Programación

EXAMEN PRÁCTICO RECUPERACIÓN (TEMAS 1-7)

(29/05/2025)

LEE ATENTAMENTE LAS SIGUIENTES INSTRUCCIONES ANTES DE EMPEZAR:

- **Recopila en un documento de texto las evidencias de todo el examen.** Guárdalo de vez en cuando para no perder el avance de tu trabajo.
- Cuando termines, **pásalo a PDF y sube el documento creado a la entrega de AULES**.

PARTE 1: *Configuración del entorno (0,5p)*

1. Crea un nuevo repositorio llamado "*EXAMEN_RECUP_[nombre]*" desde *SourceTree*. El repositorio debe crearse en local y tener su espejo en remoto, por lo tanto, sincronízalo con *GitHub*.

   Pega a continuación la URL a tu nuevo repositorio de *GitHub*:

2. Crea un nuevo proyecto *Java* (*Maven*) con *IntelliJ* -o el IDE que utilices- dentro del repositorio que acabas de crear. Llámalo "*EXAMEN PARKING*".

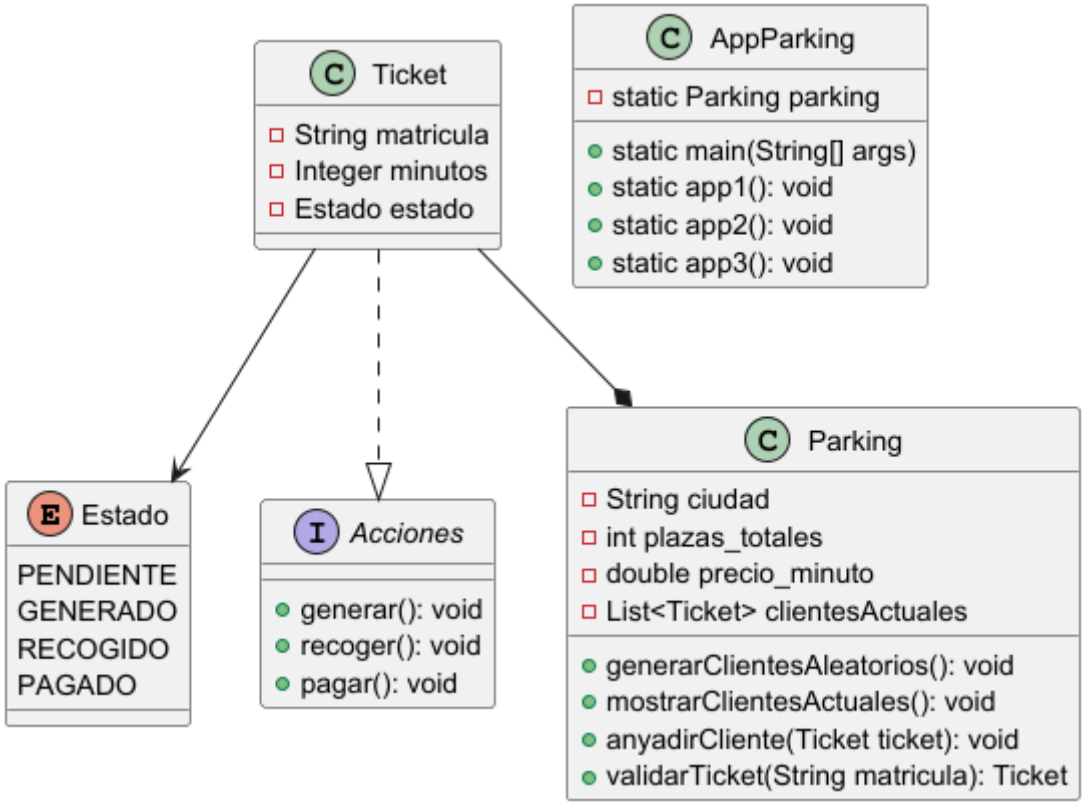
   Sincroniza los cambios en tu repositorio remoto.

PARTE 2: *Resolución de problemas*

Programa en *Java* la solución a los siguientes problemas. Usa el proyecto que te acabas de crear en el apartado anterior.

Si no has conseguido sincronizarlo correctamente en un repositorio de *GitHub*, entrégalo comprimido en un archivo .zip cuando termines.

Todos los ejercicios que se plantean a partir de ahora se construyen sobre el siguiente planteamiento de clases, correspondiente al sistema de un proyecto de un nuevo ***Parking*** que está construyendo el Ayuntamiento de Mutxamel:



Como todavía no está terminado, las clases y métodos que se plantean son temporales, y la lógica que se quiere implementar de momento es ficticia emulando situaciones que se desarrollarán en el futuro parking.

1. **(1p) Crea la clase principal *AppParking* que controla todo el sistema y el esqueleto de la estructura de clases que se muestra.**
2. **(1,5p) Para calentar, crea el objeto *Parking* en la clase *AppParking* y genérale una lista aleatoria de clientes actuales (tickets de coches aparcados) con .generarClientesAleatorios().**

   No te pases de cantidad **(máximo 10) y ten en cuenta que no se pueden repetir las matrículas de los coches (formato *1234BCD*).** Ayúdate de esto:

   String letras = "BCDFGHJKLMNPQRSTVWXYZ"

   String numeros = "0123456789"

   El estado de los tickets creados debe ser RECOGIDO en todos los casos.

(0,5p) Imprime la lista de clientes después de crearla (*.mostrarClientesActuales()*).

```
public class AppParking {

    static Parking parking = new Parking("Mutxamel Centro",1000,0.025);

    public static void main(String[] args) {

        parking.generarClientesAleatorios();
        parking.mostrarClientesActuales();

    }
}
```

```
"C:\Program Files\Java\jdk-23\bin\java.exe" ...
*** PARKING MUTXAMEL ***
1. Matrícula: 1234ABC - minutos: 15 - estado: RECOGIDO
2. Matrícula: 8524DDS - minutos: 45 - estado: RECOGIDO
3. Matrícula: 3698XZE - minutos: 123 - estado: RECOGIDO
```

3. (2,5p) App 1: Simulación de cola de entrada al Parking.

La primera app de pruebas que desarrollaremos será la encargada del acceso al parking. Será un método estático en la clase principal *AppParking*, y deberá comportarse de la siguiente manera:

a) (1p) Lo primero que debe imprimir es el estado del parking. Si está completo, mostrará ***COMPLETO***. Si quedan plazas libres, mostrará ***LIBRE ([número] plazas libres)***.

Para saber si está completo, habrá que comparar el número de plazas totales (***plazas_totales***) y la lista de clientes actuales (***clientesActuales***) del *Parking*.

Supón un número de plazas totales de 1.000. Si actualmente hay 3 coches aparcados, el programa mostrará ***LIBRE (997 plazas libres)***:

```
"C:\Program Files\Java\jdk-23\bin\java.exe" ...
*** PARKING MUTXAMEL ***
1. Matrícula: 1234ABC - minutos: 15 - estado: RECOGIDO
2. Matrícula: 8524DDS - minutos: 45 - estado: RECOGIDO
3. Matrícula: 3698XZE - minutos: 123 - estado: RECOGIDO

*** APP ACCESO ***
LIBRE (997 plazas libres).
```

b) (1,5p) A continuación, debe simular la lectura de la matrícula y generación del ***Ticket*** de un supuesto cliente que accede al parking. El encargado de hacer este proceso será el método ***.generar()*** a implementar por la clase *Ticket*.

Como todavía no tenemos sensores para leer matrículas, **por defecto, crearemos un *Ticket* con matrícula vacía, 1 minuto de estancia y estado *PENDIENTE***.

```
Ticket ticket = new Ticket("",1,Estado.PENDIENTE);
ticket.generar();
```

Durante la generación, se **creará una nueva matrícula aleatoria** y se cambiará el ***estado del ticket a GENERADO***.

```
*** APP ACCESO ***
LIBRE (997 plazas libres).
Leyendo matrícula...
Matrícula: 3245WED
Ticket generado.
```

Una vez generado, se añadirá el ticket a la lista de *clientesActuales* del parking creada en el ejercicio anterior (***.anyadirCliente(Ticket ticket)***) y llamaremos al método ***.recoger()*** para terminar el proceso de entrada.

```
Ticket ticket = new Ticket("",1,Estado.PENDIENTE);
ticket.generar();
parking.anyadirCliente(ticket);
ticket.recoger();
```

Este mostrará un mensaje de bienvenida y modificará el estado del ticket a *RECOGIDO*.

```
*** APP ACCESO ***
LIBRE (997 plazas libres).
Leyendo matrícula...
Matrícula: 3245WED
Ticket generado.
Recoja su ticket...
BIENVENIDO!
```

PRUEBA: Si volvemos a imprimir la lista de clientes (***mostrarClientesActuales()***), debería aparecer el nuevo que acaba de acceder:

```
*** APP ACCESO ***
LIBRE (997 plazas libres).
Leyendo matrícula...
Matrícula: 3245WED
Ticket generado.
Recoja su ticket...
BIENVENIDO!
1. Matrícula: 1234ABC - minutos: 15 - estado: RECOGIDO
2. Matrícula: 8524DDS - minutos: 45 - estado: RECOGIDO
3. Matrícula: 3698XZE - minutos: 123 - estado: RECOGIDO
4. Matrícula: 3245WED - minutos: 1 - estado: RECOGIDO
```

4. (2p) App 2: Pagar importe del ticket.

La siguiente app a implementar será la encargada de gestionar los pagos de los tickets cuando los clientes quieran salir del parking. Impleméntala también como método estático dentro de la clase principal ***AppParking***.

Esta app mostrará por pantalla lo siguiente, pidiendo el número de matrícula al cliente:

```
*** PAGO DE TICKETS ***
Introduce tu matrícula:
5287LKM
Validando...
```

a) (1p) El proceso de validación debe comprobar a través del método ***.validarTicket(String matricula)*** que existe un ticket en la lista de clientes actuales con la matrícula

proporcionada por el usuario. **Si existe, el método devolverá un objeto *Ticket*** con la información de ese cliente (*matrícula, minutos, estado*). **Si no existe, devolverá *null***.

```
Ticket ticket = parking.validarTicket(matricula);
```

A continuación, la app de pago que recibe el ticket comprobará que no se haya devuelto un ***null*** para mostrar por pantalla el importe a pagar por el cliente. Para calcular el importe a pagar se deben tener en cuenta los minutos que ha estado aparcado el cliente y el precio por minuto que tiene establecido el Parking.

Matrícula que existe:

```
1. Matrícula: 1234ABC - minutos: 15 - estado: RECOGIDO
2. Matrícula: 8524DDS - minutos: 45 - estado: RECOGIDO
3. Matrícula: 3698XZE - minutos: 123 - estado: RECOGIDO
*** PAGO DE TICKETS ***
Introduce tu matrícula:
3698XZE
Validando...
Minutos: 123 - Precio por minuto: 0.025€
Importe a pagar: 3.075€.
```

Matrícula que no existe:

```
1. Matrícula: 1234ABC - minutos: 15 - estado: RECOGIDO
2. Matrícula: 8524DDS - minutos: 45 - estado: RECOGIDO
3. Matrícula: 3698XZE - minutos: 123 - estado: RECOGIDO
*** PAGO DE TICKETS ***
Introduce tu matrícula:
5487LKM
Validando...
El ticket no existe.
```

b) (1p) Como todavía no podemos conectarnos con las pasarelas de pago, terminaremos de simular el pago del cliente (método ***.pagar()***) sin lógica, solamente **enseñando mensajes por pantalla y modificado el estado del ticket a *PAGADO*** directamente:

```
ticket.pagar();
System.out.println("Buen viaje!");
```

PRUEBA: Si después de completar el proceso volvemos a imprimir la lista de clientes (***mostrarClientesActuales()***), debería aparecer el estado del ticket como *PAGADO*.

```
*** PAGO DE TICKETS ***
Introduce tu matrícula:
3698XZE
Validando...
Minutos: 123 - Precio por minuto: 0.025€
Importe a pagar: 3.075€.
Pagando...
Buen viaje!
1. Matrícula: 1234ABC - minutos: 15 - estado: RECOGIDO
2. Matrícula: 8524DDS - minutos: 45 - estado: RECOGIDO
3. Matrícula: 3698XZE - minutos: 123 - estado: PAGADO
```

5. (2p) App 3: Pagar importe sin ticket (perdido).

Por si algún cliente pierde su ticket, se necesita crear una última app para que los trabajadores del Parking puedan comprobar la matrícula y realizar el cobro desde la garita. También será un método estático dentro de la clase *AppParking*.

a) (1p) Lo primero que hará esta app es **mostrar la lista de clientes por pantalla, ordenada por minutos de estancia**:

```
*** PARKING MUTXAMEL ***
1. Matrícula: 1234ABC - minutos: 250 - estado: RECOGIDO
2. Matrícula: 8524DDS - minutos: 45 - estado: RECOGIDO
3. Matrícula: 3698XZE - minutos: 123 - estado: RECOGIDO

*** APP TRABAJADORES ***
1. Matrícula: 8524DDS - minutos: 45 - estado: RECOGIDO
2. Matrícula: 3698XZE - minutos: 123 - estado: RECOGIDO
3. Matrícula: 1234ABC - minutos: 250 - estado: RECOGIDO
```

b) (1p) Una vez localizada la matrícula del cliente, deberemos proceder a cobrarle los minutos de su estancia dentro del parking.

Como los trabajadores no se sabe todavía cómo se crearán dentro de la aplicación (de momento no hay una clase *Trabajador* para ellos), **tendremos que buscar la forma de implementar el método *.pagar()* de la interfaz de *Acciones*** desde esta *app3().*

Este método ***.pagar()*** para los trabajadores deberá permitir introducir los minutos a cobrar, mostrar el importe a pagar por pantalla y terminar el proceso simulando con mensajes el pago del cliente:

```
*** APP TRABAJADORES ***
1. Matrícula: 8524DDS - minutos: 45 - estado: RECOGIDO
2. Matrícula: 3698XZE - minutos: 123 - estado: RECOGIDO
3. Matrícula: 1234ABC - minutos: 250 - estado: RECOGIDO
Introduce los minutos a cobrar:
123
Minutos: 123 - Precio por minuto: 0.025€
Importe a pagar: 3.075€.
Pagando...
Pago realizado. El cliente puede abandonar el parking.
```