

**UNIVERSIDAD ANTONIO NARIÑO**

# Ingeniería de Software

**Taller No. 4**

|  |  |
| --- | --- |
| **NOMBRE DEL PROGRAMA:** | Ingeniería de Sistemas |
| **ASIGNATURA** | Especificación y Modelado |
| **SEMESTRE:** | 2019-II |
| **CREDITOS:** | 3 |

**Juan Guillermo Conde Tique**

**Ingeniería de Sistemas**

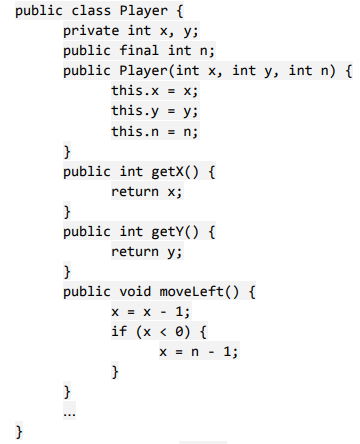
**Modalidad (distancia)**

**Ejercicio 1**

Una forma de documentar el código es haciendo uso de contratos. Para este ejercicio, usted deberá considerar:

* Precondiciones: las condiciones que el cliente del método debe cumplir para realizar el llamado (p.e. al llamar el método sqrt(x), que calcula la raíz cuadrada del argumento, el cliente debe proveer un valor x>=0).
* Postcondiciones: las garantías que el cliente del código tiene después de que el método es ejecutado (p.e. El método sqrt(x) va a retornar un resultado y donde y\*y==x).
* Invariantes: las condiciones que todas las instancias de la clase deben satisfacer cuando son observadas por los clientes (p.e. Para una BankAccount el saldo siempre es mayor o igual a cero balances>=0, después de toda operación de depósito o retiro). Los contratos son parte de la especificación, de la implementación, así que no pueden modificar el estado del programa. Así que solo se pueden usar expresiones puras en las precondiciones, post-condiciones e invariantes. Considere un juego simple, en el que el jugador se puede mover a la izquierda, derecha, arriba y abajo en un tablero de n\*n y una implementación parcial de este código en Java es la siguiente: Ejercicio 1

Los contratos son parte de la especificación, de de la implementación, así que no pueden modificar el estado del programa. Así que solo se pueden usar expresiones puras en las precondiciones, post-condiciones e invariantes. Considere un juego simple, en el que el jugador se puede mover a la izquierda, derecha, arriba y abajo en un tablero de n\*n y una implementación parcial de este código en Java es la siguiente:



Escriba una invariante para la clase Player, una precondición para el constructor y una post-condición para el método moveLeft.

Solución:

Invariante:

Para la clase Player el tablero n siempre es mayor o igual que 2 (n >= 2).

Precondiciones:

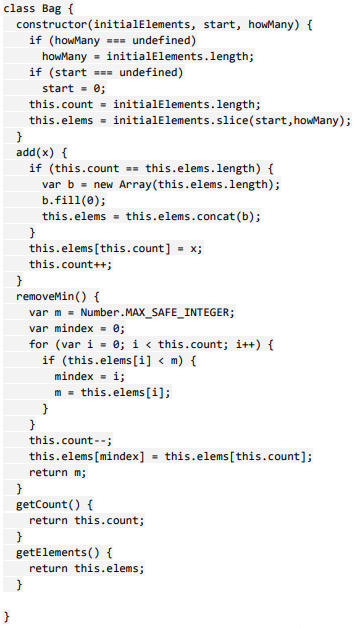
El constructor está definido como public Player, de modo que los parámetros que se pasan al constructor (int x, int y, int n) no pueden ser nulos o menores que 0. El número de N\*N debe ser mayor o igual que 2 porque es la única forma que el usuario se puede mover en las cuatro direcciones.

Postcondiciones:

El método public void moveLeft(), devuelve en la variable X, la penúltima posición en el eje X.

**Ejercicio 2**

Considere el siguiente código Javascript:



class bag{

constructor(initialElements, start, howMany){

invariant:

typeof initialElements === 'Array', "Primer Argumento debe ser un Array";

this.initialElemets.length >= 0, 'No puede ser null';

typeof start === 'Integer', "Segundo Argumento debe ser un entero";

typeof howMany === 'Integer', "Tercer Argumento debe ser un entero";

main:

if (howMany === undefined)

howMay = initialElements.length;

if (start === undefined)

start = 0;

this.count = initialElements.length;

this.elems = initialElements.slice(start, howMany);

}

add(x){

pre:

typeof x === 'Integer';

main:

if (this.count == this.elems.length){

var b = new Array(this.elems.length);

b.fill(0);

this.elems = this.elems.concat(b);

}

this.elems[this.count] = x;

this.count++;

}

removeMin(){

main:

var m = Number.MAX\_SAFE\_INTEGER;

var mindex = 0;

for (var i = 0; i < this.count; i++){

if(this.elems[i] < m){

mindex = i;

m = this.elems[i];

}

}

this.count--;

this.elems[mindex] = this.elems[this.count];

return m;

post:

\_\_result < Number.MAX\_SAFE\_INTEGER;

}

getCount(){

main:

return this.count;

post:

\_\_result > 0;

}

getElements{

main:

return this.elems;

post:

Array.isArray(\_\_result);

\_\_result.length > 0;

}

}

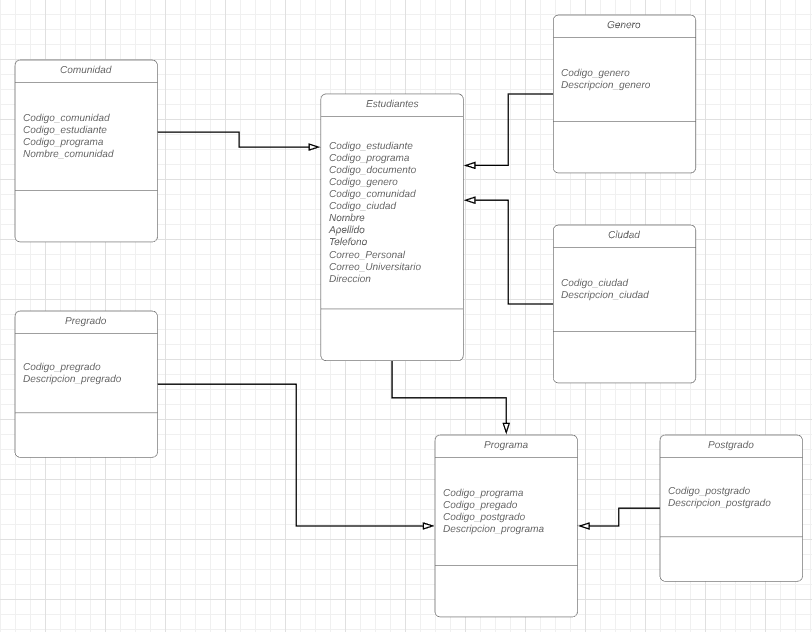
invariantes clase

precondiciones all methods

postcondiciones method add

**Ejercicio 3**

1. Dibuje un diagrama de clases UML para el sistema descrito a continuación:
2. Todo estudiante es de pregrado o de postgrado (un estudiante no puede estar en las dos categorías al mismo tiempo).
3. Un estudiante se debe registrar en la universidad y solo estudiantes registrados son estudiantes legalmente.
4. Todo estudiante tiene un identificador único, y él o ella puede tener sólo un programa inscrito.
5. Los estudiantes del mismo programa son tratados como compañeros; los estudiantes pueden tener muchos compañeros.



1. ¿Cuáles propiedades de este sistema no pueden ser representadas usando un diagrama de clases UML?

La propiedad de mayor trabajo, es los estudiantes pueden tener muchos compañeros. Para lograr representar los amigos de cada estudiante, sería necesario, filtrar por el tipo de programa, luego seleccionar la carrera que hace parte del programa y, así a través de la clase “comunidad”, se cree la comunidad. Claro está, sería a través de las fotos, los estudiantes que tiene una carrera en común, o un grupo de investigación. Que puedan crear un foro, que únicamente se puedan comunicar entre sí. Hay otras maneras, de realizar el proceso de estudiantes y amigos. En resumen, es necesario crear una comunidad o red social universitaria.

Ejercicio 4

UML permite que algunas propiedades de la generalización (como las relaciones) sean redefinidas, en lugar de ser heredadas. El propósito de la redefinición es agregar restricciones más específicas, que son particulares a las instancias especializadas, pero que no contradicen las restricciones existentes (las nuevas restricciones deben ser más fuertes que las existentes).

1. Use la redefinición (marcado en UML como {redefines property\_name}) para crear un diagrama de clases UML para el siguiente sistema:
2. Una compañía tiene uno o más empleados y cada empleado trabaja exactamente para una compañía.
3. Cada empleado puede trabajar en uno o más proyectos al mismo tiempo.
4. Empleados juniors no pueden estar asignados a más de un proyecto.
5. Los practicantes (que son empleados por tres o seis meses) pueden trabajar solo en un proyecto.
6. Discuta diferentes formas de implementar este diagrama UML en Java.

