RETO 2 – PROGRAMACIÓN BÁSICA

CONTEXTO

La casa desarrolladora UdeASoft desea desarrollar una aplicación para un peaje ya que esto presenta una gran ventaja reduciendo tiempos de tránsito y mejorando la atención al conductor.

Usted ha sido contratado como Desarrollador Experto en Java, porque ha logrado demostrar habilidades de desarrollo en este lenguaje de programación y se le ha concedido implementar la clase correspondiente al peaje.

Esta clase contará con las siguientes funcionalidades:

- Atender el próximo coche: Termina la atención del coche que esté en la taquilla, recibe el vehículo que esté entrando a la taquilla del peaje para identificarlo y cobrar la tarifa de peaje.
- Agregar coche al programa FlyPass: Si el coche que está en la taquilla del peaje no está inscrito al programa FlyPass (un programa que le permite a un coche inscrito en él pagar el peaje por anticipado y así pasar por el peaje sin detenerse) se inscribe este como usuario del programa; si el coche ya está inscrito previamente al programa FlyPass, no se ejecuta la acción. Se supondrá que no hay usuarios que se den de baja del programa.
- Cambiar el estado del peaje: Se abre o se cierra al público el peaje: si está cerrado, entonces esta funcionalidad la abre; si está abierto, esta funcionalidad cierra el peaje.

Para facilitar la implementación de la clase `Peaje`, el equipo de Ingeniería de software le hace entrega del diagrama de clase (Figura 1).

Recuerde que los métodos relacionados al constructor, getters y setters son obviados en el diagrama de clases, pero deberán ser incluidos en el código; estos métodos deben ser creados con el estándar camel case, por ejemplo, si el atributo se llama `cocheEnAtencion`, sus métodos correspondientes a get y set serían `getCocheEnAtencion` y `setCocheEnAtencion`, para el caso de los atributos de tipo `boolean`, el `get` se cambia por un `is`, por ejemplo, si el atributo se llama `estadoPeaje` y es de tipo `boolean`, su getter será `isEstadoPeaje`.

Nota: Recuerde que desde NetBeans puede generar automáticamente los getter y setters con la opción `Insert Code...` como se muestra en la Figura 2.

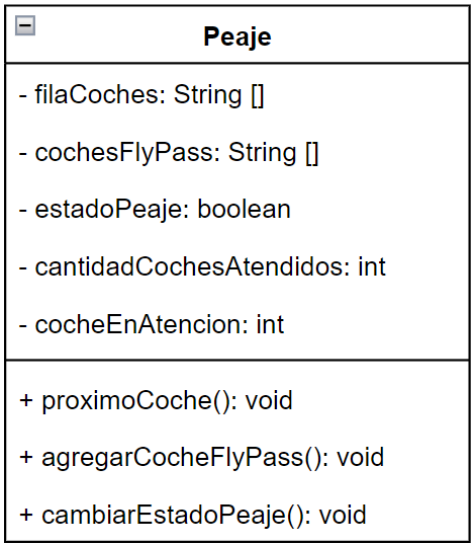


Figura 1

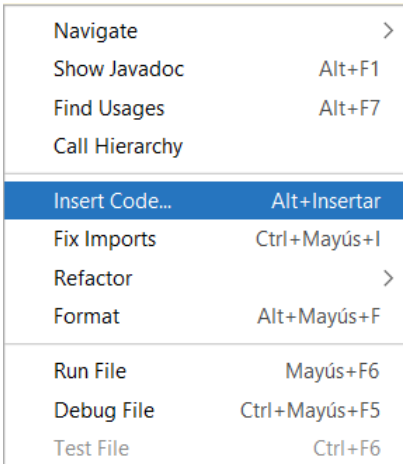


Figura 2

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

Clase Peaje

Atributos

NOMBRE	TIPO DATO	CONCEPTO	INICIALIZACIÓN
`filaCoches`	`String[]`	Contiene las placas de los coches en fila de espera para cruzar el peaje	En el método constructor
`cochesFlyPass`	`String[]`	Contiene las placas de los coches que ya están en el programa FlyPass	En el método constructor se inicializa con un espacio en blanco por cada coche que haya en `filaCoches` y con longitud igual a `filaCoches`.
`estadoPeaje`	`boolean`	Es `true` si el peaje está operando, y `false` si no lo está	Debe estar inicializada en `true`
`cocheEnAtencion`	`int`	Guarda la posición del array `filaCoches` en la que se encuentra el coche en tránsito por el peaje	Debe estar inicializada en 0
`cantidadCochesAtendidos`	`int`	Cantidad de coches que han transitado el peaje	Debe estar inicializada en 1

Métodos (Ningún método recibe parámetros, y todos retornan `void`)

NOMBRE	TIPO RETORNO	PARÁMETROS	CONCEPTO
`proximoCoche`	`void`	No recibe	Sólo si `estadoPeaje` es `true`, actualiza `cocheEnAtencion` y `cantidadCochesAtendidos`, `cocheEnAtencion` se actualizaría por el coche que siga en la lista y se incrementa en una unidad a `cantidadCochesAtendidos`
`agregarCocheFlyPass`	`void`	No recibe	Si el `cocheEnAtencion` no pertenece al array `cochesFlyPass`, entonces se agrega.
`cambiarEstadoPeaje`	`void`	No recibe	Si `estadoPeaje` es `true`, entonces lo cambia a `false`; si `estadoPeaje` es `false` entonces lo cambia a `true`

TAREAS

- En el archivo preconstruido en la plataforma Moodle, implementar la clase especificada en el diagrama de clases, teniendo en cuenta la documentación dada 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.

EJEMPLO

El calificador hará veces de usuario y será quien evalúe el funcionamiento del peaje que usted desarrolló:

1. El calificador añade catorce (14) coches a la fila del peaje:

```
String filaCoches [] = {"FNC901", "RBP250", "TPS706", "ITN503", "RSP845", "SBL560",
                        "IVD432", "LCS254", "ECT243", "RPL122", "FRS484", "TLB884",
                        "NFT948", "INS230"};
Peaje taquillaPeaje = new Peaje(filaCoches);
```

Note que usted **NO** añade las placas, lo hace quien use su clase (En este caso el calificador).

Por lo tanto, se espera que los atributos del objeto `taquillaPeaje` tengan los siguientes atributos:

```
Fila de coches:           {"FNC901", "RBP250", "TPS706", "ITN503", "RSP845", "SBL560", "IVD432",
                           "LCS254", "ECT243", "RPL122", "FRS484", "TLB884", "NFT948", "INS230",
                           "ECB437", "APQ260", "BSC602", "CRL622", "SNT254", "SGH442", "GPI524"
                          }

Coches con FlyPass:       {" ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
                           " ", " ", " ", " ", " ", " ", " ", " ", " ", " "
                          }

Estado del peaje:          true

Coche en atención:         0

Cantidad coches atendidos: 1
```

2. Si el calificador usa el método `proximoCoche` cuatro (4) veces:

```
taquillaPeaje.proximoCoche();
taquillaPeaje.proximoCoche();
taquillaPeaje.proximoCoche();
taquillaPeaje.proximoCoche();
```

Se espera que los atributos del objeto `taquillaPeaje` se actualicen de manera que queden de la siguiente manera:

```
Fila de coches:           {"FNC901", "RBP250", "TPS706", "ITN503", "RSP845", "SBL560", "IVD432",
                           "LCS254", "ECT243", "RPL122", "FRS484", "TLB884", "NFT948", "INS230",
                           "ECB437", "APQ260", "BSC602", "CRL622", "SNT254", "SGH442", "GPI524"
                          }

Coches con FlyPass:       {" ", " ", " ", " ", " ", " ", " ", " ", " ", " ",
                           " ", " ", " ", " ", " ", " ", " ", " ", " ", " "
                          }

Estado del peaje:          true

Coche en atención:         4

Cantidad coches atendidos: 5
```

Como se esperaba, el coche en atención es el quinto en la cola, en este caso el calificador atendió al primer coche y procedió a atender al siguiente, con lo que `cocheEnAtencion` se incrementa en 1; ante el siguiente coche, `cocheEnAtencion` se incrementa a 2 y así sucesivamente hasta llegar a 4.

3. Si después de lo anterior el calificador realiza las siguientes operaciones en este orden:

```
taquillaPeaje.agregarCocheFlyPass();
taquillaPeaje.proximoCoche();
taquillaPeaje.agregarCocheFlyPass();
taquillaPeaje.proximoCoche();
taquillaPeaje.agregarCocheFlyPass();
```

Se espera que los atributos del objeto `taquillaPeaje` se actualicen de manera que queden de la siguiente manera:

```
Fila de coches:           {"FNC901", "RBP250", "TPS706", "ITN503", "RSP845", "SBL560", "IVD432",
                           "LCS254", "ECT243", "RPL122", "FRS484", "TLB884", "NFT948", "INS230",
                           "ECB437", "APQ260", "BSC602", "CRL622", "SNT254", "SGH442", "GPI524"
                          }

Coches con FlyPass:       {"RSP845", "SBL560", "IVD432", " ", " ", " ", " ", " ", " ",
                           " ", " ", " ", " ", " ", " ", " ", " ", " ", " "
                          }

Estado del peaje:          true

Coche en atención:         6

Cantidad coches atendidos: 7
```

En el inciso anterior, el coche en atención era el quinto de la cola, el calificador agrega a este coche a la lista de coches con FlyPass, pasa al próximo coche (sexto de la fila), también lo agrega a la lista de coches con FlyPass y pasa al siguiente coche (séptimo de la cola) para agregarlo también a la lista de coches con FlyPass. Usted **debe** mantener el orden en el que se van agregando al array de coches con FlyPass.

4. Si ahora el calificador realiza las siguientes operaciones en ese orden:

```
taquillaPeaje.cambiarEstadoPeaje();
taquillaPeaje.cambiarEstadoPeaje();
taquillaPeaje.proximoCoche();
taquillaPeaje.agregarCocheFlyPass();
```

En las primeras dos líneas de código, el calificador cambiará dos veces el estado del peaje; este inicialmente está en `estadoPeaje=true` y la modificación se haría en este orden: `false`, `true` (2 cambios). Por lo tanto, se espera que el atributo `estadoPeaje=true`. En la tercera línea de código se atiende el siguiente coche (ahora es el octavo en la cola) y lo agrega al array de coches con FlyPass.

```
Fila de coches:           {"FNC901", "RBP250", "TPS706", "ITN503", "RSP845", "SBL560", "IVD432",
                           "LCS254", "ECT243", "RPL122", "FRS484", "TLB884", "NFT948", "INS230",
                           "ECB437", "APQ260", "BSC602", "CRL622", "SNT254", "SGH442", "GPI524"
                          }

Coches con FlyPass:       {"RSP845", "SBL560", "IVD432", "LCS254", " ", " ", " ", " ", " ",
                           " ", " ", " ", " ", " ", " ", " ", " ", " ", " "
                          }

Estado del peaje:          true

Coche en atención:         7

Cantidad coches atendidos: 8
```

Para que pueda realizar el ejemplo en Netbeans antes de pasar su propuesta de solución por el calificador, copie y pegue el siguiente código dentro de su método `main`:

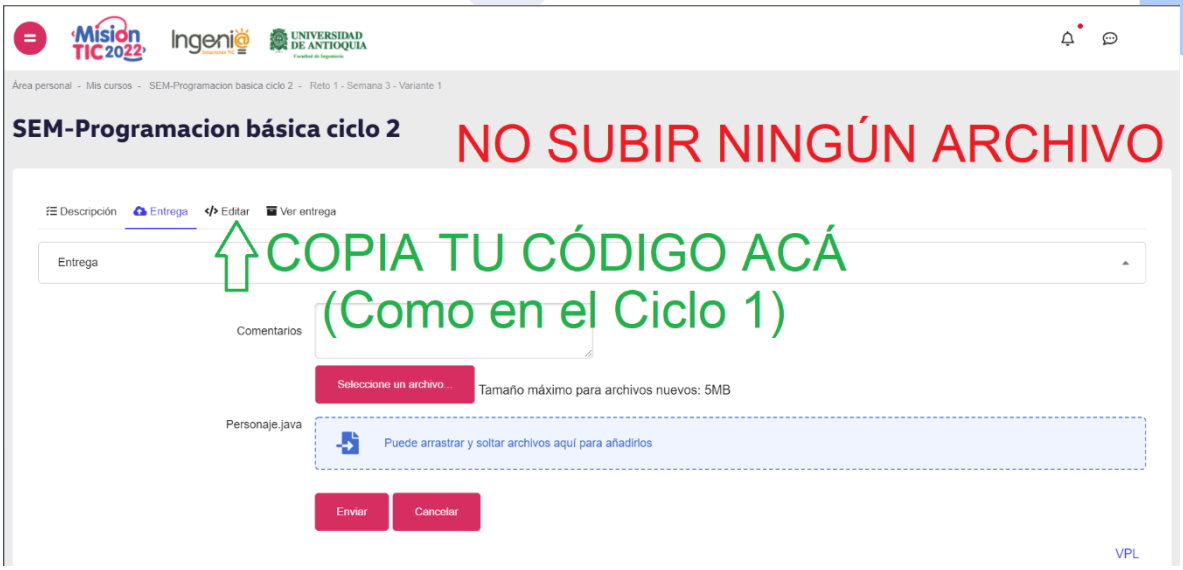
```
//Pantallazo 1

String [] filaCoches = new String [] {

"FNC901", "RBP250", "TPS706",

"ITN503", "RSP845", "SBL560",

"IVD432", "LCS254", "ECT243",

"RPL122", "FRS484", "TLB884",

"NFT948", "INS230"

};

Peaje taquillaPeaje =new Peaje(cola);

//Pantallazo 2
```

```
taquillaPeaje.proximoCoche();

taquillaPeaje.proximoCoche();

taquillaPeaje.proximoCoche();

taquillaPeaje.proximoCoche();

//Pantallazo 3

taquillaPeaje.agregarCocheFlyPass();

taquillaPeaje.proximoCoche();

taquillaPeaje.agregarCocheFlyPass();

taquillaPeaje.proximoCoche();

taquillaPeaje.agregarCocheFlyPass();

//Pantallazo 4

taquillaPeaje.cambiarEstadoPeaje();

taquillaPeaje.cambiarEstadoPeaje();

taquillaPeaje.proximoCoche();

taquillaPeaje.agregarCocheFlyPass();
```

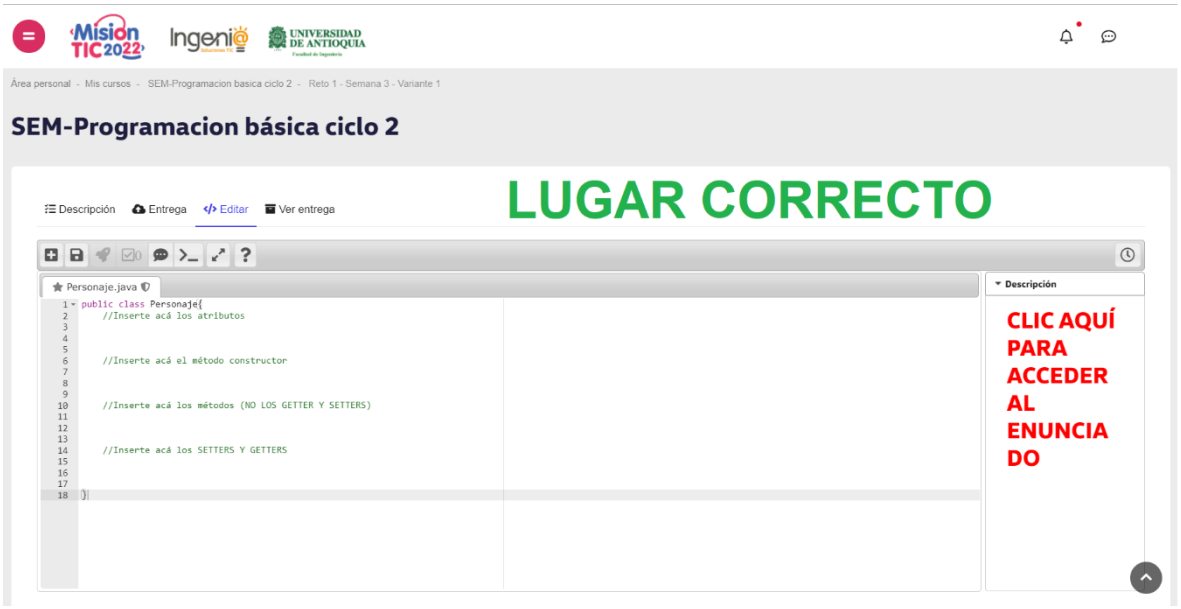
NOTA ACLARATORIA

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:



¡MUCHOS ÉXITOS EN EL DESARROLLO DEL RETO 2 TRIPULANTE!

ACLARACIÓN DE PLAGIO

El objetivo es que los tripulantes cuenten con una oportunidad de aprendizaje relacionada con la programación. La colaboración académica es buena mientras no se lleve a un engaño académico, ya que el engaño académico inflige las buenas conductas del saber y del aprendizaje. El engaño académico hace referencia al plagio o envío de ideas que no son propias.

Colaborar implica compartir ideas, explicar a alguien cómo podría hacer su trabajo (más no hacer el trabajo por el otro) y ayudar al otro si tienes problemas a la hora de ejecutar o encontrar errores en el código.

En aras de evitar el plagio se recomienda colaborar, pero no compartir su código o proyecto, no compartir sus soluciones, no usar un código encontrado en internet u otras fuentes que las propias. (Mason, Gavrilovska, y Joyner, 2019)

Los ejercicios enviados a verificación deben cumplir con la política antiplagio. Es decir, cualquier envío que sea una copia textual de otro trabajo puede ser suspendido o no aprobado por parte del equipo evaluador. El acto de copiar material de otro estudiante es un comportamiento inaceptable para el desarrollo de las competencias individuales y su progreso en este curso.

Referencia.

Mason, T., Gavrilovska, A., y Joyner, D.A. (2019). *Collaboration versus cheating. 50th ACM Technical Symposium on Computer Science Education SIGCSE 2019*, Mineapolis, MN. DOI: 10.1145/3287324.3287443