# RETO 1 – PROGRAMACIÓN BÁSICA

## VARIANTE 1

La empresa desarrolladora de videojuegos Bugland ha comenzado el desarrollo de uno de sus nuevos títulos.

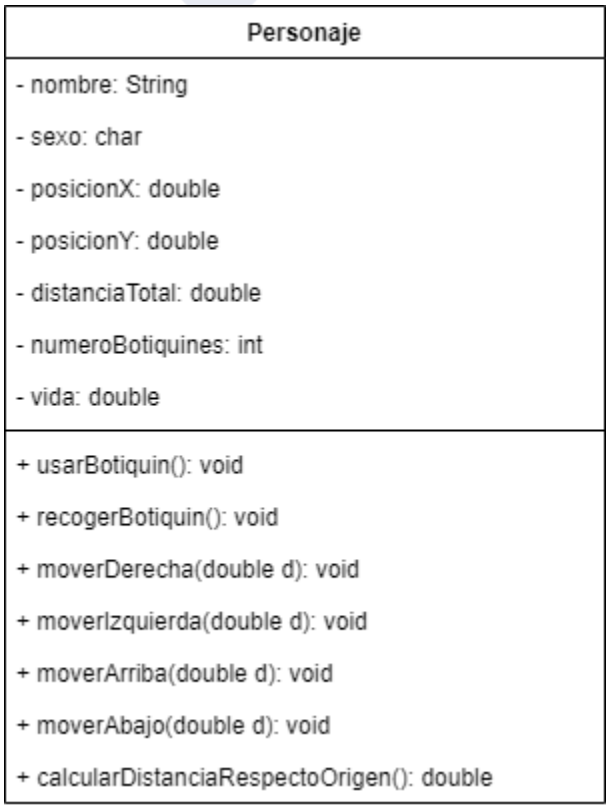
El juego será desarrollado en Java en un ambiente bidimensional (2-D) y enfocado para usuarios de PC inicialmente.

Usted ha sido contratado como Java Expert Developer, porque ha logrado demostrar habilidades de desarrollo en este lenguaje de programación y se le ha concedido implementar la clase correspondiente al personaje principal.

La empresa desea que la etapa inicial del personaje principal haga las siguientes acciones:

1. Moverse libremente a través del mapa, es decir, que el personaje pueda subir, bajar, ir a la derecha, ir a la izquierda, la unidad de medida será en pixeles (El personaje no podrá moverse en trayectorias diagonales, solo de manera horizontal y vertical).
2. Recoger botiquines y usarlos cuando se le esté acabando la vida, ya que el juego contempla luchas con poderosos enemigos, y para evitar que el personaje muera precipitadamente, podrá hacer uso de botiquines (Las luchas no están contempladas en la etapa inicial del juego).
3. Calcular la distancia recorrida con respecto al origen de coordenadas: Le permitirá dar a conocer al jugador qué tan lejano se encuentra de la casa, esto es debido a que, para poder guardar la partida, es necesario entrar en ella.

Para facilitar la implementación de la clase Personaje, el equipo de Ingeniería de software le hace entrega del diagrama de clases de la etapa inicial del personaje, recuerde que los métodos relacionados a los getters y setters son obviados en el diagrama de clases, pero deberán ser incluidos en el código (Estos métodos deberán ser creados con el estándar camel case: Por ejemplo, si el atributo se llama `distanciaTotal`, sus métodos correspondientes a get y set serían `getDistanciaTotal` y `setDistanciaTotal`).




Además del diagrama, el equipo de Ingeniería entrega esta documentación para comprender mejor los elementos del diagrama:

## Clase Personaje

### Atributos

| NOMBRE | TIPO DATO | CONCEPTO | INICIALIZACIÓN |
|---|---|---|---|
| `nombre` | `String` | Nombre del jugador | En el método constructor |
| `sexo` | `char` | Sexo del jugador (`'m'` para masculino y `'f'` para femenino) | En el método constructor |
| `posicionX` | `double` | Coordenada $x$ del personaje | Debe estar inicializada en 0 |
| `posicionY` | `double` | Coordenada $y$ del personaje | Debe estar inicializada en 0 |

| | | | |
|---|---|---|---|
| `distanciaTotal` | `double` | Distancia total recorrida por el personaje | Debe estar inicializada en 0 |
| `numeroBotiquines` | `int` | Número de botiquines que ha recolectado el personaje | Debe estar inicializada en 0 |
| `vida` | `double` | Cantidad de vida que tiene el personaje durante la partida | Debe estar inicializada en 100 |

**Métodos**

| NOMBRE | TIPO RETORNO | PARÁMETROS | CONCEPTO |
|---|---|---|---|
| `usarBotiquin` | `void` | No recibe | Resta 1 a `numeroBotiquines` y suma 5 a `vida` |
| `recogerBotiquin` | `void` | No recibe | Suma 1 a `numeroBotiquines` |
| `moverDerecha` | `void` | `double d`: Cantidad de pixeles a mover el personaje a la derecha. | Suma $d$ a `posicionX` y suma $d$ a `distanciaTotal` |
| `moverIzquierda` | `void` | `double d`: Cantidad de pixeles a mover el personaje a la izquierda. | Resta $d$ a `posicionX` y suma $d$ a `distanciaTotal` |
| `moverArriba` | `void` | `double d`: Cantidad de | Suma $d$ a `posicionY` y suma |

| | | | |
|---|---|---|---|
| | | pixeles a mover el personaje hacia arriba. | $d$ a `distanciaTotal` |
| `moverAbajo` | `void` | `double d`: Cantidad de pixeles a mover el personaje hacia abajo. | Resta $d$ a `posicionY` y suma $d$ a `distanciaTotal` |
| `calcularDistancia RespectoOrigen` | `double` | No recibe | Retorna la distancia entre el origen de coordenadas y el punto en el que se encuentra el personaje. |

## PRECISIONES

1. No hay métodos estáticos.
2. El método constructor debe asignar **SOLAMENTE** los atributos `nombre` y `sexo`.
3. `posicionX`, `posicionY`, `distanciaTotal` y `numeroBotiquines` deben ser inicializados en `0` y el atributo `vida` debe ser inicializado en `100`.
4. Deben existir `getters` y `setters` de todos los atributos de cada clase, estos deben ser escritos en la forma estándar, por ejemplo, los métodos `getter` y `setter` para la variable `posicionX` serían `getPosicionX` y `setPosicionX`.
5. Si el personaje tiene 0 botiquines, al intentar usar un botiquín **NO** modificará el número de botiquines ni la vida del personaje (Esto carece de sentido).

## TAREAS

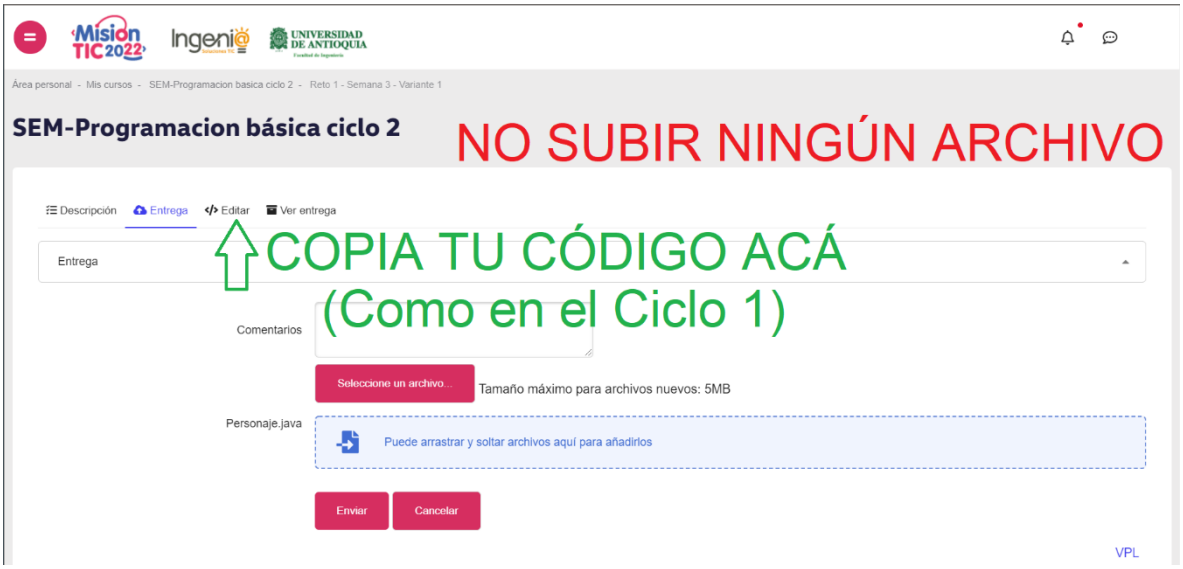
- En el archivo preconstruido en la plataforma Moodle, implementar la clase especificada en el diagrama de clases, teniendo en cuenta las precisiones dadas por el equipo de Ingeniería de software.
- Los nombres de los métodos y atributos **DEBEN** ser nombrados tal y como aparecen en el diagrama de clases.

- Usted **NO** debe solicitar datos por teclado, ni programar un método `main`, tampoco use `Java Source Package`, usted está solamente encargado de la construcción de la clase.

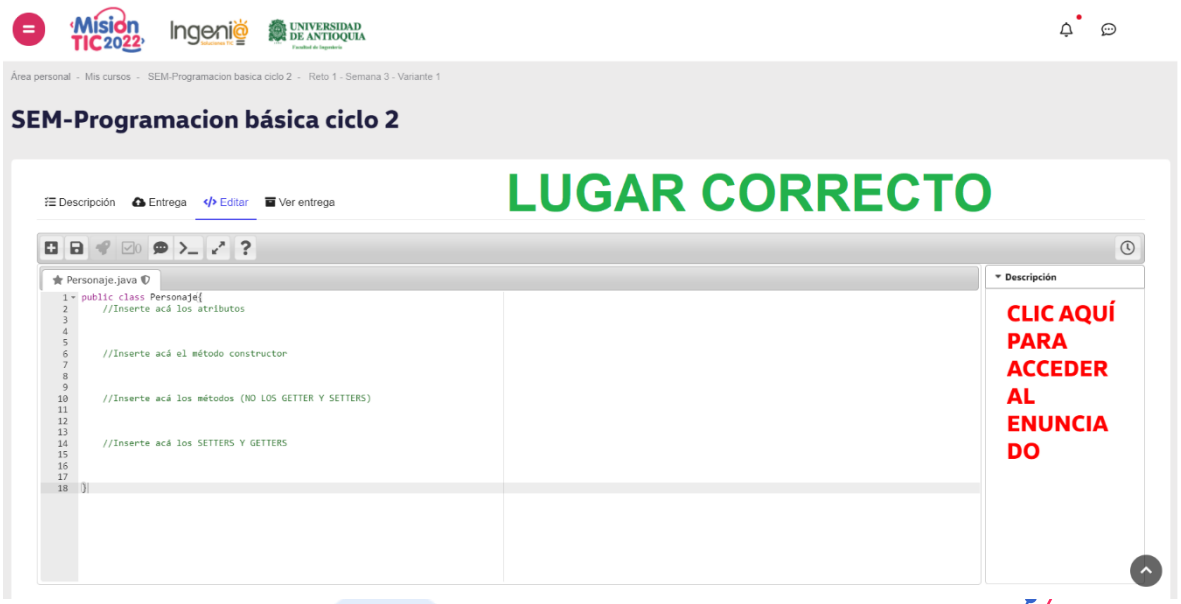
## NOTA ACLARATORIA

Usted podrá desarrollar la clase requerida en un IDE como NetBeans, y al final copiar y pegar el código en la herramienta VPL, pero **NO** deberá subir archivos, es decir:

**Modo incorrecto:**




**Modo correcto:**



## EJEMPLO

El calificador automático hará las veces de jugador, y será quien evalúe la experiencia de usuario que tuvo al jugar con su personaje:

1. El calificador mueve el personaje 2 pixeles a la derecha
   Notar el orden de los parámetros del método constructor: Primero se ingresa el nombre y luego el sexo, (Si su constructor NO cumple con esta especificación, su calificación se verá afectada).

```
Personaje explorer = new Personaje("Explorador", 'm');
explorer.moverDerecha(2);
```

| NOMBRE | CONTENIDO |
|---|---|
| nombre | "Explorador" |
| sexo | 'm' |
| posicionX | 2 |
| posicionY | 0 |
| distanciaTotal | 2 |
| numeroBotiquines | 0 |
| vida | 100 |

2. El calificador mueve el personaje 5 pixeles hacia abajo

```
explorer.moverAbajo(5);
```

| NOMBRE | CONTENIDO |
|---|---|
| nombre | "Explorador" |
| sexo | 'm' |
| posicionX | 2 |
| posicionY | -5 |

| | |
|---|---|
| distanciaTotal | 7 |
| numeroBotiquines | 0 |
| vida | 100 |

3. El calificador mueve el personaje 1 píxel a la izquierda

```
explorer.moverIzquierda(1);
```

| NOMBRE | CONTENIDO |
|---|---|
| nombre | "Explorador" |
| sexo | 'm' |
| posicionX | 1 |
| posicionY | -5 |
| distanciaTotal | 8 |
| numeroBotiquines | 0 |
| vida | 100 |

Note que `distanciaTotal = 2 + 5 + 1 = 8`.

4. El calificador pide calcular la distancia con respecto al origen de coordenadas

```
System.out.println(explorer.calcularDistanciaRespectoOrigen());
```

La salida esperada es `5.0990195135927845` (Se considerará un 1% de error en el cálculo de este valor, debido a aproximaciones que usted pueda realizar).

La distancia con respecto al origen de coordenadas se calcula de la siguiente forma:

$$\sqrt{posicionX^2 + posicionY^2}$$

Para este caso en particular: $\sqrt{(1)^2 + (-5)^2} = 5.0990195135927845$

5. El calificador le inflige daño al personaje y pide mostrar con cuanta vida quedó el personaje

```
explorer.setVida(explorer.getVida() - 40);
System.out.println(explorer.getVida());
```

La salida esperada es `60.0`

| NOMBRE | CONTENIDO |
|---|---|
| nombre | "Explorador" |
| sexo | 'm' |
| posicionX | 1 |
| posicionY | -5 |
| distanciaTotal | 8 |
| numeroBotiquines | 0 |
| vida | 60 |

6. El calificador recoge 3 botiquines

```
explorer.recogerBotiquin();
explorer.recogerBotiquin();
explorer.recogerBotiquin();
```

| NOMBRE | CONTENIDO |
|---|---|
| nombre | "Explorador" |
| sexo | 'm' |
| posicionX | 1 |
| posicionY | -5 |
| distanciaTotal | 8 |
| numeroBotiquines | 3 |
| vida | 60 |

7. El calificador usa 2 botiquines

```
explorer.usarBotiquin();
explorer.usarBotiquin();
```

| NOMBRE | CONTENIDO |
|---|---|
| nombre | "Explorador" |
| sexo | 'm' |
| posicionX | 1 |
| posicionY | -5 |
| distanciaTotal | 8 |
| numeroBotiquines | 1 |
| vida | 70 |

Note que la vida aumentó en 10, porque cada botiquín da 5 puntos más de vida.

El calificador realizará varias partidas antes de calificar su código, y mostrará en la consola si sus resultados finales coinciden con los esperados.