un arreglo de 0 o más elementos y devuelve `true` si el arreglo no contiene elementos repetidos.
b) **static int**[] `pegar(int[] arr, int[] arr2)` que toma dos arreglos de 0 o más elementos y devuelve un nuevo arreglo que es el resultado de concatenar los dos arreglos. Por ejemplo, (`pegar([1,2,3],[4,5,6])`) devuelve el arreglo [1,2,3,4,5,6].
c) **static int**[] `agregarAlFinal(int[] arr, int elem)` que toma un arreglo de 0 o más elementos y un elemento y devuelve un nuevo arreglo que es igual al pasado por parámetro, salvo que al final, se le agregó `elem`.
d) **static int**[] `sinRepetidos(int[] arr)` que devuelve un nuevo arreglo con los mismos elementos que tenía el parámetro, pero sin repeticiones de elementos.

e) **static void** `invertir(int[] arr)` que modifica los elementos del arreglo pasado por parámetro de modo que queden al revés de como estaban originalmente.

Ejercicio 7

Expandir la clase Persona del ejercicio 5, con los siguientes métodos:

a) Escribir el método de clase **static** `Persona masJoven(Persona[] grupo)` que devuelve la persona más joven de un arreglo de Personas.
b) Escribir el método de clase **static** `Persona buscar(Persona[] grupo, String nombre)` que devuelve la persona cuyo nombre coincide con el parámetro. Si hay más de una, se puede devolver cualquiera de ellas.

Ejercicio 8

Las imagenes se codifican en la computadora como una matriz de píxeles, donde cada píxel es un punto de un determinado color y está representado por 3 números: un entero para el color rojo, un entero para el color verde, y un entero para el color azul. Esos 3 números van del 0 al 255 y es análogo a lo que hace el pintor en su paleta cuando mezcla colores: decide cuánto de cada color va a poner para formar un nuevo color. Este sistema se conoce como RGB, de las siglas en inglés para los 3 colores (Red, Green, Blue). Es así como, por ejemplo, el color negro se codifica con el valor $(0, 0, 0)$, el blanco con $(255, 255, 255)$, el rojo con $(255, 0, 0)$, el amarillo con $(255, 255, 0)$, etc. Se dispone de las siguientes clases en Java

```
public class Pixel
{
    int rojo;
    int verde;
    int azul;
}

public class Imagen
{
    Pixel[][] pixels;
    int alto;
    int ancho;
}
```

La clase `Imagen` provee también un constructor que toma un String donde se especifica un nombre de archivo. Este puede ser simplemente el nombre de un archivo ubicado en el mismo directorio del proyecto (ej. `''lena.bmp''`), o bien, uno guardado en otra ubicación del disco rígido. Sin embargo, para ello, debe escribirse en forma de una ruta absoluta. Para escribir rutas absolutas se deben utilizar los caracteres '/' para separar directorios, en lugar de los clásicos '\' de Windows (ej. `''C:/Documents And Settings/Usuario/Documentos/imagen.jpg''`). Por último se provee el método **void** `Dibujador.dibujar(Imagen img)` que dada una `Imagen` la muestra por pantalla.

a) Probar en un main cargar una imagen y mostrarla por pantalla. Sólo hace faltan dos pasos para ello: construir una `Imagen` a partir de un nombre de archivo y llamar a `Dibujador.dibujar(...)` para mostrarla.

b) Escribir en la clase Imagen, el método de instancia **void** `enrojecer`(**int** `cant`) que incrementa el valor de rojo de cada pixel en `cant`. ATENCION: ¡el valor de rojo no se puede pasar de 255!
c) Escribir en la clase Imagen, el método de instancia **void** `aumentarBrillo`(**int** `cant`) que incrementa el valor de los 3 colores de cada pixel en `cant`. ¡Atención a no pasarse de 255!
d) Los grises en RGB son todos aquellos valores que tienen las tres componentes (rojo, verde y azul) iguales. Para pasar una imagen color a escala de grises, se deben establecer los tres colores al valor $0{,}3 \times r + 0{,}6 \times v + 0{,}1 \times a$, donde r es la cantidad de rojo, v la cantidad de verde y a la cantidad de azul. Escribir el método de instancia **void** `aGrises`() que convierte la imagen a escala de grises.
e) Escribir el método de instancia **void** `invertir`() que, a cada color, le pone el valor $255 - c$, donde c es el valor que tenía antes.
f) Escribir el método de instancia **void** `espejar`() que pone los píxeles que estaban a la derecha a la izquierda, y viceversa, los que estaban a la izquierda ahora pasan a la derecha.
g) Escribir el método de instancia **void** `girarDerecha`() que rota la imagen 90° a la derecha. Notar que el alto es ahora el ancho y el ancho es ahora el alto. Pista: crear una nueva matriz.

Ejercicio 9

Teniendo en cuenta la siguiente clase de Java

```
public class Agenda
{
    Persona[] contactos;
    String[] telefonos;
}
```

a) Escribir el constuctor `Agenda`(**int** `tamanio`) que inicializa `contactos` y `telefonos` con arreglos del tamaño dado.
b) Escribir el método de instancia **void** `guardar`(`Persona contacto`, `String telefono`) que guarda los datos del contacto en la primera posición libre[2] que encuentre en los arreglos. Se supone que si una posición está libre en un arreglo, también lo estará en el otro. Si no hay una posición libre, se deberán redimensionar los arreglos.
c) Escribir el método de instancia **void** `eliminar`(`Persona contacto`) que elimina de la agenda los datos del contacto. Debe eliminar tanto a la persona como a su número de teléfono. Usar el método `mismaPersona` del ejercicio 5. Si el contacto no aparece en la agenda, el método no hace nada.
d) Escribir el método de instancia **boolean** `pertenece`(`Persona contacto`) que devuelve `true` si el contacto está en la Agenda o `false` en caso contrario.
e) Escribir el método de instancia `String dameTelefono`(`Persona contacto`). Usar el método `mismaPersona` del ejercicio 5. (REQUIERE: `pertenece(contacto)`==**true**)

Ejercicio 10

Consideremos las clases `UNGS`, `Comision`, `Docente` y `Estudiante` definidas como:

[2] Comparar con `null` para saber si una posición está libre

```
public class UNGS {
        Comision[] comisiones;
        ...
}

public class Comision {
        String materia;
        int numero;
        Docente[] docentes;
        Estudiante[] inscriptos;
        int[] calificaciones;
        ...
}

public class Docente {
        String nombre;
        int dni;
        ...
}

public class Estudiante {
        String nombre;
        int legajo;
        ...
}
```

Los arreglos `inscriptos` y `calificaciones` de una `Comision` tienen el mismo tamaño y el valor de `calificaciones[i]` indica la calificación obteinda por el estudiante `inscriptos[i]`. Para la clase `UNGS`:

a) Escribir un método **boolean** `cursaCon(Estudiante e, Docente d)` que indica si el estudiante `e` cursa en alguna comisión con el docente `d`.
b) Escribir un método **boolean** `suficientesDocentes()` que indica si todas las comisiones tienen al menos un docente por cada 20 inscriptos.
c) Escribir un método `Estudiante elMasEstudioso()` que devuelve el estudiante que haya aprobado la mayor cantidad de cursos (consideramos que para aprobar un curso se requiere una calificación de al menos 4). En caso de haber más de un estudiante con la mayor cantidad de cursos aprobados se puede devolver cualquiera de ellos.
d) Se quiere dar un premio a cada estudiante que haya obtenido la mejor calificación en cada comisión. Escribir un método **int** `losMejores()` que devuelve la cantidad de estudiantes que obtuvieron la mejor nota de cada comision.
e) Escribir un método **int** `alumnosDe(Docente d)` que devuelve la cantidad de estudiantes del docente dado, es decir, todos los estudiantes que cursen una comisión que dicte el docente.
f) Escribir un método **int** `unicaComision()` que devuelve la cantidad de materias que tienen una única comisión.

Ejercicio 11

Consideremos las clases `Tripulante`, `Avion`, `Vuelo` y `Aerolinea` definidas como:

```
public class Aerolinea {
        Vuelo[] vuelos;
        ...
}

public class Tripulante {
        String nombre;
        String cargo; //"Piloto","Aeromozo", etc.
        int antiguedad;
        ...
}

public class Vuelo {
        Avion avion;
        Tripulante[] tripulacion;
        ...
}

public class Avion {
        String tipo;
        int capacidad;
        ...
}
```

Se utilizan estas clases para representar los vuelos de una aerolínea. Se pide para la clase `Aerolinea`:

a) Escribir un método **int** `vuelosEn(Tripulante t, String tipoAvion)` que recibe un tripulante `t` y un tipo de avion e indica cuántos vuelos realizó `t` en aviones del tipo dado.
b) Escribir un método **int** `antiguedadPromedio(String tipoAvion)` que recibe un tipo de avión y calcula el promedio de las antigüedades de los tripulantes en vuelos en aviones del tipo dado.
c) Escribir un método `Vuelo elMasInspeccionado()` que devuelve el vuelo en el que se hayan transportado más tripulantes con cargo de "Inspector". En caso de haber más de un vuelo con la misma cantidad de inspectores, puede devolver cualquiera de ellos.
d) Escribir un método **boolean** `hayVueloSobrecargado()` que devuelve **true** cuando hay al menos un vuelo en el que la cantidad de tripulantes con cargo de "Aeromozo" supere el 10 % de la capacidad del avión.
e) Escribir un método `Tripulante elMasViajero()` que devuelve el tripulante que haya estado en la mayor cantidad de vuelos. En caso de haber más de un tripulante con estas características, puede devolver cualquiera de ellos.

Ejercicio 12

Consideremos las clases `Modulo7`, `Laboratorio`, `PC`, `Marca` y `Componente` definidas como:

```
public class Modulo7 {
    Laboratorio[] labos;
}

public class Laboratorio {
    int numero;
    int capacidad;
    PC[] computadoras;
}

public class PC {
    String serial;
    String modelo;
    String OS;
    Componente[] componentes;
}

public class Componente {
    String nombre;
    String tipo;
    Marca marca;
}

public class Marca {
    String nombre;
    float calidad;
}
```

Estas clases modelan los laboratorios que posee el `Modulo7`. Cada laboratorio posee computadoras que están compuestas por componentes. Cada `Componente` es una determinada marca que tiene un estandar de calidad, por lo cual según la marca y especificaciones se le asigna un valor entre 1 y 5 a la calidad del componente. Para la clase `Modulo7` se pide:

a) Escribir un método `LinkedList<Componente> componentesPorMarca(Marca m)` que devuelva una lista con todos los componentes del laboratorio que coincidan con la marca recibida por parámetro. La lista no debe contener elementos repetidos.
b) Escribir un método **int** `pcsConMinimaCalidad()` que devuelva la cantidad de pcs que tienen todos sus componentes con calidad 1 en el módulo 7.
c) Escribir un método **boolean** `gamaAlta()` que determina si existe una pc en el laboratorio que sea de alta gama. Se considera que una pc es de alta gama si todos sus componentes tienen calidad mayor o igual a 4.
d) Escribir un método `LinkedList<PC> pcCompuestasPor(Marca[] ciertasMarcas)` que devuelva una lista con todas las PCs del módulo 7 donde todos sus componentes tengan su marca perteneciente al arreglo `ciertasMarcas`.

e) Escribir un método `Marca marcaMasUsada()` que devuelva la marca que más componentes provee en el laboratorio. En caso de tener más de una Marca con estas condiciones, se debe devolver cualquiera de ellas.